NTNU
Norwegian University of
Science and Technology

Dockit League: Creating a twin stick MOBA using Unity

Author(s)

Andreas Wang
Martin Langslet Lilleslåtten
Sondre Fjeldheim Svarverud

Bachelor in Game Programming
20 ECTS
Department of Computer Science
Norwegian University of Science and Technology,

16.05.2017

Supervisor(s) Simon McCallum

Sammendrag av Bacheloroppgaven

Tittel:	**Dockit League: Et "twin stick" MOBA laget med Unity**
Dato:	16.05.2017
Deltakere:	Andreas Wang Martin Langslet Lilleslåtten Sondre Fjeldheim Svarverud
Veiledere:	Simon McCallum
Oppdragsgiver:	Norwegian University of Science and Technology
Kontaktperson:	Andreas Wang, andrwan@stud.ntnu.no, 48048162
Nøkkelord:	Spill, Programmering, Unity, Spillmotor, MOBA, Nettverk, Flerspiller
Antall sider:	378
Antall vedlegg:	4
Tilgjengelighet:	Åpen

Sammendrag: *Dockit League* er et prosjekt fokusert rundt å skape et "multiplayer online battle arena" (MOBA) spill ved hjelp av spillmotoren Unity. Spillet handler om å spille mot andre spillere i et konkurransepreget miljø der man bruker "Docking Kits" som gir spillerne forskjellige verktøy og egenskaper. Denne oppgaven vil gå gjennom utviklingen av spillet og detaljere hvordan prosjektet endret seg gjennom utviklingsprosessen.

Summary of Graduate Project

Title:	**Dockit League: Creating a twin stick MOBA using Unity**
Date:	16.05.2017
Authors:	Andreas Wang Martin Langslet Lilleslåtten Sondre Fjeldheim Svarverud
Supervisor:	Simon McCallum
Employer:	Norwegian University of Science and Technology
Contact Person:	Andreas Wang, andrwan@stud.ntnu.no, 48048162
Keywords:	Thesis, Games, Programming, Unity, Engine, MOBA, Networking, Multiplayer
Pages:	378
Attachments:	4
Availability:	Open

Abstract: *Dockit League* is a project focused around creating a multiplayer online battle arena(MOBA) game using the Unity game engine. The game revolves around playing against other players in a competitive environment, using Docking Kits which provide players with different tools and abilities. This thesis will go through the development of the game and detail how the project evolved throughout the process.

Preface

We would like to thank Simon McCallum for supervising us as well as providing feedback and directions on thesis topics throughout development. We would also like to thank all of our testers for testing the game and providing input.

Contents

List of Figures

List of Tables

List of source code snippets

1 Introduction

1.1 Project Description

1.1.1 Background

We are three students who have worked together on several projects before during our bachelor program. We wanted to make a twin stick MOBA type game for PC, intended to be played with a game controller. Our wish is to further expand our knowledge of game programming with focus on networking and game balancing. Figuring out good practices for local responsiveness coupled with consistent networked behaviour, and how to implement these is something we would like to learn. This is good knowledge to have in a world where multiplayer, and the ability to stay connected is very important, even in games. Our game is based around having many different types of "Docking Kits" with four different abilities each, letting us experience what it's like to come up with and balance large amount of components.

Having worked with Unity 5.x together on previous projects, this felt like the natural choice for our development environment. Unity is also a very popular engine used worldwide [1], potentially leaving the knowledge we gain about the engine very valuable. Since we've chosen Unity, our networking will therefore be the Unity Networking and their "high-level" scripting API (HLAPI) [2] and the advantages / challenges this provides.

As a group we consider this as a project for learning, and we have no plans of making a commercial release for our game.

1.1.2 Goals

One of our primary goals for the project is to acquire more experience with larger game projects as this is our first attempt at such a large task. We wish to further our skills and understanding of Unity with a particular focus on how to implement networked functionality. In regards to the networking we want to learn good practices for local responsiveness that still allows us to handle important verifications on the server side.

We wish attain more experience with asymmetric balancing on a larger scale by designing and implementing a large variety of docking kits, each with their own tools and abilities.

We want to learn more about professional tools like Jira and Confluence as well as how we can use these in the development of the project to provide a robust workflow. As part of learning to use these professional tools we also want to improve our skills at estimating the time needed to implement features.

In the end we would like to have created a game that features robust networked functionality and an extensible framework that is easy to improve and add new components to.

1.2 Academic Background

The three authors, Andreas, Martin, and Sondre, are currently finishing the Bachelor in Game Programming at NTNU Campus Gjøvik with this thesis. Martin is also a graduate of the Bachelor in Visual Simulation at Hedmark University College (now Inland Norway University of Applied Sciences) prior to the current bachelor.

1.3 Project Audience

The thesis is written for fellow students and game programmers who are planning to work with Unity and networking in particular. The contents should allow the reader to learn more about the technical details of working with Unity and our ways of solving the various challenges that might show up when working in the engine. We expect the reader to have some knowledge with C# and Unity, but will detail core concepts wherever we feel necessary.

1.4 Thesis Structure

The thesis is divided up into nine different chapters with appendices at the end. The chapters of the thesis include:

1. Introduction: Introduction to the thesis and its contents.
2. Game Design: The initial game design and how it changed.
3. Technical Design: How the game is architectured and how the various components work on a technical level.
4. Development Process: The tools and software development model that was used.
5. Implementation: Specifics on the challenges that had to be overcome during development and their implementation.
6. Deployment: How the game is packaged and distributed.
7. Testing and User Feedback: Details the testing process and the feedback gotten from user testing.
8. Discussion: Discusses the results of the project and the development decisions that were made.
9. Conclusion: Reflects on the finished project.

2 Game Design

The early design of a game is a solid foundation to start working with, although everything is subject to change as development progresses. This Chapter will take a look at the initial design of *Dockit League* and how the designs changed in the final versions of the game.

2.1 Initial game design

The ideas contained within the initial game design is what we would have liked to implement in a full game and were written down during the first few weeks of the project. We were unsure of how much we would get implemented so we kept the scope relatively loose by having a simple base design that could quantitatively be expanded with more game modes, docking kits and other features if the time was available.

2.1.1 Design overview

We wanted to create a competitive online game using peer to peer connectivity as a base for networking. The game would let players control small robots that fight each other by "docking" into different types of docking kits that each provide different statistics and abilities. The plan for each docking kit was to have around two to four abilities and give them different prices in an in-game shop based on their strength.

Other planned features include:

- Maps based around fog of war to limit visibility and possibly contain extra objectives from time to time.
- A variety of game modes ranging from deathmatch to more objective based modes.
- Player bots which could fill in for other players in team based play if the amount of players were uneven.
- A replay AI that records the highlights of each match and plays them back at the end of the game.

2.1.2 Game modes

The initial design of the game included three different game modes:

1. The standard game mode.
2. King of the hill.
3. 1vX.

The standard game mode would be inspired by Counter-Strike: Global Offensive [3] and alternate two teams between defending and attacking objectives. The teams would play 10 rounds with each round taking a maximum of 4 minutes before switching positions and playing 10 new rounds. Docking kits would be available for purchase at the start of each round. Docking kits would end up being dropped on death while surviving players keep their kit for the next round. Having this type of asymmetric gameplay allows using time as a win condition for one of the teams, as the time running out benefits

the defending team. A game where every round needs a winner could in theory go on forever without that kind of restriction.

The king of the hill mode would only include one objective that all teams fought over. Holding the objective would tick a team timer downwards and letting the timer tick to zero meant victory. Docking kits would not be purchasable contrary to the standard game mode, but rather spawn randomly on different map locations.

The 1vX game mode would revolve around having one player play against all the others using an empowered docking kit. This game mode would use a timer that gradually adds non-beneficial effects to any players not engaging in combat to prevent stalling on both sides. Similarly to the king of the hill, docking kits would spawn randomly around the map rather than be purchasable.

2.1.3 Docking Kit ideas

The initial design included some rough ideas for eight different docking kits:

- The Marksman Kit:
  - A kit with low health and decent speed.
  - Employs a mix of ranged abilities as well as an ability that allows the player to quickly dash in any direction to either avoid damage or close in on enemies.
- The Tank Kit:
  - A kit with high health and low speed.
  - Focuses on melee attacks with abilities that allows the player to soak damage, use crowd control and protect team mates.
- The Support Kit:
  - Undecided health and speed. This would depend on the strength of the final implemented abilities.
  - A kit that deals little damage by itself and is based around supporting teammates with healing, crowd control, shields and other types of buffs.
- The Trapper Kit:
  - A kit with low health and high speed.
  - This kit is designed to be particularly effective against melee players through the use of its multiple traps that provide a variety of negative status effects.
  - Should only be able to have one of each trap active at any time.
- The Brawler Kit:
  - A kit with decent health and decent speed.
  - This kit is primarily melee based and has abilities that allows the player to fight ranged attackers by reflecting projectiles and using crowd control.
- The Sniper Kit:
  - A kit with low health and high speed.
  - Has a larger line of sight compared to other kits and uses long ranged attacks to take advantage of this.
- The Scout Kit:

- A kit with medium or low health and very high speed.
- A melee kit that focuses on a hit and run playstyle where the player is able to quickly get into skirmishes, do damage and then escape.

- The Bomber Kit:
  - A kit with medium health and low speed.
  - Uses a variety of explosives that deal large amounts of damage.

2.2 Changes from the initial design

2.2.1 General changes

This section will take a look at the general changes from the initial design.

One of the major differences was that we ended up implementing two simpler game modes compared to the original ones we designed. We implemented a "free for all" as well as a team based deathmatch game mode due to limited time towards the end of the development period. In the case of the "free for all" deathmatch mode, the last remaining player is the winner of the round while on the team deathmatch mode, one team wins whenever all players of the opposing team have been defeated.

Player bots and replay AI functionalities were cut, but weren't necessary features for the scope of our project as we primarily wanted to focus on players playing against each other. We also ended up deciding to use four abilities per docking kit as mentioned in Section 8.1.4 because it provided us with the most ergonomic control scheme.

2.2.2 Docking kit changes

Several of the kits ended up receiving redesigns during the development of *Dockit League*. The finalized design for each docking kit can be found in Section 3.6 while this Section will take a look at how the current docking kits differ from their original designs.

The marksman and sniper kits ended up being somewhat similar in design as we worked on them, so we tried to differentiate their abilities. The marksman kit ended up more stealth oriented, while the sniper kit had far less mobility in general with a playstyle that requires the player to slowly focus and aim before firing. Looking at the current abilities of the marksman kit, the "Rogue" kit would be a less deceiving name given its current tools.

The tank kit was originally planned to be melee oriented, but we wanted to make it less similar to the brawler kit so we introduced the ranged saw blades to the kit. The brawler and tank kits still share some similarities as both are capable of reflecting projectiles.

The support, trapper, bomber and brawler kit generally stayed fairly close to their original designs, although there were some changes done. The trapper kit was given a flamethrower ability to provide some other means of doing damage rather than having four traps. The original plan for the bomber kit was to have more trap like bombs/mines, but given that the trapper kit already focused on this we tried to differentiate the two by giving the bomber kit a remote controlled mine that had to be triggered manually.

The crowd control ability of the brawler kit also ended up being somewhat different as we had originally planned for it to be a close range ability. We realized throughout development that the kit would struggle fighting the others as the majority are ranged

attackers so we gave the brawler kit a ranged stun grenade as a means of coping with this.

The scout kit was completely redesigned into the boomerang kit. It still retains the low health and high speed with a hit and run playstyle, but we changed the kit from being melee oriented to using the ranged boomerangs instead. The reason for this is that the low health of the kit made it very hard to stay alive as a melee attacker and the idea of using boomerangs felt more novel compared to the rest of the kits.

3 Technical Design

This chapter includes the technical details of *Dockit League*'s various components as well as a combination of design and technical information on all of the docking kits.

3.1 Architectures in Unity

When developing in Unity, or any existing game engine, the engine will heavily influence the basic architecture of the solution. Therefore, a basic understanding of the Unity engine can give a better understanding of the architecture in our solution. This section will include a basic description of the core concepts and terminology in Unity.

3.1.1 General overview

Unity is an component based engine, which means it moves away from the traditional OOP design with complex hierarchies. The traditional use of monolithic class hierarchies limits our design, and may force us to inherit from classes we don't really require. It makes it difficult to extend functionality, as it may affect objects further down the hierarchy. However, using a component design allows isolating the various features in a single compact service. In that way the components are decoupled, and functions independently of each other. To maintain the components a container/hub is needed. When designing an object, you can then add whichever component required to the container, giving us a lot of flexibility. [4]

In Unity every object in the game will be a *GameObject*, which is Unity's container for components. Custom scripts can be placed on these *GameObjects* if they derive from the built-in class named *MonoBehaviour*. This is a necessary step to connect the script to the internal Unity pipeline, and will let Unity handle the component management. [5] You can store a particular GameObject setup in something called a *prefab*, which makes it easy to reference this object. Unity also has a container for *GameObjects* called scenes. These are used to separate different parts of the game, different levels for instance.

3.1.2 Networking overview

Unity's networking system is referred to as their High Level API (HLAPI). This gives developers a way to access networking functionality without dealing with low level networking. The lower transport layer supports any kind of network topology, but the HLAPI uses a server authoritative system, and it's what our solution is built with.

The standard base class for scripts is *MonoBehaviour*, however, for networked scripts the base class is *NetworkBehaviour*. This derives from *MonoBehaviour*, thus has all the same functionality, but will in addition be included in the networking system. *NetworkBehaviours* may have member variables that takes advantage of the *SyncVar* attribute. This is a way for the server to synchronize the state to each remote client. However, the data types are restricted to basic data types (byte, int, float, string, etc), built-in Unity math types (Vector3, Quaternion, etc), and structs containing the previous two.

The networking system has three different ways of calling remote actions across the

network by adding custom attributes to a function. This means the function will be invoked through the HLAPI. *Command* is called by a client with authority and will run on the server. *ClientRpc* is called by the server call and will run on every client. *TargetRpc* is called by the server and will run on a specific client. The restrictions to the available parameter data type is the same as *SyncVars*. [2]

3.2 General architecture

3.2.1 Unity's example project

When it was time to wrap the parts of the game into a playable state with networked menu, lobbies and scene management a few parts in our solution were heavily based on a reference project provided by Unity themselves [6]. As these parts of the solution weren't our focus, and time was restricting, this was a way to actually get a playable prototype ready on time. The things needed were extracted from that asset, and tweaked to fit our use.

3.2.2 GameManager

The *GameManager* exists in every scene, and after we integrated Unity's example project it contains mostly code from that solution. The script also contains references to all of the Docking Kit prefabs and the player UI which includes both the in-game and pause menus.

3.2.3 SpawnableFactory

The *SpawnableFactory* handles all the spawning (instantiation) of GameObjects in the solution. It contains a list of all spawned objects, which makes it easy for the game to clean-up spawned objects when a round ends for instance. The majority of objects spawned derive from the *SpawnableObject* class, which handles the ownership related data. This includes the player owning the object, as well as the associated team ID. The only other type of object spawned is the Docking Kit Pickup. The pickup doesn't need the ownership data, which is why it's not a *SpawnableObject*, but it basically has the same functionality in the SpawnableFactory. By collecting all the spawning in one place, it makes the handling easier.

3.2.4 In-game UI

The *PlayerUIHandler* component handles the UI elements for the player. It has functionality for updating the health elements, ability elements, and status elements. The handler receives calls whenever a Docking Kit changes, extracts the ability information from the *DockingKit* component, and delegates them to the *AbilityUI* classes, which handles the individual abilities. The *StatusUI* prefab is instantiated on the status bars for every status effect added to the player. This component displays the information regarding the individual effect, like duration and effect icon. This utilizes the Unity UI *Grid Layout Group* component, to automatically format the UI elements instantiated, meaning the only thing necessary for the status effect to appear correctly is to spawn the object as a child of that layout group.

3.3 Player architecture

NetworkPlayer

The *NetworkPlayer* component is heavily based on the solution in the Unity example project. [6] And as such, won't be covered much here. However, in short, this component is not destroyed in between scenes, and handles the overall networking of a player both in the lobby and in-game.

LobbyPlayer

This is the player representation in the lobby, and is also heavily based on the example project.

3.3.1 Player

The *Player* component has parts that are based on the Unity example project, and part our own solution. This component is the player representation in the actual game. Our solution have three *TargetRpcs* for adding force to the player, why this is necessary is discussed further in Section 5.2.1. These three functions are all different ways to add force to the player. They all take a parameter determining the strength of the force added, but the direction calculation differs. The first calculates the direction from the player position towards the given position. The second calculates from the given position towards the player position. The third uses a separate force function to add explosion force, and simply takes the explosion force and radius as parameters, along with the explosion position.

3.3.2 Input

The *PlayerInput* component handles the input for each player, since this only happens locally it is a standard *MonoBehaviour*. In addition to handling the input, it also does the actual movement for the player. This class needs to restrict certain types of inputs: movement, rotation, abilities, docking, and interactions, but a simple on/off solution would not work in our scenario. The restriction could come from different sources with different lengths, therefore the source applying the initial restriction might not be the one who should remove it, since other sources might arrive in the meantime and still want to apply the restriction. The solution to this is a simple stack, in which the sources simply add a count initially, and remove one when ending. When the count is 0 we know no one is applying a restriction, and the player is free to use that input type. Figure 1 shows an example of how the movement stack changes over time when a stun and root is applied and removed at different times, thus restricting the movement input.

This component also handles interactable objects by keeping a reference to any object that implements the *IInteractable* interface through *OnTriggerEnter*. The interface function should be called on the server, to make sure only one player interacts with it at a time. This is handled through the *Player* component with a command called by the *PlayerInput* component, since the input component isn't networked.

3.3.3 Field of View

The basic implementation of the *FieldOfView* component consists of creating the view mesh by casting rays from the player position with a certain angle step between, and then use the stencil buffer to only draw pixels covered by this mesh. The base implementation

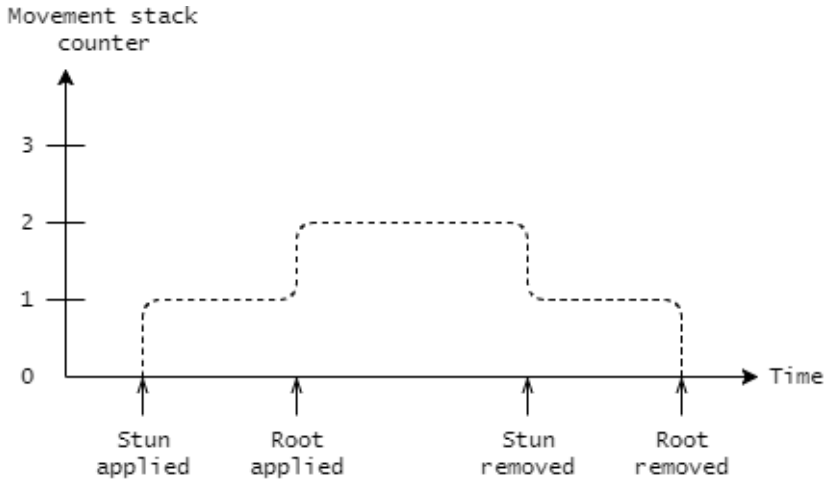


Figure 1: An example of how the input restriction stack works.

is not our work. [7] Our additions to this implementation are functions for smoothly changing the view angle and view radius. This is handled by *Coroutines* in the *FieldOfView* component as seen here. The coroutine could been used as separate update loops, that runs and interpolates the value until it's at the target value, and then remove itself. The initialization of the angle coroutine can be seen in Listing 1.

```
public void SetViewAngle(float newAngle, float speed) {
    if(angleCoroutine != null) {
        StopCoroutine(angleCoroutine);
    }
    angleCoroutine = StartCoroutine(ViewAngleLerp(newAngle, speed));
}
```

Listing 1: Code snippet for starting the angle coroutine.

Keeping a reference to the started coroutine is necessary to prevent starting multiple coroutines, that all try to change the angle. In the case where multiple coroutines try to change the angle in opposite directions the loops could be stuck forever. By stopping any currently running routine before starting the new one there's always only one running, which will only move the current angle to the target angle with the speed given.

3.3.4 Health and currency

Player Health

The *PlayerHealth* component handles the functionality related to the player health, and is what other components use to apply damage to a player. It takes advantage of the *SyncVar* functionality to synchronize the health across all clients, allowing the script to display the correct health color indicator for each player. Whenever damage is taken it will display a damage flash for every client, and update the UI for the local player. In addition to taking the damage, the source player of the damage will be stored. This allows the component keeping track of the score to extract which player got the killing blow to a player.

Currency

The *PlayerCurrency* component works very much like the *PlayerHealth* component, so we grouped them together in this section. It uses a *SyncVarHook* to keep the currency value synchronized between the server and the client. When the currency is changed by the server, the *SyncVarHook* will check the difference between old and new currency and tells the *PlayerUIHandler* to update the UI to the new value. When the *PlayerUIHandler* updates the currency, it will also play an animation showing the difference being added or subtracted, as seen in Figure 2.

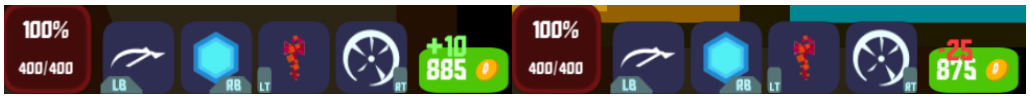


Figure 2: Showing clearly when currency is added or removed

3.3.5 Player Status

The *PlayerStatus* component handles the modifier instances and status effects applied to the player. Modifiers are applied on the server, which then synchronizes these out to every client. The instances of a modifier are split up in two: *ModifierInstanceServer* and *ModifierInstanceClient*. The server instance actually controls the functionality of the modifier, and uses either a duration loop, or a tick loop with a given interval between each tick (count). The client instance is the representation of this modifier on each client, and holds a reference to the actual *Modifier* class. This gives the instance access to the modifier data, like the visual element added by the modifier, and in case of the local client, the modifier UI element.

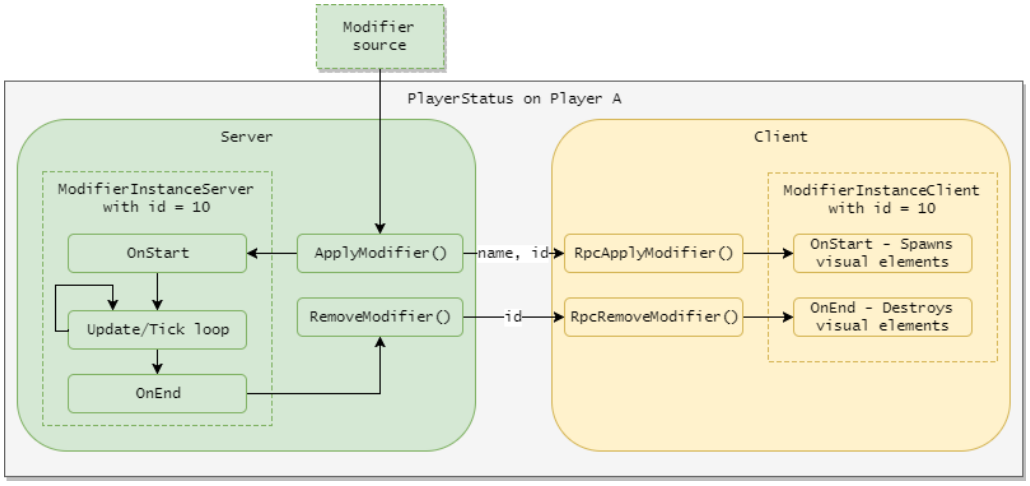


Figure 3: The program flow on Player A when the server applied a modifier.

Figure 3 shows the flow when the server applies a modifier to a player. Having two separate instances simplifies the synchronization process, and the server tells the clients to apply a certain modifier by sending the modifier name. However, in cases where multiple modifiers with the same name can be applied we need a way to tell these modifier

instances apart other than their name. Otherwise, whenever a modifier effect ends on the server, and it wants to synchronize that to the clients, the client wouldn't know which modifier to remove. By giving a unique id to any modifier applied on the server, it can use that id for both the server and client instance. The *Modifier* architecture will be described further in Section 3.4.

3.4 Modifier architecture

All modifiers derive from an abstract *Modifier* class. These do not exist as instances, but utilize the *ScriptableObject* class in Unity that is discussed further in Section 8.1.1. They rely on the *ModifierInstanceServer* and *ModifierInstanceClient* instances to actually run the code present in them. They act more like data containers, and have references to the modifier icon, visual elements and stats. This is why all the functions in the *Modifier* class takes the *PlayerStatus* instance as a parameter, as it can then use that to get references to whichever component on a player it wants to modify, whether it is the player input, or the player object.

The *Modifier* class has quite a few virtual functions, which can be overridden by modifiers depending on what they want to do. They are handled, and called by the modifier instance classes, which means modifiers can be networked through these virtual callbacks. These callbacks are called when the modifier is applied, and when it's removed, with separate functions for the server, local client, and every client. The *ModifierInstanceServer* calls these on the server, in addition to the *OnServerTick* function used for tick loops. And the *ModifierInstanceClient* calls these both for the local client, and every other clients. The code snippet in Listing 2 is an example of this in the stun modifier, which cancels any abilities being used, and restricts player input.

```
public override void OnLocalClientStart(PlayerStatus playerStatus) {
    playerStatus.GetComponent<Docking>().CancelAbilities();
    playerStatus.GetComponent<PlayerInput>().SetInputRestrictions(...);
}
```

Listing 2: Code snippet that runs for the local client when the stun modifier is applied.

3.5 Docking, Docking Kit, and Ability architecture

These three components are the underlying architecture for the main player gameplay, and are tightly intertwined.

3.5.1 Docking

The *Docking* component is central to the networking in *Dockit League* and handles the connection between the player and the *DockingKit* component. All networked objects in Unity needs a *NetworkIdentity* component, and these are constrained to root objects. [8] Since the *DockingKit* object is a child of the player object, it can't be networked directly. Our solution to this is therefore to handle all the networking needed for the Docking Kits through the *Docking*. The alternative would be to have the Docking Kit as the root in the scene hierarchy, and either move the entire object position to the owning player each frame, or put the kit visuals as a child of the player. This would keep the *DockingKit* as the root, allowing it to be a *NetworkBehaviour*. However, the same issue would have

to be faced for each ability, as they're a child of the docking kit. Having every ability in the root of the scene would lead to a very cluttered scene hierarchy, and way more networked objects than necessary.

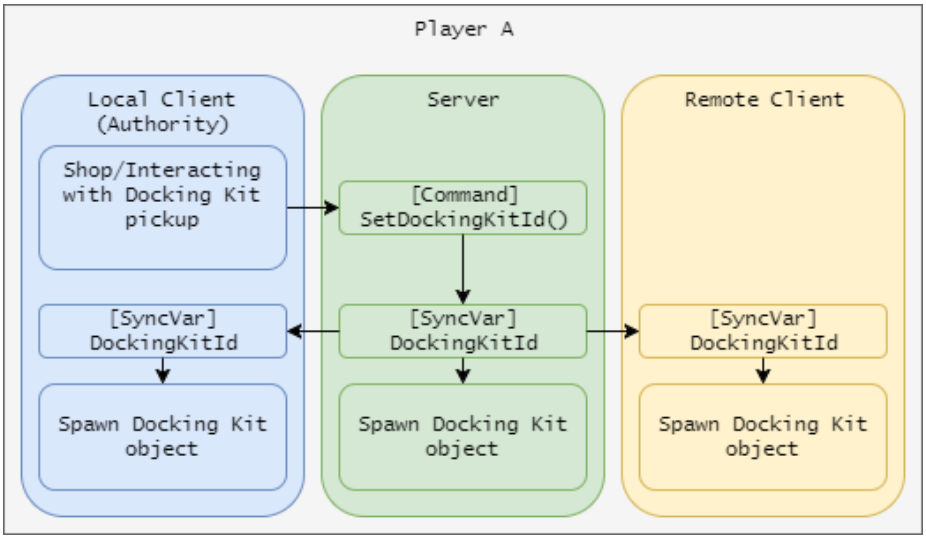


Figure 4: The program flow on Player A when the local player changes docking kit.

Since the docking kit object itself isn't networked, the clients need to spawn in the object themselves. As shown in Figure 4, this is initiated by the local client, the player with authority, in this case *Player A*. It's synchronized by passing an enum of the docking kit ID. Keeping this enum as a SyncVar will let the server update this value on every client, a function hook then runs whenever it changes, and updates the representation of *Player A* across every client (and the server) by spawning the object locally.

3.5.2 Docking Kit

The *Docking Kit* was initially the only connector between the abilities and the docking. The abilities only knew the kit it belonged to, and not the docking. However, this lead to a lot of code bloat in the *Docking Kit* that essentially just called a function in the *Docking* component for the abilities. Later this was changed so the abilities have a reference to the *Docking* directly. This reference is set up through an initialization process handled through the *Docking Kit*, which serves its purpose more as a container class for the abilities, and has all it's abilities as children.

3.5.3 Ability

Ability is an abstract base class that all abilities inherit from. It has a mix of virtual and abstract functions that can be overridden by abilities if needed. This gives the different abilities flexibility in the way they're implemented.

Two of these functions are related to the local player input: *ButtonDown* & *ButtonUp*. These are called by the *PlayerInput* component, through the *Docking* & *Docking Kit* whenever an ability button is pressed, along with the ability id, and if the button was pressed or released. This gives no synchronization of the ability, as mentioned in the previous section, that has to go through the Docking. This is shown in Figure 5, where functions with background color blue, green, and yellow run on the local client, server, and remote

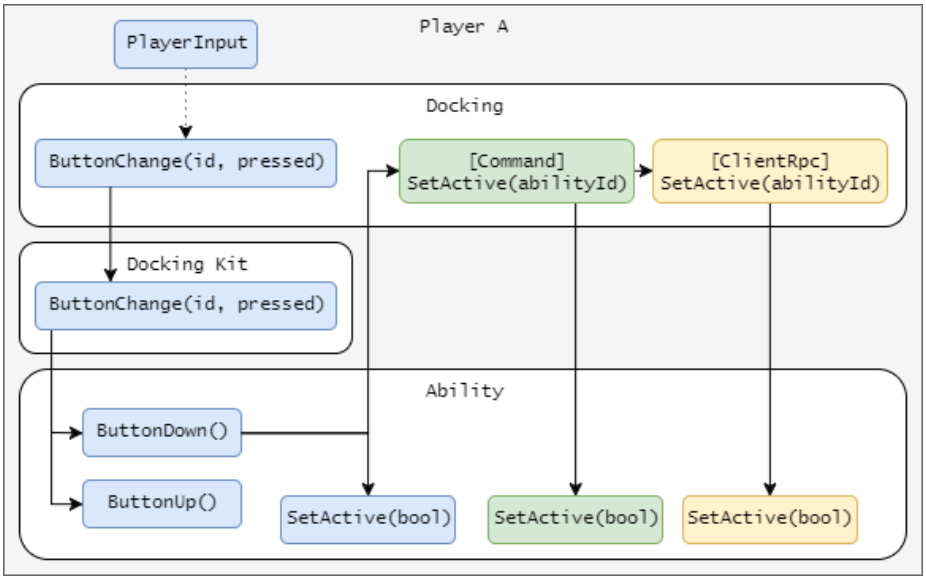


Figure 5: The program flow on Player A when the local player presses an ability button.

clients respectively. In order for the different remote clients to run related ability code, like play animations or sounds for instance, the local client calls *Commands* from the input functions on the docking. The docking then runs the *SetActive(bool)* function on the server, and then uses a *ClientRpc* to run it on every client, except the local client, as that client already ran the function. This means that in the current solution the abilities only have two active states, active or deactivated. This could for instance be expanded to an int, if abilities needed more states.

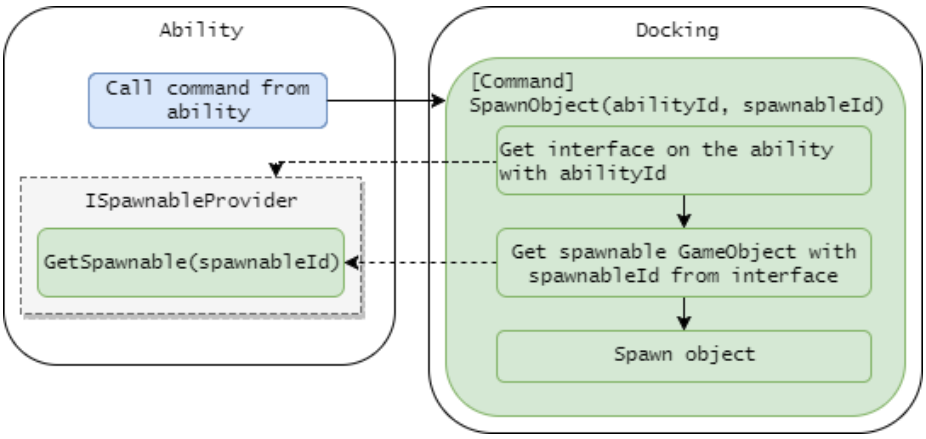


Figure 6: The program flow when an ability on the local player spawns an object.

In addition to the virtuals from the base class, abilities can further expand their functionality with interfaces. *ISpawnableProvider* is an interface used if the ability needs to spawn an object. The ability can't handle that itself, because it needs to be networked. A command is called on the docking, but in order for the docking to know which ability called for an object to be spawned, it takes an int as the ability ID. The docking can then get a reference to the ability list through the docking kit, and get the *ISpawnableProvider*

interface to call the function that gets the object to be spawned, as seen in Figure 6. This is a necessary step, since the network calls only support basic data types, we can't pass the spawn object reference directly. It allows for the object reference to be held in the ability directly, and the interface allows the docking to get a reference to the correct object when running on the server. In some cases the ability might want to keep a reference to the object spawned. Therefore an additional interface, *ISpawnableReferenceProvider*, and a *Command* & *TargetRpc* function in the docking was added to allow the server to pass a reference back to the local client. *IModifierProvider* uses the same principles for applying ability modifiers.

These provider interfaces are good for generic functionality in the abilities, but it doesn't give the abilities much flexibility when implementing the gameplay. Therefore additional interfaces were implemented to give the abilities more options: *IServerCallback*, *IClientCallback*, and *ITargetCallback*. These callbacks are tied to the networking attributes *Command*, *ClientRpc*, and *TargetRpc* respectively. This basically gives the abilities a way to be networked without being *NetworkBehaviours*. These interfaces were implemented with generics, to allow different abilities to use different parameters for these callbacks. However, the Unity networking doesn't support generic functions, nor overloading, with these attributes. Therefore, separate functions with different names had to be created in the docking. Unity's networking limitations are discussed further in Section 5.3.

3.6 Docking Kits

This section covers a mix of the final design of the kits as well as well as some more technical details on their abilities. Additional documentation on docking kits can be found in Appendix D.

3.6.1 Basic Kit

The Basic Kit is the starter kit, and a kit that's always available for the player. It is the only kit with just one ability, and exists to make sure the player has a kit, in case they can't afford or pick up any other kits.

Basic Attack

A forward melee attack with low cooldown and low damage. It extends the little spike in front of the player model to prod enemies and deal damage. Figure 7 illustrates the maximum range of the Basic Attack.

3.6.2 Bomber Kit

The Bomber Kit is a kit based around explosions. It has medium health, low speed and high damage. The abilities of the bomber kit are focused on area of effect damage and knockback to maneuver itself and displace enemies, while also focus on placement and timing for placeable mines. For all the explosions in this kit the damage and knockback is reduced the further away from the explosion the affected unit is.

Explosive Mine

The first ability is an explosive mine that the player can place, which will trigger when an enemy steps on it. The ability itself uses a script, *ExplosiveMineSpawner*, to spawn and keep a *List<GameObject>* for the mines it spawns. There is a limit of active mines

Figure 7: Showing the Basic Kit with its basic attack active and extended.

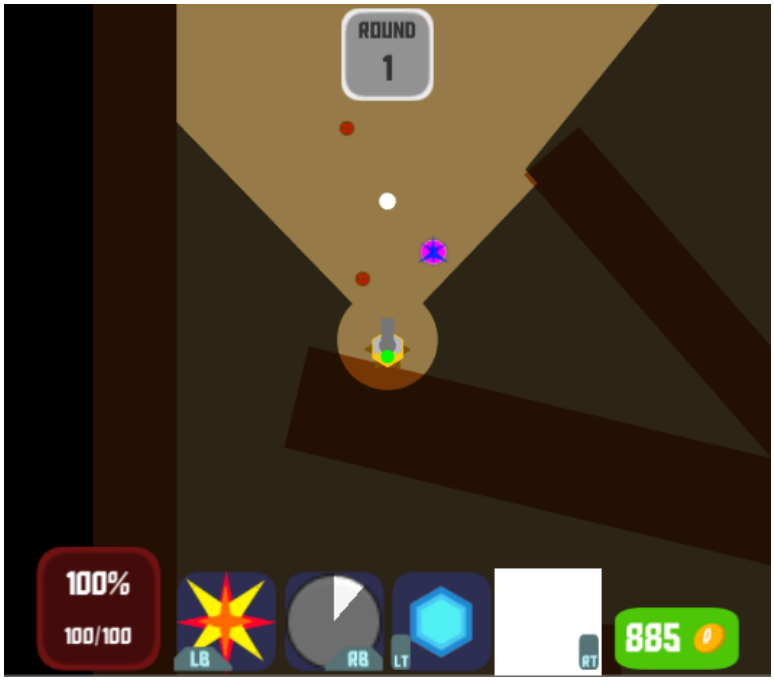


Figure 8: what the Bomber Kit looks like in game, with a few of the icons still as placeholders.

at the same time. When the limit is exceeded, the ability will remove the first mine placed from the list and destroy it. Since each mine handles collision themselves with *OnTriggerEnter*, there was a need to know which mine to remove from the list of active mines. To quickly get a unique ID for the mine, we used *GameObject.GetInstanceID()* at the time of creation and stored this to check the list for which mine to remove in the future. When the mine is triggered by an enemy player, it will start a function *Explode()* which will start an animation and apply forces as well as damage to the players in the area. When the animation ends the mine will destroy itself with an *AnimationEvent* and tell the *ExplosiveMineSpawner* that it's destroyed and should be removed from the list of active mines. Figure 8 shows two mines placed and waiting to trigger.

Grenade Launcher

The second ability is a grenade launcher. It fires slow, powerful shells that will explode after a certain time or on contact with an enemy. These shells will not explode on contact with the player who fired them, but will deal half the damage to the player when it explodes. They will also bounce off obstacles like walls using a bouncy *Physics Material* added to the shell's *SphereCollider* to make it potentially harder to dodge and master. When the grenade shell explodes, it will check for colliders using *Physics.OverlapSphere* which will return all the colliders in the radius of the sphere created from this. From these colliders the valid targets, self and enemies, will be affected by the explosion. The explosive mine and the grenade shell are much the same in the way they both check for targets with *Physics.OverlapSphere* and apply force with *Rigidbody.AddExplosionForce*. This force is applied by the server to prevent varying results depending on the world state when it gets applied, see Section 5.2.1 for more on consistent force application.

Remote Mine

The third ability in this kit is a remote mine that uses much of the same code as the explosive mine, but the main difference here is that you can only have one(1) active at a time and will not trigger on contact. When the player uses this ability again while having a mine active already, it will trigger the remote mine and apply a stun in an area around the mine to all enemies. This ability gives the kit great utility to start a team fight with the proper placement. Like the *Explosive Mine*, this mine gets spawned by the ability using the *ISpawnableReferenceProvider*. This way the ability which spawns the mine will keep a reference to the mine it spawned, and can therefore trigger it when the player activates it a second time. In Figure 8 you can see a purple remote mine placed by the player.

Blast

The final ability of the Bomber Kit. As the name suggests, it fires a rectangular blast in front of the player that will knock the player backwards, and enemies away. Due to the kit's low movement speed, this ability is a great tool to get out of a bad situation and keeping distance to the enemy. The reason for using a *BoxCollider* instead of something like a cone, is due to the limitation of primitive colliders in Unity3D, briefly explained in Section 8.1.5. For applying this ability the box collider is disabled until the ability is used. Then it is activated for a frame so the *OnTriggerEnter* function can apply forces before it is disabled again.

3.6.3 Boomerang Kit

The Boomerang Kit is a high speed, high damage kit with low health. The kit's abilities are primarily focused on augmenting the boomerangs that the player can throw. The boomerang itself has a small field of view circle around itself which allows the player to still see the boomerang if it has been thrown across a wall or other areas where vision is limited.

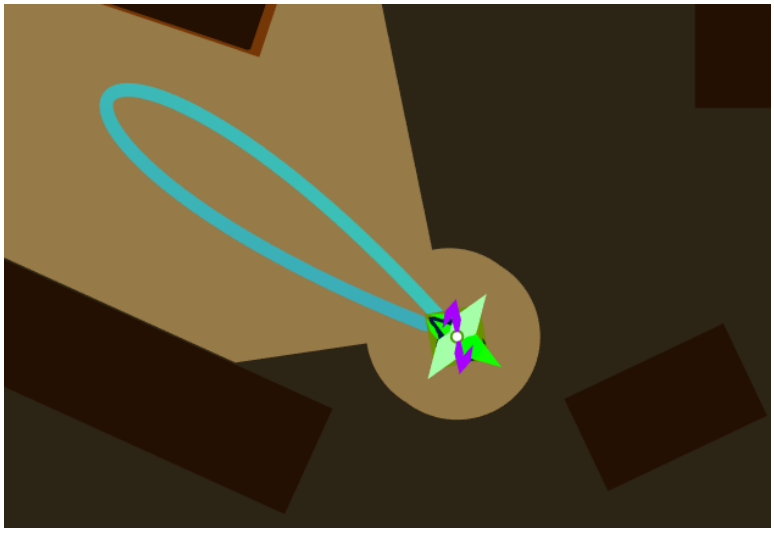


Figure 9: A screenshot from the game showing the approximate travel path of the boomerang when the ability button is held.

Boomerang Throw

The primary ability of the Boomerang Kit is the boomerang throw. It uses cubic bezier curves to construct and display the approximate path that the boomerang will travel using a *LineRenderer* component. Figure 9 shows how the approximate path looks like for the local player using the kit. The bezier curves are also used for interpolating the position of the boomerang as it is thrown by the player. The interpolation itself is handled using a timer variable that gets updated with *Time.deltaTime* per update loop. The timer is then used as input into an evaluation function used for controlling the speed of the animation and returns a smoothed version of its input. This output is then used as the input time for the bezier curve's interpolation. A more in-depth look at the interpolation of the boomerang throw can be found in Section 5.4.

There are also two additional boomerangs that can be activated with the final ability in the kit. These have their own bezier curve control points and *LineRenderer* components. In order to perform the same operations on all boomerangs we have a array of structures containing the bezier control points and stored curve handles of each boomerang. This allows us to modify the positions of all the bezier control points directly in the editor which makes it easy to control the shape of the curves.

Boomerang Root

The second ability in the kit applies a root modifier to any enemy players within a certain range of the boomerang at the time of activation. It displays a range indicator for the local

player whenever the ability is off cooldown, making it easier to time the activation of the ability. Activating the ability will play a short ability animation around the boomerang on all clients and enables the *SphereCollider* component that checks for enemy players. The collider stays active for 0.5 seconds to make it a bit easier for the player using the ability to hit others as the boomerang moves at a fairly high speed.

Boomerang Vision

The next ability in the kit is a fairly simple ability that revolves around using a self applied modifier to control the state of its effects. Using the ability takes the *FieldOfView* component of the boomerangs and quickly interpolates the radius of these to a higher value, giving the player a large circle of vision for a short duration. The additional vision is only something the local player can see while other players will see a circle indicator around the boomerang that showcases the new vision range. Once the duration of the modifier runs out the vision range quickly interpolates back to its default value.

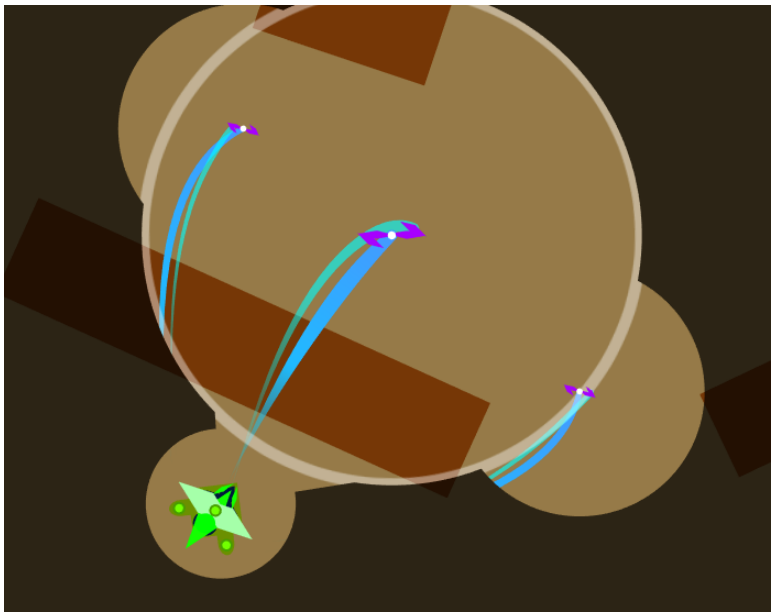


Figure 10: A screenshot illustrating a player using the final ability of the boomerang kit to throw multiple boomerangs in conjunction with the vision ability.

Multirangs

The final ability of the Boomerang Kit is another self applied modifier that lasts for a few seconds and makes the next boomerang throw contain three boomerangs instead of one. This is the key ability of the kit that allows for a truly massive damage output given that the player is able to hit with all of the boomerangs. The self applied modifier is removed prematurely once a boomerang throw is made, but can also wear of naturally if the player avoids throwing any boomerangs during its duration.

The additional boomerangs granted from this ability are also affected by the root and vision abilities, albeit with a lesser range as seen in Figure 10. The script for the boomerang throw handles the interpolation of the two additional boomerangs while the script for this ability primarily manages the state of the buff. This includes self applying

the modifier buff, managing the state of any extra visual elements and playing animations.

3.6.4 Brawler Kit

The Brawler Kit is a slow moving melee oriented kit, but has high health and multiple tools for dealing with enemies who fight at range.

Axe Slash

The first ability is a short cooldown slash with the Brawler Kit's axe. We are using *OnTriggerStay* instead of *OnTriggerEnter* as the collision callback for this ability. This is due to the fact that the animation for the slash is very quick and only lasts a few frames at its start position. Using *OnTriggerEnter* in this case would make the callback frequency too low at the start of the animation, essentially making it impossible to hit players nearby the start position of the axe. This happened because the first few frames would only trigger collision callbacks for the player's own overlapping collider instead of others. This issue is alleviated by using *OnTriggerStay* instead with a stored list of hit players. The list is reset after each swing and makes sure that any damage is only applied to each player once.

Lifesteal

The second ability in the kit is a self applied modifier that lasts for a certain duration, making the next axe slash deal increased damage and heal a certain percentage of the damage done. It works in a similar fashion to the final ability of the boomerang kit. The self applied modifier can wear off naturally or be removed after colliding with another player. Having the modifier active also changes the visuals of the axe to show that the ability has been used.

Projectile Reflect

The third ability is a self applied modifier that reflects the velocity of any projectiles that hits the player while active. This is handled using C# interfaces. Any projectile that is reflectable needs to implement the *IReflectable* interface. This in turn allows the ability to check whether the interface exists on any colliding projectiles and ask them to reflect their velocity. The ability itself does not perform any reflection directly as this is what the projectiles implementing the interface have to contain.

Stun Grenade

The final ability makes the player throw a stun grenade that explodes after a short while, providing temporary vision for all players(see Figure 11) and applying a stun modifier to anyone looking towards the explosion center. The grenade itself is a *SpawnableObject* and controls the application of modifiers, the temporary vision and any visual elements related to the grenade. The player on the other hand has a simple script that handles the spawning of the grenade. The grenade enables a large sphere collider on explosion which checks for any players within its range. A raycast is then used in conjunction with a dot product to check for players looking towards the center of the explosion. Players with obstacles between themselves and the explosion center will not get stunned as the raycast check will end up failing.

Figure 11: A screenshot from the game showcasing the additional vision range granted from an exploded stun grenade

3.6.5 Marksman Kit

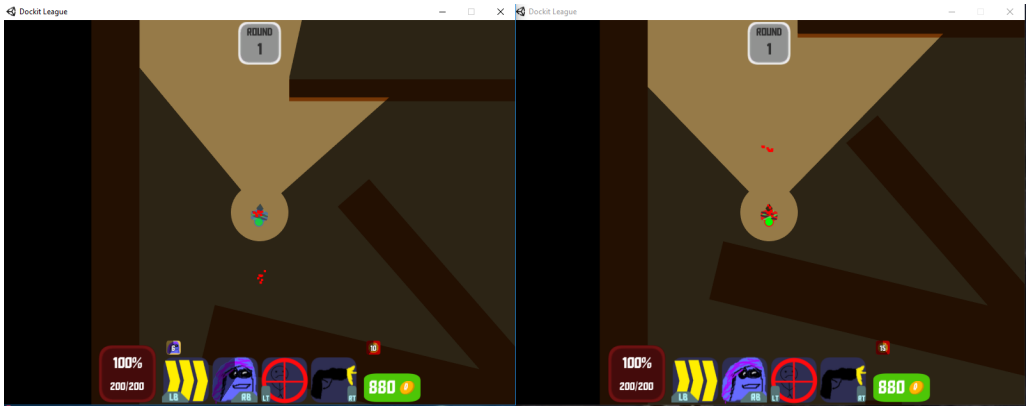


Figure 12: Demonstrates how the stealth and track abilities works, attaching red particles to the player showing the location through fog of war.

The Marksman Kit's main focus is to gather information and being elusive to the enemy. With a toolkit that allows it to quickly dash out of harms way, turn invisible and give information about the other team, it is able to give your team the upper hand in a fight. It has medium health, damage, utility and speed, but in the right hands has the potential to be a powerful kit if not dealt with. After playtesting it is worth mentioning that this kit will be renamed later to fit its purpose better.

Dash

The first ability is the dash ability. A short cooldown ability that applies a small amount of force in the direction you're moving. Since the *PlayerInput* script already has this direction vector stored for player movement, we're simply retrieving the direction from

there.

Stealth

Next up is the stealth ability. This ability lets the player hide its visuals from other players by changing the alpha value to zero. To give it a short fade time, a *coroutine* using a *for-loop* decreases the alpha value over time while accounting for fade time. Stealth can only be broken by using the ability again, fire the projectile ability of the Marksman Kit or when the duration runs out. There was a decision made to not break out of stealth upon taking damage, which is a game mechanic we see often related to stealth or invisibility in games. Therefore the balance around the up-time and frequency of which you can use the ability is important when there is no real way to force the player out of stealth. While in stealth the player also benefits from the stealth modifier. This is a modifier that amplifies the Marksman Kit's next projectile shot from stealth, boosting damage while applying a slow and silence to an enemy if it hits. This is applied to the *projectile* fired when the projectile is initialized, giving it a reference to the stealth buff that applied it with the data of the buff.

Track

This ability is a tracking debuff the Marksman Kit can apply to enemies. It will then reveal the position of the marked player while amplifying the damage taken from all sources, see Figure 12. To apply this debuff the player fires in a straight line in front of him with a *Physics.RaycastAll* and applies it to the first player it hits that isn't covered by an obstacle. We could potentially have done this differently by using *Layers* in Unity to filter out which game objects should be ignored from raycasts and not. Although *Physics.RaycastAll* doesn't guarantee a sorted order, all *RaycastHit* have a variable *distance* from the point of impact to the origin of the raycast which we can sort with a lambda. Listing 3 shows how this is sorted and applied to the first enemy hit that isn't covered.

```
void IServerCallback.ServerCallback(int functionId) {
    var hits = Physics.RaycastAll(transform.position, transform.forward, castRange)
                    .OrderBy(h => h.distance);

    foreach(RaycastHit hit in hits) {
        if (hit.collider != null && hit.collider.CompareTag("Obstacle")) {
            return;
        }
        if (hit.collider != null &&
            this.docking.CheckDamagable(hit.collider.GetComponent<Player>())) {

            PlayerStatus playerStatus = hit.collider.GetComponent<PlayerStatus>();
            if (playerStatus != null) {
                playerStatus.ApplyModifier(trackInfo);
                return;
            }
        }
    }
}
```

Listing 3: Code snippet for applying Track to first enemy not behind an obstacle.

Projectile

Finally we have the last ability we simply called Projectile. It is a basic "fire in a straight line"-type of projectile, and it is the only source for this kit to deal damage directly with. It

has an *Ability* script *ProjectileSpawner* which instantiates projectile prefabs and *Initialize* them with the stealth buff reference and a *Boolean* indicating if it's active. Each projectile has its own lifetime, destroying itself after a while if it doesn't hit any players or obstacles. For physics we've chosen to not have a constant velocity, but rather apply *Impulse Force* when the projectile is fired for this. This worked fine when we tested it in the editor, but after playtesting it proved that the projectile would lag, unlike the projectile from the Sniper Kit. Therefore this is a change to be made to use the *Transform* instead of the *Rigidbody* to move the projectile.

3.6.6 Sniper Kit

The Sniper Kit is the long range focused kit. Keeping your distance gives you time to charge, which is important with varying damage output and precision depending on charge time.

Shackle

The first ability is Shackle, which launches a bola that travels a short distance before disappearing. It is designed to give the player in this kit a way to get away from enemies getting close, since staying at range is beneficial. If the bola collides with an enemy player it has three different outcomes depending on the scenario. If there is nothing behind the player hit, it will apply a slow modifier. If there is an obstacle behind them, the stun modifier is applied instead. And the third option is if there is another enemy player behind the first one hit, they're both stunned. This is done by firing a raycast from the center of the player hit, in the direction the bola was travelling. If this doesn't hit anything, we can apply the slow. If it hits something, we first check if it's an obstacle, if so we apply the stun. If it's not a wall, we can check if it's another player, and stun both of them if that's the case. The three outcomes are shown in Figure 13.

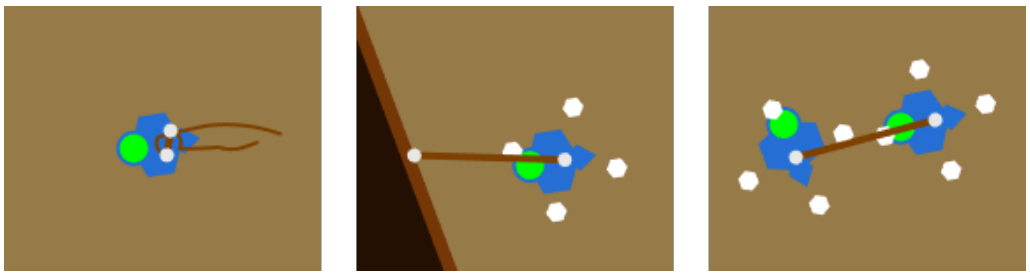


Figure 13: Three screenshots from the game showing the slow, stun against wall, and stun against another player from left to right respectively.

Focus

The second ability is Focus, and is more of a utility ability used in addition to the Slingshot ability. It will root the player in place, and increase the line of sight distance, but decrease the view angle using the lerp functionality in the *FieldOfView* script for a smooth transition. It will also move the camera forward slightly with a similar functionality in the *PlayerCamera* script. The use of this is displayed by moving the sling mounting forward, and retracting the sling.

Slingshot

The third ability is Slingshot, and the damage ability for this kit. On button press it starts charging the slingshot precision, starting at the view angle and moving towards 0 using an animation curve. The ability uses a different curve for the initial charge towards 0, and a hold curve that reduces the charge angle back to the view angle over a few seconds after reaching max precision, preventing the player to keep max precision for a long time. When the button is released it will fire a projectile at a random angle between the charge angles, meaning the smaller the charge angle is, the more precise it'll be. The projectile damage, speed, and lifetime depends on both the fire angle and the charge angle, which means that two projectiles fired at the same angle might have different stats depending on the charge time. This is simply to make sure a charged attack will always do more damage than a lucky shot with no charge. The ability mid-charge is shown in Figure 14 where the white lines represents the charge angles.

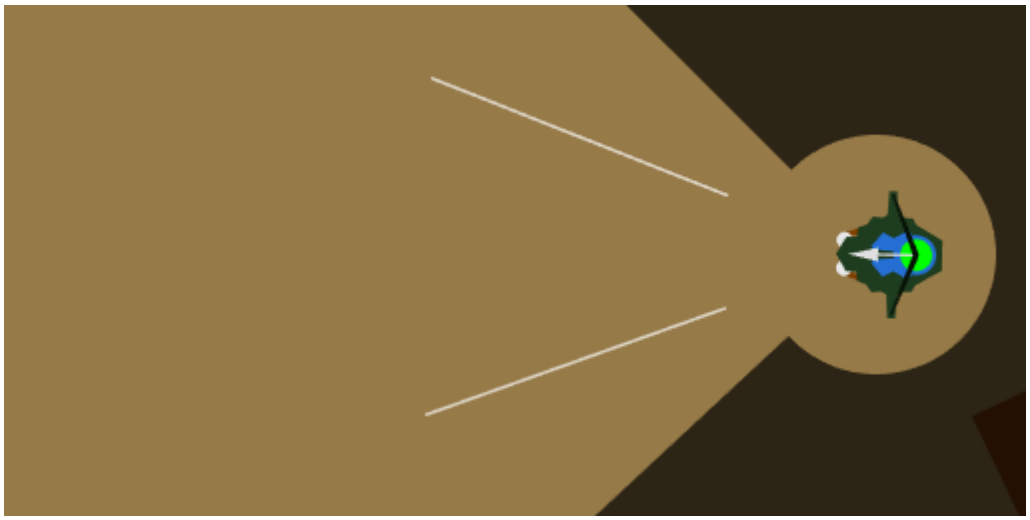


Figure 14: A screenshot from the game showing the Slingshot ability mid-charge.

Zipline Gun

The final ability is the Zipline Gun, and like Shackle, it's a way to get away from enemies by setting up an escape route for when a situation gets rough, or just quickly move to another vantage point. This ability has two different states, the first is the initial state, when nothing has been fired. When used in this state a zipline object is spawned on the server, and the server side of the ability keeps a reference to that object. The second state is when we've already set up the first zipline point, we then fire the second point to complete the zipline. There are a few restrictions when setting up the zipline, and it needs to constantly check if any of these are broken. Initially, when we fire a point it needs to hit an obstacle, this is checked by firing a raycast in the direction, with the given range, the zipline is fired. If this is anything other than an obstacle or out of range, the zipline setup is invalid, and the zipline object is destroyed. After firing the first point, a *Coroutine* is started that keeps checking if there is anything interfering the zipline by firing raycasts between the player and the first point. When firing the second point, this

check fires between the first and the second point instead, preventing setting up ziplines that cuts across a corner for instance. Figure 15 shows the kit between the first and the second state, and areas where a setup is invalid.

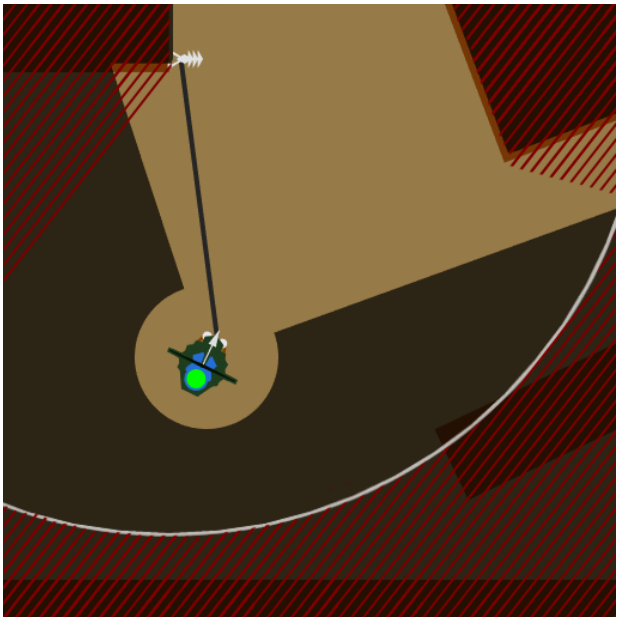


Figure 15: Showing the state between the first and second point fired, with invalid areas covered. (Red lines are not present in-game)

3.6.7 Tank Kit

The Tank Kit's purpose is being the front line for the team, having a lot of health allows it to soak up the damage dished out by enemies by redirecting projectiles. It is a very slow kit, but compensates by having abilities for pulling both itself and others around.

Reflect Shield

The first ability is Reflect Shield. This is a shield that can be activated, and will stay active for a certain duration. Any object colliding with this shield that implements the *IRedirectable* interface will be redirected, and the Tank Kit player will take ownership of the object redirected. The new direction of the object is the same direction as the Tank Kit player is facing. This is done using the *OnTriggerEnter* callback, and getting the interface from the object passed in. An object is restricted to only be redirected once per ability use, this is handled by adding any redirected objects to a list, and simply checking if the list already contains the object that's to be redirected. This list is then cleared when the ability is deactivated.

Force Field

The second ability is Force Field. While the Reflect Shield is all about moving things away, the Force Field is about pulling things in. This also affects objects with the *IRedirectable* interface, and the implementation of that part of this ability is similar to the Reflect Shield. However, this ability redirects object towards yourself, and doesn't take ownership of the object, meaning the Tank Kit player will take damage from the objects

pulled in. Here the synergy with the Reflect Shield comes to fruition, as the shield can reflect objects pulled in by the field. In addition to pulling objects in, this ability also pulls in other players, compensating for the lack of movement speed, as well as synergizing with the Power Saw. This is also handled in the *OnTriggerEnter* callback, and calls one of the *TargetRpc* function in the player for adding force. The implementation of those are discussed in Section 5.2.1.

Power Saw

The third ability is the Power Saw. This has two stages, the initial state is a wind-up for the release of the sawblades. This will do damage to enemy players coming in contact with the sawblades, using the *OnTriggerEnter* callback. To prevent players being hit multiple times consecutively, they're added to a list the first time hit. If the callback triggers on a player, it only does damage if the player isn't already in the list. This list is then cleared when the swing is completed. When the wind-up state is complete the sawblades are released. At this point they're spawned in as actual objects by the local client calling a command on the *Docking* script, which spawns the object on the server. The cooldown on this ability can be reduced by picking up sawblades that has come to a halt. This is handled by the *Sawblade* script, which is a component on the sawblade object spawned. If an object with a *Docking* component enters its trigger, it will retrieve all the abilities, and check if any of the abilities is of the type *PowerSaw*. If so reduce the cooldown of that ability through the docking, and delete the sawblade object, as shown in Listing 4.

```
for(int i = 0; i < dockingKit.abilities.Count; i++) {
    if(dockingKit.abilities[i] is PowerSaw) {
        docking.TargetReduceCooldown(..., i);
        NetworkServer.Destroy(gameObject);
        break;
    }
}
```

Listing 4: Code snippet for checking if the player trying to pick up the sawblade has the PowerSaw ability.

By using a *for* loop here, we can use the iterator count as a parameter to the reduce cooldown function in the docking, that way the docking knows which ability to reduce the cooldown for without the need to specify it anywhere, and having to manually update it if the ability order is reordered.

Hook Shot

The final ability is the Hook Shot, which fires out a hook when used. Whenever the hook hits an obstacle, or a player, it will pull the Tank Kit player towards the point the hook connected with its target. If the hook hits an enemy player it will also apply a root to that player. The hook animation is handled locally by every client, and is in this case animated by the script. The ability utilizes both the *IServerCallback and IClientCallback* interface. The server callback is for launching the hook, and passes along two *Vector3*: The launch position, and launch direction. The server then uses the client callbacks to pass these to every client, allowing them to replicate the hook animation as precisely as possible. The reason for this is that the hook is launched in the player forward direction, and the player rotation on the server and rotation on the local client isn't necessarily the exact same, especially when a player rotates as they're triggering the ability.

The server can't simply launch the hook forward, because the forward direction the player had when they triggered it, isn't the forward direction when the server receives the trigger message. The hit registration is handled by the server, which will use callbacks to let the clients know what to do. The default behaviour is to extend the hook to max range, and then retract it. This is what the clients will do as long as the server doesn't tell them otherwise. If the hook hits an obstacle or a player the server will tell the clients to freeze the hook in a certain position for a moment, before retracting. The hook can also hook certain objects, like the sawblades. The sawblades implements the *IHookable* interface, which is used in the *OnTriggerEnter* callback in the Hook Shot. This allows the Hook Shot to be used for picking up sawblades without actually moving to them.

3.6.8 Trapper Kit

The Trapper Kit is a utility docking kit that focuses on using its three traps to provide various types of crowd control as well as being adequately capable of fighting enemies in close to mid range using the kit's flamethrower.

Flamethrower

The first ability in the kit is the flamethrower, a self applied modifier that activates a large sphere collider that checks for objects that it can burn. One of the limitations of working with Unity3D rather than Unity2D is that we don't have access to polygon colliders. Using polygon colliders would have made it possible to create a cone collider for this ability. We are instead using a large sphere collider for the flamethrower although its shape is not necessarily as fitting. The collision callback checks for any enemy players and applies a burn modifier to these. It also checks for game objects with the *IElement* interface and applies a fire element to any such objects if found. This means that abilities from other kits that support elemental modifiers can be buffed by the flamethrower.

Trap Overview

The three other abilities use traps deriving from a base *Trap* class. A shared *TrapSpawner* script is used to spawn each of the three traps by providing it with different prefabs per ability. The *TrapSpawner* script handles spawning its given trap prefab as well as updating the visuals of the docking kit to reflect that traps have been placed. Each trap has its own visual element on the docking kit that is modified to allow other players to know which traps the user currently has placed. Figure 16 showcases these. The trap spawner script contains a reference to the spawned trap while the spawned trap is provided with a reference back to its owner. This allows the trap to only stay visible for its owner as well as communicating back to its owner whenever it has been triggered.

Acquiring the reference to the traps is handled by implementing the *ISpawnableReferenceProvider* interface. Only one trap of each type can be placed at a time so trying to replace a trap that already is placed destroys the old trap and places a new one at the player's current position.

The *Trap* base class handles generic functionality like changing the visual state of the trap to invisible for other players as well as playing animations whenever the trap is triggered. A virtual function is used to allow any children of the base class to provide custom behaviour as the trap is triggered. The base class also contains a list of players that triggered it to make sure that no modifiers are applied twice in the case that a player

Figure 16: The trapper kit has three traps. Placing one trap fades the color of its associated visual element on the docking kit

quickly moves in and out of the trap collider.

The Slow Burn Trap and the Blind Trap

The first two traps have fairly simple behaviour. Both have modifiers that they apply to the list of players who triggered the trap. The first applies a burn and slow modifier while the second applies a blind modifier.

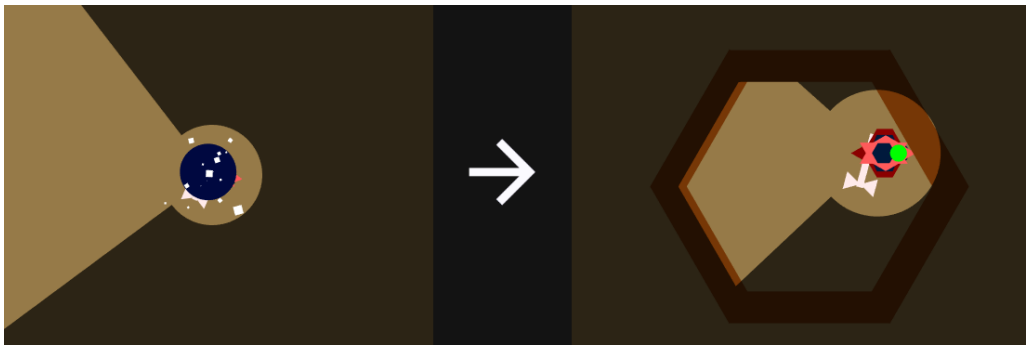


Figure 17: These two screenshots show the two main stages of the capture trap's life cycle.

The Capture Trap

The third trap has more custom behaviour compared to the previous two. It pulls in any nearby enemy players after triggering and places a temporary wall around these players to "capture" them for a short duration as seen in Figure 17. There are a lot of different visual elements like particle systems, animations, colliders and sprites that are enabled/disabled throughout the trap's lifecycle.

Applying the force for pulling players into the trap is done using a *TargetRpc* function located in the main player script while a C# coroutine is used to enable the walls after

a short duration. More information on adding consistent force to both the server and clients can be found in Section 5.2.1.

3.6.9 Support Kit

The Support Kit is a kit made to keep your team alive. It is centered around just staying alive with your team, healing them over time while slowly draining your enemies of their lives. The kit have medium health and speed to make it fairly hard to kill, while being able to negate damage and remove debuffs further increasing its sustainability.

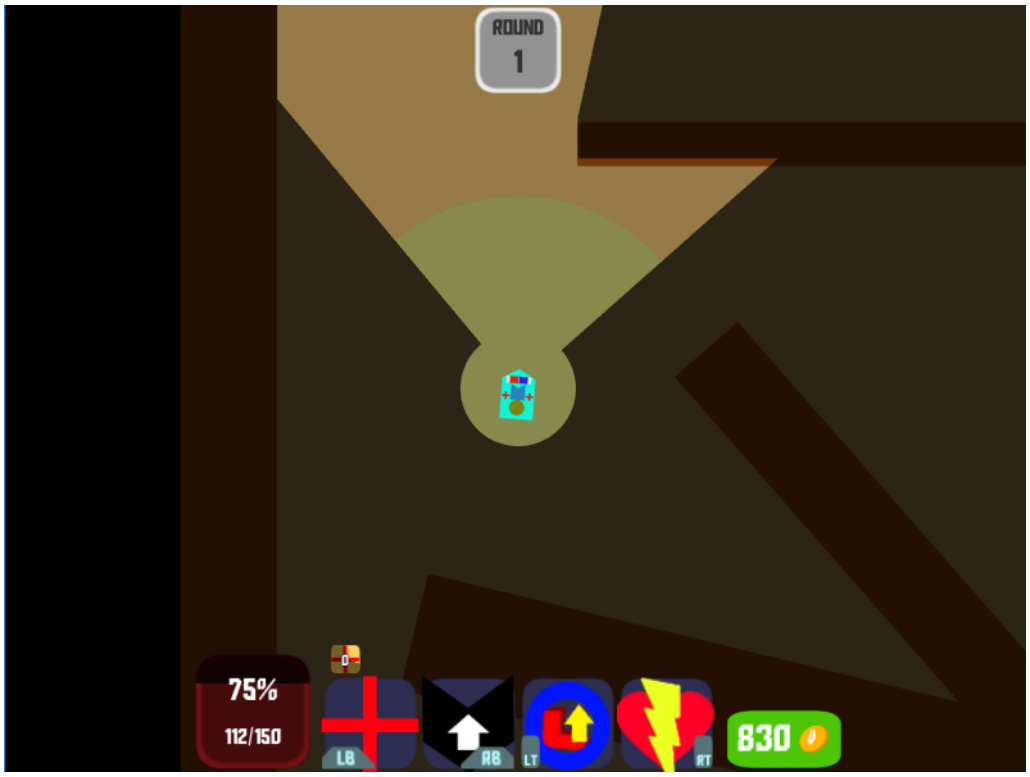


Figure 18: Shows the Support Kit with healing aura toggled on.

Healing Aura

The first ability in the Support Kit. This aura applies healing in a circle around the player that can be toggled on and off, see the green circle in Figure 18. It uses a *List* to keep track of players in the aura, and this list updates whenever players enters and exits the aura. Using the *IServerCallback* interface, the healing aura applies a buff to the players every healing-interval to that will heal them. This script also has a reference to the second ability of the kit, the Fortification Buff, which also gets applied here at the same time, strengthening the power of the healing aura.

Fortification Buff

This is a power-up to the Healing Aura. The buff received from this makes players affected by the Healing Aura take reduced damage. Unlike the Healing Aura, this is a lingering buff that lasts for a few seconds after leaving the area as well. Gives the kit a

better tool against burst-damage when used correctly.

Cleanse

This ability will emit a ring of pure energy, cleansing friendly players hit of any debuff they might have and giving a slight move speed buff for a short amount of time. As mentioned before, Unity3D has a limited selection of primitive colliders, but for this ability we made a ring collider using *Blender*, an open source 3D creation suite. When used, an animation plays that enables the collider and takes the *Scale* of the *Transform* to make it look like the ring expands. Figure 19 demonstrates what this looks like in game.

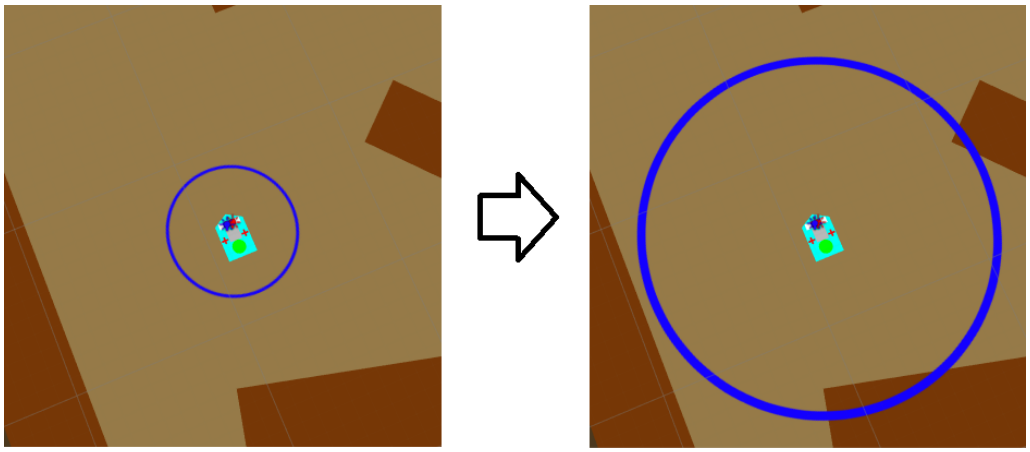


Figure 19: Screenshot of the Cleanse Ability in the editor, showing the covered area.

Health Drain

The only damaging ability of the Support Kit, draining the life of enemies around you and distributing the health drained to teammates around you. The more enemies drained, the more healing to distribute. The More allies, the less healing each ally get. For the time being we are using this simple way of doing it, in the future it should also make sure to check how much health each player loses and heals and take this into account when distributing the healing. Some variables have their names replaced to fit the page:

```
foreach (GameObject player in friendlyPlayersInAura) {
    float healthToHeal = (baseDrain * hostilePlayersInAura.Count) / friendlyPlayersInAura.Count;
    player.GetComponent<PlayerHealth>().Heal(healthToHeal);
}
```

Listing 5: Code snippet for distributing the health drained to players around.

3.7 Shop Architecture

3.7.1 Visual Layout

The visual layout of the shop can be seen in Figure 20. The layout itself is split into two halves. The left half contains purchasable shop items while the right half contains individual information on the currently selected shop item. The currently equipped docking kit is signified by a "E" while any unpurchasable docking kits due to lack of currency

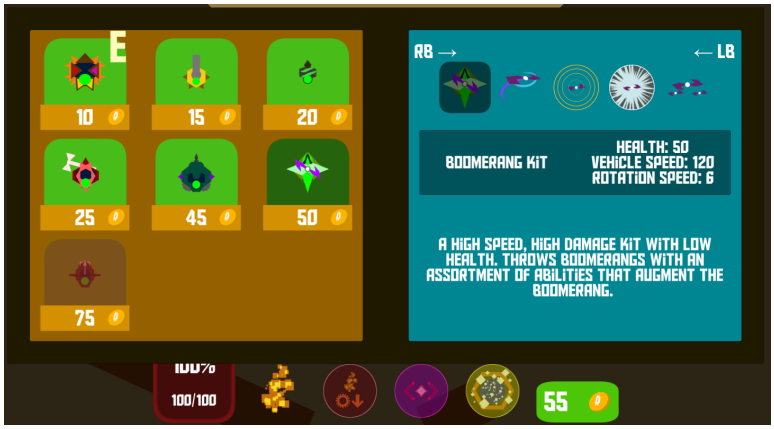


Figure 20: A screenshot showing off the in-game shop

are greyed out. The information display on the right side contains several text boxes that scripts can use to display arbitrary information about the shop item like names, descriptions and additional stats. Each docking kit contains information on the kit and its abilities which is split into five different "tabs". The information tabs for each item can be cycled through by pressing the left and right shoulder buttons on the controller as displayed on the top of the panel.

3.7.2 Scriptable objects for shop items

Scriptable objects provide an easy to use interface for developers, especially if used as described in Section 8.1.1. It allows us to simply go the the correct resource folder, right click and choose "New shop item". We can then directly fill in any data related to the new shop item in the inspector as seen in Figure 21. The scriptable object for shop items contains several pieces of data:

- The name of the docking kit.
- The sprite that is displayed in the left panel of the shop.
- The prefab for the related docking kit.
- The price of the docking kit
- An enum which we use as a ID to identify the docking kit.
- A list of structures containing description data. These structures include the sprites used for the tabs on the right panel of the shop, the name of the ability/kit and a description.

The scriptable object itself does not contain any code and is used as a data container. The data within the scriptable objects are then loaded by the menu handler and attached to instantiated shop item instances.

3.7.3 Internal shop management

The shop architecture in *Dockit League* is split up into several components:

- A script, *IngameMenuHandler* manages the shop and any interaction with it.
- Scriptable Objects are used to contain data related to the individual shop items.
- Each item instance in the shop contains the *ShopItemInstance script* which stores a

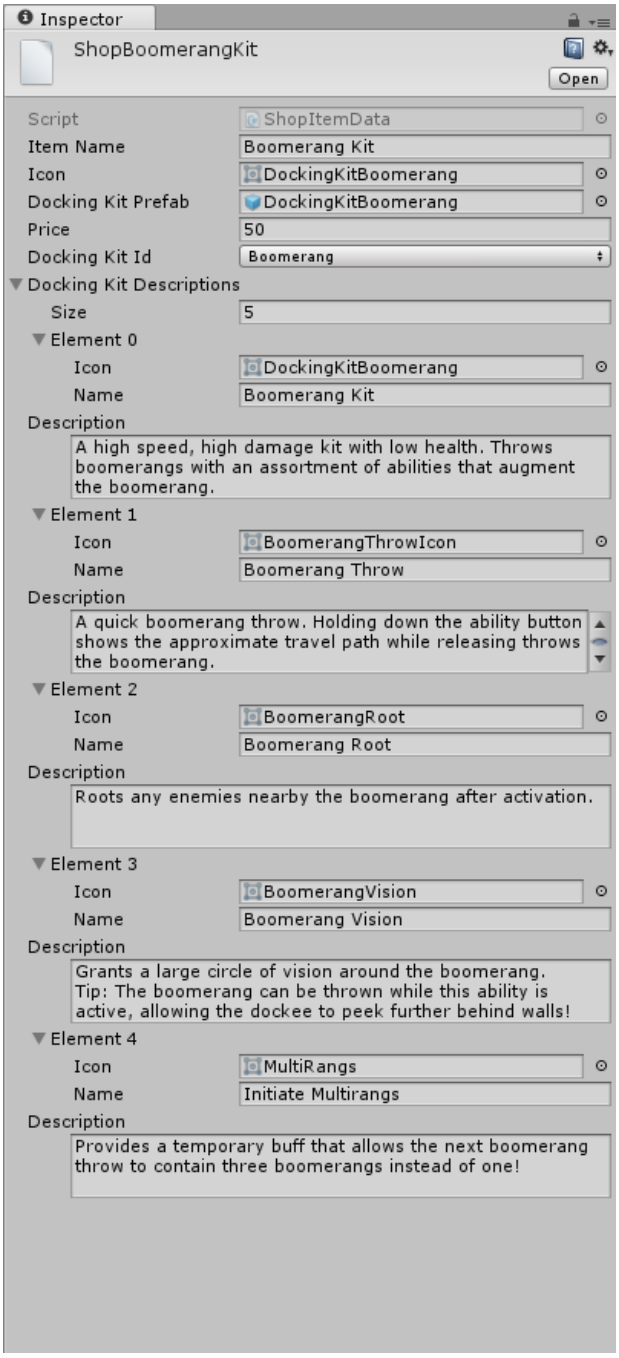


Figure 21: An image of the Unity inspector that illustrates how one of the scriptable objects from the shop looks like

scriptable object and handles the data display for it.

The *IngameMenuHandler* script starts by loading all scriptable objects within the project that contain item data. These scriptable objects are then placed in a list and sorted by price before the script starts instantiating *ShopItemInstance*'s with a respective scriptable object. The script also handles input for scrolling through the information tabs of the currently selected shop item. This is handled by incrementing and decrementing a range limited integer which is used as a index to access the correct descriptions from the scriptable object.

We use a prefab with uninitialized visual elements and a *ShopItemInstance* script to spawn in individual shop items. After being spawned, the instance is then initialized by passing a scriptable object over to it and updating all of its visual elements with the acquired data. The *ShopItemInstance* script also contains a callback for whenever it is pressed which displays a purchase verification prompt with a "yes" and "no" answer. Answering "yes" to the prompt tells *IngameMenuHandler* to complete the purchase and equips the new docking kit to the player.

4 Development Process

4.1 Agile game development

When working on any project, choosing the right software development model can help a lot. For games in particular, agile software development is a near perfect match [9]. The incremental nature of both allows developers to easily change the product based on feedback during development as well as providing workflows where clear and concise tasks can be set per increment. For this project we decided to use Scrum as our software development model due to its agile capabilities. We also wanted to grow more acquainted with professional tools like Jira and Confluence for project management as these integrate well with Scrum.

4.1.1 Our configuration of Scrum

There are many different ways of using a software development model like Scrum. In our case we decided to stick with fortnightly sprints to provide incremental progress at a steady pace. We combined the retrospective, review and planning meetings into a single "Sprint Meeting" on the Thursdays that the sprint ended as it would be easier to find the time for all group members to meet up and discuss the project's progress. Meeting notes for the sprint meetings can be found in Appendix B. Given that we didn't have any product owner, the review session of the sprint meeting consistent of showing each other the things we had been working on throughout the sprint. We also used daily standup meetings from Mondays to Wednesdays as these days had no lectures and were primarily used to work on the project.

4.2 Development Tools

4.2.1 Atlassian toolkit

We used several of the Atlassian tools when developing Dockit League as they integrate nicely with each other:

Bitbucket: Used to contain the source code

Confluence: Used to store documentation from meetings.

Jira: Used to manage the project. This includes managing sprints, playing planning poker, managing issues and updating the product backlog.

We integrated Bitbucket with Jira to allow for smart commits. This allowed us to link any commits to issues using the DOCKL-# tag where # is substituted with the issue number and track progress on individual issues. Any commits unrelated to issues were simply pushed to our development branch in the project.

Feature branching for individual issues/product backlog items was also employed. Each feature branched out from the development branch and then merged back in through the use of pull requests as it allowed all group members to review the new code when it was finished. These branches were then closed as the pull request was merged.

4.2.2 Unity and Git compatibility

In the case of source code, Unity projects generally have a tendency to not perfectly mesh with version control systems like Git. This is due to the fact that commits generally contain a large amount of binary files for prefabs, assets and other non-source code files, but there are some tricks that can be pulled to improve the situation [10]. It is possible to turn these binary files into YAML text format by tweaking a setting in the Unity Editor which allows the developer to actually properly solve merge conflicts. This in turn introduces another problem where commit messages are bloated with huge changes to these files, making it somewhat harder to have an overview over the actual source code changes. We would like to think that this trade-off still is better than not being able to handle merge conflicts on prefabs and assets.

4.2.3 Code quality and conventions

We primarily used Microsoft's naming conventions [11] and code guidelines [12] while developing Dockit League. An exception to these naming conventions was that we used camelCase rather than PascalCase for member variables. In contrast with Microsoft's C# code guidelines we used the One True Brace code convention [13] for formatting. Doxygen documentation was required for functions and optional, but encouraged, for classes depending on their complexity. The documentation was then later generated using the doxygen tools and added to the thesis as a PDF document. This means that any hyperlinking was lost in the documentation and while doxygen generates LaTeX files, these are unfeasible to directly add to the thesis as they use different style sheets with their own commands and dependencies. This would alter the style of the whole thesis which is unwanted behaviour. The full doxygen documentation can be found in Appendix D.

4.2.4 Game Engine

Using a game engine can help developers work more efficiently on a project due to not having to implement the lower level functionality like physics, rendering and core architectures. Given the three months long development period of the project, using a game engine was necessary to consistently implement the core functionality of the game that was needed. Unity is one of the most widespread game engines used in the industry to this date [1] and while other engines like Unreal also could be seen as an alternative, all of our group members already had worked with Unity before and wished to gain more knowledge on engine functionalities like networking. This means that less time would be spent on learning the basics of the tools and allow for faster progression on the project.

We generally stayed up to date with the latest versions of Unity whenever released and fixed any deprecated functionality as it appeared. The reason for this is that the Unity network functionality is constantly in a state of development and the updates include important bug fixes and optimizations related to networked components.

4.2.5 Integrated Development Environment

Unity offers two IDE's on installation:

MonoDevelop: A cross platform IDE for C# that has been packaged with previous versions of Unity as standard.

Visual Studio 2015: Newer versions of Unity have also begun packaging Visual Studio

as an alternative to MonoDevelop.

In the case of our project we chose to use Visual Studio over MonoDevelop. This is due to the fact that Visual Studio provides a wide array of quality of life functionality like automatic formatting and body generation of doxygen comments and better editor customization, including dark themes that ease development when there is limited light. Visual Studio also has Unity plugins that allow for step by step debugging and full use of the Visual Studio debugger which is a powerful tool for testing purposes.

4.2.6 Communication Tools

Our group members used Discord as primary means of communication. This includes using the provided chat functionality for discussing the project and asking each other for help as well as using the voice chat for daily scrum meetings. One of the advantages of Discord is that it supports printing of code blocks so whenever one of the group members wanted to provide code for questions, this was directly supported with syntax highlighting and formatting.

4.2.7 Miscellaneous Tools

Other miscellaneous tools were also used throughout development. This section will detail these and how they were used. Asset creation was generally handled through a variety of tools like Photoshop, GIMP, Clip Studio Paint and Blender. GIMP in particular worked well as an intermediary tool for centering sprites, which is useful for making sure that the pivot for sprite rotations is properly centered. We used Microsoft Visio as a tool for creating the Gantt chart in the initial project plan, which can be found in Appendix A. Flow charts and similar types of diagrams were created using draw.io and then imported as either images or .svg files directly into the thesis. We also used Toggl as a means of time tracking our individual time usage on the project.

5 Implementation

This chapter is focused around the challenges the group met during development and how these were solved.

5.1 Limited field of view

As mentioned in Section 3.3.3 the game contains a component that creates a view mesh, and uses a shader on this mesh that utilizes the stencil buffer for a per pixel masking. Any other object in our game that should be masked out in the fog of war also needs a different shader that implements the stencil buffer for masking. Having the fog of war was a design decision to limit the player's field of view, and give them imperfect information, which emulates how it would be in any first person game. [14] Since the stencil buffer is a binary masking, the area outside of the view mesh would be completely dark. Even if this also enforces the fps view, it's too restrictive, almost claustrophobic, as it makes traversal around the map difficult. The solution was to use a second camera that only renders the environment with a shader that adds a fog color to every pixel. Unity's replacement shader functionality was utilized to easily replace the shader an object originally had, with the fog shader. This means that the *FogCamera* can render the environment with depth -1. The second camera, the player camera, renders only the elements within the view mesh, using camera depth 1 and a clear flag to only clear the depth. Allowing it to render on top of the already rendered image from the fog camera. The rendered image from the different cameras are shown in Figure 22.

Figure 22: Showing the fog camera render on the left, and the player camera render on the right.

5.2 Responsive user experience

When working on a networked game, providing a responsive user experience for clients is important in the wake of less than optimal network connections [15]. This section will provide a brief overview on how networking in *Dockit League* is handled to take local responsiveness into account.

Dockit League performs any important calculations like collision checks, damage calculations and cooldowns on the server before synchronizing all of the clients. This is

primarily handled through the use of *Command* and *ServerCallback* attributes to make sure certain blocks of code only run on the server before synchronzing through the use of *ClientRpc* functions. *ServerCallback* attributes are particularly useful for when the developer wants collision callbacks like *OnTriggerEnter* or *OnTriggerStay* to only execute on the server. The attribute is only available for *NetworkBehaviour*'s although some workarounds can be made to allow for simiar functionality within standard *MonoBehaviour*'s as mentioned in Section 5.3.

Visual effects on the other hand are handled locally on the clients through the use of synchronized callbacks while client prediction in terms of movement is handled using the interpolation functionality of Unity's *Rigidbody* component.

5.2.1 Consistent force for server and clients

A rather common component to work with in Unity is the *Rigidbody*. These are used for any physics based movement and are generally something the developer would like to keep synchronized across the network. Rigidbodies are synchronized by the *NetworkTransform* component and while one might believe that applying force to the rigidbodies in this case would keep them consistently synchronized across the network, this is actually not the case. In the case of trying to apply force from the player's position towards a point, the server player will experience a stronger amount of force than the clients will. In the case of a peer to peer based game like *Dockit League*, this is unwanted behaviour as the server and clients should experience the same or very similar forces.

We tried a couple of different workarounds like synchronising the code for adding force through the use of *ClientRpc*'s or increasing the synchronization rate of the rigidbodies, but none of these ended up being a usable solution. The issue in general is a result of calculating force vectors on the server based on player positions. This does not necessarily work as by the time the code executes on the clients, their new and updated positions have changed enough to make the force direction different. We ended up trying to use *TargetRpc*'s as a workaround and send the strength of the force, the force mode and the position we wanted to add force towards as parameters. Using this approach allowed the player to locally calculate the force vector and provided far more consistent results than any previous approaches. This also means that the player with authority gets to add the force, as they override the force added on the server.

5.3 Unity's networking limitations

When developing a networked game in Unity, there were a few cases where we wished that the network interface of Unity allowed for certain types of functionality that would make implementation of various features easier. There are a couple of inherent limitations with Unity's current networking capabilities. These include:

- *Command*'s, *ClientRPC*'s and *TargetRPC*'s:
    1. Are unable to return any data. All functions using these attributes need to return *void*.
    2. Are unable to take component references as parameters. This means that for example sending the *Transform* component of a player is impossible.
    3. Cannot be overloaded. A separate function with a different name is necessary if different parameters need to be passed.

4. Does not support generic parameters.

- Calling any *Command*s requires authority. This is something only the player object has, meaning that any child objects of the player are unable to call *Command*s by themselves.
- Standard Unity callbacks will run on all clients by default. This means that any collision callbacks also run on all clients. Unity provides tools for managing this with classes deriving from *NetworkBehaviour*, but not *MonoBehaviour*.

This section will take a look at how some of these limitations can be worked around.

5.3.1 Network spawned objects

The inability to return data from *Commands* meant that we were unable to acquire references to any game object that the players spawned. A workaround for this problem can be implemented through the use of *TargetRPC* functions which make it possible to selectively run code on a single client.

In our case, whenever we needed an ability to acquire references back to their spawned game object, we used a combination of interfaces and *TargetRPC* functions in *Docking* to send a reference to the player who spawned the object. This is handled by letting an ability implement the *ISpawnableReferenceProvider* interface and acquire the reference in the implemented function *SetSpawnedObjectReference(GameObject spawnedObject)*.

Spawning objects is handled through the use of a custom spawning function in *Docking*. This function then looks for the interface and sends a reference of the newly spawned object, as seen in Listing 6.

5.3.2 Network functionality for MonoBehaviours

Deriving from the *NetworkBehaviour* class requires that the game object a script is attached to has the *NetworkIdentity* component. The *NetworkIdentity* component automatically handles disabling and enabling of objects during the life cycle of a hosted game. There are cases where the developer might want to manually control this by using *MonoBehaviour* instead of *NetworkBehaviour*, as discussed in Section 3.5.1.

In our case, we let the *Ability* base class stay as a *MonoBehaviour*. In order to provide network functionality for the abilities, we added a reference to *Docking* in the base class as a proxy for networking. This means that any abilities can go through *Docking* to call *Command* functions. The *Ability* base class also includes a variety of virtual callbacks that in turn are called by *Docking*. This allows abilities to execute synchronized code by overriding the virtual callbacks.

5.3.3 Unity callbacks

Unity offers several different callbacks that helps the developer manage the life cycle of a script or for collision handling. When working with networked code there are times where the developer wants code to only run on the server or the clients, local or remote. This is easy to do for *NetworkBehaviour* derived classes as they, for example, can use the *ServerCallback* attribute to tell Unity that the attached function only should run on the server. This is of course not possible for *MonoBehaviour* classes although the workaround for this is fairly simple.

In the case of wanting to make sure that a collision callback for an ability only gets

```
[Command]
public void CmdSpawnObjectReference(int abilityId,
                                    int prefabId,
                                    Vector3 position,
                                    Vector3 rotation) {

    ISpawnableReferenceProvider ability = dockingKit.abilities[abilityId]
                                          as ISpawnableReferenceProvider;
    if(ability != null) {
        SpawnableObject spawnObject = SpawnableFactory
                                      .Instance
                                      .SpawnObject(ability.GetSpawnablePrefab(prefabId),
                                                   position,
                                                   rotation,
                                                   netId.Value,
                                                   player.GetPlayerTeamId());

        TargetSetSpawnObjectReference(GetConnectionToClient(),
                                      spawnObject.gameObject,
                                      abilityId);
    }
}

[TargetRpc]
public void TargetSetSpawnObjectReference(NetworkConnection connection,
                                          GameObject spawnedObject,
                                          int abilityId) {

    ISpawnableReferenceProvider ability = dockingKit.abilities[abilityId]
                                          as ISpawnableReferenceProvider;
    if(ability != null) {
        ability.SetSpawnedObjectReference(spawnedObject);
    }
}
```

Listing 6: Code snippet for spawning game objects and providing a reference back to the owner

executed on the server, we check for *docking.isServer* at the start of the callback. The *isServer* boolean is part of *NetworkBehaviour*'s and is true if the current code is running on the server. The only issue with this approach is that the callback itself will still get called on all clients, creating some unnecessary overhead even if the body of the function only is executed on the server. We do not see this as a particularly large issue given that collision callbacks won't happen frequently enough for this to be a performance problem.

5.4 Programmatic interpolations

Unity as a game engine already has fairly robust and powerful animation tools that allows the developer to create and manage animations directly in the editor. These animation tools have a lot of versatility, but there are times where the developer might want a bit of extra control and use a programmatic interpolation instead.

In our case we needed to produce a curve for a boomerang to travel through while developing the boomerang kit. We decided to perform this interpolation through the use of a *Cubic Bezier curve* as these curves are flexible, very simple to control and can provide a nice squeezed arc for the boomerang to interpolate through. Another option would be to use a Hermite curve, but we ended up sticking with Bezier curves as it would make the code more readable for all group members due to everyone's preexisting familiarity

with these. The formula for interpolating through cubic Bezier curves is as follows:

$$B(t) = (1-t)^3 P0 + 3(1-t)^2 t * P1 + 3(1-t)t^2 * P2 + t^3 P3$$

The function takes five parameters:

P0: The start position of the curve

P1: The handle of the start position

P2: The handle of the end position

P3: The end position of the curve

t: The input time of the interpolation. Has to be in range [0, 1]

We use the Bezier curve in two ways:

1. To create vertices for a *LineRenderer* component that displays the approximate travel path of the boomerang while holding down the ability button
2. To interpolate the boomerang itself as the ability button is released

When we say approximate path we mean the path that the boomerang would travel if the player stood perfectly still. The point of displaying an approximate path is to give the player an idea of how far the boomerang will travel before returning. It is generated using local coordinates rather than world coordinates. On the other hand, actually throwing the boomerang stores the world position of both handles while using current position of the player as start and end point. This means that the Bezier curve will dynamically change based on how the player is moving, but still travel to the peak of the approximate path.

While the Bezier curve is nice for providing a curve to interpolate through, the animation itself looked fairly uninteresting as the change in ***t*** was linear due to the fact that we simply used a timer variable for it. The boomerang kit's abilities are built around the position of the boomerang so we needed to provide ample time for the player to use the kit's abilities as the boomerang approached the peak of its curve.

5.4.1 Handling the velocity of the animation

While developing, we drew several graphs that could represent a non linear change in our input parameter ***t***, but were unsure of how we could get representations of these into our code. This section includes two different ways of providing graphs that can be evaluated by the game to control the velocity of the animation.

The first solution we found by looking at Unity's documentation was to use Unity's built in *AnimationCurve* [16] type. *AnimationCurve*s can be public members of any *MonoBehaviour* class, allowing the developer to use the inspector to generate curves in the range of [0, 1] for both axes by default. These curves also have an evaluation function that allows the developer to provide a input time parameter and get an output back.

In our case, we wanted the boomerang to accelerate from the start, decelerate towards the halfway point of the animation and then accelerate again towards the end. This was fairly simple to implement using the *AnimationCurve* type since it works in the range of [0, 1] by default. All we needed to do was to provide the evaluation function a time variable and use its output as ***t*** into the Bezier curve interpolation.

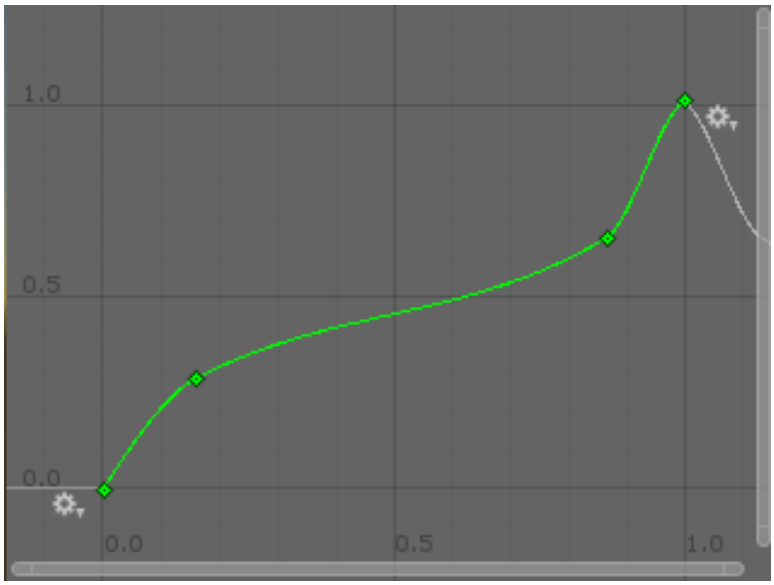


Figure 23: This curve shows the change in t as time goes from 0 to 1 on the x axis using Unity's *AnimationCurve* type.

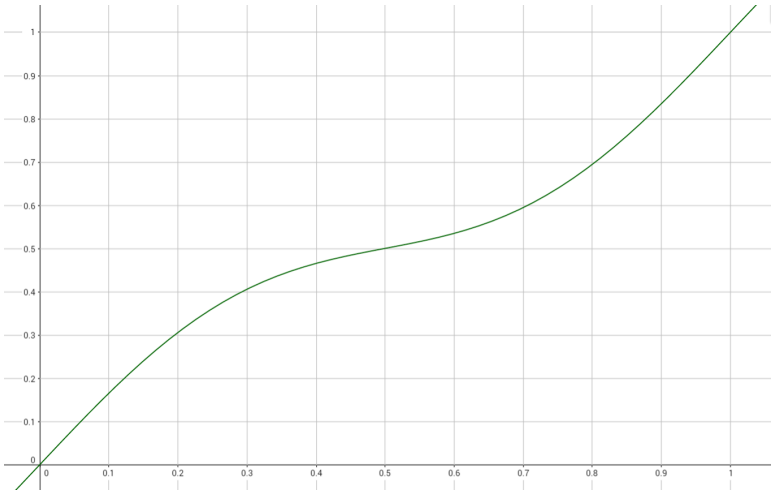


Figure 24: This curve shows the change in t as time goes from 0 to 1 on the x axis using the mathematical curve $f(x) = x + \sin(6.28x)/9$

As seen in Figure 23, the *AnimationCurve* approach provides an interface with control points and control point handles which is fairly easy to use, but there is also another way of solving the problem.

A more generic approach would be to use a mathematical function like $f(x) = x + \sin(x)/c$ to create a similar looking curve, but this approach has some problems that need to be dealt with. First of all, the curve needs to be scaled so that the wanted part of it is within the range of [0, 1] for both axes in order for the output to work with the interpolation function. This could be achieved by playing around with a graph plotting tool like *GeoGebra* or similar. In our case, the function $f(x) = x + \sin(6.28x)/9$ as seen in Figure 24 would provide similar behaviour to Figure 23, but still lack the strong acceleration towards the end of the interpolation. This approach gives less control to the developers who work in the engine and spending time trying to scale the curves can be quite time consuming.

We also made sure to check the performance difference between the two approaches by measuring the execution time of both. We measured the time spent on each function per update and calculated the mean of the time values after 10 boomerang throws. This gave us the following results:

- The average execution time of the *AnimationCurve*'s evaluation function took $\sim 0.36\mu s$
- The average execution time of the math function took $\sim 0.20\mu s$.

The time difference was calculated using Unity's *Time.realTimeSinceStartup* variable. The *Time.realTimeSinceStartup* float is measured in seconds so we multiplied the average time difference by 10^6 and used a output precision of two decimals for these results. The mathematical approach provides a small increase in performance as seen from the results, but in our case the difference is too small to warrant using it. The *AnimationCurve* approach is far easier to work with directly in the editor instead of using external graph tools to modify the curve to our needs. On the contrary, the mathematical approach is more generic and might see use in non Unity applications.

5.5 Interpolation using coroutines

Interpolations in Unity are generally handled in each script's *Update()* function through the use of timer variables and adding Time.deltaTime to these per update. In some cases this adds unnecessary logic to the update loop of a component and the developer might wish to further decouple the interpolation from the loop itself to improve code readability. This can be handled using C# coroutines as these functions are capable of stopping execution during a frame and then resuming the next frame to provide similar functionality to that of the standard Unity update loop callback.

An example of how this is implemented can be seen in Listing 7 which contains a snippet from the *FieldOfView* component and includes a coroutine for interpolating the view radius of the component.

This function will stop at the end of each while loop execution and resume on the next frame using *yield return null;*. This allows the interpolation to move forwards each frame although the example function in particular will not provide fully linear interpolation. This is due to fluctuations in deltaTime between frames and the observed behaviour of an interpolation using this function will appear as a smooth interpolation that slows

```
public float viewRadius;

private IEnumerator ViewRadiusLerp(float targetRadius, float speed) {
    while (Mathf.Abs(targetRadius - viewRadius) > 0.1f) {
        viewRadius = Mathf.Lerp(viewRadius, targetRadius, speed * Time.deltaTime);
        yield return null;
    }

    viewRadius = targetRadius;
}
```

Listing 7: A coroutine used for interpolation of the field of view radius

down towards its end. A different way of implementing the interpolation, this time in an actually linear fashion would be to use a local timer variable that moves in the range of [0, 1] by adding Time.deltaTime each frame. A modified version of the radius interpolation using this thought process can be seen in Listing 8.

```
public float viewRadius;

private IEnumerator ViewRadiusLerp(float targetRadius, float speed) {
    float lerpTimer = 0;
    while (lerpTimer <= 1f) {
        lerpTimer += Time.deltaTime * speed;
        viewRadius = Mathf.Lerp(viewRadius, targetRadius, lerpTimer);
        yield return null;
    }

    viewRadius = targetRadius;
}
```

Listing 8: A modified coroutine used for linear interpolation of the field of view radius

A benefit of decoupling interpolations from the update loop is that there is no need for an additional conditional variable that checks whether an interpolation is active per frame. When reading the code, the interpolation approach also makes it more clear when the interpolation starts using the *StartCoroutine()* function. An apparent thought when working with coroutines and interpolation is the possibility of providing a generic solution for all basic interpolations by using of a static utility class. This will not work as Unity requires that all coroutines need to exist in classes deriving from *MonoBehaviour* which is incapable of being static. This can be worked around by having a singleton *MonoBehaviour* class and call functions through its static instance.

5.6 Initial game balancing

The full game functionality required for proper user testing ended up being implemented rather late into the project's development. Due to this we had limited time for user testing and needed to provide the testers with a build that already had some initial balancing done. *Dockit League* is an asymmetrical game due to the variety of available docking kits so we used a mathematical model [17] for the initial balance iteration. Doing so allowed us to have a overview over the different kits including their strengths and weaknesses.

Balance Table 1 contains columns for kit name, kit health, kit movement speed, damage and utility. The damage column has values assigned based on the potential damage output of the kit while the utility column has values assigned based on the overall the

Table 1: Initial balance table

Docking Kit	Health	Movement Speed	Damage	Utility	Totals
Boomerang Kit	Low (1)	High (3)	High (3)	Medium (2)	9
Brawler Kit	High (3)	Low (1)	Medium (2)	Medium (2)	8
Bomber Kit	Medium (2)	Low (1)	High (3)	Medium (2)	8
Marksman Kit	Medium (2)	Medium (2)	Medium (2)	Medium (2)	8
Sniper Kit	Medium (2)	Medium (2)	High (3)	Medium (2)	9
Tank Kit	High (3)	Low (1)	Low (1)	High (3)	8
Trapper Kit	Low (1)	Medium (1)	Medium (2)	High (3)	8
Support Kit	Medium (2)	Medium (2)	Low (1)	High (3)	8

utility the kit provides through its abilities. This includes abilities that apply modifiers and other effects without necessarily focusing on damage. Further tweaking on cooldowns, damage values and modifier durations is the focus of the user testing rather than the initial balancing as this Section is supposed to give a rough estimate of each kit's capabilities. Additional information on the various docking kits and their abilities can be found in Section 3.6

5.6.1 Observations from the initial balance table

As seen in Table 1, the sniper and boomerang kits end up with a slightly higher total than the other kits. This is a deliberate decision as both kits have a high damage potential if played well, while having a low to medium damage potential otherwise. We believe that both kits are fairly challenging to play in order to achieve their full damage potential, so the difficulty offsets the fact that the two kits are a bit stronger than the rest. The boomerang kit requires the player to hit with multiple boomerangs while they move forwards and then again once they move back for the maximum amount of damage. The sniper kit requires time and preparation before shooting and rewards a large amount of damage if the projectile actually connects. These aspects of the two kits allows us to provide a high risk, high reward ratio for playing well.

An interesting thing to note with our current design is that none of the implemented kits have a low utility value. All the kits feature at least two abilities that provide utility through either the use of modifiers or other effects. The utility abilities for the kits are generally designed to help patch up the weaknesses of each kit. The brawler kit, for example, struggles fighting ranged enemies due to its low speed and melee weapon, but contains abilities for reflecting projectiles and stunning opponents to get closer. If any future development were to take place on the project, creating kits with less utility and a larger focus on the other statistics could improve the variety of of the kits.

One important aspect of game balance is the fact that distinct weaknesses is a great way to provide counterplay and avoiding dominant strategies [18]. While the kits generally have abilities that help them somewhat with their weaknesses, these abilities have long enough cooldowns to not always be available. Balancing cooldowns isn't the only way to expose the weaknesses of kits though. The boomerang kit is the only kit with low health and has a large damage potential with high movement speed. One of the weaknesses of the boomerang kit is the low health, and being caught unaware by enemy players will lead to a swift death. The limited field of view also helps creating situations where the player might have been inattentive and not seen an enemy sneaking in

from behind. The high speed of the kit might be seen as beneficial, but can result in the player carelessly rushing into enemies who are right around the corner of walls and other obstacles.

Another thing to note about the balance table, is that it does not take shop price into account. This is due to the fact that we were unsure of how strong each kit was before performing playtesting, so we kept the same price for all docking kits. Starting to think of price as part of the initial balance table would allow for certain kits to be stronger than others by justifying their strength with a higher price. Given a larger pool of docking kits fulfilling similar roles than currently implemented in the game, the price could end up being used to a larger extent to create more variety for kits with similar use cases.

5.7 Controller based menu navigation in Unity

Dockit League originally had a different main menu and lobby handling script that was more integrated for controllers, but we ended up having to cut it from the last iteration of the game as there was limited time to integrate it with the new lobby and main menu that supported different game modes. We would like to spend some time detailing the old implementation in this section as it might provide additional insight into how controller based menu navigation can be handled in Unity.

There are two primary things that should be handled differently to accommodate for controller navigation compared to just navigating with the mouse:

1. The controller needs an entry point for navigation compared to the mouse. These usually consist of clickable buttons and can be navigated through using the dpad or left analog stick of the controller.
2. While having a back button on menu screens is common for mouse and touch navigation, controller users might find it more intuitive to be able and navigate backwards using a physical button. One needs only look back at the era of the *Super Nintendo Entertainment System* and its games to see that developers already were using a physical button for backwards menu navigation.

Backwards menu navigation can be handled in multiple ways. The current implementation in *Dockit League* allows for arbitrary menu navigation and works well with mice, while the old implementation used a stack instead, inspired by how *Android* handles backwards navigation using its fragment back stack [19].

The stack based approach allows for simple controls when using the *OnClick* interface to handle individual button presses. The developer simply has to drag the next menu object as a parameter to the menu handling script and the script then takes care of putting the old menu panel into the stack. Once the new menu panel had been displayed the script would quickly look for buttons and set the first found button as the selected one, providing an automated entry point for controller navigation.

This old implementation also supported adding properties to the top of the stack, allowing custom behaviour on backwards navigation. We handled this by storing enums with the menu stack objects and perform custom behaviour depending on the enum of the popped menu. This was particularly useful for telling Unity to stop hosting or matchmaking whenever the player left certain types of menus. The stack based approach also made it easy to use physical buttons for backwards navigation as all the script needed

to do was to wait for the correct button to be pressed and then trigger a backwards navigation by popping the stack.

6 Deployment

6.1 Automated Unity Builds

While Unity and Git don't necessarily work too well together out of the box as mentioned in Section 4.2.2 one might start to wonder whether automatic build processing is possible. Having a central hub with automatically built binaries can be quite useful for testing the game and providing the correct binaries for testers. Unity offers a free service called *Unity Cloud Build* which provides this functionality.

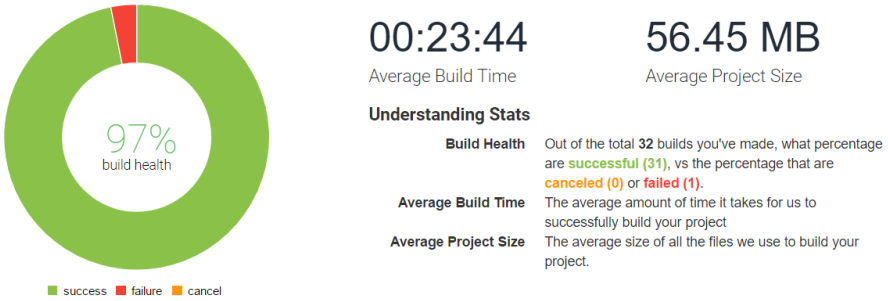


Figure 25: Unity Cloud Build's project statistics illustrate the status of the build process.

Unity Cloud Build allows the developer to set up automatic builds for different target platforms and integrates with BitBucket using SSH keys to get access to the project repository. Specific branches to be built can be specified directly within the service and provides shareable links which can be given to others for both deployment and testing purposes. The service also offers build statistics as seen in Figure 25 and allows for automatic execution of specific unit tests.

We have integrated our BitBucket repository with Unity Cloud Build to make sure that we always have an up to date build available for Windows, MacOS and Linux.

6.1.1 Dockit League binaries

This section contains the binaries for the game which are built at the time of thesis hand in through the use of Unity Cloud Build's shared links. In order to download the latest build of *Dockit League* we recommend visiting the BitBucket repository [20] and download the binaries from the links on the front page. There is no necessary installation needed when downloading the game as the downloaded folder includes all the files for the game.

These are the links to the different builds:

- Windows x86: [21]
- Windows x64: [22]

- macOS: [23]
- Linux: [24]

7 Testing and User Feedback

7.1 Internal testing

One of the important parts of working with a networked game like *Dockit League* is making sure that all synchronized behaviour acts correctly through testing. We would individually test our docking kits as we worked on them and their abilities. In order to properly check that behaviours were properly synchronized, we generated builds of the game that we used in conjunction with the editor to play as multiple players on the local network. Doing so allowed us to test kits and abilities from the perspective of both the server and client, as well as debugging each side individually by using the editor to host or join games.

We also spent some of the review part during sprint meetings to perform integration testing between the newly developed kits and components as everyone were present at the same location with easy access to the source code. Doing so made it easier for each group member to test their newly implemented features with the others and make sure that everything worked well together.

7.2 User testing

We performed user testing towards the end of development to acquire some general feedback on particular areas that needed improvement. We had two different playtesting sessions consisting of three playtesters and one of our developers. The testers tried the team and regular deathmatch modes, and were given a survey made with *Google Forms* on completion where they could provide feedback from their playtest experience. This Section will take a look at the questions we asked in the survey, the answers the testers provided and reflect upon the feedback we were given. The full questions and answers, including answer charts and statistics can be found in Appendix C.

We asked the testers the following questions in the survey:

1. How did the controls feel?
2. How would you rate the in-game UI?
3. How would you rate the in-game shop?
4. Which Docking Kit(s) did you try?
5. How easy/hard was it to understand the game mechanics
6. Do you have any additional feedback?

Each of the questions also had an additional field where the testers could provide additional thoughts and input.

7.2.1 Feedback on the feel of controls

The testers gave an average score of 3/5 in relation to the feel of the controls. Testers particularly familiar with twin stick control schemes felt that the controls were fluid, while testers less familiar with the scheme felt that some additional help text or a section

that displays all of the controls would have been very beneficial.

7.2.2 Feedback on the in-game UI

The in-game UI received an average score of 2.67/5 and was one of the major sticking points that testers wanted improvement on. The testers felt that the UI in general was pretty poor due to the large amount of placeholder text and icons for a variety of docking kits, making it hard to understand what each ability button does. On the contrary, the docking kits without any placeholder text or icons were praised for having icons that communicated how the abilities worked without having to use them.

One of the testers brought up the fact that there is a inconsistency in relation to which button the primary damage ability of each kit is mapped to. The tester suggested that these all should be mapped to the same button regardless of kit.

Another tester noted that it was somewhat confusing that the left/right shoulder buttons and the left/right triggers were paired with each other on the UI. The tester suggested that pairing the left button with the left trigger, and similarly for the right button and trigger, would make the UI more intuitive, given that the pairing is more logical in relation to the physical placement of the buttons.

The input from the testers in general from this question suggested a lack of visual feedback in the UI which is something that certainly could be improved upon. This was one of the questions that we expected most of the testers to give a low score since we were aware of the lack of visual polish.

7.2.3 Feedback on the in-game shop

The in-game shop received an average score of 3.67/5 from the testers. The general feedback of the testers was that the shop was easy to navigate although given some of the answers it seems like a few testers were unaware of the fact that the shop provided descriptions for each docking kit and its abilities. This could possibly be more clearly stated by using the same button icons from the in-game UI to communicate which buttons were used to navigate the description tabs. A fair few number of docking kits also had placeholder text and icons for their shop descriptions, making it hard to understand what each kit was capable of.

Some testers also wished for there to be somewhat of an introduction to how the shop worked so they could clearly understand how to use it. One of the testers were particularly confused with the highlighting of the verification prompt options as the highlighting had a darker color than the original and the tester was used to having a brighter color used for highlighting the "yes" option.

7.2.4 Docking Kit feedback

Among the docking kits that the testers tried, the brawler kit ended up being the most tried one while the support kit was the least tried one.

The testers seemed to like the different abilities of each kit and felt that they were intuitive and interesting to use. Still, as mentioned by most testers earlier they would really have liked to have less placeholder icons to properly understand what each ability did and which button it was mapped to.

In the case of the kits that the testers liked best, the boomerang and brawler kits

were the most favoured ones while the tank and marksman kits came in second place. A thing to note with these results is the fact that both the boomerang and brawler kits were without any placeholder icons.

7.2.5 Feedback on understanding the game mechanics

The testers provided an average score of 3/5 in relation to how easy it was to understand the game mechanics. The primary point of improvement that the testers wanted was better visual feedback on things like player death, player damage taken and some basic information on the start of the game that explains the game mode.

7.2.6 Additional feedback from the playtesters

The testers who provided additional feedback wrote that they found the overall game experience to be fun, albeit somewhat confusing due to their earlier mentions of placeholder icons on the UI as well as lack of visual feedback.

7.2.7 Reflection on the feedback of the playtesters

In general, the user testing sessions provided us with a good amount of valuable feedback. The primary concern of testers being the lack of visual polish is something we are aware of and want to improve. In our case we had deprioritized the visual polish of the game as it was less important than making sure that the core components of the game worked properly for the thesis. We will not be doing any additional visual polishing by the time this thesis is handed in, but we have planned to spend some time polishing the visuals in preparation for the coming thesis presentations. Our current plan for improving the game is as follows:

- Improve visual feedback.
    - More pronounced feedback for players taking damage.
    - More pronounced feedback for when traps and mines are triggered.
    - Players fade out on death rather than simply disappearing.
- Providing more visual information.
    - Displaying the rules of the game mode in short at round start rather than only on the "Create Game" part of the main menu.
    - Provide UI icons that display the controller mappings for the shop and pause menu.
    - Include controller mapping information in the pause menu.
- Fixing inconsistency and improving the intuitiveness.
    - Switch the ability pairings on the UI to pair the buttons with their respective triggers.
    - Highlighting the "yes" option of the purchase verification in the shop with a brighter color, rather than a darker one.
    - Consistent button mapping of abilities with similar functionality across all docking kits.
- Remove all placeholder text and information, replacing them with finalized versions.

8 Discussion

8.1 Development decisions

8.1.1 Using scriptable objects

When we first started working on the modifier architecture for Dockit League, we had to find an easy to use and extendable way of creating new modifiers. Having recently seen a talk from *Unity Europe 2016* [25] about how scriptable objects can be used for these types of use cases, we decided to base the architecture around them. Using scriptable objects instead of *MonoBehaviour* has a wide range of advantages:

- A scriptable object is not reset when exiting play mode compared to a *MonoBehaviour* object. This means that any values tweaked in the inspector while playing stay saved.
- Scriptable objects can be referenced instead of copied when instantiated, helping decrease unwanted redundancy for complex objects.
- A scriptable object can be referenced from any scene while keeping cross-scene references of *MonoBehaviour* objects can be tricky.
- Scriptable objects provide better version control granularity as one file gives one object without anything additional like transform components.

Scriptable objects also have some disadvantages:

- Scriptable objects have very few callbacks compared to a *MonoBehaviour* object. *OnEnable()*, *OnDisable()* and *OnDestroy()* are the only ones. This can be worked around by having a proxy *MonoBehaviour* that calls functions within the scriptable object.
- Scriptable objects contain shared data. This means that any object references during runtime only should be acquired and used in method scope.

In the case of our project. Using scriptable objects for modifiers and shop items gives us the ability to create and quickly edit data directly in the editor as well as being able to attach some additional code if necessary. In the case of modifiers, using scriptable objects makes it very easy to define custom behaviour through the use of the multiple callbacks that exist within the base *Modifier* class. These can be overridden to perform code for the server only, on all clients and locally which gives us the amount of control we want when working on a networked multiplayer game. In the case of shop items, using scriptable objects provides a reusable interface that allows us to quickly add items without having to worry about editing other dependent components as the shop itself handles instantiation and placement of these.

A more technical look at how scriptable objects are used for modifiers and shop items can be found in Section 3.4 and Section 3.7.2.

8.1.2 Moving from Confluence to ShareLaTeX for writing the thesis

As seen in Appendix A we had originally planned to use Confluence as the thesis container as that would mean having a centralized hub for documentation and the thesis. We eventually started moving away from Confluence and ended up only using it to document any Scrum related meetings and started using ShareLaTeX instead. The reason for this is that it allows the thesis to look more professional due to the template style sheet as well as providing a lot of utility tools that make writing the thesis easier. Some of these include easy access to PDF compilation, BibTeX for formatting and automating references as well as being able to work on the thesis in a parallel manner similar to Google Docs. Another benefit of using ShareLaTeX is that less time is spent worrying about text formatting as this is mostly handled by the stylesheet from the thesis template.

8.1.3 Decreasing the amount virtual functions using interfaces

The *Ability* base class in an important intermediary component for making the docking kits work with networked code. The script itself is not a *NetworkBehaviour* so it instead employs virtual functions that get called by networked components to provide synchronized behaviour. One of the issues we experienced throughout development was the fact that this class would get exceedingly bloated with virtual functions as more features were required. The worst offender were the functions that allowed abilities to have server callbacks, providing similar functionality to the *ClientRpc* attributes of *NetworkBehaviour*'s. The issue here was that whenever we wanted to pass parameters we had to first create a new command in the *Docking* script due to not being able to overload commands or provide generic parameters. We then had to create a corresponding virtual function taking the parameters in the *Ability* class.

We ended up using interfaces to somewhat alleviate the code bloat. The difference with using interfaces is that we no longer need to create the additional virtual functions in *Ability* for each new type of parameter. Instead, individual abilities can implement the *IServerCallback* interface which supports generic parameters. Using this approach decreases the code bloat a little bit, but we still need to create new commands in *Docking* due to the inherent limitations of commands. In any case it helps reduce a little bit of code bloat so we found it a worthwhile solution.

8.1.4 Providing ergonomic controls when using a twin stick scheme

During the early design phase one of the things we had to decide on was how many abilities each docking kit would have. We wanted each kit to have enough depth so that players would have to spend some time properly learning each kit and improving their play through experience, but we also had to take into account the control scheme that we were targeting. The abilities of each kit should be easy to access when using a controller while moving about and aiming at the same time. This leaves us with a rather limited selections of buttons to use.

We have the shoulder buttons, triggers and stick buttons available. We decided that each kit would have four abilities each as the triggers and shoulder buttons are the most ergonomic to use with the dual stick scheme. Using the stick buttons for additional abilities would also have been possible, but we could imagine some imprecise aiming if for example an ability was bound to the right stick button. Less frequently used functionality like opening the shop and docking/undocking could then mapped to the primary ABXY,

Start and Select buttons.

8.1.5 Updating game engine versions during development

Whenever developing games in a constantly updated engine like Unity or Unreal it is inevitable that new versions are released. These new versions come in different shapes and forms. Minor releases mostly contain bug fixes and small changes while major releases introduce new functionality and might end up deprecating old features. In cases where changes or new functionality might be beneficial for the project, one has to see whether spending the time and resources on the upgrade is worth it.

While testing the game during sprint reviews, one of the issues we ended up experiencing at times were random disconnects with seemingly limited error messaging given for debugging. Researching into the issue a bit on the Unity forums suggested it could be related to the 4KB/s bandwidth limit of Unity. At the same time the issue could simply be related to a inconsistent wireless network at the location we were testing, but we started to think a bit about bandwidth optimization anyways.

One of the most apparent optimization's we could perform was to move from Unity3D back to Unity2D with the release of Unity 5.6. We originally started working with Unity3D due to the fact that it provided *NavMeshes* [26] which would be beneficial in the case that we had the time to work on player bots. Unity 5.6 introduced *NavMeshes* for the (x, y) plane making it possible for us to port the game back to 2D. Doing so would substantially decrease network bandwidth usage since there were no dependencies on 3D components.

One of the pieces of data we synchronise often are *Vector3*'s consisting of three floating point values. Synchronized *Rigidbody* components in particular consist of many physics related vectors [27]. Moving to Unity2D would cut the bandwidth usage for any Rigidbody and vector synchronization by $\frac{1}{3}$ which is a fairly significant improvement in network performance. Another side effect of moving over to 2D would be cheaper ray casting. The *FieldOfView* component uses a lot of ray casts to create its view mesh so the performance of this script in particular would improve. 2D Raycasts also give some additional quality of life functionality like providing sorted arrays in order of distance from the origin point when raycasting for multiple colliders.

Another useful piece of functionality that only works with Unity2D is the *PolygonCollider2D* component which allows Unity to dynamically create colliders out of sprites. Unity offers similar functionality for 3D using the *MeshCollider* component, but creating sprites is generally far quicker and easier than using 3D models for simple shapes like cones and hollow circles.

We ultimately decided against transferring over to Unity2D due to the fact that the release of Unity 5.6 happened very late into the development cycle. We thought that moving over to Unity2D would take too much time and given that we had no conclusive evidence of hitting the bandwidth limit we would rather spend the resources on writing and improving the thesis instead.

8.1.6 Player field of view versus raycasts for visibility checking

One idea that we thought of while developing was to use the field of view component for visibility checking rather than regular ray casting. This would allow abilities like the flash grenade in the brawler kit to only stun players who had the grenade in their field

of view at the time of explosion rather than using a raycast + dot product. We ultimately decided against implementing this due to a few reasons.

The first reason is that the field of view is fairly computationally expensive. The component uses many raycasts to dynamically generate a mesh that we use as a visibility mask for the players. It would not be possible to directly synchronise the generated meshes so we would have to synchronise the various variables for the component itself and reconstruct the field of view on the server. Doing so would allow us to have a player local field of view for the sake of responsiveness while using the server's version for any visibility checking. This could work if the server was ran as a dedicated server, but our project is primarily focused on trying to create a MOBA using peer to peer connectivity so a implementation like this would not be very effective as it would place a lot of extra load on the host player.

A server authoritative visibility check like this would be far better for security reasons compared to local checking. We currently have a middle ground where the server uses raycasts and dot products to check player face directions. This is independent of the field of view component so we have less control in regards to limiting the view angles as the server would need to acquire the view angle of the clients to check consistently with their current field of view.

Another issue is that keeping the field of view synchronized would take a fair share of additional bandwidth if we want the server to keep itself updated frequently. Given the fact that players move and are able to rotate quickly and arbitrarily, a very high update rate or interpolation would have to be employed for the server to stay consistent with the local player. Since players can rotate arbitrarily and quickly, using interpolation for the field of view would be somewhat of a challenge as determining whether the player rotated clockwise or counter clockwise to the current rotation would be hard without synchronising additional data. This would require a fairly massive rework of the current implementation.

8.1.7 Sticking with dual stick controls

Dockit League is primarily developed with a twin stick controller setup in mind. This is mostly due to the fact that other control options were outside of the scope for the project. One might think that providing similar controls through the use of a mouse would be simple, but it brings somewhat of a balancing issue. Making the player face towards the direction of the mouse pointer is generally what we would think of as the simplest and most intuitive implementation of the mouse control scheme. The main problem with this implementation is that aiming becomes easier for players using a mouse. To give an example, two players standing still at different positions want to fire at each other with a projectile. One uses a mouse while the other uses a controller. The player using the mouse can simply hover the mouse cursor over the other player for accurate aim while the player using the controller needs to aim by approximately pointing the right stick in the correct direction.

To provide similar behaviour between the controller and mouse options we would need to make the mouse controls work more similarly to that of a controller stick. The mouse cursor could be hidden and reset to the center of the screen each frame while recording any changes in mouse movement as a direction vector. This vector could then

be used to make the player point in the same direction that the mouse is moving. This would balance the two control schemes to some degree, but we believe that mouse controls like the ones we described might end up feeling too unintuitive for players. Due to this we ended up deciding to stick with the controller dual stick scheme for our project scope.

8.2 Experiences with the HLAPI of Unity

The HLAPI of Unity has been very useful for us when developing *Dockit League*. It provides a good high level API that has somewhat of a learning curve, but is generally easy to use once we had worked more with it. It is not without its flaws however. Certain functionality like host migration is still broken to this date as noted by other developers on the Unity forums [28] and Reddit [29]. Host migration in particular is a very essential feature for a peer to peer based game as it prevents everyone from disconnecting when the host disconnects. There is also somewhat lacking debugging tools available for the networking. A lot of error messages related to disconnects are vague and it is also not possible to directly measure bandwidth usage in the profiler. This would be useful for developers wanting to know how much of the bandwidth limit they used so they could optimize parts of the networking accordingly.

The base documentation for Unity's network components is fairly decent in our opinion, but there are parts where the quality of documentation drops to an unacceptable level. The *LobbyManager* and its lobby related systems are horribly documented with only two example projects available in the Asset Store. There is a distinct lack of overarching documentation for how the lobby components work in these projects. To quote the documentation from one of the sample projects on the Asset Store [30]:

> "The main prefab is in "Prefabs/LobbyManager". This is a canvas with the LobbyManager script on it. It have multiple child that setup the UI & different "screens" of the lobby (i.e. Server List, Player Lsit etc...)
>
> Everything above the "Unity UI Lobby" section in the Manager Inspector is from UnityEngine.Networking.NetworkLobbyManager, so see the doc for it to see an explaination for all of them.
>
> Prematch countdown is the time between all players being ready & the game starting.
>
> The Lobbymanager script have reference to all the different screens for easy access. *if you totally replace one of those screens, set its reference there*"

The documentation in the project is filled with badly written grammar and refers to documentation that barely exists on Unity's web pages.

8.3 Observations from sprint statistics

We will take a brief look at Figure 26 which illustrates the burndown chart of the 4th sprint during development as it shows data that was fairly consistent throughout the other sprints.

The first observation is that most issues generally were finished throughout the second week of the sprint rather than issues regularly finished as the guideline suggests. This is mostly due to the fact that the issues were rather large and not split up into subtasks very often. Doing so would better reflect project progress on Jira, but at the same time this

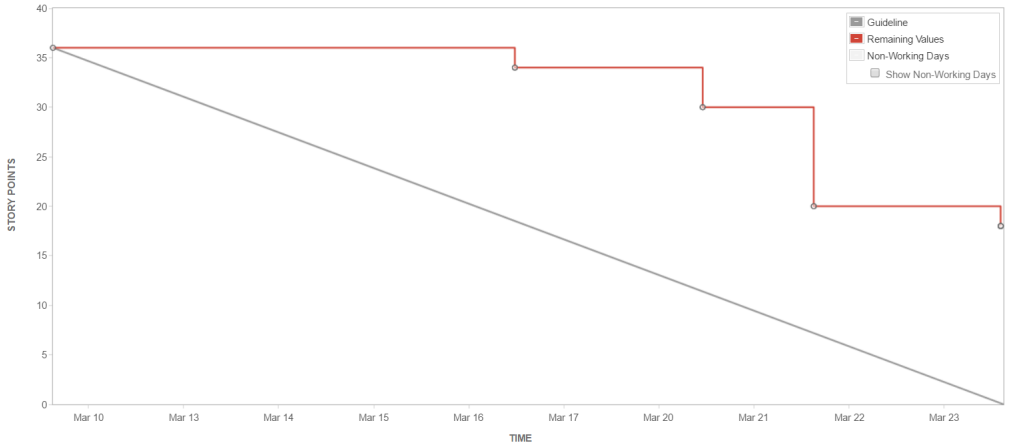


Figure 26: The burndown chart for the 4th sprint shows how sprints generally progressed on average throughout the project.

would incur a fair amount of additional overhead per sprint by having to set up subtasks per issue.

The next apparent observation is overscope. Most of the sprints after sprint 1 ended up having some overscope. The overscope was mostly related to being able to finish a docking kit while also working on other functionality of the game at the same time. This could certainly have been improved by looking at the statistics of previous sprints and manage expectations accordingly. The estimations for some of the game functionalities ended up being estimated for lower story point values than they required due to unexpected issues and bugs which resulted in a lack of resources needed to finish some of the docking kits for the sprint.

8.3.1 Looking at the use of Scrum in retrospect

In relation to actually working on docking kits, fortnightly sprints ended up fitting well. Designing and implementing the base functionality of most kits required one week of time while the other week was spent on bug fixing and polish.

Working with Jira allowed us to setup sprints and issues in a tidy manner, but had a lot of additional functionality we never used although we can imagine a lot of it being useful for managing larger teams.

Despite there having been some overscope during development, using Scrum allowed us to continue developing new increments without necessarily cutting or rushing core functionality of the game due to direct deadlines. The benefit of also having a rather relaxed scope for the game was that it meshed well with Scrum and agile development in general.

The daily standup meetings in particular were quite useful to us. They helped each group member to stay updated with the different components of the project and worked well as a daily discussion around core functionality requirements as the development progressed. There was a fair amount of uncertainty around the requirements for the core components at the start of the development process. As more kits were implemented we

started being able to use these daily meetings to discuss the various requirements our components needed to work with the designs of the different docking kits.

9 Conclusion

We are happy with choosing Unity as our engine as we learned a lot of new things in the process of developing the game. We got a better understanding of the HLAPI and its strengths as well as weaknesses. The Unity low level API was also available, but we didn't delve too much into its functionalities. There are currently alternatives to the HLAPI in development [31] which could fix some of the issues we experienced and having already used the HLAPI allows us to better judge and compare these API's in the future.

We ended up with a good and flexible architecture in terms of adding new Docking Kits, Abilities and Modifiers as seen from the amount of kits we ended up making. This is also in part thanks to *Scriptable Objects* which we learned to use during development. The knowledge of how to use *Scriptable Objects* is something that will be very useful for any future development in Unity.

The game might not have gotten as robust networking wise as we might have wanted due to there not being time for much polish, but despite this, we didn't run into any major network related bugs during playtesting so the code and architecture is fairly stable in that regard. We managed to meet our learning goals by implementing a large project with large networked components although there wasn't much time for balancing other than the initial balance iteration. Designing and implementing docking kits with varying and interesting abilities ended up being quite the challenge, but we are very happy with the amount of kits we managed to implement in the end.

We gained experience with how to use tools like Jira by integrating it with BitBucket and managing issues in the backlog through the use of smart commits. We also found Jira to be a very helpful tool for Scrum activities like planning poker and managing sprints. Confluence was less used than originally intended, but we still attained some further knowledge by using it as a tool for meeting notes and design decision documentation.

9.1 Future Work

As we mentioned in Section 7.2.7 we will be spending some time before the presentation to polish the game and visuals. We might also end up taking some of the other feedback from the playtesters and tweak various parts of the game while doing the visual polish for a better game experience. Other than that, we have no future plans for the development of the game after the presentation.

Bibliography

[1] 2017. Unity engine's usage statistics. https://unity3d.com/public-relations. (Visited May 2017).

[2] 2017. Unity manual - networking overview. https://docs.unity3d.com/Manual/UNetOverview.html. (Visited May 2017).

[3] Valve Corporation. 2012. Counter-strike: Global offensive. [PC Digital Download].

[4] Gregory, J. 2014. *Game Engine Architecture, 2nd Edition*. A K Peters/CRC Press, Chapter 15.2.

[5] 2017. Unity manual - creating and using scripts. https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html. (Visited May 2017).

[6] 2017. Unity asset store - tanks!!! reference project. https://www.assetstore.unity3d.com/en/#!/content/80165. (Visited May 2017).

[7] 2017. Github - field-of-view. https://github.com/SebLague/Field-of-View. (Visited May 2017).

[8] 2017. Unity manual - networking: Object spawning. https://docs.unity3d.com/Manual/UNetSpawning.html. (Visited May 2017).

[9] Keith, C. 2010. *Agile game development with Scrum*. Pearson Education.

[10] Pettersen, T. 2012. The complete guide to unity & git. http://www.gamasutra.com/blogs/TimPettersen/20161206/286981/The_complete_guide_to_Unity__Git.php#Unity_Git_Hosting. (Visited May 2017).

[11] Microsoft naming guidelines. https://msdn.microsoft.com/en-us/library/ms229002(v=vs.110).aspx. (Visited May 2017).

[12] Microsoft c# code guidelines. https://msdn.microsoft.com/en-us/library/ff926074.aspx. (Visited May 2017).

[13] Raymond, E. S. Indent styles. http://www.catb.org/jargon/html/I/indent-style.html. (Visited May 2017).

[14] Burgun, K. 2012. *Game Design Theory: A New Philosophy for Understanding Games*. A K Peters/CRC Press, Chapter 2.

[15] Bernier, Y. W. 2001. Latency compensating methods in client/server in-game protocol design and optimization. In *Game Developers Conference*, volume 98033.

[16] 2017. Unity documentation - animationcurves. https://docs.unity3d.com/ScriptReference/AnimationCurve.html. (Visited April 2017).

[17] Schell, J. 2014. *The Art of Game Design: A book of lenses*. CRC Press, Page 174-176.

[18] Burgun, K. 2011. Understanding game balance in video games. http://www.gamasutra.com/view/feature/134768/understanding_balance_in_video_.php. (Visited April 2017).

[19] Android developer tutorial - menu navigation. https://developer.android.com/training/implementing-navigation/temporal.html. (Visited May 2017).

[20] Dockit league - bitbucket repository. https://bitbucket.org/Knitram/dockitleague.

[21] Dockit league - windows x86 build. https://developer.cloud.unity3d.com/share/W1f8FymOHG/.

[22] Dockit league - windows x64 build. https://developer.cloud.unity3d.com/share/byS7cJX_Bz/.

[23] Dockit league - macos build. https://developer.cloud.unity3d.com/share/-yLt5kQurM/.

[24] Dockit league - linux build. https://developer.cloud.unity3d.com/share/ZkwbsymuBG/.

[25] 2016. Unite europe 2016 - overthrowing the monobehaviour tyranny in a glorious scriptableobject revolution. https://www.youtube.com/watch?v=VBA1QCoEAX4.

[26] 2017. Unity documentation - navmeshes. https://docs.unity3d.com/ScriptReference/AI.NavMesh.html. (Visited May 2017).

[27] Unity documentation - rigidbodies. https://docs.unity3d.com/ScriptReference/Rigidbody.html. (Visited May 2017).

[28] Unity forums - feedback thread for the hlapi. https://forum.unity3d.com/threads/official-multiplayer-improvements.390823/page-3.

[29] Reddit - game developer shares issues with unity's hlapi. https://www.reddit.com/r/gamedev/comments/5l26if/6_months_later_this_is_why_were_migrating_from/.

[30] Unity asset store - example lobby project. https://www.assetstore.unity3d.com/en/#!/content/41836.

[31] Main page for photon thunder - a current in development networking alternative to hlapi. https://www.photonengine.com/en/Thunder.

A Initial Project Plan

A.1 Background

We're three game programming students, and naturally we are going to make a game to end our degree as that is what this bachelor is about. We want to use this project to learn valuable skills preparing us for the future. The design of the game allows us to focus on the networking aspects of game development. Figuring out good practices for local responsiveness coupled with consistent networked behaviour and how to implement these is something we would like to learn. This is good knowledge to have in a world where multiplayer, and the ability to stay connected is very important, even in games. The amount of abilities will challenge us when it comes to game design and balance between a large amount of components, and will push our knowledge in the design aspect of game development.

A.2 Technology

We'll use Unity as a game engine. Unity is an engine used by a lot by Norwegian developers, and an engine heavily used worldwide [1]. We already have some experience with Unity, so we want to expand our knowledge. Using a game engine is beneficial because it allows us to focus on making a game, rather than constructing the components we need to create a game.

We will be using Toggl as our primary tool for time tracking. It's easy to use and allows us to track time spent by the team and what we spent time on. Another benefit of using Toggl is that it can be integrated to Jira allowing us to display time data directly in Jira.

We will use a combination of Jira, Confluence and Bitbucket for project management and issue tracking, documenting and source code respectively. These are to be part of the professional programming course, and we already have some experience using these tools. They're all part of Atlassian software development tools, which makes them fairly easy to integrate with each other.

Our primary IDE will be Visual Studio 2015 as it provides us with integrated testing tools with Unity like breakpoints and step by step debugging.

Group communication will be done through Discord as all the group members are familiar with it and uses it regularly.

A.3 Project Goals

The goals of this project is to have a balanced and entertaining game at the end of semester. In addition to being robust in the sense of proper handling of disconnects, packet delays and packet loss. During this project we want to learn and improve our Unity knowledge. We want to learn more about networking in Unity, how we effectively handle client/server verification and how we provide a consistent and responsive user experience locally. We also want to improve our knowledge regarding Artificial Intelligence by implementing player-like bots (possibly self-learning) and a "replay highlight

selection”-AI if there is enough time.

One of the other things we want more experience with is asymmetric balancing on a larger scale. We will have to balance individual abilities as well as entire kits. This has to be balanced towards multiple players using different kits in combination with each other. We will have to make sure that overall docking kit balance is somewhat good to avoid scenarios where everyone just uses the same docking kit because it is superior in each and every case. In the Standard Game Mode docking kits will have different prices, and therefore their strength needs to correspond the price.

Although the visuals are not our priority we’ll use Unity shaders for different visual effects, for instance abilities. The fog of war will be solved using masking shaders. We want to learn how to make use of professional tools such as Jira and Confluence for project management. We want to improve our ability to estimate the time it takes to implement features and work with smart/semantic commits for issue management.

A.4 Scope

Areas of Expertise

This project will include many disciplines from game development and game programming. The main parts of this will be gameplay design and balancing abilities for docking kits, and then network these. We’ll also need to handle user input, and the responsiveness challenge when it comes to users triggering networked abilities. We’ll have a user interface to display each ability and player health. Graphics and animations will be simplistic and somewhat abstract since this isn’t a priority, and we’re no artists.

Scope Limitations

We will be focusing on finishing the standard game mode first and add a few docking kits. The scope is variable due to the nature of the game. It focuses on having a variety of abilities, and we can simply add/drop making more as the project goes on. The same goes for game modes.

Task Description

Dockit League is a top-down multiplayer battle arena game in which players control vehicles that gets different abilities by equipping docking kits. Taking inspiration from key features in other popular games, we hope to make a fun and unique experience.

The Standard Game Mode features two teams that play against each other in multiple short rounds, before swapping sides. The sides are asymmetric, meaning if the round timer runs out, the attacking team loses. There will be control points around the map, the attacking team needs to conquer one of these in order to win the round.

Each player can equip one docking kit. These kits consist of four abilities each. In the Standard Game Mode these kits may be bought during the buy time at round start, and the price will vary depending on kit strength. This means that a player will have to save their currency over multiple rounds in order to get any kit. Most of the abilities have some sort of interaction between each other, both within a docking kit, and with other kits. This allows teams to choose between different strategies, either for executing a strategy themselves, or to counter the enemy’s strategy.

The players will have a field of view and a sight radius, the size of these may vary

depending on their equipped docking kits. Areas outside of this field will be fog of war.

A.5 Project Structure

Roles, Responsibilities

We have divided the group into roles and responsibilities. Martin has been appointed as project lead while Andreas and Sondre are developers. Other responsibilities:

Andreas: Documenting meetings/decisions

Martin: Scrum Master

Group rules and routines

We have outlined some group rules as an attachment to the project agreement. Other than those, some general routines will be to document all design decisions and write short summaries from all meetings in Confluence. All group members are required to show up to meetings. If this is not possible, a notification from the missing group member is needed beforehand.

A.6 Planning, supervision and documentation

Choice of system engineering model

We have decided to use Scrum for this project. Developing games is generally an agile and iterative process, and being flexible is a must. Therefore an agile model like Scrum is ideal for the project. Another reason for Scrum is that we wish to learn using professional tools like Jira and Confluence. Scrum works nicely with both of these. Finally, using Scrum allows us to generate a lot of extra documentation during the project due to the amount of meetings, which is going to be quite useful when writing the thesis itself.

We are planning to have fortnightly sprints with daily standup meetings. Thursdays will be used for "sprint meetings" consisting of the review, retrospective and planning meeting. Each meeting will have a short summary written which will be added to the Confluence pages.

Plan for status meetings and decision points

We do not have any external clients for this project, but we will have regular status meetings with Simon McCallum who is the project supervisor.

A.7 Quality Assurance

Documentation and code standard

We will be using a good amount of Atlassian's available tools for the project. Confluence will be used for documentation and as the thesis container while Jira will be used for management of Scrum and issues. We are going to enforce a common coding convention that all group members are required to use. The project's coding convention is going to be the "One True Brace" style. We will also be using doxygen style comments for functions.

Configuration management

We will be using a Bitbucket repository with git as our primary means of version control and source code storage. The repository will be separated into several branches:

- We will have a master branch that always has a stable build of the project. This branch will generally get updated after sprints are done.
- There will be a development branch where we add features that are in development during sprints.
- Each member also has his own branch where work is done and merged into the development branch once the work is done.

Testing and issue tracking

Playtesting is something that we will do on a daily basis as we implement the various features we are working on. Issue tracking is going to be handled through Jira using smart commits when pushing code to the Bitbucket repository.

A.8 Implementation plan

Gantt chart

ID	Task Name	Duration	Start	Finish
1	**Bachelor**	**92 days**	**Mon 09.01.17**	**Tue 16.05.17**
2	Semester start	0 days	Mon 09.01.17	Mon 09.01.17
3	**Pre-planning**	**16 days**	**Mon 09.01.17**	**Sat 28.01.17**
4	Project planning and d	16 days	Mon 09.01.17	Sat 28.01.17
5	Meet with Simon	0 days	Tue 24.01.17	Tue 24.01.17
6	Project plan delivered	0 days	Sat 28.01.17	Sat 28.01.17
7	**Sprints**	**79 days**	**Thu 26.01.17**	**Tue 16.05.17**
8	Sprint 1	10 days	Thu 26.01.17	Wed 08.02.17
9	Sprint 2	10 days	Thu 09.02.17	Wed 22.02.17
10	Sprint 3	10 days	Thu 23.02.17	Wed 08.03.17
11	Sprint 4	10 days	Thu 09.03.17	Wed 22.03.17
12	Sprint 5	10 days	Thu 23.03.17	Wed 05.04.17
13	Sprint 6	10 days	Thu 06.04.17	Wed 19.04.17
14	Sprint 7	10 days	Thu 20.04.17	Wed 03.05.17
15	Sprint 8	9 days	Thu 04.05.17	Tue 16.05.17

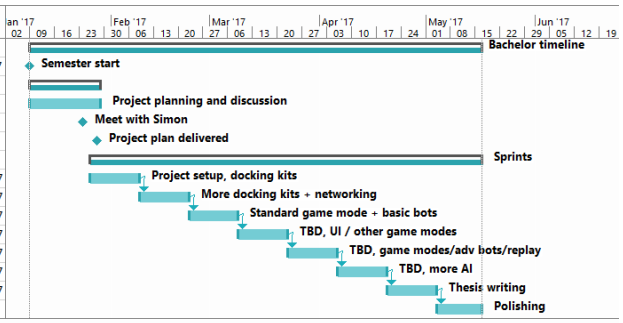


Figure 27: Gantt chart from initial project plan

Milestones and Decisions

We currently only have two major milestones in mind for the project. The first milestone is a working prototype of the game with a finished docking system. The second milestone is to have a working prototype with the standard game mode implemented. Due to the variable scope of the project it's hard to estimate further milestones after this at this time.

All important design decisions and sprint meeting summaries will be documented in Confluence. The thesis itself will also be written directly in Confluence and later extracted to a PDF file.

Time and resouce plan

Time and resource planning will be done with the help of planning poker using story points, which will be done during the fortnightly sprint meetings, in accordance to our Scrum model.

B Meeting Logs

B.1 Temporal record of meetings

11.01.2016 - Bachelor Information Meeting

Discussion of the process and setup of the thesis. Deadlines for submission of documentation. Introduction to the process and the sessions to help with writing the thesis.

24.01.2017

Met with supervisor to discuss the project. Actions:

1. Decide on a writing tool
2. Install development environment
3. Finish project plan

30.01.2017

First Sprint Planning Meeting for the project. Topics included:

Verify product backlog: The focus of the first sprint would be on project start up and docking kit architecture

Primary issues of sprint:

1. Main Menu lobby containing join/leave functionality
2. Docking kit architecture
3. Marksman kit

09.02.2017

Second Sprint meeting consisting of review, retrospective and planning:

- What did we do well?
  - Scoped well for the first sprint.
  - Initial docking kit architecture is very robust
  - Meeting up for daily scrum stand up
  - Good use of Toggl
  - Lobby seems to be fairly robust so far
- What should we have done better?
  - The start was a bit messy especially for Sondre and Martin as they struggled to parallellise the process of creating the first docking kit and architecture.
  - Everyone could get a bit better at using smart commits more often and properly
- Plan for next sprint:
  - Primary focus on docking kits

- Bomber, Tank and Brawler kits
- Health management
- Status effects

23.02.2017

Third Sprint meeting consisting of review, retrospective and planning.

The scope for modifiers/status effects was a bit off so we had to spend a fair share of extra time to get it working properly. The Tank and Bomber kit were also unfinished by the end of the sprint so their completion has been put in the backlog for the next sprint.

- What did we do well?
  - Made a good and generic base architecture for our modifiers
- What should we have done better?
  - Remembering to set Jira issues to "in progress" once work has started.
  - Sondre would have liked to start his work a bit earlier.
  - Andreas got a bit stuck trying to restructure brawler abilities. Should have taken some extra time to properly get into the new modifiers architecture.
  - Modifiers took longer to implement than expected
- Plan for next sprint:
  - Finish the docking kits (Tank and Bomber) that were left over from the previous sprint
  - Trapper Kit.

09.03.2017

Fourth Sprint meeting consisting of review, retrospective and planning.

We got some decent progress during the previous sprint, but there are still some unfinished components for certain kits. We are kind of building the docking and ability architecture as we go. This is mainly because it is really hard for us to predict what we are going to need in the future. Working in a agile manner like this adds some additional overhead to the issues when working in several of the sprints. This sprint in particular required us to do a fair share of architectural work in order to provide abilities with the tools they needed to function as intended.

- Plan for the next sprint:
  - Finish and polish all current kits as well as adding a few new ones to prepare for the implementation of the standard game mode.
  - Sniper and Boomerang kits
  - Revamp of the Marksman kit
  - Getting familiar with ShareLaTeX

10.03.2017

Met with supervisor to discuss progress. Actions:

1. Start thinking about thesis topics.

23.03.2017

Fifth Sprint meeting consisting of review, retrospective and planning.

The new kits developed during this sprint had pretty fun and interesting mechanics so we are fairly happy with their implementation. At this stage in development, the general workflow of creating docking kits has also been fairly solidified so efficiency slowly keeps increasing as we make more kits. We also performed a lot of code cleanup by trading virtual ability functions for interfaces which helps reduce boilerplate and improve code readability.

In hindsight, we probably should have started to use interfaces earlier to make the code more clear and remove redundancy. There was still a bit of underscope this sprint as well for some kits, but not by much this time.

The plan for the next sprint is to focus more on gameplay now that we have a decent amount of docking kits. We want to implement the standard game mode, in-game shop and start making sure that the various docking kits properly synchronise their behaviour based on teams rather than just local/non-local players. Since there are three of us on the group we decided to add another docking kit to the scope of this sprint so that everyone has something to work with as well.

24.03.2017

Met with supervisor to discuss progress. Actions:

1. Transition more and more into thesis writing. Finish the most important functionality of the game that is required to be properly playable and write about the interesting challenges that we have encountered throughout the development of the game.

07.04.2017

Sixth Sprint meeting consisting of review, retrospective and planning.

The shop functionality ended up being finished a bit earlier than expected. This allowed us to spend some extra time on writing the thesis, which can be thought of as a good thing. We should probably have split up the standard game mode rather than thinking of it as a singular large task that would take multiple sprints to finish as it would reflect the progress better on Jira.

The upcoming sprint takes place during Easter holidays so we don't expect too much progress taking place. What we would like to do is to move from Unity3D back to Unity2D as it cuts down a lot of data to synchronise across the network. We will end up saving one float for each Vector3 that we synchronise, which is a considerable amount. We believe that we are already close to hitting Unity's 4KB bandwidth limit so this is something we think might be worth spending some time to do. It might take some time to fix, but ultimately we think it might be worth it. We would also like to take some time to write more in the thesis.

20.04.2017

Seventh Sprint meeting consisting of review, retrospective and planning.

As expected, we did not get that much work done during the easter holidays. We did manage to get some writing done on the thesis, including a good draft for different

topics to write about though. While moving to Unity2D might be good for the network performance of the game, the time spent porting the code might end up being better used on the thesis instead.

The plan for the upcoming sprint is to primarily focus on writing the thesis and finishing up the last few components needed for the game to be complete. User testing will start taking place the moment the standard game mode is finished.

04.05.2017

Eighth and final Sprint meeting consisting of only review given that this was the final sprint.

Game functionality is pretty much finished at this point and we're fully focusing on writing the thesis. We still need to integrate the shop with the game mode rounds, but that should be fairly trivial. We also got a lot of progress on the thesis the last two weeks so that is also good. There are no further sprints, but we will be focusing the rest of our time on finishing up the thesis and performing user testing.

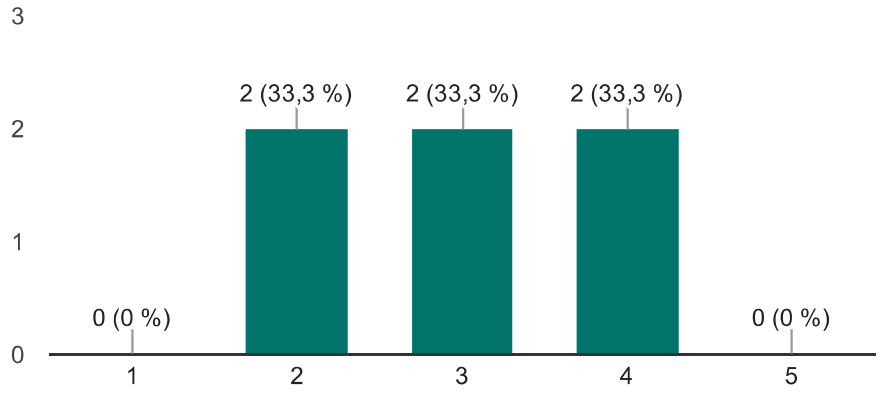
C Playtesting feedback and survey statistics

This appendix includes the Google Form questions and answers for the playtesters and includes statistics for the different answers. One of the longer answers ended up not being printed fully due to limitations with Google Form and printing to PDF.

Dockit League Feedback

6 svar

How did the controls feel? (6 svar)



Any comments on the controls? (6 svar)

Its hard to understand what you actually do

Provide an "option" section to show the controls and anything related to how to change them and so on (doesn't have to be included at all costs though considering the type of game they are made for); camera a bit awkward and slow at times, but working at all times.

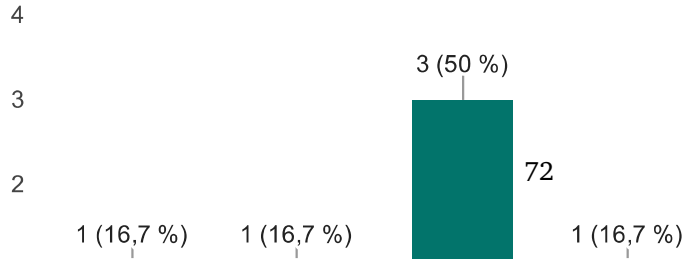
No real description on what the individual buttons worked. I sort of had to base my attempts on previous game experiences.

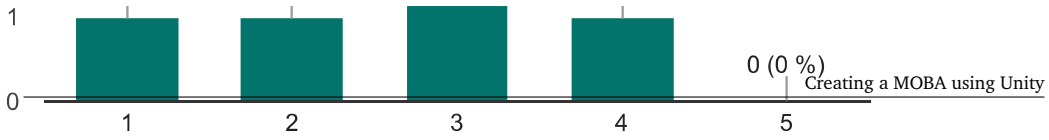
The controls felt somewhat fluid, and familiar to other twin-stick titles.

Brukte tastatur, litt vanskelig å snu kameraet slik man vil.

Good icons goes a long way to communicate abilities, that's good. Would've helped if similar abilities were mapped to the same controls across all the kits. The main attack should be mapped to the same button on melee and marksman etc. Maybe it was and I didn't get it, but then that's kind of a problem too. Also I think four different abilities per kit might actually be a bit too much, I only used like two from each and based my plans on those (unless you count randomly spamming all buttons to see what they do). Maybe focus even more on what is unique for each kit. The stealth thing, for instance, was really cool, but its attack was almost impossible to get good mileage out of. Felt like I could set up an ambush beautifully but not actually execute on it.

How would you rate the in-game UI? (6 svar)





Any comments on the UI? (5 svar)

I felt like I was put in blind

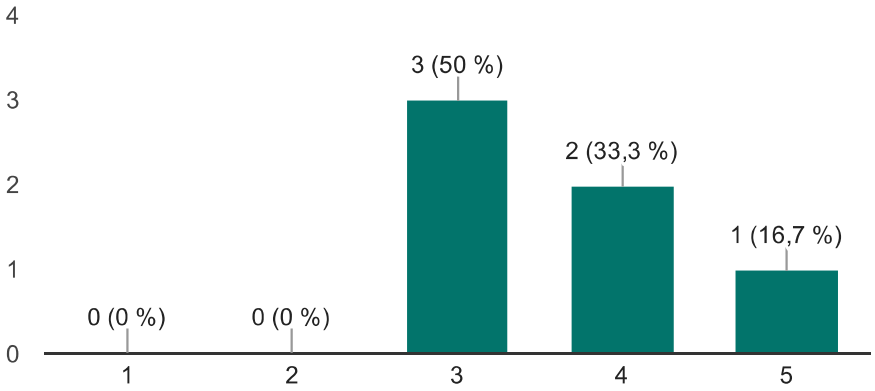
Specify how to access the menu to buy the enhancements, maybe give a name to the icons/moves or a brief pop-up explanation on the move (consider something liker a mouse-over); maybe provide a quick loading screen at the start hinting to the controls of the game; life percentages need sorting out and balancing.

The map and characters was looking nice. There were icons and such, but they were a bit small. Perhaps they would be more informative of their function if they were slightly bigger. there were also some icons missing.

The UI was rather poor, mostly due to missing icons for alot of abilities, and most of them was not very descriptive of what each ability actually did. The abilities were also placed in a weird order in terms of mapped button for the controller. One would expect the right two abilities to be mapped on the right side of the controller, and the left abilities to the left side. This was however mixed, and made the already lacking UI very confusing.

Lacking in the feedback section, but I'm guessing you're well aware of this. Bit hard to see if I actually hit anyone, or whether or not things I had placed around the board were triggered.

How would you rate the in-game shop? (6 svar)



Any comments on the shop? (4 svar)

Was simple enough but could put more work on navigation

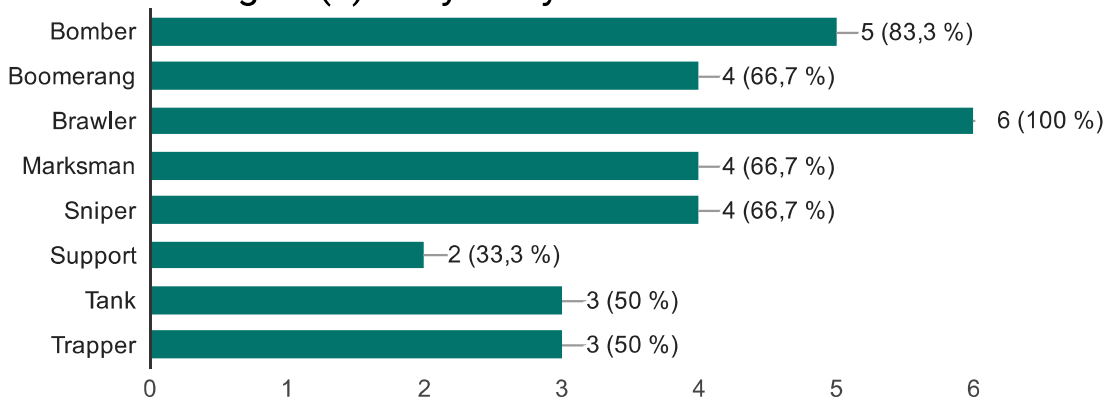
You could think of adding the option of buying more than weapons, like physical/ranged enhancements, speed boosts, etc.

You could change your kit, but there could be a intro to the shop that explained how it works. It was sort of pick a class and move on.

The kits all look a bit alike, which is confusing at first. But again, I guess your focus was not on art. I had a lot of trouble at first with buying because I couldn't see clearly whether I was at 'yes' or 'no' – THAT you might want to communicate more clearly. I think it was because the option I wasn't aiming for was

highlighted when I'm used to it being the other way round! ('Yes' should light up when hovered).

Which Docking Kit(s) did you try? (6 svar)



Any comments to the kits? (6 svar)

Of the abilities that I could understand, most of them fit the role. However, the kit for marksman was not something I would expect. Being called marksman, I probably expected something more along the lines of what boomerang had. Instead of feeling like I played a marksman, it felt more like I was playing a rogue or assassin type.

Support healer fienden og litt for sterkt at den har konstant så mye heal. Marksman burde muligens ha litt cooldown på stealth, men fint at aoe dmg tar den ut av stealth.

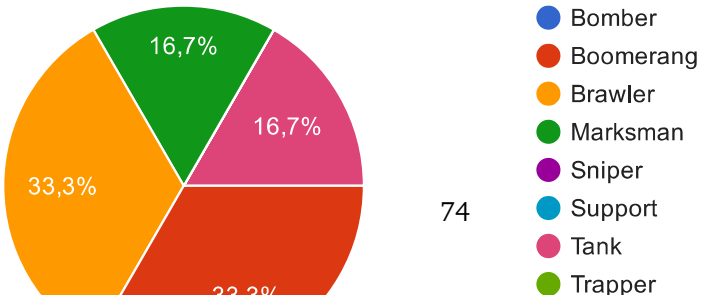
The trapper was fun, but I only understood like half of its abilities. Did I plant any actual traps or was it mostly AOE?

Brawler. One word: HAMMERTIME. Funnyyy, especially with a support to keep you forever healthy (balanced lol). Didn't understand anything about it other than the fact that I could whack things, but I'm alright with that. A bit hard to adjust my position to try land a swing, though, felt like I had to split my focus a lot.

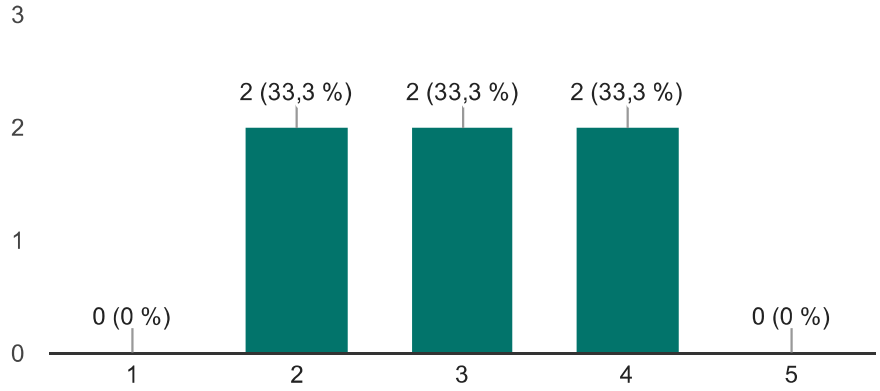
The bomber I didn't understand at all. I thought nobody could see my bombs, thought they were mines. Also I kept thinking I was firing rockets or smth but they never seemed to do anything.

LOVED the marksman, although it felt more like a sneaky ninja to me than some triggerhappy gunner, I wish its main form of attack was a bit less unwieldy. More focus on the sneaking and less on the pew pew might be worth a shot. I want to feel like I'm really stabbing someone in the back >:3

Which Docking Kit did you like the most? (6 svar)



How easy/hard was it to understand the game mechanics? (6 svar)



Comments regarding game mechanics? (6 svar)

It was simple but still confusing as most of the time it felt like your inputs did nothing

Add a more lively menu; remember icons; small loading screen with controls and/or explanation of the game mode selected.

It could use a turtorial, still a very early stage of the game test so didn't expect too much at this point.

The game mechanics were hard to understand simply due to the fact that there were a serious lack of feedback in the game. The game really doesnt tell you all that much, and during my time playing i relied heavily on guesswork based on my general gaming experience.

Burde vært mulig å lese hva de forskjellige abilitiesene gjør. Burde vært litt tydeligere når man dør, f.eks. gjort karakteren mørkere eller noe

The map wasn't ideal for all classes, the ambulance/support is super broken and those placeholders needs to go.

Any additional feedback? (4 svar)

Its a fun game when played with friends but I don't see it having any other sustainability outside of that

As said above. the testing stage was very early, but that's to be expected. an overfall fair game experience given the circumstances.

Det var et morsomt spill når man først kom inn i det, til tross for at det var litt buggs :)

When we began understanding the kits a bit, we had fun. The chaos part might actually be a good thing, although the confusion should preferably come from somewhere other than the kits. I think it could be beneficial to dumb everything down a bit, seeing as most of us just picked the abilities that were easier to

spam/understand and ran with it, ignoring the possibility of more sophisticated plays. I would easy mode the shit out of this if I could just because I didn't feel like I had time to set up the more elaborate plays.

Antall svar per dag

D Doxygen documentation

This appendix includes a full doxygen documentation output from the project source code starting from the next page.

Dockit League

Generated by Doxygen 1.8.13

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

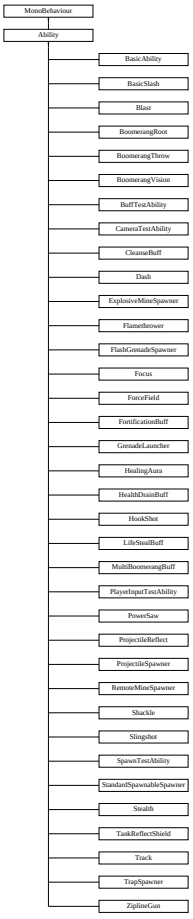
Chapter 3

Class Documentation

3.1 Ability Class Reference

Base class for all abilities.

Inheritance diagram for Ability:



Public Member Functions

- virtual void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- virtual void InitializeLocalPlayer (AbilityUI abilityUI)

  Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.
- virtual void CooldownReady ()

  Called by the cooldown whenever it's ready.
- abstract void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- virtual void ButtonUp ()

  Called when the associated ability button is released.
- virtual void CancelAbility ()

  Call for cancelling abilities. Override in abilities that may be interrupted.
- abstract void SetActive (bool state)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- virtual void SetModifier (bool state)

  Called by the Modifier. Appropriate place for doing local changes.
- void ReduceCooldown (float reductionAmount)

  Reduces the current cooldown for the ability.
- virtual void SetElement (ElementalContainer.ComboableElements element)

Public Attributes

- float **cooldownDuration**
- Sprite **icon**

Protected Member Functions

- virtual void Update ()

  Runs on every client, but only the local player has cooldown initialized.

Protected Attributes

- Docking **docking**
- Animator **animator**
- int **abilityId**
- AbilityCooldown **cooldown**

Properties

- bool AbilityLock `[get, set]`

  Get and Set ability lock. Lock prevents the player from using abilities.

3.1.1 Detailed Description

Base class for all abilities.

3.1.2 Member Function Documentation

3.1.2.1 ButtonDown()

```
abstract void Ability.ButtonDown ( )  [pure virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implemented in BoomerangThrow, Slingshot, BasicSlash, BoomerangRoot, PlayerInputTestAbility, ProjectileReflect, Focus, BoomerangVision, HookShot, CameraTestAbility, LifeStealBuff, Stealth, PowerSaw, MultiBoomerangBuff, HealthDrainBuff, HealingAura, ZiplineGun, TankReflectShield, BuffTestAbility, Flamethrower, ForceField, CleanseBuff, Blast, Dash, Track, FortificationBuff, TrapSpawner, ExplosiveMineSpawner, StandardSpawnableSpawner, FlashGrenadeSpawner, SpawnTestAbility, RemoteMineSpawner, GrenadeLauncher, ProjectileSpawner, BasicAbility, and Shackle.

3.1.2.2 ButtonUp()

```
virtual void Ability.ButtonUp ( )  [virtual]
```

Called when the associated ability button is released.

Reimplemented in BoomerangThrow, Slingshot, and ZiplineGun.

3.1.2.3 CancelAbility()

```
virtual void Ability.CancelAbility ( )  [virtual]
```

Call for cancelling abilities. Override in abilities that may be interrupted.

Reimplemented in Slingshot, PlayerInputTestAbility, CameraTestAbility, and Focus.

3.1.2.4 CooldownReady()

```
virtual void Ability.CooldownReady ( )  [virtual]
```

Called by the cooldown whenever it's ready.

Reimplemented in PowerSaw, and Shackle.

3.1.2.5 Initialize()

```
virtual void Ability.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented in BoomerangThrow, ProjectileReflect, Stealth, HealthDrainBuff, BoomerangVision, LifeSteal↩ Buff, PowerSaw, BasicSlash, HealingAura, Flamethrower, TankReflectShield, BuffTestAbility, Track, CleanseBuff, FortificationBuff, Blast, ForceField, and Dash.

3.1.2.6 InitializeLocalPlayer()

```
virtual void Ability.InitializeLocalPlayer (
            AbilityUI abilityUI )  [virtual]
```

Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.

Reimplemented in Slingshot, Focus, CameraTestAbility, PlayerInputTestAbility, and ZiplineGun.

3.1.2.7 ReduceCooldown()

```
void Ability.ReduceCooldown (
            float reductionAmount )
```

Reduces the current cooldown for the ability.

Parameters

reductionAmount	The amount deducted for the current cooldown.

3.1.2.8 SetActive()

```
abstract void Ability.SetActive (
            bool state )  [pure virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implemented in BoomerangThrow, Slingshot, PlayerInputTestAbility, BasicSlash, CameraTestAbility, BoomerangRoot, PowerSaw, Focus, ProjectileReflect, Stealth, TankReflectShield, ZiplineGun, HookShot, BoomerangVision, Track, HealthDrainBuff, LifeStealBuff, MultiBoomerangBuff, HealingAura, RemoteMineSpawner, BuffTestAbility, Shackle, Flamethrower, TrapSpawner, CleanseBuff, ForceField, Blast, ExplosiveMineSpawner, SpawnTestAbility, StandardSpawnableSpawner, FortificationBuff, FlashGrenadeSpawner, Dash, ProjectileSpawner, GrenadeLauncher, and BasicAbility.

3.1.2.9 SetElement()

```
virtual void Ability.SetElement (
            ElementalContainer.ComboableElements element )  [virtual]
```

Used for local spawning of elemental effect prefabs

Parameters

element	The element we want to set

Reimplemented in BoomerangThrow, and BasicSlash.

3.1.2.10 SetModifier()

```
virtual void Ability.SetModifier (
            bool state )  [virtual]
```

Called by the Modifier. Appropriate place for doing local changes.

Parameters

state	If the modifier should be activated or deactivated.

Reimplemented in BoomerangThrow, BasicSlash, HealthDrainBuff, ProjectileReflect, BoomerangVision, Stealth, LifeStealBuff, MultiBoomerangBuff, BuffTestAbility, and Flamethrower.

3.1.2.11 Update()

```
virtual void Ability.Update ( )  [protected], [virtual]
```

Runs on every client, but only the local player has cooldown initialized.

Reimplemented in HealthDrainBuff, BoomerangThrow, CleanseBuff, FortificationBuff, ProjectileReflect, BoomerangRoot, and BasicSlash.

3.1.3 Property Documentation

3.1.3.1 AbilityLock

```
bool Ability.AbilityLock  [get], [set]
```

Get and Set ability lock. Lock prevents the player from using abilities.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/Ability.cs

3.2 AbilityCooldown Class Reference

Handles the cooldown for abilities. Runs on the local player.

Public Member Functions

- AbilityCooldown (Ability ab, float duration, AbilityUI abUI)
  Constructor.
- void ReduceCooldown (float reductionAmount)
  Reduces the current cooldown for the ability.
- void Update ()
  Update loop. Handles timer and ability ui update.
- void Activate ()
  Called on ability activation. Activates cooldown.
- void ActivateHiddenCooldown (float hiddenCooldown)
  Can be called from abilities whenever they need a hidden cooldown, a simple short cooldown in addition to the standard cooldown for instance.
- bool IsReady ()
  Used for checking if the ability is on cooldown.

3.2.1 Detailed Description

Handles the cooldown for abilities. Runs on the local player.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 AbilityCooldown()

```
AbilityCooldown.AbilityCooldown (
            Ability ab,
            float duration,
            AbilityUI abUI )
```

Constructor.

Parameters

duration	Length of cooldown.

3.2.3 Member Function Documentation

3.2.3.1 Activate()

```
void AbilityCooldown.Activate ( )
```

Called on ability activation. Activates cooldown.

3.2.3.2 ActivateHiddenCooldown()

```
void AbilityCooldown.ActivateHiddenCooldown (
            float hiddenCooldown )
```

Can be called from abilities whenever they need a hidden cooldown, a simple short cooldown in addition to the standard cooldown for instance.

3.2.3.3 IsReady()

```
bool AbilityCooldown.IsReady ( )
```

Used for checking if the ability is on cooldown.

Returns

Whether the ability is on cooldown or not.

3.2.3.4 ReduceCooldown()

```
void AbilityCooldown.ReduceCooldown (
            float reductionAmount )
```

Reduces the current cooldown for the ability.

Parameters

reductionAmount	The amount deducted for the current cooldown.

3.2.3.5 Update()

```
void AbilityCooldown.Update ( )
```

Update loop. Handles timer and ability ui update.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/AbilityCooldown.cs

3.3 AbilityUI Class Reference

Handles the update of the abilitys UI.

Public Member Functions

- void Initialize (PlayerUIHandler uiHandler)

  Initialize the ability UI.
- void Activate ()

  Called on ability activation. Activates cooldown.
- void UpdateCooldown (float newTimeLeft)

  Updates the current cooldown time with the new time.
- void SetAbility (Ability newAbility)

  Changes sprites and cooldown to the new ability.
- void ClearAbility (Sprite emptySlot)

  Stops the update loop and resets the UI to its original empty state.

Public Attributes

- Image **abilityIcon**
- Image **darkMask**

3.3.1 Detailed Description

Handles the update of the abilitys UI.

3.3.2 Member Function Documentation

3.3.2.1 Activate()

```
void AbilityUI.Activate ( )
```

Called on ability activation. Activates cooldown.

3.3.2.2 ClearAbility()

```
void AbilityUI.ClearAbility (
            Sprite emptySlot )
```

Stops the update loop and resets the UI to its original empty state.

Parameters

emptySlot	Sprite used in an empty slot.

3.3.2.3 Initialize()

```
void AbilityUI.Initialize (
            PlayerUIHandler uiHandler )
```

Initialize the ability UI.

Parameters

uiHandler	Reference to associated PlayerUIHandler.

3.3.2.4 SetAbility()

```
void AbilityUI.SetAbility (
            Ability newAbility )
```

Changes sprites and cooldown to the new ability.

Parameters

newAbility	Reference to the new ability.

3.3.2.5 UpdateCooldown()

```
void AbilityUI.UpdateCooldown (
            float newTimeLeft )
```

Updates the current cooldown time with the new time.

Parameters

newTimeLeft	The new current cooldown time.

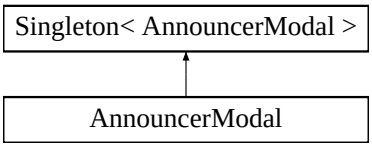
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/AbilityUI.cs

3.4 AnnouncerModal Class Reference

This class controls a generic modal object used for generic status popups in the UI.

Inheritance diagram for AnnouncerModal:



Public Member Functions

- void **Show** (string text)
- void **Hide** ()

Protected Member Functions

- override void Awake ()

  Awake method to associate singleton with instance

Additional Inherited Members

3.4.1 Detailed Description

This class controls a generic modal object used for generic status popups in the UI.

3.4.2 Member Function Documentation

3.4.2.1 Awake()

```
override void AnnouncerModal.Awake ( )  [protected], [virtual]
```

Awake method to associate singleton with instance

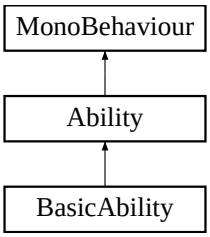
Reimplemented from Singleton< AnnouncerModal >.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/AnnouncerModal.cs

3.5 BasicAbility Class Reference

Inheritance diagram for BasicAbility:



Public Member Functions

- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- float **damage** = 10f
- string **animatorTrigger**

Additional Inherited Members

3.5.1 Member Function Documentation

3.5.1.1 ButtonDown()

```
override void BasicAbility.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.5.1.2 SetActive()

```
override void BasicAbility.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

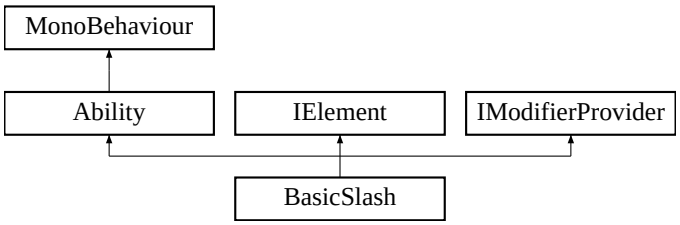
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BasicAbility.cs

3.6 BasicSlash Class Reference

Inheritance diagram for BasicSlash:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Callback for what this ability should do once its associated button has been pressed
- override void SetActive (bool state=false)

  Callback for what this ability is supposed to do depending on given state. State is always false here
- override void SetModifier (bool state=false)

  Called by the Modifier. Appropriate place for doing local changes.
- override void SetElement (ElementalContainer.ComboableElements element)

  Callback for what this ability is supposed to do locally when applying a element

Public Attributes

- float **damageDealt** = 20f
- string **animatorTrigger**
- LifeStealBuff **lifeStealBuff**
- ElementalModifiers **elementalModifiers** = new ElementalModifiers()
- bool **swingActive**

Protected Member Functions

- override void Update ()

  Runs on every client, but only the local player has cooldown initialized.

Additional Inherited Members

3.6.1 Member Function Documentation

3.6.1.1 ButtonDown()

```
override void BasicSlash.ButtonDown ( )  [virtual]
```

Callback for what this ability should do once its associated button has been pressed

Implements Ability.

3.6.1.2 Initialize()

```
override void BasicSlash.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.6.1.3 SetActive()

```
override void BasicSlash.SetActive (
            bool state = false )  [virtual]
```

Callback for what this ability is supposed to do depending on given state. State is always false here

Parameters

state	Whether the ability is to be active or now

Implements Ability.

3.6.1.4 SetElement()

```
override void BasicSlash.SetElement (
            ElementalContainer.ComboableElements element )  [virtual]
```

Callback for what this ability is supposed to do locally when applying a element

Parameters

element	

Reimplemented from Ability.

3.6.1.5 SetModifier()

```
override void BasicSlash.SetModifier (
            bool state = false )  [virtual]
```

Called by the Modifier. Appropriate place for doing local changes.

Parameters

state	If the modifier should be activated or deactivated.

Reimplemented from Ability.

3.6.1.6 Update()

```
override void BasicSlash.Update ( )  [protected], [virtual]
```

Runs on every client, but only the local player has cooldown initialized.

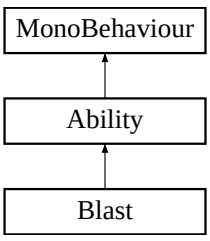
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BrawlerKit/BasicSlash.cs

3.7 Blast Class Reference

Inheritance diagram for Blast:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- void OnTriggerEnter (Collider other)

  Handles blast area, ignore friendly players (still applies to self)

Public Attributes

- float **blastForce**
- string **animatorTrigger**

Additional Inherited Members

3.7.1 Member Function Documentation

3.7.1.1 ButtonDown()

```
override void Blast.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.7.1.2 Initialize()

```
override void Blast.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.7.1.3 OnTriggerEnter()

```
void Blast.OnTriggerEnter (
            Collider other )
```

Handles blast area, ignore friendly players (still applies to self)

Parameters

other	

3.7.1.4 SetActive()

```
override void Blast.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

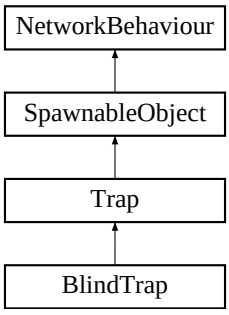
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BomberKit/Blast.cs

3.8 BlindTrap Class Reference

Inheritance diagram for BlindTrap:



Public Member Functions

- override void HandleTrigger (PlayerStatus playerStatus)

  Callback that allows this trap to do whatever it wants whenever it is triggered This one simply applies the member structs containing modifier info

Public Attributes

- ModifierInfo **blindInfo**

Additional Inherited Members

3.8.1 Member Function Documentation

3.8.1.1 HandleTrigger()

```
override void BlindTrap.HandleTrigger (
            PlayerStatus playerStatus )  [virtual]
```

Callback that allows this trap to do whatever it wants whenever it is triggered This one simply applies the member structs containing modifier info

Parameters

playerStatus	The PlayerStatus component of the player that is in the trap

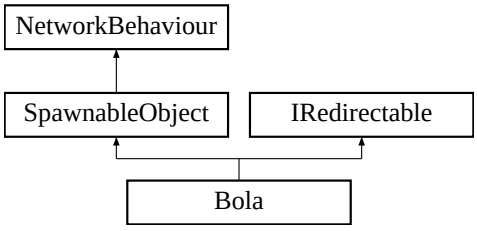
Reimplemented from Trap.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TrapperKit/BlindTrap.cs

3.9 Bola Class Reference

Inheritance diagram for Bola:



Public Attributes

- float **moveSpeed** = 8f
- float **moveSpeedOnHit** = 20f
- float **lifetime** = 10f
- float **hitRadius** = 2f
- float **rotationSpeed** = 500f
- ModifierInfo **slowModifier**
- ModifierInfo **stunModifier**
- Transform **visuals**
- Transform **leftBall**
- Transform **rightBall**
- LineRenderer **lineRenderer**

Additional Inherited Members

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SniperKit/Bola.cs

3.10 BoomerangDataContainer Class Reference

Public Attributes

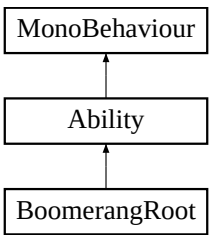
- const int **NUM_CONTROL_POINTS** = 4
- Transform [ ] **bezierControlPoints** = new Transform[NUM_CONTROL_POINTS]
- Vector3 [ ] **storedPositions** = new Vector3[NUM_CONTROL_POINTS]

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BoomerangKit/BoomerangThrow.cs

3.11 BoomerangRoot Class Reference

Inheritance diagram for BoomerangRoot:



Public Member Functions

- override void ButtonDown ()

  Callback for what this ability does locally when its associated button is pressed
- override void SetActive (bool state=false)

  Callback for what this ability is supposed to do locally on all clients when the ability state is changed

Public Attributes

- ModifierInfo **rootInfo**
- SpriteRenderer [ ] **rootIndicators**
- Animator **animationController**
- float **activeDuration** = 0.5f
- string **animationTrigger** = "Root"
- Color **activeColor**
- bool **rootActive** = false

Protected Member Functions

- override void Update ()

  Runs on every client, but only the local player has cooldown initialized.

Additional Inherited Members

3.11.1 Member Function Documentation

3.11.1.1 ButtonDown()

```
override void BoomerangRoot.ButtonDown ( )  [virtual]
```

Callback for what this ability does locally when its associated button is pressed

Implements Ability.

3.11.1.2 SetActive()

```
override void BoomerangRoot.SetActive (
            bool state = false )  [virtual]
```

Callback for what this ability is supposed to do locally on all clients when the ability state is changed

Parameters

state	The new ability state

Implements Ability.

3.11.1.3 Update()

```
override void BoomerangRoot.Update ( )  [protected], [virtual]
```

Runs on every client, but only the local player has cooldown initialized.

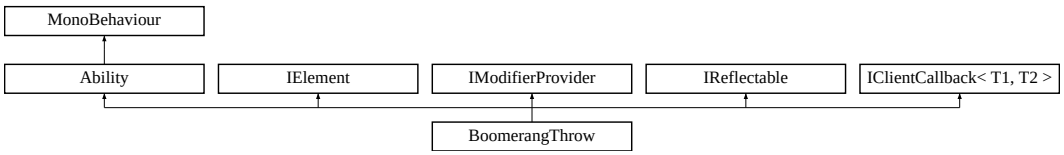
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BoomerangKit/BoomerangRoot.cs

3.12 BoomerangThrow Class Reference

Inheritance diagram for BoomerangThrow:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Callback for what this ability does locally when its associated button is pressed
- override void ButtonUp ()

  Callback for what this ability does locally when its associated button is released
- override void SetActive (bool state=false)

  Callback for what this ability is supposed to do locally on all clients when the ability state is changed
- override void SetModifier (bool state=false)

  Callback for what this ability is supposed to do when a modifier state changes
- override void SetElement (ElementalContainer.ComboableElements element)

  Callback for what this ability is supposed to do locally when applying a element

Public Attributes

- List< LineRenderer > **approximatePathRenderers** = new List<LineRenderer>()
- BoomerangDataContainer [ ] **boomerangData** = new BoomerangDataContainer[NUM_BOOMERANGS]
- List< TrailRenderer > **trailRenderers** = new List<TrailRenderer>()
- GameObject [ ] **boomerangObjs** = new GameObject[NUM_BOOMERANGS]
- BoomerangRoot **boomerangRootScript**
- MultiBoomerangBuff **boomerangBuffScript**
- AnimationCurve **velocityCurve**
- float **damageDealt** = 10f
- float **boomerangSpeed** = 5f
- float **spinMultiplierWhileActive** = 4f
- const int **NUM_BOOMERANGS** = 3
- ElementalModifiers **elementalModifiers** = new ElementalModifiers()

Protected Member Functions

- override void Update ()

  Runs on every client, but only the local player has cooldown initialized.

Additional Inherited Members

3.12.1 Member Function Documentation

3.12.1.1 ButtonDown()

```
override void BoomerangThrow.ButtonDown ( )  [virtual]
```

Callback for what this ability does locally when its associated button is pressed

Implements Ability.

3.12.1.2 ButtonUp()

```
override void BoomerangThrow.ButtonUp ( )  [virtual]
```

Callback for what this ability does locally when its associated button is released

Reimplemented from Ability.

3.12.1.3 Initialize()

```
override void BoomerangThrow.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.12.1.4 SetActive()

```
override void BoomerangThrow.SetActive (
            bool state = false )  [virtual]
```

Callback for what this ability is supposed to do locally on all clients when the ability state is changed

Parameters

state	The new ability state

Implements Ability.

3.12.1.5 SetElement()

```
override void BoomerangThrow.SetElement (
            ElementalContainer.ComboableElements element )  [virtual]
```

Callback for what this ability is supposed to do locally when applying a element

Parameters

element	

Reimplemented from Ability.

3.12.1.6 SetModifier()

```
override void BoomerangThrow.SetModifier (
            bool state = false )  [virtual]
```

Callback for what this ability is supposed to do when a modifier state changes

Parameters

state	The new modifier state

Reimplemented from Ability.

3.12.1.7 Update()

```
override void BoomerangThrow.Update ( )  [protected], [virtual]
```

Runs on every client, but only the local player has cooldown initialized.

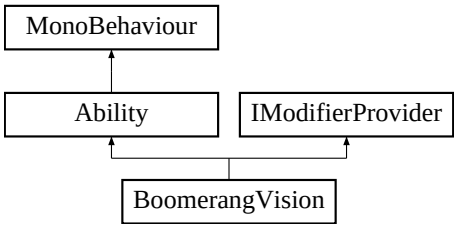
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BoomerangKit/BoomerangThrow.cs

3.13 BoomerangVision Class Reference

Inheritance diagram for BoomerangVision:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Callback for what this ability does locally when its associated button is pressed
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- override void SetModifier (bool state)

  Callback for what this ability is supposed to do when a modifier state changes

Public Attributes

- BoomerangThrow **boomerangThrowScript**
- MultiBoomerangBuff **boomerangBuffScript**
- GameObject **visionIndicator**
- float **visionRadiusWhileActive** = 10f
- float **visionRadiusExtraBoomerangs** = 5f
- float **visionLerpSpeed** = 5f
- ModifierInfo **visionModifier**

Additional Inherited Members

3.13.1 Member Function Documentation

3.13.1.1 ButtonDown()

```
override void BoomerangVision.ButtonDown ( )  [virtual]
```

Callback for what this ability does locally when its associated button is pressed

Implements Ability.

3.13.1.2 Initialize()

```
override void BoomerangVision.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.13.1.3 SetActive()

```
override void BoomerangVision.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

3.13.1.4 SetModifier()

```
override void BoomerangVision.SetModifier (
            bool state ) [virtual]
```

Callback for what this ability is supposed to do when a modifier state changes

Parameters

state	The new modifier state

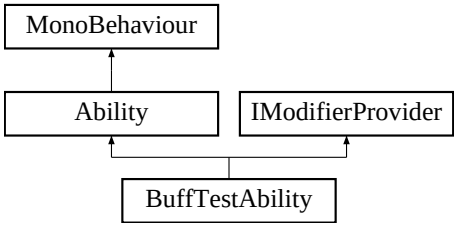
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BoomerangKit/BoomerangVision.cs

3.14 BuffTestAbility Class Reference

Inheritance diagram for BuffTestAbility:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- override void SetModifier (bool state=false)

  Called by the Modifier. Appropriate place for doing local changes.

Public Attributes

- SpriteRenderer [ ] **visuals**
- Color **activeColor**
- ModifierInfo **buff**

Additional Inherited Members

3.14.1 Member Function Documentation

3.14.1.1 ButtonDown()

```
override void BuffTestAbility.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.14.1.2 Initialize()

```
override void BuffTestAbility.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.14.1.3 SetActive()

```
override void BuffTestAbility.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

3.14.1.4 SetModifier()

```
override void BuffTestAbility.SetModifier (
            bool state = false )  [virtual]
```

Called by the Modifier. Appropriate place for doing local changes.

Parameters

state	If the modifier should be activated or deactivated.

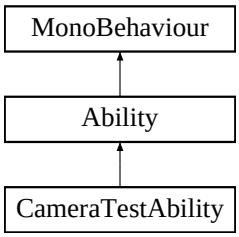
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TestAbilities/BuffTestAbility.cs

3.15 CameraTestAbility Class Reference

Inheritance diagram for CameraTestAbility:



Public Member Functions

- override void InitializeLocalPlayer (AbilityUI abilityUI)

  Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void CancelAbility ()

  Call for cancelling abilities. Override in abilities that may be interrupted.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- Transform **target**
- float **targetOrthoSize**
- float **targetViewAngle**
- float **targetViewRadius**
- float **lerpSpeed**

Additional Inherited Members

3.15.1 Member Function Documentation

3.15.1.1 ButtonDown()

```
override void CameraTestAbility.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.15.1.2 CancelAbility()

```
override void CameraTestAbility.CancelAbility ( )  [virtual]
```

Call for cancelling abilities. Override in abilities that may be interrupted.

Reimplemented from Ability.

3.15.1.3 InitializeLocalPlayer()

```
override void CameraTestAbility.InitializeLocalPlayer (
            AbilityUI abilityUI )  [virtual]
```

Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.

Reimplemented from Ability.

3.15.1.4 SetActive()

```
override void CameraTestAbility.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

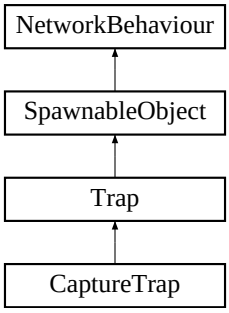
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TestAbilities/CameraTestAbility.cs

3.16 CaptureTrap Class Reference

Inheritance diagram for CaptureTrap:



Public Member Functions

- override void HandleTrigger (PlayerStatus playerStatus)

  Callback for when the trap is triggered. Sets relevant gameobjects as active to display visuals and starts a coroutine for spawning the walls.

Public Attributes

- GameObject **walls**
- float **timeBeforeWallsSpawn** = 1f
- float **pullForce** = 10f
- float **fadeSpeed** = 10f
- float **fadeTimeOffsetMultiplier** = 1.5f

Additional Inherited Members

3.16.1 Member Function Documentation

3.16.1.1 HandleTrigger()

```
override void CaptureTrap.HandleTrigger (
            PlayerStatus playerStatus )  [virtual]
```

Callback for when the trap is triggered. Sets relevant gameobjects as active to display visuals and starts a coroutine for spawning the walls.

Parameters

playerStatus	The PlayerStatus component of the player that is in the trap

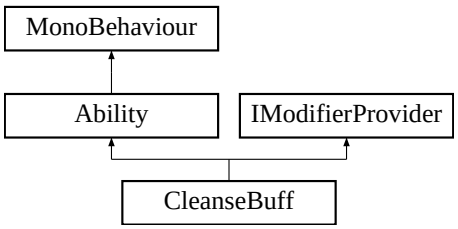
Reimplemented from Trap.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TrapperKit/CaptureTrap.cs

3.17 CleanseBuff Class Reference

Inheritance diagram for CleanseBuff:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- void **OnTriggerEnter** (Collider other)
- int **GetAbilityId** ()
- int **GetBuffModifierId** ()

Public Attributes

- ModifierInfo **buff**
- string **animatorTrigger**
- List< GameObject > **cleansedPlayers** = new List<GameObject>()

Protected Member Functions

- override void Update ()

  Just to trigger the active state when animation ends, had some issues with Animation Events.

Additional Inherited Members

3.17.1 Member Function Documentation

3.17.1.1 ButtonDown()

```
override void CleanseBuff.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.17.1.2 Initialize()

```
override void CleanseBuff.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.17.1.3 SetActive()

```
override void CleanseBuff.SetActive (
            bool state )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

3.17.1.4 Update()

```
override void CleanseBuff.Update ( )  [protected], [virtual]
```

Just to trigger the active state when animation ends, had some issues with Animation Events.

Reimplemented from Ability.

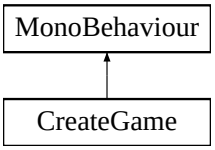
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SupportKit/CleanseBuff.cs

3.18 CreateGame Class Reference

Governs the Create Game functionality in the main menu.

Inheritance diagram for CreateGame:



Public Member Functions

- void OnBackClicked ()
  Back button method. Returns to main menu.
- void OnCreateClicked ()
  Create button method. Validates entered server name and launches game server.

Protected Member Functions

- virtual void **Start** ()

Protected Attributes

- InputField **matchNameInput**

3.18.1 Detailed Description

Governs the Create Game functionality in the main menu.

3.18.2 Member Function Documentation

3.18.2.1 OnBackClicked()

```
void CreateGame.OnBackClicked ( )
```

Back button method. Returns to main menu.

3.18.2.2 OnCreateClicked()

```
void CreateGame.OnCreateClicked ( )
```

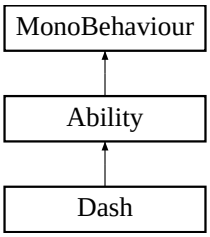
Create button method. Validates entered server name and launches game server.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/CreateGame.cs

3.19 Dash Class Reference

Inheritance diagram for Dash:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Additional Inherited Members

3.19.1 Member Function Documentation

3.19.1.1 ButtonDown()

```
override void Dash.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.19.1.2 Initialize()

```
override void Dash.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.19.1.3 SetActive()

```
override void Dash.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

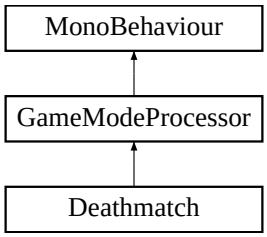
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/MarksmanKit/Dash.cs

3.20 Deathmatch Class Reference

Game mode rules processor for the deathmatch game mode

Inheritance diagram for Deathmatch:



Public Member Functions

- override void StartRound ()

  Function called on round start
- override void PlayerDies (Player player)

  Handles the death of a player - the player is removed from the local list
- override void PlayerDisconnected (Player player)

  Called when a player disconnects - removed from the local list
- override bool IsEndOfRound ()

  Determines whether it is end of round - if there is one or no players
- override void HandleRoundEnd ()

  Handles the round end.
- override string GetRoundEndText ()

  Gets the round end text - winner or draw if appropriate
- override string GetGameOverText ()

  Gets the game over text - winner or draw if appropriate

Properties

- override int ScoreWinTarget `[get]`

  Gets the score target.

Additional Inherited Members

3.20.1 Detailed Description

Game mode rules processor for the deathmatch game mode

3.20.2 Member Function Documentation

3.20.2.1 GetGameOverText()

```
override string Deathmatch.GetGameOverText ( )  [virtual]
```

Gets the game over text - winner or draw if appropriate

Returns

The game over text.

Reimplemented from GameModeProcessor.

3.20.2.2 GetRoundEndText()

```
override string Deathmatch.GetRoundEndText ( )  [virtual]
```

Gets the round end text - winner or draw if appropriate

Returns

The round end text.

Reimplemented from GameModeProcessor.

3.20.2.3 HandleRoundEnd()

```
override void Deathmatch.HandleRoundEnd ( )  [virtual]
```

Handles the round end.

Reimplemented from GameModeProcessor.

3.20.2.4 IsEndOfRound()

```
override bool Deathmatch.IsEndOfRound ( )  [virtual]
```

Determines whether it is end of round - if there is one or no players

Returns

true

`false`

Reimplemented from GameModeProcessor.

3.20.2.5 PlayerDies()

```
override void Deathmatch.PlayerDies (
            Player player )  [virtual]
```

Handles the death of a player - the player is removed from the local list

Parameters

player	Player.

Reimplemented from GameModeProcessor.

3.20.2.6 PlayerDisconnected()

```
override void Deathmatch.PlayerDisconnected (
            Player player )  [virtual]
```

Called when a player disconnects - removed from the local list

Parameters

player	The player that disconnects

Reimplemented from GameModeProcessor.

3.20.2.7 StartRound()

```
override void Deathmatch.StartRound ( )  [virtual]
```

Function called on round start

Reimplemented from GameModeProcessor.

3.20.3 Property Documentation

3.20.3.1 ScoreWinTarget

```
override int Deathmatch.ScoreWinTarget  [get]
```

Gets the score target.

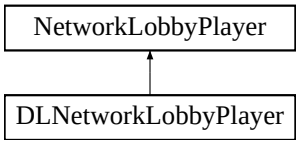
The score target.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/GameModes/Deathmatch.cs

3.21 DLNetworkLobbyPlayer Class Reference

Inheritance diagram for DLNetworkLobbyPlayer:



Public Member Functions

- override void OnClientEnterLobby ()

  Callback that initialises all necessary data for when a player enters the lobby. This includes player name, player color, adding the player to a team and telling the network manager that a new player has joined
- override void OnStartAuthority ()

  Callback that simply sets up the client side parts of a lobby for the connecting player
- override void OnClientReady (bool readyState)

  Makes the local UI uninteractable once the player has chosen to be ready
- void OnNameChange (string newName)

  SyncVarHook for handling name changes
- void OnColorChange (Color newColor)

  SyncVarHook for handling color changes
- void OnReadyStateChange (bool state)

  SyncVarHook for handling ready states
- void OnColorClicked ()

  A function that simply calls the CmdColorChange() command
- void OnReadyClicked ()

  A simple function that tells the network that this player is ready to begin
- void OnNameChanged (string str)

  A simple function that calls the CmdNameChanged(str) command
- void ToggleReadyButton (bool enabled)

  Sets the state of the ready button on the UI to the parameter one
- GameObject GetVisuals ()

  Returns the UI elements of a player
- void CmdColorChange ()

  Updates the server when a player has chosen a new team/color
- void CmdNameChanged (string name)

  Updates the server when a player has chosen a new name
- void CmdUpdateReadyState (bool state)

  Updates the server when a player is ready
- void OnDestroy ()

  Callback for when a lobby player leaves the lobby and gets destroyed It tells the lobbyHandler to remove this player and tells the networkManager that a player has left.

Public Attributes

- Button **colorButton**
- InputField **nameInput**
- Button **readyButton**
- GameObject **visuals**
- string **playerName** = ""
- Color **playerColor** = Color.white
- bool **isReady** = false

3.21.1 Member Function Documentation

3.21.1.1 CmdColorChange()

```
void DLNetworkLobbyPlayer.CmdColorChange ( )
```

Updates the server when a player has chosen a new team/color

3.21.1.2 CmdNameChanged()

```
void DLNetworkLobbyPlayer.CmdNameChanged (
            string name )
```

Updates the server when a player has chosen a new name

Parameters

name	The new player name

3.21.1.3 CmdUpdateReadyState()

```
void DLNetworkLobbyPlayer.CmdUpdateReadyState (
            bool state )
```

Updates the server when a player is ready

Parameters

state	The ready state

3.21.1.4 GetVisuals()

```
GameObject DLNetworkLobbyPlayer.GetVisuals ( )
```

Returns the UI elements of a player

Returns

The player visuals

3.21.1.5 OnClientEnterLobby()

```
override void DLNetworkLobbyPlayer.OnClientEnterLobby ( )
```

Callback that initialises all necessary data for when a player enters the lobby. This includes player name, player color, adding the player to a team and telling the network manager that a new player has joined

3.21.1.6 OnClientReady()

```
override void DLNetworkLobbyPlayer.OnClientReady (
            bool readyState )
```

Makes the local UI uninteractable once the player has chosen to be ready

Parameters

readyState	Whether the client is ready or not

3.21.1.7 OnColorChange()

```
void DLNetworkLobbyPlayer.OnColorChange (
            Color newColor )
```

SyncVarHook for handling color changes

Parameters

newColor	The new team color

3.21.1.8 OnColorClicked()

```
void DLNetworkLobbyPlayer.OnColorClicked ( )
```

A function that simply calls the CmdColorChange() command

3.21.1.9 OnDestroy()

```
void DLNetworkLobbyPlayer.OnDestroy ( )
```

Callback for when a lobby player leaves the lobby and gets destroyed It tells the lobbyHandler to remove this player and tells the networkManager that a player has left.

3.21.1.10 OnNameChange()

```
void DLNetworkLobbyPlayer.OnNameChange (
            string newName )
```

SyncVarHook for handling name changes

Parameters

newName	The new player name

3.21.1.11 OnNameChanged()

```
void DLNetworkLobbyPlayer.OnNameChanged (
            string str )
```

A simple function that calls the CmdNameChanged(str) command

Parameters

str	The new player name

3.21.1.12 OnReadyClicked()

```
void DLNetworkLobbyPlayer.OnReadyClicked ( )
```

A simple function that tells the network that this player is ready to begin

3.21.1.13 OnReadyStateChange()

```
void DLNetworkLobbyPlayer.OnReadyStateChange (
            bool state )
```

SyncVarHook for handling ready states

Parameters

state	If the is player ready or not

3.21.1.14 OnStartAuthority()

```
override void DLNetworkLobbyPlayer.OnStartAuthority ( )
```

Callback that simply sets up the client side parts of a lobby for the connecting player

3.21.1.15 ToggleReadyButton()

```
void DLNetworkLobbyPlayer.ToggleReadyButton (
            bool enabled )
```

Sets the state of the ready button on the UI to the parameter one

Parameters

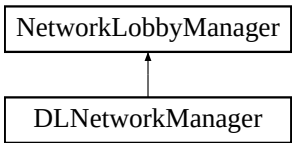
enabled	The state of the button

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Networking/DLNetworkLobbyPlayer.cs

3.22 DLNetworkManager Class Reference

Inheritance diagram for DLNetworkManager:



Public Member Functions

- void OnPlayerNumberModified (int count)

  Updates the playerCount variable by adding the parameter
- override bool OnLobbyServerSceneLoadedForPlayer (GameObject lobbyPlayer, GameObject gamePlayer)

  A callback for when all players are ready and the game is about to start. It takes each lobby player and applies the saved data of those to the actual game players
- override void **OnLobbyServerDisconnect** (NetworkConnection conn)
- override GameObject OnLobbyServerCreateLobbyPlayer (NetworkConnection conn, short playerControllerId)

  Callback what the server has to do once it creates a lobby player The server instantiates the player and toggles relevant UI for all players
- override void OnClientError (NetworkConnection conn, int errorCode)

  Callback for handling client errors. It currently only sends the player out of the lobby.

Public Attributes

- int **playerCount** = 0

3.22.1 Member Function Documentation

3.22.1.1 OnClientError()

```
override void DLNetworkManager.OnClientError (
            NetworkConnection conn,
            int errorCode )
```

Callback for handling client errors. It currently only sends the player out of the lobby.

Parameters

conn	The network connection
errorCode	The error code

3.22.1.2 OnLobbyServerCreateLobbyPlayer()

```
override GameObject DLNetworkManager.OnLobbyServerCreateLobbyPlayer (
            NetworkConnection conn,
            short playerControllerId )
```

Callback what the server has to do once it creates a lobby player The server instantiates the player and toggles relevant UI for all players

Parameters

conn	The network connection. Currently not used
playerControllerId	The local player contoller Id. Currently not used

Returns

The instantiated lobby player object

3.22.1.3 OnLobbyServerSceneLoadedForPlayer()

```
override bool DLNetworkManager.OnLobbyServerSceneLoadedForPlayer (
            GameObject lobbyPlayer,
            GameObject gamePlayer )
```

A callback for when all players are ready and the game is about to start. It takes each lobby player and applies the saved data of those to the actual game players

Parameters

lobbyPlayer	The lobby player
gamePlayer	The game player that we are transferring data to

Returns

3.22.1.4 OnPlayerNumberModified()

```
void DLNetworkManager.OnPlayerNumberModified (
            int count )
```

Updates the playerCount variable by adding the parameter

Parameters

count	The amount of new players

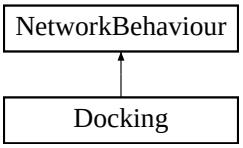
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Networking/DLNetworkManager.cs

3.23 Docking Class Reference

Handles the DockingKit interactions for each Player.

Inheritance diagram for Docking:



Public Member Functions

- void Initialize (Player pl)

  Called when this object is activated on a client. Sets up the initial state and references.
- void **SetDefaults** ()
- NetworkConnection **GetConnectionToClient** ()

- bool CheckDamagable (Player otherPlayer)

  Check if the other player is damagable by this player. Unassigned team id means teams aren't used.
- void CmdSetDockingKitId (DockingKitId newKit)

  Command which sets the SyncVar "dockingKitId". This is synchronized to all clients.
- void CmdSetSwitchState (bool state)

  Command for setting the state of the switchingKit member. switchingKit determines whether we want to spawn a pickup on undocking or not
- void RpcSetSwitchState (bool state)

  ClientRpc for synchronizing the switchingKit state
- void SetDockingKit (DockingKitId newKitId)

  Spawns in the DockingKit locally for the given new DockingKitId. Updates UI when called for local player.
- DockingKit GetDockingKit ()

  Get the active DockingKit for this Docking.
- void RemoveDockingKit (bool spawnPickup=false)

  Remove the current docking kit.
- void SetDockingKitStats (DockingKit kit)

  Updates the stats given by the current DockingKit.
- void SetPlayerInputRestriction (bool state, params InputType[ ] inputTypes)

  Passes the parameters along to the PlayerInput if called by the local player.
- void OnDockingButtonDown ()

  Called when the dock button is pressed.
- void CmdOnPlayerDocking (GameObject pickup)

  Command called when the local player wants to dock to a DockingKitPickup.
- void OnUndockingButtonDown ()

  Called when the undock button is pressed.
- void OnAbilityButtonChange (int abilityId, bool down)

  Called when the ability button is initially pressed or released.
- void CancelAbilities ()

  Cancels all the abilities in the current docking kit.
- void CmdSetActive (int abilityId, bool state)

  Command for activating an ability. Synchronizes activation to all clients.
- void RpcSetActive (int abilityId, bool state)

  ClientRpc for activating an ability. Runs locally on every client. Returns immediately for the local player, as the activation already happened locally.
- void CmdSpawnObject (int abilityId, int prefabId, Vector3 position, Vector3 rotation)

  Command for spawning prefab objects. Used by the abilities.
- void CmdSpawnObjectReference (int abilityId, int prefabId, Vector3 position, Vector3 rotation)

  Command for spawning prefab objects. Used by the abilities. Returns a reference to the spawned GameObject to the client/ability that called the Command.
- void TargetSetSpawnObjectReference (NetworkConnection connection, GameObject spawnedObject, int abilityId)

  TargetRpc for getting the reference to a spawned object.
- void CmdDestroyObject (GameObject destroyGameObject)

  Command used to destroy objects by objects that don't have authority to Command themselves.
- void CmdSpawnDockingKitPickup (DockingKitId kitId)

  Command for spawning docking kit pickup on undocking.
- void CmdSetModifier (int abilityId, int modifierId, bool apply)

  Command called by abilities by the local player to apply or remove a modifier.
- void SetModifier (int abilityId, bool state)

  Called by Modifiers OnClient functions to change the state of the modifier on each client.
- void CmdServerCallback (int abilityId, int functionId)

Command used by abilities to run code on the server, as they're not NetworkBehaviour (or has authority) to call commands.

- void **CmdServerCallbackTwoVector3** (int abilityId, int functionId, Vector3 firstVec3, Vector3 secondVec3)
- void **CmdServerCallbackGameObject** (int abilityId, int functionId, GameObject go)
- void **CmdServerCallbackFloat** (int abilityId, int functionId, float param)
- void **CmdServerCallbackBool** (int abilitiyId, int functionId, bool param)
- void **CmdServerCallbackGameObjectFloat** (int abilityId, int functionId, GameObject param1, float param2)
- void RpcClientCallback (int abilityId, int functionId)

  ClientRpc used by abilities to run code on every client, as they're not NetworkBehaviour (or has authority) to call client rpcs.
- void **RpcClientCallbackVector3** (int abilityId, int functionId, Vector3 firstVec3)
- void **RpcClientCallbackTwoVector3** (int abilityId, int functionId, Vector3 firstVec3, Vector3 secondVec3)
- void **RpcClientCallbackGameObject** (int abilityId, int functionId, GameObject go)
- void **RpcClientCallbackFloat** (int abilityId, int functionId, float param)
- void **RpcClientCallbackBool** (int abilityId, int functionId, bool param)
- void TargetClientCallback (NetworkConnection connection, int abilityId, int functionId)

  TargetRpc used by abilities to run code on a target client, as they're not NetworkBehaviour (or has authority) to call target rpcs.
- void TargetReduceCooldown (NetworkConnection connection, int abilityId, float reductionAmount)

  TargetRpc for reducing the cooldown an ability by a certain amount.
- void **CmdSetElement** (ElementalContainer.ComboableElements element, int abilityId)
- void **RpcSetElement** (ElementalContainer.ComboableElements element, int abilityId)

Public Attributes

- GameObject **dockingKitPickupPrefab**
- DockingKit **basicDockingKit**
- float **dockingTime** = 2f
- DockingKitId **dockingKitId** = DockingKitId.Empty

3.23.1 Detailed Description

Handles the DockingKit interactions for each Player.

3.23.2 Member Function Documentation

3.23.2.1 CancelAbilities()

```
void Docking.CancelAbilities ( )
```

Cancels all the abilities in the current docking kit.

3.23.2.2 CheckDamagable()

```
bool Docking.CheckDamagable (
            Player otherPlayer )
```

Check if the other player is damagable by this player. Unassigned team id means teams aren't used.

Parameters

otherPlayer	The other player.

Returns

True if damagable, false otherwise.

3.23.2.3 CmdDestroyObject()

```
void Docking.CmdDestroyObject (
            GameObject destroyGameObject )
```

Command used to destroy objects by objects that don't have authority to Command themselves.

Parameters

destroyGameObject	The reference to the object to be destroyed.

3.23.2.4 CmdOnPlayerDocking()

```
void Docking.CmdOnPlayerDocking (
            GameObject pickup )
```

Command called when the local player wants to dock to a DockingKitPickup.

Parameters

pickup	Reference to the networked pickup object.

3.23.2.5 CmdServerCallback()

```
void Docking.CmdServerCallback (
            int abilityId,
            int functionId )
```

Command used by abilities to run code on the server, as they're not NetworkBehaviour (or has authority) to call commands.

Parameters

abilityId	The id of the ability calling the command.
functionId	The id of the function to be run on the server.

3.23.2.6 CmdSetActive()

```
void Docking.CmdSetActive (
            int abilityId,
            bool state )
```

Command for activating an ability. Synchronizes activation to all clients.

Parameters

abilityId	Index of the ability to activate.
state	If the ability should be activated or deactivated.

3.23.2.7 CmdSetDockingKitId()

```
void Docking.CmdSetDockingKitId (
            DockingKitId newKit )
```

Command which sets the SyncVar "dockingKitId". This is synchronized to all clients.

Parameters

newKit	The new DockingKitId.

3.23.2.8 CmdSetModifier()

```
void Docking.CmdSetModifier (
            int abilityId,
            int modifierId,
            bool apply )
```

Command called by abilities by the local player to apply or remove a modifier.

Parameters

abilityId	The id of the ability that applied the modifier.
modifierIndex	The index of the modifier.
apply	If the modifier should be applied or removed.

3.23.2.9 CmdSetSwitchState()

```
void Docking.CmdSetSwitchState (
            bool state )
```

Command for setting the state of the switchingKit member. switchingKit determines whether we want to spawn a pickup on undocking or not

Parameters

state	The new state of the bool

3.23.2.10 CmdSpawnDockingKitPickup()

```
void Docking.CmdSpawnDockingKitPickup (
            DockingKitId kitId )
```

Command for spawning docking kit pickup on undocking.

Parameters

kitId	Which docking kit to spawn.

3.23.2.11 CmdSpawnObject()

```
void Docking.CmdSpawnObject (
            int abilityId,
            int prefabId,
            Vector3 position,
            Vector3 rotation )
```

Command for spawning prefab objects. Used by the abilities.

Parameters

abilityId	Index of the ability calling the Command.
prefabId	Index of the prefab to spawn from the ability.
position	Position of the new object.
rotation	Orientation of the new object (in eulerAngles).

3.23.2.12 CmdSpawnObjectReference()

```
void Docking.CmdSpawnObjectReference (
            int abilityId,
            int prefabId,
            Vector3 position,
            Vector3 rotation )
```

Command for spawning prefab objects. Used by the abilities. Returns a reference to the spawned GameObject to the client/ability that called the Command.

Parameters

abilityId	Index of the ability calling the Command.
prefab↩ Id	Index of the prefab to spawn from the ability.
position	Position of the new object.
rotation	Orientation of the new object (in eulerAngles).

3.23.2.13 GetDockingKit()

```
DockingKit Docking.GetDockingKit ( )
```

Get the active DockingKit for this Docking.

Returns

The current DockingKit.

3.23.2.14 Initialize()

```
void Docking.Initialize (
            Player pl )
```

Called when this object is activated on a client. Sets up the initial state and references.

3.23.2.15 OnAbilityButtonChange()

```
void Docking.OnAbilityButtonChange (
            int abilityId,
            bool down )
```

Called when the ability button is initially pressed or released.

Parameters

abilityId	Index of the ability where the button state changed.
down	If this was the initial press.

3.23.2.16 OnDockingButtonDown()

```
void Docking.OnDockingButtonDown ( )
```

Called when the dock button is pressed.

3.23.2.17 OnUndockingButtonDown()

```
void Docking.OnUndockingButtonDown ( )
```

Called when the undock button is pressed.

3.23.2.18 RemoveDockingKit()

```
void Docking.RemoveDockingKit (
            bool spawnPickup = false )
```

Remove the current docking kit.

Parameters

spawnPickup	Whether to spawn a pickup

3.23.2.19 RpcClientCallback()

```
void Docking.RpcClientCallback (
            int abilityId,
            int functionId )
```

ClientRpc used by abilities to run code on every client, as they're not NetworkBehaviour (or has authority) to call client rpcs.

Parameters

abilityId	The id of the ability calling the rpc.
function↩ Id	The id of the function to be run on every client.

3.23.2.20 RpcSetActive()

```
void Docking.RpcSetActive (
            int abilityId,
            bool state )
```

ClientRpc for activating an ability. Runs locally on every client. Returns immediately for the local player, as the activation already happened locally.

Parameters

ability↩ Id	Index of the ability to activate.
state	If the ability should be activated or deactivated.

3.23.2.21 RpcSetSwitchState()

```
void Docking.RpcSetSwitchState (
            bool state )
```

ClientRpc for synchronizing the switchingKit state

Parameters

state	The new state of the bool

3.23.2.22 SetDockingKit()

```
void Docking.SetDockingKit (
            DockingKitId newKitId )
```

Spawns in the DockingKit locally for the given new DockingKitId. Updates UI when called for local player.

Parameters

new↩ KitId	The new DockingKitId.

3.23.2.23 SetDockingKitStats()

```
void Docking.SetDockingKitStats (
            DockingKit kit )
```

Updates the stats given by the current DockingKit.

Parameters

kit	Which DockingKit to retrieve the stats from.

3.23.2.24 SetModifier()

```
void Docking.SetModifier (
            int abilityId,
            bool state )
```

Called by Modifiers OnClient functions to change the state of the modifier on each client.

Parameters

abilityId	The id of the ability that applied the modifier.
state	The active state of the modifier.

3.23.2.25 SetPlayerInputRestriction()

```
void Docking.SetPlayerInputRestriction (
            bool state,
            params InputType [] inputTypes )
```

Passes the parameters along to the PlayerInput if called by the local player.

Parameters

state	The new state of the input restriction.
inputTypes	The types to set restriction for.

3.23.2.26 TargetClientCallback()

```
void Docking.TargetClientCallback (
            NetworkConnection connection,
            int abilityId,
            int functionId )
```

TargetRpc used by abilities to run code on a target client, as they're not NetworkBehaviour (or has authority) to call target rpcs.

Parameters

connection	Needed so TargetRpc finds the correct client.
abilityId	The id of the ability calling the target rpc.
functionId	The id of the function to be run on the targeted client.

3.23.2.27 TargetReduceCooldown()

```
void Docking.TargetReduceCooldown (
            NetworkConnection connection,
            int abilityId,
            float reductionAmount )
```

TargetRpc for reducing the cooldown an ability by a certain amount.

Parameters

connection	The NetworkConnection associated with the player given the reduction.
abilityId	The id of the ability to get cooldown reduction.
reductionAmount	The amount deducted for the current cooldown.

3.23.2.28 TargetSetSpawnObjectReference()

```
void Docking.TargetSetSpawnObjectReference (
            NetworkConnection connection,
            GameObject spawnedObject,
            int abilityId )
```

TargetRpc for getting the reference to a spawned object.

Parameters

connection	Needed so TargetRpc finds the correct client.
spawnedObject	Reference to the GameObject spawned.
abilityId	The id of the ability that called the spawn command.

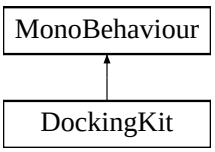
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Player/Docking.cs

3.24 DockingKit Class Reference

Handles the interaction between the Docking and the abilities.

Inheritance diagram for DockingKit:



Public Member Functions

- void Initialize (Docking dock)

  Initialization that happens locally on every client.
- void OnLocalPlayerInitialization (PlayerUIHandler playerUIHandler)

  Initialization that only happens for the local player (Player controlling this docking kit).
- void OnLocalPlayerDocking (float dockingTime, PlayerUIHandler playerUIHandler)

  Initialization called for the local player (Player controlling this docking kit) on docking.
- void OnUndocking (float dockingDuration, DockingKitId spawnPickupId, bool spawnPickup=true)

  Called for every client when undocking.
- void **OnRemoveKit** (DockingKitId spawnPickupId=DockingKitId.Empty)
- void OnAbilityButtonChange (int abilityId, bool down)

  Called when the ability button is initially pressed or released.
- void CancelAbilities ()

  Cancels all the abilities in this docking kit.
- void SetAbilityLock (bool state, params int[ ] abilityNumbers)

  Used by Abilities to lock abilities in this docking kit.

Public Attributes

- float **moveSpeed** = 60f
- float **rotationSpeed** = 6f
- float **maxHealth** = 100f
- List< Ability > **abilities**

3.24.1 Detailed Description

Handles the interaction between the Docking and the abilities.

3.24.2 Member Function Documentation

3.24.2.1 CancelAbilities()

```
void DockingKit.CancelAbilities ( )
```

Cancels all the abilities in this docking kit.

3.24.2.2 Initialize()

```
void DockingKit.Initialize (
            Docking dock )
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.

3.24.2.3 OnAbilityButtonChange()

```
void DockingKit.OnAbilityButtonChange (
            int abilityId,
            bool down )
```

Called when the ability button is initially pressed or released.

ButtonDown may be called without ButtonUp running afterwards, handle this in Ability.CancelAbility (if the ability is locked in between). ButtonUp may be called without ButtonDown running first (if the ability is unlocked in between).

Parameters

abilityId	Index of the ability where the button state changed.
down	If this was the initial press.

3.24.2.4 OnLocalPlayerDocking()

```
void DockingKit.OnLocalPlayerDocking (
            float dockingTime,
            PlayerUIHandler playerUIHandler )
```

Initialization called for the local player (Player controlling this docking kit) on docking.

Parameters

dockingTime	The time used to dock. (Immobile duration)

3.24.2.5 OnLocalPlayerInitialization()

```
void DockingKit.OnLocalPlayerInitialization (
            PlayerUIHandler playerUIHandler )
```

Initialization that only happens for the local player (Player controlling this docking kit).

3.24.2.6 OnUndocking()

```
void DockingKit.OnUndocking (
            float dockingDuration,
            DockingKitId spawnPickupId,
            bool spawnPickup = true )
```

Called for every client when undocking.

Parameters

dockingDuration	The time used to undock. (Immobile duration)
spawnPickupId	The DockingKitId of the pickup to be spawned on undocking.
spawnPickup	Whether to spawn the pickup.

3.24.2.7 SetAbilityLock()

```
void DockingKit.SetAbilityLock (
            bool state,
            params int [] abilityNumbers )
```

Used by Abilities to lock abilities in this docking kit.

Parameters

state	To lock or unlock.
abilityNumbers	Toggles lock for these abilities.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/DockingKit.cs

3.25 DockingKitDescriptions Struct Reference

Public Attributes

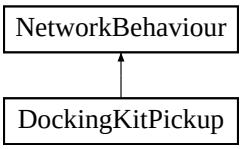
- Sprite **icon**
- string **name**
- string **description**

The documentation for this struct was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/ShopItemData.cs

3.26 DockingKitPickup Class Reference

Inheritance diagram for DockingKitPickup:



Public Member Functions

- override void OnStartClient ()

  Calls the SyncVar hook manually to get the correct initial state. Used by clients connecting after pickup already spawned.
- void OnPlayerDocking (GameObject player)

  Server call from the Docking called when a player tries to dock.

Public Attributes

- DockingKitId **dockingKitId** = DockingKitId.Empty

3.26.1 Member Function Documentation

3.26.1.1 OnPlayerDocking()

```
void DockingKitPickup.OnPlayerDocking (
            GameObject player )
```

Server call from the Docking called when a player tries to dock.

Parameters

player	Reference to the player docking.

3.26.1.2 OnStartClient()

```
override void DockingKitPickup.OnStartClient ( )
```

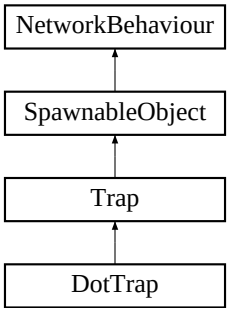
Calls the SyncVar hook manually to get the correct initial state. Used by clients connecting after pickup already spawned.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/DockingKitPickup.cs

3.27 DotTrap Class Reference

Inheritance diagram for DotTrap:



Public Member Functions

- override void HandleTrigger (PlayerStatus playerStatus)

  Callback that allows this trap to do whatever it wants whenever it is triggered. This one simply applies the member structs containing modifier info

Public Attributes

- ModifierInfo **dotInfo**
- ModifierInfo **slowInfo**

Additional Inherited Members

3.27.1 Member Function Documentation

3.27.1.1 HandleTrigger()

```
override void DotTrap.HandleTrigger (
            PlayerStatus playerStatus )  [virtual]
```

Callback that allows this trap to do whatever it wants whenever it is triggered. This one simply applies the member structs containing modifier info

Parameters

playerStatus	The PlayerStatus component of the player that is in the trap

Reimplemented from Trap.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TrapperKit/DotTrap.cs

3.28 ElementalModifiers Class Reference

Public Member Functions

- void **Initialize** ()
- void **SetModifier** (bool state)
- ModifierInfo **GetModifierInfo** (int modifierId)
- void TransferElementalModifier (Collider other, Docking docking, int abilityId)

  Handles the transferring of the elemental buff by applying it as a debuff to the player that was hit
- void **ApplyElement** (ElementalContainer.ComboableElements element, Docking docking, int abilityId)
- void **SetElement** (ElementalContainer.ComboableElements element)

Public Attributes

- Transform **elementEffectTransform**
- ModifierInfo **fireBuff**
- ModifierInfo **iceBuff**
- ModifierInfo **electricBuff**
- ModifierInfo **fireDebuff**
- ModifierInfo **iceDebuff**
- ModifierInfo **electricDebuff**

3.28.1 Member Function Documentation

3.28.1.1 TransferElementalModifier()

```
void ElementalModifiers.TransferElementalModifier (
            Collider other,
            Docking docking,
            int abilityId )
```

Handles the transferring of the elemental buff by applying it as a debuff to the player that was hit

Parameters

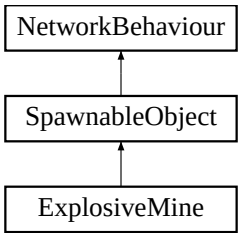
other	The collider we want to apply the debuff to

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ElementalModifiers.cs

3.29 ExplosiveMine Class Reference

Inheritance diagram for ExplosiveMine:



Public Member Functions

- void **Initialize** (GameObject owner)
- void **Start** ()
- void **OnTriggerEnter** (Collider other)
- void RpcRemoveMine ()

  Destroy the mine and remove it from the list of mines.
- void **RpcPlayAnimation** ()
- void **RpcMineVisualState** (bool state)
- void **RpcExplodeVisualState** (bool state)

Public Attributes

- float **baseDamage**
- float **maxDamageTapering**
- float **explosionForce**
- float **explosionRadius**
- float **activationTime**
- string **animationTrigger**
- GameObject **explodeSprite**
- GameObject **mineSprite**
- Animator **animator**
- int **myId**
- GameObject **spawnerReference**

Additional Inherited Members

3.29.1 Member Function Documentation

3.29.1.1 RpcRemoveMine()

```
void ExplosiveMine.RpcRemoveMine ( )
```

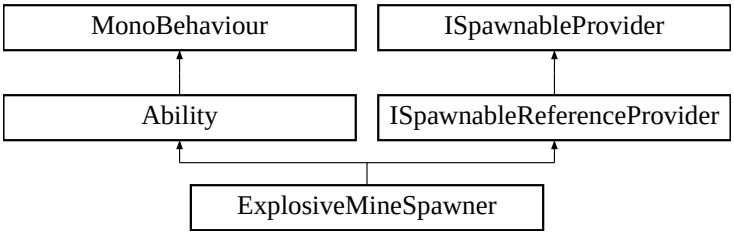
Destroy the mine and remove it from the list of mines.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BomberKit/ExplosiveMine.cs

3.30 ExplosiveMineSpawner Class Reference

Inheritance diagram for ExplosiveMineSpawner:



Public Member Functions

- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- void RemoveMine (int mineId)

  Removes the mine that got triggered
- void OnDestroy ()

  Clean up mines when docking kit is not equipped anymore.

Public Attributes

- string **animatorTrigger**
- GameObject [ ] **minePrefab**
- int **maxMineAmount**

Additional Inherited Members

3.30.1 Member Function Documentation

3.30.1.1 ButtonDown()

```
override void ExplosiveMineSpawner.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.30.1.2 OnDestroy()

```
void ExplosiveMineSpawner.OnDestroy ( )
```

Clean up mines when docking kit is not equipped anymore.

3.30.1.3 RemoveMine()

```
void ExplosiveMineSpawner.RemoveMine (
            int mineId )
```

Removes the mine that got triggered

Parameters

mineId	The ID of the mine.

3.30.1.4 SetActive()

```
override void ExplosiveMineSpawner.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

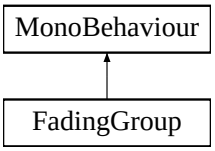
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BomberKit/ExplosiveMineSpawner.cs

3.31 FadingGroup Class Reference

Inheritance diagram for FadingGroup:



Public Member Functions

- Fade **GetCurrentFade** ()
- void StartFade (Fade fade, float fTime, Action finishFade=null, bool reactivate=true)

  Starts the fading of a panel.
- void FadeOutToValue (float fTime, float fOutValue, Action finishFade=null)

  Fades the panel to a given value.
- void StartFadeOrFireEvent (Fade fade, float fadeTime, Action finishFade=null)

  Starts a fade, and fires the provided event if the gameObject is disabled.
- void StopFade (bool setVisible)

  Stops the fade, snapping the alpha and activating or deactivating the gameObject.

3.31.1 Member Function Documentation

3.31.1.1 FadeOutToValue()

```
void FadingGroup.FadeOutToValue (
            float fTime,
            float fOutValue,
            Action finishFade = null )
```

Fades the panel to a given value.

Parameters

fadeTime	Fade time.
fadeOutValue	Value to fade to.
finishFade	Delegate to fire once fade is complete.

3.31.1.2 StartFade()

```
void FadingGroup.StartFade (
            Fade fade,
            float fTime,
            Action finishFade = null,
            bool reactivate = true )
```

Starts the fading of a panel.

Parameters

fade	The fade type to use.
fadeTime	Fade time.
finishFade	Delegate to fire once fade is complete.
reactivate	Whether to reactivate this gameobject for the purposes of the fade.

3.31.1.3 StartFadeOrFireEvent()

```
void FadingGroup.StartFadeOrFireEvent (
            Fade fade,
            float fadeTime,
            Action finishFade = null )
```

Starts a fade, and fires the provided event if the gameObject is disabled.

Parameters

fade	Fade type to use.
fadeTime	Fade time.
finishFade	Delegate to fire if object is disabled.

3.31.1.4 StopFade()

```
void FadingGroup.StopFade (
            bool setVisible )
```

Stops the fade, snapping the alpha and activating or deactivating the gameObject.

Parameters

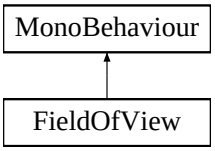
setVisible	Whether the panel should be visible or invisible on stop.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/FadingGroup.cs

3.32 FieldOfView Class Reference

Inheritance diagram for FieldOfView:



Public Member Functions

- Vector3 **DirFromAngle** (float angleInDegrees, bool angleIsGlobal)
- void **SetViewRadius** (float newRadius, float speed)
- void **ResetViewRadius** (float speed)
- void **SetViewAngle** (float newAngle, float speed)
- void **ResetViewAngle** (float speed)

Public Attributes

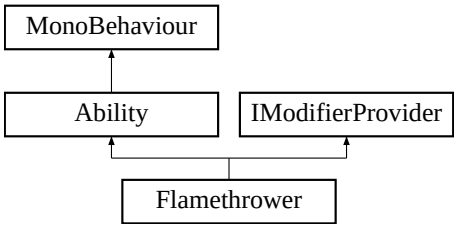
- float **viewRadius**
- float **viewAngle**
- LayerMask **obstacleMask**
- float **meshResolution** = 1
- int **edgeResolveIterations** = 1
- float **edgeDstThreshold** = 0.5f
- float **maskCutawayDst** = 0.4f
- MeshFilter **viewMeshFilter**

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Player/FieldOfView.cs

3.33 Flamethrower Class Reference

Inheritance diagram for Flamethrower:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- override void SetModifier (bool state)

  Called by the Modifier. Appropriate place for doing local changes.
- void SetBuffState (bool state)

  Sets the visual state of the flamethrower to the given parameter state

Public Attributes

- SpriteRenderer **head**
- ModifierInfo **buff**
- ModifierInfo **dot**
- GameObject **flamethrowerContainer**
- Color **headColorWhileActive**

Additional Inherited Members

3.33.1 Member Function Documentation

3.33.1.1 ButtonDown()

```
override void Flamethrower.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.33.1.2 Initialize()

```
override void Flamethrower.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.33.1.3 SetActive()

```
override void Flamethrower.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

3.33.1.4 SetBuffState()

```
void Flamethrower.SetBuffState (
            bool state )
```

Sets the visual state of the flamethrower to the given parameter state

Parameters

state	The state of the flamethrower

3.33.1.5 SetModifier()

```
override void Flamethrower.SetModifier (
            bool state )  [virtual]
```

Called by the Modifier. Appropriate place for doing local changes.

Parameters

state	If the modifier should be activated or deactivated.

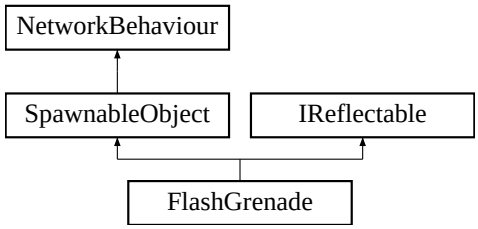
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TrapperKit/Flamethrower.cs

3.34 FlashGrenade Class Reference

Inheritance diagram for FlashGrenade:



Public Attributes

- float **timeBeforeExplosion** = 2f
- float **initialSpeed** = 5f
- SphereCollider **explosionCollider**
- GameObject **visuals**
- int **lifeTimeAfterExplosion** = 1
- float **visionRadius** = 20
- float **lerpSpeed** = 10f
- ParticleSystem **explosionParticles**
- ModifierInfo **stunInfo**

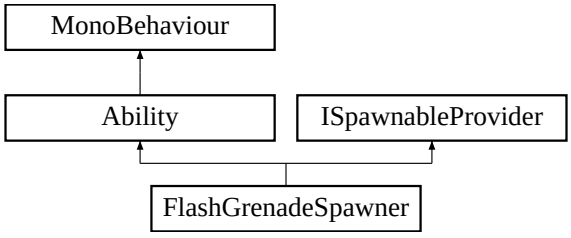
Additional Inherited Members

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BrawlerKit/FlashGrenade.cs

3.35 FlashGrenadeSpawner Class Reference

Inheritance diagram for FlashGrenadeSpawner:



Public Member Functions

- override void ButtonDown ()

  Callback for what this ability should do once its associated button has been pressed
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- GameObject **flashGrenadePrefab**
- float **offset** = 5

Additional Inherited Members

3.35.1 Member Function Documentation

3.35.1.1 ButtonDown()

```
override void FlashGrenadeSpawner.ButtonDown ( )  [virtual]
```

Callback for what this ability should do once its associated button has been pressed

Implements Ability.

3.35.1.2 SetActive()

```
override void FlashGrenadeSpawner.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

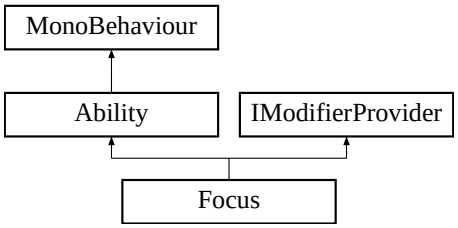
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BrawlerKit/FlashGrenadeSpawner.cs

3.36 Focus Class Reference

Inheritance diagram for Focus:



Public Member Functions

- override void InitializeLocalPlayer (AbilityUI abilityUI)

  Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void CancelAbility ()

  Call for cancelling abilities. Override in abilities that may be interrupted.
- override void SetActive (bool state)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- string **animatorBool**
- float **maxDuration** = 15f
- Transform **target**
- float **targetOrthoSize**
- Slingshot **slingshot**
- Transform **leftSlingHandle**
- Transform **rightSlingHandle**
- ModifierInfo **focusModifier**
- float **targetViewAngle**
- float **targetViewRadius**
- float **lerpSpeed**

Additional Inherited Members

3.36.1 Member Function Documentation

3.36.1.1 ButtonDown()

```
override void Focus.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.36.1.2 CancelAbility()

```
override void Focus.CancelAbility ( )  [virtual]
```

Call for cancelling abilities. Override in abilities that may be interrupted.

Reimplemented from Ability.

3.36.1.3 InitializeLocalPlayer()

```
override void Focus.InitializeLocalPlayer (
            AbilityUI abilityUI )  [virtual]
```

Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.

Reimplemented from Ability.

3.36.1.4 SetActive()

```
override void Focus.SetActive (
            bool state )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

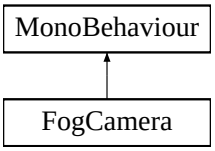
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SniperKit/Focus.cs

3.37 FogCamera Class Reference

Inheritance diagram for FogCamera:



Public Attributes

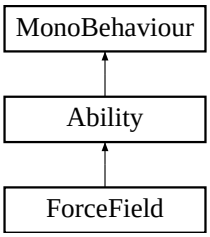
- Shader **replacementShader**
- Color **fogColor**

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/FogCamera.cs

3.38 ForceField Class Reference

Inheritance diagram for ForceField:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- float **playerForce** = 10f
- string **animatorTrigger**

Additional Inherited Members

3.38.1 Member Function Documentation

3.38.1.1 ButtonDown()

```
override void ForceField.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.38.1.2 Initialize()

```
override void ForceField.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.38.1.3 SetActive()

```
override void ForceField.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

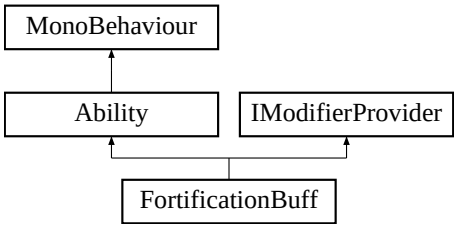
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TankKit/ForceField.cs

3.39 FortificationBuff Class Reference

Inheritance diagram for FortificationBuff:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- bool **IsActive** ()
- int **GetBuffModifierId** ()
- int **GetAbilityId** ()

Public Attributes

- ModifierInfo **buff**
- float **activeDuration**

Protected Member Functions

- override void Update ()

  Runs on every client, but only the local player has cooldown initialized.

Additional Inherited Members

3.39.1 Member Function Documentation

3.39.1.1 ButtonDown()

```
override void FortificationBuff.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.39.1.2 Initialize()

```
override void FortificationBuff.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.39.1.3 SetActive()

```
override void FortificationBuff.SetActive (
            bool state )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

3.39.1.4 Update()

```
override void FortificationBuff.Update ( )  [protected], [virtual]
```

Runs on every client, but only the local player has cooldown initialized.

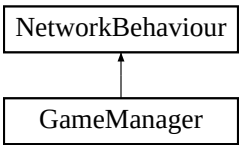
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SupportKit/FortificationBuff.cs

3.40 GameManager Class Reference

Inheritance diagram for GameManager:



Public Member Functions

- GameObject GetDockingKit (DockingKitId id)

  Used for retrieving a DockingKit prefab from a DockingKitId.
- void RemovePlayer (Player player)

  Removes the player.
- void HandleEveryoneBailed ()

  Handles everyone bailed.
- void ExitGame (MenuPage returnPage)

  Exits the game.
- void HandleKill (Player killed)

  Handles the kill
- void ServerResetAllPlayers ()

  Resets all the players on the server
- void RespawnPlayer (int playerNumber, TeamId playerTeamId)

  Respawns the player
- void RpcRespawnPlayer (int playerNumber, int spawnPointIndex)

  Rpc for respawning the player
- void ClientReady ()

  Clients the ready
- void EnablePlayerControl ()

  Enables the player control
- void DisablePlayerControl ()

  Disables the player control

Static Public Member Functions

- static void AddPlayer (Player player)

  Add a player from the lobby hook

Public Attributes

- List< GameObject > **dockingKitPrefabs**
- PlayerUIHandler **playerUIHandler**
- IngameMenuHandler **ingameMenuHandler**

Static Public Attributes

- static GameManager **Instance**
- static List< Player > **Players** = new List<Player>()

Protected Member Functions

- void StartUp ()

  State up state function
- void Preplay ()

  Preplay state function

Protected Attributes

- GameState **gameState** = GameState.Inactive

Properties

- GameState **CurrentGameState** `[get]`
- GameModeProcessor **ModeProcessor** `[get]`
- bool **HasEveryoneBailed** `[get]`

3.40.1 Member Function Documentation

3.40.1.1 AddPlayer()

```
static void GameManager.AddPlayer (
            Player player )  [static]
```

Add a player from the lobby hook

3.40.1.2 ClientReady()

```
void GameManager.ClientReady ( )
```

Clients the ready

3.40.1.3 DisablePlayerControl()

```
void GameManager.DisablePlayerControl ( )
```

Disables the player control

3.40.1.4 EnablePlayerControl()

```
void GameManager.EnablePlayerControl ( )
```

Enables the player control

3.40.1.5 ExitGame()

```
void GameManager.ExitGame (
            MenuPage returnPage )
```

Exits the game.

Parameters

returnPage	Return page.

3.40.1.6 GetDockingKit()

```
GameObject GameManager.GetDockingKit (
            DockingKitId id )
```

Used for retrieving a DockingKit prefab from a DockingKitId.

Parameters

id	Index of DockingKit to return.

Returns

The DockingKit prefab for the given DockingKitId.

3.40.1.7 HandleEveryoneBailed()

```
void GameManager.HandleEveryoneBailed ( )
```

Handles everyone bailed.

3.40.1.8 HandleKill()

```
void GameManager.HandleKill (
            Player killed )
```

Handles the kill

Parameters

killed	Killed

3.40.1.9 Preplay()

```
void GameManager.Preplay ( )  [protected]
```

Preplay state function

3.40.1.10 RemovePlayer()

```
void GameManager.RemovePlayer (
            Player player )
```

Removes the player.

Parameters

player	Player.

3.40.1.11 RespawnPlayer()

```
void GameManager.RespawnPlayer (
            int playerNumber,
            TeamId playerTeamId )
```

Respawns the player

Parameters

playerNumber	Player number

3.40.1.12 RpcRespawnPlayer()

```
void GameManager.RpcRespawnPlayer (
            int playerNumber,
            int spawnPointIndex )
```

Rpc for respawning the player

Parameters

playerNumber	Player number
spawnPointIndex	Spawn point index

3.40.1.13 ServerResetAllPlayers()

```
void GameManager.ServerResetAllPlayers ( )
```

Resets all the players on the server

3.40.1.14 StartUp()

```
void GameManager.StartUp ( )  [protected]
```

State up state function

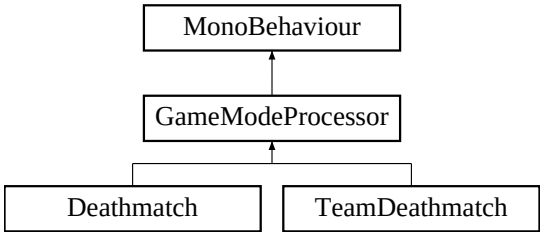
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/GameManager.cs

3.41 GameModeProcessor Class Reference

Game mode rules processor - a base class for all game modes.

Inheritance diagram for GameModeProcessor:



Public Member Functions

- virtual string GetRoundMessage ()

  Gets the round message.
- void SetGameManager (GameManager gManager)

  Sets the game manager.
- virtual bool IsEndOfRound ()

  Determines whether it is end of round.
- virtual void StartGame ()

  Called on game start
- virtual void StartRound ()

  Called on round start
- virtual void MatchEnd ()

  Called on Match end
- virtual void PlayerDies (Player player)

  Handles the death of a player
- virtual void HandleKillerScore (Player killer, Player killed)

  Handles the killer score - this differs per game mode
- virtual void HandleSuicide (Player killer)

  Handles the player's suicide - this differs per game mode
- virtual void PlayerDisconnected (Player player)

  Called when a player disconnects
- virtual void HandleRoundEnd ()

  Handles the round end.
- virtual string GetRoundEndText ()

  Gets the round end text.
- virtual string GetGameOverText ()

  Gets the game over text.
- virtual void Bail ()

  Handles bailing (i.e. leaving the game)
- virtual void CompleteGame ()

  Handles the game being complete (including the transitions)
- virtual void **RegenerateHudScoreList** ()

Static Public Attributes

- static float **endGameTransitionTime** = 10f

Protected Attributes

- MenuPage **returnPage**
- GameManager **gameManager**
- Player **winner**
- bool **isMatchOver** = false

Properties

- MenuPage **ReturnPage** `[get]`
- bool **IsMatchOver** `[get]`
- virtual int **ScoreWinTarget** `[get]`
- virtual bool **HasWinner** `[get]`

3.41.1 Detailed Description

Game mode rules processor - a base class for all game modes.

3.41.2 Member Function Documentation

3.41.2.1 Bail()

```
virtual void GameModeProcessor.Bail ( )  [virtual]
```

Handles bailing (i.e. leaving the game)

3.41.2.2 CompleteGame()

```
virtual void GameModeProcessor.CompleteGame ( )  [virtual]
```

Handles the game being complete (including the transitions)

3.41.2.3 GetGameOverText()

```
virtual string GameModeProcessor.GetGameOverText ( )  [virtual]
```

Gets the game over text.

Returns

The game over text.

Reimplemented in TeamDeathmatch, and Deathmatch.

3.41.2.4 GetRoundEndText()

```
virtual string GameModeProcessor.GetRoundEndText ( )  [virtual]
```

Gets the round end text.

Returns

The round end text.

Reimplemented in TeamDeathmatch, and Deathmatch.

3.41.2.5 GetRoundMessage()

```
virtual string GameModeProcessor.GetRoundMessage ( )  [virtual]
```

Gets the round message.

Returns

The round message.

3.41.2.6 HandleKillerScore()

```
virtual void GameModeProcessor.HandleKillerScore (
            Player killer,
            Player killed )  [virtual]
```

Handles the killer score - this differs per game mode

Parameters

killer	Player that did the killing
killed	Player that was killed

3.41.2.7 HandleRoundEnd()

```
virtual void GameModeProcessor.HandleRoundEnd ( )  [virtual]
```

Handles the round end.

Reimplemented in TeamDeathmatch, and Deathmatch.

3.41.2.8 HandleSuicide()

```
virtual void GameModeProcessor.HandleSuicide (
            Player killer )  [virtual]
```

Handles the player's suicide - this differs per game mode

Parameters

killer	The player that kill themself

3.41.2.9 IsEndOfRound()

```
virtual bool GameModeProcessor.IsEndOfRound ( )  [virtual]
```

Determines whether it is end of round.

Returns

`true` if is end of round; otherwise, `false`.

Reimplemented in TeamDeathmatch, and Deathmatch.

3.41.2.10 MatchEnd()

```
virtual void GameModeProcessor.MatchEnd ( )  [virtual]
```

Called on Match end

3.41.2.11 PlayerDies()

```
virtual void GameModeProcessor.PlayerDies (
            Player player )  [virtual]
```

Handles the death of a player

Parameters

player	Player.

Reimplemented in TeamDeathmatch, and Deathmatch.

3.41.2.12 PlayerDisconnected()

```
virtual void GameModeProcessor.PlayerDisconnected (
            Player player )  [virtual]
```

Called when a player disconnects

Parameters

player	The player that disconnects

Reimplemented in TeamDeathmatch, and Deathmatch.

3.41.2.13 SetGameManager()

```
void GameModeProcessor.SetGameManager (
            GameManager gManager )
```

Sets the game manager.

Parameters

gameManager	Game manager.

3.41.2.14 StartGame()

```
virtual void GameModeProcessor.StartGame ( )  [virtual]
```

Called on game start

Reimplemented in TeamDeathmatch.

3.41.2.15 StartRound()

```
virtual void GameModeProcessor.StartRound ( )  [virtual]
```

Called on round start

Reimplemented in TeamDeathmatch, and Deathmatch.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/GameModes/GameModeProcessor.cs

3.42 GameSettings Class Reference

Public Member Functions

- string **GetLobbySceneName** ()
- void SetMapIndex (int index)

  Sets the index of the map.
- void SetModeIndex (int index)

  Sets the index of the mode.

Properties

- MapInfo **Map** [get]
- int **MapIndex** [get]
- ModeInfo **Mode** [get]
- int **ModeIndex** [get]
- int **ScoreTarget** [get]

Events

- Action< MapInfo > **mapChanged**
- Action< ModeInfo > **modeChanged**

3.42.1 Member Function Documentation

3.42.1.1 SetMapIndex()

```
void GameSettings.SetMapIndex (
            int index )
```

Sets the index of the map.

Parameters

index	Index.

3.42.1.2 SetModeIndex()

```
void GameSettings.SetModeIndex (
            int index )
```

Sets the index of the mode.

Parameters

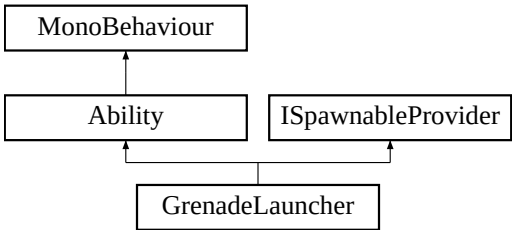
index	Index.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/GameSettings.cs

3.43 GrenadeLauncher Class Reference

Inheritance diagram for GrenadeLauncher:



Public Member Functions

- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- void Fire ()

Public Attributes

- string **animatorTrigger**
- float **spawnOffset**
- GameObject **shellPrefab**

Additional Inherited Members

3.43.1 Member Function Documentation

3.43.1.1 ButtonDown()

```
override void GrenadeLauncher.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.43.1.2 Fire()

```
void GrenadeLauncher.Fire ( )
```

3.43.1.3 SetActive()

```
override void GrenadeLauncher.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

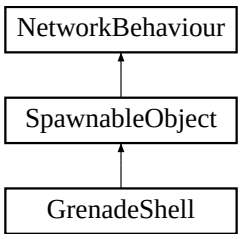
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BomberKit/GrenadeLauncher.cs

3.44 GrenadeShell Class Reference

Inheritance diagram for GrenadeShell:



Public Attributes

- float **launchForce**
- float **lifetime**
- float **explosionRadius**
- float **explosionForce**
- float **baseDamage**

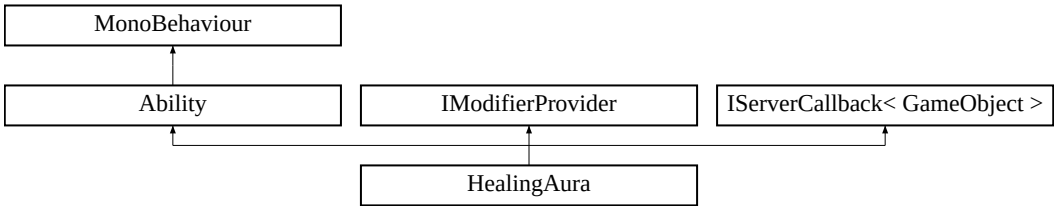
Additional Inherited Members

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BomberKit/GrenadeShell.cs

3.45 HealingAura Class Reference

Inheritance diagram for HealingAura:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- int **GetAbilityId** ()
- int **GetBuffId** ()
- IEnumerator ApplyHealingInArea (float interval)

  Applies healing buff in the area

Public Attributes

- ModifierInfo **healBuff**
- FortificationBuff **fortificationBuff**
- float **reapplyInterval**
- SpriteRenderer **visuals**

Additional Inherited Members

3.45.1 Member Function Documentation

3.45.1.1 ApplyHealingInArea()

```
IEnumerator HealingAura.ApplyHealingInArea (
            float interval )
```

Applies healing buff in the area

Parameters

interval	How often it should be applied

Returns

3.45.1.2 ButtonDown()

```
override void HealingAura.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.45.1.3 Initialize()

```
override void HealingAura.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.45.1.4 SetActive()

```
override void HealingAura.SetActive (
            bool state )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

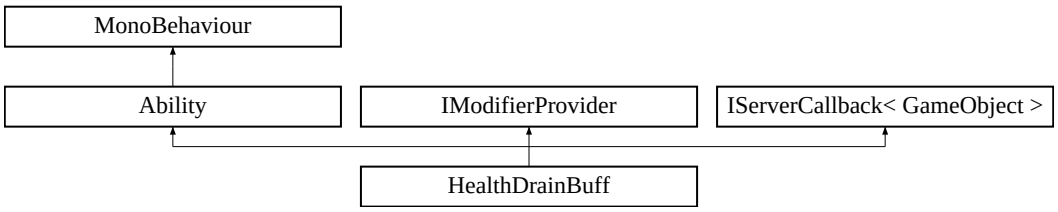
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SupportKit/HealingAura.cs

3.46 HealthDrainBuff Class Reference

Inheritance diagram for HealthDrainBuff:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state)

  Activates/deactivates collider. Cleans up lists when skill is over.
- override void SetModifier (bool state)
- void OnTriggerEnter (Collider other)

  Adds players that enters the aura in list of players in aura
- void OnTriggerExit (Collider other)

  Removes players who leave the aura from the list of players in aura
- void **ClearOnDurationEnd** ()
- IEnumerator Drain ()

  Applies the drain damage / heal to the players at a set interval.
- int **GetBuffModifierId** ()
- int **GetAbilityId** ()
- bool **IsActive** ()

Public Attributes

- float **duration**
- float **drainInterval**
- float **baseDrain**
- ModifierInfo **buff**
- ModifierInfo **debuff**
- List< GameObject > **friendlyPlayersInAura** = new List<GameObject>()
- List< GameObject > **hostilePlayersInAura** = new List<GameObject>()

Protected Member Functions

- override void Update ()

  Override to end the skill after duration is over.

Additional Inherited Members

3.46.1 Member Function Documentation

3.46.1.1 ButtonDown()

```
override void HealthDrainBuff.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.46.1.2 Drain()

```
IEnumerator HealthDrainBuff.Drain ( )
```

Applies the drain damage / heal to the players at a set interval.

Returns

3.46.1.3 Initialize()

```
override void HealthDrainBuff.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.46.1.4 OnTriggerEnter()

```
void HealthDrainBuff.OnTriggerEnter (
            Collider other )
```

Adds players that enters the aura in list of players in aura

Parameters

other	the other collider

3.46.1.5 OnTriggerExit()

```
void HealthDrainBuff.OnTriggerExit (
            Collider other )
```

Removes players who leave the aura from the list of players in aura

Parameters

other	the other collider

3.46.1.6 SetActive()

```
override void HealthDrainBuff.SetActive (
            bool state )  [virtual]
```

Activates/deactivates collider. Cleans up lists when skill is over.

Parameters

state	current state of the skill

Implements Ability.

3.46.1.7 SetModifier()

```
override void HealthDrainBuff.SetModifier (
            bool state )  [virtual]
```

Parameters

state	

Reimplemented from Ability.

3.46.1.8 Update()

```
override void HealthDrainBuff.Update ( )  [protected], [virtual]
```

Override to end the skill after duration is over.

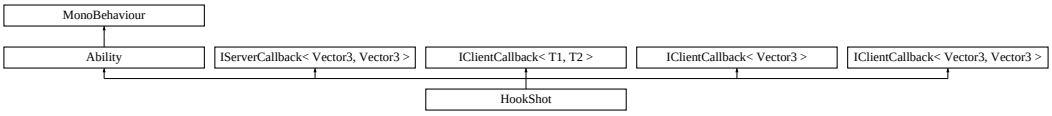
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SupportKit/HealthDrainBuff.cs

3.47 HookShot Class Reference

Inheritance diagram for HookShot:



Public Member Functions

- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- string **animatorBool**
- Transform **hookSpawnPoint**
- Collider **hook**
- LineRenderer **lineRenderer**
- float **hookSpeed** = 40f
- float **hookReturnSpeed** = 30f
- float **hookRange** = 80f
- float **hookPullForce** = 5.8f
- float **hookOnHitHoldTime** = 0.5f
- ModifierInfo **hookModifier**

Additional Inherited Members

3.47.1 Member Function Documentation

3.47.1.1 ButtonDown()

```
override void HookShot.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.47.1.2 SetActive()

```
override void HookShot.SetActive (
            bool state )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

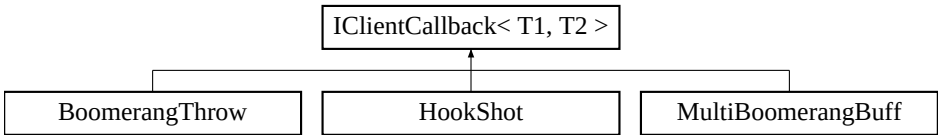
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TankKit/HookShot.cs

3.48 IClientCallback< T1, T2 > Interface Template Reference

Can recieve client callbacks from the Docking with two parameters.

Inheritance diagram for IClientCallback< T1, T2 >:



Public Member Functions

- void ClientCallback (int functionId)

  Called from the Docking to give abilities a way to run code on every client.
- void **ClientCallback** (int functionId, T param)
- void **ClientCallback** (int functionId, T1 first, T2 second)

3.48.1 Detailed Description

Can recieve client callbacks from the Docking with two parameters.

3.48.2 Member Function Documentation

3.48.2.1 ClientCallback()

```
void IClientCallback< T1, T2 >.ClientCallback (
            int functionId )
```

Called from the Docking to give abilities a way to run code on every client.

Parameters

functionId	The id of the function to be run on every client.

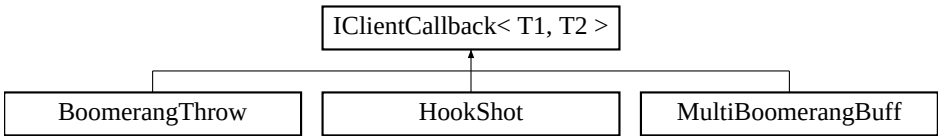
The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/IClientCallback.cs

3.49 IClientCallback< T1, T2 > Interface Template Reference

Can recieve client callbacks from the Docking with two parameters.

Inheritance diagram for IClientCallback< T1, T2 >:



Public Member Functions

- void ClientCallback (int functionId)

  Called from the Docking to give abilities a way to run code on every client.
- void **ClientCallback** (int functionId, T param)
- void **ClientCallback** (int functionId, T1 first, T2 second)

3.49.1 Detailed Description

Can recieve client callbacks from the Docking with two parameters.

3.49.2 Member Function Documentation

3.49.2.1 ClientCallback()

```
void IClientCallback< T1, T2 >.ClientCallback (
            int functionId )
```

Called from the Docking to give abilities a way to run code on every client.

Parameters

functionId	The id of the function to be run on every client.

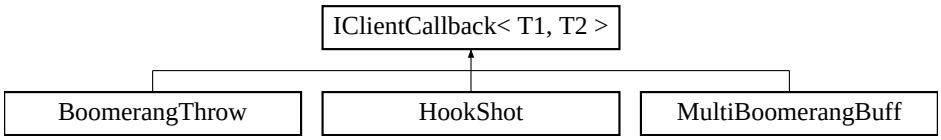
The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/IClientCallback.cs

3.50 IClientCallback< T1, T2 > Interface Template Reference

Can recieve client callbacks from the Docking with two parameters.

Inheritance diagram for IClientCallback< T1, T2 >:



Public Member Functions

- void ClientCallback (int functionId)
  Called from the Docking to give abilities a way to run code on every client.
- void **ClientCallback** (int functionId, T param)
- void **ClientCallback** (int functionId, T1 first, T2 second)

3.50.1 Detailed Description

Can recieve client callbacks from the Docking with two parameters.

3.50.2 Member Function Documentation

3.50.2.1 ClientCallback()

```
void IClientCallback< T1, T2 >.ClientCallback (
            int functionId )
```

Called from the Docking to give abilities a way to run code on every client.

Parameters

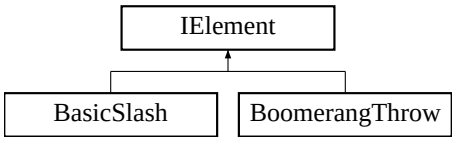
functionId	The id of the function to be run on every client.

The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/IClientCallback.cs

3.51 IElement Interface Reference

Inheritance diagram for IElement:



Public Member Functions

- void **ApplyElement** (ElementalContainer.ComboableElements element)

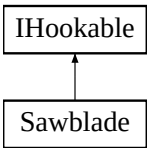
The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/IElement.cs

3.52 IHookable Interface Reference

Used by spawnables that can be hooked.

Inheritance diagram for IHookable:



Public Member Functions

- void Hooked (GameObject playerObject, Transform hook)

  Hooks the spawnable.

3.52.1 Detailed Description

Used by spawnables that can be hooked.

3.52.2 Member Function Documentation

3.52.2.1 Hooked()

```
void IHookable.Hooked (
            GameObject playerObject,
            Transform hook )
```

Hooks the spawnable.

Parameters

playerObject	The player that owns the hook.
hook	The hook transform.

Implemented in Sawblade.

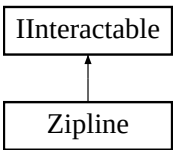
The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/IHookable.cs

3.53 IInteractable Interface Reference

Used by objects that can receive interaction calls from PlayerInput.

Inheritance diagram for IInteractable:



Public Member Functions

- void Interact (Player player)

  Called when the object is interacted with.

3.53.1 Detailed Description

Used by objects that can receive interaction calls from PlayerInput.

3.53.2 Member Function Documentation

3.53.2.1 Interact()

```
void IInteractable.Interact (
            Player player )
```

Called when the object is interacted with.

Parameters

player	Reference to the Player interacting.

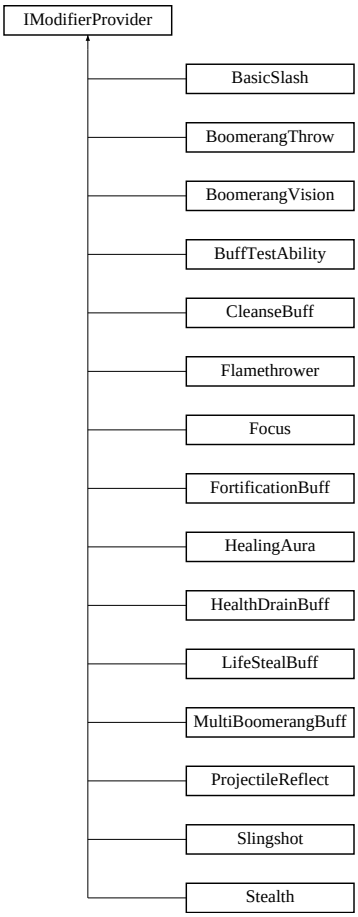
The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/IInteractable.cs

3.54 IModifierProvider Interface Reference

Can return reference to modifier info.

Inheritance diagram for IModifierProvider:



Public Member Functions

- ModifierInfo GetModifierInfo (int modifierId)

  Used by the Docking to get the correct modifier from the abilities. Parameter only used if the ability has a list of modifiers.

3.54.1 Detailed Description

Can return reference to modifier info.

3.54.2 Member Function Documentation

3.54.2.1 GetModifierInfo()

```
ModifierInfo IModifierProvider.GetModifierInfo (
            int modifierId )
```

Used by the Docking to get the correct modifier from the abilities. Parameter only used if the ability has a list of modifiers.

Parameters

modifier↩ Id	The Id of the modifier info.

Returns

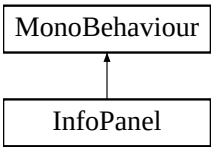
Reference to the ModifierInfo.

The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/IModifierProvider.cs

3.55 InfoPanel Class Reference

Inheritance diagram for InfoPanel:



Public Member Functions

- void **Display** (string info, UnityEngine.Events.UnityAction buttonClbk, bool displayButton=true)

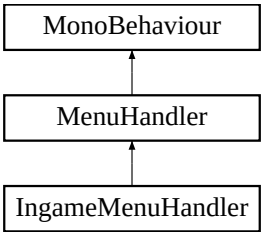
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/InfoPanel.cs

3.56 IngameMenuHandler Class Reference

Handles ingame menus like the Shop and "Pause" menu

Inheritance diagram for IngameMenuHandler:



Public Member Functions

- void **ToggleShop** ()
- void **OnGameStateChange** (bool canShopBeActivated)
- void OnShopDisplay ()

  Gets called whenever the player activates the Shop UI. Caches references to the local player if not already cached.
- void OnShopSelectionChange ()

  Handles the updating of the shop UI as different docking kits are selected
- void DisplayVerificationPrompt ()

  Displays the verification prompt for shop purchases
- void CompleteShopPurchase ()

  Completes a shop purchase and tells docking to switch kit
- void SetFirstSelectedShopObject ()

  Makes sure to set the selection of the first element in the shop as the menu is opened
- void SetLastSelectedShopObject ()

  Can be used when going back from menus like the verification prompt to set the last highlighted shop item as selected again
- void StopHost ()

  Simple function that calls the NetworkManager to disconnect from the game. Can be called from UI buttons using their OnClick interface in the editor
- void CheckPriceAndEquipAvailability ()

  Checks all shop item prices and adds a dark overlay to items that the player is unable to purchase. Also displays a "e" on the currently equipped docking kit
- void **OnLeaveGameClicked** ()

Public Attributes

- GameObject **pauseMenu**
- GameObject **shopMenu**
- GameObject **shopDescriptionsContainer**
- GameObject **shopItemPrefab**
- GameObject **purchaseVerificationPrompt**

Additional Inherited Members

3.56.1 Detailed Description

Handles ingame menus like the Shop and "Pause" menu

3.56.2 Member Function Documentation

3.56.2.1 CheckPriceAndEquipAvailability()

```
void IngameMenuHandler.CheckPriceAndEquipAvailability ( )
```

Checks all shop item prices and adds a dark overlay to items that the player is unable to purchase. Also displays a "e" on the currently equipped docking kit

3.56.2.2 CompleteShopPurchase()

```
void IngameMenuHandler.CompleteShopPurchase ( )
```

Completes a shop purchase and tells docking to switch kit

3.56.2.3 DisplayVerificationPrompt()

```
void IngameMenuHandler.DisplayVerificationPrompt ( )
```

Displays the verification prompt for shop purchases

3.56.2.4 OnShopDisplay()

```
void IngameMenuHandler.OnShopDisplay ( )
```

Gets called whenever the player activates the Shop UI. Caches references to the local player if not already cached.

3.56.2.5 OnShopSelectionChange()

```
void IngameMenuHandler.OnShopSelectionChange ( )
```

Handles the updating of the shop UI as different docking kits are selected

3.56.2.6 SetFirstSelectedShopObject()

```
void IngameMenuHandler.SetFirstSelectedShopObject ( )
```

Makes sure to set the selection of the first element in the shop as the menu is opened

3.56.2.7 SetLastSelectedShopObject()

```
void IngameMenuHandler.SetLastSelectedShopObject ( )
```

Can be used when going back from menus like the verification prompt to set the last highlighted shop item as selected again

3.56.2.8 StopHost()

```
void IngameMenuHandler.StopHost ( )
```

Simple function that calls the NetworkManager to disconnect from the game. Can be called from UI buttons using their OnClick interface in the editor

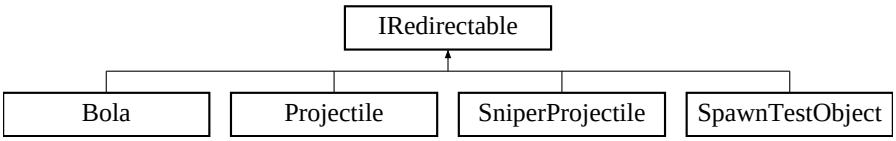
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/IngameMenuHandler.cs

3.57 IRedirectable Interface Reference

Used by spawnables that can be redirected.

Inheritance diagram for IRedirectable:



Public Member Functions

- void RedirectDirection (Vector3 newDirection, int newPlayerId=-1, TeamId newTeamId=TeamId.Unassigned)
  Redirects direction of the spawnable.

3.57.1 Detailed Description

Used by spawnables that can be redirected.

3.57.2 Member Function Documentation

3.57.2.1 RedirectDirection()

```
void IRedirectable.RedirectDirection (
            Vector3 newDirection,
            int newPlayerId = -1,
            TeamId newTeamId = TeamId.Unassigned )
```

Redirects direction of the spawnable.

Parameters

newDirection	The new direction.
newPlayerId	The player id of the new owner, -1 if current owner is kept.
newTeamId	The team id of the new owner, TeamId.Unassigned if current owner is kept.

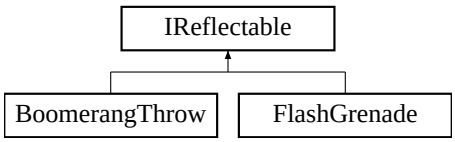
Implemented in SpawnTestObject.

The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/IRedirectable.cs

3.58 IReflectable Interface Reference

Inheritance diagram for IReflectable:



Public Member Functions

- void **ReflectVelocity** ()

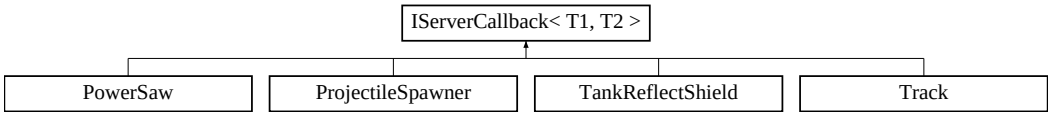
The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/IReflectable.cs

3.59 IServerCallback< T1, T2 > Interface Template Reference

Can recieve server callbacks from the Docking with two parameters.

Inheritance diagram for IServerCallback< T1, T2 >:



Public Member Functions

- void ServerCallback (int functionId)

  Called from the Docking to give abilities a way to run server code.
- void **ServerCallback** (int functionId, T param)
- void **ServerCallback** (int functionId, T1 first, T2 second)

3.59.1 Detailed Description

Can recieve server callbacks from the Docking with two parameters.

3.59.2 Member Function Documentation

3.59.2.1 ServerCallback()

```
void IServerCallback< T1, T2 >.ServerCallback (
            int functionId )
```

Called from the Docking to give abilities a way to run server code.

Parameters

functionId	The id of the function to be run on the server.

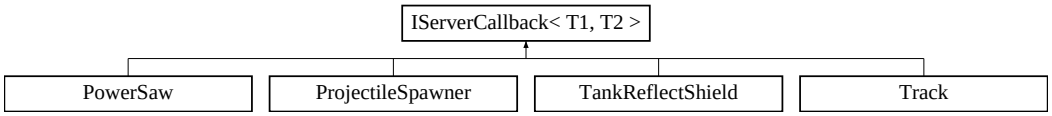
The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/IServerCallback.cs

3.60 IServerCallback< T1, T2 > Interface Template Reference

Can recieve server callbacks from the Docking with two parameters.

Inheritance diagram for IServerCallback< T1, T2 >:



Public Member Functions

- void ServerCallback (int functionId)

  Called from the Docking to give abilities a way to run server code.
- void **ServerCallback** (int functionId, T param)
- void **ServerCallback** (int functionId, T1 first, T2 second)

3.60.1 Detailed Description

Can recieve server callbacks from the Docking with two parameters.

3.60.2 Member Function Documentation

3.60.2.1 ServerCallback()

```
void IServerCallback< T1, T2 >.ServerCallback (
            int functionId )
```

Called from the Docking to give abilities a way to run server code.

Parameters

functionId	The id of the function to be run on the server.

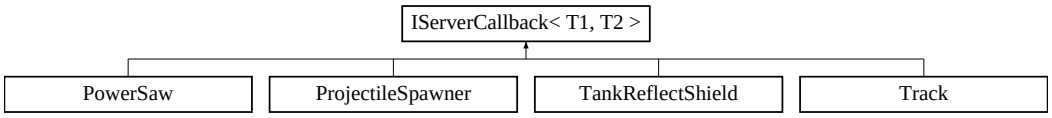
The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/IServerCallback.cs

3.61 IServerCallback< T1, T2 > Interface Template Reference

Can recieve server callbacks from the Docking with two parameters.

Inheritance diagram for IServerCallback< T1, T2 >:



Public Member Functions

- void ServerCallback (int functionId)

  Called from the Docking to give abilities a way to run server code.
- void **ServerCallback** (int functionId, T param)
- void **ServerCallback** (int functionId, T1 first, T2 second)

3.61.1 Detailed Description

Can recieve server callbacks from the Docking with two parameters.

3.61.2 Member Function Documentation

3.61.2.1 ServerCallback()

```
void IServerCallback< T1, T2 >.ServerCallback (
            int functionId )
```

Called from the Docking to give abilities a way to run server code.

Parameters

functionId	The id of the function to be run on the server.

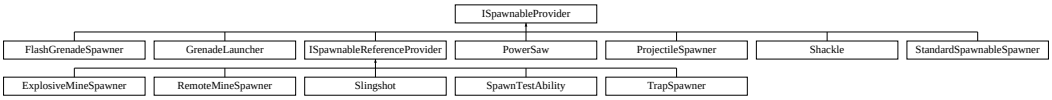
The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/IServerCallback.cs

3.62 ISpawnableProvider Interface Reference

Can return reference to a spawnable prefab.

Inheritance diagram for ISpawnableProvider:



Public Member Functions

- GameObject GetSpawnablePrefab (int spawnableId)

  Used by the Docking to get the correct prefab to spawn from the abilities. Parameter only used if the ability has a list of prefabs.

3.62.1 Detailed Description

Can return reference to a spawnable prefab.

3.62.2 Member Function Documentation

3.62.2.1 GetSpawnablePrefab()

```
GameObject ISpawnableProvider.GetSpawnablePrefab (
            int spawnableId )
```

Used by the Docking to get the correct prefab to spawn from the abilities. Parameter only used if the ability has a list of prefabs.

Parameters

spawnableId	The Id of the spawnable object.

Returns

Reference to the prefab GameObject.

Implemented in StandardSpawnableSpawner.

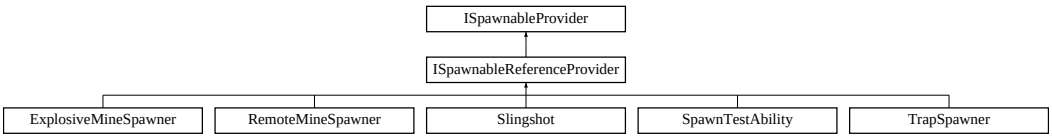
The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/ISpawnableProvider.cs

3.63 ISpawnableReferenceProvider Interface Reference

Can return reference to a spawnable prefab and catch the reference to the spawned object.

Inheritance diagram for ISpawnableReferenceProvider:



Public Member Functions

- void SetSpawnedObjectReference (GameObject spawnedObject)
  Called from the Docking to set up local references from spawned network objects.

3.63.1 Detailed Description

Can return reference to a spawnable prefab and catch the reference to the spawned object.

3.63.2 Member Function Documentation

3.63.2.1 SetSpawnedObjectReference()

```
void ISpawnableReferenceProvider.SetSpawnedObjectReference (
            GameObject spawnedObject )
```

Called from the Docking to set up local references from spawned network objects.

Parameters

spawnedObject	Reference to spawned object.

The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/ISpawnableProvider.cs

3.64 ITargetClientCallback< T > Interface Template Reference

Can recieve target client callbacks from the Docking with one parameter.

3.64.1 Detailed Description

Can recieve target client callbacks from the Docking with one parameter.

The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/ITargetClientCallback.cs

3.65 ITargetClientCallback< T > Interface Template Reference

Can recieve target client callbacks from the Docking with one parameter.

3.65.1 Detailed Description

Can recieve target client callbacks from the Docking with one parameter.

The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/ITargetClientCallback.cs

3.66 ITargetClientCallback< T > Interface Template Reference

Can recieve target client callbacks from the Docking with one parameter.

3.66.1 Detailed Description

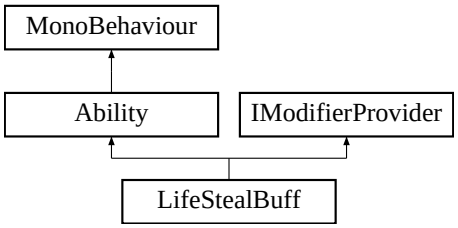
Can recieve target client callbacks from the Docking with one parameter.

The documentation for this interface was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Interfaces/Ability/ITargetClientCallback.cs

3.67 LifeStealBuff Class Reference

Inheritance diagram for LifeStealBuff:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Callback for what this ability should do once its associated button has been pressed
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- override void SetModifier (bool state=false)

  Callback for what this ability should do when a new modifier state is set
- bool IsBuffActive ()

  A simple getter function for whether the life steal buff is currently active
- int **GetAbilityId** ()
- int **GetBuffModifierId** ()

Public Attributes

- float **damageMultiplier** = 1.5f
- float **healPercentage** = 0.5f
- SpriteRenderer [ ] **axeVisuals**
- ParticleSystem **activeParticles**
- Color **axeColorWhileActive**
- ModifierInfo **buff**

Additional Inherited Members

3.67.1 Member Function Documentation

3.67.1.1 ButtonDown()

```
override void LifeStealBuff.ButtonDown ( )  [virtual]
```

Callback for what this ability should do once its associated button has been pressed

Implements Ability.

3.67.1.2 Initialize()

```
override void LifeStealBuff.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.67.1.3 IsBuffActive()

```
bool LifeStealBuff.IsBuffActive ( )
```

A simple getter function for whether the life steal buff is currently active

Returns

Whether the buff is currently active

3.67.1.4 SetActive()

```
override void LifeStealBuff.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

3.67.1.5 SetModifier()

```
override void LifeStealBuff.SetModifier (
            bool state = false )  [virtual]
```

Callback for what this ability should do when a new modifier state is set

Parameters

state	The modifier state

Reimplemented from Ability.

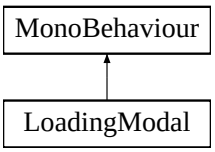
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BrawlerKit/LifeStealBuff.cs

3.68 LoadingModal Class Reference

Loading modal - used to handle loading fades

Inheritance diagram for LoadingModal:



Public Member Functions

- void FadeIn ()

  Wraps fade in on FadingGroup
- void FadeOut ()

  Wraps fade out on FadingGroup
- void **CloseModal** ()
- void **Show** ()

Properties

- static LoadingModal **Instance** `[get]`
- bool **readyToTransition** `[get]`
- FadingGroup Fader `[get]`

  Getter for Fader - used in game manager

3.68.1 Detailed Description

Loading modal - used to handle loading fades

3.68.2 Member Function Documentation

3.68.2.1 FadeIn()

```
void LoadingModal.FadeIn ( )
```

Wraps fade in on FadingGroup

3.68.2.2 FadeOut()

```
void LoadingModal.FadeOut ( )
```

Wraps fade out on FadingGroup

3.68.3 Property Documentation

3.68.3.1 Fader

```
FadingGroup LoadingModal.Fader  [get]
```

Getter for Fader - used in game manager

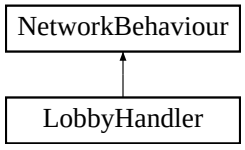
The fader.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/LoadingModal.cs

3.69 LobbyHandler Class Reference

Inheritance diagram for LobbyHandler:



Public Member Functions

- int GetPlayerCount ()

  Gets the amount of connected players to the lobby
- void AddPlayer (DLNetworkLobbyPlayer player)

  Adds a player to the connectedPlayers list and then calls DecideEntryTeam(player)
- void SetPlayerTeam (DLNetworkLobbyPlayer player)

  Adds the player to the correct team list and puts sets the parent of the player's visuals to the correct team panel
- void DisplayLobby ()

  Displays the lobby on the client and hides the "please wait while connecting" text
- void ResetLocalLobby ()

  Does the opposite of DisplayLobby()
- void RemovePlayer (DLNetworkLobbyPlayer player)

  Removes a disconnecting player from the correct team and destroys the visuals of that player.
- List< DLNetworkLobbyPlayer > GetConnectedPlayers ()

  Returns a list of connected players

Public Attributes

- RectTransform **redTeamPanel**
- RectTransform **blueTeamPanel**
- GameObject **waitingScreenObj**

3.69.1 Member Function Documentation

3.69.1.1 AddPlayer()

```
void LobbyHandler.AddPlayer (
            DLNetworkLobbyPlayer player )
```

Adds a player to the connectedPlayers list and then calls DecideEntryTeam(player)

Parameters

player	The player that we are adding

3.69.1.2 DisplayLobby()

```
void LobbyHandler.DisplayLobby ( )
```

Displays the lobby on the client and hides the "please wait while connecting" text

3.69.1.3 GetConnectedPlayers()

```
List<DLNetworkLobbyPlayer> LobbyHandler.GetConnectedPlayers ( )
```

Returns a list of connected players

Returns

A list of connected players

3.69.1.4 GetPlayerCount()

```
int LobbyHandler.GetPlayerCount ( )
```

Gets the amount of connected players to the lobby

Returns

The number of connected players

3.69.1.5 RemovePlayer()

```
void LobbyHandler.RemovePlayer (
            DLNetworkLobbyPlayer player )
```

Removes a disconnecting player from the correct team and destroys the visuals of that player.

Parameters

player	The player that just disconnected

3.69.1.6 ResetLocalLobby()

```
void LobbyHandler.ResetLocalLobby ( )
```

Does the opposite of DisplayLobby()

3.69.1.7 SetPlayerTeam()

```
void LobbyHandler.SetPlayerTeam (
            DLNetworkLobbyPlayer player )
```

Adds the player to the correct team list and puts sets the parent of the player's visuals to the correct team panel

Parameters

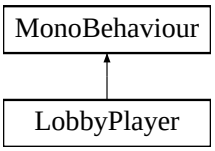
player	The player that we are adding

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/LobbyHandler.cs

3.70 LobbyPlayer Class Reference

Inheritance diagram for LobbyPlayer:



Public Member Functions

- void **Init** (NetworkPlayer netPlayer)
- void **RefreshJoinButton** ()
- void **OnTeamClicked** ()
- void **OnReadyClicked** ()
- void **OnNameChanged** (string str)

Protected Member Functions

- virtual void **PlayerJoined** (NetworkPlayer player)
- virtual void **PlayerLeft** (NetworkPlayer player)
- virtual void **OnDestroy** ()

Protected Attributes

- InputField **nameInput**
- Button **readyButton**
- Transform **waitingLabel**
- Transform **readyLabel**
- Button **teamButton**
- Text **teamButtonText**

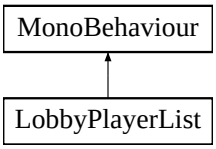
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/LobbyPlayer.cs

3.71 LobbyPlayerList Class Reference

Handles the player list in the Lobby.

Inheritance diagram for LobbyPlayerList:



Public Member Functions

- void **AddPlayer** (LobbyPlayer player, TeamId teamId)
- void **OnBackClick** ()

Protected Member Functions

- virtual void Start ()

  Subscribe to events on start
- virtual void OnDestroy ()

  Unsubscribe to events on destroy
- virtual void **PlayerJoined** (NetworkPlayer player)
- virtual void **PlayerLeft** (NetworkPlayer player)
- virtual void **PlayersReadied** ()

3.71.1 Detailed Description

Handles the player list in the Lobby.

3.71.2 Member Function Documentation

3.71.2.1 OnDestroy()

```
virtual void LobbyPlayerList.OnDestroy ( )  [protected], [virtual]
```

Unsubscribe to events on destroy

3.71.2.2 Start()

```
virtual void LobbyPlayerList.Start ( )  [protected], [virtual]
```

Subscribe to events on start

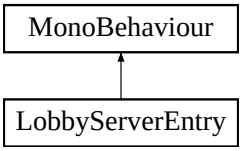
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/LobbyPlayerList.cs

3.72 LobbyServerEntry Class Reference

Represents a server in the server list

Inheritance diagram for LobbyServerEntry:



Public Member Functions

- void **Populate** (MatchInfoSnapshot match, Color c)

Protected Member Functions

- virtual void **OnEnable** ()

Protected Attributes

- Text **serverInfoText**
- Text **modeText**
- Text **slotInfo**
- Button **joinButton**
- NetworkManager **networkManager**

3.72.1 Detailed Description

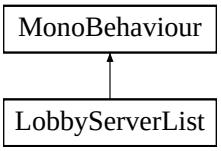
Represents a server in the server list

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/LobbyServerEntry.cs

3.73 LobbyServerList Class Reference

Inheritance diagram for LobbyServerList:



Public Member Functions

- void **OnBackClick** ()
- void **OnGuiMatchList** (bool flag, string extraInfo, List< MatchInfoSnapshot > response)
- void **ChangePage** (int dir)
- void **RequestPage** (int page)
- void **RefreshList** ()

Protected Member Functions

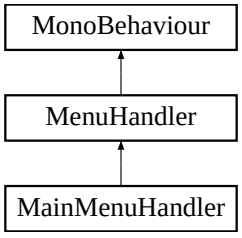
- virtual void **OnEnable** ()
- void **ClearUi** ()
- virtual void **OnDisable** ()
- virtual void **OnError** (UnityEngine.Networking.NetworkConnection conn, int errorCode)
- virtual void **OnDisconnect** (UnityEngine.Networking.NetworkConnection conn)
- virtual void **OnDrop** ()
- virtual void **Update** ()

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/LobbyServerList.cs

3.74 MainMenuHandler Class Reference

Inheritance diagram for MainMenuHandler:



Public Member Functions

- void NavigateTo (GameObject nextMenu)

  Navigates to a given menu gameObject and places the current one in the stack
- void NavigateBack ()

  Pops all menus from back stack until it hits a stopPop menu and navigates to that.
- void AddPropertyToStackTop (int enumId)

  Adds a property to the previous menu that is in the stack. This is mostly used as a workaround to the fact that the Unity Inspector's OnClick interface only supports none/single parameter functions
- void CreateOnlineMatch ()

  Uses the Unity match maker to create a new online match
- void StartMatchMaker ()

  Starts the Unity match maker

Public Attributes

- Text **hostRoomNameText**
- LobbyHandler **lobbyHandler**

Additional Inherited Members

3.74.1 Member Function Documentation

3.74.1.1 AddPropertyToStackTop()

```
void MainMenuHandler.AddPropertyToStackTop (
            int enumId )
```

Adds a property to the previous menu that is in the stack. This is mostly used as a workaround to the fact that the Unity Inspector's OnClick interface only supports none/single parameter functions

Parameters

enumId	The id of the property we are adding

3.74.1.2 CreateOnlineMatch()

```
void MainMenuHandler.CreateOnlineMatch ( )
```

Uses the Unity match maker to create a new online match

3.74.1.3 NavigateBack()

```
void MainMenuHandler.NavigateBack ( )
```

Pops all menus from back stack until it hits a stopPop menu and navigates to that.

3.74.1.4 NavigateTo()

```
void MainMenuHandler.NavigateTo (
            GameObject nextMenu )
```

Navigates to a given menu gameObject and places the current one in the stack

Parameters

nextMenu	The menu we are navigating to

3.74.1.5 StartMatchMaker()

```
void MainMenuHandler.StartMatchMaker ( )
```

Starts the Unity match maker

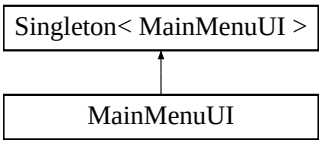
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/MainMenuHandler.cs

3.75 MainMenuUI Class Reference

Handles main menu UI and transitions

Inheritance diagram for MainMenuUI:



Public Member Functions

- void **ShowPanel** (CanvasGroup newPanel)
- void **ShowDefaultPanel** ()
- void **ShowLobbyPanel** ()
- void **ShowLobbyPanelForConnection** ()
- void **ShowServerListPanel** ()
- void ShowInfoPopup (string label, UnityEngine.Events.UnityAction callback)

  Shows the info popup with a callback
- void **ShowInfoPopup** (string label)
- void **ShowConnectingModal** (bool reconnectMatchmakingClient)
- void **HideInfoPopup** ()
- void DoIfNetworkReady (Action task)

  Wait for network to disconnect before performing an action
- void **OnCreateGameClicked** ()
- void **OnFindGameClicked** ()
- void **OnQuitGameClicked** ()

Static Public Attributes

- static MenuPage **ReturnPage**

Protected Member Functions

- virtual void **Update** ()
- virtual void **Start** ()

Properties

- LobbyPlayerList **PlayerList** `[get]`

3.75.1 Detailed Description

Handles main menu UI and transitions

3.75.2 Member Function Documentation

3.75.2.1 DoIfNetworkReady()

```
void MainMenuUI.DoIfNetworkReady (
            Action task )
```

Wait for network to disconnect before performing an action

3.75.2.2 ShowInfoPopup()

```
void MainMenuUI.ShowInfoPopup (
            string label,
            UnityEngine.Events.UnityAction callback )
```

Shows the info popup with a callback

Parameters

label	Label.
callback	Callback.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/MainMenuUI.cs

3.76 MapInfo Class Reference

Public Member Functions

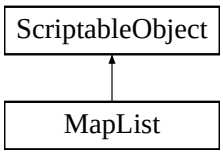
- string **GetName** ()
- string **GetDescription** ()
- string **GetSceneName** ()
- Sprite **GetMapImage** ()

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Map/MapInfo.cs

3.77 MapList Class Reference

Inheritance diagram for MapList:



Properties

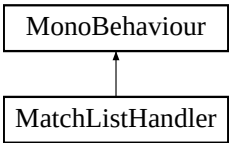
- MapInfo **this[int index]** `[get]`
- int **Count** `[get]`

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Map/MapList.cs

3.78 MatchListHandler Class Reference

Inheritance diagram for MatchListHandler:



Public Member Functions

- void OnMatchButtonClick (int buttonNumber, UnityEngine.Networking.Match.MatchInfoSnapshot match)

  A button listener callback that makes the client join the match that has been selected. Also navigates to the lobby menu screen.

Public Attributes

- GameObject **dynamicMatchButtonPrefab**
- int **matchButtonOffset** = 90
- MainMenuHandler **mainMenuHandler**
- GameObject **lobbyObj**
- GameObject **lobbyVerfifPromptObj**
- GameObject **noMatchesFoundObj**

3.78.1 Member Function Documentation

3.78.1.1 OnMatchButtonClick()

```
void MatchListHandler.OnMatchButtonClick (
            int buttonNumber,
            UnityEngine.Networking.Match.MatchInfoSnapshot match )
```

A button listener callback that makes the client join the match that has been selected. Also navigates to the lobby menu screen.

Parameters

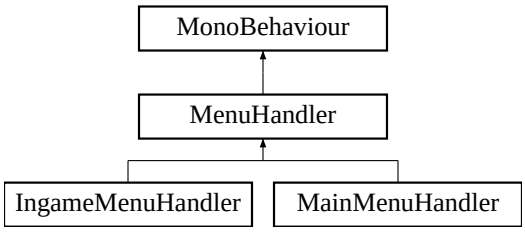
buttonNumber	The index of the button
match	The match maker match

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/MatchListHandler.cs

3.79 MenuHandler Class Reference

Inheritance diagram for MenuHandler:



Public Member Functions

- void SetCurrentMenuVerificationPrompt (GameObject verifPrompt)

  Takes a verification prompt as parameter and connects it to the current menu.
- void OnClickSetFirstSelected ()

  Allows OnClick interfaces to use SetFirstSelectedGameObject(). Useful when a menu has submenus or verification prompts and you need to return control to the user after using these
- IEnumerator SetFirstSelectedGameObject (GameObject specific)

  Sets a button as selected the next frame after it has been called. If null is passed it sets the first selected button it finds. If a specific gameObject is passed it will look for buttons on that one instead

Public Attributes

- MenuStackComponent **currentActiveMenu**
- GameObject **menuRoot**

Protected Member Functions

- void **Start** ()

3.79.1 Member Function Documentation

3.79.1.1 OnClickSetFirstSelected()

```
void MenuHandler.OnClickSetFirstSelected ( )
```

Allows OnClick interfaces to use SetFirstSelectedGameObject(). Useful when a menu has submenus or verification prompts and you need to return control to the user after using these

3.79.1.2 SetCurrentMenuVerificationPrompt()

```
void MenuHandler.SetCurrentMenuVerificationPrompt (
            GameObject verifPrompt )
```

Takes a verification prompt as parameter and connects it to the current menu.

Parameters

verifPrompt	The gameObject of the verification prompt

3.79.1.3 SetFirstSelectedGameObject()

```
IEnumerator MenuHandler.SetFirstSelectedGameObject (
            GameObject specific )
```

Sets a button as selected the next frame after it has been called. If null is passed it sets the first selected button it finds. If a specific gameObject is passed it will look for buttons on that one instead

Parameters

specific	A gameObject containing buttons

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/MenuHandler.cs

3.80 MenuStackComponent Class Reference

Public Member Functions

- **MenuStackComponent** (GameObject obj, menuStackProperty prop, bool hasVerPrompt, GameObject ver↩ PromptObj)

Public Attributes

- GameObject **menuObject**
- menuStackProperty **property**
- bool **currentMenuHasVerificationPrompt**
- GameObject **verificationPromptObj**

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/MenuHandler.cs

3.81 ModeInfo Class Reference

Public Member Functions

- **ModeInfo** (string name, string description)
- **ModeInfo** (string name, string description, GameModeProcessor processor)
- string **GetModeName** ()
- string **GetAbbreviation** ()
- string **GetDescription** ()
- GameModeProcessor **GetModeProcessor** ()
- bool **IsTeamMode** ()
- int **GetMinimumPlayers** ()

Properties

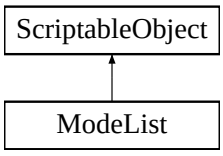
- int **Index** `[get, set]`

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/GameModes/ModeInfo.cs

3.82 ModeList Class Reference

Inheritance diagram for ModeList:



Properties

- ModeInfo **this[int index]** `[get]`
- int **Count** `[get]`

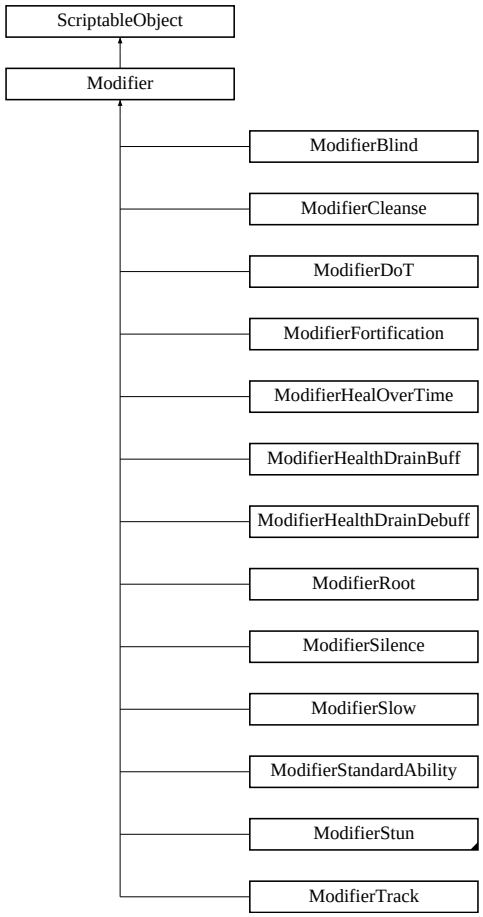
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/GameModes/ModeList.cs

3.83 Modifier Class Reference

Base class for every modifier.

Inheritance diagram for Modifier:



Public Member Functions

- virtual void OnServerStart (PlayerStatus playerStatus, int abilityId)

  Called on the server when the modifiers starts.
- virtual void OnClientStart (PlayerStatus playerStatus, int abilityId)

  Called on every client when the modifiers starts.
- virtual void OnLocalClientStart (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers starts.
- virtual void OnServerEnd (PlayerStatus playerStatus, int abilityId)

  Called on the server when the modifiers ends.
- virtual void OnClientEnd (PlayerStatus playerStatus, int abilityId)

  Called on every client when the modifiers ends.
- virtual void OnLocalClientEnd (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers ends.
- virtual void OnServerTick (PlayerStatus playerStatus)

  Called on the server whenever the modifier applies a tick.

Static Public Member Functions

- static Modifier GetModifierAsset (string modifierName)

  Looks up the Modifier with modifierName from the Resource/PlayerModifiers folder.

Public Attributes

- string **modifierName**
- Sprite **icon**
- GameObject **playerEffectObject**
- GameObject **localPlayerEffectObject**
- bool **unique**
- StatusType **statusType**

3.83.1 Detailed Description

Base class for every modifier.

3.83.2 Member Function Documentation

3.83.2.1 GetModifierAsset()

```
static Modifier Modifier.GetModifierAsset (
            string modifierName )  [static]
```

Looks up the Modifier with modifierName from the Resource/PlayerModifiers folder.

Parameters

modifierName	The modifier file name.

Returns

The modifier at path if found, otherwise null.

3.83.2.2 OnClientEnd()

```
virtual void Modifier.OnClientEnd (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on every client when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

Reimplemented in ModifierHealOverTime, ModifierBlind, and ModifierStandardAbility.

3.83.2.3 OnClientStart()

```
virtual void Modifier.OnClientStart (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on every client when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

Reimplemented in ModifierHealOverTime, ModifierBlind, and ModifierStandardAbility.

3.83.2.4 OnLocalClientEnd()

```
virtual void Modifier.OnLocalClientEnd (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented in ModifierFortification, ModifierSlow, ModifierHealthDrainDebuff, ModifierTrack, ModifierHealth↩DrainBuff, ModifierSilence, ModifierStun, and ModifierRoot.

3.83.2.5 OnLocalClientStart()

```
virtual void Modifier.OnLocalClientStart (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented in ModifierFlashStun, ModifierFortification, ModifierSlow, ModifierHealthDrainDebuff, ModifierHealthDrainBuff, ModifierTrack, ModifierRoot, ModifierSilence, and ModifierStun.

3.83.2.6 OnServerEnd()

```
virtual void Modifier.OnServerEnd (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on the server when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

Reimplemented in ModifierCleanse.

3.83.2.7 OnServerStart()

```
virtual void Modifier.OnServerStart (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on the server when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

Reimplemented in ModifierCleanse.

3.83.2.8 OnServerTick()

```
virtual void Modifier.OnServerTick (
            PlayerStatus playerStatus )  [virtual]
```

Called on the server whenever the modifier applies a tick.

Parameters

playerStatus	Reference to the associated PlayerStatus.

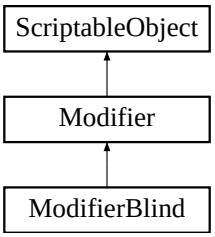
Reimplemented in ModifierHealOverTime, and ModifierDoT.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/Modifier.cs

3.84 ModifierBlind Class Reference

Inheritance diagram for ModifierBlind:



Public Member Functions

- override void OnClientStart (PlayerStatus playerStatus, int abilityId)
  Called on every client when the modifiers starts.
- override void OnClientEnd (PlayerStatus playerStatus, int abilityId)
  Called on every client when the modifiers ends.

Public Attributes

- float **blindLerpSpeed** = 10

Additional Inherited Members

3.84.1 Member Function Documentation

3.84.1.1 OnClientEnd()

```
override void ModifierBlind.OnClientEnd (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on every client when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

Reimplemented from Modifier.

3.84.1.2 OnClientStart()

```
override void ModifierBlind.OnClientStart (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on every client when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

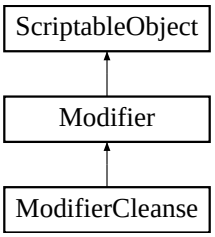
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierBlind.cs

3.85 ModifierCleanse Class Reference

Inheritance diagram for ModifierCleanse:



Public Member Functions

- override void OnServerStart (PlayerStatus playerStatus, int abilityId)

  Called on the server when the modifiers starts.
- override void OnServerEnd (PlayerStatus playerStatus, int abilityId)

  Called on the server when the modifiers ends.

Public Attributes

- float **movespeedMulitplier**

Additional Inherited Members

3.85.1 Member Function Documentation

3.85.1.1 OnServerEnd()

```
override void ModifierCleanse.OnServerEnd (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on the server when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

Reimplemented from Modifier.

3.85.1.2 OnServerStart()

```
override void ModifierCleanse.OnServerStart (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on the server when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

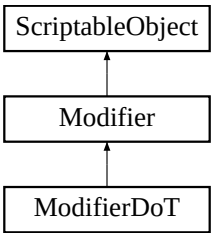
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierCleanse.cs

3.86 ModifierDoT Class Reference

Inheritance diagram for ModifierDoT:



Public Member Functions

- override void OnServerTick (PlayerStatus playerStatus)

  Called on the server whenever the modifier applies a tick.

Public Attributes

- float **damagePerTick** = 5f

Additional Inherited Members

3.86.1 Member Function Documentation

3.86.1.1 OnServerTick()

```
override void ModifierDoT.OnServerTick (
            PlayerStatus playerStatus )  [virtual]
```

Called on the server whenever the modifier applies a tick.

Parameters

playerStatus	Reference to the associated PlayerStatus.

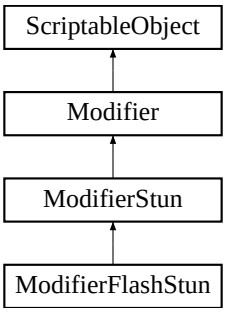
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierDoT.cs

3.87 ModifierFlashStun Class Reference

Inheritance diagram for ModifierFlashStun:



Public Member Functions

- override void OnLocalClientStart (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers starts.

Public Attributes

- GameObject **flashPrefab**

Additional Inherited Members

3.87.1 Member Function Documentation

3.87.1.1 OnLocalClientStart()

```
override void ModifierFlashStun.OnLocalClientStart (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.

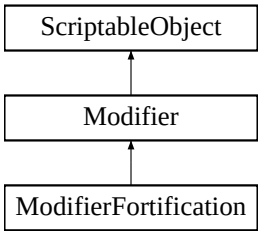
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierFlashStun.cs

3.88 ModifierFortification Class Reference

Inheritance diagram for ModifierFortification:



Public Member Functions

- override void OnLocalClientStart (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers starts.
- override void OnLocalClientEnd (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers ends.

Public Attributes

- float **damageMultiplier**

Additional Inherited Members

3.88.1 Member Function Documentation

3.88.1.1 OnLocalClientEnd()

```
override void ModifierFortification.OnLocalClientEnd (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented from Modifier.

3.88.1.2 OnLocalClientStart()

```
override void ModifierFortification.OnLocalClientStart (
            PlayerStatus playerStatus ) [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.

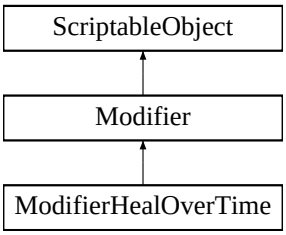
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierFortification.cs

3.89 ModifierHealOverTime Class Reference

Inheritance diagram for ModifierHealOverTime:



Public Member Functions

- override void OnClientStart (PlayerStatus playerStatus, int abilityId)
  Called on every client when the modifiers starts.
- override void OnServerTick (PlayerStatus playerStatus)
  Called on the server whenever the modifier applies a tick.
- override void OnClientEnd (PlayerStatus playerStatus, int abilityId)
  Called on every client when the modifiers ends.

Public Attributes

- float **healthPerTick**

Additional Inherited Members

3.89.1 Member Function Documentation

3.89.1.1 OnClientEnd()

```
override void ModifierHealOverTime.OnClientEnd (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on every client when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

Reimplemented from Modifier.

3.89.1.2 OnClientStart()

```
override void ModifierHealOverTime.OnClientStart (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on every client when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

Reimplemented from Modifier.

3.89.1.3 OnServerTick()

```
override void ModifierHealOverTime.OnServerTick (
            PlayerStatus playerStatus )  [virtual]
```

Called on the server whenever the modifier applies a tick.

Parameters

playerStatus	Reference to the associated PlayerStatus.

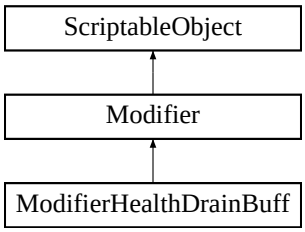
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierHealOverTime.cs

3.90 ModifierHealthDrainBuff Class Reference

Inheritance diagram for ModifierHealthDrainBuff:



Public Member Functions

- override void OnLocalClientStart (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers starts.
- override void OnLocalClientEnd (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers ends.

Additional Inherited Members

3.90.1 Member Function Documentation

3.90.1.1 OnLocalClientEnd()

```
override void ModifierHealthDrainBuff.OnLocalClientEnd (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented from Modifier.

3.90.1.2 OnLocalClientStart()

```
override void ModifierHealthDrainBuff.OnLocalClientStart (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.

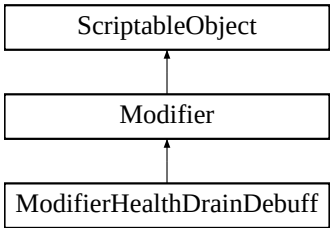
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierHealthDrainBuff.cs

3.91 ModifierHealthDrainDebuff Class Reference

Inheritance diagram for ModifierHealthDrainDebuff:



Public Member Functions

- override void OnLocalClientStart (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers starts.
- override void OnLocalClientEnd (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers ends.

Additional Inherited Members

3.91.1 Member Function Documentation

3.91.1.1 OnLocalClientEnd()

```
override void ModifierHealthDrainDebuff.OnLocalClientEnd (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented from Modifier.

3.91.1.2 OnLocalClientStart()

```
override void ModifierHealthDrainDebuff.OnLocalClientStart (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierHealthDrainDebuff.cs

3.92 ModifierInfo Struct Reference

Struct used in abilities to store modifier information.

Public Attributes

- Modifier **modifier**
- float **duration**
- int **tickCount**
- float **tickInterval**

3.92.1 Detailed Description

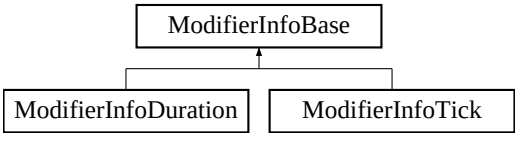
Struct used in abilities to store modifier information.

The documentation for this struct was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/Modifier.cs

3.93 ModifierInfoBase Class Reference

Inheritance diagram for ModifierInfoBase:



Public Attributes

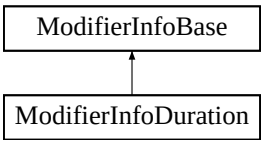
- Modifier **modifier**

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/Modifier.cs

3.94 ModifierInfoDuration Class Reference

Inheritance diagram for ModifierInfoDuration:



Public Attributes

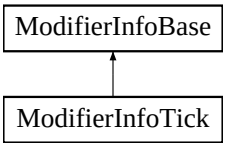
- float **duration**

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/Modifier.cs

3.95 ModifierInfoTick Class Reference

Inheritance diagram for ModifierInfoTick:



Public Attributes

- int **tickCount**
- float **tickInterval**

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/Modifier.cs

3.96 ModifierInstanceClient Class Reference

The instance used when a modifier is active. Only exists on the clients.

Public Member Functions

- ModifierInstanceClient (Modifier mod, PlayerStatus plStatus, PlayerUIHandler playerUIHandler, int modId, int abId, float duration)

  Constructor that instantiates effect objects and calls the correct modifier functions.
- void OnEnd ()

  Called when the modifier effect has ended.
- void SetNewDuration (float newDuration)

  Updates the UI elements with the new duration.
- Modifier GetModifier ()

  Returns the modifier used by this instance.
- int GetAbilityId ()

  Returns the ability ID in this instance, equal to or above 0 means this modifier was applied by an ability, -1 otherwise.
- int GetModifierId ()

  Returns the unique modifier ID for this instance.

3.96.1 Detailed Description

The instance used when a modifier is active. Only exists on the clients.

3.96.2 Constructor & Destructor Documentation

3.96.2.1 ModifierInstanceClient()

```
ModifierInstanceClient.ModifierInstanceClient (
            Modifier mod,
            PlayerStatus plStatus,
            PlayerUIHandler playerUIHandler,
            int modId,
            int abId,
            float duration )
```

Constructor that instantiates effect objects and calls the correct modifier functions.

Parameters

mod	The modifier for this instance.
plStatus	The playerStatus the modifier is applied to.
playerUIHandler	Reference to the UIHandler, used by the local client to add modifier UI.
modId	Unique identifier for this modifier instance.
abId	The Id of the ability that applied the modifier if any, -1 otherwise.
duration	The initial modifier duration.
effectParent	Either null or the transform we want to put as parent for this modifier

3.96.3 Member Function Documentation

3.96.3.1 GetAbilityId()

```
int ModifierInstanceClient.GetAbilityId ( )
```

Returns the ability ID in this instance, equal to or above 0 means this modifier was applied by an ability, -1 otherwise.

Returns

The ability ID.

3.96.3.2 GetModifier()

```
Modifier ModifierInstanceClient.GetModifier ( )
```

Returns the modifier used by this instance.

Returns

The active modifier.

3.96.3.3 GetModifierId()

```
int ModifierInstanceClient.GetModifierId ( )
```

Returns the unique modifier ID for this instance.

Returns

The modifier ID.

3.96.3.4 OnEnd()

```
void ModifierInstanceClient.OnEnd ( )
```

Called when the modifier effect has ended.

3.96.3.5 SetNewDuration()

```
void ModifierInstanceClient.SetNewDuration (
            float newDuration )
```

Updates the UI elements with the new duration.

Parameters

newDuration	The new duration.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierInstanceClient.cs

3.97 ModifierInstanceServer Class Reference

The instance used when a modifier is active. Only exists on the server.

Public Member Functions

- ModifierInstanceServer (ModifierInfo info, PlayerStatus plStatus, int modId, int abId)
  Constructor which starts the correct update loop as a Coroutine on the playerStatus MonoBehaviour.
- IEnumerator DurationLoop ()
  Update loop when the duration is used.
- IEnumerator TickLoop ()
  Update loop when the ticks are used.
- void OnEnd ()
  Called when the modifier effect has ended.
- void OnCancel ()
  Called when the ability modifier effect is cancelled (e.g. undocking).
- void MaxDuration (float newDuration)
  Used for unique modifiers that doesn't stack. Uses the largest of the given durations.
- Modifier GetModifier ()
  Returns the modifier used by this instance.
- int GetAbilityId ()
  Returns the ability ID in this instance, equal to or above 0 means this modifier was applied by an ability, -1 otherwise.
- int GetModifierId ()
  Returns the unique modifier ID for this instance.

3.97.1 Detailed Description

The instance used when a modifier is active. Only exists on the server.

3.97.2 Constructor & Destructor Documentation

3.97.2.1 ModifierInstanceServer()

```
ModifierInstanceServer.ModifierInstanceServer (
            ModifierInfo info,
            PlayerStatus plStatus,
            int modId,
            int abId )
```

Constructor which starts the correct update loop as a Coroutine on the playerStatus MonoBehaviour.

Parameters

info	Information about this modifier.
plStatus	The playerStatus the modifier is applied to.
modId	Unique identifier for this modifier instance.
abId	The Id of the ability that applied the modifier if any, -1 otherwise.

3.97.3 Member Function Documentation

3.97.3.1 DurationLoop()

```
IEnumerator ModifierInstanceServer.DurationLoop ( )
```

Update loop when the duration is used.

3.97.3.2 GetAbilityId()

```
int ModifierInstanceServer.GetAbilityId ( )
```

Returns the ability ID in this instance, equal to or above 0 means this modifier was applied by an ability, -1 otherwise.

Returns

The ability ID.

3.97.3.3 GetModifier()

```
Modifier ModifierInstanceServer.GetModifier ( )
```

Returns the modifier used by this instance.

Returns

The active modifier.

3.97.3.4 GetModifierId()

```
int ModifierInstanceServer.GetModifierId ( )
```

Returns the unique modifier ID for this instance.

Returns

The modifier ID.

3.97.3.5 MaxDuration()

```
void ModifierInstanceServer.MaxDuration (
            float newDuration )
```

Used for unique modifiers that doesn't stack. Uses the largest of the given durations.

Parameters

newDuration	The duration to compare the current duration with.

3.97.3.6 OnCancel()

```
void ModifierInstanceServer.OnCancel ( )
```

Called when the ability modifier effect is cancelled (e.g. undocking).

3.97.3.7 OnEnd()

```
void ModifierInstanceServer.OnEnd ( )
```

Called when the modifier effect has ended.

3.97.3.8 TickLoop()

```
IEnumerator ModifierInstanceServer.TickLoop ( )
```

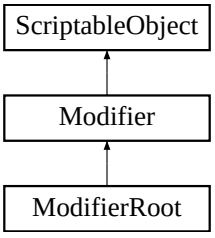
Update loop when the ticks are used.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierInstanceServer.cs

3.98 ModifierRoot Class Reference

Inheritance diagram for ModifierRoot:



Public Member Functions

- override void OnLocalClientStart (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers starts.
- override void OnLocalClientEnd (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers ends.

Additional Inherited Members

3.98.1 Member Function Documentation

3.98.1.1 OnLocalClientEnd()

```
override void ModifierRoot.OnLocalClientEnd (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented from Modifier.

3.98.1.2 OnLocalClientStart()

```
override void ModifierRoot.OnLocalClientStart (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.

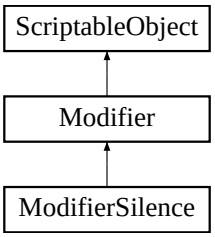
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierRoot.cs

3.99 ModifierSilence Class Reference

Inheritance diagram for ModifierSilence:



Public Member Functions

- override void OnLocalClientStart (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers starts.
- override void OnLocalClientEnd (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers ends.

Additional Inherited Members

3.99.1 Member Function Documentation

3.99.1.1 OnLocalClientEnd()

```
override void ModifierSilence.OnLocalClientEnd (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented from Modifier.

3.99.1.2 OnLocalClientStart()

```
override void ModifierSilence.OnLocalClientStart (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.

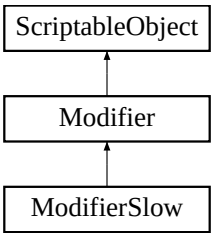
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierSilence.cs

3.100 ModifierSlow Class Reference

Inheritance diagram for ModifierSlow:



Public Member Functions

- override void OnLocalClientStart (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers starts.
- override void OnLocalClientEnd (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers ends.

Public Attributes

- float **slowPercentage** = 0.5f

Additional Inherited Members

3.100.1 Member Function Documentation

3.100.1.1 OnLocalClientEnd()

```
override void ModifierSlow.OnLocalClientEnd (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented from Modifier.

3.100.1.2 OnLocalClientStart()

```
override void ModifierSlow.OnLocalClientStart (
            PlayerStatus playerStatus ) [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.

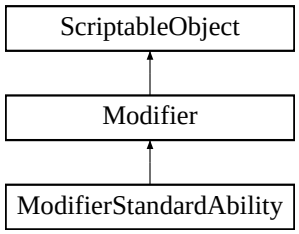
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierSlow.cs

3.101 ModifierStandardAbility Class Reference

Inheritance diagram for ModifierStandardAbility:



Public Member Functions

- override void OnClientStart (PlayerStatus playerStatus, int abilityId)

  Called on every client when the modifiers starts.
- override void OnClientEnd (PlayerStatus playerStatus, int abilityId)

  Called on every client when the modifiers ends.

Additional Inherited Members

3.101.1 Member Function Documentation

3.101.1.1 OnClientEnd()

```
override void ModifierStandardAbility.OnClientEnd (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on every client when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

Reimplemented from Modifier.

3.101.1.2 OnClientStart()

```
override void ModifierStandardAbility.OnClientStart (
            PlayerStatus playerStatus,
            int abilityId )  [virtual]
```

Called on every client when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

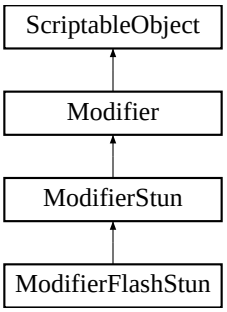
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierStandardAbility.cs

3.102 ModifierStun Class Reference

Inheritance diagram for ModifierStun:



Public Member Functions

- override void OnLocalClientStart (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers starts.
- override void OnLocalClientEnd (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers ends.

Additional Inherited Members

3.102.1 Member Function Documentation

3.102.1.1 OnLocalClientEnd()

```
override void ModifierStun.OnLocalClientEnd (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented from Modifier.

3.102.1.2 OnLocalClientStart()

```
override void ModifierStun.OnLocalClientStart (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.

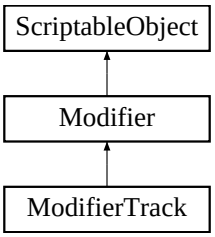
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierStun.cs

3.103 ModifierTrack Class Reference

Inheritance diagram for ModifierTrack:



Public Member Functions

- override void OnLocalClientStart (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers starts.
- override void OnLocalClientEnd (PlayerStatus playerStatus)

  Called on the local client (the client the modifier is applied to) when the modifiers ends.

Public Attributes

- float **damageMultiplier**

Additional Inherited Members

3.103.1 Member Function Documentation

3.103.1.1 OnLocalClientEnd()

```
override void ModifierTrack.OnLocalClientEnd (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers ends.

Parameters

playerStatus	Reference to the associated PlayerStatus.

Reimplemented from Modifier.

3.103.1.2 OnLocalClientStart()

```
override void ModifierTrack.OnLocalClientStart (
            PlayerStatus playerStatus )  [virtual]
```

Called on the local client (the client the modifier is applied to) when the modifiers starts.

Parameters

playerStatus	Reference to the associated PlayerStatus.

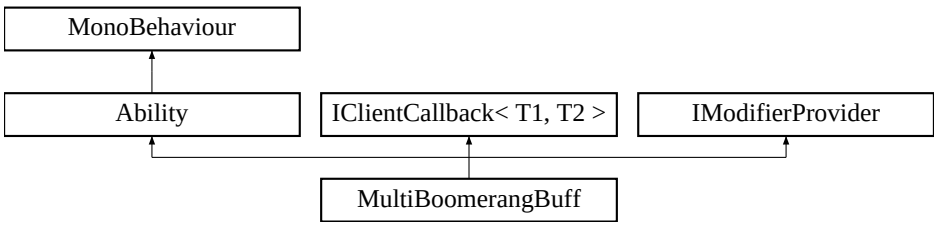
Reimplemented from Modifier.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Modifiers/ModifierTrack.cs

3.104 MultiBoomerangBuff Class Reference

Inheritance diagram for MultiBoomerangBuff:



Public Types

- enum **ClientCallback** { **BuffApplied** }

Public Member Functions

- override void ButtonDown ()

  Callback for what this ability does locally when its associated button is pressed
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- override void SetModifier (bool state=false)

  Callback for what this ability is supposed to do when a modifier state changes
- IEnumerator ResetBuff ()

  Coroutine used for resetting any visuals to default state. It waits for the end of the deactivation animation before doing anything.
- int **GetAbilityId** ()
- int **GetBuffModifierId** ()

Public Attributes

- BoomerangThrow **boomerangScript**
- ModifierInfo **buff**
- GameObject [ ] **otherBoomerangVisuals**
- Animator **boomerangAnimator**
- AnimationClip **boomerangAnimationClip**
- string **animationTrigger**
- bool **buffActive** = false
- bool **buffApplied** = false

Additional Inherited Members

3.104.1 Member Function Documentation

3.104.1.1 ButtonDown()

```
override void MultiBoomerangBuff.ButtonDown ( )  [virtual]
```

Callback for what this ability does locally when its associated button is pressed

Implements Ability.

3.104.1.2 ResetBuff()

```
IEnumerator MultiBoomerangBuff.ResetBuff ( )
```

Coroutine used for resetting any visuals to default state. It waits for the end of the deactivation animation before doing anything.

3.104.1.3 SetActive()

```
override void MultiBoomerangBuff.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

3.104.1.4 SetModifier()

```
override void MultiBoomerangBuff.SetModifier (
            bool state = false )  [virtual]
```

Callback for what this ability is supposed to do when a modifier state changes

Parameters

state	The new modifier state

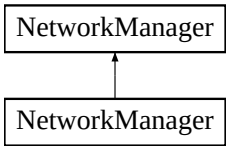
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BoomerangKit/MultiBoomerangBuff.cs

3.105 NetworkManager Class Reference

Inheritance diagram for NetworkManager:



Public Member Functions

- GameSettings **GetGameSettings** ()
- void Disconnect ()

  Causes the network manager to disconnect
- void DisconnectAndReturnToMenu ()

  Disconnect and return the game to the main menu scene
- void StartMatchmakingGame (string gameName, Action< bool, MatchInfo > onCreate)

  Create a matchmaking game
- void StartMatchingmakingClient ()

  Initialize the matchmaking client to receive match lists
- void JoinMatchmakingGame (NetworkID networkId, Action< bool, MatchInfo > onJoin)

Join a matchmaking game

- void ProgressToGameScene ()

  Makes the server change to the correct game scene for our map, and tells all clients to do the same

- void ReturnToMenu (MenuPage returnPage)

  Makes the server change to the menu scene, and bring all clients with it

- NetworkPlayer GetPlayerById (int id)

  Gets a network player by its index

- bool AllPlayersReady ()

  Gets whether all players are ready

- void ClearAllReadyStates ()

  Reset the ready states for all players

- TeamId **GetInitialTeamId** ()
- void RegisterNetworkPlayer (NetworkPlayer newPlayer)

  Register network players so we have all of them

- void DeregisterNetworkPlayer (NetworkPlayer removedPlayer)

  Deregister network players

- override void **OnClientError** (NetworkConnection conn, int errorCode)
- override void **OnClientConnect** (NetworkConnection conn)
- override void **OnClientDisconnect** (NetworkConnection conn)
- override void **OnServerError** (NetworkConnection conn, int errorCode)
- override void **OnServerSceneChanged** (string sceneName)
- override void **OnClientSceneChanged** (NetworkConnection conn)
- override void **OnServerAddPlayer** (NetworkConnection conn, short playerControllerId)
- override void **OnServerRemovePlayer** (NetworkConnection conn, PlayerController player)
- override void **OnServerReady** (NetworkConnection conn)
- override void **OnServerConnect** (NetworkConnection conn)
- override void **OnServerDisconnect** (NetworkConnection conn)
- override void **OnMatchCreate** (bool success, string extendedInfo, MatchInfo matchInfo)
- override void **OnMatchJoined** (bool success, string extendedInfo, MatchInfo matchInfo)
- override void **OnDropConnection** (bool success, string extendedInfo)
- override void OnStartServer ()

  Server resets networkSceneName

- override void OnStopServer ()

  Server destroys NetworkPlayer objects

- override void OnStopClient ()

  Clients also destroy their copies of NetworkPlayer

- override void OnStartHost ()

  Fire host started messages

- virtual void OnPlayerSetReady (NetworkPlayer player)

  Called on the server when a player is set to ready

Static Public Member Functions

- static NetworkPlayer GetPlayerForConnection (NetworkConnection conn)

  Gets the NetworkPlayer object for a given connection

Protected Member Functions

- virtual void Awake ()

  Initialize our singleton
- virtual void Update ()

  Progress to game scene when in transitioning state
- virtual void OnDestroy ()

  Clear the singleton
- void **StopMatchmakingGame** ()
- void UnlistMatch ()

  Sets the current matchmaking game as unlisted
- void ListMatch ()

  Causes the current matchmaking game to become listed again
- virtual void **UpdatePlayerIDs** ()
- void **FireGameModeUpdated** ()

Properties

- NetworkState state `[get]`

  Gets whether we're in a lobby or a game
- static bool IsServer `[get]`

  Gets whether or not we're a server
- List< NetworkPlayer > connectedPlayers `[get]`

  Collection of all connected players
- int playerCount `[get]`

  Gets current number of connected player
- bool hasSufficientPlayers `[get]`

  Gets whether we've currently got enough players to start a game
- static NetworkManager Instance `[get]`

  Gets the NetworkManager instance if it exists
- static bool **InstanceExists** `[get]`

Events

- Action< NetworkPlayer > playerJoined

  Called on all clients when a player joins
- Action< NetworkPlayer > playerLeft

  Called on all clients when a player leaves
- Action hostStarted

  Called on a host when their server starts
- Action serverStopped

  Called when the server is shut down
- Action clientStopped

  Called when the client is shut down
- Action< NetworkConnection > clientConnected

  Called on a client when they connect to a game
- Action< NetworkConnection > clientDisconnected

  Called on a client when they disconnect from a game
- Action< NetworkConnection, int > clientError

  Called on a client when there is a networking error

- Action< NetworkConnection, int > serverError

  Called on the server when there is a networking error
- Action< bool, string > sceneChanged

  Called on clients and server when the scene changes
- Action serverPlayersReadied

  Called on the server when all players are ready
- Action serverClientDisconnected

  Called on the server when a client disconnects
- Action< bool, MatchInfo > matchCreated

  Called when we've created a match
- Action gameModeUpdated

  Called when game mode changes
- Action< bool, MatchInfo > matchJoined

  Called when we've joined a matchMade game
- Action matchDropped

  Called when we've been dropped from a matchMade game

3.105.1 Member Function Documentation

3.105.1.1 AllPlayersReady()

```
bool NetworkManager.AllPlayersReady ( )
```

Gets whether all players are ready

3.105.1.2 Awake()

```
virtual void NetworkManager.Awake ( )  [protected], [virtual]
```

Initialize our singleton

3.105.1.3 ClearAllReadyStates()

```
void NetworkManager.ClearAllReadyStates ( )
```

Reset the ready states for all players

3.105.1.4 DeregisterNetworkPlayer()

```
void NetworkManager.DeregisterNetworkPlayer (
            NetworkPlayer removedPlayer )
```

Deregister network players

3.105.1.5 Disconnect()

```
void NetworkManager.Disconnect ( )
```

Causes the network manager to disconnect

3.105.1.6 DisconnectAndReturnToMenu()

```
void NetworkManager.DisconnectAndReturnToMenu ( )
```

Disconnect and return the game to the main menu scene

3.105.1.7 GetPlayerById()

```
NetworkPlayer NetworkManager.GetPlayerById (
            int id )
```

Gets a network player by its index

3.105.1.8 GetPlayerForConnection()

```
static NetworkPlayer NetworkManager.GetPlayerForConnection (
            NetworkConnection conn )  [static]
```

Gets the NetworkPlayer object for a given connection

3.105.1.9 JoinMatchmakingGame()

```
void NetworkManager.JoinMatchmakingGame (
            NetworkID networkId,
            Action< bool, MatchInfo > onJoin )
```

Join a matchmaking game

3.105.1.10 ListMatch()

```
void NetworkManager.ListMatch ( )  [protected]
```

Causes the current matchmaking game to become listed again

3.105.1.11 OnDestroy()

```
virtual void NetworkManager.OnDestroy ( )  [protected], [virtual]
```

Clear the singleton

3.105.1.12 OnPlayerSetReady()

```
virtual void NetworkManager.OnPlayerSetReady (
            NetworkPlayer player )  [virtual]
```

Called on the server when a player is set to ready

3.105.1.13 OnStartHost()

```
override void NetworkManager.OnStartHost ( )
```

Fire host started messages

3.105.1.14 OnStartServer()

```
override void NetworkManager.OnStartServer ( )
```

Server resets networkSceneName

3.105.1.15 OnStopClient()

```
override void NetworkManager.OnStopClient ( )
```

Clients also destroy their copies of NetworkPlayer

3.105.1.16 OnStopServer()

```
override void NetworkManager.OnStopServer ( )
```

Server destroys NetworkPlayer objects

3.105.1.17 ProgressToGameScene()

```
void NetworkManager.ProgressToGameScene ( )
```

Makes the server change to the correct game scene for our map, and tells all clients to do the same

3.105.1.18 RegisterNetworkPlayer()

```
void NetworkManager.RegisterNetworkPlayer (
            NetworkPlayer newPlayer )
```

Register network players so we have all of them

3.105.1.19 ReturnToMenu()

```
void NetworkManager.ReturnToMenu (
            MenuPage returnPage )
```

Makes the server change to the menu scene, and bring all clients with it

3.105.1.20 StartMatchingmakingClient()

```
void NetworkManager.StartMatchingmakingClient ( )
```

Initialize the matchmaking client to receive match lists

3.105.1.21 StartMatchmakingGame()

```
void NetworkManager.StartMatchmakingGame (
            string gameName,
            Action< bool, MatchInfo > onCreate )
```

Create a matchmaking game

3.105.1.22 UnlistMatch()

```
void NetworkManager.UnlistMatch ( )  [protected]
```

Sets the current matchmaking game as unlisted

3.105.1.23 Update()

```
virtual void NetworkManager.Update ( )  [protected], [virtual]
```

Progress to game scene when in transitioning state

3.105.2 Property Documentation

3.105.2.1 connectedPlayers

```
List<NetworkPlayer> NetworkManager.connectedPlayers  [get]
```

Collection of all connected players

3.105.2.2 hasSufficientPlayers

```
bool NetworkManager.hasSufficientPlayers  [get]
```

Gets whether we've currently got enough players to start a game

3.105.2.3 Instance

```
NetworkManager NetworkManager.Instance  [static], [get]
```

Gets the NetworkManager instance if it exists

3.105.2.4 IsServer

```
bool NetworkManager.IsServer  [static], [get]
```

Gets whether or not we're a server

3.105.2.5 playerCount

```
int NetworkManager.playerCount  [get]
```

Gets current number of connected player

3.105.2.6 state

```
NetworkState NetworkManager.state  [get]
```

Gets whether we're in a lobby or a game

3.105.3 Event Documentation

3.105.3.1 clientConnected

```
Action<NetworkConnection> NetworkManager.clientConnected
```

Called on a client when they connect to a game

3.105.3.2 clientDisconnected

```
Action<NetworkConnection> NetworkManager.clientDisconnected
```

Called on a client when they disconnect from a game

3.105.3.3 clientError

```
Action<NetworkConnection, int> NetworkManager.clientError
```

Called on a client when there is a networking error

3.105.3.4 clientStopped

```
Action NetworkManager.clientStopped
```

Called when the client is shut down

3.105.3.5 gameModeUpdated

```
Action NetworkManager.gameModeUpdated
```

Called when game mode changes

3.105.3.6 hostStarted

```
Action NetworkManager.hostStarted
```

Called on a host when their server starts

3.105.3.7 matchCreated

```
Action<bool, MatchInfo> NetworkManager.matchCreated
```

Called when we've created a match

3.105.3.8 matchDropped

```
Action NetworkManager.matchDropped
```

Called when we've been dropped from a matchMade game

3.105.3.9 matchJoined

```
Action<bool, MatchInfo> NetworkManager.matchJoined
```

Called when we've joined a matchMade game

3.105.3.10 playerJoined

```
Action<NetworkPlayer> NetworkManager.playerJoined
```

Called on all clients when a player joins

3.105.3.11 playerLeft

```
Action<NetworkPlayer> NetworkManager.playerLeft
```

Called on all clients when a player leaves

3.105.3.12 sceneChanged

```
Action<bool, string> NetworkManager.sceneChanged
```

Called on clients and server when the scene changes

3.105.3.13 serverClientDisconnected

```
Action NetworkManager.serverClientDisconnected
```

Called on the server when a client disconnects

3.105.3.14 serverError

```
Action<NetworkConnection, int> NetworkManager.serverError
```

Called on the server when there is a networking error

3.105.3.15 serverPlayersReadied

```
Action NetworkManager.serverPlayersReadied
```

Called on the server when all players are ready

3.105.3.16 serverStopped

```
Action NetworkManager.serverStopped
```

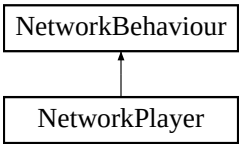
Called when the server is shut down

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Networking/NetworkManager.cs

3.106 NetworkPlayer Class Reference

Inheritance diagram for NetworkPlayer:



Public Member Functions

- override void OnStartLocalPlayer ()

  Set initial values
- override void OnStartClient ()

  Register us with the NetworkManager
- override void OnNetworkDestroy ()

  Deregister us with the manager
- void OnEnterGameScene ()

  Fired when we enter the game scene
- void OnEnterLobbyScene ()

  Fired when we return to the lobby scene, or are first created in the lobby
- void **ClearReady** ()
- void **SetPlayerName** (string newName)
- void **SetPlayerId** (int newPlayerId)
- void **RpcSetGameSettings** (int mapIndex, int modeIndex)
- void **RpcPrepareForLoad** ()
- void **CmdTeamChange** ()
- void **CmdNameChanged** (string name)
- void **CmdSetReady** ()

Protected Member Functions

- virtual void Start ()

  Get network manager
- virtual void OnDestroy ()

  Clean up lobby object for us

Protected Attributes

- GameObject **playerPrefab**
- GameObject **lobbyPrefab**

Properties

- int PlayerId [get]
  Gets this player's id
- string PlayerName [get]
  Gets this player's name
- TeamId PlayerTeamId [get]
  Gets this player's team id
- bool IsReady [get]
  Gets whether this player has marked themselves as ready in the lobby
- Player PlayerInstance [get, set]
  Gets the player script associated with this player
- LobbyPlayer LobbyObject [get]
  Gets the lobby object associated with this player
- static NetworkPlayer LocalPlayerInstance [get]
  Gets the local NetworkPlayer object

Events

- Action< NetworkPlayer > **syncVarsChanged**
- Action< NetworkPlayer > **becameReady**
- Action **gameDetailsReady**

3.106.1 Member Function Documentation

3.106.1.1 OnDestroy()

```
virtual void NetworkPlayer.OnDestroy ( )  [protected], [virtual]
```

Clean up lobby object for us

3.106.1.2 OnEnterGameScene()

```
void NetworkPlayer.OnEnterGameScene ( )
```

Fired when we enter the game scene

3.106.1.3 OnEnterLobbyScene()

```
void NetworkPlayer.OnEnterLobbyScene ( )
```

Fired when we return to the lobby scene, or are first created in the lobby

3.106.1.4 OnNetworkDestroy()

```
override void NetworkPlayer.OnNetworkDestroy ( )
```

Deregister us with the manager

3.106.1.5 OnStartClient()

```
override void NetworkPlayer.OnStartClient ( )
```

Register us with the NetworkManager

3.106.1.6 OnStartLocalPlayer()

```
override void NetworkPlayer.OnStartLocalPlayer ( )
```

Set initial values

3.106.1.7 Start()

```
virtual void NetworkPlayer.Start ( )  [protected], [virtual]
```

Get network manager

3.106.2 Property Documentation

3.106.2.1 IsReady

```
bool NetworkPlayer.IsReady  [get]
```

Gets whether this player has marked themselves as ready in the lobby

3.106.2.2 LobbyObject

```
LobbyPlayer NetworkPlayer.LobbyObject  [get]
```

Gets the lobby object associated with this player

3.106.2.3 LocalPlayerInstance

```
NetworkPlayer NetworkPlayer.LocalPlayerInstance  [static], [get]
```

Gets the local NetworkPlayer object

3.106.2.4 PlayerId

```
int NetworkPlayer.PlayerId  [get]
```

Gets this player's id

3.106.2.5 PlayerInstance

```
Player NetworkPlayer.PlayerInstance  [get], [set]
```

Gets the player script associated with this player

3.106.2.6 PlayerName

```
string NetworkPlayer.PlayerName  [get]
```

Gets this player's name

3.106.2.7 PlayerTeamId

```
TeamId NetworkPlayer.PlayerTeamId  [get]
```

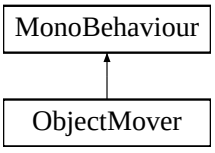
Gets this player's team id

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Networking/NetworkPlayer.cs

3.107 ObjectMover Class Reference

Inheritance diagram for ObjectMover:



Public Attributes

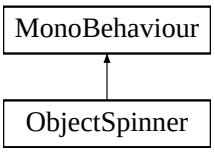
- float **timeToUse**
- bool **backwardsWhenDone**
- Vector3 **startPoint**
- Vector3 **endPoint**

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Utilities/ObjectMover.cs

3.108 ObjectSpinner Class Reference

Inheritance diagram for ObjectSpinner:



Public Attributes

- float **rotationSpeed** = 10f
- Vector3 **axis** = new Vector3(0, 1, 0)

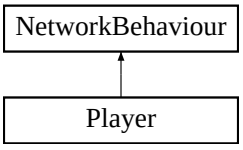
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Utilities/ObjectSpinner.cs

3.109 Player Class Reference

Handles the initialization for the local and remote events for each Player.

Inheritance diagram for Player:



Public Member Functions

- override void **OnStartClient** ()
- override void **OnNetworkDestroy** ()
- void **DisableControl** ()
- void **EnableControl** ()
- void **SetPlayerActive** (bool state)
- void Prespawn ()

  Prespawning, used by round based modes to ensure the player is in the correct state prior to running spawn flow
- void **RespawnReposition** (Vector3 position, Quaternion rotation)
- void RespawnReactivate ()

  Reactivates the player as part of the spawn process.
- void IncrementScore ()

  Convenience function for increasing the player score
- void DecrementScore ()

  Convenience function for decreasing the player score
- FieldOfView **GetPlayerFOV** ()
- GameObject **GetPlayerMask** ()
- TeamId **GetPlayerTeamId** ()
- void **MarkPlayerAsRemoved** ()
- void **SetPlayerId** (int id)
- void TargetAddForce (NetworkConnection connection, float strength, ForceMode mode, Vector3 towards↩ Position)

  TargetRpc for adding force to the player rigidbody. Needed because the local player has authority, and needs to be the one adding force.
- void TargetAddForce2 (NetworkConnection connection, float strength, ForceMode mode, Vector3 forceOrigin)

  TargetRpc for adding force to the player rigidbody where the force origin relative to the player matters
- void TargetAddExplosionForce (NetworkConnection connection, float explosionForce, Vector3 explosion↩ Origin, float explosionRadius)

  TargetRpc for adding explosion force to the player rigidbody.
- void CmdInteract (GameObject interactableObject)

  Command called from PlayerInput when interacting with networked interactable objects.

Public Attributes

- ToggleEvent **onToggleShared**
- ToggleEvent **onToggleLocal**
- ToggleEvent **onToggleRemote**
- Material **redPlayer**
- Material **bluePlayer**
- Material **unassignedPlayer**
- List< SpriteRenderer > **playerVisuals**

Properties

- NetworkPlayer **NetworkPlayerInstance** `[get, protected set]`
- PlayerCamera **PlayerCameraInstance** `[get, protected set]`
- Docking **DockingInstance** `[get, protected set]`
- PlayerInput **PlayerInputInstance** `[get, protected set]`
- PlayerHealth **PlayerHealthInstance** `[get, protected set]`
- PlayerStatus **PlayerStatusInstance** `[get, protected set]`
- PlayerCurrency **PlayerCurrencyInstance** `[get, protected set]`

- string **PlayerName** [get]
- int **PlayerNumber** [get]
- bool **RemovedPlayer** [get]
- bool **Ready** [get]
- bool **Initialized** [get]
- int **Score** [get]

3.109.1 Detailed Description

Handles the initialization for the local and remote events for each Player.

3.109.2 Member Function Documentation

3.109.2.1 CmdInteract()

```
void Player.CmdInteract (
            GameObject interactableObject )
```

Command called from PlayerInput when interacting with networked interactable objects.

Parameters

interactableObject	The networked gameobject interacted with.

3.109.2.2 DecrementScore()

```
void Player.DecrementScore ( )
```

Convenience function for decreasing the player score

3.109.2.3 IncrementScore()

```
void Player.IncrementScore ( )
```

Convenience function for increasing the player score

3.109.2.4 Prespawn()

```
void Player.Prespawn ( )
```

Prespawning, used by round based modes to ensure the player is in the correct state prior to running spawn flow

3.109.2.5 RespawnReactivate()

```
void Player.RespawnReactivate ( )
```

Reactivates the player as part of the spawn process.

3.109.2.6 TargetAddExplosionForce()

```
void Player.TargetAddExplosionForce (
            NetworkConnection connection,
            float explosionForce,
            Vector3 explosionOrigin,
            float explosionRadius )
```

TargetRpc for adding explosion force to the player rigidbody.

Parameters

connection	Needed so TargetRpc finds the correct client.
explosionForce	Amount of force in the explosion
explosionOrigin	Center of the explosion
explosionRadius	Radius of the explosion

3.109.2.7 TargetAddForce()

```
void Player.TargetAddForce (
            NetworkConnection connection,
            float strength,
            ForceMode mode,
            Vector3 towardsPosition )
```

TargetRpc for adding force to the player rigidbody. Needed because the local player has authority, and needs to be the one adding force.

Parameters

connection	Needed so TargetRpc finds the correct client.
strength	The force applied.
mode	The force mode used.

3.109.2.8 TargetAddForce2()

```
void Player.TargetAddForce2 (
            NetworkConnection connection,
            float strength,
            ForceMode mode,
            Vector3 forceOrigin )
```

TargetRpc for adding force to the player rigidbody where the force origin relative to the player matters

Parameters

connection	Needed so TargetRpc finds the correct client.
strength	Amount of force applied.
mode	The force mode used.
forceOrigin	Origin of the force.

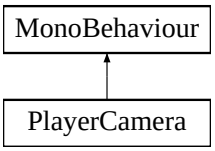
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Player/Player.cs

3.110 PlayerCamera Class Reference

Handles all Camera interactions.

Inheritance diagram for PlayerCamera:



Public Member Functions

- void SetPlayerTransform (Transform newPlayerTarget, bool returnToPlayer=false, bool smoothReturn=false)

  Sets the associated player transform.
- void SetTarget (Transform newTarget, bool smoothing=false)

  Set temporary target to follow, which will override the player transform. This will use the default move speed.
- void SetTarget (Transform t, bool smoothing, float speed)

  Set temporary target to follow using custom move speed, which will override the player transform.
- void SetOrthoSizeTarget (float targetSize)

  Set the orthographicSize for the cameras, will lerp between current and targetSize using the default speed.
- void SetOrthoSizeTarget (float targetSize, float speed)

  Set the orthographicSize for the cameras, will lerp between current and targetSize using the given speed.
- void ReturnToPlayer (bool smooth)

  Call for returning to the player transform using the default speed.
- void ReturnToPlayer (bool smooth, float speed)

  Call for returning to the player transform using the given speed.

Public Attributes

- float **height** = 25f
- float **defaultMoveSpeed** = 20f
- float **defaultScaleSpeed** = 2.5f

3.110.1 Detailed Description

Handles all Camera interactions.

3.110.2 Member Function Documentation

3.110.2.1 ReturnToPlayer() [1/2]

```
void PlayerCamera.ReturnToPlayer (
            bool smooth )
```

Call for returning to the player transform using the default speed.

Parameters

smooth	Whether the return is smooth or instant.

3.110.2.2 ReturnToPlayer() [2/2]

```
void PlayerCamera.ReturnToPlayer (
            bool smooth,
            float speed )
```

Call for returning to the player transform using the given speed.

Parameters

smooth	Whether the return is smooth or instant.
speed	The move speed utilized.

3.110.2.3 SetOrthoSizeTarget() [1/2]

```
void PlayerCamera.SetOrthoSizeTarget (
            float targetSize )
```

Set the orthographicSize for the cameras, will lerp between current and targetSize using the default speed.

Parameters

targetSize	The new orthographicSize.

3.110.2.4 SetOrthoSizeTarget() [2/2]

```
void PlayerCamera.SetOrthoSizeTarget (
            float targetSize,
            float speed )
```

Set the orthographicSize for the cameras, will lerp between current and targetSize using the given speed.

Parameters

targetSize	The new orthographicSize.
speed	The lerp speed utilized.

3.110.2.5 SetPlayerTransform()

```
void PlayerCamera.SetPlayerTransform (
            Transform newPlayerTarget,
            bool returnToPlayer = false,
            bool smoothReturn = false )
```

Sets the associated player transform.

Parameters

newPlayerTarget	The new player transform.
returnToPlayer	Whether to move the camera to this transform.
smoothReturn	Whether the return is smooth or instant.

3.110.2.6 SetTarget() [1/2]

```
void PlayerCamera.SetTarget (
            Transform newTarget,
            bool smoothing = false )
```

Set temporary target to follow, which will override the player transform. This will use the default move speed.

Parameters

newTarget	The new transform to follow.
smoothing	Whether to smoothly follow target.

3.110.2.7 SetTarget() [2/2]

```
void PlayerCamera.SetTarget (
            Transform t,
            bool smoothing,
            float speed )
```

Set temporary target to follow using custom move speed, which will override the player transform.

Parameters

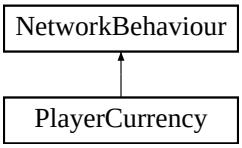
newTarget	The new transform to follow.
smoothing	Whether to smoothly follow target.
speed	The move speed utilized.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Player/PlayerCamera.cs

3.111 PlayerCurrency Class Reference

Inheritance diagram for PlayerCurrency:



Public Member Functions

- void **Initialize** (Player pl)
- void CmdAddCurrency (int amount)

  Command for adding a new amount to the currency. This will automatically trigger the OnCurrencyChange hook

Public Attributes

- int **currency** = 0

3.111.1 Member Function Documentation

3.111.1.1 CmdAddCurrency()

```
void PlayerCurrency.CmdAddCurrency (
            int amount )
```

Command for adding a new amount to the currency. This will automatically trigger the OnCurrencyChange hook

Parameters

amount	The amount we add/decrease from the currency total

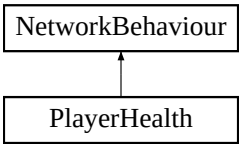
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Player/PlayerCurrency.cs

3.112 PlayerHealth Class Reference

Handles functionality related to the player health.

Inheritance diagram for PlayerHealth:



Public Member Functions

- void Initialize (Player pl)

  Initializes this object.
- void SetDefaults ()

  Set the default/initial state of this object.
- void CmdSetMaxHealth (float newMaxHealth)

  Command called when a DockingKit changes the maxHealth.
- void CmdSetDamageMultiplier (float multiplier)

  Command called when the player receives damageMultiplier change, multiplicative.
- void TakeDamage (float damage, uint playerNetId)

  ServerCallback called when the player takes damage.
- void TakeDamage (float damage)

  ServerCallback called when the player takes damage.
- void Heal (float healing)

  ServerCallback called when the player receives health.

Public Attributes

- float **maxHealth** = 100f
- float **damageMultiplier** = 1f
- SpriteRenderer **damageHealthObject**
- float **flashSpeed** = 8f

Properties

- int **LastDamagedByPlayerNetId** `[get]`

3.112.1 Detailed Description

Handles functionality related to the player health.

3.112.2 Member Function Documentation

3.112.2.1 CmdSetDamageMultiplier()

```
void PlayerHealth.CmdSetDamageMultiplier (
            float multiplier )
```

Command called when the player receives damageMultiplier change, multiplicative.

Parameters

multiplier	change to multiplier

3.112.2.2 CmdSetMaxHealth()

```
void PlayerHealth.CmdSetMaxHealth (
            float newMaxHealth )
```

Command called when a DockingKit changes the maxHealth.

Parameters

newMaxHealth	

3.112.2.3 Heal()

```
void PlayerHealth.Heal (
            float healing )
```

ServerCallback called when the player receives health.

Parameters

healing	The amount of health received.

3.112.2.4 Initialize()

```
void PlayerHealth.Initialize (
            Player pl )
```

Initializes this object.

Parameters

pl	Reference to the associated player.

3.112.2.5 SetDefaults()

```
void PlayerHealth.SetDefaults ( )
```

Set the default/initial state of this object.

3.112.2.6 TakeDamage() [1/2]

```
void PlayerHealth.TakeDamage (
            float damage,
            uint playerNetId )
```

ServerCallback called when the player takes damage.

Parameters

damage	The amount of damage taken.
playerNetId	The player doing the damage.

3.112.2.7 TakeDamage() [2/2]

```
void PlayerHealth.TakeDamage (
            float damage )
```

ServerCallback called when the player takes damage.

Parameters

damage	The amount of damage taken.

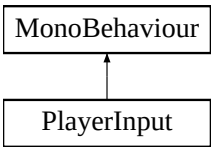
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Player/PlayerHealth.cs

3.113 PlayerInput Class Reference

Handles all player inputs.

Inheritance diagram for PlayerInput:



Public Member Functions

- void **Initialize** (Player pl)
- void **SetDefaults** ()
- void **SetInputActive** (bool state)
- Vector3 GetDirectionVector ()

  directionVector is set every frame based on the movement axis from player input.
- Vector2 GetRotationVector ()

  rotationVector is set every frame based on the rotation axis from player input.
- void SetInputRestrictions (bool state, InputType[ ] inputTypes)

  Used by the local player to self restrict input type. Using int stacks for situations where one modifier removes the restriction, but the restriction is still active by another.

Public Attributes

- string **moveHorizontal** = "Horizontal"
- string **moveVertical** = "Vertical"
- string **rotateHorizontal** = "HorizontalRotation"
- string **rotateVertical** = "VerticalRotation"
- string **dock** = "Dock"
- string **undock** = "Undock"
- string **interact** = "Interact"
- string [ ] **abilityButtons**
- float **moveSpeed**
- float **rotationSpeed**
- IngameMenuHandler **menuHandler**

3.113.1 Detailed Description

Handles all player inputs.

3.113.2 Member Function Documentation

3.113.2.1 GetDirectionVector()

```
Vector3 PlayerInput.GetDirectionVector ( )
```

directionVector is set every frame based on the movement axis from player input.

Returns

The direction vector.

3.113.2.2 GetRotationVector()

```
Vector2 PlayerInput.GetRotationVector ( )
```

rotationVector is set every frame based on the rotation axis from player input.

Returns

The rotation vector.

3.113.2.3 SetInputRestrictions()

```
void PlayerInput.SetInputRestrictions (
            bool state,
            InputType [] inputTypes )
```

Used by the local player to self restrict input type. Using int stacks for situations where one modifier removes the restriction, but the restriction is still active by another.

Parameters

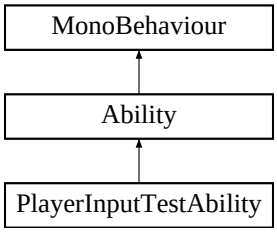
state	The new state of the input restriction.
types	The types to set restriction for.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Player/PlayerInput.cs

3.114 PlayerInputTestAbility Class Reference

Inheritance diagram for PlayerInputTestAbility:



Public Member Functions

- override void InitializeLocalPlayer (AbilityUI abilityUI)

  Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void CancelAbility ()

  Call for cancelling abilities. Override in abilities that may be interrupted.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- Transform **target**
- float **moveSpeed** = 4f
- float **maxDistance** = 10f

Additional Inherited Members

3.114.1 Member Function Documentation

3.114.1.1 ButtonDown()

```
override void PlayerInputTestAbility.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.114.1.2 CancelAbility()

```
override void PlayerInputTestAbility.CancelAbility ( )  [virtual]
```

Call for cancelling abilities. Override in abilities that may be interrupted.

Reimplemented from Ability.

3.114.1.3 InitializeLocalPlayer()

```
override void PlayerInputTestAbility.InitializeLocalPlayer (
            AbilityUI abilityUI )  [virtual]
```

Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.

Reimplemented from Ability.

3.114.1.4 SetActive()

```
override void PlayerInputTestAbility.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

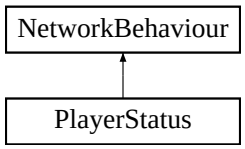
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TestAbilities/PlayerInputTestAbility.cs

3.115 PlayerStatus Class Reference

Handles the modifiers and status effects for the player.

Inheritance diagram for PlayerStatus:



Public Member Functions

- void **Initialize** ()
- void ApplyModifier (ModifierInfo modifierInfo, int abilityId=-1)

  ServerCallback for applying a modifier. Searches through list of current modifiers if the modifier is unique.
- void RemoveModifier (ModifierInstanceServer instance, bool sync=true)

  Removes the modifier instance passed to it. Called by the ModifierInstance when the modifier has ended.
- void RemoveModifier (Modifier modifier)

  Removes the first instance equal to the modifier passed in. Used by abilities through the Docking.
- void RemoveAllModifiers ()

  Iterates through modifier list and stops everything.
- void RemoveAllAbilityModifiers ()

  Iterates through modifier list and stops ability (self applied) modifiers. (Modifiers with a valid abilityId).
- void RemoveAllDebuffModifiers ()

  Iterates through modifier list and removes debuffs.
- void TargetSetUIDuration (NetworkConnection connection, int modifierId, float newDuration)

  TargetRpc for updating UI elements duration.

Public Attributes

- ModifierInfo **stun**
- ModifierInfo **root**
- ModifierInfo **silence**
- ModifierInfo **dot**

3.115.1 Detailed Description

Handles the modifiers and status effects for the player.

3.115.2 Member Function Documentation

3.115.2.1 ApplyModifier()

```
void PlayerStatus.ApplyModifier (
            ModifierInfo modifierInfo,
            int abilityId = -1 )
```

ServerCallback for applying a modifier. Searches through list of current modifiers if the modifier is unique.

Parameters

modifierInfo	The information needed to apply the modifier.
abilityId	The Id of the ability that applied the modifier if any, -1 otherwise.

3.115.2.2 RemoveAllAbilityModifiers()

```
void PlayerStatus.RemoveAllAbilityModifiers ( )
```

Iterates through modifier list and stops ability (self applied) modifiers. (Modifiers with a valid abilityId).

3.115.2.3 RemoveAllDebuffModifiers()

```
void PlayerStatus.RemoveAllDebuffModifiers ( )
```

Iterates through modifier list and removes debuffs.

3.115.2.4 RemoveAllModifiers()

```
void PlayerStatus.RemoveAllModifiers ( )
```

Iterates through modifier list and stops everything.

3.115.2.5 RemoveModifier() [1/2]

```
void PlayerStatus.RemoveModifier (
            ModifierInstanceServer instance,
            bool sync = true )
```

Removes the modifier instance passed to it. Called by the ModifierInstance when the modifier has ended.

Parameters

instance	The ModifierInstance that should be removed.
sync	Should this be synced to the clients.

3.115.2.6 RemoveModifier() [2/2]

```
void PlayerStatus.RemoveModifier (
            Modifier modifier )
```

Removes the first instance equal to the modifier passed in. Used by abilities through the Docking.

Parameters

modifier	The modifier to remove.

Works as long as abilities only self apply unique instances of modifiers, as this only removes based on modifier type (not unique id).

3.115.2.7 TargetSetUIDuration()

```
void PlayerStatus.TargetSetUIDuration (
            NetworkConnection connection,
            int modifierId,
            float newDuration )
```

TargetRpc for updating UI elements duration.

Parameters

connection	Needed so TargetRpc finds the correct client.
modifierId	Used to find correct modifier instance.
newDuration	The new duration.

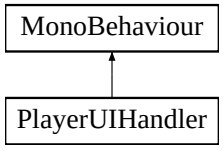
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Player/PlayerStatus.cs

3.116 PlayerUIHandler Class Reference

Handler for the player UI (Abilities, status modifiers, health).

Inheritance diagram for PlayerUIHandler:



Public Member Functions

- void SetDockingKitUI (DockingKit newDockingKit)

  Initialize the AbilityUI with the new DockingKit abilities.
- StatusUI AddStatusModifier (Modifier modifier, float duration)

  Adds a StatusUI element to the PlayerUI.
- void RemoveStatusModifier (StatusUI statusModifier)

  Removed the status modifier from the list of elements.
- void SetCurrentHealth (float health, float maxHealth)

  Updates the HealthUI based on health and maxHealth.
- void PlayCurrencyChangeAnimation (float currencyDifference)

  Starts a coroutine that interpolates text containing the amount of currency earned/spent

Public Attributes

- AbilityUI [ ] **abilities**
- Sprite **emptySlot**
- Transform [ ] **statusBars**
- GameObject **statusPrefab**
- Text **currencyText**
- Text **animatedCurrencyText**
- Text **healthPercentageText**
- Text **healthRatioText**
- Image **healthMask**
- Color **currencyAddColor**
- Color **currencyRemoveColor**
- float **animatedTextTargetOffset** = 75f
- IngameMenuHandler **ingameMenuHandler**

3.116.1 Detailed Description

Handler for the player UI (Abilities, status modifiers, health).

3.116.2 Member Function Documentation

3.116.2.1 AddStatusModifier()

```
StatusUI PlayerUIHandler.AddStatusModifier (
            Modifier modifier,
            float duration )
```

Adds a StatusUI element to the PlayerUI.

Parameters

modifier	The modifier to be added.
duration	The initial duration of the status modifier.

Returns

The instantiated statusUI element.

3.116.2.2 PlayCurrencyChangeAnimation()

```
void PlayerUIHandler.PlayCurrencyChangeAnimation (
            float currencyDifference )
```

Starts a coroutine that interpolates text containing the amount of currency earned/spent

Parameters

currencyDifference	The currency difference from the old total

3.116.2.3 RemoveStatusModifier()

```
void PlayerUIHandler.RemoveStatusModifier (
            StatusUI statusModifier )
```

Removed the status modifier from the list of elements.

Parameters

statusModifier	The statusUI removed.

3.116.2.4 SetCurrentHealth()

```
void PlayerUIHandler.SetCurrentHealth (
            float health,
            float maxHealth )
```

Updates the HealthUI based on health and maxHealth.

Parameters

health	The current health.
maxHealth	The current max health.

3.116.2.5 SetDockingKitUI()

```
void PlayerUIHandler.SetDockingKitUI (
            DockingKit newDockingKit )
```

Initialize the AbilityUI with the new DockingKit abilities.

Parameters

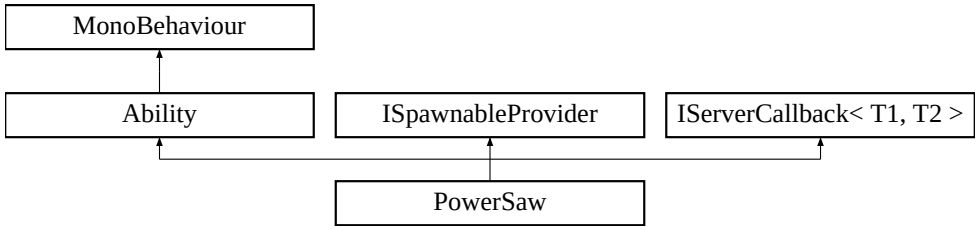
newDockingKit	Reference to the new DockingKit.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/PlayerUIHandler.cs

3.117 PowerSaw Class Reference

Inheritance diagram for PowerSaw:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void CooldownReady ()

  Called from AbilityCooldown when the ability is ready. Setting active to false returns the sawblades to the docking kit visuals.
- override void SetActive (bool state)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- float **triggerToSpawnTime**
- float **sawDamage**
- string **animatorBool**
- Collider **leftBlade**
- Collider **rightBlade**
- GameObject **bladePrefab**

Additional Inherited Members

3.117.1 Member Function Documentation

3.117.1.1 ButtonDown()

```
override void PowerSaw.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.117.1.2 CooldownReady()

```
override void PowerSaw.CooldownReady ( )  [virtual]
```

Called from AbilityCooldown when the ability is ready. Setting active to false returns the sawblades to the docking kit visuals.

Reimplemented from Ability.

3.117.1.3 Initialize()

```
override void PowerSaw.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.117.1.4 SetActive()

```
override void PowerSaw.SetActive (
            bool state )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

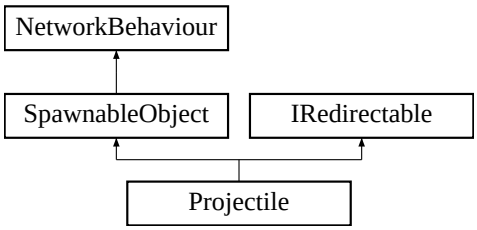
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TankKit/PowerSaw.cs

3.118 Projectile Class Reference

Inheritance diagram for Projectile:



Public Member Functions

- void **Initialize** (Stealth stealthRef, bool firedFromStealth=false)
- void **OnTriggerEnter** (Collider other)

Public Attributes

- float **projectileSpeed**
- float **lifetime**
- bool **hasStealthBonus**
- float **projectileDamage**
- Stealth **stealthBuff**

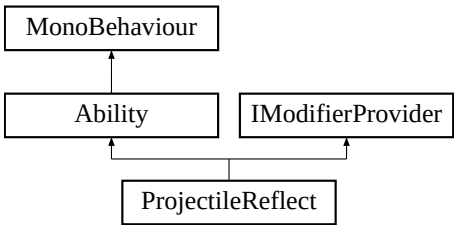
Additional Inherited Members

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/MarksmanKit/Projectile.cs

3.119 ProjectileReflect Class Reference

Inheritance diagram for ProjectileReflect:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Callback for what this ability should do once its associated button has been pressed
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- override void SetModifier (bool state=false)

  Callback for what this ability is supposed to do depending on given state.

Public Attributes

- Transform **shieldTransform**
- float **fadeSpeed** = 5f
- float **fadeOutTimeOffset** = 0.5f
- ModifierInfo **buff**

Protected Member Functions

- override void Update ()

  Runs on every client, but only the local player has cooldown initialized.

Additional Inherited Members

3.119.1 Member Function Documentation

3.119.1.1 ButtonDown()

```
override void ProjectileReflect.ButtonDown ( )  [virtual]
```

Callback for what this ability should do once its associated button has been pressed

Implements Ability.

3.119.1.2 Initialize()

```
override void ProjectileReflect.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.119.1.3 SetActive()

```
override void ProjectileReflect.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

3.119.1.4 SetModifier()

```
override void ProjectileReflect.SetModifier (
            bool state = false )  [virtual]
```

Callback for what this ability is supposed to do depending on given state.

Parameters

state	Whether the ability is to be active or now

Reimplemented from Ability.

3.119.1.5 Update()

```
override void ProjectileReflect.Update ( )  [protected], [virtual]
```

Runs on every client, but only the local player has cooldown initialized.

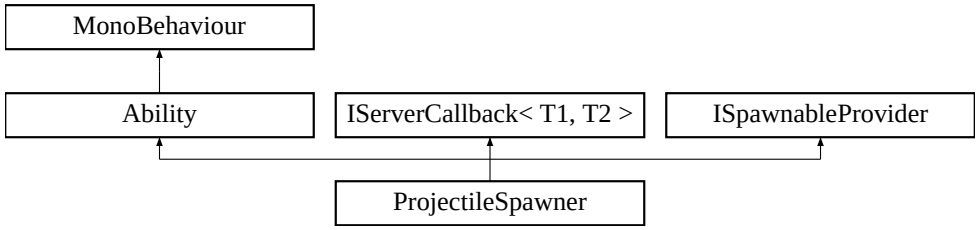
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BrawlerKit/ProjectileReflect.cs

3.120 ProjectileSpawner Class Reference

Inheritance diagram for ProjectileSpawner:



Public Member Functions

- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- GameObject **projectilePrefab**
- float **spawnOffset**
- Stealth **stealthBuff**

Additional Inherited Members

3.120.1 Member Function Documentation

3.120.1.1 ButtonDown()

```
override void ProjectileSpawner.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.120.1.2 SetActive()

```
override void ProjectileSpawner.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

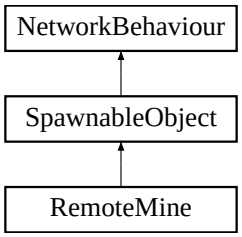
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/MarksmanKit/ProjectileSpawner.cs

3.121 RemoteMine Class Reference

Inheritance diagram for RemoteMine:



Public Member Functions

- void **Initialize** (GameObject owner)
- void Explode ()

  Called when the remote mine is triggered, checking for enemy players in a sphere.
- bool **IsActive** ()

Public Attributes

- float **baseDamage**
- float **explosionRadius**
- float **activationTime**
- ModifierInfo **stunInfo**

Additional Inherited Members

3.121.1 Member Function Documentation

3.121.1.1 Explode()

```
void RemoteMine.Explode ( )
```

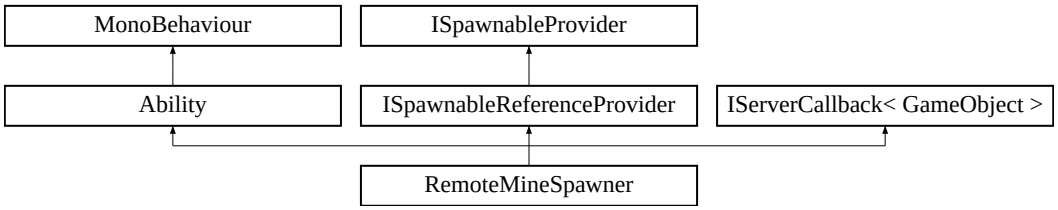
Called when the remote mine is triggered, checking for enemy players in a sphere.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BomberKit/RemoteMine.cs

3.122 RemoteMineSpawner Class Reference

Inheritance diagram for RemoteMineSpawner:



Public Member Functions

- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- string **animatorTrigger**
- GameObject [ ] **minePrefab**
- GameObject **remoteMineReference**

Additional Inherited Members

3.122.1 Member Function Documentation

3.122.1.1 ButtonDown()

```
override void RemoteMineSpawner.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.122.1.2 SetActive()

```
override void RemoteMineSpawner.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

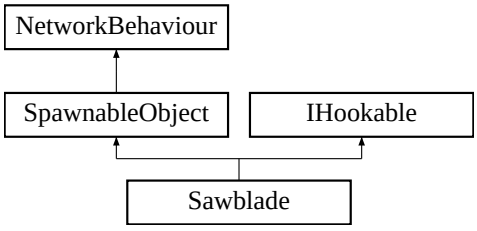
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BomberKit/RemoteMineSpawner.cs

3.123 Sawblade Class Reference

Inheritance diagram for Sawblade:



Public Member Functions

- void Hooked (GameObject playerObject, Transform hook)

  IHookable called when the sawblade has been hooked.

Public Attributes

- float **force** = 30f
- float **damage** = 20f
- float **lifetime** = 10f
- float **cooldownReduction** = 3f

Additional Inherited Members

3.123.1 Member Function Documentation

3.123.1.1 Hooked()

```
void Sawblade.Hooked (
            GameObject playerObject,
            Transform hook )
```

IHookable called when the sawblade has been hooked.

Parameters

playerObject	The hook's assosiated player object.
hook	The hook transform.

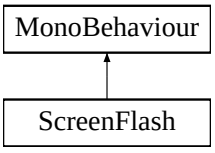
Implements IHookable.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TankKit/Sawblade.cs

3.124 ScreenFlash Class Reference

Inheritance diagram for ScreenFlash:



Public Attributes

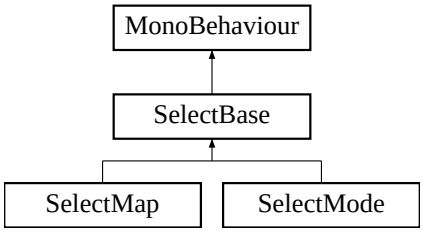
- float **fadeSpeed** = 5f

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/BrawlerKit/ScreenFlash.cs

3.125 SelectBase Class Reference

Inheritance diagram for SelectBase:



Public Member Functions

- int **GetCurrentIndex** ()
- void **OnNextClick** ()
- void **OnPreviousClick** ()

Protected Member Functions

- void **OnIndexChange** ()
- virtual void **AssignByIndex** ()
- void **HandleBounds** ()

Protected Attributes

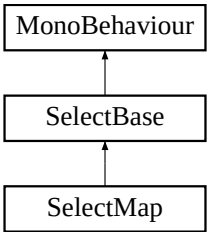
- int **currentIndex** = 0
- int **listLength**

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/SelectBase.cs

3.126 SelectMap Class Reference

Inheritance diagram for SelectMap:



Public Member Functions

- MapInfo **GetSelectedMap** ()

Protected Member Functions

- override void **AssignByIndex** ()

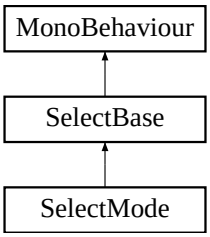
Additional Inherited Members

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/SelectMap.cs

3.127 SelectMode Class Reference

Inheritance diagram for SelectMode:



Public Member Functions

- ModeInfo **GetSelectedMode** ()

Protected Member Functions

- override void **AssignByIndex** ()

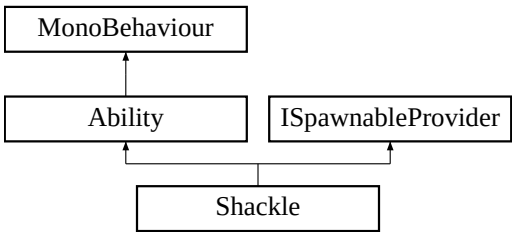
Additional Inherited Members

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/SelectMode.cs

3.128 Shackle Class Reference

Inheritance diagram for Shackle:



Public Member Functions

- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void CooldownReady ()

  Called from AbilityCooldown when the ability is ready. Setting active to true returns the bola to the docking kit visuals.
- override void SetActive (bool state)

  State is here the active of the bola visuals (Opposite of normal).

Public Attributes

- string **animatorTrigger**
- GameObject **spawnablePrefab**
- Transform **spawnPoint**

Additional Inherited Members

3.128.1 Member Function Documentation

3.128.1.1 ButtonDown()

```
override void Shackle.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.128.1.2 CooldownReady()

```
override void Shackle.CooldownReady ( )  [virtual]
```

Called from AbilityCooldown when the ability is ready. Setting active to true returns the bola to the docking kit visuals.

Reimplemented from Ability.

3.128.1.3 SetActive()

```
override void Shackle.SetActive (
            bool state )  [virtual]
```

State is here the active of the bola visuals (Opposite of normal).

Parameters

state	Visual state of the bola.

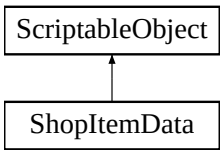
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SniperKit/Shackle.cs

3.129 ShopItemData Class Reference

Inheritance diagram for ShopItemData:



Public Attributes

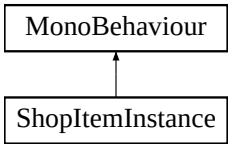
- string **itemName**
- Sprite **icon**
- GameObject **dockingKitPrefab**
- int **price**
- DockingKitId **dockingKitId**
- List< DockingKitDescriptions > **dockingKitDescriptions** = new List<DockingKitDescriptions>(5)

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/ShopItemData.cs

3.130 ShopItemInstance Class Reference

Inheritance diagram for ShopItemInstance:



Public Member Functions

- void **Initialize** (ShopItemData iData, IngameMenuHandler handler)
- void **OnSelectionChange** ()
- void **OnClick** ()

Public Attributes

- ShopItemData **itemData**
- Image **uiIcon**
- Text **priceText**
- Image **unavailableOverlay**
- Text **isEquippedText**

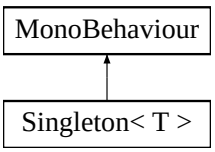
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/ShopItemInstance.cs

3.131 Singleton< T > Class Template Reference

Singleton class of a MonoBehaviour, using Awake and OnDestroy calls.

Inheritance diagram for Singleton< T >:



Protected Member Functions

- virtual void Awake ()

  Awake method to associate singleton with instance
- virtual void OnDestroy ()

  OnDestroy method to clear singleton association

Properties

- static T Instance `[get, protected set]`

  The static reference to the instance
- static bool InstanceExists `[get]`

  Gets whether an instance of this singleton exists

3.131.1 Detailed Description

Singleton class of a MonoBehaviour, using Awake and OnDestroy calls.

Template Parameters

T	Type of the singleton

Type Constraints

***T* : *Singleton*<*T*>**

3.131.2 Member Function Documentation

3.131.2.1 Awake()

```
virtual void Singleton< T >.Awake ( )  [protected], [virtual]
```

Awake method to associate singleton with instance

Reimplemented in AnnouncerModal, SpawnableFactory, and SpawnManager.

3.131.2.2 OnDestroy()

```
virtual void Singleton< T >.OnDestroy ( )  [protected], [virtual]
```

OnDestroy method to clear singleton association

3.131.3 Property Documentation

3.131.3.1 Instance

```
T Singleton< T >.Instance  [static], [get], [protected set]
```

The static reference to the instance

3.131.3.2 InstanceExists

```
bool Singleton< T >.InstanceExists  [static], [get]
```

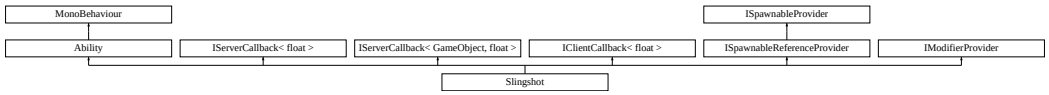
Gets whether an instance of this singleton exists

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Utilities/Singleton.cs

3.132 Slingshot Class Reference

Inheritance diagram for Slingshot:



Public Member Functions

- override void InitializeLocalPlayer (AbilityUI abilityUI)

  Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.
- override void ButtonDown ()

  Start the firing process if cooldown is ready.
- override void ButtonUp ()

  Fires the projectile if the ability is active.
- override void CancelAbility ()

  Cancel the firing process if active.
- override void SetActive (bool fire)

  Synchronizing states, either fires or resets.

Public Attributes

- GameObject **projectilePrefab**
- Transform **projectileSpawnPoint**
- Transform **leftFireIndicator**
- Transform **rightFireIndicator**
- ModifierInfo **snipingSlow**
- Transform **projectileVisuals**
- float **projectileMaxPrecisionY** = -1.5f
- LineRenderer **slingRenderer**
- float **startCurveModifier** = 0.5f
- float **holdCurveModifier** = 0.125f
- float **resetSpeed** = 2f
- AnimationCurve **startCurve**
- AnimationCurve **holdCurve**
- AnimationCurve **projectileFireAnimation**

Additional Inherited Members

3.132.1 Member Function Documentation

3.132.1.1 ButtonDown()

```
override void Slingshot.ButtonDown ( )  [virtual]
```

Start the firing process if cooldown is ready.

Implements Ability.

3.132.1.2 ButtonUp()

```
override void Slingshot.ButtonUp ( )  [virtual]
```

Fires the projectile if the ability is active.

Reimplemented from Ability.

3.132.1.3 CancelAbility()

```
override void Slingshot.CancelAbility ( )  [virtual]
```

Cancel the firing process if active.

Reimplemented from Ability.

3.132.1.4 InitializeLocalPlayer()

```
override void Slingshot.InitializeLocalPlayer (
            AbilityUI abilityUI )  [virtual]
```

Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.

Reimplemented from Ability.

3.132.1.5 SetActive()

```
override void Slingshot.SetActive (
            bool fire )  [virtual]
```

Synchronizing states, either fires or resets.

Parameters

fire	If true fire, otherwise reset.

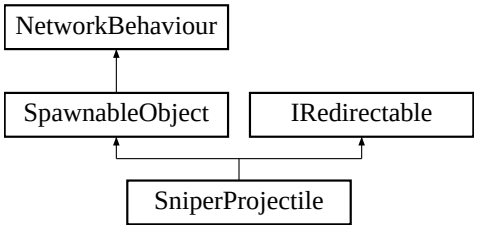
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SniperKit/Slingshot.cs

3.133 SniperProjectile Class Reference

Inheritance diagram for SniperProjectile:



Public Member Functions

- void Initialize (float forceModifier)

  Server call for initializing the projectile based on the forceModifier.
- void RpcInitialize (float forceModifier)

  ClientRpc for synchronizing the forceModifier.

Public Attributes

- float **moveSpeed** = 60f
- float **damage** = 50f
- float **lifetime** = 8f

Additional Inherited Members

3.133.1 Member Function Documentation

3.133.1.1 Initialize()

```
void SniperProjectile.Initialize (
            float forceModifier )
```

Server call for initializing the projectile based on the forceModifier.

Parameters

forceModifier	Modifier in the 0-1 range which affects the stats.

3.133.1.2 RpcInitialize()

```
void SniperProjectile.RpcInitialize (
            float forceModifier )
```

ClientRpc for synchronizing the forceModifier.

Parameters

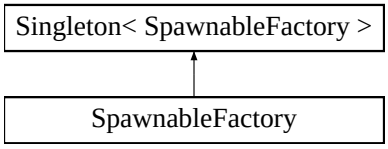
forceModifier	Modifier in the 0-1 range which affects the stats.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SniperKit/SniperProjectile.cs

3.134 SpawnableFactory Class Reference

Inheritance diagram for SpawnableFactory:



Public Member Functions

- SpawnableObject **SpawnObject** (GameObject obj, Vector3 position, Vector3 rotation, uint player, TeamId team)
- void **SpawnDockingKitPickup** (DockingKitId kitId, Vector3 position, Quaternion rotation)
- void **SpawnableDestroyed** (SpawnableObject spawnObject)
- void **CleanupSpawnableList** ()
- void **CleanupPickupList** ()

Public Attributes

- GameObject **dockingKitPickupPrefab**

Protected Member Functions

- override void Awake ()

  Awake method to associate singleton with instance

Additional Inherited Members

3.134.1 Member Function Documentation

3.134.1.1 Awake()

```
override void SpawnableFactory.Awake ( )  [protected], [virtual]
```

Awake method to associate singleton with instance

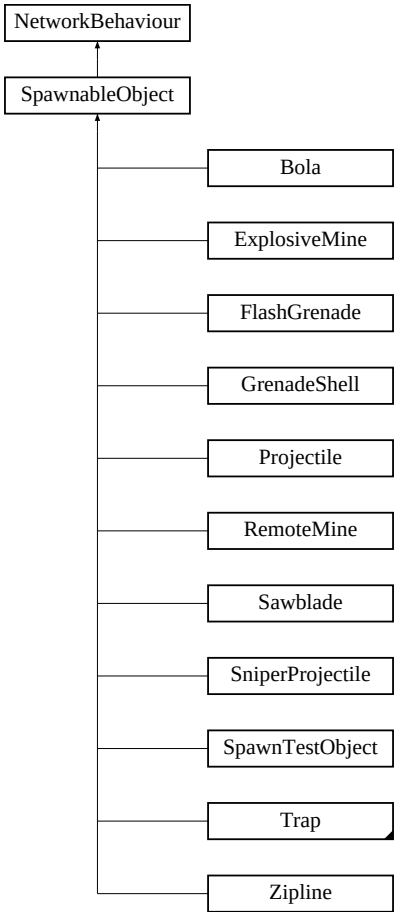
Reimplemented from Singleton< SpawnableFactory >.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/SpawnableFactory.cs

3.135 SpawnableObject Class Reference

Inheritance diagram for SpawnableObject:



Public Member Functions

- uint **GetOwnerPlayerId** ()
- TeamId **GetOwnerTeamId** ()
- void **SetOwner** (uint player, TeamId team)
- bool CheckDamagable (NetworkBehaviour otherObject)

  Check if the other player is damagable by this spawnable. Unassigned team id means teams aren't used.

Protected Member Functions

- virtual void **OnDestroy** ()

Protected Attributes

- uint **playerId**
- TeamId **teamId**

3.135.1 Member Function Documentation

3.135.1.1 CheckDamagable()

```
bool SpawnableObject.CheckDamagable (
            NetworkBehaviour otherObject )
```

Check if the other player is damagable by this spawnable. Unassigned team id means teams aren't used.

Parameters

otherObject	The other player object.

Returns

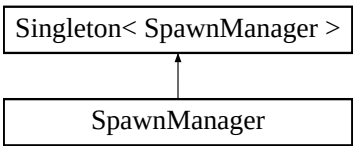
True if damagable, false otherwise.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SpawnableObject.cs

3.136 SpawnManager Class Reference

Inheritance diagram for SpawnManager:



Public Member Functions

- int GetRandomEmptySpawnPointIndex (TeamId teamId)

  Gets index of a random empty spawn point
- SpawnPoint **GetSpawnPointByIndex** (int i)
- Transform **GetSpawnPointTransformByIndex** (int i)
- void CleanupSpawnPoints ()

  Cleans up the spawn points.

Protected Member Functions

- override void Awake ()

  Awake method to associate singleton with instance

Additional Inherited Members

3.136.1 Member Function Documentation

3.136.1.1 Awake()

```
override void SpawnManager.Awake ( )  [protected], [virtual]
```

Awake method to associate singleton with instance

Reimplemented from Singleton< SpawnManager >.

3.136.1.2 CleanupSpawnPoints()

```
void SpawnManager.CleanupSpawnPoints ( )
```

Cleans up the spawn points.

3.136.1.3 GetRandomEmptySpawnPointIndex()

```
int SpawnManager.GetRandomEmptySpawnPointIndex (
            TeamId teamId )
```

Gets index of a random empty spawn point

Returns

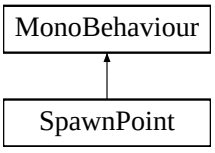
The random empty spawn point index.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Map/SpawnManager.cs

3.137 SpawnPoint Class Reference

Inheritance diagram for SpawnPoint:



Public Member Functions

- TeamId **GetTeamId** ()
- void Decrement ()

  Safely decrement the number of players in the zone and set isDirty to false
- void SetDirty ()

  Used to set the spawn point to dirty to prevent simultaneous spawns from occurring at the same point
- void Cleanup ()

  Resets/cleans up the spawn point

Properties

- Transform **SpawnPointTransform** `[get]`
- bool **isEmptyZone** `[get]`

3.137.1 Member Function Documentation

3.137.1.1 Cleanup()

```
void SpawnPoint.Cleanup ( )
```

Resets/cleans up the spawn point

3.137.1.2 Decrement()

```
void SpawnPoint.Decrement ( )
```

Safely decrement the number of players in the zone and set isDirty to false

3.137.1.3 SetDirty()

```
void SpawnPoint.SetDirty ( )
```

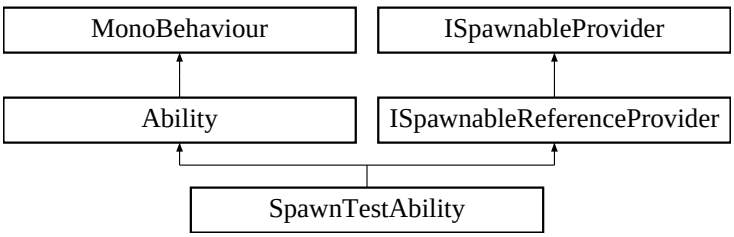
Used to set the spawn point to dirty to prevent simultaneous spawns from occurring at the same point

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Map/SpawnPoint.cs

3.138 SpawnTestAbility Class Reference

Inheritance diagram for SpawnTestAbility:



Public Member Functions

- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- GameObject **spawnTestPrefab**
- string **animatorTrigger**
- int **maxObjects** = 5
- List< GameObject > **spawnedObjects**

Additional Inherited Members

3.138.1 Member Function Documentation

3.138.1.1 ButtonDown()

```
override void SpawnTestAbility.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.138.1.2 SetActive()

```
override void SpawnTestAbility.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

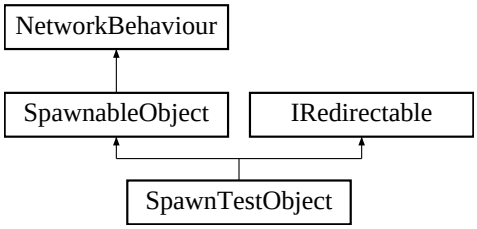
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TestAbilities/SpawnTestAbility.cs

3.139 SpawnTestObject Class Reference

Inheritance diagram for SpawnTestObject:



Public Member Functions

- void RedirectDirection (Vector3 newDirection, int newPlayerId=-1, TeamId newTeamId=TeamId.Unassigned)
  Redirects direction of the spawnable.

Public Attributes

- float **moveSpeed**
- float **damage**

Additional Inherited Members

3.139.1 Member Function Documentation

3.139.1.1 RedirectDirection()

```
void SpawnTestObject.RedirectDirection (
            Vector3 newDirection,
            int newPlayerId = -1,
            TeamId newTeamId = TeamId.Unassigned )
```

Redirects direction of the spawnable.

Parameters

newDirection	The new direction.
newPlayerId	The player id of the new owner, -1 if current owner is kept.
newTeamId	The team id of the new owner, TeamId.Unassigned if current owner is kept.

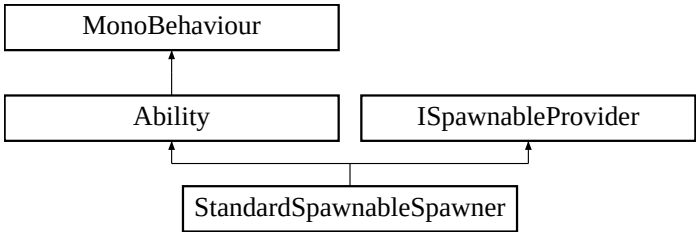
Implements IRedirectable.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TestAbilities/SpawnTestObject.cs

3.140 StandardSpawnableSpawner Class Reference

Inheritance diagram for StandardSpawnableSpawner:



Public Member Functions

- override void ButtonDown ()

  Callback for what this ability should do once its associated button has been pressed
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- GameObject GetSpawnablePrefab (int spawnableId)

  Used by the Docking to get the correct prefab to spawn from the abilities. Parameter only used if the ability has a list of prefabs.

Public Attributes

- string **animatorTrigger**
- GameObject **spawnablePrefab**
- Transform **spawnPoint**

Additional Inherited Members

3.140.1 Member Function Documentation

3.140.1.1 ButtonDown()

```
override void StandardSpawnableSpawner.ButtonDown ( )  [virtual]
```

Callback for what this ability should do once its associated button has been pressed

Implements Ability.

3.140.1.2 GetSpawnablePrefab()

```
GameObject StandardSpawnableSpawner.GetSpawnablePrefab (
            int spawnableId )
```

Used by the Docking to get the correct prefab to spawn from the abilities. Parameter only used if the ability has a list of prefabs.

Parameters

spawnableId	The Id of the spawnable object.

Returns

Reference to the prefab GameObject.

Implements ISpawnableProvider.

3.140.1.3 SetActive()

```
override void StandardSpawnableSpawner.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

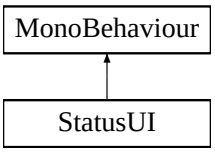
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/StandardSpawnableSpawner.cs

3.141 StatusUI Class Reference

Class for UI status modifiers.

Inheritance diagram for StatusUI:



Public Member Functions

- void Initialize (PlayerUIHandler playerUI, Sprite statusIcon, StatusType statusType, float startDuration)

  Initializes the UI element.
- void SetNewDuration (float newDuration)

  Sets the duration text of the UI element to the parameter.
- void Remove ()

  Remove and destroy this UI element.

Public Attributes

- Color **buffColor**
- Color **debuffColor**
- Image **frame**
- Image **darkMask**
- Text **durationText**
- Image **icon**

3.141.1 Detailed Description

Class for UI status modifiers.

3.141.2 Member Function Documentation

3.141.2.1 Initialize()

```
void StatusUI.Initialize (
            PlayerUIHandler playerUI,
            Sprite statusIcon,
            StatusType statusType,
            float startDuration )
```

Initializes the UI element.

Parameters

playerUI	Reference to the PlayerUIHandler.
statusIcon	The sprite that will be displayed in the UI element
statusType	Status type, buff or debuff.
startDuration	The start duration of the status effect.

3.141.2.2 Remove()

```
void StatusUI.Remove ( )
```

Remove and destroy this UI element.

3.141.2.3 SetNewDuration()

```
void StatusUI.SetNewDuration (
            float newDuration )
```

Sets the duration text of the UI element to the parameter.

Parameters

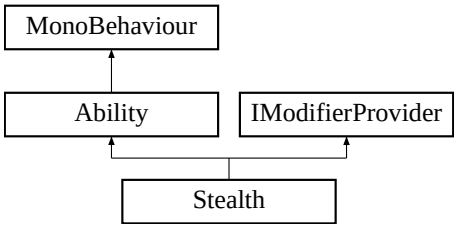
newDuration	The new duration we want to update with.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/UI/StatusUI.cs

3.142 Stealth Class Reference

Inheritance diagram for Stealth:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.
- override void SetModifier (bool state=false)

  Called by the Modifier. Appropriate place for doing local changes.
- void FindPlayerSpriteRenderers (List< string > names)

  Function to find the sprite rendererers relevant to fading into stealth
- bool **IsStealthed** ()
- int **GetAbilityId** ()
- int **GetBuffId** ()

Public Attributes

- float **stealthDamageBonus**
- List< string > **namesOfVisuals**
- List< SpriteRenderer > **visuals**
- ModifierInfo **buffInfo**
- ModifierInfo [ ] **modifierInfos**
- float **fadeTime**

Additional Inherited Members

3.142.1 Member Function Documentation

3.142.1.1 ButtonDown()

```
override void Stealth.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.142.1.2 FindPlayerSpriteRenderers()

```
void Stealth.FindPlayerSpriteRenderers (
            List< string > names )
```

Function to find the sprite rendererers relevant to fading into stealth

Parameters

name	The name of the parent

Returns

3.142.1.3 Initialize()

```
override void Stealth.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.142.1.4 SetActive()

```
override void Stealth.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

3.142.1.5 SetModifier()

```
override void Stealth.SetModifier (
            bool state = false )  [virtual]
```

Called by the Modifier. Appropriate place for doing local changes.

Parameters

state	If the modifier should be activated or deactivated.

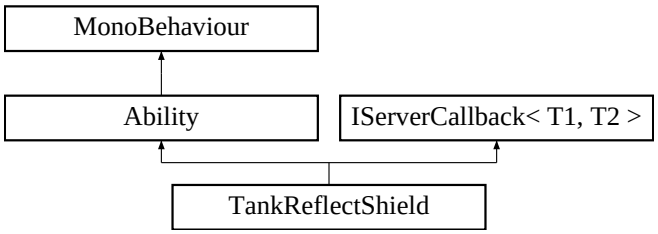
Reimplemented from Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/MarksmanKit/Stealth.cs

3.143 TankReflectShield Class Reference

Inheritance diagram for TankReflectShield:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- float **duration**
- string **animatorBool**
- GameObject **shieldCollider**

Additional Inherited Members

3.143.1 Member Function Documentation

3.143.1.1 ButtonDown()

```
override void TankReflectShield.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.143.1.2 Initialize()

```
override void TankReflectShield.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.143.1.3 SetActive()

```
override void TankReflectShield.SetActive (
            bool state )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TankKit/TankReflectShield.cs

3.144 Team Class Reference

Public Member Functions

- **Team** (TeamId tId)
- void **Reset** (List< Player > playerList)
- void **PlayerDies** (Player player)
- void **PlayerDisconnected** (Player player)
- bool **IsTeamAlive** ()
- int **GetScore** ()
- void **IncrementScore** ()
- string **GetTeamName** ()

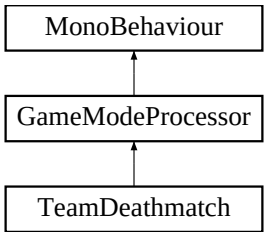
The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/GameModes/Team.cs

3.145 TeamDeathmatch Class Reference

Game mode rules processor for the team deathmatch game mode

Inheritance diagram for TeamDeathmatch:



Public Member Functions

- override void StartGame ()

  Function called on game start
- override void StartRound ()

  Function called on round start
- override void PlayerDies (Player player)

  Handles the death of a player - the player is removed from the local list
- override void PlayerDisconnected (Player player)

  Called when a player disconnects - removed from the local list
- override bool IsEndOfRound ()

  Determines whether it is end of round - if a team has 0 alive
- override void HandleRoundEnd ()

  Handles the round end.
- override string GetRoundEndText ()

  Gets the round end text - winner or draw if appropriate
- override string GetGameOverText ()

  Gets the game over text - winner or draw if appropriate

Properties

- override int ScoreWinTarget [get]

  Gets the score target.

Additional Inherited Members

3.145.1 Detailed Description

Game mode rules processor for the team deathmatch game mode

3.145.2 Member Function Documentation

3.145.2.1 GetGameOverText()

```
override string TeamDeathmatch.GetGameOverText ( )  [virtual]
```

Gets the game over text - winner or draw if appropriate

Returns

The game over end text.

Reimplemented from GameModeProcessor.

3.145.2.2 GetRoundEndText()

```
override string TeamDeathmatch.GetRoundEndText ( )  [virtual]
```

Gets the round end text - winner or draw if appropriate

Returns

The round end text.

Reimplemented from GameModeProcessor.

3.145.2.3 HandleRoundEnd()

```
override void TeamDeathmatch.HandleRoundEnd ( )  [virtual]
```

Handles the round end.

Reimplemented from GameModeProcessor.

3.145.2.4 IsEndOfRound()

```
override bool TeamDeathmatch.IsEndOfRound ( )  [virtual]
```

Determines whether it is end of round - if a team has 0 alive

Returns

true

`false`

Reimplemented from GameModeProcessor.

3.145.2.5 PlayerDies()

```
override void TeamDeathmatch.PlayerDies (
            Player player )  [virtual]
```

Handles the death of a player - the player is removed from the local list

Parameters

player	Player.

Reimplemented from GameModeProcessor.

3.145.2.6 PlayerDisconnected()

```
override void TeamDeathmatch.PlayerDisconnected (
            Player player )  [virtual]
```

Called when a player disconnects - removed from the local list

Parameters

player	The player that disconnects

Reimplemented from GameModeProcessor.

3.145.2.7 StartGame()

```
override void TeamDeathmatch.StartGame ( )  [virtual]
```

Function called on game start

Reimplemented from GameModeProcessor.

3.145.2.8 StartRound()

```
override void TeamDeathmatch.StartRound ( )  [virtual]
```

Function called on round start

Reimplemented from GameModeProcessor.

3.145.3 Property Documentation

3.145.3.1 ScoreWinTarget

```
override int TeamDeathmatch.ScoreWinTarget  [get]
```

Gets the score target.

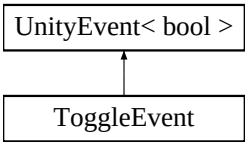
The score target.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/GameModes/TeamDeathmatch.cs

3.146 ToggleEvent Class Reference

Inheritance diagram for ToggleEvent:

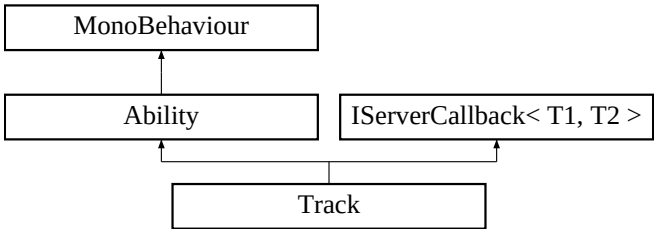


The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Player/Player.cs

3.147 Track Class Reference

Inheritance diagram for Track:



Public Member Functions

- override void Initialize (Docking dock, Animator anim, int abId)

  Initialization that happens locally on every client.
- override void ButtonDown ()

  Called when the associated ability button is pressed. Must be overriden.
- override void SetActive (bool state=false)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- float **castRange**
- LayerMask **layerMask**
- ModifierInfo **trackInfo**

Additional Inherited Members

3.147.1 Member Function Documentation

3.147.1.1 ButtonDown()

```
override void Track.ButtonDown ( )  [virtual]
```

Called when the associated ability button is pressed. Must be overriden.

Implements Ability.

3.147.1.2 Initialize()

```
override void Track.Initialize (
            Docking dock,
            Animator anim,
            int abId )  [virtual]
```

Initialization that happens locally on every client.

Parameters

dock	Reference to the associated Docking.
anim	Reference to the DockingKit animator.
abId	The ability's id in DockingKit abilities list.

Reimplemented from Ability.

3.147.1.3 SetActive()

```
override void Track.SetActive (
            bool state = false )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

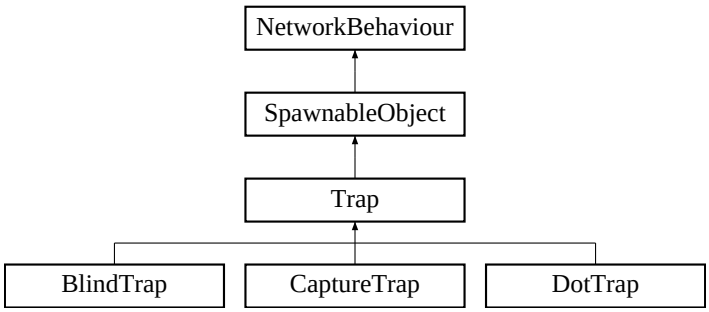
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/MarksmanKit/Track.cs

3.148 Trap Class Reference

Inheritance diagram for Trap:



Public Member Functions

- void Initialize (TrapSpawner owner)

  An initialisation function for caching the script reference to this trap's owner
- void SetVisualState (bool state)

  Sets the visual state of this trap
- virtual void HandleTrigger (PlayerStatus playerStatus)

  A virtual function that allows children of this class to handle what they want to do when a trap is triggered.

Public Attributes

- GameObject **visuals**
- GameObject **extraVisuals**
- float **timeAfterTriggerDestroy** = 1
- string **animatorTrigger**
- Animator **animator**

Protected Member Functions

- void RpcSetExtraVisualsState (bool state)

  ClientRpc used for synchronising the visual state of the trap
- override void OnDestroy ()

  Unity callback for when this trap is destroyed. Tells the owner that this trap is being destroyed

Protected Attributes

- List< Player > **appliedToList** = new List<Player>()
- List< Rigidbody > **appliedToListRbodies** = new List<Rigidbody>()

3.148.1 Member Function Documentation

3.148.1.1 HandleTrigger()

```
virtual void Trap.HandleTrigger (
            PlayerStatus playerStatus )  [virtual]
```

A virtual function that allows children of this class to handle what they want to do when a trap is triggered.

Parameters

playerStatus	The PlayerStatus component of the triggered player

Reimplemented in CaptureTrap, DotTrap, and BlindTrap.

3.148.1.2 Initialize()

```
void Trap.Initialize (
            TrapSpawner owner )
```

An initialisation function for caching the script reference to this trap's owner

Parameters

owner	

3.148.1.3 OnDestroy()

```
override void Trap.OnDestroy ( )  [protected], [virtual]
```

Unity callback for when this trap is destroyed. Tells the owner that this trap is being destroyed

Reimplemented from SpawnableObject.

3.148.1.4 RpcSetExtraVisualsState()

```
void Trap.RpcSetExtraVisualsState (
            bool state )  [protected]
```

ClientRpc used for synchronising the visual state of the trap

Parameters

state	The state of the visuals

3.148.1.5 SetVisualState()

```
void Trap.SetVisualState (
            bool state )
```

Sets the visual state of this trap

Parameters

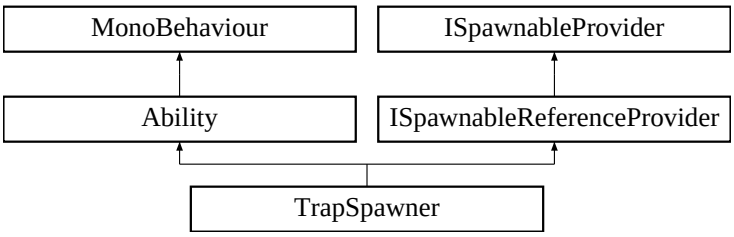
state	The visual state of this trap

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TrapperKit/Trap.cs

3.149 TrapSpawner Class Reference

Inheritance diagram for TrapSpawner:



Public Member Functions

- override void ButtonDown ()

  Callback for what the local client is supposed to do when this ability's button is pressed
- override void SetActive (bool state=false)

  Callback for synchronising visual state based on the given parameter
- void DisplayTrapState (bool state)

  A public function that allows traps to update the visual state of the docking kit's placed trap indicator

Public Attributes

- GameObject **trapPrefab**
- float **trapActiveAlpha** = 0.2f
- List< SpriteRenderer > **trapActiveSprites** = new List<SpriteRenderer>()
- float **lerpSpeed** = 5f

Additional Inherited Members

3.149.1 Member Function Documentation

3.149.1.1 ButtonDown()

```
override void TrapSpawner.ButtonDown ( )  [virtual]
```

Callback for what the local client is supposed to do when this ability's button is pressed

Implements Ability.

3.149.1.2 DisplayTrapState()

```
void TrapSpawner.DisplayTrapState (
            bool state )
```

A public function that allows traps to update the visual state of the docking kit's placed trap indicator

Parameters

state	The display state

3.149.1.3 SetActive()

```
override void TrapSpawner.SetActive (
            bool state = false )  [virtual]
```

Callback for synchronising visual state based on the given parameter

Parameters

state	The state of the ability.

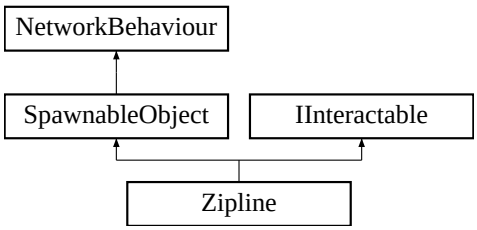
Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/TrapperKit/TrapSpawner.cs

3.150 Zipline Class Reference

Inheritance diagram for Zipline:



Public Member Functions

- bool FirePoint (GameObject player, Vector3 position, Vector3 direction)

  Server call from ZiplineGun whenever a point is fired.

Public Attributes

- Transform **wallEndPoint**
- Transform **lineStartPoint**
- Transform **lineEndPoint**
- SphereCollider **sphereCollider**
- LineRenderer **lineRenderer**
- Transform **handles**
- Transform **radiusTransform**
- LayerMask **interruptionLayerMask**
- float **maxFireRange** = 10f
- float **maxLineDistance** = 20f
- float **hookPointFireSpeed** = 40f
- float **normalRotationSpeed** = 10f
- float **playerMoveSpeed** = 20f
- int **uses** = 3

Additional Inherited Members

3.150.1 Member Function Documentation

3.150.1.1 FirePoint()

```
bool Zipline.FirePoint (
            GameObject player,
            Vector3 position,
            Vector3 direction )
```

Server call from ZiplineGun whenever a point is fired.

Parameters

player	The player firing.
position	Fired from this position.
direction	Fired in this direction.

Returns

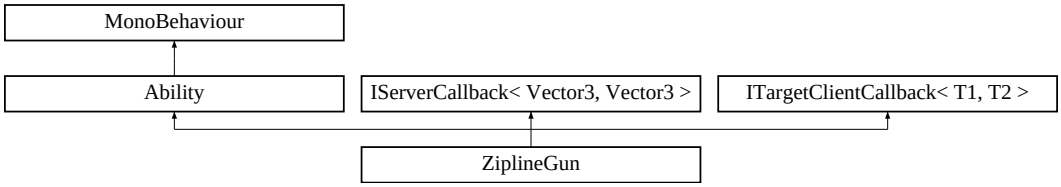
If the shot was successful.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SniperKit/Zipline.cs

3.151 ZiplineGun Class Reference

Inheritance diagram for ZiplineGun:



Public Member Functions

- override void InitializeLocalPlayer (AbilityUI abilityUI)

  Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.
- override void ButtonDown ()

  Activate the radius indicator if cooldown is ready.
- override void ButtonUp ()

  Fire the zipline if the radiusObject is active, this means ButtonDown was called when the cooldown was ready.
- override void SetActive (bool state)

  Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Public Attributes

- GameObject **ziplinePrefab**
- Transform **spawnPoint**
- GameObject **radiusObject**

Additional Inherited Members

3.151.1 Member Function Documentation

3.151.1.1 ButtonDown()

```
override void ZiplineGun.ButtonDown ( )  [virtual]
```

Activate the radius indicator if cooldown is ready.

Implements Ability.

3.151.1.2 ButtonUp()

```
override void ZiplineGun.ButtonUp ( )  [virtual]
```

Fire the zipline if the radiusObject is active, this means ButtonDown was called when the cooldown was ready.

Reimplemented from Ability.

3.151.1.3 InitializeLocalPlayer()

```
override void ZiplineGun.InitializeLocalPlayer (
            AbilityUI abilityUI )  [virtual]
```

Initialization that only happens for the local player (Player controlling this ability). Called after Initialize, so the references are already set up.

Reimplemented from Ability.

3.151.1.4 SetActive()

```
override void ZiplineGun.SetActive (
            bool state )  [virtual]
```

Synchronized state call between clients. Appropriate place for starting local animations/sounds.

Parameters

state	If the ability should be activated or deactivated.

Implements Ability.

The documentation for this class was generated from the following file:

- C:/Users/Andreas/Git Repos/dockitleague/Assets/Scripts/Abilities/SniperKit/ZiplineGun.cs

Index