BACHELOROPPGAVE:

Pyroeis

FORFATTERE:
Magnus Bjerke Vik
Asbjørn Sporaland

DATO:
19.05.2014

Sammendrag av Bacheloroppgaven

Tittel:	**Pyroeis**
Dato:	19.05.2014
Deltakere:	Magnus Bjerke Vik Asbjørn Sporaland
Veiledere:	Mariusz Nowostawski
Oppdragsgiver:	Ådne Midtlien, Ko-aks AS
Kontaktperson:	Asbjørn Sporaland, sporaland@gmail.com
Nøkkelord:	Spill, Programmering, Mars, Utenomjordisk, Mysterie, C++
Antall sider:	99
Antall vedlegg:	16
Tilgjengelighet:	Åpen

Sammendrag:

Pyroeis er et 2D action exploration spill hvor spilleren er en astronaut på Mars. Spillet bruker en spesialbyget spillmotor. Spillerenes oppdrag er å utforske hulene i bakken på planeten. Spillets verden er prosedyrisk generert og evig. Verden kan ødelegges av spilleren ved å kaste bomber. Navigasjon gjennom verden skjer ved bruk av en jetpack. Spillerens reise inn i planeten vil ta dem gjennom et variert landskap som forandres konstant. Fiendlige romvesener angriper spilleren, for å forsvare seg må spilleren bruke skytevåpen. Våpnene kan bli oppgradert ved å bruke resurser som finnes i verden. Ved å bruke bomber kan spilleren lage nye veier gjennom hulene. Lys kan brukes for at spilleren skal se hva som skjuler seg i de mørke gangene på mars. Dersom spilleren tar skade kan han helbrede seg selv ved a bruke førstehjelps skrin. Denne oppgaven beskriver prosessen for å lage Pyroeis, fra ide til et spill. Oppgaven beskriver spilldesign, teknisk design, implementering, brukertesting og distribusjon. Utvilingen av spillet videreutvikler en spillmotor som er utviklet i et tidligere prosjekt.

Summary of Graduate Project

Title:	**Pyroeis**
Date:	19.05.2014
Participants:	Magnus Bjerke Vik Asbjørn Sporaland
Supervisor:	Mariusz Nowostawski
Employer:	Ådne Midtlien, Ko-aks AS
Contact Person:	Asbjørn Sporaland, sporaland@gmail.com
Keywords:	Game, Programming, Mars, Alien, Mystery, C++
Pages:	99
Attachments:	16
Availability:	Open

Abstract: Pyroeis is a 2D action exploration game for PC that uses a custom built engine to send the player to Mars as an astronaut. Players will explore the mysterious caves and creatures below the planet's surface. The game features procedurally generated caves and content in an infinite world, where the terrain can be destroyed. To move through the caves, the astronaut uses a jetpack. When exploring the caves, they will encounter varying terrain formation and biota. Hostile alien monsters will attack them, so to defend themselves a set of guns are available. The guns can be upgraded by creating gems from dust found in the world. Using a set of equipment, they can destroy caves, provide light in the world, and heal themselves. This document describes the process of making Pyroeis from the idea, to the final result. Game design, technical design, implementation, user testing, and deployment are discussed. The game expands upon an engine created for a previous project.

Preface

We would like to thank Mariusz Nowostawski for being our supervisor, providing us with feedback and being engaged in our bachelors project. A big thanks also goes to Jakob Solheim for providing us with art, helping out with the game design and providing us with feedback throughout the project. Thanks to Edvin Avdagic, Håkon Nordling and Tomas Klungerbo Olsen for testing our game. Thanks to Jayson Mackie for providing feedback and helping us get started with the project. We would also like to thank Simon McCallum for providing feedback and helping out with the LaTex document.

Contents

List of Figures

List of Tables

List of Listings

1 Introduction

1.1 Project Description

1.1.1 Background

For a course project at Gjøvik University College IMT 3601 Game Programming, Magnus Bjerke Vik, Asbjørn Sporaland, Håkon Nordling, Edvin Avdagic and Tomas Klungerbo Olsen, developed a game engine and game called Project Nox from scratch. This group of people will henceforth be referred to as "nox group". The goal for the project was to create a reusable, maintainable and flexible game engine with realistic physics and dynamic lighting, that potentially would be used for our bachelors degree, and later for our company: Suttung Digital.

The result from Project Nox was satisfying (the report can be read in Appendix A). While it was missing a lot of game mechanics, the engine was well developed. With this, the nox group decided together to develop new games with the engine and further enhance the engine for the bachelors project. Because of the group's large size of five people, the nox group split into two separate bachelor project groups, Pyroeis and Hatchling respectively. Pyroeis is the name of the game developed for this bachelor.

The idea for Pyroeis is entirely our own. It's based on a mine collapse idea where players are required to support cave walls in order to avoid cave-ins and survive. We found this idea intriguing, and wanted to further develop a game where cave exploration is a key aspect. Other factors affecting our choices in the early stages of our bachelors project, was personal interests in area of technology and research relevant to game programming. Magnus was interested in doing procedural generation of the world, and Asbjørn was interested in world destruction. These two concepts fit well together. We agreed on a couple of terms:

- We will use our own engine created for Project Nox.
- The game will be 2D and rendered from the side.
- We will incorporate terrain destruction and procedural generation.
- We will strengthen the game engine.

1.1.2 Goal

The project will both create a finished game that later potentially can be commercially published, and further enhance the capabilities of the engine created for Project Nox. This is split into two goals. The first is to create a game which has infinite explorable caves procedurally generated with each new game. The second is to further enhance the capabilities of the engine by adding new functionality and improving already existing functionality. The first goal is there to drive the development of engine by requiring new and better functionality. In the project plan, found in Appendix C, only the first goal is mentioned, but the second goal is just as important. The first goal is broken into several sub-goals:

- A two-dimensional view from the side into the caves of Mars.
- A playable character who can both walk, jump, and use a jetpack.

- Provide infinite procedurally generated cave systems to explore.
- Provide varying and interesting biomes with wilderness inside the caves.
- An environment that can be destructed by explosives and other means allowing the player to explore the caves further.
- Enemies in the form of alien life will attack the player on sight.
- Collectable resources around the caves.
- Upgradable gear by using resources collected.
- Realistic physics simulation letting the player interact with the world.
- Dynamic lighting were objects in the world cast shadows.

1.2 Project Audience

1.2.1 Product Audience

The game is targeted towards males in the age range 13–30. Is is not targeted towards casual players, but rather for more hardcore players that want action and an exploration adventure filled with mystery and challenge. Since the game is made for PC, only players with a PC is targeted. The players should have played other games with similar control schemes.

1.2.2 Thesis Audience

The thesis is written in English to target the international community. The content is targeted to game developers, software developers interested in developing games, and students studying game development. It is expected that the reader has knowledge in programming, C++ and software development. For sections about rendering, knowledge in graphics programming on the GPU, especially with OpenGL, is also expected.

1.3 Academic Background

Both of us attended the game programming course and therefore have the same academic background. In addition to the courses, both have programmed several game demos and other software during their free time. Both are most experienced with C++, but each person has a set of other languages that they have been using. We have experience with making games for both PC and mobile, and creating games from scratch rather than using existing game engine. Asbjørn is skilled with the OS X and iOS operating systems, while Magnus is skilled with the Linux and Android OS. Both also know the Windows OS well. With this, the team has a good background in multiplatform support.

None of us have practical experience with planning and implementing a large project, such as this. Our knowledge in planning, designing and the various development methodologies is only theoretical.

1.4 Roles

Asbjørn Sporaland and Magnus Bjerke Vik are developers and designers of the product. Magnus Bjerke Vik is in addition to this, the project manager through the whole project. His responsibility is to manage the tasks, the time schedule, distributing work, and ensuring that the goals are met and necessary meetings are held. For the project we co-operate with Jakob Solheim for the visual graphics. He does this as a part-time non-binding role, and is free to leave at any time. Although his primary role in the development has been supplying us with art, he has also been involved in the game design. Ådne Midtlien at

Ko-Aks, who advises our company, is our employer and helps us start Suttung Digital. Mariusz Nowostawski is our supervisor, helping us with feedback on both the game and the thesis.

1.5 Development Framework

1.5.1 Software Development Methodology

Creating a game with an enjoyable experience requires an iterative process, so that the design can iteratively grow to the desired experience. At the same time the project has a strict time schedule and the goal is ambitious, so a framework that has strict check points is needed. This led us to using the Scrum framework. [1]

Because of the short project period, Scrum sprints were set to last one week each, letting us weekly review the product and our progress. Scrum review and planning meetings were held at the end of each Friday. Daily Scrums were held each workday morning to synchronize the developers and plan the day. A team of two can not fit the Scrum roles, so some roles were shared. Both developers act as a Product Owner, managing the product backlog. Magnus took the responsibility of the Scrum Master, ensuring that the framework is used correctly, and that necessary meetings take place. The implementation of Scrum is discussed in more detail in Section 4.3.

1.5.2 Schedule

A gantt chart was created for the project including each sprint and other tasks. [2] It is found int Appendix D. There are four milestones that each describe a set of tasks and features to be completed. The earlier milestones ensure that the product will end up as a working game. The later milestones ensure that the game is properly tested and that it is polished. Both the gantt chart and the milestones were based on an estimated 40 hours of work effort each week by each developer. The gantt chart and each milestone are described in the project plan, in Appendix C. How the milestones are used is described in Section 4.3.

1.6 Document Structure

The document is structured into eight chapters with appendices at the end:

1. Introduction: Introduction to the document, background of the project and the planning process.
2. Requirements and Design: Describes the game idea, game design process, game design document and what was changed.
3. Technical Design: Describes the system technology, architecture and each engine subsystem that is needed to implement the game design.
4. Development Process: Describes our workflow, the tools used and the implementation of the methodology.
5. Implementation: Describes details about the implementation of the game design and research needed to reach the implementation.
6. Deployment: Describes how the game was packaged and published.
7. Testing and User Feedback: Describes the planned testing process, what was used and feedback received from users.
8. Discussion: Discusses the results, group work, and further development.
9. Conclusion: Evaluates the project.

1.7 Terminology

AABB Axis Aligned Bounding Box. Rectangular 2D box aligned with the x and y axis. Often used for checking collisions.

PRNG Pseudo random number generator. Generates numbers that are similar to real random numbers.

soname Shared object name. Used in Unix systems to identify a version of a shared object (dynamic library).

GUI Graphical User Interface. Interface displayed on the screen that a user can interact with.

LTO Link-time optimization. Optimize the code when linking objects together.

WKT Well-known text. Text format used to store different types of geometries.

JSON JavaScript Object Notation. Data interchange format based on JavaScript.

2 Requirements and Design

Instead of doing the normal process of specifying a set of functional requirements, how the software should work is specified in a game design document (GDD). This document can be found in Appendix E. The functionality in the GDD is specified by the developers to please potential customers.

2.1 Game Design

2.1.1 Game Design Document

The GDD is the basis for all of the features planned to be implemented. It works in coherence with the project plan and the development workflow (Section 4.3). From the GDD, more specific goals are extracted and added to each milestone, which then further are used in the development process. It is also an important tool for the developers to synchronize their visualization of the game. Often they think they have the same idea of how the game looks and acts, but in reality they don't. Writing down exactly how the game should be ensures that there won't be conflicts. It also helps drive the idea to new interesting points.

The GDD was created in parallel with the project plan during the first week of the project. After the delivery of the pre-project plan, the GDD has only had minor changes. One of the larger is the addition of the progression section. Though the design of the game changed during development, only details changed, and the GDD still acts as the basis for the features. We decided to not update such details in the document, since these change often and would only slow down the process. The GDD will need a review and refactor process after a while, such as in the current state of the project, to ensure that everyone has the same idea of where the game will be going from this point on. This is especially important with teams larger than two persons.

The GDD starts with an introduction to the game idea. This section is meant to be readable by people other than game developers, and gives a view of the overall vision of the game. It describes the idea, story, key features and the style. The next section goes into details about each game mechanic, the flow of the game, and the user experience. This is the part that is used to derive the milestone goals.

2.1.2 Initial Design

The game design is based around three core features that were decided on early in the process: Destructible terrain, infinite random worlds, and upgradable guns. The large vision is to have the players feel like explorers, looking for new interesting things in an unknown world, while they have to defend themselves against dangerous alien monsters. This is the reason that it takes place under Mars' surface. We know a lot about Mars' surface, but not what's deep underneath it. We initially thought this as a unique story, but later discovered that we were wrong. A game called Waking Mars has a very similar story. [3]

Guns

A type of resources called dust exist to allow the player to upgrade their weapons, so that they can have nice weapon progression capabilities in the game. Dust is the reward the player receives when doing something like killing an enemy, plant, or destroying the terrain. There is a lot of dust dropped with each event to enhance the feeling of being rewarded. A larger amount of dust is needed to create a gem that can be used to upgrade a weapon, so that upgrading is a bigger achievement of the player. This gives the player a feeling of progression and reward on a larger time scale.

There are four different base weapons, each providing a different combat strategy. One can be better to use at many smaller enemies, while another can be better to use on one large. Each of the weapons have four different properties that each can be upgraded by one type of gem. This is so that upgrading a gun is not a linear experience, but rather a strategic choice, making it more interesting. It also gives the player more creative freedom to create the gun of their choice.

To utilize an upgraded weapon property, the weapon must have power available for it. Power can be upgraded by using one of each type of gem. This gives the player a more strategic choice when upgrading their weapons. The power can be redistributed to other properties so that the player can change tactics whenever they want to. The system is inspired by FTL: Faster Than Light's reactor system, where the reactor provides an upgradable number of power bars. Each of these power bars can be allocated to a subsystem on the ship. [4]

World

The world is vast and varying to encourage players to explore it. To make the world varying, it uses biomes that will be defined by some parameters. A biome is a "a major ecological community type", such as a jungle, or desert. [5] Placing different biomes throughout the world provide different experiences. The biome system is inspired from the games Minecraft and Terraria. [6] It bases the biomes on heat and humidity, just like Minecraft until version 1.8. [7] Though instead of randomly defining the heat and humidity level, in our implementation they vary with the depth of the world. An example of the heat and humidity variation is shown in figure 1. What's not mentioned in the document, is that it will on top of vary with depth, randomly vary like in Minecraft. This system ensures that the player gets a varied experience in addition to having a progression downward by having large changes in biomes.

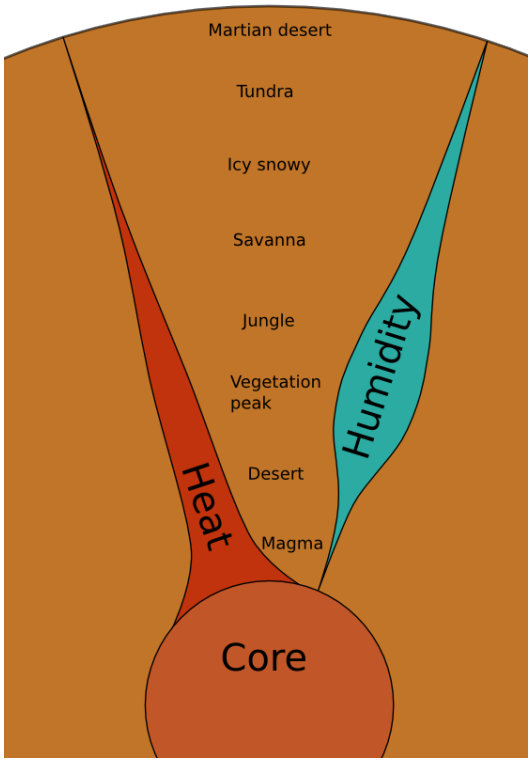


Figure 1: Heat and humidity varying with depth and defining different biomes. Based on the sketch from the GDD in Appendix E.

The world will also be destructible, so that the player can collect resources and create new paths. Many caves will be completely separated by thin walls to enhance the feeling of exploration. The player then needs to find these walls and destroy them to progress or find new things.

Movement

In addition to moving on foot and jumping, the player will have a jetpack that can be used. It would practically be impossible to move without this in many parts of the caves. Normally jetpacks in a 2D side-view game are static. They are controlled by thrusting left, right and up, and the player never rotates. With Pyroeis, one unique element that will make the jetpack more difficult to use, but provides a more realistic feeling, and should make players feel more skilled when they master it, is the realistic physics. Instead of thrusting left or right, they rotate their body and thrust the direction the character sits (feet to head).

Items

The player is equipped with several items to help them through their travels. One of these is light sources. The world is completely dark and only lit by the player's flashlight to make the game challenging. They can use a limited amount of flares and wall lights so that they can see potential dangers. Wall lights are placed on the wall and stay there forever. The lamps are provided so that the player may set up a more permanent safe base. Flares are temporary and affected by physics, but light up a larger area. They are

meant to be used as a tool for the player when they need to see what's around a corner, or in a large open cave.

As mentioned in world section above, the world can be destroyed. To do this the player has a limited amount of bombs. They are dangerous, so the player will need to stay at a safe distance when triggering them. Bombs help the player destroy the world, but to find thin walls that easily provide access to other caves, a sonar device is available.

Items are limited in count, so that the player has to be tactical on when to use them, but if they go empty, they would be completely stuck. To alleviate this, they can create new items by using dust.

Goal

To let the player have something specific to progress towards, a goal has been defined. They need to collect something valuable at near the core of Mars and bring it back to the surface. To provide more replay value, the player can continue in a new world with the same gear in a more difficult mode.

Monsters

Monsters are present to give the player a challenge when progressing. Four types of monsters will be created to make the challenge varied and encourage different tactics.

Visual Design

We have not focused on visual design for the project. Initially we would create placeholder art, but someone stepped in to create production art. This person did not have much experience, so we chose to go for low resolution pixel art. Other than the art, the lighting is important to create a dangerous and mysterious atmosphere.

Audible Design

For audio it was the same as for visual design, except that there was no person to do this. The audio should also try to create a dangerous and mysterious atmosphere.

Tutorial

While designing elements well could minimize the need for a tutorial, it would still be needed for some player. They would also need to be introduced to the controls. The tutorial will first introduce the player to the controls, items and weapons by using a pre-constructed course that is connected to the generated world.

User Interface

The specifics of the GUI was not designed, but an overall flow of the game was created as a flowchart indicating all menus with all actions. One important feature of the GUI is that there is no distinction between the main menu and pause menu. When the player pauses the game, they go back to the main menu.

Difficulty

When designing how the player navigates the world, one of the key elements to keep in mind is the difficulty of the game. The difficulty should have a balance between easy to get started and hard to master. The design focuses on the player feeling like they are actually developing a skill while playing the game. Players should be able to pick up the game and feel that they know how to do what they are required to do. At the same time the experience should be one a journey of growing skill.

2.1.3 Final Design

Several elements in the design changed through the project. One important thing is the goal. Since the game didn't reach a state with enough content to let the player explore and go for a goal a long way down, a temporary goal was created. The goal is a boss at about 1000 meters depth. For later iterations there can be several such bosses as you go down marking larger points of progression, but currently the player wins the game by defeating the boss. Because of the limited time, the boss was designed as a swarm of child monsters attacking the player from all directions. The player has to survive for a certain number of seconds before winning the game. The sonar was not added, because of the limited amount of time and that all caves are connected. To create a similar experience, some cave paths were made very small so that the player can see the path, but would still need to use their bombs. There is no tutorial, again because of the limited amount of time. Many players are confused about how the game works without this.

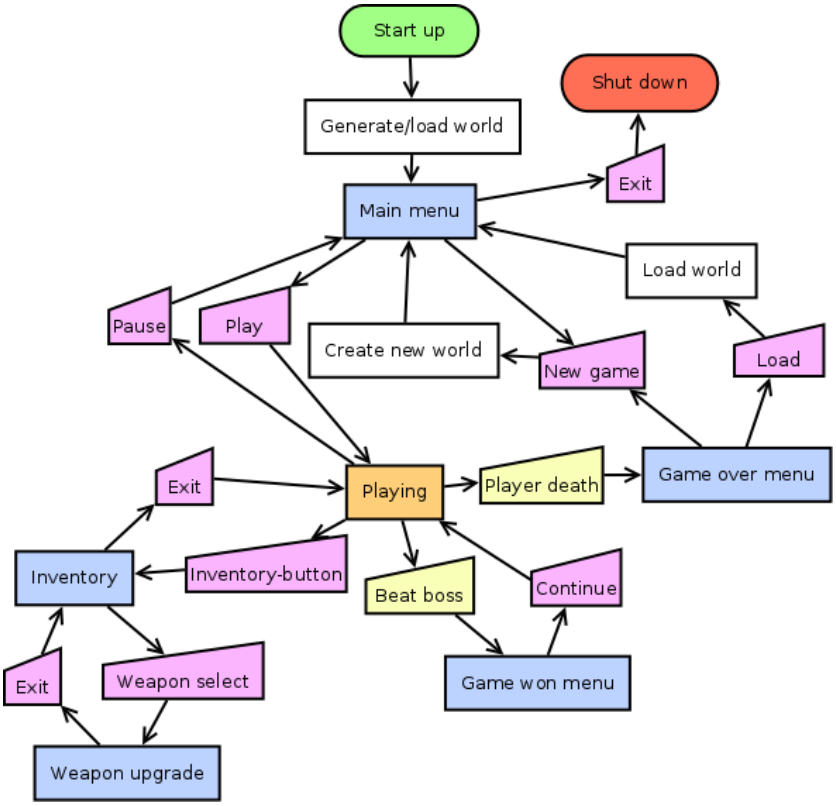


Figure 2: Current flow of the game. For legend and the original flow, see "Game Flow" in Appendix E.

The initial game flow was changed during development. Settings was removed because of bad planning and limited time. The high score list was removed since the game isn't a competitive game and therefore there is little reason to display high scores. The increased difficulty mode "new game+" was also removed because of bad planning and limited time. Otherwise the flow has been simplified, and the world will already be loaded when the player starts the game. The "Generate/load world" box happens while the player is in the menu. When the world is ready, the play button becomes available.

If a save is found, this is loaded, otherwise a new world is generated. Having the world loaded immediately when starting the game, lets the player get into action faster. They can also see the world that they will play in from the menu, as if the game is just paused. The “new game” button is always available and will, within some milliseconds, stop the world generation or loading. This is to not make the player frustrated on having to wait. The save selection was removed because of limited time. Currently only one save can be available, and this is loaded when the game starts. This also removes the unnecessary steps to choose a save when pressing “Continue”. Auto saving when pausing the game has been removed since it is a slow process, and it would be a frustrating experience being forced to wait each time they pause the game. The saving is now part of the main menu, letting the player save whenever they want to. Loading has also been added to the main menu, so that they can load an old save if they get stuck. When they die they would by the original design re-spawn without their items. To let the player have another chance, they can now load an old save when dead. With the old design only the weapon upgrade system was available in the inventory. This was because using items would consume dust, but with the new system, items must be crafted by dust before using them. Therefore the weapon upgrade menu has been separated into its own accessed from the inventory, while the inventory menu has functionality for crafting.

The GUI was never designed before implementation, and was rather designed when development started, based on the game flow chart. It is designed to be scalable since the game window can be resized in any direction. The inventory and weapon customization screens were designed by the artist. The design of the inventory tried to utilize natural mapping to make it easy to understand, but as described in Subsection 7.4.1, it failed on some of the elements.

The weapon upgrading design didn’t change, but many of the properties of each gun were changed to properties that were more appropriate and that could provide more interesting combinations. On the handgun, magazine size and projectile velocity were changed to accuracy and movement stability (less movement stability results in accuracy being affected more when moving). With the current design, there is no limit on the ammunition, so upgrading the magazine size does not make sense. Accuracy and movement stability were added as this can make the handgun more usable. On the sniper rifle, projectile velocity was replaced with fire rate, as the sniper fires fast moving bullets by default, and upgrading fire rate can make the sniper usable in situations with many enemies. On the machine gun the magazine size property was replaced with bullet count. Increasing bullet count can create an interesting weapon that is like an automatic shotgun.

The world biome system was changed to dynamically populate the world directly based on the world heat and moisture rather than defining biome areas. This can lead to unique combinations of biota resulting in unique biome experiences. Expanding the environment with 1–3 more parameters would increase the number of possible unique combinations providing the player with a varied exploration experience.

2.2 Requirements

Functional requirements are, as mentioned in the start of the chapter, defined by the game design document. In addition to this, a set of non-functional requirements has been defined. These are found in Appendix F. Four requirements are defined and their

purpose is to keep the implementation clean and well documented, and the performance of the game good. For the game, two requirements are defined: the game should perform well and it should be reliable and not crash. For the engine and its code, two other requirements are defined: the implementation should be well documented, and the code should be easily maintanable and expandable.

3 Technical Design

3.1 Technology

The entire system is written in C++, using the C++11 standard. To keep the source code as platform independent as possible, system or compiler dependent libraries, features and syntax should not be used. If the C++ standard library supports a required feature, this should be used rather than any third party library.

The program uses SDL 2.0 to create and manage the game window, and its input and output. To render the game state, OpenGL 3.0 is used rather than SDL's own render API. To play audio, OpenAL is used. For Windows and Linux the OpenAL Soft implementation is used, and for Mac OS X, Apple's own implementation of OpenAL is used. Audio files are stored in the Ogg Vorbis format, and libogg and libvorbis is used to decompress and parse the file content. The system is heavily reliant on the JSON (JavaScript Object Notation) format, so jsoncpp is used to parse and write it. Other libraries used are described in the sections about the systems using them.

3.2 System Architecture

Since the game engine used for Pyroeis is based Project Nox, the basic structure and subsystems are described in the report from that project. But there are some new subsystems, some changed subsystems, and some subsystems that aren't described well enough in the previous report. These will be described here. In addition, the very core of the architecture will be described.

3.2.1 Overview

The architecture of the system is two-layered: Application and logic. Application communicates with the platform (SDL2, OpenGL, OpenAL, and the OS), while logic handles the game's states and is platform-independent. In the logic there are three main components: Actor, game view, and event. An actor is an entity in the world. A game view is an interface to the program for a human, ai, or remote player, that often control an actor. Events are passed throughout the program to communicate between subsystems. Several other subsystems that are part of the application or logic layer are also present to handle the world, actors, network, events, game views, physics, rendering, data storage, resources and audio playback. The subsystems are exposed to other subsystems through logic and application context interfaces. While networking is not supported or maintained by Pyroeis, it is still part of the engine, and will therefore be mentioned in some contexts.

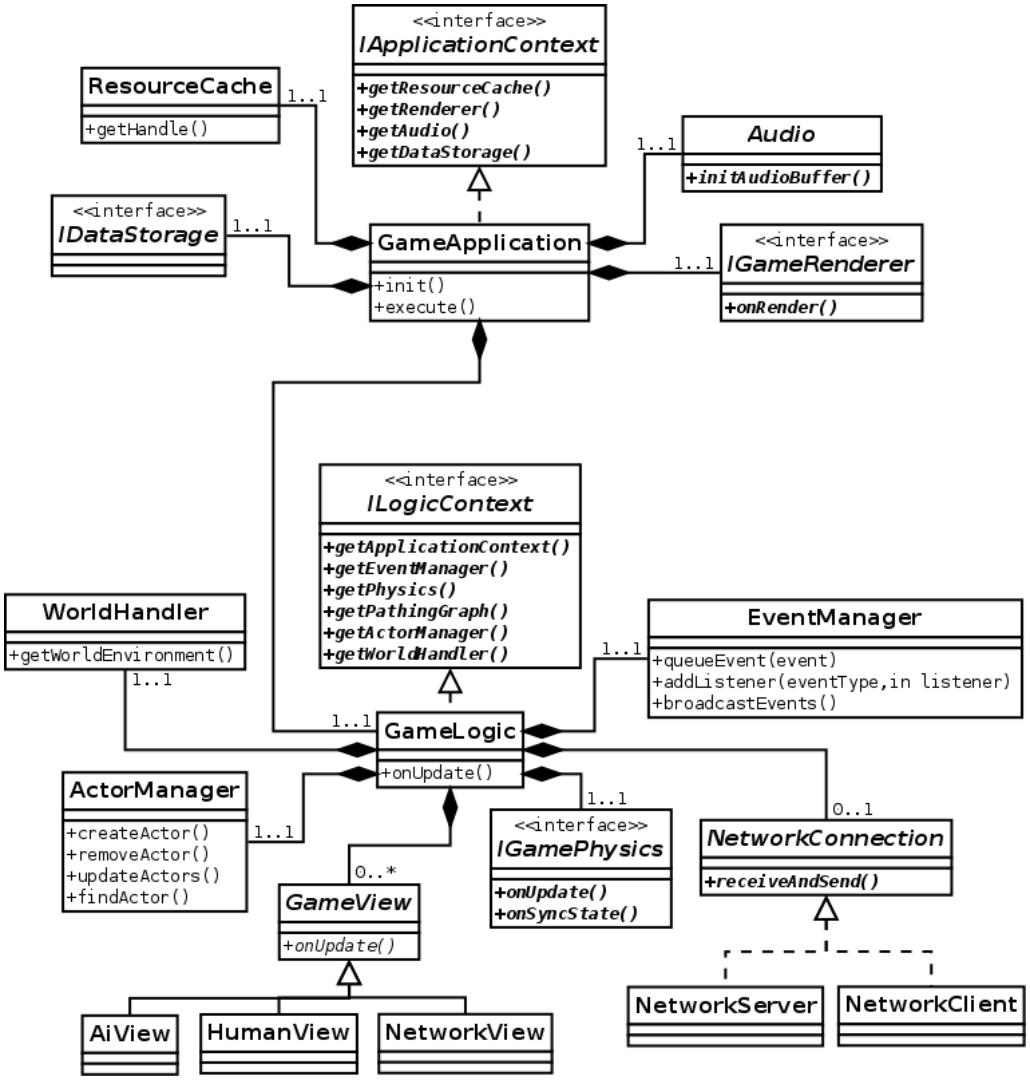


Figure 3: Architecture of Pyroeis. Only the most important classes, functions and data are included.

3.3 Program Flow

The program runs in a loop managed by the GameApplication class. During the loop, the application first loops through all system and window events generated by SDL, handles some of them, and passes the rest to the human view. It then updates the logic before rendering to the window.

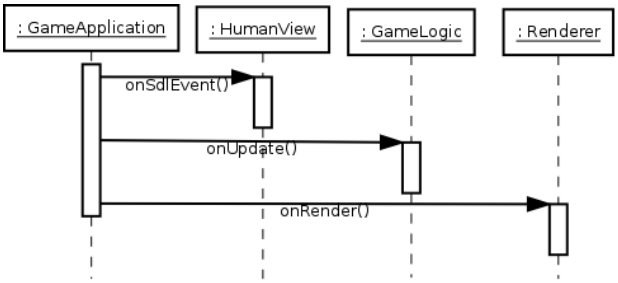


Figure 4: Flow of the application layer. The steps are looped for each frame.

In the update function of the game logic, all of its subsystems are updated. First received network packets are parsed, and packets queued are sent. Next the rest of the subsystems are updated before broadcasting all queued events.

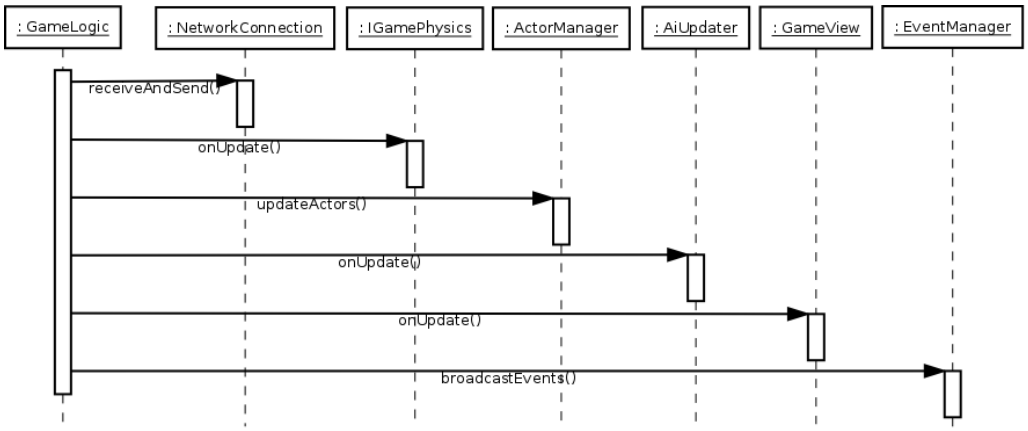


Figure 5: Flow of the logic layer. The steps are looped for each onUpdate call.

3.4 Subsystems

This section describes the game engine's systems. To see the actual API documentation for each system with all the methods and data, refer to the doxygen documentation.

3.4.1 Actor

An actor is an entity in the world: Something that exists with some purpose. Actors are defined by the Actor class and uses the entity-component-system where each component is an instance of the ActorComponent class. Each component has a name to identify it and can be used on an actor to get access to a component.

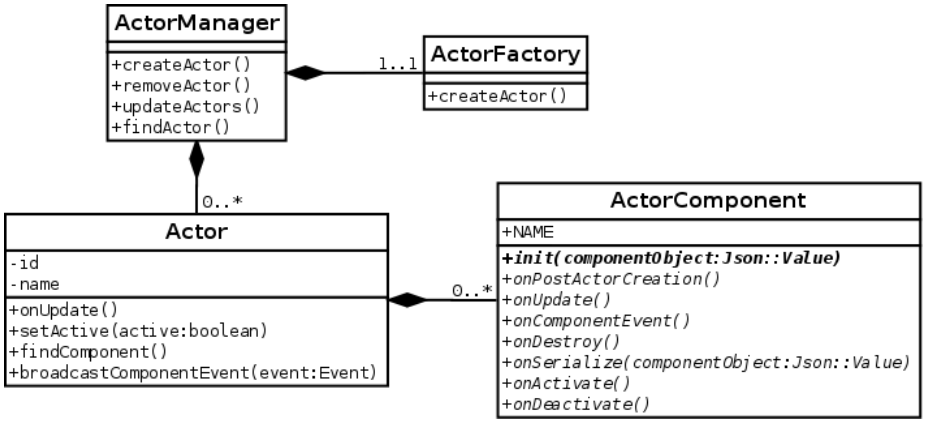


Figure 6: The actor subsystem. ActorManager with its Actors and ActorFactory.

Actors are created from JSON files that describe all of the actor's components and their data. Using JSON to define actors makes it possible to change the behaviour of a game element without recompiling the source code. It also makes it simple to serialize and deserialize actors. When serializing an actor, it is written, together with its components, to a JSON file with exactly the same format as the original file it was initialized from. Deserializing an actor initializes an actor from the file as if it is a completely new actor.

The actor JSON format supports extending other actor files by using the "extend" property. The extend property specifies the location of another actor file to extend from. In addition to the original data of the actor, it gets all the data of the extended actor. This happens recursively through the document objects, and if a property conflicts, the version from the original actor is preserved.

Listing 3.1: Example JSON file for an actor.

```
{
  "name": "Some Name",
  "extend": "someActor",
  "components":
  [
    {
      "name": "TransformComponent",
      "position": {"x": 0, "y": 0}
    }
  ]
}
```

Using this format has been working well, but the component array will probably be changed to an object with later iterations. Using an array does not naturally map to how components are stored in an actor. Components are mapped to a name, and no two components of the same name may exist. With an array several components may exist, but changing it to an object ensures that only one of each exists. It is also more natural to write a component's name, then all its data rather than write its name together with the data.

Listing 3.2: Example actor JSON file with enhanced format.

```
{
  "name": "Some Name",
  "extend": "someActor",
  "components":
  {
    "TransformComponent":
    {
      "position": {"x": 0, "y": 0}
    }
  }
}
```

Actors and their components are updated once each frame. By using this update, or listening for events, the components can interact with system. The components can communicate directly with each other by either directly calling a function on another component, or sending a component event through the actor.

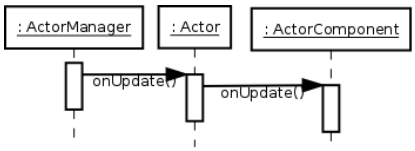


Figure 7: Sequence for actor updating.

Actors are essentially the same as they were in Project Nox, except that they now can be set to inactive, making them stop all their processing. When an actor is set to inactive or active, all of its components receive a notification. Components that usually do processing, like rendering or updating logic, should stop this when inactive, and resume when active.

In Project Nox all actors were directly managed by the game logic. This management has now been moved to a separate class called ActorManager. The actor manager manages all actors and the factory creating them, while the game logic manages the actor manager. To be able to create or remove actors, the actor manager needs to be accessed through the logic context.

The actor factory is of the class ActorFactory, and its job is to create actors either from a JSON value, or from a resource, and then return the actor created. Actors are not stored within the factory, it only gets input and outputs actors. To create an actor, the factory has registered all the component classes on their name in an object factory of the class GenericObjectFactory. When the factory parses the JSON of an actor, it asks the object factory to create each component from their name.

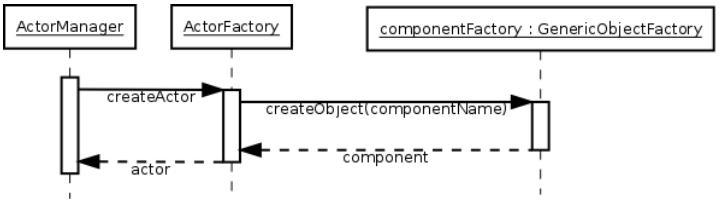


Figure 8: Sequence for actor creation. createObject is called for each component.

3.4.2 Game View

Game views control the actors in the game and the interface for a player, either locally or over the network, and artificial intelligence. Game views are defined by the GameView class that has three specializations: HumanView, AiView, and NetworkView.

Figure 9 is a class and object diagram mix demonstrating how game views are related to actors and actors to components. Player of the class HumanView controls the Astronaught actor that uses several components. It also demonstrates how actors with different purposes use different components.

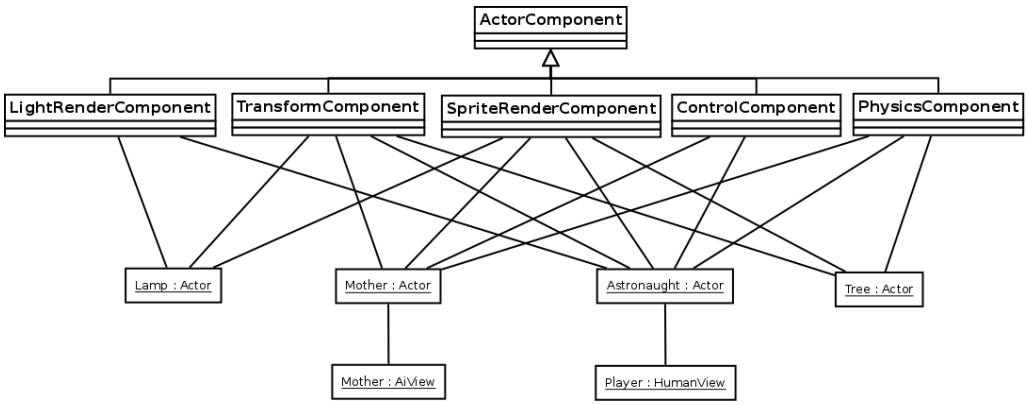


Figure 9: Relationship between views, actors and their components.

3.4.3 Event

Events are broadcast through an event manager. All systems having access to the logic context can listen for events, or send them. The types of event are defined by the EventType enumerator. Events can either be a generic event having only a type and one data segment of a variable type, or a specialized event that can incorporate any data or functionality.

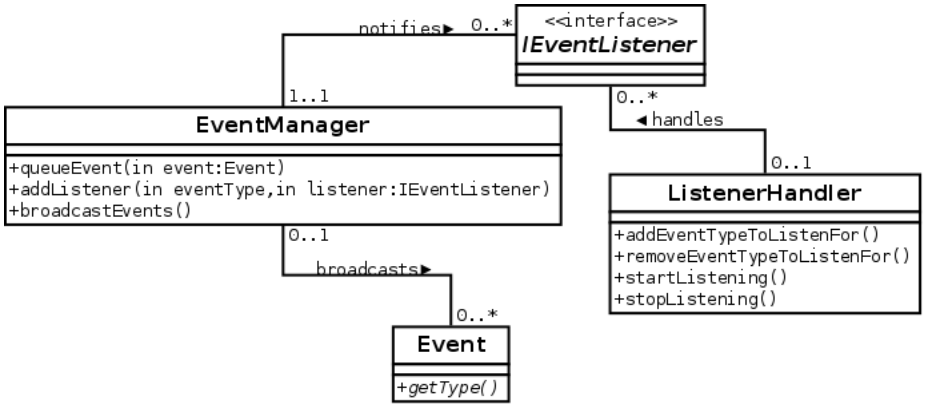


Figure 10: Class diagram for the event subsystem.

To broadcast an event there are two methods: Triggering and queuing. The preferred is to queue the event. All queued events will be broadcast at the end of each logic update from the main thread. The event manager uses a double buffered event queue, so that while broadcasting queued events, new events queued are stored in another buffer that will be broadcast next time.

To make sure than an event is received immediately, an event can be triggered. This method should be used in cases such as when sending control events, so that the control actions are applied during the current update and not the next. It can also be used to trigger an event from another thread without blocking the main thread.

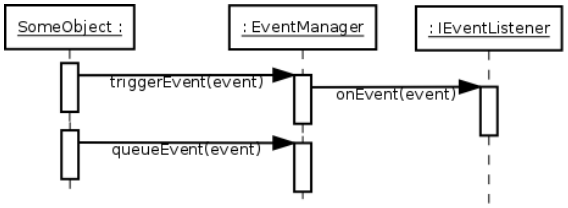


Figure 11: Sequence when triggering or queueing an event.

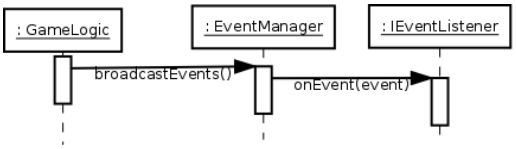


Figure 12: Sequence when broadcasting events queued.

To listen for events, the listener needs to inherit from IEventListener class and override onEvent(), then register it in the event manager for a certain event type. The listener will receive all events of the types it registers for through the onEvent() function.

To make listeners more safe, a class called ListenerHandler has been added to Pyroeis. This class handles a listener and the event types it want to listen for. It is a cleaner solution by letting the user control several event type listeners from one place. It can stop and start listening for all event types at the same time. When the handler is destroyed, the listener is automatically removed to ensure that no dead listener is in the event manager.

Because of this the handler is not copyable.

3.4.4 Physics

The physics is accessed through the IGamePhysics interface, so that the implementation is hidden. This is to be able to easily change the physics engine to either our own, or another third party library without breaking the logic. The current implementation uses Box2D as the backend.

The physics interface could originally only have one physics body per actor, but because of several features required to have more than one body, support for body parts was added to the interface. An actor now has one main body that controls the transformation of the actor, and zero to many body parts. A physics body can have either a circle shape, or a convex or concave polygon shape. The properties, such as position, density and velocity, of each body can be modified and accessed from the physics interface.

To ensure that the physics is updated at a fixed delta time, the time used on the program is accumulated until it exceeds the physics delta time. At this point the physics consumes an amount equal to the physics delta time from the time accumulation and simulates one step. [8]

Initially with Project Nox, the physics interface was meant to be a general interface for both 2D and 3D physics, but because of the differences between a 3D and 2D physics engines, it has become apparent that the interface should be specialized for 2D physics. Because of this there is a mix between 2D and 3D structures used for the interface with the current version, which should be changed to only use 2D.

3.4.5 Renderer

The renderer is also accessed through an interface, so it can support several implementations. This interface is called IGameRenderer. It uses a custom scene graph implementation to handle the object rendering. The scene graph has a transform node, and multiple render nodes, including sprite node and tile node. All objects are transformed in model space on the CPU to minimize the number draw calls, then transformed into view space and projected to the viewport on the GPU. The current implementation is based on OpenGL 3.0.

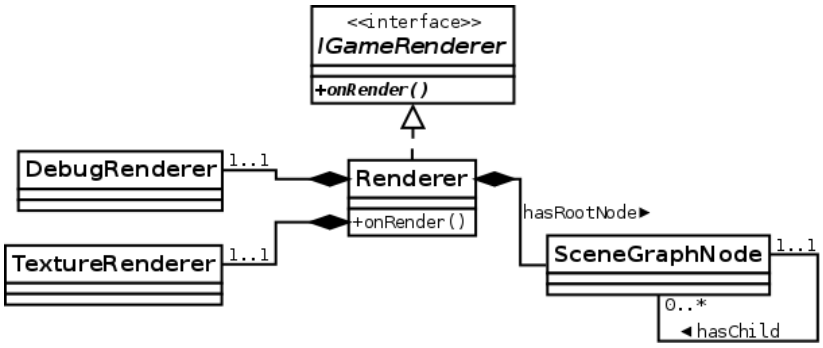


Figure 13: Class diagram of renderer subsystem.

3.4.6 Process Management

The process system is a utility system that can be used for several purposes. A process runs over several frames until it enters a finished state, or is aborted. It is handled by

a process manager and is destroyed when completed. Processes can be chained so that when a process succeeds its job, a successor in the chain can take over. This could for instance be used to play a chain of sounds or animations.

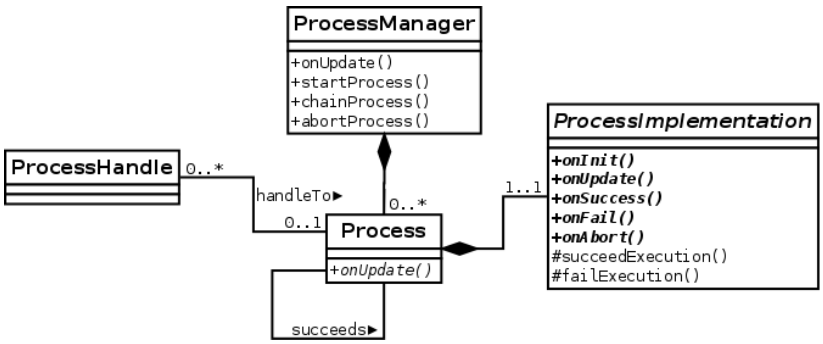


Figure 14: Process classes and their relations.

Processes can be implemented by specializing the ProcessImplementation class. This class has callbacks that are called when entering a state, or updating the process. The implementation itself has to decide through protected member functions when to finish the work by either succeeding or failing.

When starting a process in the process manager, the process is moved to the manager and a handle of the class ProcessHandle is returned. With this handle, the process can be controlled through the process manager.

3.4.7 Resource Cache

All assets loaded into the systems are managed by a resource cache. It has a fixed maximum size that cannot be exceeded. If, when loading a resource, it will exceed the maximum size, old resources will be discarded from the cache. What resource to discard is decided using a least recently used (LRU) list. When a resource is accessed, the resource is added to the front of the list. When a resource needs to be discarded, it discards the resource at the end of the list.

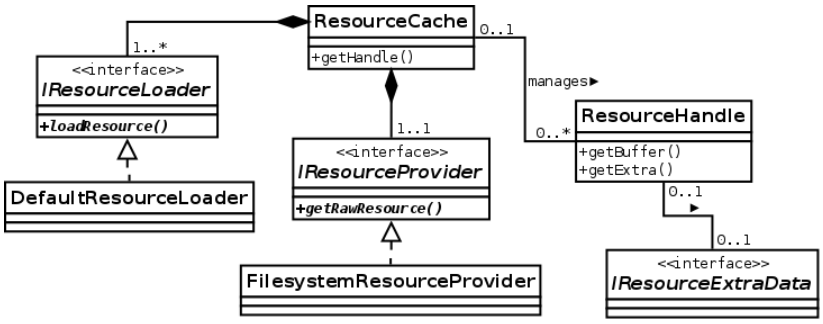


Figure 15: Class diagram for the resource cache.

Resources are stored in a resource handle of class ResourceHandle. The resource cache stores these handles in its LRU list as shared pointers, so that even if the cache were to discard a resource, it will exist as long as someone is referencing it. A handle stores the resource contents either in a raw buffer, or in a structured way, like a Json::Value. If

the resource needs more data than the buffer, or stores it in another way than a buffer, these data are stored in a specialization of the IResourceExtraData interface.

An implementation of the IResourceProvider gives the cache access to the resources on disk. The current implementation uses boost::filesystem to discover and load files from a directory, but other implementations like one loading from a compressed archive could be implemented.

To load a resource from the IResourceProvider, it uses an implementation of the IResourceLoader interface. A resource loader's job is to load a raw buffer into a resource handle. The loader provides a pattern to match files with. All files that match that pattern should be loaded by that loader. A default resource loader that matches all files is always provided, and it loads the buffer directly into the resource handle. For special files such as sound or JSON files, specialized loaders are used.

3.4.8 Audio

The audio system consists of two base classes; Audio and AudioBuffer, that are specialized into backends, and a process that plays a sound. The Audio class is the interface to the sound system and can be accessed through the logic context. Its job is to initialize and store audio buffers. An audio buffer is an audio stream buffered and ready for playback.

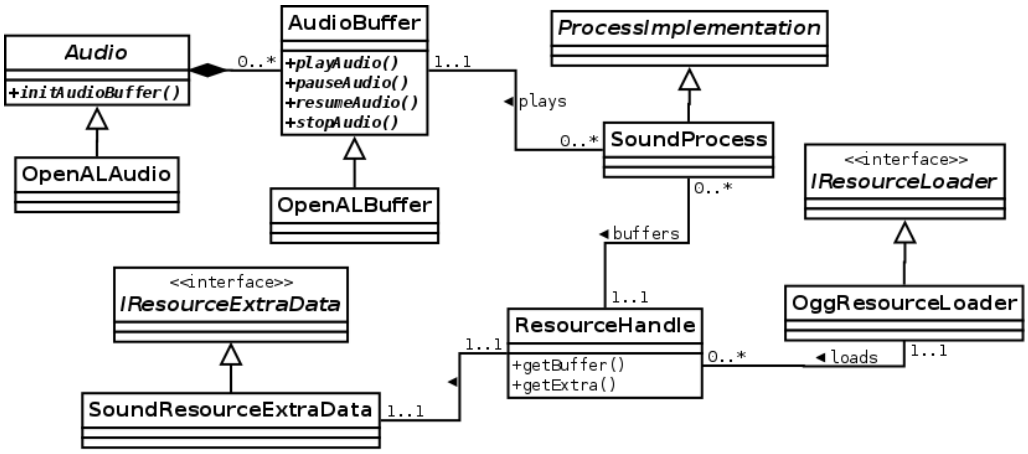


Figure 16: Audio classes and their releationships.

To play sounds, the sound process, which is specialized from ProcessImplementation, is used. The sound process is initialized with a resource handle that contains the original uncompressed audio stream. When the process is initialized, it initialized an audio buffer from the resource handle through the audio system. It then starts playing the audio buffer. When the buffer has finished playing back the audio, it cleans up the audio buffer and succeeds its execution.

A sound resource is loaded from a resource loader specialized for a specific file format. The sound is decompressed and stored in the resource handle's buffer. An extra data instance might be stored with the resource handle to provide extra data about the audio stream, such as the number of channels and bits per sample.

When initializing an audio buffer in the audio subsystem, it puts the resource handles buffer contents into the audio backend. Then when the audio buffer is played, the backend streams the audio from its audio the the hardware.

The current implementation uses OpenAL for playing sounds and Ogg Vorbis for storing and uncompressing the audio. OpenAL is supported by Apple, and a well maintained software implementation called OpenAL Soft is available for Windows, Linux and other systems. Ogg Vorbis "is a fully open, non-proprietary, patent-and-royalty-free, general-purpose compressed audio format". [9] Because of its non-proprietary nature and cross-platform support we use it to store the audio.

3.4.9 Data Storage

For a game it is necessary to not only load resources, but save them as well. This requirement was new for Pyroeis. This is the job for the data storage: storing data created at runtime, and loading these later. The data storage uses an interface called IDataStorage and is currently implemented with boost::filesystem in the DataStorageBoost class. It is initialized to one directory on the filesystem that cannot change. All paths used are relative to this directory. To find a suitable directory to store files generated by a non-administrator user, SDL_GetPrefPath() is used. [10] The directory returned is writable by the program without administrator rights.

The data storage does not cache files like the resource cache does. It is meant to be used for purposes such as saving the game progress. With later iterations, it might be merged with the resource cache so that we have a cached storage that can both read and write. The storages would still have to be split in two: one read and one read write.

3.4.10 Artificial Intelligence

Project Nox already had support for multiple AI entities that controlled actors through a special game view called `AiView`, much like how the player controls the character through the human view. This system however did not support having different types of AI. All AIs would simply attack the player on sight, and search when they could not see them.

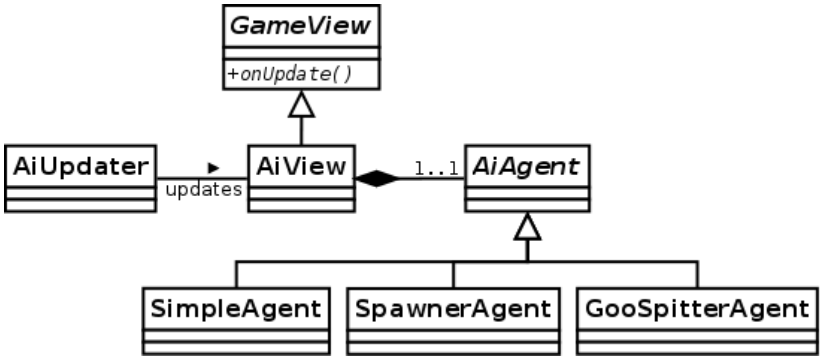


Figure 17: Class diagram of AI clases.

Pyroeis has different types of AI. Some shoot the player on sight, others flee to cover and attack the player using other methods. For this we implemented a simple Strategy-like pattern. [11] All AI views now have a specialized implementation of an AIAgent that handles the choices that the AI will make. The AIAgent base class has all the possible actions any AI can take. The individual specialization has the actual update function that is called every frame. Based on the state of the agent, the update function chooses what parent functions get called.

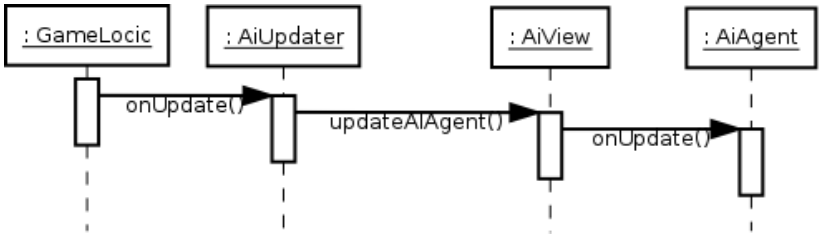


Figure 18: Sequence for updating AIs

Previously with Project Nox, all AI logic was within the AiView class, but with Pyroeis it is in the AiAgent class. To update the agent, the AiView, from its updateAiAgent() function, calls the update function on the agent. Just as in Project Nox, an AiUpdater instance updates all AIViews in multiple threads.

3.4.11 Graphical User Interface

for certain interactions a graphical user interface is necessary. The interface is provided through the human view. Project Nox had partly implemented the GUI system from an earlier project, but did not use any of it. Pyroeis took that system, completed it, and added another system to handle scenes.

View

The GUI is a hierarchy of views (not to be confused with game views). It is inspired from GUI systems such as Android's View system, which uses the same name. [12] A view is a single graphical object on the screen and a container for child views. They are either positioned and sized manually, or automatically based on parameters. For example a view that is a child of another may have a centered position meaning that it is automatically positioned in the center of its parent. To specify size, either pixels or percentage can be used.

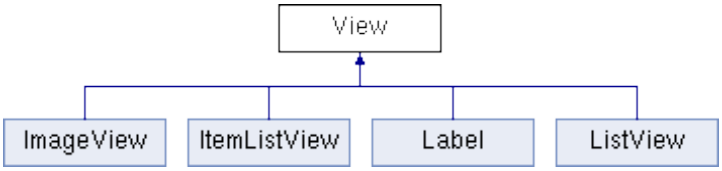


Figure 19: View class and its implementations.

Different types of views exist. The base view class is a rectangular view that by default is blank, but it can also have a color. It can also act as a button where it will trigger a callback when it is clicked. ImageView displays a single image. Label displays one line of text. ListView and ItemListView lists a series of views.

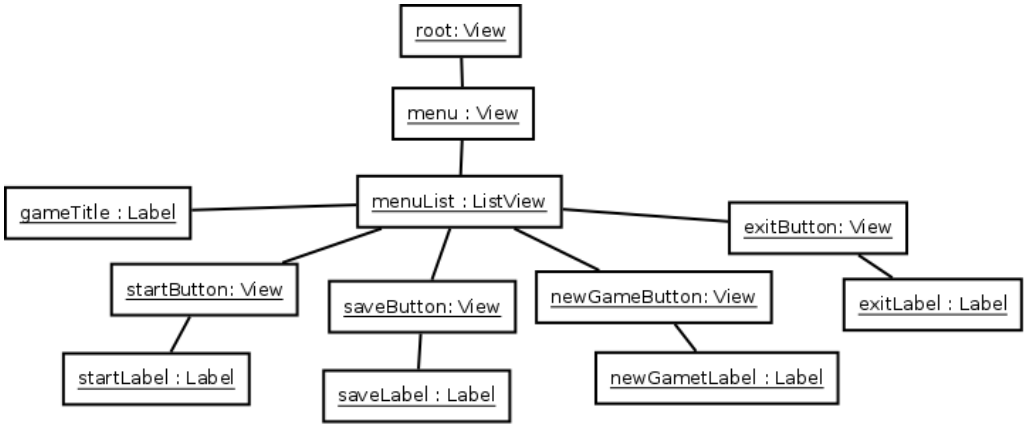


Figure 20: Example view hierarchy for the main menu. The actual implementation is more complex.

Having the button not as a separate View class implementation, but as a part of the base view, enables all views to act as button. A button has three states: default, hover and click. It can have different colors for each state, and with an ImageView it can have different images. Hover state is used when the mouse hovers over the button. It enters click state when the mouse is pressed down while hovering over it. When not in one of these states it is in the default state.

ListView lists all its children either horizontally or vertically. The children still use automatic sizing, but the position is decided by the list view. The total size of all the child views, whether using pixels or percentage, should not exceed the size of the list view since it does not change the size to automatically fit. This list view is good when different elements in the list should take a different amount of space. The list view uses an implementation of ListView::ChildPlacer to place its children. Several placers are implemented that place the children in different ways.

Figure 21: Different ChildPlacer implementations.

The other list view class called ItemListView does not list its children, but rather a set of items. When listing the items, it uses a view that wraps around the item view. Each item wrapper is equal in size and automatically resized to fit the list with a specified spacing. The item views contained in the wrapper views are positioned like normal as if the wrapper views were their parents. This list type is good when the whole list view always should be filled, while items may be added or removed. This list view uses a system for placing the items similar to that of ListView::ChildPlacer.

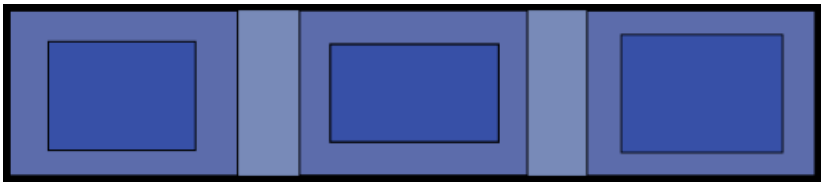


Figure 22: ItemListView. Light blue is the list. Darker blue is each item wrapper. Darkest blue is the actual item view. The item wrappers have equal size.

With Pyroeis a new functionality required for the inventory was added to the view system: tooltips. A tooltip can be set on any view and will show when the cursor hovers over the view. The tooltip is a normal view and is used like a normal view, except for its positioning, which as a tooltip is relative to the view it is attached to.

Scene

At different times, different GUI might be appropriate, such as when using an inventory, or when browsing the main menu. To handle this in a clean way, scenes are used together with a scene manager. One scene represents a single state of the interface, and only one is active at the time. With the current game implementations the main menu, game, inventory, weapon customization, game over and game won are separate scenes.

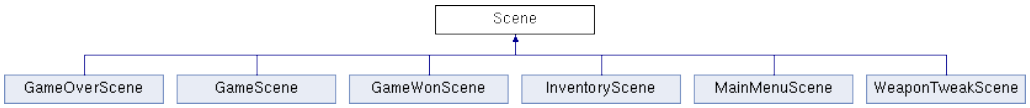


Figure 23: Scene class and its implementations.

The scene system is inspired from Android's Activity system where only one activity is active at a time, and they are in a certain state. [13] When a scene is added to the scene manager, it enters the created state. When it is removed, it enters the destroyed state. When coming into the foreground (e.g. pressing "play" in menu makes game scene enter foreground), it enters the active state. When leaving the foreground it enters the created state.

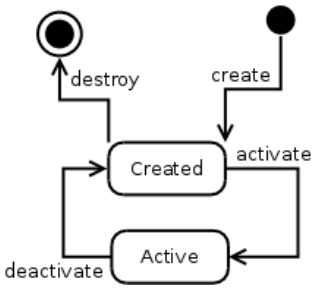


Figure 24: Scene states and transitions.

An active scene receives all keyboard and mouse input from the human view, so that it can handle the GUI or player interaction. The human view stores the root GUI view that represents the whole screen. When a scene that uses GUI is activated, it appends its own view onto the root view, and removes the view when deactivated so that it won't conflict with the other scenes' GUI.

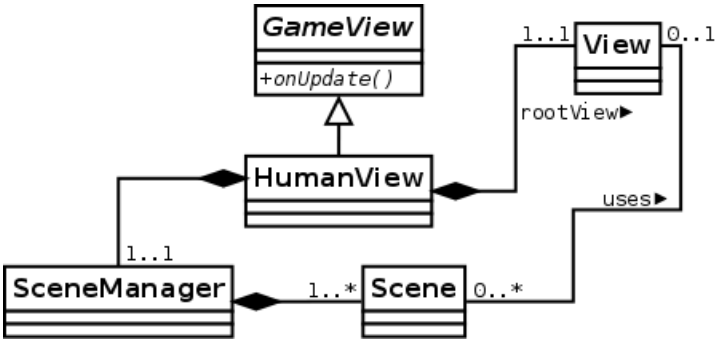


Figure 25: Class diagram of HumanView with its scens and root view.

3.4.12 Logging

The logging system was created with Project Nox, but has been enhanced with some minor functionality. A short description can be found in Appendix A. This section will go into a bit more detail.

The system provides a class called Logger for output. This should be used rather than C/C++ IO functionality like printf and std::cout. When printing a message to the logger, it uses the same format as printf. It writes to both the standard output and a file in the data storage directory. Five categories of logging are available: info, verbose, warning, error and debug. Info output should provide informative messages to the user. Verbose output provides extra information that normally is not necessary, but could provide useful information for the user. Warning warns when something might have dangerous consequences. Debug outputs debug information used for debugging the program. Error outputs messages when something wrong happens. For later iterations, a fatal category could be added to separate errors that crash the program, and those that don't. The logger can selectively enable or disable the output of a category. By default info, warning and error are enabled.

The system provides a global Logger instance that is accessible to everyone. Normally, global objects are avoided, but in this case it is more helpful than problematic since it eases outputting information by the need of only including one header file and using one object. It is not a singleton, so new instances can be created. This can be useful if a subsytem wants to output more isolated messages to a separate file.

3.4.13 World

The world handling was not part of Project Nox since it had all of its actors loaded all the time. With Pyroeis there were several new challenges with an infinite world, so world handling was made part of the system. An instance of the WorldHandler class manages the parts of the world. A part is one chunk of class Chunk, and a chunk loader of class ChunkLoader handles the loading and unloading of them.

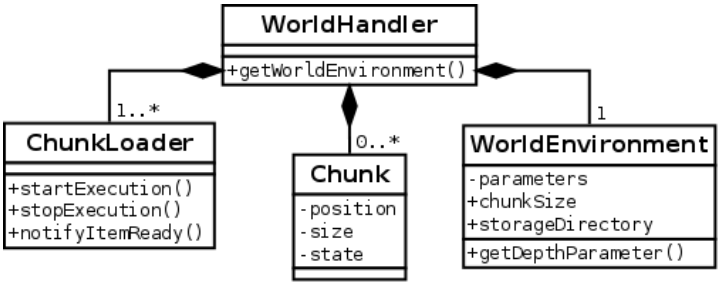


Figure 26: World handling class diagram.

The details of how the system works and is implemented is discussed in Chapter 5.5.

4 Development Process

4.1 Working Hours

For the project period we decided on a strict rule that we would be present at our office from 08:00 to 16:00 each workday. This was to ensure that we at least worked 40 hours a week, and to enhance the teamwork. Working in a team from different locations is possible, but diminishes our teamwork and performance. It is important that we are synchronized and are available to discuss issues.

4.2 Development Tools

4.2.1 Version Control

Both are experienced with git as a version control system, since this has been used for previous projects. We haven't found any other systems that provide better functionality, and the system has to be distributed so that we easily can create and manage branches and are not dependent on a central remote repository. An alternative is Mercurial. It provides similar functionality as git, but as we have experience with git and not with Mercurial, the decision to use git was made.

With git we needed a remote repository so we could synchronize our versions. For Project Nox we used Atlassian's Bitbucket to host our git repository. It works well and lets us have private repositories, which is important for this project. The choice was therefore made to use Bitbucket also for this project.

4.2.2 Project Management

In addition to versioning the source code, we needed to manage our project's tasks. Atlassian's Jira was chosen to do this since it is well integrated with Bitbucket and it has a plugin to handle agile methodologies, specifically Scrum. In Jira all tasks are managed: Features, bugs and enhancements. In addition it also manages the sprints so that we easily can set up a sprint, start it, and pick tasks to complete.

4.2.3 Coding Environment

Linux

In Linux we primarily used Eclipse CDT to code and debug the code, but for building the release build we used CMake. With CMake we had more control of exactly how the binary should be compiled and linked. Valgrind was used to track down uninitialized variables and memory leaks, while perf together with graphing tools were used to profile the application.

OS X

Apple's Xcode was used for developing and debugging on OS X. As Xcode is very well integrated with the tools found in Apple's Instruments it was very convenient to use the profiler here to track down performance issues.

Windows

Although most of the development was done on either Linux or OS X, some issues for the windows specific version had to be fixed. For this we used Visual Studio 2013.

4.3 Project Workflow

4.3.1 Scrum

The workflow mostly follows a standard scrum workflow. Every morning a daily scrum is arranged where we tell each other what we did and what we will do until next time, and write this down. By having this talk and writing it down we both get a clear idea of how both are doing and what we will do today. It is also easy to see your daily progress by comparing what was planned and what was done. Our daily scrum is not as strict as it normally is. Sometimes it can stretch to an hour long discussion about a feature or design choices. This would be a problem for larger teams since the discussion would only waste the time of some of the team member, but for a two-man team, it works very well.

Each Friday, the sprint is reviewed and the next is planned. This often takes several hours. During the review, the new iteration of the game is playtested by us. Sometimes experimental features, like networking, are updated and tested. The result is compared with the tasks planned and a discussion is written down. The reviews can be found in Appendix K. After reviewing, a short retrospective meeting is held were we discuss things that is bad with the process and what should change.

After doing the sprint review and retrospective, we start planning the next sprint. Normally the product backlog would be filled with tasks from by product owner and we would estimate and prioritize the tasks, but we don't have a product owner. Because of this and because the flexible nature of the game design, tasks are added to the backlog underways. With each planning meeting, the current completed tasks are compared with the current milestone goals. Then tasks are created for the most important milestone goals and added to the backlog in prioritized order. Task lengths are estimated until enough time to fill a sprint is estimated. Top tasks fitting the sprint length is added to the sprint, and then it starts.

4.3.2 Milestones

In addition to the sprints, we review milestones. With each milestone a review process, alike the sprint review, is held, after the sprint is reviewed. The current implementation is compared with the goal of the milestone and an evaluation of the progress is done. The review tells us how we are progressing on a larger scale. If some goals for the milestone were not met, an evaluation for the next milestones has to made. If the goal is important it needs to be shifted to a later milestone. If not it might be discarded. The next milestones' goals are reviewed to possibly removed or add new goals.

4.4 Development Workflow

4.4.1 Source Code

During Project Nox, a workflow of taking a feature, implementing it, getting it reviewed and merged with the rest, was developed. Since not all were experienced with git, a manual on the workflow and how to implement it with git, was made (see Appendix B). For Pyroeis we expanded on the same workflow, making it fit our tools, team size and expertise.

During a sprint, we use the scrum task board to pick a feature to implement. The ones picked are based on priority and interest. When a task is picked, it is transitioned into the “in progress” state from “to do”.

Implementing a task is always done in a separate branch. No commits should be pushed to the remote master branch, except if they are trivial fixes like a grammatical error. To organize the branches they are separated into categories: feature, bug, enhancement and experimental. This is done by using the format “<category>/<task>”, just like a directory on the file system. For example,. a new feature that adds a lamp will be named “feature/lamp”. All the development is done on this branch. If the code needs to be synchronized with other computers it can be pushed to the remote repository.

Branches should focus on a small area. Often a feature can be split into several branches. For example, the lamp feature requires light functionality, which might not be implemented. The light functionality is then developed in a separate feature branch, which can be merge before the lamp if it is needed. This is useful if another developer needs the light feature, but not necessary the lamp. For example, another developer might implement a flare feature. Rather than wait for the lamp feature to be complete, they can branch from the light feature branch and develop the flare.

When the developer is finished with a task, the task branch is pushed to the remote repository, a pull request into master is created and the issue in Jira is transitioned into “in review”. Another developer is set as the reviewer of the pull request and is notified of this. When the reviewer has time, preferably before the end of the sprint, they pull the branch, builds the binary and tests the new change. If the code builds, meets the task criteria and design, doesn’t crash or have performance issues and follows the conventions, the pull request is accepted and the branch is merged into master. The task is then transitioned to “done”.

4.4.2 Assets

During the pre-project planning we decided to create and manage a prioritized list of assets, based on Jayson Mackie’s tip. An exported version of the asset list is found in Appendix H The list contains a list of all assets that need to be implemented. Each list item has a name, category, type, priority and status. The list is always sorted by priority so that the most important asset can be created first. This is very useful for our part-time artist, and for us to easily say what needs to be done.

All assets has a current state that initially is nothing. When a placeholder is used in the game, the state changes to “placeholder”. When the artist start working on an asset, it is changed to “in progress”. When the asset is ready, it becomes “ready”, and when it is in the game: “done”. New assets are added to the list during the development of a new feature. For example if a developer start working on a new monster, the assets needed for that monster is added to the list.

5 Implementation

5.1 Player Movement

When we were designing the system to be used for the player movement, we wanted to make sure that the result was interesting gameplay and also that the design fit our game engine. Our game engine is heavily reliant on the Box2D physics engine. Box2D handles all the simulation and physics in the game. Some limitations are imposed because of this. Any altering of positions or velocities is not advised when using Box2D as this can break the simulation and cause undefined behaviour. Box2D allows users to add forces, impulses and torque in order to alter the position, speed and rotation of the objects in the world.

When designing the movement of the character one of the focuses was that the gameplay would feature weight and momentum for the player movement. Box2D's system of only affecting the forces that again affect the velocities and positions of objects in the world is well suited for this purpose.

The character in the game has two modes of movement: Walking and using the jetpack. Players can toggle between the two at any time. When the character is in walk mode, the rotation of the player is locked to the upright position and the (default) A and D keys are used to move the player left and left and right. Pressing the (default) C key will toggle the jetpack mode and let the player control the jetpack, in this mode, the (default) A and D keys control the tilt of the character and holding the (default) space key will make the jetpack fire at full force. The jetpack will push the character in the direction they currently are tilted. When the character lands on the ground and the difference in the angle to the upright position is small enough they will automatically be straightened and stand on the ground.

The character consists of the following parts: a collision body, this is the main component of the player and handles the collision and most of the physical simulation of the character. The collision body is weighted to make the feel of the characters movement correct. A Wheel that represents the feet of the player. The wheel is used to move the character when it is on the ground. To do this we add torque to the wheel in the clockwise or counterclockwise direction depending on the direction the player wants to move. Having a wheel represent the player instead of adding the force directly to the player's collision body has several benefits in our case. We want the movement to be weighted so the player feels like they have to put some meaning behind the movements they are enforcing onto the character. We want the momentum of the character to matter, in order for the player to feel that the character has weight and is an actual object in the world. This also helps when the character is moving up a hill for example, the character will move slower and needs to conserve the momentum. The final part of the player's body is a sensor that sits at the bottom of the feet. This sensor is used to register when the player is touching the ground. For this we use the contact callback implemented in Box2D. We set up a listener for this and increment a counter on the sensor. If the sensor has more than one contact we know that the player is on the ground.

5.2 Items

In any game, items are an essential part of the experience. For Pyroeis we wanted players to be able to gather items in the environment and use them to enhance their character. For the implementation we wanted to be as generic as possible. The inventory is a container, and a container should be something that all actors in a game could possibly have. To allow this the actor component system is used for the inventory. Using the actor component system is inherent to the the design in our engine. The inventory component allows actors to store, gain and lose items.

The player characters inventory also needs to be presented to the player. This is done using the InventoryScene class. The inventory scene is a part of the scene system. After the inventory scene is created, the actor being controlled by the player is set as the controlled actor for that scene. The scene updates the items in its inventory by accessing the items through the inventory component of the controlled actor.

The inventory scene also creates a weapon upgrade scene for each of the weapons that the player has. This is done when the weapons are created. For each actor that is created it is checked to see if it has a weapon power component. If it has the component, the weapon upgrade scene is created. There are some problems with the current system. A view is created for any actor that has a weapon power component. This means that if one of the enemies had a weapon with a weapon power component, a scene would be created every time such a weapon is created. This could easily be fixed by checking that the weapon belongs to the actor that is controlled by the inventory scene.

The inventory scene functions as the player's interface to the character's inventory. Through the inventory, players are able to access the crafting system. The crafting system allows players to use the resources they have collected in the world to create items. To accomplish this, the inventory scene has functions that allow an item to be created at a cost. The cost of these items are stored in a file in the assets folder and loaded as the inventory is created. Currently the inventory scene is responsible for handling the crafting, telling the inventory component what items to add and what to remove. As the inventory scene is created, the configuration for the inventory is read from a JSON file. Using this information, the inventory scene creates buttons for each of the crafting possibilities that are available. A callback is set for each button that contains information about what item is crafted and what the cost is. This is not a flexible system, as it only allows crafting from the inventory scene. The crafting logic and data should not be in the human interface, but rather be part of the logic, thus it should be recreated to use the component system. For this, a crafting component would be needed that would keep track of the different costs of creating an item. This would allow objects, like a workbench or anvil, the ability to let players craft items by using them. It also allows AIs to craft items.

In order to spawn dust in the world, a debris component is used. The debris component specifies what type of debris that should spawn, and at what frequency it should spawn. The debris component will listen to events of specific types and spawn the debris when these are received. For example, the world has a debris component that spawns debris when a part of the world is destroyed, and monsters spawn debris when they die. Any actor can be used as debris, which creates a flexible system. Dust is a type of debris that is spawned from destroying the world and killing plants and enemies. Players pick up the dust using an item attraction component to make the items move towards the

player, and a pick-up component to pick up the item and put it in the inventory. The attraction component is a simple component that finds all actors of a type, using the type component, and then applies a force towards the owning actor's position. The pick-up component uses a physics sensor box that is a few centimeters larger than the main body, and picks up any actor matching a type when it collides with the sensor. When picking up an actor, an event is generated, and the actor is removed. This event is picked up by the inventory, and it increments its count.

5.3 Equipment

Players are able to wield equipment through using the `EquipmentWieldComponent`. As all equipment is a self contained actor the equipment wield component is needed in order to keep track of and interact with the equipment. When the player uses the current equipment a check is done to find the type of the equipment. The engine currently supports three types of equipment: throwable, placeable and useable. This allows the system to operate on a number of different types of equipment. Adding more types would only require adding to the already existing system. There are currently four types of equipment in Pyroeis: bombs, flares, wall lights and health packs.

5.3.1 Bombs

Bombs are a throwable equipment type. They will stick to objects in the world if the collision between them is strong enough. This is done by setting the stickiness value for the physics component to something higher than zero. If the stickiness is higher than zero the object is added to a map that contains all the sticky bodies. Whenever a collision occurs in the world, Box2D's contact listener is used to check if the collision that occurred involved a sticky object. The check is done in the post-solve step of the physics. This means that all the forces and impulses for the contacts that occurred in the last physics step have already been calculated. These are then compared to the stickiness of the bomb itself. If the impulses surpass the required impulse to stick, calculated from the stickiness, a joint will be created for the two objects that collided.

The player is able to trigger the bombs to explode. This is done using the `BombTrigger-Component`. When the player presses the button to trigger the bombs, the bomb trigger component sends a signal to all the bombs that are within its range. In order to simulate the signal traveling from the player to the bombs a small delay is added that is based on the distance. A small random amount is also added on top of that to simulate imperfections in the fuse on the bomb. One major problem with this is that the bombs can be triggered by anything that has a bomb trigger component. This is not a problem in the game as it is now, because the player is the only actor that has a bomb trigger component. It does create a severe problem if we want enemies to be able to use bombs, or we want to add multiplayer. An easy fix for this is to add frequencies to the bombs. This would mean that a bomb could only be triggered by a bomb trigger component that matched the frequency.

After the bomb has been triggered, the fuse starts burning. When it reaches zero, the bomb explodes. The explosion will damage and apply an impulse to actors near it, it will also destroy any part of the world that is near enough. Damage is handled by the `ExplosionDamageComponent`. The damage decreases exponentially the further away from the explosion an object is. The `BombComponent` will add the impulse to any

object that is within range. The impulse is larger if the object is close to the center of the explosion and decreases exponentially the further away the object is. The bomb component also sends out a world destruction event. The event contains information about the scale, rotation and shape of the destruction. The `OpenWorldComponent` of the world actor listens to this event and tells the world generator to apply the destruction to the world. Data about what parts of the world are destroyed is returned. This is then used to send out another event: the destructed area event. The debris component of the world actor listens to this, and create debris, based on the area that was destroyed.

5.3.2 Wall Lights

The wall lights provide the player with permanent light sources. They are of the placeable equipment type. The `LightRenderComponent` adds lights to the scene. A light render component can add numerous light sources to the world. It also provides two distinct types of light. Lights can either be blocked by the physical objects in the world, or not blocked. This is specified by the `castsShadows` boolean property. When the light should cast shadows in the scene, the visible area of that light needs to be calculated. This is explained in section 5.10. Where as lights that are not casting shadows simply need to be rendered into the light map. For light that is not casting shadows, it's shape is created and updated by the light render component.

5.3.3 Flare

The flare provide temporary light for the player to see dark areas and are throwable. Like the wall light, the light is added using the light render component. The flare's physics is fully simulated to allow players to throw them over and under obstructions. The `LightFadeComponent` allow the light to fade out as the flare burns up. After the flares burn time is over, the light fade component starts decreasing the range of the light until it reaches zero. The light is then disabled, and no longer a part of the calculation.

5.3.4 Health Pack

Health packs allow the player to regain health and is of the use equipment type. The health pack uses the `HealthRegainUseComponent` that inherits from the `UseComponent`. When the player uses the health component it will check if the player already has full health. If they do not, the health component will restore an amount decided by the JSON file for the health pack actor. The health component will never restore an actor's health above full.

5.4 Weapons

Weapons in Pyroeis are implemented by having them as sub-actors of the wielding actor. Any actor can have a weapon wield component, allowing the actor to control different weapons. The weapon wield component is implemented to allow actors to handle any number of weapons. It handles the switching between weapons, aiming the weapon, and pulling the trigger. All weapons in the game are actors. This allows us to easily modify the weapons however we want, and also for us to abstract the weapons to a degree that allows for flexible and varying weapons. For example, when the player pulls the trigger on a weapon that is semi-automatic, the weapon will only fire once, while on a weapon that is fully automatic, the weapon will keep firing until the trigger is released.

Triggering a weapon requires several steps through the system, as shown in Figure 27.

It starts with the game application receiving a mouse press event from `SDL_PollEvent()`. This is passed directly to the human view, where it converts the data to a suitable format for the scene system before passing it to the scene manager. The scene manager, in play mode, sees that the press event should send a "WEAPON_TRIGGER" control event, creates the event, and triggers it through the event manager. The event manager sends the event to the control component of the player actor, where it is parsed. The control component tells the weapon wield component of the actor to pull the trigger, and the weapon wield component pulls the trigger on the weapon component.

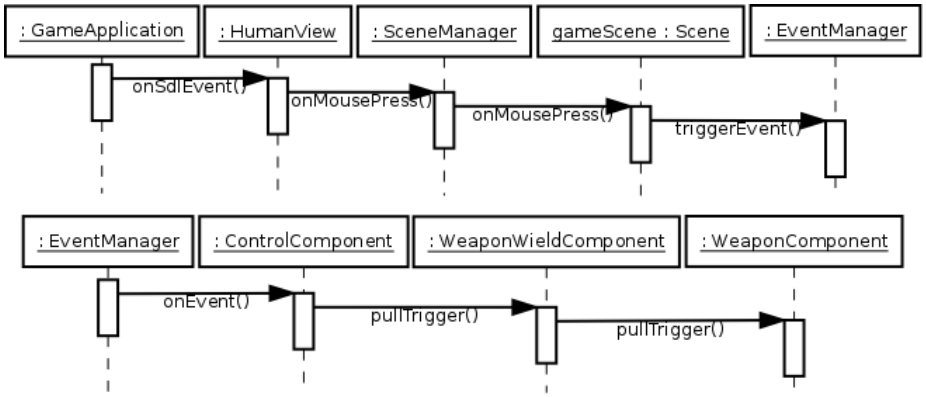


Figure 27: Sequence diagram for firing a weapon. Split into two parts to fit the text width. The event manager in the top and bottom sequence is the same and is part of the same sequence.

Weapons can also be upgraded by the player. To allow this, each weapon has its own weapon tweak scene. These can be accessed through the inventory by clicking the weapon to tweak. To keep the abstraction of the weapons, and the support for weapons that are not upgradeable, we have implemented the weapon upgrading system through the weapon power component. This works together with the weapon's base properties to create the final properties of the weapon.

When weapons are fired, they create bullet actors in the world. These actor are separated from the weapons entirely, any weapon in the game, could potentially fire any actor we wanted. This means, we could potentially have a weapon that fired friendly AI characters. One of the later changes we made in the game was to add damage from collisions on actors that have the collision damage component. One problem we have with this is that the damage is also applied when the actors are hit by bullets. Although this seems natural, it creates an issue when balancing the game. Bullets apply damage when they hit an object, and the actor that are hit apply damage to themselves from the collision response.

The current implementation of the saving system has one major flaw in the way that it handles weapons. When players save the game, all the weapons are saved. The problem is that the relationship between the actor controlling the weapon through its weapon wield component is not saved. This means that every time you load, the world, the old weapons will be there, but since they are not related to a weapon wield component, they are essentially only taking up memory space. Initially they were rendered as weapons hovering in the sky, but an easy fix to hide this bug was to only enable the rendering if

they are wielded. New weapons will be created for the player which also means that any upgrades the player has applied to the weapons are lost. The fix for this problem would be to implement a proper sub-actor functionality as part of the actor system, where the weapons are sub-actors of the wielding actor and not only of the weapon wield component. With this the weapon could be saved as sub-actors and restored as sub-actors. Listing 5.1 shows an example for a format that could be used for a sub-actor system. Sub-actors would be global to the actor and defined in "subactors" array. Each actor in the array has the same format as other actors, thus they can easily extend pre-defined actors. The weapon wield component's "weapon" property refers to a sub-actor of the same name, and links to it when it loads. The sub-actors are created by the actor system before the components are loaded.

Listing 5.1: Example JSON file for an actor with sub-actor system.

```
{
  "name": "Astronaught",
  "subactors":
  {
    "handgun":
    {
      "extend": "actor/HandGun"
    },
    ...
  }
  "components":
  [
    {
      "name": "WeaponWieldComponent",
      "weaponSlots":
      [
        {
          "slot": 1,
          "weapon": "handgun"
        },
        ...
      ]
    },
    ...
  ]
}
```

5.5 Handling the World

5.5.1 Infinite Worlds

One of the most important requirements in this game is that the worlds are infinite in size. The design document in Appendix E says: "The cave system will have a finite top point which is the surface, and a finite bottom point which is the core of the planet. Left and right go to infinity." This requires the world to only be infinite to the left and right, but for the implementation we changed the requirement to be infinite in all directions. There are two reasons to this: The first is that we are creating a game engine, and the more flexible the implementation is, the easier it will be to tweak the game design or add

new features. The second reason is that there are no real difference between left, right, up and down, so if we can go infinitely to the left and right, it should not be any different going up and down.

Making the world infinite in size is not too difficult to accomplish. The biggest problem with having infinite worlds, is that you can't possibly have all of the world in RAM, or generate all of it at the same time. The solution to this is to split the world into independent chunks, placed in a grid, where each chunk is one cell. The idea of splitting into chunks is inspired from the game Minecraft, where the same idea is used in a 3D block-based world. [14] This allows us to only keep the chunks that are currently visible, or close to visible, in RAM. A rectangular area called the "loaded area" is used to find these chunks. The area is 200 meters wider and taller than the player's visible area (the game window) to ensure that chunks are completely generated before they are visible to the player.

A problem with having a large area of the world active is that it requires larger amounts of CPU processing power to simulate the world actors. This is solved by defining a smaller area that still is larger than the screen, called the "active area". It is 100 meters wider and taller than the visible area. Only actors within the active area are handled by the game engine logic. This means that actors outside this area won't be updated or simulated, thus requiring close to zero processing power.

As the player keeps moving out, the generated world will get larger and take up more memory space. A chunk doesn't occupy much memory, and the computers today have gigabytes of memory available, so we can keep a lot of chunks in memory at the same time. Since loading the chunks into the physics from memory is faster than generating them over again, we keep an area larger than the loaded area of chunks in memory called "unloaded area". All chunks outside this area are stored on in the filesystem.

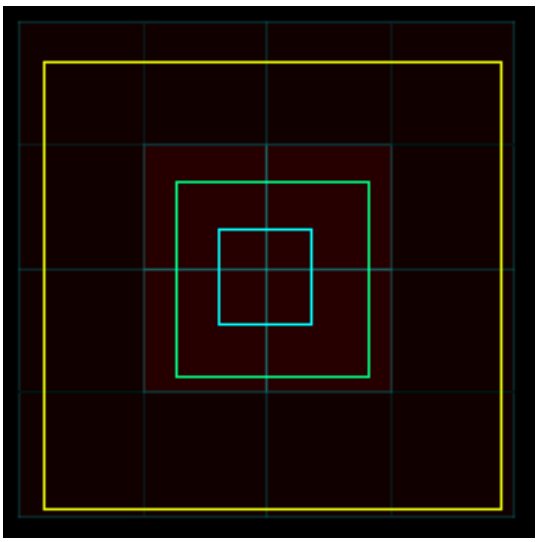


Figure 28: Shows the different areas. Yellow square is the unloaded area. Green square is the loaded area. Blue square is the active area.

All of the areas' positions are based on the center position of the camera. To prevent too many chunk changes happening at the same time, the areas themselves only move after the camera have moved a certain distance from its previous position. The loaded

area requires a smaller distance than the unloaded area, and the active area requires a smaller distance than the loaded area. This is because the inner areas are smaller and therefore need to move more often so that the camera won't go outside an area. When the unloaded area moves one step, the loaded area have moved about 3-4 steps, and the active area have moved 6-8 steps.

5.5.2 Managing the Chunks

Since the world handling is a central part of the game logic, it is implemented as a subsystem of the game logic. See Figure 26 for the class diagram. It is a class called WorldHandler that has the responsibility to handle the world areas (unloaded, loaded and active). To do this it uses techniques described Subsection 5.5.1: Splitting the world into chunks and using different world areas to decide what should be loaded and what should not. The unloaded area is not implemented in the current iteration of the product.

Each chunk is an instance of the class Chunk. A chunk has a certain position and is in a certain state. The position of a chunk is not the world position, but the cell position. So a chunk with size (10, 10) at position (3, -5) will be at the world position (30, -50). All chunks are stored in a map structure with a key made from stringifying its position. For example a chunk at position (3, -5) will be mapped from the string "3x-5". This makes it fast to access a chunk at a certain position in the world.

A chunk is in a certain state. These states are mirrored from the world areas. The state can only change from active to loaded, loaded to unloaded, and vice versa. Each chunk also has a "new state" that is set by the world handler. This state indicates which state the chunk should transition into.

Figure 29: The chunk life cycle

When the position of the camera changes, the world handler updates its world areas. When a chunk enters or leaves an area, the new state is set for the chunk and a state change is pushed on a queue. A chunk loader that runs on a separate thread gets a signal when this happens and starts processing all the state changes on the queue. Changing the chunk state, meaning loading or unloading all the actors and terrain, is a slow process and can therefore not be done on the main thread. It would completely freeze the logic and rendering for a few frames.

When the chunk state change is processed by the chunk loader, it does the operations that are needed to transition it to the new state. If the state change goes from unloaded to active, or vice versa, it first transitions to the loaded state before transitioning to the final state. For each of these transitions there is triggered an event. This event can be received by other subsystems to execute some action. One of these systems is the world generator described in Subsection .

The chunk loader manages the saving and loading of actors in a chunk. To save the actors within a chunk, it has to find all actors positioned inside the chunk perimeters without blocking the main thread. To do this, all actors that are set to inactive are stored in the chunk they are positioned in. When the chunk transitions into the unloaded state, all these actors are written to a file and removed from the logic. When chunks transition back to the loaded state, these actors are parsed and re-created. This save system requires

that all actors within the chunk are set to inactive before it is saved, but with the small size of the active area compared to the unloaded area, this is not a problem.

5.5.3 Saving the World

Chunks are saved to file when they transition to the unloaded state, but they can also be saved while still being kept in the active or loaded state. This is used in cases where it is necessary to save the currently active world, such as when a player presses the save button. In the current state of Pyroeis chunks never transition into the unloaded state, so when a player saves the world, all chunks that have been loaded are saved this way.

Saving the world sorts all the world actors into the chunks they are positioned in. Then for each chunk, all actors, and the views controlling them, are written. For each chunk saved, an event is triggered so that other systems might save their data. All this runs in a separate thread so that it won't block the main thread, but the game logic is completely paused until this is done.

5.6 Destroying the World

One of the most core and complex features in the game is the world destruction. The player should be able to destroy the terrain, which refers to the ground made of soil, rock and other materials.

What we didn't initially think of is that the destructions can be used for cave generation. The initial thought was that the destruction and generation are two different topics that have different solutions, but in our first meeting with Jayson David Mackie, PhD student at Gjøvik University College, he saw a clear link between the two. The caves can be created by applying destructions over a cave path, almost like how man-made tunnels are created with explosives. This lets us use the same technique and code for two problems, thus reducing our required research and implementation effort.

5.6.1 Designing the Terrain

In games like Terraria [15], the terrain is destructible, but it consists of a number of small blocks forming it. Other games, like the Worms series [16], has a terrain acting like our design, where it all is destructible. Worms' terrain is also formed by blocks, but these blocks are as small as screen pixels, making the blocks invisible to the human eye. The problem with this method is that it is block-based where each block either exists or doesn't, so that removing all the blocks within a shape destroys the terrain in that shape. This block-based terrain won't let us easily simulate the realistic physics required for our game, and it is not in any way compatible with Box2D since it is polygon and circle based.

Because of the incompatibilities with the methods described above, we decided to do a pure polygon based terrain where polygons can be applied as destructions. Several hours of research had to be done, and after several iterations we ended up with a terrain that could be destroyed by any non-complex polygon everywhere.

5.6.2 Destructions – First Iteration: Holes

Two web articles [17][18] written by Emanuele Feronato describe a technique to create a destructible terrain with Box2D. Still this technique has one important limitation: No interior holes can be created in a polygon, so the terrain has to be split into a grid where destructions never can be completely within one cell. If we were to have small

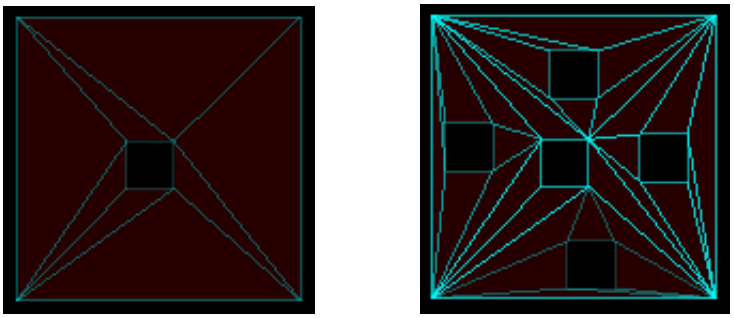
destructions, the grid cells would have to be as small as a few centimeters or millimeters, and the number of cells would be high, making the physics run slowly because of the number of polygons. It would also require us to ensure that all destructions applied are larger than a cell. To counter this limitation, holes had to be supported. The library poly2tri can convert polygons with holes into triangles, so we decided to use this. [19]

While we now can triangulate polygons with holes – giving us the ability to create holes in the terrain – we could not handle holes intersecting multiple polygons since the holes applied with poly2tri can't touch the polyline boundary. To make sure that the holes don't touch the boundary we use the boost::geometry library. [20] This library supports several operations on different geometries. Using boost::geometry::intersects and boost::geometry::within we could check if a terrain polygon intersected the destruction and if all the points of the destruction polygon is within the terrain polygon. [21] [22]

To be able to more rapidly test the code, the terrain is tightly connected with the physics and has access to all its functionality and data. This is not a good practice for production, but for testing and research purposes it greatly improves the efficiency of implementation.

This algorithm for destruction is as follows:

1. Query what physics fixtures intersect the AABB box of the destruction polygon.
2. Check each fixture if they intersect the destruction by converting the fixture shape into a boost::geometry represented polygon and calling `boost::geometry::intersection()`.
3. If a fixture intersects, check if all destruction polygon points are inside the fixture polygon with `boost::geometry::within()`.
4. If the destruction is completely within a fixture, convert the fixture and destruction to poly2tri before triangulating the fixture together with the destruction hole.
5. Create a new fixture for each triangle created with poly2tri.



(a) One destruction.

(b) Four more holes. Cyan lines are stronger since the old fixtures weren't removed.

Figure 30: Destructions applied. Polygon edges are represented with cyan lines.

5.6.3 Destructions – Second Iteration: Destructions Everywhere

What remained after the first iteration was handling destructions that intersected several terrain polygons. To do this, each terrain polygon has to be handled independently. Since poly2tri can't handle holes that touch the polyline boundary, the same method can't be used for this case.

The solution for this case was the difference $terrainpolygon - destructionpolygon$, as described in the documentation of boost::geometry::difference. [23] The result of this operation is the set of polygons representing the area of the terrain that did not intersect the destruction. The boost::geometry library provided us with this functionality.

Figure 31 shows how the difference $green - blue$ is calculated, resulting in the polygons indicated by orange dots.

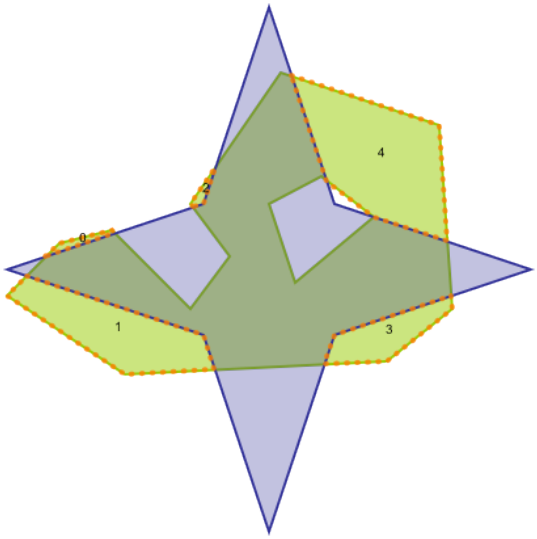


Figure 31: Difference operation. boost::geometry::difference[23]

After the difference is calculated, the difference has to replace the old terrain polygon. The polygons returned by the difference algorithm might not be convex, which is a problem when using Box2D since it only supports convex polygons. To solve this, all concave polygons have to be split into convex polygons.

Our first solution was to use the b2Separator class, described in a web article by Emanuele Feronato, to separate the polygons directly into convex Box2D fixtures. [24] [18] This class has some issues and it requires us to include yet another library and convert each polygon to b2Separator's polygon structure.

Our final solution was to replace b2Separator with what we already are using: poly2tri. It works well, but results into a lot more polygons since it creates triangles rather than convex polygons. Using poly2tri also made the code easier to maintain in later iterations.

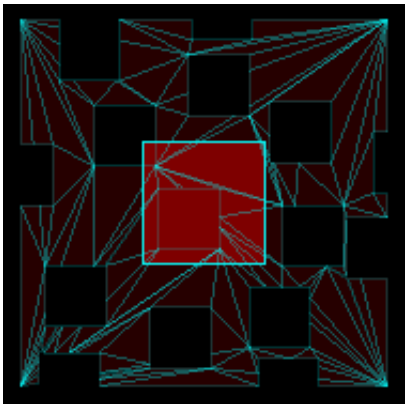


Figure 32: Holes intersecting several fixtures. Center square is a minor bug that later was resolved.

This algorithm for destruction is as follows:

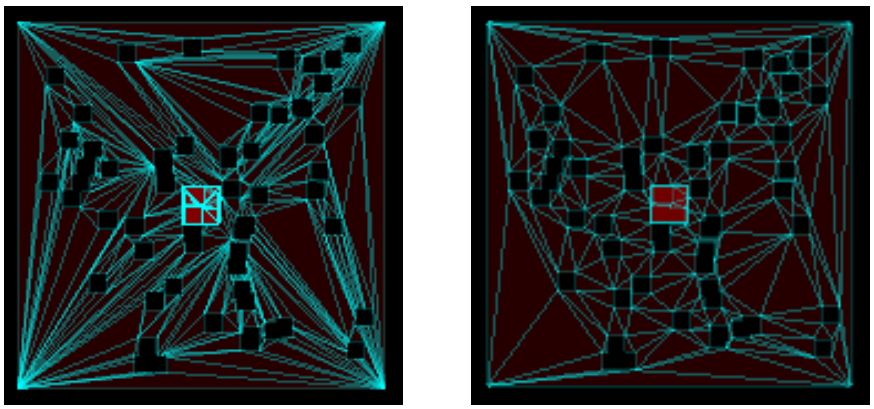
1. Query what physics fixtures intersect the AABB box of the destruction polygon.
2. Check each fixture if they intersect the destruction by converting the fixture shape into a boost::geometry represented polygon and calling `boost::geometry::intersects()`.
3. If a fixture intersects, check if all destruction polygon points are inside the fixture polygon with boost::geometry::within().
4. If the destruction is completely within a fixture:
    1. Convert the fixture and destruction to poly2tri before triangulating the fixture together with the destruction hole.
    2. Create a new fixture for each triangle created with poly2tri.
5. If the destruction intersects several fixtures:
    1. Calculate the difference $terrain - destruction$ with boost::geometry::difference for each terrain fixture polygon.
    2. Convert each difference to poly2tri polygons and triangulate them.
    3. Create a new fixture for each triangle created with poly2tri.

5.6.4 Destructions – Third Iteration: Unification

With the previous iterations there were several problems, and the polygons have to be converted between several formats, resulting in a much more complex terrain with each destruction. When a fixture is split it creates several new fixtures, then if those fixtures are split it creates even more fixtures, and so on. Triangulating triangulations over and over again ends up in a large number of small triangles. Representing the terrain as physical fixtures is the source of this problem. The solution is to represent the terrain with a system that can handle large concave polygons with holes without the need to split them. boost::geometry does this.

All off the terrain was changed to use boost::geometry polygons. Since it doesn't handle polygon islands we store each island as a separate polygon on a vector. When a destruction takes place, it takes the difference $terrain - destruction$ on all of the islands, replaces all old terrain polygons with the difference, triangulates them and replaces the physics fixture with the new triangulated polygons. The result of this is a large difference

in number of fixtures for the terrain. This solution also made the code simpler by not handling two different cases (completely within and intersecting) separately. The two cases are the same and are handled in the same way. Because of this there is only one triangulation code that triangulates the whole boost::geometry terrain polygons with each destruction.



(a) Triangulating before third iteration with 60 destructions.

(b) Triangulating after third iteration with 60 destruction.

Figure 33: Triangulating before and after the third iteration.

By using a code block to create 40 destructions randomly with equal seed within the terrain polygon we could see the improvement in complexity. The old version created 491 fixtures with the last destruction, while the new version created 222 fixtures. With 60 random destructions the old version created 1492, while the new version created 324.

From the results we can easily see that the complexity of the old version rises faster than the new. With 40 destructions the old version created 2.21 times more fixtures than the new. With 60 destructions the old version created 4.60 times more fixtures than the new.

Listing 5.2: Code used to test terrain complexity.

```
std::mt19937 randomEngine(0);
std::uniform_real_distribution<float> positionDist(-9.0f,
    9.0f);

for (int i = 0; i < 40; i++)
{
  glm::vec2 worldClickPos(positionDist(randomEngine),
      positionDist(randomEngine));
  auto leftMouseButtonEvent = std::make_shared<
      LeftMouseButtonEvent>(worldClickPos,
      LeftMouseButtonEvent::ButtonState::PRESS);
  this->getLogicContext()->getEventManager()->queueEvent(
      leftMouseButtonEvent);
}
```

5.6.5 Storing the Terrain

To store the terrain, the class DestructibleTerrain is used. It represents a rectangle with a defined size at a defined position. This makes a single terrain fit into one chunk.

The operations involved in creating destructions are slow when the terrain contains several hundreds or thousands of vertices. This isn't always the case, but when there have been applied a lot of destructions with complex shapes, it quickly becomes the case. A short frame freeze due to calculations of terrain destruction can be a bad experience for a player, and might lead them to stop destructing the terrain or playing the game altogether.

To make the geometry calculation faster, the terrain is split into a defined number of terrain bodies making a grid. Each body store its own terrain polygons, and is represented independently in the physics. This means that the physics can be queried with the AABB box enveloping the destruction polygon, and terrain bodies intersecting the destruction will be found. Then applying the destruction only on these bodies is more efficient than applying it on the whole terrain. The physics will only have to update the polygons in the bodies affected instead of the whole terrain. Having the terrain split into bodies also lets a larger part of the terrain be inactive, requiring less computing power for the physics.

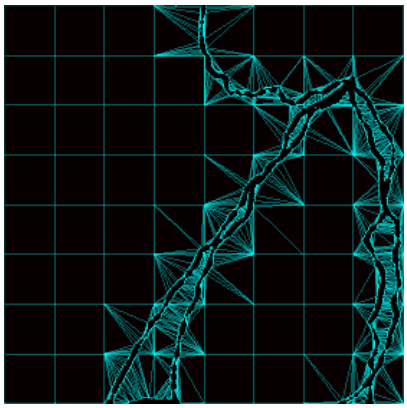


Figure 34: Terrain chunk split into 8x8 grid of bodies.

5.6.6 Optimal Polygon Decomposition

poly2tri uses triangulation to decompose the polygons. A more optimal solution is to decompose them into convex polygons. This would result into fewer polygons, thus putting less strain on the physics simulation. Other polygon splitting algorithms were considered because of this. One of these is Mark Bayazit's Algorithm. [25] The only shortcoming of this algorithm is that it does not support interior holes like poly2tri does. A solution to this would be to decompose the terrain with Mark Bayazit's algorithm when there are no interior holes, and use poly2tri when there are.

5.7 Generating the World

The world generation has a set of requirements that need to be met:

- The world has to be different each time you start a new game.
- A world has to be reproducible. This means that one player can produce the exact same world over again on a new save, if they want to.

- The world has to be reproducible across all supported operating systems and hardware.

5.7.1 Unique Worlds

The first requirement – supporting different worlds – is pretty easy to meet: use a pseudorandom number generator (PRNG) with a different seed each time. To be sure that the seed is different, we use the C++11 std::random_device class, which produces a true non-deterministic random value if hardware is available to do it, otherwise a pseudorandom value. Either way, the randomness is sufficient to provide a good random seed for our world generator.

While there still is the extremely tiny possibility of getting the same seed, the chance is so small that it will likely never happen to a player. We decided to ignore working on this case due to the microscopic chance of it happening, and if it should happen, it most likely wouldn't bother the player.

5.7.2 Reproducible Worlds

The second requirement – reproducible world – is much more difficult. It might seem like an easy task since the PRNGs are deterministic and will produce the same results with the same seed each time, but this is not enough. As an example, the player walks straight down two chunks in a procedurally generated world based on a seed, then starts a new world with the same seed, first walks up one chunk, then down two. With this example, the resulting world will be different. The numbers generated by the PRNG (e.g. 1, 2, 3) will be used on different chunks in the different worlds. The first world will have the first numbers used on the chunks downward, while the second world will have the first used on the one upward. This results in that the first chunk down in the first world, will be exactly the same as the first chunk up in the second world. And the first chunk down in the second world, will be exactly the same as the second chunk down in the first world.

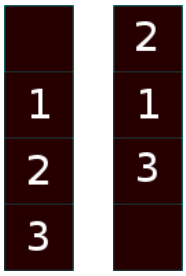


Figure 35: The chunk generation order. First: down down. Second: up down down. Only the spawn chunk (1) is identical.

To prevent this, a chunk has to be the same regardless of the order in which it is generated. For this there has to be something unique to the chunk that can be used as the PRNG seed. The obvious candidate is the position. Each chunk has a unique position in the world. The position consists of an x and y value that has to be combined. std::seed_seq is used for this. The y position, x position, and world seed are used to initialize the seed sequence. Then the seed sequence is used to initialize the PRNG. This ensures that the world is reproducible with the same world seed. This seed sequence is called the chunk seed.

5.7.3 Cross-platform Reproducible Worlds

For the last requirement – reproducible across all supported operating systems and hardware – we had some problems. At first this looked like a simple task because C++11 defines different deterministic PRNG algorithms that will produce the same values for the same seed on all platforms. This is true, and the generators alone will produce the same sequence of numbers. However, the C++11 distributions (e.g. `std::uniform_real_distribution`), used to distribute the random values over a defined range, produced different values for each operating system (Linux, OS X, Windows). This is because the C++ standard doesn't define the algorithms for these distributions, and their results are therefore implementation defined. To go around this we created our own simplified distributions that mimic the behavior of the standard distributions, but use the same implementation independent of platform. They might not produce as good distributions as the libraries' implementation, but they are reproducible and the numbers are good enough for this game's requirements, as we are not doing actual statistical analyses.

5.7.4 The World Generator

All the requirements discussed in the above sections are implemented by the world generator system. This system is an instance of the WorldGenerator class. Its responsibility is to generate the terrain and populate it with actors.

The world generator system is not part of the central game logic, but it is used by an actor component called the open world component. The terrain itself is a part of the actor using the open world component, and the physics bodies for the terrain are all body parts of that actor. The generator is initialized with various data from the open world component. The most important of these is the seed. If the seed value is not defined, it will generate a random seed with std::random_device.

Initially the world handler was merged with the world generator. With this solution an actor managed the world, while being part of the world. This led to several problems where the biggest was that to save the world, the game logic would have to find an actor that did this. It later became apparent that the world shouldn't be handled by an actor, but by the logic itself. Thus the world handler was split from the world generator and added to the game logic as a permanent subsystem. The generator listens to chunk change events generated from the world handler and takes appropriate actions needed to transition into a state. This includes generating and saving the terrain, and populating the world with actors.

How a chunk is generated is decided by a chunk generator and its parameters. There are several chunk generator classes that implements an abstract class called `ChunkGenerator`. A set of chunk generation rules for depth intervals are defined in the open world component. Each depth interval defines the type of chunk generator used that is directly mapped against the chunk generator implementations, and a set of parameters that are passed to the generator. The same chunk generator with the same parameters will be used for every chunk inside this interval. When a new chunk has to be generated by the world generator, the appropriate generator is looked up based on the chunk position, and the chunk data is passed to the generators generate function. A chunk generator both decides how the terrain formation should be, and what actors should be placed in the chunk.

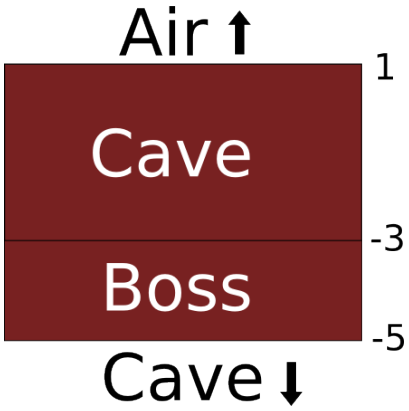


Figure 36: The rules defined for pyroeis. Depth 1 and above is air. Depth 0 to -3 are caves. Depth -4 to -5 are boss caves. Depth -6 and below are caves.

The chunk generator system was developed during the prototyping of another game called “Dreki”. This game was based on the Pyroeis engine and required a hill terrain with solid chunks below. The chunk generator system was thus created where three types of generators were defined: Air, hills and solid. Later came the cave and boss generators that is used in Pyroeis.

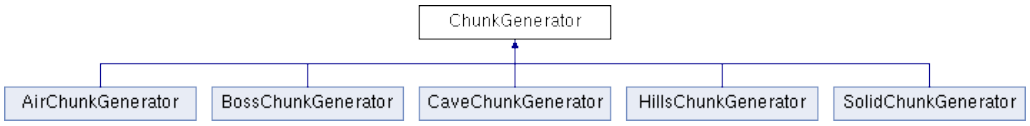


Figure 37: The ChunkGenerator implementations.

Further enhancements to the chunk generator system would be to have rules based on the position in the x axis and not only the y axis.

5.8 Handling the Terrain

5.8.1 Generating Caves

New terrain have to be generated. This happens through the world generator based on chunk state change events. Each chunk has its own DestructibleTerrain instance that has the same width and height as the chunk. This instance is passed to the generator used for generating the chunk.

The two chunk generators used for Pyroeis are the cave and boss generators. It also uses the air generator, but this practically does nothing and is therefore not discussed further. The boss and cave generator are very similar. The only difference is the destruction polygon and how the paths are created. The system in itself is the same.

The most important system used to generate the caves in the generator is the cave digger. A cave digger creates cave paths based on some parameters and then creates destructions over that path.

One of the problems that had to be solved was how to make sure that each chunk is connected to its neighbours. They can not just have random cave paths since they might both start and end inside the cave. The solution is to generate a number of entrances at each chunk wall, then generate a number of connection points inside the chunk and con-

nect the entrances to these points. Each entrance point connects to a randomly selected connection point. These connections form the cave paths. The number of entrances per meter per wall and the number of connection points are parameterized.

With both entrances and holes placed it can start generating the paths. Each path start at an entrance and goes to a randomly selected hole. The paths are initially completely straight. To make them more natural they can be displaced with the midpoint displacement algorithm. [26] The roughness, direction and number of iteration for the displacement can be parameterized per path. The midpoint displacement implementation was initially developed as a part of Dreki to create hill terrain, but is just as useful for creating caves in Pyroeis.

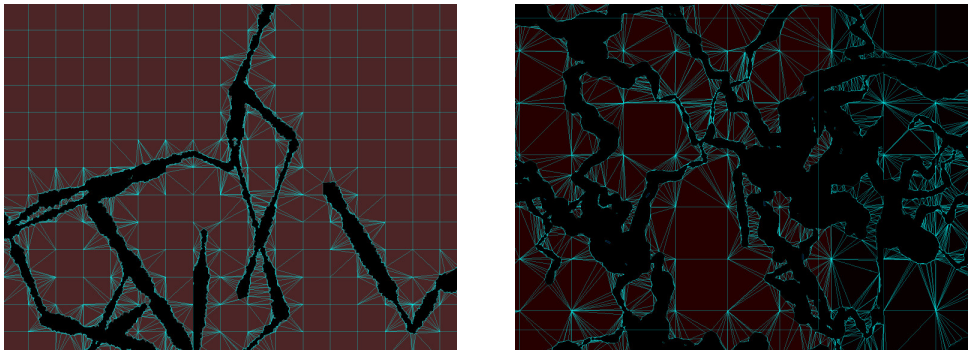


Figure 38: Caves without (left) and with (right) midpoint displacement.

When all the paths are ready, they are dug one after one. The digging process traverses the whole path and applies destructions at a parameterized interval. A specified destruction polygon is used during the path digging. For each step the polygons scale is added or subtracted with a value randomly selected from a parameterized range. A random rotation is used, and the destruction is translated with a value randomly selected from a parameterized jaggedness range. This results into unique cave systems.

Each chunk uses their own unique chunk seed for generation, except the selection of the chunk entrances. If the entrance placement uses the chunk seed, the result will be like the left image in Figure 39 where four chunks completely mismatch. To solve this each wall has its own seed that is independent of the chunk. In the right image in Figure 39 the bottom left chunk's right wall uses the same seed as the bottom right chunk's left wall. This makes all entrances match at the chunk borders. The seed used for each wall is a unique combination of the position.

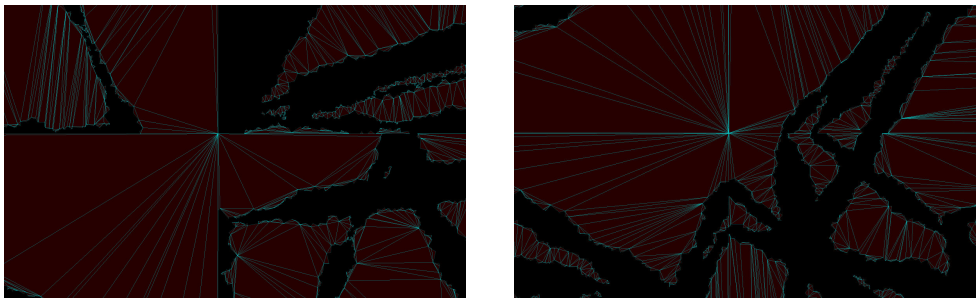


Figure 39: Before and after the chunk walls matched.

Left wall uses the combination “world, x, y”. Right wall uses the combination “world, x+1, y”. Bottom wall uses the combination “world, y, x”. Top wall uses the combination “world, y+1, x”. As an example: A chunk at position (3, 5) will have the left wall seed (3, 5). A chunk at position (2,5) will have its right wall seed (2+1, 5) = (3, 5). The seeds for the wall between the chunks match exactly.

The destructions applied when generating is only applied to the chunk being generated. Because of this caves often end up being clipped when they intersect a neighbour chunk. When the cave paths are displaced it is worse. The paths can go several meters into another chunk before returning, which creates dead ends. This is solved by storing all the destructions that intersects neighbouring chunks and apply them on the chuks they intersect when they are generated. This solution leads to another problem when populating the world with flora that is described at the end of Subsection 5.9.2.

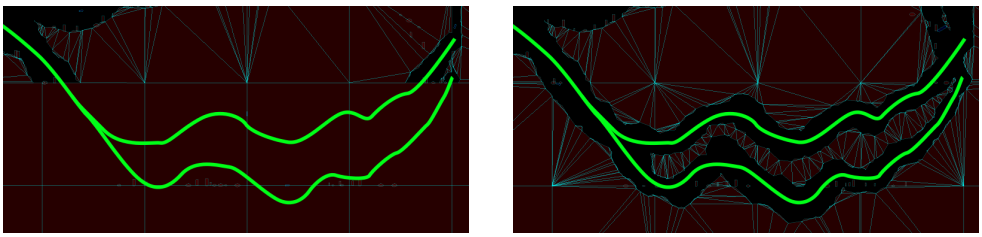


Figure 40: Before and after fixing destructions applied outside chunk.

5.8.2 Saving Terrain

To easily serialize and deserialize the terrain, the DestructibleTerrain class provides an init function that initializes from a terrain JSON object, and a serialize function that writes to a JSON object. The terrain writes its position and size, and lets all the terrain bodies write their terrain polygons. The terrain polygons are written as a string using the well-known text (WKT) format. The reason for using this is that boost::geometry has a writer and reader for it, which makes it a simple task to write and read the polygons, and the output can easily be edited. An alternative that was considered is to store all the destructions applied to a chunk. The ID of the destruction polygon would be saved together with the translation, scale and rotation. The destructions would be store in another file and mapped with IDs. This would be more space efficient, but require more CPU-time to re-create the terrain.

5.9 Varying the Environment

5.9.1 Varying Environment

The game design document states that the game will feature biomes based on the heat and moisture at a position in the world. The biota, including flora and fauna, present in the world, and also the terrain formations, will be based on the biome type.

Heat and moisture are two world environment parameters that vary with the world depth. It is possible to add more parameters. These parameters will be referred to as “environment parameters” from this point on. Heat uses the celsius scale. Moisture uses a custom scale where 0 is completely dry, and 100 is so moist that it is liquid.

The initial design thought of biomes as defined areas in the world. The areas would be defined by the heat and moisture level. This was changed with the implementation.

Instead of defining areas, the biota of the world is dynamically placed based on the environment parameters.

The current version varies the environment parameters only based on the depth. They are configured in a JSON file called “world_environment.json”. Each parameter has several value points defined. For example the heat parameter is -20 at height 100, 0 at -256, 40 at -1024, and 70 at -2048. In between the values are linearly interpolated, and above the highest and below the deepest they are locked to the value specified. The moisture has a different range. With the parameters using different ranges there will dynamically appear unique combinations.

The depth based parameter values creates a baseline for the environment definition, but only varying with depth creates a repetitive experience. Adding a randomness on top of this would create a more varied and interesting environment. One technique that could be applied for this is 2D Perlin Noise. [27] Each chunk could generate a noise map that cover it at a low resolution. Then the noise could be added to the depth based parameter. One noise map could be generated per parameter based on a seed.

With the current implementation, only flora is placed based on environment parameters. The current fauna is not varied enough to use this mechanic, but with more fauna variation this can easily be implemented.

5.9.2 Creating Flora

Each plant type in the game has a set of requirements that specify an environment parameter name (e.g. “heat”) and an acceptance range (e.g. [4, 20]). When planting plants at a certain position, each requirement is checked against the value for the environment at that position. If all parameters’ value is within the acceptance range, the plant can live there.

All plants are sorted into categories. The current categories are bush, hanging, and tree. These categories have unique properties. The bush and tree types can be planted at floors that are no steeper than $\pi/4$ rad. The hanging category can be planted on ceilings that are no steeper than $\pi/4$ rad.

When caves are being generated, the path for each cave is stored and used to plant the plants. The world planter (class WorldPlanter) goes over each of these paths for each plant category with a parameterized plants per meter. With the current configuration there are 0.2 trees per meter, 0.4 bushes per meter, and 0.6 hanging plants per meter.

For each step that the world planter takes over the cave path, it checks if the plant category currently being planted can be planted there. To do this it casts a ray against the terrain to find the floor or ceiling depending on the category. If the ray doesn’t hit anything within ten meters, the spot is not suitable. If the ray hits the terrain, the normal of the surface hit is checked against the accepted steepness of the plant category. If the surface is not too steep, the point is accepted as a planting point.

With a planting point selected, it now needs to find a suitable plant. It randomly selects a plant from a bucket until a plant suitable is found. This method uses our RandomSelectionBucket class. This class acts as a bucket containing elements, where elements can be randomly picked from the bucket. When an element is picked, it is removed from the bucket. The bucket can be reset to contain all the initial elements before they were picked.

For each plant picked from the bucket, all of its environment parameters are checked

against their value at the planting point. If all of them are inside the accepted range, the plant will be planted and the selection will end. If the requirements are not met, a new plant will be selected from the bucket until either a plant is accepted, or the bucket is empty. If no plant could be selected, the point is discarded.

Figure 41: Very dry and cold environment where only dry grass live. Heat: -13. Moisture: 0

Figure 42: Some more heat means that cacti can live there. Heat: 1. Moisture: 1.

Figure 43: Alien plant requiring more moisture appears. Heat: 5. Moisture: 8.

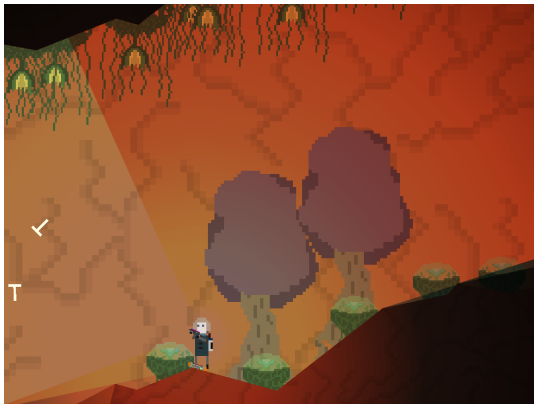


Figure 44: With a warm and moist environment, the flora is rich, but cacti and grass can't live. Heat: 22. Moisture: 34.

Hovering Plants

With the current implementation plants that hover in the air might be present. This is a bug that we have been working on for several hours. The problem is caused by the intersection chunk destruction fix applied by the world generator when generating the terrain. There are two cases where this causes a problem.

If the generator creates a path that intersects a neighbour chunk it will plant plants that are inside the terrain of that chunk because the planting algorithm selects the bottom of a terrain body as a suitable planting point. When the path then is later fixed on that chunk by applying the missing destructions, the plants there will not be placed correctly, and because they are static, will hower. An example of this can be seen in Figure 40. This can be fixed by not planting plants outside the chunk, but it would result in that there are no plants inside that cave path.

If the generator fixes a missing path in a chunk that has plants, it might destruct the ground that the plants stand on, causing the plants to hower. This could be solved by pre-generating the paths of each neighbouring chunk and applying these at the current chunk, then planting the plants. This also solves the previous case. The drawback with this solution is that it requires refactoring of the generation code, and it will take longer to generate chunks.

5.9.3 Creating Fauna

The creation of fauna is not as complicated as flora since there are only two types of animals to spawn. The AI spawner is a basic form of the world planter. It is set up by adding actors with controlling AI views to be spawned at a specified rate.

The AI spawner goes over each cave path the same way as the world planter does. It jumps a specified number of meters for each step. At each step it checks if it should spawn here based on a random chance. The spawn rate can be set as a parameter between 0 and 1. If the rate is 0 it never spawns, if it is 1 it spawns at each step, if it is 0.5 it spawns at 50% of the steps. When a step is selected for AI spawning, it selects an AI randomly. The chance to pick an AI is controlled by a rate that is set by each AI. WIth a rate at 1, it has a uniform chance to be picked. With a rate lower than 1 it has a lower chance to be picked than ones with a rate higher than it. The actor selected is spawned at the step position and a view with an agent is created for it. The spawn rate of AIs vary with the

depth of the world to make the progression more difficult when getting deeper.

5.9.4 Varying Terrain

The terrain is also varied based on the environment parameters. As the terrain is generated per chunk, the terrain formation is consistent throughout the chunk. The heat and moisture value used for the chunk is found in the center of the chunk.

The heat and moisture change the terrain formations in several ways. With more heat the maximum destruction scale is higher, thus creating larger caves. It also makes the terrain more rough by decreasing the midpoint displacement roughness constant. With more moisture the minimum destruction scale is lower, creating tiny paths in some places. More moisture also creates more cave paths.

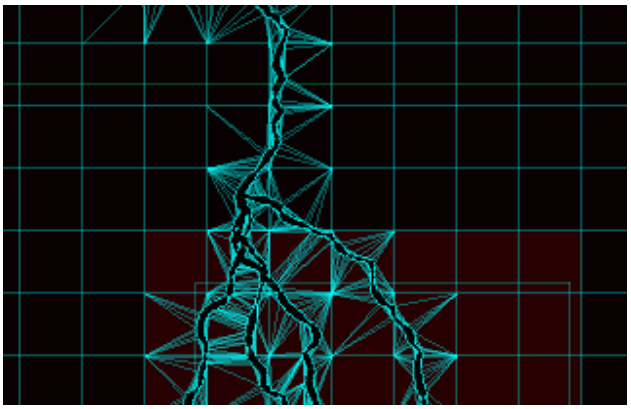


Figure 45: Top chunk 0: Small paths, but not so small that the player can't fit. The paths are not very rough because of the cold environment. The number of cave paths is pre-defined since it is the spawn chunk. Heat: -20. Moisture: 0.

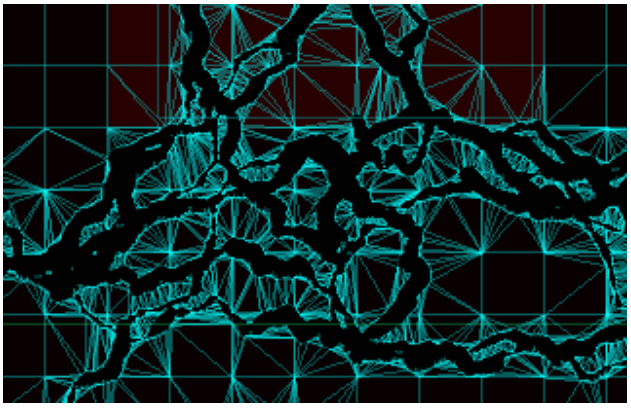


Figure 46: Chunk -1: Larger and more rough caves because of the increasing temperature. Heat: -7. Moisture: 0

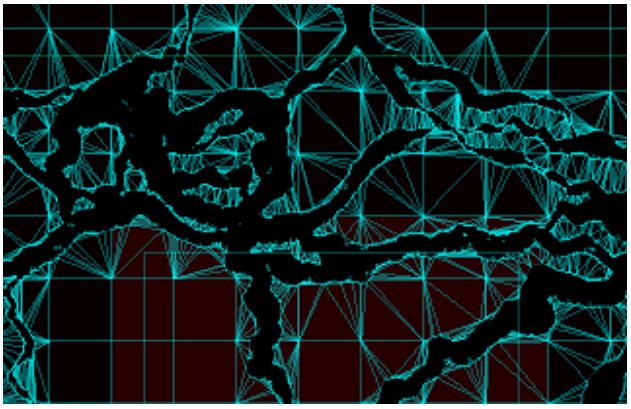


Figure 47: Chunk -2: Even larger and more rough caves. Heat: 7. Moisture: 10

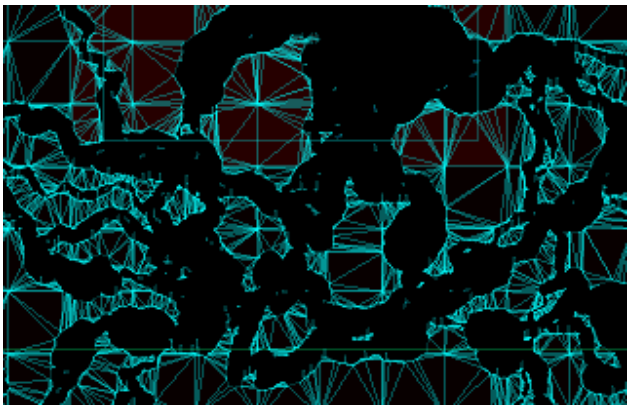


Figure 48: Chunk -3: Caves are still growing larger, but with more moisture cave paths that are smaller than the player can appear. The number of cave paths also has increased because of the moisture. Heat: 20. Moisture: 30

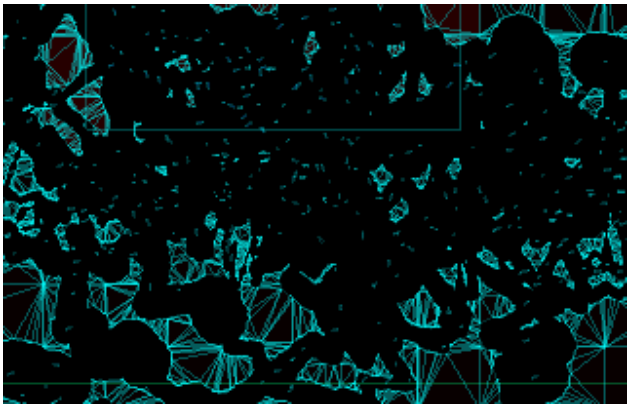


Figure 49: Chunk -6: Below the boss chunk. Very big destructions at its maximum, and very small destructions at its minimum. The number of cave paths is very high making the caves large and open. The generation was only balanced down to the boss chunk, not below. Heat: 51. Moisture: 40

The terrain would be more dynamic if it did not depend on a central chunk heat, but

rather based the formation on the environment parameters at the position it is currently digging. For instance the midpoint displacement roughness could be based on the value in the middle between the start and end point of the path. Or the destruction scale could be calculated from the value at the current destruction position.

5.10 Light Calculation

5.10.1 Calculating Light Area

One of the key components in creating the desired atmosphere for the game is dynamic lighting. The lighting was originally created for Project Nox, which can be read about in Appendix A. Although most of the lighting was complete before the start of this project, there have been made significant improvements and enhancements to the code and versatility of the algorithm. Project Nox had a severely limited game world and thus the lighting was not developed to accommodate all the different shapes in Pyroeis.

Pyroeis has a vast number of different shapes and shape types, due to the fact that the world is randomly generated. There are however a few known facts of the shapes and sizes of the objects:

- The objects consists of convex polygons, this is a limitation in the Box2D physics engine itself.
- Polygons have a limited maximum size.

To increase the number of dynamic lights that can be used simultaneously, the lighting system is running multiple threads. Each light is a self contained data structure, thus this becomes a data parallel problem, ideal for multithreading. For this we have chosen a producer/consumer relationship between the main thread and the consumer threads doing the calculations. The number of consumer threads will be equal to the number of threads available on the hardware. To get the number of threads, `std::thread::hardware_concurrency` is used. [28] This ensures that the lighting system scales very well with the hardware. The main thread is the producer, and pushes all the lights that needs to be calculated onto a thread safe queue. When the lights are all pushed to the queue, the main thread waits on a barrier that releases the consumer threads to do their work. The consumer threads grab and process lights until there are no more lights to be processed on the queue. When all the lights have been processed, the barrier is released and the main thread continues the update process.

Listing 5.3: Function being run by each ligting calculation thread.

```
while (!this->quitThreads)
{
  bool empty = false;
  this->lightAreaProcessingBarrier->wait();

  while (!empty)
  {
    std::shared_ptr<Light> light;
    std::unique_lock<std::mutex> lightLock(this->
        lightQueueMutex);

    if (this->lightQueue.empty() == false)
    {
      light = this->lightQueue.front();
```

```
        lightQueue.pop();
      }
      else
      {
        empty = true;
      }

      lightLock.unlock();

      if (empty == false)
      {
        this->calculateVisibility(light);
      }
    }
    this->lightAreaInsertBarrier->wait();
  }
```

The light calculation algorithm in its basic form is the following for each light:

1. Query what fixtures are inside the AABB box of the light range.
2. Loop through each fixture found and check if they have at least one vertex inside the light range. If any vertices are inside, add all the vertices of that fixture to the list of targets.
3. For each of the fixtures, find the angle that is smallest, and the one that is the greatest. These are set as the min and max angle for that fixture.
4. Sort the targets by their angle relative to the light direction.
5. Send a ray towards each target in order, smallest to largest, we keep track of how far the ray gets before hitting any objects.
6. If the target is a min or a max target, and the ray reaches the target, also send an extra ray past the target. This creates the shadow outlining of the fixture.
7. When all the rays are calculated, loop through the resulting points, and create triangles that represent the visibility from the lights position.

The triangles that are created during the last step of each light is saved as the visibility of the light and then used in the rendering step to render the lights. 5.11.2

5.10.2 Improving Performance

Although we are using multiple threads to do the lighting calculation, it is still a very heavy operation. Doing thousands of ray castings every frame is very demanding of the cpu, thus we are limited in the amount of lights we can dynamically simulate at any given time. Another issue is that the performance of the lighting calculation is highly dependent on the generation of the world. Since the world generation is random, and has a high degree of variance, we can never fully know what will be produced. The current state of the lighting calculation provides satisfactory results in most normal gameplay scenarios. As long as player is not putting up unnecessary amounts of lights, the gameplay will run at a steady 60 frames per seconds on most modern systems. There are however various methods that could be used to increase the performance of the lighting calculation. Some of the methods implemented are described below.

Query Fixture

Instead of iterating through all the world bodies and checking each fixture of each body, it does an AABB query on the world with rectangle with width and height equal to the light diameter. It then iterates through all the fixtures the query returned.

If a light covers a large part of the world, a query might be slower than iterating since nearly all of the fixtures in the world will be found by the query. But in most cases, especially with the large world of Pyroeis, each light only covers a small portion of the world. Therefore doing a query will be faster.

Fixed Size Vertex Array

When handling the lighting against a fixture, all vertices are checked and added to an array. The original version used std::vector to store the vertices, which is a variable size container. [29] With nearly each push_back call to the vector, new space has to be allocated. By using a fixed size array instead of the vector, this space is only allocated once on the stack per fixture. Box2D polygons has a fixed maximum vertices defined by b2_maxPolygonVertices. By using this for the array size we ensure that it always has space for all vertices. Allocation of memory is a slow process, so this saves time.

Removing Equal Points

Due to the physics representation of the world is using convex polygons as the building blocks, we end up with alot of similar raycasting targets. Eliminating these targets should improve the efficiency of the lighting calculation. During testing this has been proven true in some cases. When there are a limited amount of rays that need to be sent, this test has a big improvement on the performance. This is because with a small number of vertices, the check only needs to iterate through a small set of elements. However, when there are a large number of vertices, the results are flipped. With a large number of vertices, the check has to iterate through a large set of elements, which in the end is slower than casting a ray against all vertices. This becomes worse as the number or targets rises. Because of this, the checking is removed from the implementation until we have a better method of doing so.

Listing 5.4: Check and ignore duplicate vertices.

```
for (size_t i = 0; i < numVertices; i++)
{
  bool unique = true;
  const auto rayTargetEnd = rayTargets.end();
  auto rayTargetIt = rayTargets.begin();

  while (unique == true && rayTargetIt != rayTargetEnd)
  {
    if (approximatelyEqual(fixtureVertices[i].position,
        rayTargetIt->position, 1.0e-5f) == true)
    {
      unique = false;
    }

    rayTargetIt++;
  }

  if (unique == true)
  {
    rayTargets.push_back(fixtureVertices[i]);
  }
}
```

5.11 Rendering

5.11.1 Back End

When choosing the rendering back end to use for Project Nox, there were numerous factors that needed to be considered. The choice would have to be as multi platform compatible as possible. The engine should run on Windows, Linux, Mac OS X, and should be easily portable to other systems like Android and iOS. When rendering 2D sprite based graphics there are numerous choices that need to be considered based on the requirements of the game.

Simpler APIs like SDL 1.2 and SDL 2.0 gives developers a lot of ready to go render code. At the same time this takes a lot of the fine control away from developers. In our case we were dependent on having fine control over the rendering pipeline, due to the advanced lighting effects, and also because of the learning benefits associated with this. OpenGL 3.0 was chosen since it gives us direct access to the GPU at a hardware level, and it is an open standard supported by most hardware and software. The benefits of this are significant when doing some of the more advanced rendering calls like stencil rendering, and changing the multsplication of colors when rendering lights. Since OpenGL does not provide a window for rendering, SDL 2.0 is used for creating the window and handling interaction with it.

5.11.2 World Rendering

World rendering in Pyroeis currently consists of the following steps:

1. World background is rendered.
2. The scene graph is parsed, and the world is rendered to a texture.

3. Lights are rendered to another texture.
4. The results are combined, and rendered to the screen.
5. The terrain foreground is rendered.

Both the world object rendering and the lighting was made in Project Nox, and have not changed much since. Some enhancements have been made that are described in the subsections below. The background and foreground rendering is completely new.

Rendering the Background

The world is rendered to a texture using a framebuffer object (FBO). FBOs allow the binding of textures as targets for the color buffer, allowing us to create a screen sized texture to represent the world every frame. Though this is not necessary for normal rendering, it allows for post-processing of the image. Rendering the world consists of three different rendering operations:

1. Rendering the background.
2. Rendering the world objects.
3. Rendering the terrain.

First the background has to be renderer. To do this, a system for tiling textures in a certain area has been implemented. The tiling system had certain requirements that needed to be achieved in order for the system to be effective at what we wanted.

- Infinite tiling in all directions.
- Tiles need to always match.
- Need to be efficient when rendering.
- Support for different colors and textures.
- Support for any tile size.

Infinite tiling is easily accomplished by simply creating the tiles in order, placing one after the other. But we can not have an infinite number of tiles rendered in the world. This is also a limitation of the performance of rendering the tiles, we should not use any more GPU power than what is needed. To solve these problems, the tile creation is limited to an area that is given by the camera controlled by the player. All tiles are based around the world origin in order to make sure that the position of all the tiles in the world match each other. The tiles that are generated will always cover the area that the camera covers. Effectively this means that there is always some outer overlap from the edge of the camera. Another effect of this is that the tiles only need to be generated and the data only needs to be passed to the GPU whenever the camera has moved enough. This saves some time on the rendering side, since I/O is slow.

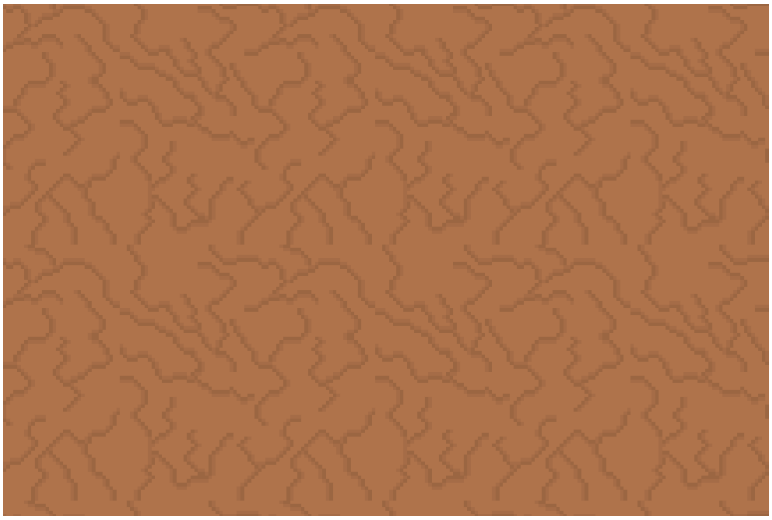


Figure 50: Background rendered using tiles.

Rendering the World Objects

As previously explained, objects in the world are handled using a scene graph structure. The renderer uses the data created by parsing the scene graph to render the objects in the world. To make the rendering more efficient we are only updating the data in the renderer when the data has actually changed. More importantly is that the renderer does all the transformation on the CPU and then passes the data to the GPU for rendering. Although the GPU is generally very well suited for doing transformations, more so than the CPU, the fact that we are only doing one I/O call makes the rendering more efficient.

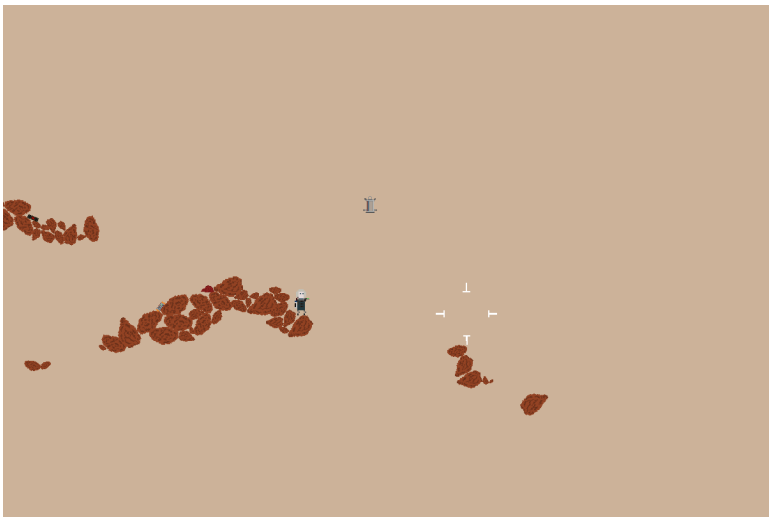


Figure 51: World objects rendered.

Rendering the Terrain

When designing the rendering for the terrain we were faced with various issues on how to accomplish this as efficiently as possible. Since the terrain created in pyroeis is very complex and represented by polygons, rendering a tiling texture of any size is not easily done using normal rendering methods. OpenGL's stencil buffer allows us to apply a "cut

out" to what pixels on the screen we want to render with the texture. This allows us to use the previously mentioned tile renderer as a base for the rendering. We are also required to get the stencil of the area where we want to apply the tiles. Rendering the terrain into the world consists of the two steps: Render the cut out from the active world area. Render the tiles of the terrain onto the cutout.

Figure 52: Terrain rendered using stencil together with tiles.

Completing the World Rendering Without Lighting

The rendered background, world objects, and foreground are all rendered onto the same texture. The texture is fully lit as if the world has constant white lighting.

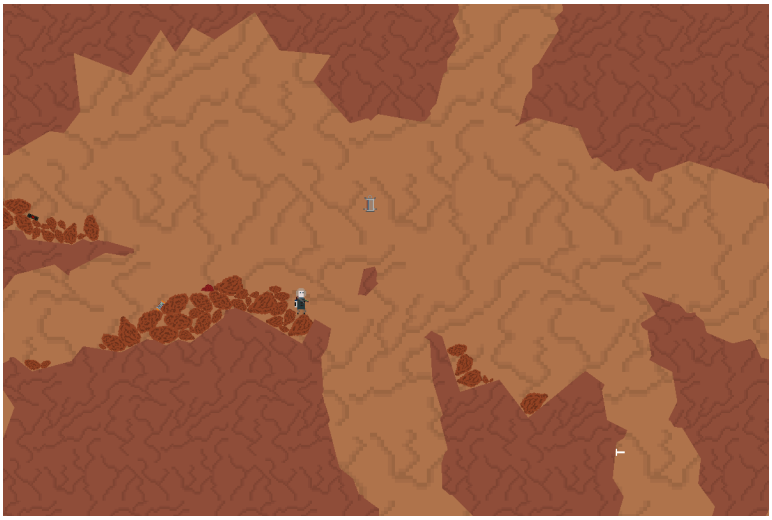


Figure 53: Complete world rendering.

Rendering the Light

To set the mood of exploring the uncharted caves on mars, light is needed. The light is rendered on a separate texture using an additive blending method for the colors. To set the additive blending model in OpenGL we use the call glBlendFunc(GL_SRC_ALPHA,

GL_ONE);

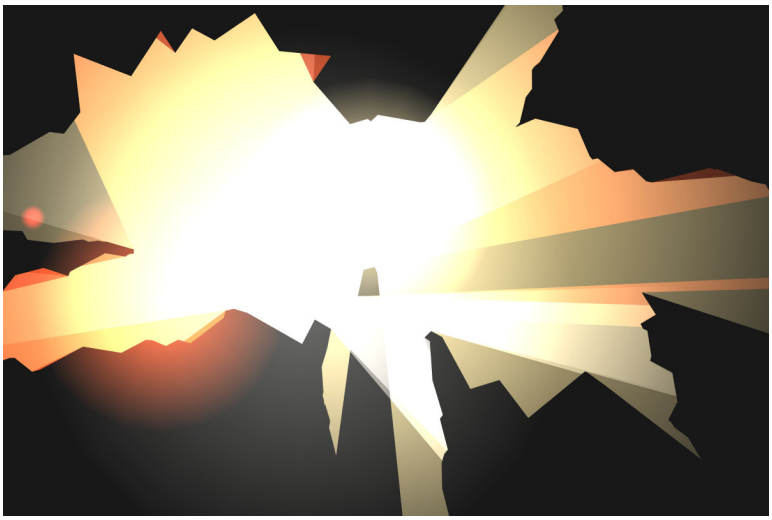


Figure 54: Light rendered to the texture.

Combining the Results

The world texture and the light textures are combined using a multiplicative results of each of their colors. Effectively this means that where there is no light, the world will be rendered black. Listing 5.5 shows the code used to render the light onto the world. The shader gets the two textures as its input: world texture and light texture. It also gets the fragment colors using the fragment position (fragTexCoord) and the texture() function. The colors of the two texture fragments are then multiplied to produce the resulting color for that fragment. The Alpha value is not of any interest and is set to 1.0.

Listing 5.5: Texture and light combining shader

```
in vec2 fragTexCoord;

uniform sampler2D renderTexture;
uniform sampler2D lightMapTexture;

out vec4 outputFragColor;

void main()
{
    vec3 renderColor = texture(renderTexture, fragTexCoord).
       rgb;
    vec3 lightMapColor = texture(lightMapTexture,
       fragTexCoord).rgb;

    outputFragColor = vec4(renderColor * lightMapColor, 1.0)
       ;
}
```

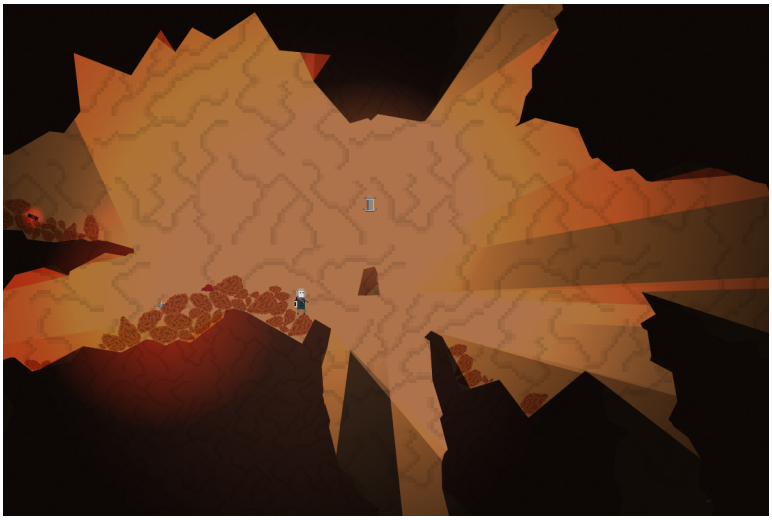


Figure 55: The combined world and light render.

When combining the light with the image of the world, we were slightly disappointed by the quality of the light. The textures can not get brighter than their original color, it makes the light seem dull and not lifelike. The reason to this is the multiplication: A light with the strongest color (1, 1, 1) multiplied with a texture of a color results into the texture color. In order to correct this we could use a bloom or glow effect detailed in nvidia's GPU Gems 3. [30] This would allow us to model the high dynamic range of lighting that we as humans experience every day.

Another, simpler, alternative is to use the fragment shader to blend the layers a little differently. This lets us model a higher dynamic range while also keep the crispness of our lighting. In this version of the shader, we have some cross-bleed between the lights colors. The color calculation is exponential so that colors that have low values do not affect the others as much. This makes the Light in the scene appear much more alive, and much closer to the vision we have imagined.

Listing 5.6: Improved lighting shader.

```
vec3 lightCubed;
lightCubed.r = lightColor.r * lightColor.r * lightColor.r;
lightCubed.g = lightColor.g * lightColor.g * lightColor.g;
lightCubed.b = lightColor.b * lightColor.b * lightColor.b;

lightColor.r += 0.5 * (lightCubed.g + lightCubed.b);
lightColor.g += 0.5 * (lightCubed.r + lightCubed.b);
lightColor.b += 0.5 * (lightCubed.r + lightCubed.g);

outputFragColor = vec4(renderColor * lightMapColor, 1.0);
```

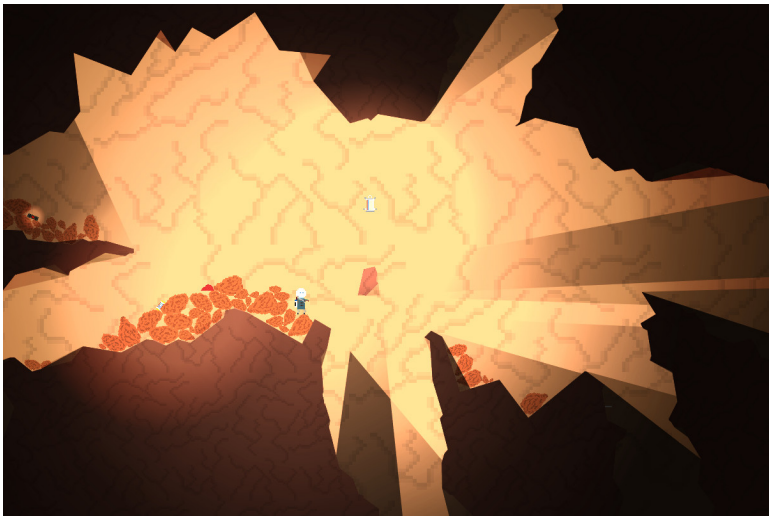


Figure 56: Scene rendered with new lighting shader.

5.11.3 Debug Rendering

In order for us to be able to easily check various states of the world, we need a debug rendering system. The current system incorporates only the physical states of the world, but could easily be extended to incorporate anything in the future. The debug rendering can be toggled on and off while the game is running (using ctrl + 2, and then the Q key). This has been very convenient for debugging various issues that have happened during play.

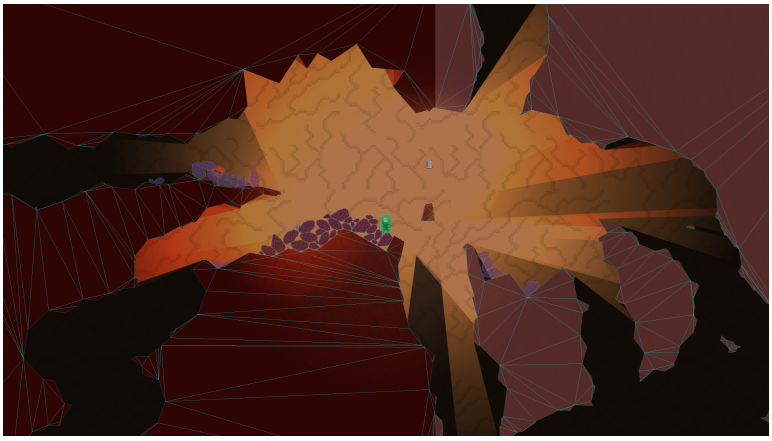


Figure 57: Debug rendering enabled. The red represents the static objects, blue the dynamic and green are objects with joints. The cyan lines represent how the bodies of the terrain are split for the physics.

The debug renderer is originally from Project Nox, but has been extended with support for rendering object edges. The way the geometry is updated and renderer has also changed to be faster in various ways. With the version in Project Nox, the game would often hang for several seconds because it was deleting single elements within a large vector. This was not a problem in Nox, but in Pyroeis where the world is large, complex, and can be changed, this is a serious problem. It is still slow and could be optimized

more, but for debugging purposes it is perfectly fine. An end user would never use it.

5.11.4 Light Blur

The shadows in Pyroeis are all based around point lights. The idea that every ray from a light comes from one discrete point. In the real world this is something that is technically impossible to achieve, and in the very least not the normal way lights appear. Lights in the world usually come from a larger area in space, this creates two features missing from the lighting system in Pyroeis.

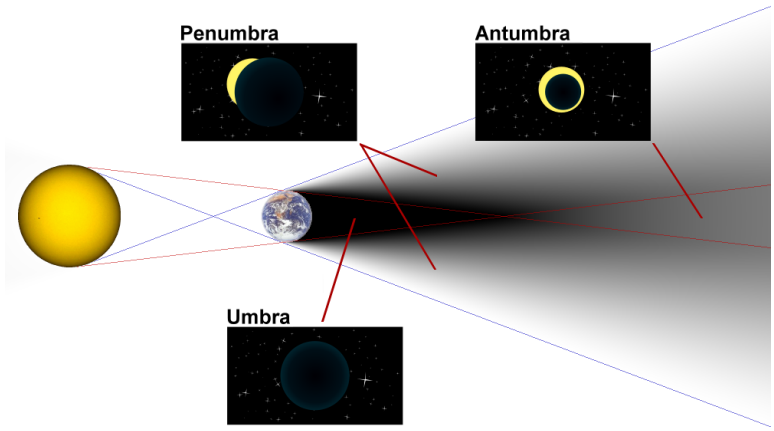


Figure 58: Overview showing the different types of shadow created by a light source. [31]

The image above illustrates the three types of shadows that are cast from a light-source. The Penumbra and Antumbra is only partially hit by the light source, while the Umbra is not getting hit by any light at all. For point lights the umbra is the only type of shadow being cast, any point in space is either hit by the light, or not hit. This creates very sharp edges to any shadows in the world. Penumbra is the most defining feature that is missing, it provides the softness of shadows, and allows the brightness of the light to ease in. Although solving the issue is possible, it is also one that is processing heavy. As shown by Figure 58, two rays, instead of one, per ray target needs to be sent. The rays are sent from a point placed at radius' distance, perpendicular to the direction to the target, at each side of the light source. The area these two rays create after having passed the target creates penumbra. The area is then used together with a texture to recreate the penumbra shade. The procedure is explained more thouroughly here: [32] Because the light already takes up a lot of processing time, we have opted not to go for the correct solution.

Another way of giving the shadows a softer look is to use a blurring technique. Since we are already using a deferred shader, adding a blur shader is easily done. The blur is done using the following steps:

1. A low resolution lighting texture is created.
2. The low resolution texture is blurred horizontally and rendered to another texture.
3. The horizontally blurred texture is blurred vertically and rendered to another texture.
4. The resulting texture is used as the lighting texture for the world.

The reason for blurring the texture in two passes instead of doing the blur in one pass, is that this is more effective. Doing the algorithm in one pass results in the GPU having to do $r * r * w * h$ multiplications in the fragment shader. Where as doing the blur in a two pass method results in $2(r * w * h)$ calculations. Where r is the radius in pixels to blur, w is the width of the image in pixels, and h is the height of the image in pixels. In our case we are blurring the texture in a radius of 7 meaning that 14 multiplications (7 in each direction) are required for each pixel. For a 1920x1080 resolution screen we are using a 960x540 resolution blur texture, resulting in the following calculations.

One pass $14 * 14 * 960 * 540 = 101606400$

Two pass $2(14 * 960 * 540) = 7257600$

The number of calculations is severely reduced. Around 14.29% the number of calculations are required when doing the two pass algorithm compared to the one pass algorithm.

Creating the Low Resolution Texture

The low resolution texture is created by rendering the lighting map to it normally. The resolution does not need to be lower than the normal rendering resolution. Because the textures are blurred anyway, there is no reason for keeping resolution at the highest detail level.

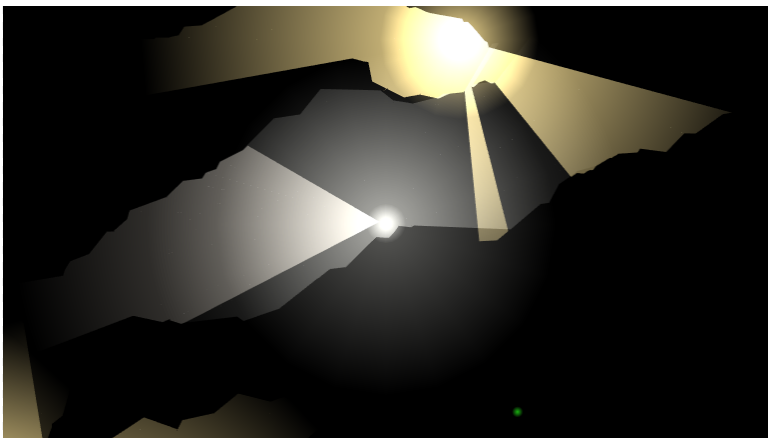


Figure 59: The low resolution lighting texture. Just a normal lighting map.

Horizontal Blur

Horizontal blur is applied to the texture. The horizontal blur is applied to the texture using the Gaussian blur. [33] In order to increase the performance of the blur we are doing some of the calculation on the vertex shader and storing it in an array that is passed to the fragment shader. The results are then interpolated when passed to the fragmentshader, yielding the same results as doing the multiplication per fragment.

Listing 5.7: Horizontal blur vertex shader

```
in vec2 vertex;
in vec2 textureCoord;

uniform mat4 viewProjectionMatrix;
```

```
uniform float blurSize;

out vec2 fragTexCoord;
out vec2 blurTexCoords[14];

void main()
{
  gl_Position = viewProjectionMatrix * vec4(vertex, 1.0,
     1.0);
  fragTexCoord = textureCoord;
  blurTexCoords[ 0] = fragTexCoord + vec2(7.0 * blurSize,
     0.0);
  blurTexCoords[ 1] = fragTexCoord + vec2(6.0 * blurSize,
     0.0);
  blurTexCoords[ 2] = fragTexCoord + vec2(5.0 * blurSize,
     0.0);
  blurTexCoords[ 3] = fragTexCoord + vec2(4.0 * blurSize,
     0.0);
  blurTexCoords[ 4] = fragTexCoord + vec2(3.0 * blurSize,
     0.0);
  blurTexCoords[ 5] = fragTexCoord + vec2(2.0 * blurSize,
     0.0);
  blurTexCoords[ 6] = fragTexCoord + vec2(blurSize, 0.0);
  blurTexCoords[ 7] = fragTexCoord + vec2(-blurSize, 0.0);
  blurTexCoords[ 8] = fragTexCoord + vec2(-2.0 * blurSize,
     0.0);
  blurTexCoords[ 9] = fragTexCoord + vec2(-3.0 * blurSize,
     0.0);
  blurTexCoords[10] = fragTexCoord + vec2(-4.0 * blurSize,
     0.0);
  blurTexCoords[11] = fragTexCoord + vec2(-5.0 * blurSize,
     0.0);
  blurTexCoords[12] = fragTexCoord + vec2(-6.0 * blurSize,
     0.0);
  blurTexCoords[13] = fragTexCoord + vec2(-7.0 * blurSize,
     0.0);
}
```

Note that all the y coordinates in the array are added 0.0 this is because we don't want the blur to affect them at all.

Listing 5.8: Horizontal blur fragment shader.

```
in vec2 fragTexCoord;
in vec2 blurTexCoords[14];

uniform sampler2D renderTexture;

out vec4 outputFragColor;


void main()
{
  outputFragColor = vec4(0.0);
```

```
  outputFragColor += texture(renderTexture, blurTexCoords
     [0]) * 0.0044299121055113265;
  outputFragColor += texture(renderTexture, blurTexCoords
     [1]) * 0.00895781211794;
  outputFragColor += texture(renderTexture, blurTexCoords
     [2]) * 0.0215963866053;
  outputFragColor += texture(renderTexture, blurTexCoords
     [3]) * 0.0443683338718;
  outputFragColor += texture(renderTexture, blurTexCoords
     [4]) * 0.0776744219933;
  outputFragColor += texture(renderTexture, blurTexCoords
     [5]) * 0.115876621105;
  outputFragColor += texture(renderTexture, blurTexCoords
     [6]) * 0.147308056121;
  outputFragColor += texture(renderTexture, fragTexCoord) *
     0.159576912161;
  outputFragColor += texture(renderTexture, blurTexCoords
     [7]) * 0.147308056121;
  outputFragColor += texture(renderTexture, blurTexCoords
     [8]) * 0.115876621105;
  outputFragColor += texture(renderTexture, blurTexCoords
     [9]) * 0.0776744219933;
  outputFragColor += texture(renderTexture, blurTexCoords
     [10]) * 0.0443683338718;
  outputFragColor += texture(renderTexture, blurTexCoords
     [11]) * 0.0215963866053;
  outputFragColor += texture(renderTexture, blurTexCoords
     [12]) * 0.00895781211794;
  outputFragColor += texture(renderTexture, blurTexCoords
     [13]) * 0.0044299121055113265;
}
```

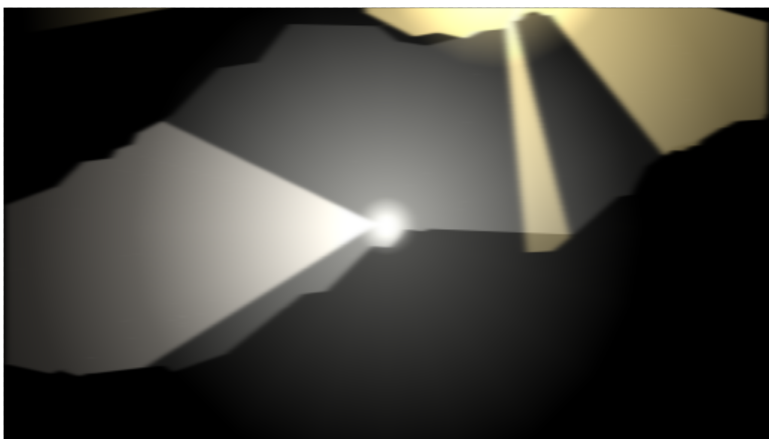


Figure 60: Horizontal Gaussian blur is applied to the texture.

Vertical Blur

To achieve the fully blurred image we also need to apply the same Gaussian blur vertically on the image. This is the exact same process as the horizontal blur, except that the blur is applied vertically. Note the 0.0 values in the x position for all added values in the array.

Listing 5.9: Vertical blur vertex shader.

```
in vec2 vertex;
in vec2 textureCoord;

uniform mat4 viewProjectionMatrix;
uniform float blurSize;

out vec2 fragTexCoord;
out vec2 blurTexCoords[14];

void main()
{
  gl_Position = viewProjectionMatrix * vec4(vertex, 1.0,
     1.0);
  fragTexCoord = textureCoord;
  blurTexCoords[ 0] = fragTexCoord + vec2(0.0, 7.0 *
     blurSize);
  blurTexCoords[ 1] = fragTexCoord + vec2(0.0, 6.0 *
     blurSize);
  blurTexCoords[ 2] = fragTexCoord + vec2(0.0, 5.0 *
     blurSize);
  blurTexCoords[ 3] = fragTexCoord + vec2(0.0, 4.0 *
     blurSize);
  blurTexCoords[ 4] = fragTexCoord + vec2(0.0, 3.0 *
     blurSize);
  blurTexCoords[ 5] = fragTexCoord + vec2(0.0, 2.0 *
     blurSize);
  blurTexCoords[ 6] = fragTexCoord + vec2(0.0, blurSize);
  blurTexCoords[ 7] = fragTexCoord + vec2(0.0, -blurSize);
  blurTexCoords[ 8] = fragTexCoord + vec2(0.0, -2.0 *
     blurSize);
  blurTexCoords[ 9] = fragTexCoord + vec2(0.0, -3.0 *
     blurSize);
  blurTexCoords[10] = fragTexCoord + vec2(0.0, -4.0 *
     blurSize);
  blurTexCoords[11] = fragTexCoord + vec2(0.0, -5.0 *
     blurSize);
  blurTexCoords[12] = fragTexCoord + vec2(0.0, -6.0 *
     blurSize);
  blurTexCoords[13] = fragTexCoord + vec2(0.0, -7.0 *
     blurSize);
}
```

Figure 61: The fully blurred texture.

Final Rendering

The blurred light map texture is then used to create the lit world normally.

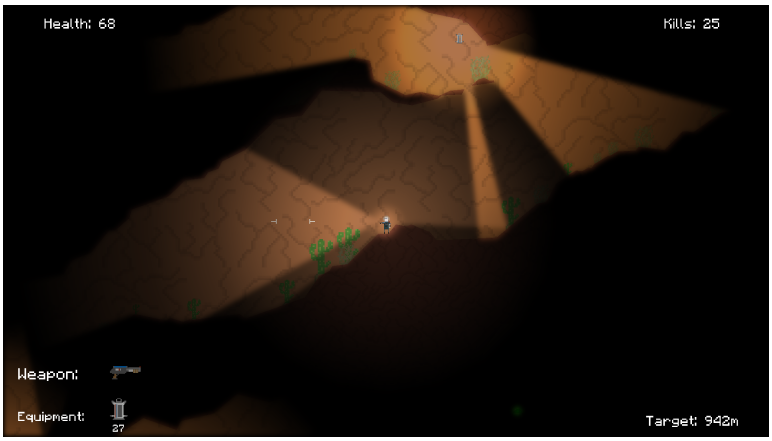


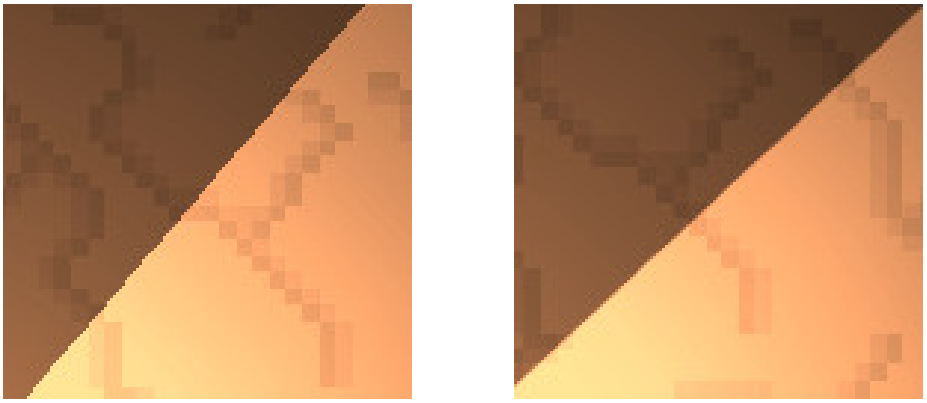
Figure 62: The world rendered with the blurred light map.

Due to some issues we have found with the blur we have not yet included it as the default light rendering. We are particularly unhappy with the blur that is on the edges of the world. Due to the blur also being applied into the light the edges appear unsharp and the look is not the one we are going for.

5.11.5 Anti-aliasing

Light

Aliasing is a common problem when rendering to a pixel-based screen. This is especially true when rendering lines that are not going along the pixels of the screen. For the light this is a major problem. Since the current implementation of the lighting uses the sharp lighting edges the aliasing is very apparent. The effect is amplified when the light source is moving around. To fix this we use a technique called Anti aliasing. Anti aliasing creates edges that are softer, by blending the pixels near the edges of the lines.

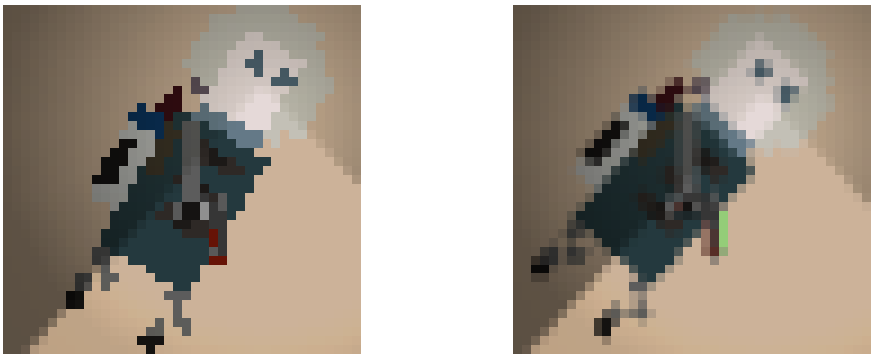


(a) No multisampling. Edge is jagged. *(b) 8x multisampling. Edge is not jagged.*

Figure 63: Result of multisampling.

Textures

Due to the resolution of the textures we are using not being matched to the screens resolution, we are experiencing some aliasing within the textures themselves. Although it is not a big problem for the game itself, it does degrade the visual quality. In order to alleviate the issues we implemented the possibility for super sampled rendering. This is achieved by rendering the world to a texture that is higher resolution than the screen and then using this texture when rendering to screen. The size of the super sampling is decided by a multiplication factor. One times super sampling, that is a multiplication factor of 1, is equal to no super sampling. Two times super sampling, that is a multiplication factor of 2, doubles the width and height of the texture.



(a) No super sampling *(b) Two times super sampling*

Figure 64: Result of super sampling.

Super sampling is a brute force method for anti-aliasing, and is therefore slow. Doing a two times super sampling increases the number of pixels to render to by 2^n, where n is the super sampling multiplier. The high increase in pixels is tough for the fragment shader, and only the fastest GPUs can handle it.

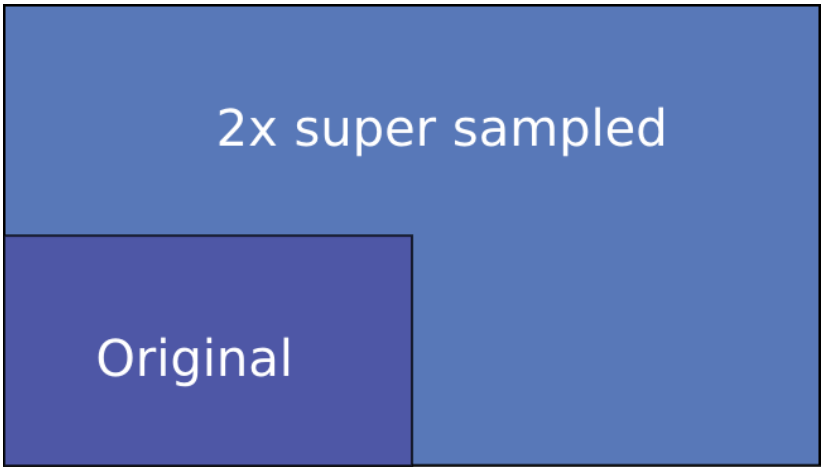


Figure 65: Size increase of the super sample texture.

5.12 Game Artificial Intelligence

In our previous version of the engine, designed for Project Nox [link to sources], we relied on A* pathfinding and nodes for AI to navigate the world. For Pyroeis however, placing the nodes was not something that could be done by a designer. The world is infinitely huge, and dynamically generated as players traverse it.

There were various solutions to the problem, one was to dynamically create a path using the same paths that were used to create the world. This solution has multiple obvious problems that would make the system complicated. As the AI traversed the world, they would have to dynamically handle what nodes were kept in their graph. Having a potentially unlimited amount of nodes would result the pathfinding potentially taking forever. Because of the potential difficulty of implementing such a system, and our needs, we chose to go for another approach.

The AI in Pyroeis is designed around the sense of sight. The AI will traverse the world based on what they see in front of them. The method uses ray casting to determine the best course of action for where the AI is at the moment. To avoid the performance hit of the ray casting, the AI will only check its surroundings once a small amount of time has passed since the last time. The sight algorithm is as follows:

1. The AI will send out a number of rays, spread over a set angle in the direction they are currently traveling.
2. From each ray they cast they get back the distance to the nearest object in that direction. If there is nothing closer than the maximum ray length, the max ray length is set as the distance.
3. The AI will select the ray that got furthest as the direction they want to travel.
    1. If there are multiple rays that have an equal length to the best length, the one with the smallest angular difference from the direction they are now traveling will be selected.
    2. If all the rays are under the minimum length threshold, the same thing will be done in the opposite direction of the current travel direction.

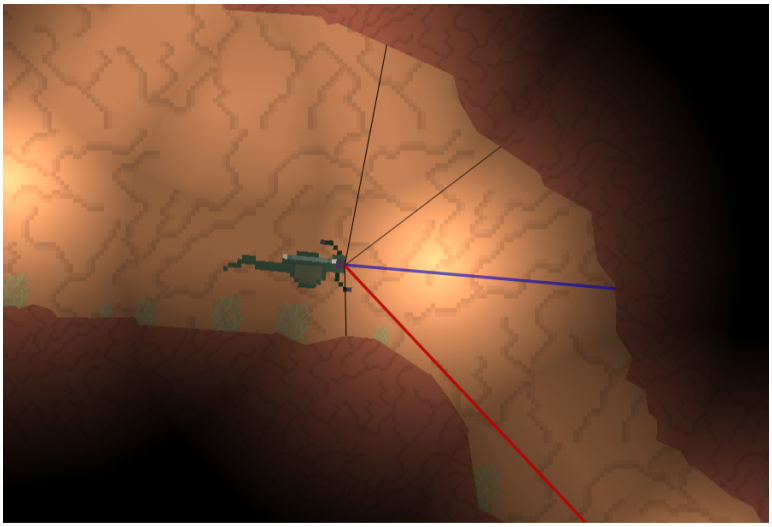


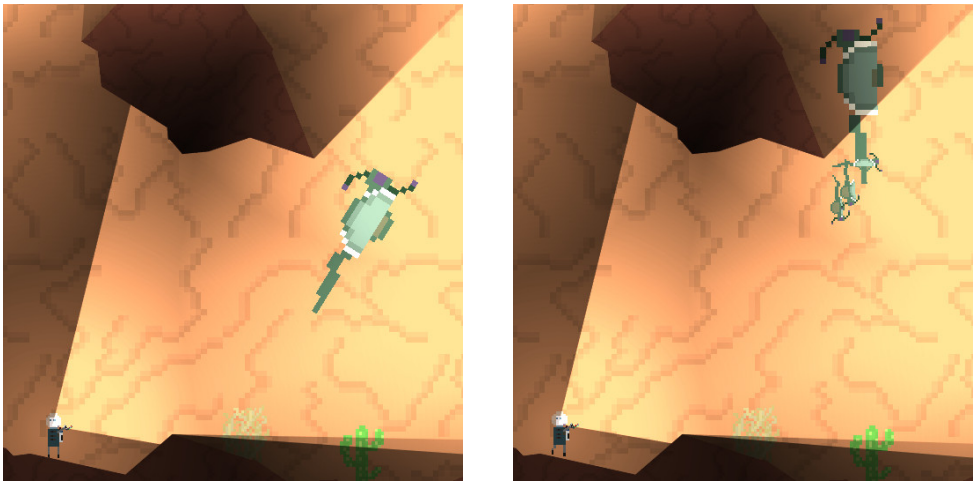
Figure 66: The blue ray represents the direction the AI is currently moving, while the red represents the new direction they will start moving, and the black are the other rays sent.

As mentioned in subsection 3.4.10 in the technical design, the AI is multithreaded. This enables us to achieve a higher level of performance, allowing us to have more simultaneous AI agents roaming the world. This is done using OpenMP on Windows and Linux. [34] Since OpenMP is not currently supported in Clang LLVM compiler (work in progress by Intel: [35]) we are using Apple's similar Grand Dispatch Central. [36] These libraries allow us to tell the compiler to iterate the for loop using as many threads as the system has available.

Listing 5.10: Code for parallelizing the AI update.

```
#ifdef PYRO_OS_APPLE
  dispatch_apply(this->aiViews.size(), aiQueue, ^(size_t i)
           {
             this->aiViews[i]->updateAIAgent();
           });
#else
  #pragma omp parallel for
  for (int i = 0; i < (int)this->aiViews.size(); i++)
  {
    this->aiViews[i]->updateAIAgent();
  }
#endif
```

All AIs in the game are controlled by an AI Agent. The AI Agents make selections based on the current situation that the AI Actor finds itself in. In Pyroeis there are currently three types of actors, each have their own set of behaviors. The mother type will roam around the cave. If she spots the player she will immediately start fleeing. She flees using an algorithm similar to the one used for roaming. When she encounters the player, she will immediately start moving in the direction furthest from the player. When she is no longer able to see the player, she will start giving birth to her spawns: the child enemies. The child type will charge the player if they see them and attack them by destroying themselves near them.



(a) Mother flees from the player. *(b) Mother is safe and gives birth to children.*

Figure 67: Mother enemy flees and gives birth to its deadly children.

The goo spitter type will also roam around the caves. When they see the player, they will attack from a distance. They attack using acid spit. The Goo spitter will try to always keep a safe distance between themselves and their target.

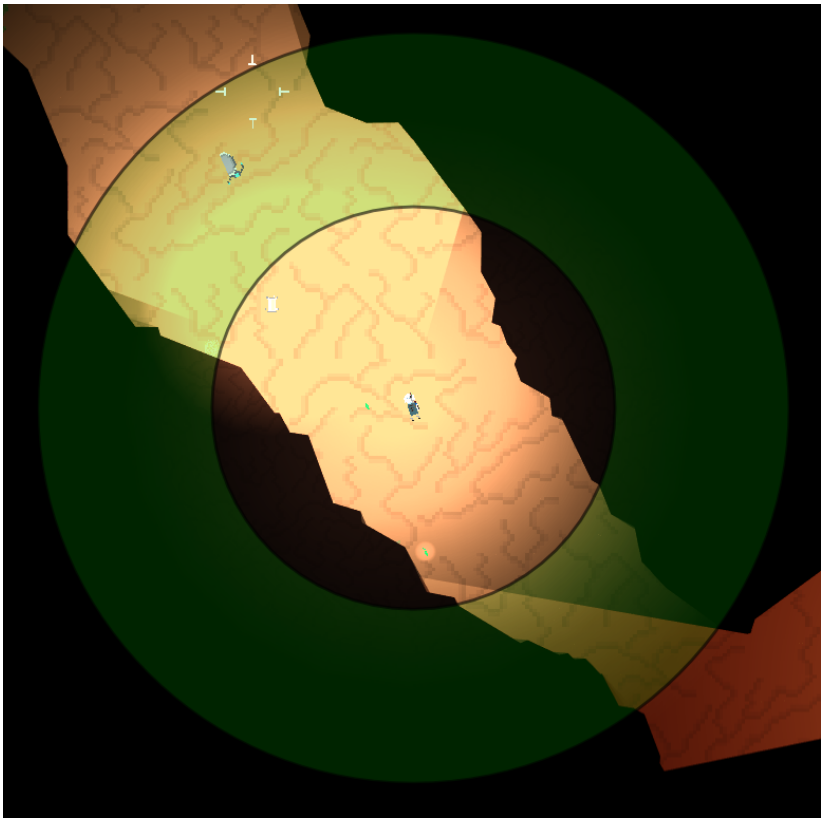


Figure 68: The safe area of the goo spitter is indicated by the green donut shape. It always tries to stay within the area as long as it can see the player.

5.13 Multi-platform Support

One of the most important features of the game engine is the need to support multiple platforms. To make this easier, libraries used have been carefully chosen to match this need.

5.13.1 Mac OS X

Due to XCode, the development environment used for OS X, handling the path to included header files itself, we had some issues when the code was used on the other platforms. As we got used to the development flow, it got routine to always use the full, relative, path to the header file.

5.13.2 OpenGL

Although our experience using OpenGL has been satisfactory, we have had a some interesting compatibility issues. The different versions and iterations of OpenGL has different requirements that the GPU need to support in order to be able to use specific features. Hardware manufactures can also introduce and support other features. This leads to the platform being very flexible, yet very hard to manage from a programming standpoint. What might work perfectly well on a large number of machines, will not work for other machines. One such issue is the usage of different types of Framebuffers.

Our world Rendering system is heavily reliant on using a stencil buffer for rendering the terrain and the background. Stencil buffers come in different variants, two of the major variants are the pure stencil buffer and the combined depth and stencil buffer. Since we are only using the stencil buffer, and have no need for the depth buffer, the natural choice was using the pure stencil buffer. Using the pure stencil buffer was fine for our normal testing machines. The problem was discovered when a tester tried running the game, the result was an incomplete Framebuffer object. After a bit of research, we discovered that the cause of the problem was that the pure stencil buffer was not available on the testers machine. The reason for this is that the pure stencil buffer is not required in OpenGL 3.0, infact for our usage it is not required until the latest version, OpenGL 4.4. [37] This highlights one of the key problems with OpenGL, but it is also one of it's greatest strengths, hardware manufacturers are able to implement features that might take years to become an actual required feature of the library. In our case the AMD cards of our daily working machines and the Nvidia cards of our home machines were all capable of doing the pure stencil buffer, while Nvidia GPU in our testers machine was not.

5.13.3 iOS

Although it was not a part of the initial plan for the bachelors, late in the process, we ported the engine to iOS. The porting process was relatively painless, the only major time consumer was the OpenGL implementation.

On iOS, the implementation of OpenGL is OpenGL ES2. This comes with a few problems when porting. Some of the issues were easy to fix, like the glBindVertexArray(), we simply had to exchange for glBindVertexArrayOES(); Other issues required some research, and other implementations for that specific platform. One of the bigger issues was the binding of other framebuffer objects. For most systems the default framebuffer object, the screen, is bound by calling glBindFramebuffer(GL_FRAMEBUFFER, 0); binding the "zero" framebuffer. All drawing calls when this is bound will then be ren-

dered to the screen. On iOS however, the implementation is slightly different. The default framebuffer is not 0, but 1. This caused some major headaches while porting the engine. There were no error messages, just a black screen being presented. To ensure that we created a good solution for the problem, we use the function call glGetIntegerv(GL_FRAMEBUFFER_BINDING, &defaultFBO); right after initializing OpenGL. This gives us the default framebuffer object for any systems. We can then keep this, and use it to bind the default framebuffer object.

5.14 Optimizations

Throughout the project we were faced with various performance problems in the engine. The number of AI actors and other actors increased, and the world was much more complex, making the lighting take longer to calculate visibility. We ran a number of profiling tests in order to pinpoint the worst issues in the code. The profiling tools found in Apple's "Instruments" package offer a very clear view into what functions are taking the most processing time.

In order to improve the effectiveness of the code we made several changes to some of the engine features. Inlining code was one of the first tasks we started doing. This was usually done for functions that were small, and were used frequently. We also tried to remove as much of the virtual functions as we could as these proved to be very slow. When virtual functions still are necessary, it might be a solution to cache the result from the function. A case like this is when accessing a subsystem of the logic or application logic. The pointer to the system returned could be stored in a local variable, thus often reducing a call each frame to one total call.

One virtual function of a class that is called often is `Event::getType()`. It could possibly be improved by making it non-virtual and rather store the value returned in the class. Therefore, a test was written to see if making virtual functions not virtual would improve the speed. In addition to this, inlining was tested. The test was done using a simple program that timed the three different methods of calling functions on the same class. Each version of the test (normal, virtual, inline) is compiled into a separate binary. The test is a bash script, shown in Listing 5.11, that runs all the binaries. Listing 5.12 shows the code in the header file for the class with the function inlined. The other two classes are similar, but have their function definitions in the source file so that the compiler won't inline them. The class with the virtual function stores the value used in the derived class to simulate the `Event::getType()` case. To make sure that the compiler won't optimize away the code, the functions tested use `time(nullptr)`, and write the result to file at the end. Listing 5.13 shows the code used for the program that tests inline performance. The other two tests are similar, but use the normal and virtual classes rather than inline. The number of runs for the test results, defined by `NUM_RUNS`, is 500,000,000. The results from the test is listed in Table 1.

Listing 5.11: Bash script to run virtual performance test

```
normal=$(./normal)
virtual=$(./virtual)
inline=$(./inline)

normalSpeedup=$(echo "$virtual␣/␣$normal" | bc -l | awk '{
    printf "%f", $0}')
```

```
inlineSpeedup=$(echo "$virtual / $inline" | bc -l | awk '{
    printf "%f", $0}')

echo "Virtual used $virtual ns"
echo "Normal used $normal ns and is $normalSpeedup faster."
echo "Inline used $inline ns and is $inlineSpeedup faster."
```

Listing 5.12: Class with inlined function.

```
struct BaseInline
{
  int data;

  BaseInline(int data):
    data(data)
  {}

  inline int getData() const
  {
    // Ensure that the return value is not constant, so
    // that the optimizer can't predict the result.
    return this->data % time(nullptr);
  }
};

struct DerivedInline: public BaseInline
{
  DerivedInline():
    BaseInline(1)
  {}
};
```

Listing 5.13: Program for running the inline test.

```
int main()
{
  BaseInline* test = new DerivedInline();

  auto startPoint = std::chrono::high_resolution_clock::now
      ();

  int count = 0;

  for (long int i = 0; i < NUM_RUNS; i++)
  {
    count += test->getData();
  }

  auto endPoint = std::chrono::high_resolution_clock::now();
  auto duration = endPoint - startPoint;

  printf("%li", duration.count());

  // Write data collected from loop to file, so
```

```
  // that the optimizer can't remove the loop.
  std::ofstream output("tmp");
  output << count << std::endl;

  return 0;
}
```

Compiler	Virtual	Normal	Inline
GCC 4.8 -O0	1.00	1.09	1.09
GCC 4.8 -O3	1.00	1.10	1.23
GCC 4.8 -O3 -flto	1.00	1.01	1.15
Clang 3.4.1 -O0	1.00	1.06	1.06
Clang 3.4.1 -O3	1.00	1.01	1.18
Clang 3.4.1 -O3 -flto	1.00	1.22	1.22

Table 1: Speedup by not using virtual and inlining on Linux 3.12 with Intel Core i5-430m. Values shown are how many times faster they are than "virtual".

With no optimization on GCC the normal and inline versions are both 1.09 times faster, running at 109% of the virtual funtcion's speed. With level 3 optimization on GCC, the normal version has about the same speed increase, while the inline version runs 1.23 faster (123% of the virtual). This is most likely because the compiler now inlines the function. The results from Clang are similar, though the speed increase of not using virtual is smaller. With level 3 optimization on Clang, the difference between virtual and normal is close to nothing, which likely is caused by better virtual optimization. The inline version is still much better. When it comes to LTO, the difference between all versions on GCC is smaller, and the improvement of not using virtual is nearly non-existent. This is likely caused by optimizations that could only be done at link time. On Clang, normal and inline have the same speed increase, but the increase from the virtual version is much larger than any of the other Clang results.

Although the test we have done here is very clinical, and not applicable to all real world scenario, it is still a good basis, and it would be a good practice to avoid using the virtual functions when possible and use the inline functions where applicable. This is especially true for functions that are called very often in the program, for example when iterating through actors and calling the same function on each of them.

When an application does becomes large, even small things matter for performance. Another small change we made to the code was to avoid the use of std::static_pointer_cast where we do not need a new shared pointer reference. The std::static_pointer_cast was replaced with a standard static_cast. The difference is that the std::static_pointer_cast will give us a new shared_ptr for the object, while the static_cast will give us a raw pointer. Getting the raw pointer is as simple as creating a pointer and assigning its value. Getting the smart pointer is a little more complex. We need to create a new smart pointer, assign its ownership to the same raw pointer that the other smart pointer owns. We also need to do an atomic increase of the reference count being kept for that pointer. In cases where we don't need the extra shared pointer, eg. we are only operating on the pointer once, we might as well not have the extra overhead associated with the shared_ptr. One example of this is the onEvent call being made to every listener that listens for that specific event. Since we are accessing the reference so many times, and the reference is not

being kept, we only need to get a raw pointer to the object. Although each call takes a small amount of time on its own, when the call is being made possibly thousands of times per frame, you can save a significant amount of time in total.

References are being used for all functions parameters or return values where they are applicable. This avoids allocating new memory and using the copy constructor where it is not needed, and can also be a performance increase. To avoid the danger of modifying the content of a reference, const references are used when applicable.

To avoid using the copy semantics, move semantics are used as much as possible. To improve the creation of actors, we created our own move copy constructor and assignment operator for the Json::Value type. This allows the creation to be faster by not creating a new object to contain the data. This is also useful in some cases where we are moving elements of a vector around. simply using the std::move functionality in C++11 allows us to avoid expensive copy constructors in some cases.

6 Deployment

To let users test the game, and to let future consumers buy the game, it needs to be made available. This chapter describes how the game goes from source code to a player's computer, and the challenges with this process.

6.1 Publishing

To make the packaged game easily available for users, it is published to the HiG provided website. The latest version of the game can be found on top. Older versions are still available at below so that users or developers can test an old released version. Each released version includes a package for Windows, Linux and OS X. The binaries built for each system can not run on any of the other systems, and they way applications are packaged on each system is different, so packaging it into one package would both bloat it and confuse users.

Each release is named with a version scheme that is as follows: pyroeis_<version>-<commit>-<builddate>_<platform>. Versions for the releases made during the project period are named after the latest milestone or sprint. If milestone 2 has just been reached, the version will be m2, while if sprint 5 after milestone 2 is complete, the name will be s5. The commit part is the short hash of the git commit that the game is built from. This lets developers quickly checkout the commit for a build to for example fix an error. For later releases, the version and git hash will change to a normal numbering scheme (e.g. v1.2), to not confuse users. The git repository could be tagged with the version for each release, so that developers still easily can checkout a build. Build date is the date when the package was build. While the code behind the binary, or the binary itself not necessarily changes, the package may, and therefore a build date helps differentiate these versions. The platform identifies the platform the package is built for, and the version or architecture of that platform. For example it is "linux_x86_64" for the Linux 64-bit build. If a 32-bit build were to be released, it would be "linux_x86_i686".

When all builds are ready to be published, they are copied to the web server by using scp. The download links are then updated to reflect the new packages, and a post is published to notify users of the new release.

Each blog post is shared on different social media including Facebook, Twitter and Google+ to attract more readers and testers. On Facebook, Suttung Digital's page[1] is used. On Twitter, a new Pyroeis account[2] was created. In addition some updates are shared from the authors personal Twitter accounts, and from Suttung Digital's account. On Google+, Suttung Digital's page[3] is used. In addition a video has been published on Suttung Digital's YouTube page[4] for almost every update. Posts were shared on Suttung Digital's Facebook page all the time, while twitter and Google+ started later. At the end

[1] Facebook URL: https://www.facebook.com/SuttungDigital
[2] Twitter URL: https://twitter.com/Pyroeis
[3] Google+ URL: https://plus.google.com/112767851181154643021/posts
[4] YouTube URL: https://www.youtube.com/channel/UCpO_AZf1yXPdZQ1Q-o7g9oA

of the project period, the game was also shared on reddit[5] and tigsource[6] to gather more feedback. All places that the game has been shared to has helped gathering interest for it and some feedback. On Tigsource one person replied with feedback. On Twitter, we have received at least three interactions from people who played the game. The Gaming Ground wrote a post about the game and tweeted it four times in total. On Google+ there haven't been any interactions. Facebook has been the biggest source for website visits.

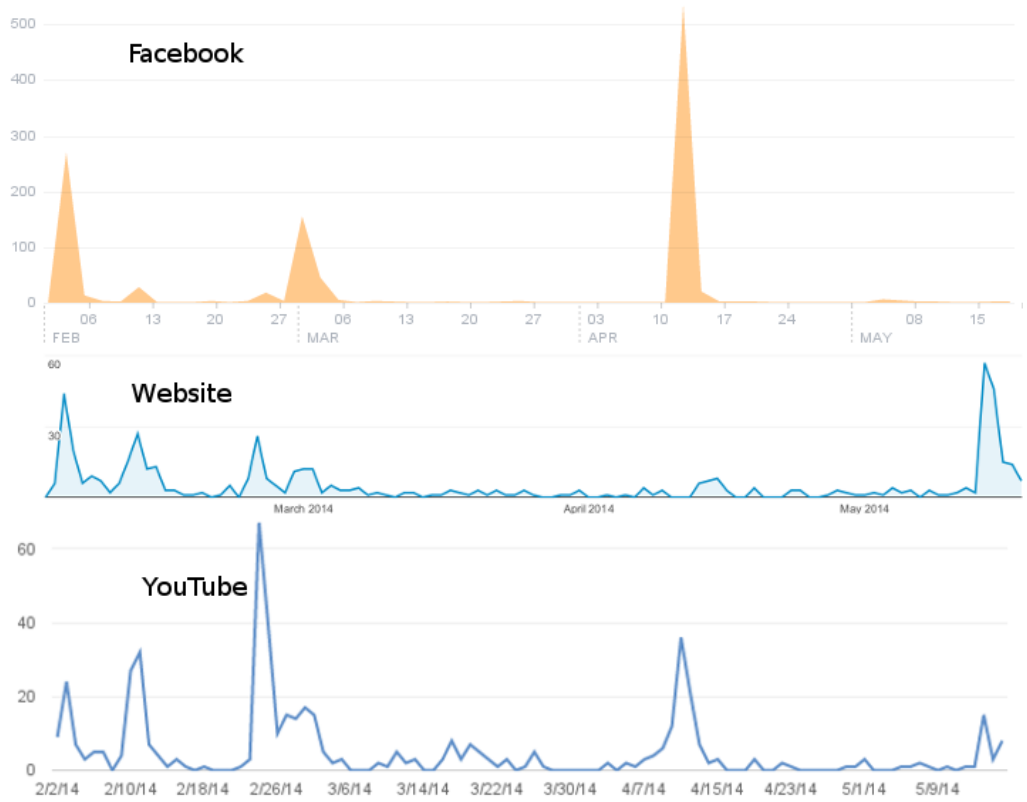


Figure 69: Top graph shows the post reach on Suttung Digital's Facebook page (y-axis is how many post views). Middle graph shows the number of website sessions per date (y-axis is number of page sessions). Bottom graph shows the total video views on Suttug Digital's YouTube channel (y-axis is views). The graphs have been scaled to match the same timeline.

Figure 69 shows graphs for interactions on the website, Facebook page and YouTube channel. The first spike on February 2nd is from the milestone 1 blog update[7] that was shared on Facebook. Two posts were shared: one with links to the site and one with the video embedded. The next spike is from sprint 4 blog update[8] February 12th. This time it was posted as a link, which was seen by far less people on Facebook, and therefore also gathered less traffic on the website. The spike at February 24th was a blog post

[5] reddit URL: http://www.reddit.com/r/gamedev/comments/25op7q/feedback_friday_81_sky_high/chjvhdv

[6] TigSource URL: http://forums.tigsource.com/index.php?topic=40672.0

[7] Milestone 1 post: http://hovedprosjekter.hig.no/v2014/imt/spill/pyroeis/2014/02/02/first-milestone-reached/

[8] Sprint 4 post: http://hovedprosjekter.hig.no/v2014/imt/spill/pyroeis/2014/02/11/menu-guns-and-optimization/

about sprint 6[9]. This was shared as a link to the YouTube video on Facebook and did not get much reach there, but it gathered the highest number of views on YouTube. It was also shared on twitter as link and as a YouTube video. On March 1st, a blog update for milestone 2[10] was posted and shared. This time it was shared as a normal text post on Facebook, and also as a YouTube video on Twitter. The Facebook post reached more people, but the YouTube channel and the website received less traffic. After this, there was a long period without any updates. The website and YouTube channel got a few views each day during the period. On April 10th, a new post was published on the blog for sprint 12[11]. It was shared as a text post on Facebook, and as a video on Twitter. The Facebook post reached a record number of people, and the YouTube video got a decent amount of views. The website analytics was broken because of an update, so the graph is also broken. On May 14th a new version was added to the download site, but no blog post was written. Both of the authors asked their friends and family through Facebook to test the game and linked to the download page. They also shared posts on their personal twitter accounts and on the Pyroeis twitter account. No post was published on Suttung Digital's Facebook page. The website gained a record number of views, and YouTube got a few more views.

The statistics show some important points:

- A normal text post on Facebook reaches more people than a link.
- Embedding the video in a Facebook post provides the most video views.
- Website traffic is heavily affected by the Facebook posts, but also tweeting about the game gathers pageviews.

A full export of the website analytics is in Appendix N. Most of the visitors are Norwegians, which most probably is only friends and family of the authors, or friends of friends. This is due to that the Facebook page almost only has Norwegian followers, and the ones that re-share posts have Norwegian friends. Some visitors are foreign and most likely come from Twitter, reddit, or TigSource.

6.2 Building Packages

Each system uses a unique way installing software. Windows is known for the exe installer files. OS X uses app files. While Linux uses different methods depending on the distribution. The various methods of packaging the game on each distribution will be discussed below.

6.2.1 Linux

Normally, systems are distributed as one system with a certain version, like Windows and Mac OS X. Linux is not like this. It is only a kernel, and the tools and libraries needed to make the system usable are packaged together with the kernel in the form of distributions, which there are several hundreds of. The nature of Linux distributions makes it hard to distribute a commercial program. One could target one or two specific

[9] Sprint 6 post:
http://hovedprosjekter.hig.no/v2014/imt/spill/pyroeis/2014/02/24/its-a-game/

[10] Milestone 2 post:
http://hovedprosjekter.hig.no/v2014/imt/spill/pyroeis/2014/03/01/sounds-inventory-and-new-enemies/

[11] Sprint 12 post:
http://hovedprosjekter.hig.no/v2014/imt/spill/pyroeis/2014/04/10/nearing-the-end/

distributions at a specific version, but a lot of users would be left in the dark.

Packaging

Normally Linux distributions provide software through a package management system, like Arch Linux's pacman, or Gentoo Linux's portage, where they fetch packages from a repository. The packages in the repository used for the package management is mostly maintained by the developer community around the specific distro, and not the upstream software developers. Because of this, licensing restrictions and library version incompatibilities, it is mostly open source software that makes it to the distribution's repository. Even if Pyroeis is compatible with the a distribution's libraries, it would require a community that want to maintain the package for their distribution to keep it in the repositories.

It is possible to distribute the software as a package without having it in any regpository, but this would require us to package it for all of the different package managers. To avoid this, the software is provided as a simple portable compressed tarball. The user of the software only need to unpack the tarball and run the binaries or scripts provided. The users are not able to install the software to their system as with a package manager, but it can easily be run by users on many distributions.

Libraries

Libraries on Linux systems are traditionally packaged and provided by the distribution's maintainers and contributors. Because of this, software provided must be linked with the library versions available, so that that the soname of each library matches the soname the binary linked with. With open source software this is not a problem since the package maintainer can link the software against the libraries provided by the distribution. This is not possible with closed source software like Pyroeis.

There are at least three possibilities to go around this problem: Statically linking, dynamically open libraries at runtime, or bundling dynamically linked libraries (shared objects) with the package. Statically linking, bundling the shared objects, or a combination are the most commonly used techniques as they will ensure that the user always has all libraries necessary. Pyroeis first statically linked all libraries except for system libraries like libstdc++, but moved to bundling shared objects after some problems were discovered.

Both bundling shared objects and statically linking have problems. When bundling shared objects, the dynamic linker has to know where the libraries are. The linker searches through the system defined paths like "/usr/lib", but not the directory that the binary is run from. To let the linker know where the libraries are, the environment variable LD_LIBRARY_PATH can be set to the library directory, or the rpath linker variable can be set to a relative path to $ORIGIN.

Linking statically adds all library code used to the final binary, so the dynamic linker doesn't need to load the libraries at runtime. It also removes the need for bundling extra files that might confuse the user. The problem with this solution is that many copy-left licenses on libraries doesn't allow programs to statically link them without releasing the program source code under the same license, making it open source. None of the libraries that Pyroeis link with has a license like this, except libstdc++, which is an important library since it provides the C++ standard library. Normally the library provided by the system is used, but since Pyroeis uses C++11 standard features, and some distributions use old versions of libstdc++, a new version of the library has to be bundled with the

software. If not, older distributions will get an error when running it.

Because of the limitations of statically linking, dynamically linking and bundling the shared objects were chosen as the solution. Libraries other than libstdc++ can be statically linked, but this has no benefits when there already are bundled shared objects. Having shared objects also gives the user the ability to update a library used, or use a modified version. This can for instance be useful if a system update breaks the library. Package maintainers can also choose to use the system libraries if they are compatible. To let the dynamic loader know where to find the libraries, LD_LIBRARY_PATH environment variable is set when launching the game. To accomplish this, a script that handles the launch process has to be used.

Listing 6.1: Linux launch script.

```
LD_LIBRARY_PATH="$(pwd)/lib:$LD_LIBRARY_PATH" ./data/pyroeis
    $@
```

When building the game with GCC 4.9, it also needs to bundle the libgomp shared object for OpenMP support. This is because GCC 4.9 uses OpenMP 4.0, which earlier versions do not support.

Installing

It is mentioned above in packaging that the application is not installed to the system. While this is true, the package provides a script that will create a desktop entry for the application. This lets the user start the application from an application menu such as Ubuntu's Unity, dmenu, or Gnome's activities. The script is written in bash.

Listing 6.2: Linux desktop entry script.

```
game_name=pyroeis
application_dir=~/.local/share/applications
application_entry=$application_dir/${game_name}.desktop
game_dir="$(pwd)"

if [ ! -e "$game_dir/launch_pyroeis.sh" ]; then
  echo "Did␣not␣find␣launch_pyroeis.sh.␣Make␣sure␣you␣are␣
      executing␣this␣file␣from␣the␣same␣directory."
  exit 0
fi

real_game_path=$(readlink -f "$game_dir")

if [ ! -d $application_dir ]; then
  read -p "$application_dir␣does␣not␣exist.␣Create␣it?␣[Y/n]
      " answer

  if [[ $answer =~ [Nn]$ ]]; then
    echo "Cancelling␣installation."
    exit 0
  fi

  mkdir -p $application_dir
fi

echo "[Desktop␣Entry]
```

```
Version=1.0
Type=Application
Name=Pyroeis
Comment=Play␣Pyroeis
Icon=$real_game_path/data/icon.png
Exec=$real_game_path/launch_pyroeis.sh
Path=$real_game_path
Categories=Game;" > $application_entry

echo "Desktop␣entry␣installed␣as␣$application_entry"
```

Creating the Package

First the binary has to be build. A CMake build file is used for this. By defining the symbol `PYRO_BUILD_RELEASE=ON` with the CMake script, the script will try to enable as much optimizations as possible and strip away any unnecessary debug symbols. It looks for link time optimization (-flto), the best optimization (-Ofast), finite math (-ffinite-math-only), and symbol strippint (-s). Then it enables all options available.

To package the binary, libraries and assets together into a package, a bash script is used. It copies all data that should be in the package to a new directory, then includes all shared objects if desired, sets the file format, and creates the package with “tar -cJfv”. The -J option compresses the package with xz, which is slow, but has a good compression ratio.

To generate the name for the package, the date command is used together with “git log”, and a version name provided as an argument to the script.

Listing 6.3: Linux package creation script.

```
if [ ! -d package ]; then
        mkdir package
fi

pushd package

if [ ! -d pyroeis ]; then
        mkdir pyroeis
fi

cp -r ../assets pyroeis/
cp ../docs/controls.txt pyroeis/
cp ../docs/readme_linux.txt pyroeis/
cp ../scripts/linux/install_desktop_entry.sh pyroeis/
cp ../scripts/linux/launch_pyroeis.sh pyroeis/

if [ ! -d pyroeis/data ]; then
        mkdir pyroeis/data
fi

cp ../build/bin/pyroeis pyroeis/data/
cp ../data/icon.png pyroeis/data/

shared_objects=false
```

```
for arg in $@; do
        if [ $arg == "--include-shared" ]; then
                shared_objects=true
        fi
done

if [ $shared_objects == true ]; then
        mkdir pyroeis/lib
        cp -d ../lib/*.so* pyroeis/lib/
fi

date=$(date +%y%m%d)
commit=$(git log --pretty=format:'%h' -n 1)
version=$1

tar -cJvf pyroeis_${version}-${commit}-${date}_linux-x86_64.
   tar.xz pyroeis
rm -r pyroeis
popd
```

6.2.2 Mac OS X

Distribution on the OS X platform can be done in two different ways, through the creation of a Unix Executable File or a app file. Apple's own App Store uses the app format exclusively, and the format is overall simpler for the platform. If the application was distributed using a normal Unix Executable File, all handling of assets would be left to us.

Packaging

OS X packaging is done with Apple's app sandbox. [38] The app package wraps around the needed libraries, the executable and the needed assets. As the OS X version is built using Apple's XCode, the process is very simple. We simply need to indicate what Frameworks, libraries and assets that need to be copied to the app, and their location.

Libraries

We did however have a few issues with the inclusion of some of the libraries. XCode does not handle dynamic libraries well. Developers have to manually set the paths or run scripts that set the proper search paths for the runtime executables. While this is possible to do, We found it easier to simply build our own, static versions of the libraries. We also found no benefits of using dynamic libraries on Mac OS X.

Version Incompatibilities

Since Mac OS X was used as one of the core development machines for the project, we have not had too many Mac specific problems. However we have had a great deal of issues with backwards compatibility in relation to the newly released OS X 10.9, Mavericks.

With the new SDK 10.9 the backwards compatibility for machines running the previous versions is gone for some application. In our case this is caused by a missing ___sincos_stret symbol in the /usr/lib/system/libsystem_m.dylib library. Research lead us to finding that this is a performance optimization done in the latest SDK. [39] Due to a lack of documentation and a lack of users experiencing the issue, the bug was hard to track

down. We were ultimately able to solve the problem by compiling everything using the previous version of the SDK: 10.8.

We also had some limitations based on the choice we made to use C++11 as the platform language for our engine. C++11 is not supported by machines running OS X 10.6 or earlier install. When deciding if this would be considered a threat or not, we simply looked at the adoption rates for the various versions of the OS X operating system. 80% of machines are running 10.7 or later. [40] This and the fact that you will need a fairly recent machine to run the game at decent frame rates lead us to the conclusion that using C++11 was a good choice.

Installing

To make the process of installing the application a little more convenient for the user, we package the Pyroeis.app together with a shortcut for the users Applications folder in a dmg file. This allows users to simply mount the dmg, and then drag the application into the Applications folder. The installation process has a big problem currently. As we have not yet signed the application, it is seen as a threat on OS X, and users have to allow the application to run by going to their security settings. Although this is annoying for our testers, it is easily solved by signing the code. [41] The reason we haven't done this is that it requires a Mac developer licence. As we are not distributing for sales yet, this seems unnecessary.

6.2.3 Windows

We use the Inno Setup software to package the executable and the required libraries and assets for distribution. [42] Inno Setup allow us to set the installation path for the program and set how the files should be organized for the application to run. Because we are compiling the application using Microsoft's Visual Studio, we also need to distribute the needed packages for this. [43] After the installer is created using Inno Setup, we bundle both the redistributable Visual Studio install and the Pyroeis install using the built in windows tool: iexpress.exe. This allows us to install both the executables.

Problems

Because of Windows' anti malware protection, users who try to download and install Pyroeis will be promted with warnings that the file is a risk to the system. Users will then have to go through a tedious process to download the installer. When the installer is downloaded and the user tries to use it, Windows will again complain that the file is not safe. The fact that the users have to jump through so many hoops in order to play the game is bad. In order for us to fix the issues, we would have to sign the exe. This allows Microsoft to verify that the executable is trusted and safe.

When an application is installed in Windows, it is not given write access to the folder where it is installed. Although we were not trying to write in the install folder we were using fstream in order to read asset files. Because fstream opens up both a read and a write stream, the operating system complained when trying to run the executable from the install folder. The fix was to simply use ifstream instead, as this only opens a read stream, it does not violate the access control system.

7 Testing and User Feedback

During development a few users played the game and provided some feedback. We also got to see how they handled the controls. One of the first pieces of feedback most players would give us was that the jetpack is too difficult to control. They were struggling with keeping the balance of the jetpack and kept bumping into the ground. Players that have played the game a few times however seem to love the experience of learning how to control the player in the environment.

7.1 Planned Feedback System

During our first meeting with Jayson Mackie, we discussed how to get user feedback and collect game play statistics. The log for the meeting is found at the first page in Appendix I. We decided that we would use two systems: one would collect anonymous game play statistics about how the player plays the game, the other would let users easily provide feedback about the game in it. These systems were planned for milestone three, found in Appendix C, as the product in this stage would be a playable game. Because other core features was still missing at this stage, the testing system was completely removed from the plan. Collecting these data would be useful, but the game never reached the stage at which it could be used. Only at the current state of Pyroeis, the systems could start being incorporated. None of the systems were planned in detail and written down, but the idea behind them and how they would work is discussed in the subsections below.

7.1.1 User Feedback System

It is tedious for players to send a mail, comment on the web page, or send a tweet to give feedback on the game. If there is a simple way to give feedback directly from the game, it is likely that more players will provide feedback. It can also provide the developer with more useful data of the game state when the feedback is sent. This was the idea behind it.

To access the feedback system, the user will need to press a specific key, or a button on the screen. Then they can write a message containing their feedback, which can been an issue or just something they like. When the player then sends the feedback, the program collects a defined set of data about the game state and packages it with the message. This data can be any useful. With only the world seed and player position we can load the exact same world on the exact same position and see the game state. It can also collect information of all actors and destructions that happened in a certain area, so that developers can properly re-create the game state. The data will be stored in a database with a simple web interface for developers to access.

7.1.2 Data Collection

Data about each game session can be collected to analyze how players play the game. This can be used by developers and designers to tweak the game so that the players get the intended experience, or don't get bored. Simple data like the playtime, length moved,

time in jetpack mode, bombs used, dust collected, etc. can be stored in a database and be analyzed later. Other more interesting data like the actual path the player moves, together with the speed, can also be collected. Event a whole game session can be recorded and stored. There will be too much data if the state is recorded every frame, but taking snapshots a few times a minute is possible. The developers can then replay a session and see nearly exactly how a player plays the game.

7.2 Feedback During Development

During development, a few users tested the game and provided feedback. Close to the end of the project, the game was sent to friends, family, and various websites for feedback. The websites published to are described in Section 6.1. A raw list of some of the feedback received is in Appendix M.

One of the first pieces of feedback most players would give us was that the jetpack is too difficult to control. They were struggling with keeping the balance of the jetpack and kept bumping into the ground. Players that have played the game a few times however seem to love the experience of learning how to control the player in the environment. Players also found the GUI and control interface difficult. Many did not look at the controls description and therefore did not understand that there is more equipment, or that there is a jetpack. Other bugs and issues that they thought was unfair was also reported. Players liked the atmosphere and the ability to destroy the world with explosives.

Almost all of the feedback we received was positive, except for the things mentioned above. Although this is very fun for us, it is not the most useful experience. This highlights one of the more difficult tasks of the testing: People will be very positive and kind when they have something to gain from that. For our friends it's probably because it's hard to criticise a friend. For the people on twitter it's hard because they're interested in getting feedback from us.

7.3 Usability Test Process

Since the planned feedback system didn't make it into the game, and because it won't directly answer design questions, a testing process was made. There are two main categories of usability testing generally used for games: usability expert testing, and usability testing. [44] As the first requires experts in the game usability area, and that it doesn't directly represent users' opinions, the usability testing method was chosen.

Normally when doing usability testing, a user in the target group plays the game in a room with an instructor. The instructor is there to ensure that the user tests the areas planned. It is important that the user isn't affected by the reviewers, so that the game experience is as close to as it would be for a normal user buying the game. To do this, game designers are evaluating the experience through a one-way mirror. The game session is also recorded so that the experience can be reviewed in closer detail. Not everything can be caught by watching once.

Our test is based on this, but merges the game designer and inspector. The user sits in a room alone with the inspector and plays the game. They can do whatever they want to, as if they played the game for the first time, alone. The inspector looks at what the player does and notes any thing of interest. They also ensures that the player tests all features of the game. The user talks about their thoughts out loud, so that the inspector can note important things down. At the end of the test, the user is asked to draw one

curve for the experience, and one curve for the difficulty, through time from start to end game.

The key elements that the test will analyze is the experience, difficulty, and problems.

7.4 Usability Test Results

Three users have gone through the test. Some of them have played the game a few times already in earlier stages, so they know some of the functionality. For balanced and realistic results a larger group of testers is required.

7.4.1 GUI

One of the biggest problems is the GUI of the inventory and weapon screen, especially the weapon screen. Everyone has problem understanding how to accomplish a task, and what different buttons does. In the inventory screen, one user though the quantity of items available was the number of items that you will get when crafting. Another user didn't see that you could craft items from dust until the inspector told them.

The weapon screen is worse. Every user had problems understanding how the power system for the weapons worked. Most of them did not see the mapping between the power bar and the weapon property bars, but understood that they can upgrade each property. They were also experimental by using their gems on a stat, then later realizing that they wasted valuable resources and the wrong thing. Realizing this made the experience bad.

7.4.2 Controls

One thing that has been difficult through the whole project, is controlling the jetpack. Every player that tried the game struggle with it, using a lot of time getting to know how to control it. What was interesting with the test was that all players that already had played the game for a while liked the controls. They said it felt natural and logical, and felt like they mastered it. Though they still felt that it was hard in the beginning. Some mastered the controls better than other, but everyone learned the controls well after playing 3–5 times. This also reflects the feedback received during development.

All of the testers nearly never moved using the normal on-ground controls. They always used the jet pack. One user tried to move on the ground just to test it, but because of the difficult terrain, they went back to the jetpack.

Users also had trouble controlling jetpack while aiming with the weapon and looking around. Keeping focus on both the body rotation and environment around them is difficult. Aiming was also difficult when reticle is hidden in the dark, or in the bright light.

7.4.3 Weapon Customization

None of the players used the weapon customization system much. Mostly they experimented with the weapon GUI a bit, using up all their gems. Then they never touched it again. One user tried to collect enough dust to upgrade one more power, but the effect was so minor that they didn't bother doing it more. When asked why they didn't use the system they said it was because of the short game length. There was no reason to upgrade the weapons, and the number of dust they collected wasn't enough to upgrade much.

During the game play, the inspector demonstrated some of the possibilities with the

weapon customization system. When the player finished the game, they were asked if they wanted to play it again, and why. They all answered that they wanted to play again to experiment with the possibilities of upgrading their weapons.

7.4.4 Dust Collection

One of the most enjoyable experiences in the game was collecting dust, especially yellow dust dropped in large piles by monsters. The way that the dust flies toward you was enjoyable, and seeing a large pile of dust flying towards you after surviving a large fight was very satisfying for the player.

Green and blue dust is easy to collect since all plants drop it. All players seemed to understand this and while flying through the caves they repeatedly shot plants to collect their dust. None of the players especially liked collecting red dust. The only way to collect it is to destroy the world with bombs, and there was no incentive to use bombs. The only time they used bombs was when a path was too tight to get through. One player also used bombs to get down to the boss layer, but they said it was a very boring and repeating experience: throw a bomb, explode a bomb, and repeat.

7.4.5 Weapon and Item Usage

Every player used the shotgun through most of the game. Some were more experimental, trying out the different weapons, but finally went back to using shotgun. When showing them the possibilities of upgrading weapons, especially the multi-bullet ability of the machine gun, the weapon usage changed to be more experimental.

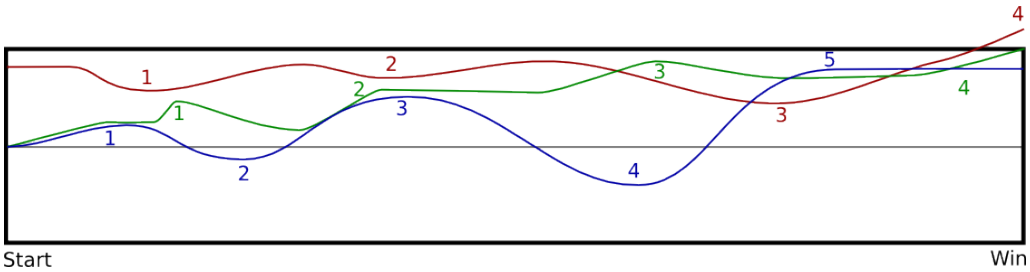
One item was used all the time, one was nearly never used, and the two other were used a few times. The item used all the time is the health kit. All users quickly understood that having health kits available could save their life in many situations. They used time to collect green dust from plants so they could use health kits. Flares was nearly never used. One player used it once and thought it looked cool, but never used it again. The lamp was never used during normal play, but most of the players used them to light up the boss cave before or while fighting the boss swarm. Bombs was only used to get through small paths, though one player used it to get through a large chunk of wall.

7.4.6 Boss Figth

Every player liked the excitement of fighting the boss swarm, but two of the players also mentioned that the game should feature a large single boss rather than, or in addition to, the swarm. Because of the large open areas they expected something like this. They also expected something special to happen when they beat the boss, like unlocking new equipment.

7.4.7 Game Experience

Each tester drew a curve representing their game experience through time from when they started playing to when they won the game. The bottom of the graph is where they would quit playing because they think it is bad. The top is where they think that the game is really cool. The middle line is an OK experience. Red and blue player won the game at their first try, while the green player won at their third try. The graph is from their last game where they won. The players draw their curves in different ways since they have different perception of what is the top, middle, and bottom. This does not matter. The most important thing is the variation in each curve.



Red player thought the game was fun through the whole game. Early in the game, at 1, they tested the inventory and because of its bad design it was confusing and they were frustrated. Continuing they encountered some plants and enemies and thought the game was fun, but then they realized that they wasted all their resources on upgrading things that did not matter at 2. Continuing it was fun, but then they got stuck and decided to explode a path down with bombs at 3. This was a boring experience. After getting through the exploded path, a large amount of enemies were encountered and the excitement increased. Then the boss swarm was encountered and the excitement was so high that they wanted to draw it above the graph limit, at 4.

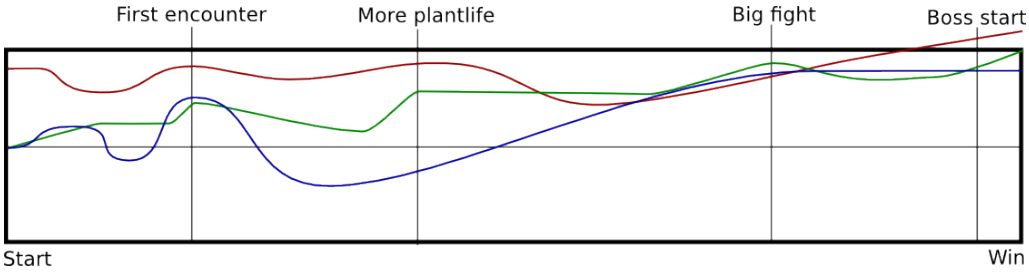
Green player had an experience that increased through the game, with some pits. At 1 they met their first enemy, but then there was nothing happening until 2, where new exciting plants were encountered. Because of the more interesting plants and terrain, the excitement was stable. Then, at 3, they encountered a large amount of enemies. They had a hard time trying to survive, but managed it. After the big fight, they went down to the boss cave, at 4, where the excitement started to rise again because they knew that the boss was coming. When fighting the boss the excitement went to the top.

Blue player thought it was exciting in the beginning because of the creepy atmosphere when descending into the caves at 1. It builds up a tension, but when time passes and there still are no enemies, the excitement falls to 2. After a while they encountered their first enemy, at 3, which was a very exciting experience. Then the excitement fell again since no enemies appeared and the player got lost at 4. They had to start flying upward. This was a boring experience that could have been OK if there were any enemies present. When they finally found the way down, they encountered a large cluster of enemies at 5. The fighting was intense and the excitement rose. Then they went down into the boss cave and had the same amount of fun fighting it.

All players, except blue, thought the beginning was boring since there were not plants or enemies. Blue player thought this was good since it built up tension, but because of the long time before the first encounter, it wasn't utilized very well.

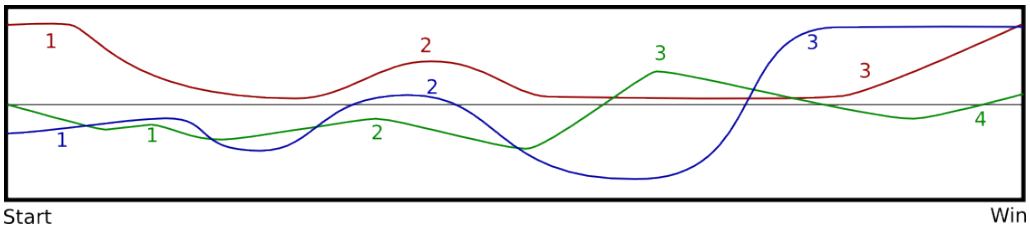
The graphs have some similarities and some differences. From each game play there was a matching pattern of events. First there were no enemies, then a few of them are met a few times with long boring intervals. Then they reach a larger cave with many enemies making them fight for their life. After this they prepare for the boss fight and then fight off the swarm of boss enemies. Some of these events are easy to see that match from the graph, but some are not since the players encountered them at different times. To help visualize the experience based on events rather than time, a second version of the graph has been made where points have been moved in the x axis to match the events. The new graphs match much better. Between enemy encounters the experience is bad.

But with each encounter, and especially the larger ones, they get really excited and their experience is great. Each player liked when the plant life became more varied, but since blue player got lost and had problems finding their way down, they didn't like it as much.



7.4.8 Difficulty

The difficulty curve for each player is similar to the experience curve, with one exception: While the experience is better with more plants, the difficulty isn't harder. Bottom of the graph is too easy, while top is too difficult. As with the experience graphs they are drawn differently because of the different perception of bottom, middle and top. Again, the most important here is the variation.

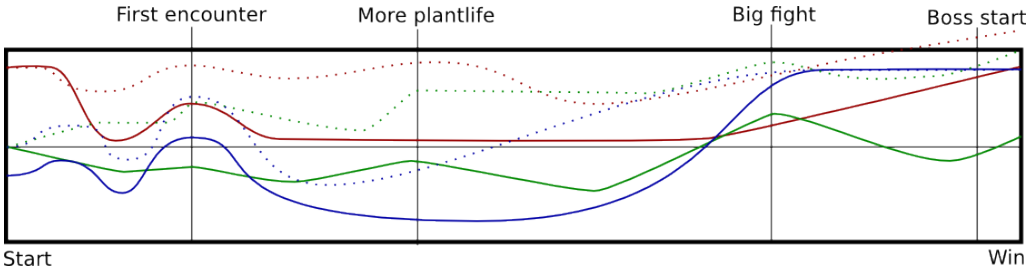


Red player had trouble in the beginning, at 1, since the inventory GUI and controls were difficult. The after learning the controls, everything was easy, until 2, where they met their first enemy. After killing the enemy, they knew how the enemies acted and didn't think it was difficult anymore. But at 3 they met the large group of enemies. This made it very difficult, and it became even more difficult with the boss at the end.

Green player knew the controls well and thought it was easy until they meet their first enemy at 1. Nothing much happened before reaching 2 where more enemies were encountered. Then the gameplay was easy before encountering the large group of enemies at 3. The boss was easier.

Blue player was very similar to red player. At 1 they had trouble with the controls, though they quickly learned to master them. At 2 they met their first enemy. Then at 3 the big group of enemies were encountered.

To more easily match the graphs, they were shifted like the experience graphs. From the graph we can see that the green player is the most unique. They didn't have trouble with the controls, the first enemy, or the boss. And they also met more enemies between the first enemy and the big fight.



The figure also compares the difficulty with the experience. For the three testers, the experience is affected by the difficulty. The first encounter is both difficult and exciting, and it is the same with the big fight and the boss. Green player deviates at the boss with thinking that it was fun, but not that difficult. Even though the difficulty gives the player a good experience, it isn't the only thing. The varied plant life enhances the experience.

Each player has their own opinion about the difficulty. Red player thinks it is a bit too easy, and that shooting should be punished by having a limited amount of ammunition. They also thought fall damage sometimes was too punishing. All of them thought that the difficulty was too varying. It was either not difficult at all, or very difficult.

7.5 Analysis of Test Results

The weapon GUI is too difficult to understand. It ends up frustrating the player when they waste all their resources, and often they won't use it after this. To solve this it has to either be redesigned to be more intuitive, have a tutorial describing how it works, or a combination of both. The biggest problem was that the player didn't understand that they needed to power up the property slots in addition to unlocking them. The system is based on FTLs power system. FTL has a more natural mapping between the power bar and the properties, ensuring that the user sees the connection. The power slots are also completely transparent, except for the borders, when empty, better indicating that it is just is an empty slot that needs to be filled. Enhancing the GUI with a more natural mapping like in FTL would enhance the experience.

The jetpack controls were surprisingly well received after they had played a few times. It is still very difficult for new players to handle the controls, which might result in them quitting before they learn them properly. The results tells us that the jetpack controls should stay as they are, but the learning curve should not be as steep. The ground movement controls were never used and it therefore has no use in the game. Removing it completely would lessen the complexity by removing a mode that the user have to switch between.

The short game length with the straight progression gave no incentive to upgrade or experiment with weapons. By looking at the test results, the upgrading system and all weapons except the shotgun could be removed, as they had no use. This conclusion is correct for the state that the game is in now, but not for the larger vision of the game. For the current state of the game, upgrading weapons could be made cheaper to let people more easily experiment with the weapons. The possibilities could also be introduced to the player through a tutorial, as all testers found the system to be interesting when they saw what it could do.

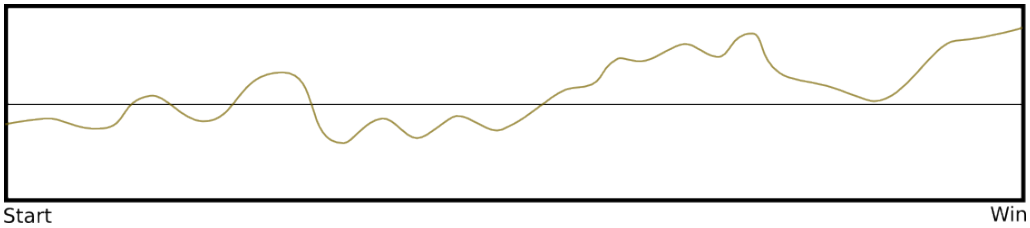
Lamps were used when the player was prepared for an event, like the boss warm,

but not anywhere else. None used flares, though one player stated that if the flare would slowly descend in the air, like it is parachute-suspended.

As suggested by one tester, the urge to explore the world could be enhanced by adding some mystery to the world. Putting interesting or useful artifacts hidden around the world that the player has to look for.

Especially one player had problems noticing that their health was low. One time they died because of it, and other times they nearly died. It should be more apparent when the player has low health. This could be done in several ways, but what often is used is to put a red vignette on the screen, or play a strong heartbeat sound. Indicating how much health that is lost when taking damage could also help the player see when they need to be careful, or heal.

One big problem throughout the game that repeated itself for all testers, is that the difficulty curve and the experience curve is not varying enough: that is, it is either not difficult or exciting at all, or very difficult and very exciting, and it takes a long time until it varies. Having a curve that is varying is better than a straight line, since the experience then is not repeating. The ideal difficulty curve for the current game would be an overall linear curve that starts off easy, but steadily rises, with many smaller variations.



This is an example of a difficulty curve that could be better. It has many smaller variations providing the player with a varying level of challenge. The varying difficulty also makes the player feel that they master the game after each peak. At the beginning it is easy, but there are some difficult peaks challenging the player, but when they are done with the big peaks they get to feel powerful for a while before the difficulty starts climbing. It is a steep climb, but during the climb there are several small pits where the player can get the feeling the mastery. After climbing the big peak, their skills have gotten better, and they easily take down enemies. Then, after some more climbing, the boss appears, giving them a good challenge.

Several factors could improve the difficulty curve. One important is to have a more balanced number of enemies through the caves, while still having some places where there are a lot of enemies packed. The types of enemies could also vary with the progression. It could start with just being some simple ranged enemies or the small children spawned by mothers, then progress to more dangerous enemies like the mother, and then combine them. The enemies could also have a damage and resistance that increase with the progression.

8 Discussion

8.1 Results

When starting the development of Pyroeis, we wanted to have a clear goal for what Pyroeis should be at the end of the project. In order to do this we spent the first few weeks thoroughly designing and planning every aspect of Pyroeis. The time we spent planning during the start of the project became invaluable as the development started. Although changes are bound to happen in any larger project, we were able to minimize these by having a clear plan for the development process from the very beginning.

Throughout the development of Pyroeis we have been heavily focused on the engine and the improvements we could do to it. Although we had just finished Project Nox, we had not done all the development on subsystems of the engine, as we were a part of a larger group. We started by going through every part of the engine, documenting parts that were lacking proper documentation and familiarizing ourselves with the details of the systems. This allowed us to get a good overview of the good and the bad parts of the engine, and also make sure that the engine fully fit the needs for Pyroeis. Our understanding of the limits of the game engine proved very beneficial as we started development. During the development we discovered various problems with the engine, and we had to re-write several systems to allow more scalability and to better fit our needs. The performance needs of Pyroeis were far greater than the ones we had for Project Nox. We had to do more multithreading, allowing the main thread to be free to update the game, while separate threads run to creat new areas of the world. This creates some very interesting issues from a programming perspective. The issues were extremely hard to debug, as we could not guarantee that they would happen. This caused some major slow downs for the development process. We had several weeks in development where the game did not progress, and we were focused on fixing issues.

Engagement with the public has been one of the more fun aspects of the project. Although the number of people that actually participated in the gameplay testing and provided us with feedback has been relatively small we have learned a lot from the experience. Getting people to engage in the content you are putting out is very hard. There is an extreme amount of indie game developers, all vying for public attention.

Although we are fairly happy with the point we were able to reach with Pyroeis, it is not as far along as we would have liked it to be. The gameplay aspects of the game are somewhat lacking, and there is not a clear purpose for the player. Pyroeis also lacks severely when it comes to a instructing players how to play the game. A tutorial was planned but we were never able to actually implement it in the game. We are however very happy with how far along we have come with the engine. Its very stable, flexible, and able to run on multiple platforms including Windows, OS X, various Linux distros and iOS.

8.2 Group Work and Workload

We decided early that magnus would be the group leader. This was done to ensure that we would be able to select one option in the cases where the discussion rendered us at an impasse. This was never a real problem during the development. Most of our disagreements came from cases where the other person did not clearly understand what the other person was saying. When this occurred we would spend enough time on the subject, and try to explain it differently.

As the development progressed we each became experts in our own areas. This lead to some problems for certain tasks. Although we would both be able to complete the tasks, it was much more convenient and efficient that the person who had been in charge of that particular system handled the extension of the system. This was especially true for some of the more complex systems, like the world generation.

The workload we have had during the bachelors has been hard. The amount of work each of us has put down in the bachelors (>800) hours is more than the expected amount for a full time student. The bachelors is currently only valued 66.6% of a full time student. We are happy that we have been able to focus our time primarily on the bachelors this semester.

8.3 Further Development

Although we are happy with the product we have created for our bachelors degree, we feel that the concepts in Pyroeis has a lot more game left. For every feature we implemented in the engine we came up with ten new ones that would make the product better.

8.3.1 Multiplayer

One of the bigger elements that we want to implement in the future is Multiplayer. Multiplayer in Pyroeis is something we have actually tested during development. In Project Nox, one of the requirements was that the game would feature network components. For Nox, this meant that we implemented multiplayer. Multiplayer was not a feature being developed for Pyroeis, we did however spend some Friday afternoons testing the feature just for fun. [45] Due to us being able to keep the interface between the new features in Pyroeis and the Nox Network implementation working, the implementation of network play is not hard to achieve. Multiplayer would bring a significant amount of fun to Pyroeis. The multiplayer element could be explored in various ways. Local multiplayer could let players join in by picking up a controller, helping the host to achieve his goals. Another perhaps more interesting form of multiplayer would be to let players join each other's worlds and work together to achieve the goals.

8.3.2 Crafting and Base Building

True to some of the games that have inspired us to create Pyroeis, we find adding a crafting and base construction element to the survival elements of Pyroeis is something that would allow the game's features to come to life even more. Crafting Is a commonly used element in survival games. It allows users to envision their own design in the gameworld, letting players invest time in shaping the world around in order to help them. The crafting element has been in multiple games of late, and was arguably made popular Mine Craft. In pyroeis, letting the player do meaningful crafting and base building would open

up the possibility for deeper gameplay. We could let players build and defend underground bases that contained means of transportation. Allowing players that spent their resources well to take control of the world. Players that died could again spawn in the same world, and continue the journey using the tools they had created in the previous run. This would allow us to create more epic journeys for the players, since the world is persistent, and does not vanish every time the player dies. Players would get more connected and invested into the universe of Pyroeis. Allowing progression, even though the player fails a few times.

9 Conclusion

When starting the development of Pyroeis we were free to create the project we wanted to make. Although this seemed like a daunting task at first, we are very happy that we were able to shape the project into what we felt was useful. Pyroeis as a game provides some unique twists on the genres that it borrows from. The 2D aspect of games like Terraria [15] is combined with more realisitc physics to create fun emergent game play. We have developed a strong foundation for doing procedurally generated, destructible terrain in various 2D games. We are very pleased with the progress we have made in this area.

A strong focus has been put into developing the engine throughout the project. This has paid off for us as it is now a flexible multi-platform engine. During the project we have highlighted multiple weaknesses in our engine and what the limitations are in the current state of it. This will be of great help as the development of the engine progresses after this project. Our long term goal is using the engine for professional game development in our company: Suttung Digital.

Although we have not been able to achieve all the goals we set for Pyroeis we are very happy with what we have been able to achieve with Pyroeis. It provides a great base for furhter development

Developing Pyroeis has provided us with many learning possibilities and challenges. We have developed our skills in programming, cross platform compatability and cooperation. For us Pyroeis has been a labor of love all the way through.

Bibliography

[1] Schwaber, K. & Sutherland, J. The scrum guide. https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf#zoom=100, July 2013. Accessed: 2014-05-19.

[2] Clark, W. & Gantt, H. L. 1923. *The Gantt chart, a working tool of management*. New York, Ronald Press.

[3] Waking mars. http://www.wakingmars.com/. Accessed: 2014-05-16.

[4] Ftl: Faster than light - generator system. http://ftl.wikia.com/wiki/Ship#Reactor. Accessed: 2014-05-16.

[5] Dictionary: Biome. http://www.merriam-webster.com/dictionary/biome. Accessed: 2014-05-16.

[6] Minecraft: Biomes. http://minecraft.gamepedia.com/Biomeandhttp://terraria.gamepedia.com/Environment. Accessed: 2014-05-16.

[7] Stack overflow: Minecraft biome generation. http://gaming.stackexchange.com/a/26532. Accessed: 2014-05-16.

[8] Physics: Fixed timestep. http://gafferongames.com/game-physics/fix-your-timestep. Accessed: 2014-05-16.

[9] Ogg vorbis. http://xiph.org/vorbis/. Accessed: 2014-05-14.

[10] Sdl_getprefpath(). http://wiki.libsdl.org/SDL_GetPrefPath. Accessed: 2014-05-14.

[11] Ai: Strategy pattern. http://en.wikipedia.org/wiki/Strategy_pattern. Accessed: 2014-05-19.

[12] Android reference: View. http://developer.android.com/reference/android/view/View.html. Accessed: 2014-05-14.

[13] Android reference: Activity. http://developer.android.com/reference/android/app/Activity.html. Accessed: 2014-05-14.

[14] Minecraft wiki: Chunks. http://minecraft.gamepedia.com/Chunks. Accessed: 2014-05-19.

[15] Terraria. http://www.terraria.org/. Accessed: 2014-05-09.

[16] Worms series. http://en.wikipedia.org/wiki/Worms_(series)#Games. Accessed: 2014-05-09.

[17] Feronato, E. 2013. How to create destructible terrain using box2d – step 1. http://www.emanueleferonato.com/2013/10/09/how-to-create-destructible-terrain-using-box2d-step-1/. Accessed: 2014-05-09.

[18] Feronato, E. 2013. How to create destructible terrain using box2d – step 2. http://www.emanueleferonato.com/2013/10/17/how-to-create-destructible-terrain-using-box2d-step-2/. Accessed: 2014-05-09.

[19] poly2tri. https://code.google.com/p/poly2tri/. Accessed: 2014-05-09.

[20] boost::geometry. http://www.boost.org/doc/libs/release/libs/geometry/. Accessed: 2014-05-09.

[21] boost::geometry::intersects. http://www.boost.org/doc/libs/1_55_0/libs/geometry/doc/html/geometry/reference/algorithms/intersects/intersects_2_two_geometries.html. Accessed: 2014-05-09.

[22] boost::geometry::within. http://www.boost.org/doc/libs/1_55_0/libs/geometry/doc/html/geometry/reference/algorithms/within/within_2.html. Accessed: 2014-05-09.

[23] boost::geometry::difference. http://www.boost.org/doc/libs/1_55_0/libs/geometry/doc/html/geometry/reference/algorithms/difference.html. Accessed: 2014-05-09.

[24] b2separator. http://github.com/delorenj/b2Separator-cpp. Accessed: 2014-05-09.

[25] Mark bayazit: Poly decomp. http://mnbayazit.com/406/bayazit/. Accessed: 2014-01-09.

[26] Martz, P. Generating random fractal terrain. http://www.gameprogrammer.com/fractal.html#midpoint. Accessed: 2014-05-19.

[27] Perlin noise. http://freespace.virgin.net/hugo.elias/models/m_perlin.htm. Accessed: 2014-05-19.

[28] std::thread::hardware_concurrency. http://en.cppreference.com/w/cpp/thread/thread/hardware_concurrency. Accessed: 2014-05-13.

[29] std::vector. http://en.cppreference.com/w/cpp/container/vector. Accessed: 2014-05-13.

[30] Nguyen, H. 2007. Gpu gems 3. http://http.developer.nvidia.com/GPUGems/gpugems_ch21.html. Accessed: 2014-05-13.

[31] Shadow types. http://en.wikipedia.org/wiki/File:Diagram_of_umbra,_penumbra_%26_antumbra.png. Accessed: 2014-05-13.

[32] Penumbra implementation. http://www.gamedev.net/page/resources/_/technical/graphics-programming-and-theory/dynamic-2d-soft-shadows-r3065. Accessed: 2014-05-09.

[33] Gaussian blur. http://en.wikipedia.org/wiki/Gaussian_blur. Accessed: 2014-05-13.

[34] Openmp. http://openmp.org/wp/. Accessed: 2014-05-09.

[35] Openmp clang implementation. http://clang-omp.github.io/. Accessed: 2014-05-09.

[36] Apple grand dispatch central. https://developer.apple.com/library/ios/documentation/Performance/Reference/GCD_libdispatch_Ref/Reference/reference.html. Accessed: 2014-05-09.

[37] Opengl required formats. http://www.opengl.org/wiki/Image_Format#Required_formats. Accessed: 2014-05-15.

[38] Apple app sandbox. https://developer.apple.com/library/mac/documentation/Security/Conceptual/AppSandboxDesignGuide/AboutAppSandbox/AboutAppSandbox.html. Accessed: 2014-05-13.

[39] Stack overflow: ___sincos_stret undefined symbol when linking. http://stackoverflow.com/a/19017286. Accessed: 2014-05-13.

[40] Apple os x distribution. http://chitika.com/insights/2014/mac-os-x-distribution. Accessed: 2014-05-13.

[41] Apple code signing. https://developer.apple.com/library/mac/documentation/Security/Conceptual/CodeSigningGuide/Introduction/Introduction.html. Accessed: 2014-05-13.

[42] Inno setup. http://www.jrsoftware.org/isinfo.php/. Accessed: 2014-05-13.

[43] Microsoft visual c++ redistributable packages for visual studio 2013. http://www.microsoft.com/en-us/download/details.aspx?id=40784. Accessed: 2014-05-13.

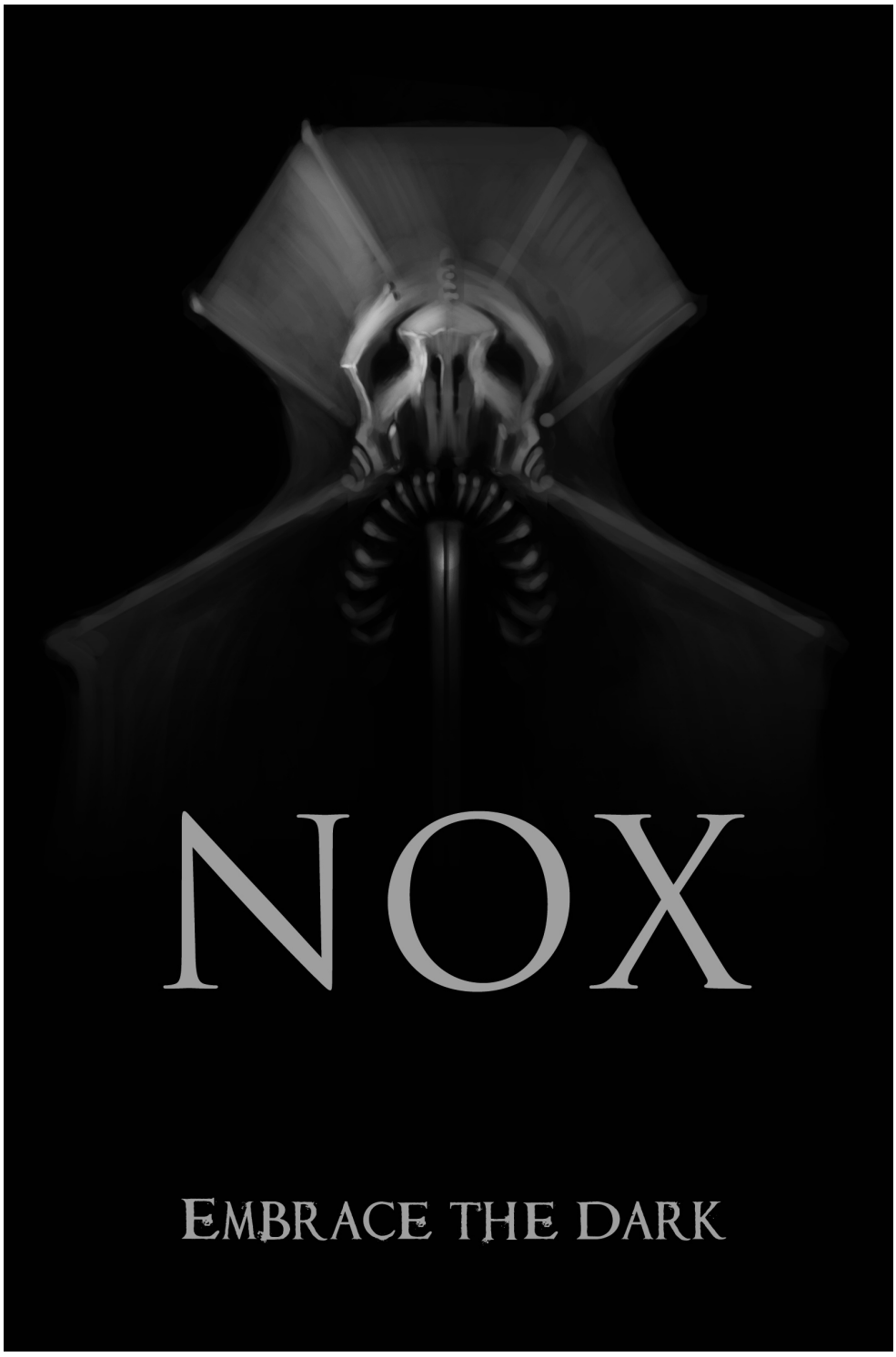
[44] Gamasutra: Better games through usability testing. http://www.gamasutra.com/view/feature/130745/better_games_through_usability_.php?print=1. Accessed: 2014-05-15.

[45] Pyroeis: Multiplayer testing. https://www.youtube.com/watch?v=xeA02d-WCX0. Accessed: 2014-05-15.

Appendices

A Project Nox: Report

Project report for “Project Nox” – the base for Pyroeis – from “IMT3601 Game Programming” at HiG. Work done by us, Tomas Klungerbo Olsen, Edvin Avdagic and Håkon Nordling. 20.12.2013.



Written by **Suttung Digital©**, for **Project NOX**
Asbjørn Sporaland, Magnus Bjerke Vik, Tomas Klungerbo Olsen, Håkon Nordling and Edvin Avdagić.

Table of Contents

Written by **Suttung Digital©**, for **Project NOX**

Asbjørn Sporaland, Magnus Bjerke Vik, Tomas Klungerbo Olsen, Håkon Nordling and Edvin Avdagić.

Introduction to NOX

Game Programming (IMT3601) of 2013 has covered a single large group-project. The goal for this project was to create a game, or at least an extensive prototype of a game. The project was in development during the whole first semester of 2013, and the assignment hand-in consists of the completed product and this report.

NOX is survival-horror game, developed mainly for the PC platform. The game is set in a modern day world, where you play the role of two brothers that inherits an old mansion from their grandmother Rosemary Jackson. Rosemary lived a solitary life in the swamps, and was never really close with the rest of the family. Puzzled by the fact that she chose to leave the house to them, they visit the mansion, unravelling secrets about their late grandmother as the brothers investigate her home. They will find out that Rosemary was involved in a secret cult, and practicing black magic. While wandering around in the house, they soon realize they're not alone in the dark, and something else lives there now!

Light and darkness play a huge role in the game experience for NOX. You will be thrown into a pitch black mansion, wielding only a flashlight that casts a small cone of light. This will be your main weapon, and it will be the only source of light at the beginning. The monsters that hunt you will do what they can to stay out of direct light and hunt you down from behind, or from a shadow in the corner of a room. Throughout the game you will find candles and other light sources that can be placed on the ground, creating a surrounding of light that will give you a little more safety. These light sources can be placed in rooms that are often visited, or any other place that you feel needs a little more visibility.

You are not alone in the dark. As NOX is a CO-OP multiplayer game, you will be joined by another player that will work with you to survive in the mansion. This creates possibilities for more tactical play. As both players have flashlights, you can work together to watch each others backs while solving puzzles or completing tasks, or you can split up and try to solve the mysteries faster.

The goal of the game is to unlock the door to the cellar, located in the middle of the mansion. Grandmother Jackson wrote numerous pages in her journal about some dark source of power residing in her basement. Exactly what it is, you do not know, but it seems to be the only key out of the mansion.

Development Process

In the development process for NOX we used the Atlassian suite of tools, mainly Bitbucket. This offers an effective way to use Git for version control, with a Bitbucket repository that has issue tracking. We assigned issues and bugs according to what parts of the code group members have been working on, and had code reviews for new implementations or fixes. These reviews were then assigned to those who wanted to do them, and that way they could have some knowledge of other parts of the system apart from the one that person is working on. Eventually we discovered that Bitbucket has a pull-request feature for the repository. Prior to this, we just assigned these pull-requests by oral agreement. Thus in development we had a more structured system for the requests when we started using Bitbucket's integrated feature.

We used the issue tracker throughout the whole development process for everything from bugs to major implementations. This was regularly updated with new issues, and solved issues were removed when merged with the master branch. Each issue had it's own branch until it was successfully tested and merged with the master.

What did we plan to do?

The game described in the introduction part of this report is how we planned to make the game. Although we had to scope the project down from it's initial design, we implemented the core features of the game, the light and AI mechanic.

The game mechanics and experience is based around the main mechanic of the game, the light. We wanted to create a horror-inspired atmosphere for the player using lights and shadows, and create an AI that takes advantage of the dark. The AI will react to the light sources by staying away, or at least move along the edge of them. This way, the player knows that they are present, but won't be able to see them completely. We wanted to implement smaller puzzle challenges that accumulates to a larger challenge, where the ultimate goal of the game is to open the cellar door. The initial idea was to give the player keys by completing the puzzle challenges, and that all the keys combined open the cellar door. The challenge would be to move through the mansion solving these puzzles and tasks while avoiding being killed by the stalking monsters. This could be done by using the light to your advantage and keeping the monsters away. Since there is no way of killing the monsters, there will always be a constant threat to the player. Only the light offers a safe haven, but the placed candles do go out eventually.

What did we implement?

We did not manage to reach all of our implementation goals for this project. We knew from the start the the project was a little over-scoped, but we had a list of priorities. We worked mostly with the core elements of the game to make sure we had a functioning system and a playable game. During development we updated the priorities according to what had been finished and

what elements were yet to be implemented. This meant that we finished a playable tech demo, but important gameplay aspects were left out, such as the puzzles and locked doors.

The core system was what we spent most time on, building the whole engine from different modules and connecting them together after successful testing. The system was designed during the planning phase, and a diagram of the classes and components was made to use as a reference when traversing through the code. This was especially helpful for group members working on a module that was written by another member. Slight changes to the system design was made during development, and this was added to the diagram.

The physics and event systems were some of the first components we implemented, along with the logger and actors that came eventually. Networking and resource manager where finished late during the development process, due to being large components to work on. The physics where changed rather early as well as we decided to use Box2D instead of writing our own physics.

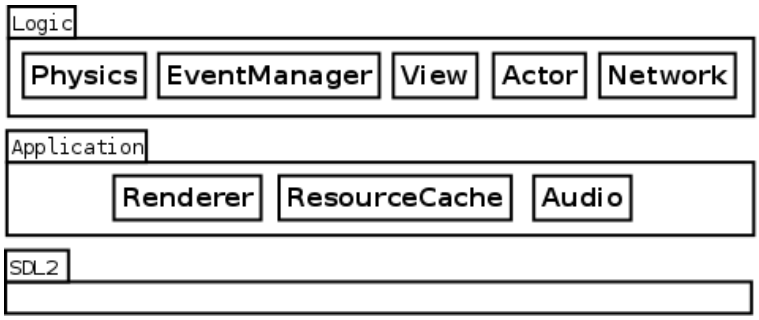
The light mechanic in the game was completed with a satisfying result. The AI will dynamically adapt to the environment of the mansion, seeking out the player by patrolling when it is idle, and hunting the player actively when they spot him. We use the A* pathfinding algorithm for the AI in the game. Each AI keeps their own graph of their current known state of the world. They need to be within a certain range, and have a clear view of an area, to update that particular part of their graph. This causes the AI to behave more dynamically in the world, avoiding collision with objects and areas that are lit up.

Program Architecture

The main focus of this project was implementing a good, reusable, and extendable architecture, therefore a lot of time has gone into implementing this rather than creating game play. The architecture is based on the theory from Game Coding Complete, Fourth Edition by Mike McShaffry and David Graham. It has two layers: Application and logic.

The application layer is the one communicating with the platform, which in our case is SDL 2, OpenGL, and OpenAL. It manages the rendering, the audio system, the resources, and handles events from the platform.

The logic is the layer that actually executes the game and should be independent of the platform. It handles the physics updating and syncing, broadcasts the events from the event manager, updates all the views and actors, and communicates over the network with potential clients/server.



Actors

The actor system is one of the most important systems in the program. An actor is a “thing” existing in the world. It can be a man, monster, plant, wall, etc. All that is in the world is an actor. The actors use the entity-component-system where each actor only is a container for components. Each component of the actor describe a piece of functionality such as transformation (position in the world), physics, audio, control, damage, health. Together they describe the complete functionality of an actor.

Actor components can communicate with each other by either sending/receiving events, or calling functions on the other components. This way the control component can apply forces to control the actor, and the light component can know where to cast light from.

This system gives us a great flexibility in designing actors with different sets of functionality and is a great step from the hierarchy system we have used before.

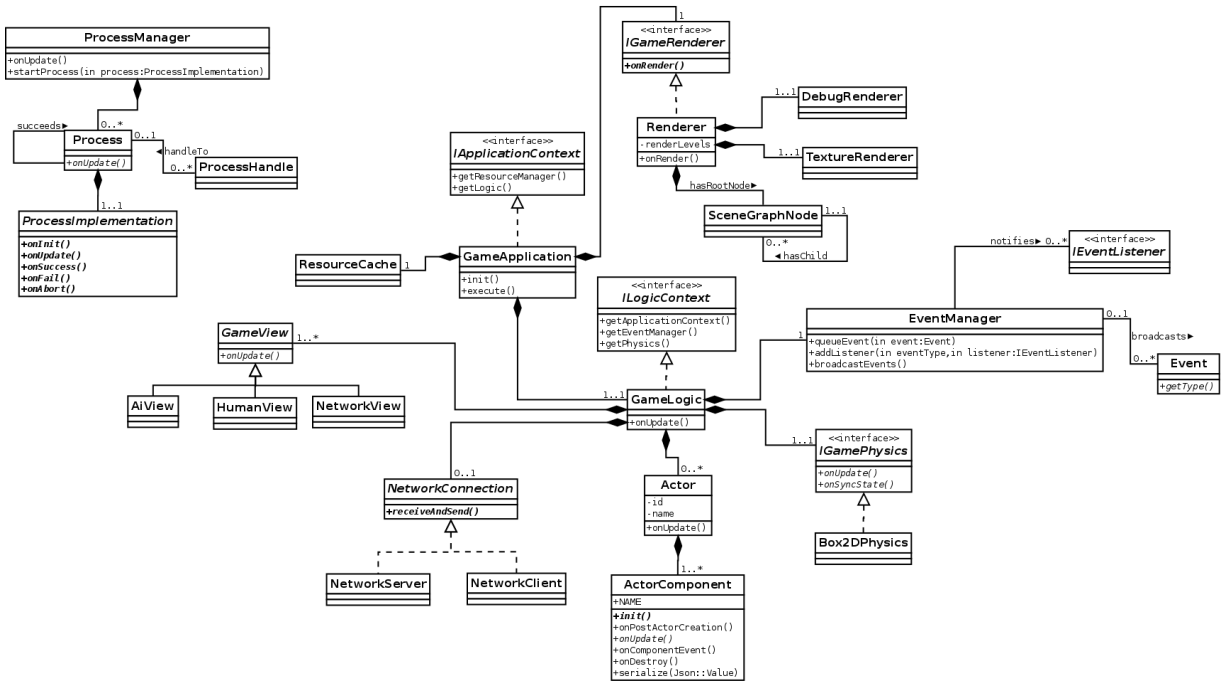
Views

A view is the interface to the game and will most probably control an actor. We have three types of views: Human, ai, and network. All the views interact with the system in the same manner, by sending events.

The human view is the one the player will interact with. It represents the window and handles input from the application layer. It will control and actor (the player actor) in the game. The ai view is a view that has an artificial intelligence and controls an actor based on the world state. The network view is a view that represents a remote player on the server side. It will receive events from a client over the network and pass it over to the local system to be handled.

Events

Most of the system is working by broadcasting events. This is to make the system more modular and with less connections. Different events are specialized by inheriting from the base Event class, and other systems can listen to specific events by asking the event manager to add a listener for that event.



Implementation

System Architecture

Game Coding Complete uses global pointers systems such as application and audio. In our system we avoid using globals and singletons. This is because singletons and and global pointers don't have a single instance that manages them. The whole program has the responsibility for them. It is also harder to manage the lifecycle of these.

What we did is a concept inspired by the Android implementation, a context. Bothe the application and the logic inherits from a context interface which gives access to the functionality of that class. The application owns the logic, so the logic is in the context of the application and therefore has a pointer to the application context interface. With this it has access to the things it needs from the application. Instances owned by the logic have a pointer to the logic context which they live in. This makes it much easier to see the layers of the system, where things belong, and the lifetime of them.

Actors

All the actors are defined with json files. We have a set of files describing different actors, like the monster, the player, a chair, a table, etc. Each of these json files have an array with each of the components of the actor. This makes it easy to change the behaviours of the actors and we don't need to recompile the program.

Actors are serializable with their components so they easily can be written back to a json file. This makes us able to easily save the state of the game by just writing all the actors to a file.

Process Manager

The process manager is a utility system used to run processes that last for several ticks, like animations and sounds. It is pretty simple and uses no load balancing, but it works for our purposes and is excellent to run the AI pathing and playing sound.

Physics

For Physics we decided to use the Box2D physics engine, this allows us to easily simulate all the objects in the world in real time, originally we wanted to make use of a self developed engine, but this was quickly changed once we tested the Box2D engine. We used the fantastic tool R.U.B.E as a makeshift level editor while setting up our own.

Rendering

We have set up a custom renderer using OpenGL. The renderer is optimized to draw a lot of both dynamic and static sprites.

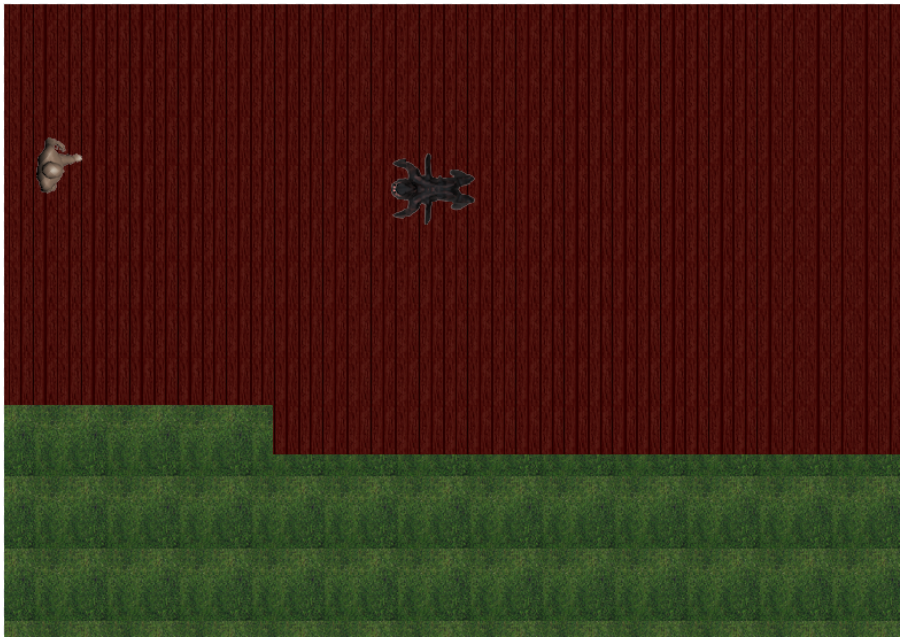
The game renderer uses a scene graph to calculate the render data. It has transformation nodes, sprite nodes, and tile nodes. The model transformations are all done on the CPU to limit the number of gl calls.

A texture renderer class manages all the textures to be rendered (sprites and tiles). Nodes that want to render a texture ask this class to get space for a specified number of quads, and updates this space. This class will automatically sort the data according to their rendering level and manage vertex buffer objects so it easily and efficiently can be rendered. In total there are two gl render calls to render all the sprites. One is for static objects and one is for dynamic.

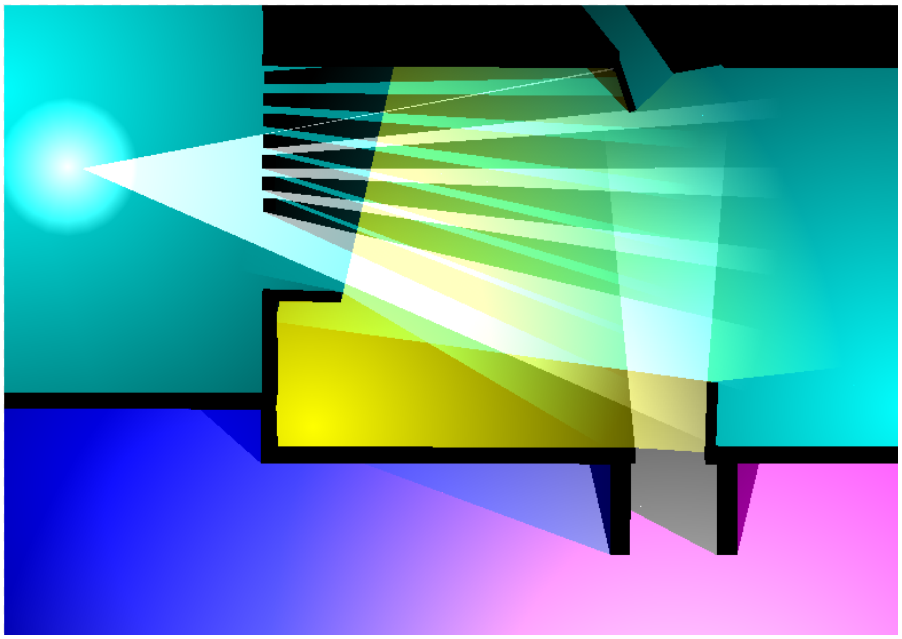
We also have a debug renderer which is a layer above the texture renderer. This is meant to render debug information like physics objects and the pathing graph. It uses a geometry set system to manage different sets of geometry (lines, triangles, etc.) and renders these efficiently by sorting them before updating its vertex buffer objects. The geometry sets makes it easy to disable/enable or remove a set of geometry.

Lighting

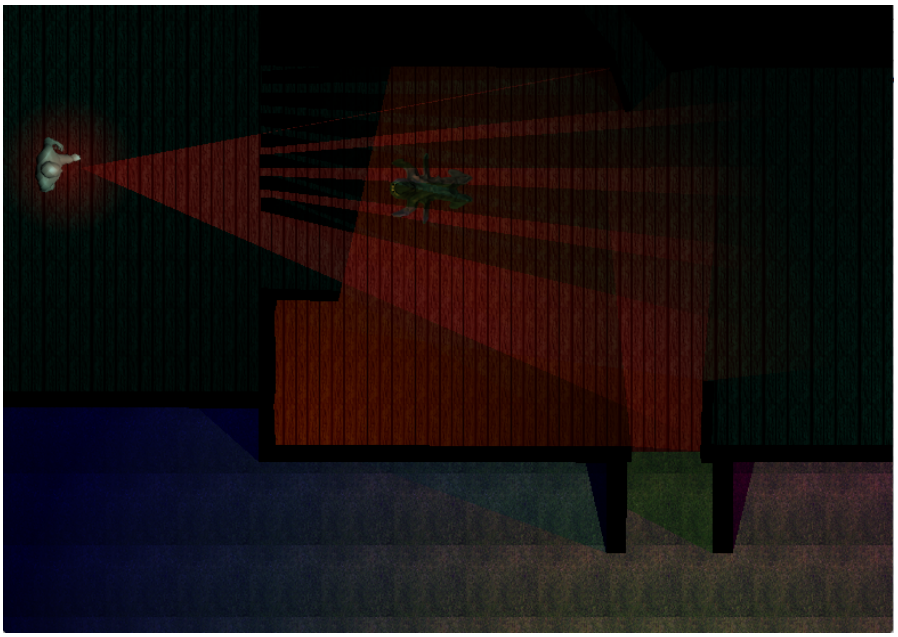
The lighting effect in the game is achieved by rendering the game in three discrete steps:
First the world is rendered normally to a screen sized texture using a Framebuffer Object (FBO.)



Then a lightmap is rendered to another screen sized texture.



Finally these are combined and presented to the screen in the normal frame buffer.



The lightmap is produced using ray casting to check where the light should and should not hit on the world. The colors produced by the lights are then blended using additive blending, this makes the light sources play naturally together when overlapping the same area. The ray casting is done by using the Box2D physics engines raycast query. This will report back when an object is hit by the ray, letting us choose what to do with it.

The ray casting algorithm is as follows:

1. First we find all the vertices of all the objects that are within range of the light.
2. We then sort them according to their angle.
3. For each object we cast a ray towards, we need to find the vertex with the smallest and the largest angle, this is set as a min or max target.
4. We then send a ray towards each of the targets, tracking how close the closest hit object is.
5. If the target is a min or a max target, we need to makes sure that we send an extra ray past the min/max of the object, this is to ensure the object casts a shadow.
6. After this is done we loop through all the endpoints of the rays and create triangles that connect with the light source, these triangles are then shaded based on how far the light traveled, this is to create a smooth shaded light.

Each light takes a lot of cpu time to calculate, so we opted for multithreaded code when designing the algorithm, each light is a self contained structure that contains it's own color, range, position etc. This makes multithreading the code very easy. We simply use thread safe queue that is filled by the main thread, after the queue is filled, the worker threads start grabbing lights and calculating the lit area.

When the lightmap and the normal texture map has been created, we combine the two by multiplying the color components of the two textures in the shader, this creates a naturally shaded world.

Logging

The logger was implemented early for the use of debugging, it has five levels of alerting, info, verbose, debug, warning and error. Each of these levels prints out a message to the console window with a prefix of what level of logging they are. There is also an option to write the log to file at the end of the session. Each logger level is a separate variadic function in the logger class, this means that it will act as printf which makes it easier to print out data useful for debugging. The logger also has different modes that determine what levels will be displayed, it is set to log all levels by default, but this can be changed to only display certain levels.

There is a global pointer to a Logger instance called g_systemLogger that can be accessed from anywhere in the program. And all warnings and debug messages are to be displayed through the logger class.

Network

There are three different modes you can play as. Server, client and offline. All three modes are within the the same application. We decided to do it this way so everyone can easily start their own server and host for friends. Selecting a connection mode is done through command line arguments. No arguments will start in offline mode, -c starts a client and -s starts a server. when creating a server you must also specify a port that will be listened to, this is the second argument. The client also needs to specify a port as second argument, this port is where to send packets to. Client must also give a third argument which is the ip address for the server. When starting a server it listens on a specified port for clients trying to connect. When a connection has been established the server creates a channel for this client and map it to client specific data. We used a self incrementing channel counter for the client channels. This was done because testing it on the same machine running a server and two clients made the server believe both clients was the same channel. One of the client specific instances is the NetworkView which acts as an interface for the client. When an event is received from a client it is passed to the clients network view for processing. As shortly explained earlier communication between server and clients are done through events. The clients sends control events to the server where it will find the clients network view and the logic will get processed. When the server is done processing, it sends back a transform event to the clients telling it the new position of actors. Both server and clients has timers to check for last packet received and will disconnect / kick if needed. When a client joins a server, the server will send over all the actor id’s it has, associated with what file to load for that actor. On the client side the actor id gets mapped from remote actor id to a local actor id. This mapping is done incase the client loads actors in a different order. So when the server sends over a request for spawning a new actor the client stores the actor id from the server as the remote actor id and creates its own local

actor id. It is important to note that we are running all the logic on the server side and the client only receives acor positions in the world.

Resource Cache

A resource gets path to an asset, then gets the data from asset. This resource is then encapsulated by a resource handle with its identifier and data. Resource handle is way for the resource cache to keep track of all of the resources. The resource handle gets created dynamically when the cache loads a resource and it allocates a buffer with the right size. This means it it can keep track of the size of the memory block, so if the cache gets full the it can be discarded and removed from the cache. A resource handle also have a pointer to the cache in case if there are multiple caches. The actual cache contains a pointer to the resource handle in two data structures. The first structure is a list of least recently used resource handles. This list is updated every time a resource is used, so that the resource that is in the back of the list is the first to be removed if the cache is full. Second structure is map containing the resource handles. There also a second list in the cache which contains resource loaders, so that the generic loaders come last in the list and the more specific loaders come first.

Audio

For our audio we are using OpenAL Soft. We chose to use ogg format to play our music. Since we are using a resource cache, we needed to have a ogg resource loader that could read the ogg from memory. After this process is done, sound process which is derived from the process implementation class is the one who plays the sound. When we want to play a sound, we must send the resource handle and the audio into it. Furthermore, can specify the volume, from 0 to 1, and if it should be looping. If we don't specify it will set the volume 1 and not looping as default. After we have sent in the information to the sound process, it then initializes the buffers with OpenAL and plays the sound.

Video Demonstration

https://www.youtube.com/watch?v=mo9Fa1EXGKU&feature=youtu.be

Group workflow

All group members for this project have worked together before on multiple projects, so there were no problems with synchronizing schedules. We had scheduled meetings every monday, wednesday and friday, this was to catch up on what every member had been working on, and what would be done for the next meeting. The meetings lasted for two hours initially, but further out into the development process we ended up working the whole day after the meetings as well.

With Magnus functioning as a project leader, it was never a problem with allocation of tasks, and we had a good overview of what had been finished and what was left to do, in addition to all bugs and problems that had to be fixed. The group was already used to using git and Bitbucket, but

this is the first time the whole group used the issue tracker extensively. This worked without any problems, and made it more easier for everyone to keep track of progress. In addition to the issue tracker, we also used the pull requests later on during development, making it easier to keep track of what needed to be merged with the master branch.

The biggest issues we had when it came to functioning as a group was some members oversleeping for the meetings. Edvin and Tomas had overslept and came late on multiple occasions. This was worst in the beginning but after a group discussion it got better, but still had some occurrences. Also the programming-skill difference between members meant that some things took longer to finish, and some tasks where better suited for certain members. Although this was a disadvantage for the group, it was still a positive experience for the members who could always ask for help from the other group members. Asbjørn and Magnus were always available when someone ran into issues or had questions. This helped all group members to work more effectively and more important, it was much easier to learn new things.

There is always room for improvements, and that applies to us as well. Although we worked well together, there are some things we could change to increase the effectivity. Structure is an important part in group work, keeping everything organized and making sure work is being done. We did have a structure when working on this project, but it was not until a couple of weeks into the project that we really started following it strictly. Next time we should be more structurized during the planning phase of the project as well. Planning large projects is crucial to keeping effectivity up, and although we did plan certain things well, we could be even more thorough next time.

Possibilities for expanding game

We have had numerous discussions on possible ways of expanding the game, and what we would have added if we had more time. As this is not nearly a complete game, we would have to have more content for the player to dig into. More levels, more variety, and puzzles and challenges. This is an essential part of any game, giving the player some entertainment value and a good immersive gaming experience.

One of the main things we would implement upon expanding the game would be puzzle challenges. Or tasks, to retrieve or find certain items in the game world and bring them to the locked door. This would force the player to explore the mansion and risk getting trapped by the monsters. This would also require some sort of inventory system for the player to carry items around. Such as quest objects and consumables. Maybe some sort of way to heal yourself, like medkits and bandages. And light-sources, such as candles that they can place around the mansion.

NOX also need some more in-game space. A larger world for the player to explore. When we started planning this project we hoped to have time to implement a second floor for the mansion, or a basement. Sadly we did not have time for this, but if we were to expand the game, then this would be a high priority task.

Adding more game space means that at some point we would run into some optimization challenges, one of them being with the pathfinding. If we were to do the project over again, we would have overhauled the pathfinding component. At the moment the A.I monsters are limited to movement through a waypoint graph overlay. So adding more space, especially large open rooms would require a large number of waypoint nodes that have to be added. While the game world is still relatively small, we can use this method, but eventually we would have to come up with a solution. This is why we should implement a navigation mesh instead of using the waypoint system. The mesh would replace all the little waypoint nodes with larger convex polygons and ultimately save time for the pathfinding algorithm as it would mean less nodes to search through. Also the mesh would create a more natural and “smooth” movement for the A.I, since it is not bound to the node-paths alone. We could safely interpolate and create smooth splines from point A to B, as long as the spline is inside mesh.

Project conclusion

Looking back on this project and it’s development-period, we can definitely conclude that this has been a vast learning experience. Everything from project planning and group work, to hands down, programming and feature implementing. This is the largest project we have worked on to this date, and it has posed challenges that we did not expect, as well as obstacles that were

expected. Most of these challenges we managed to overcome, but the main problem has been time. We feel that in the end we still don't have the game that we wanted to make, and the game we planned to make from the beginning. And the main reason for this was the lack of time. But one semester would never be enough time to implement all the features that we wanted so the goal for us for this project has always been to learn more about programming larger projects, and gaining the experience that we will need for the bachelor assignment and later on in a professional career.

B Project Nox: Development Workflow

Development workflow document for “Project Nox” – the base for Pyroeis – from “IMT3601 Game Programming” at HiG.

Development Workflow

Starting Development

1. "**git branch *branch_name***" - The branch_name should be a concise description of the task/issue, e.g. "render_crash" or "shadow_casting", and be all lower case with underscores as spaces.
2. "**git checkout *branch_name***"

In Development

This steps can (and most likely will) be repeated.

- "**git commit -m'*Description of change*'**" - The description should follow the guidelines given in this article: http://tbaggery.com/2008/04/19/a-note-about-git-commit-messages.html. Sometimes you might only need one line of descriptions and not several paragraphs, which is OK for trivial changes.
- "**git push / git pull**" - This is not necessary, but might be of use if you want to share the branch with other people or computers.

After Development

1. Developer: "**git push**" - Push your changes to the remote branch so that the reviewer can get it.
2. Reviewer: "**git fetch origin"** - This will get all the remote branches and allow the next step to be done.
3. Reviewer: If you have uncommitted changes, stash them with "**git stash**".
4. Reviewer: "**git checkout *branch_name***" - Make sure to be in the branch to review.
5. Reviewer: "**git pull**" - Pull the changes of the developer to review them.
6. Reviewer: Review the changes, and if they are NOT approved, return to "In Development". How you review is up to you, but the changes should use our coding guidelines, not ruin the architecture, and fix the issue. Some things you should do: Build the application, run it, and test the changes made. See that the changes in code is OK by e.g. using "**git diff master**" (these changes can also be viewed on bitbucket, by going to the commit messages).

After Code Approval

This is done by the reviewer so that the merge will be logged with the reviewer's name.

1. "**git checkout master**" - Be sure to be in the master branch.
2. "**git merge --no-ff -e *branch_name***" - Merge the changes into master with no fast forward. This will bring up an editor (most often Vim, use google for how to use) if there are not conflicts. Important: See "Merge Message" below for how to write the message.
3. If there are conflicts: fix the conflicts in the files. Use "**git add *file_name***" on all files that had conflicts and then use "**git commit -e**". This will bring up an editor (most often Vim, use google for how to use). See "Merge Message" below for how to write the message.
4. "**git push**" - Push the newly merged changes to upstream master!

5. If you stashed changes earlier, checkout your working branch and “**git stash apply**” to get your changes back.

Merge Message

Add “fixes #n”, where n=issue-number, to the end of the default message. Add a paragraph under the default message describing the changes this merge makes (e.g. what feature implemented, what bug fixed, api changes), not what you experienced or your opinions.

Example:
Merge branch 'clean-interactive'

Finishing touches to update the document to adjust to a new option "git clean" learned recently.

Fixes #123

See the git log of git for other examples of messages:
http://git.kernel.org/cgit/git/git.git/log/?ofs=50

Pro Tips

- Use “**git status**” as much as possible to see the current status of the files before doing changes in git (like commit, merge, or push/pull).
- Remember to do “**git push -u origin *branch_name***” the first time you push so that the branch will track the newly created remote (upstream) branch.

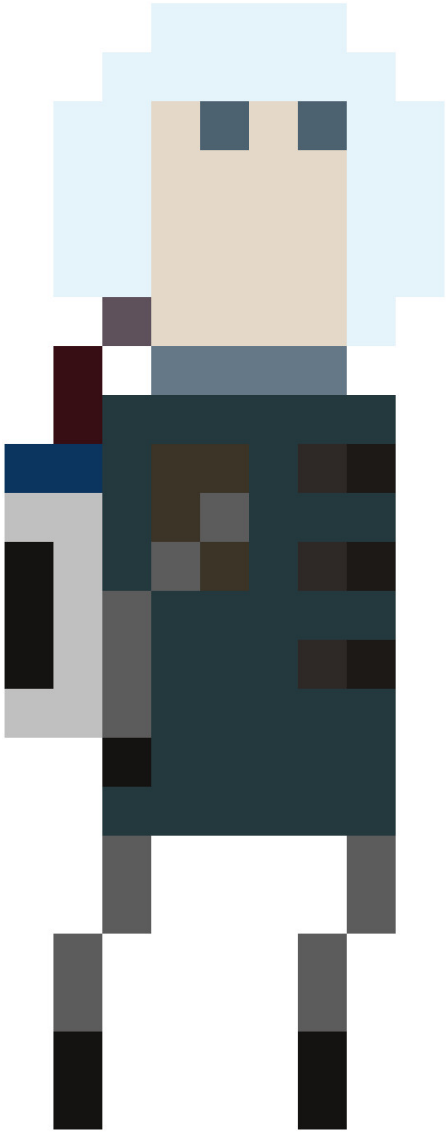
C Pyroeis: Project Plan

Project plan for Pyroeis. Minor changes were made during the project period. Some of the goals of some milestones were shifted or removed.

Magnus Bjerke Vik
Asbjørn Sporaland
28/02 2014

Pyroeis

Project Plan



Magnus Bjerke Vik
Asbjørn Sporaland
28/02 2014

Table of Contents

Goal

The goal of the project is to create a game which has infinite explorable caves procedurally generated with each new game.

This goal is broken down into several sub goals of the product:

- A two-dimensional view from the side into the caves of Mars.
- A playable character who can both walk, jump, and use a jetpack.
- Provide huge procedurally generated cave systems to explore.
- Provide varying and interesting biomes with wilderness inside the caves.
- An environment that can be destructed by explosives and other means allowing the player to explore the caves further.
- Enemies in the form of alien life will attack the player on sight.
- Collectable resources around the caves.
- Upgradable gear by using resources collected.
- Realistic physics simulation letting the player interact with the world.
- Dynamic lighting were objects in the world cast shadows.

Delimitation

The game would ideally have a lot of features and content to make the game rich and entertaining so the user would want to play it and keep playing it. With this project's limited time some features that were proposed for the design were removed. While not all features described in the GDD (Game Design Document) will be in the final product, the features specified in the milestones further below will be. Though the engine supports it, the game will not officially support networked play.

Task Description

The project task is to implement a game that fulfills the goals described above and creates an enjoyable experience for the player.

As well as creating a game, the task is about extending the functionality of the already existing code base for Project Nox created in the game programming course (IMT3601). This project will maintain the same quality of the code, and keep it usable and maintainable for later projects.

World generation and destructibility is one of the most important goals. The game will generate connected and disconnected caves that mimic properties of real caves, though still look alien and different. Different parts of the world have to be procedurally generated completely independent from other parts and still match the parts surrounding it.

The player will experience different types of biomes inside the caves depending on where under Mars' surface they are. Different biomes will have different types of content like snow, plants, trees, and animals.

The world is completely destructible and the player have to take advantage of this by e.g. exploding the caves to get to inaccessible parts of the world.

To survive in the environment, the player will have to use different kinds of weapons. These weapons are upgradable, and the player can collect resources around the world to upgrade the weapons with different abilities.

Using either the jetpack, or running and jumping, the player can get around inside the caves.

The game will be viewed from the side and the world will be rendered from this side perspective as if the planet is cut in half and the camera looks straight at the inside of the planet. Realistic looking dynamic lighting will light up the caves and shadows will be cast from objects.

The world will be simulated using realistic physics simulation from the same two-dimensional perspective as the camera has.

The game features and other design decisions is described in-depth in the GDD (Game Design Document)..

Roles and Responsibilities

The project consists of two members: Magnus Bjerke Vik and Asbjørn Sporaland. Both members will do programming and game design.

Magnus also acts as the project manager and has the responsibility of making sure the development is following the plan, that the project will be done within the time requirements, and that the plan is updated throughout the project.

Production assets (sound and art) will be created by an external person. This person has no responsibilities and may leave the project at any time. If they do, the product will use placeholder assets found or created elsewhere.

Rules for the group is described in the separate document “Group Rules”.

Development Process

The project has a set of defined functionality that needs to be implemented, but meeting the requirement of creating an enjoyable experience requires flexibility and the ability to change the design often. We will use scrum as our framework because of this. It lets us focus on a set of features for each iteration, resulting in a working and testable increment at the end of the process. Even though we are locked to the sprint backlog for a sprint, we can test the game experience after each increment and revise the design for the next increment. This gives us both the flexibility and structure that we need, letting the game design evolve through the process while still making a working product.

Atlassian’s set of tools, including Bitbucket and Jira Agile, will be used to managed the scrum process and issues.

Since the project has a time span of only four months, and the product developed is a game which needs a lot of flexibility in the product backlog a sprint period of 7 days will be used.

Each work day a daily scrum will be held at 08:15 to discuss what we have done, what we will do, what obstacles we have met, and revise the sprint backlog.

Each friday the sprint review meeting, sprint retrospective, and sprint planning meeting will be held. The sprint review meeting will be time-boxed to one hour. The retrospective will be time-boxed to half an hour. While the planning meeting will be time-boxed to two hours. This row of meetings will be held at 12:30 so that it will end at 16:00.

A work log for each day will be written at the end of the work day. If work is done after the working hours the log will be updated with what work was done.

The game will be developed on Linux and OS X, but not on Windows, and will therefore have to be tested on Windows regularly since this is an important platform. To accomplish this the program will be tested on windows with each sprint review. If something is broken, it will be added to the product backlog as a bug or enhancement.

Documentation and Quality Assurance

The C++ code will be documented using the Javadoc flavour of the Doxygen syntax, and html-documentation will be generated using Doxygen. Classes and function prototypes must always be documented with this standard. One exception is when the class name or function prototype is self-explanatory (e.g. a getter or setter). Member variables or types must also be document if they are not self-explanatory. In general: *self-explanatory ? ignore : document*.
Comments inline in code should be used when the code is not easily readable or too complex, though this should generally be avoided. Always write readable code.

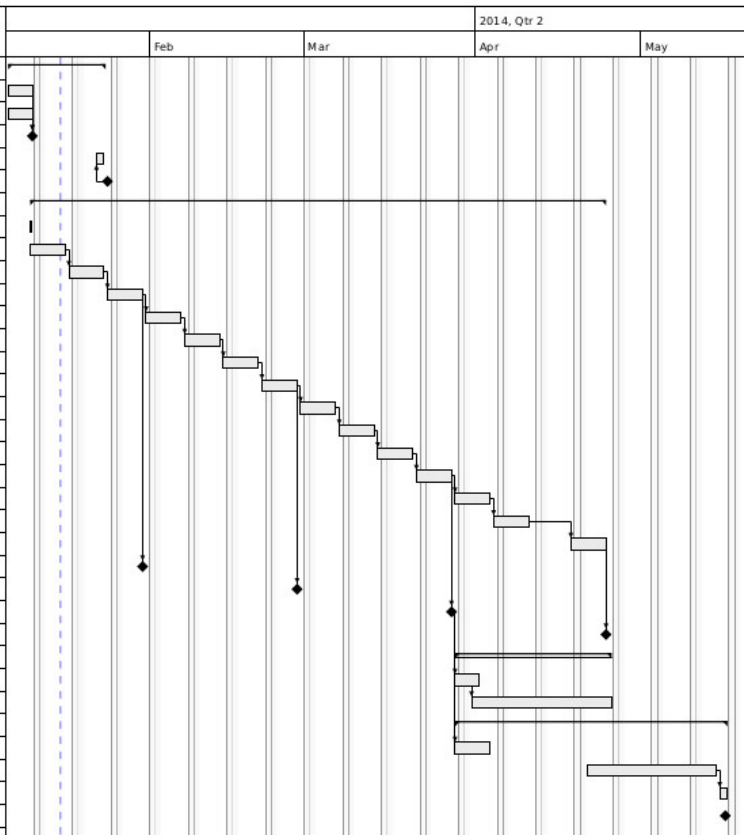
Programmers will have to follow the coding convention described in the provided document named “Coding Convention”.

Code will be versioned and stored in git on a Bitbucket server. When a programmer is implementing a task he will do this in a separate branch since the master branch always should be in a workable condition with complete features, and each task can be more independent of each other. Small fixes can be pushed directly to the master branch.

The implementation of a task must not break the program for the other developer, and the code must have proper quality in both readability, performance and stability. To accomplish this, the other developer will review and possibly test the code before either accepting and merging it, or rejecting it telling the developer what must be changed.

Development Schedule

Name	Work
Planning	**12d**
Initial project planning	5d
Initial game design	5d
Initial planning done	
Plan review	2d
Plan delivery	
Development	**70d 4h**
Product backlog planning	4h
Sprint 1	5d
Sprint 2	5d
Sprint 3	5d
Sprint 4	5d
Sprint 5	5d
Sprint 6	5d
Sprint 7	5d
Sprint 8	5d
Sprint 9	5d
Sprint 10	5d
Sprint 11	5d
Sprint 12	5d
Sprint 13	5d
Sprint 14	5d
Dev Milestone 1 (planning, prototype)	
Dev Milestone 2 (gameplay demo)	
Dev Milestone 3 (playable and testable)	
Dev Milestone 4 (complete and bug-free)	
Testing	**23d**
Identifying bugs, balancing-tweaks, and performance issues	3d
Fixing issues	20d
Documentation	**25d**
System documentation review	5d
Writing report	18d
Reviewing report	2d
Hand in report	



The development consists of 14 one-week sprints. With each sprint planning meeting, the current product backlog will be refined and populated with tasks required to reach the current milestone.

The development period is split into four phases with one milestone ending each. Each milestone is a larger checkpoint, in addition to the sprint reviews, to analyze if the product can make the deadline. Though the milestones are specific in what features to be completed, they are not fixed. The design might change throughout the development, which will cause the milestone goals to change. Some features will also be miss-calculated in size, causing us to scrap or add features.

Features listed here will be described in depth in the GDD.

Milestone 1 (prototype)

- Complete project plan.
- Complete GDD.
- Complete non-functional requirements specification.
- Core graphical user interface implementation.
- Engine refactoring.
- Simple (10-25%) gameplay complete:
    - **World generation prototype**: Basic world generation and destruction will be fully implemented. This includes a world that is playable, both in terms of performance and movability, and infinite in size. The player will be able to access the caves from the surface of Mars.

- **Simple movement**: Players will be able to move around the world by running, jumping and using the jetpack.
- **Destructible world**: The player can destruct the world with explosions.

The theme of this phase is to plan the game and make sure that the code base is ready for this project.

The code that this project is based upon was developed to be used in future projects and is therefore mostly ready to use in this project. Still some refactoring in the engine might have to be done since the game is going from a top-down view to a side-view.

The GUI system from the previous project was not fully implemented, so this have to be completed by this milestone to be ready for menus in the upcoming milestones.

With this milestone some of the gameplay will also be completed and will be prototype for the very basic gameplay showing the world generation/destruction and player movement. This is used to analyze if our game design as it is now is going to work at all and that we will have time to complete it.

Milestone 2 (gameplay demo)

- Basic (70-75%) gameplay complete:
  - **Shoot**: Players will be able to use weapons to damage the enemies he encounters in the world. This milestone will feature the all types of weapons. All [placeholder] assets for the current elements will be present; sound, effects, textures.
  - **AI enemies**: Enemies will spawn at random throughout the world, attacking players on sight. For this milestone we will focus on the flying enemies. Both the ranged and the melee enemies will be implemented and attack players on sight. Enemies will be killable and be able to kill the player presenting a danger towards the player as he travels the world.
  - **Resource pickup**: Dust will be present in the world, allowing the player to gather these. This stage will only include the red and yellow dust.
  - **GUI menus**: Main menu / pause menu with continue, exit, new game working.
  - **Inventory**: Players will have an in-game inventory at their disposal allowing them to see what items they have in their possession, and create gems from dust.
  - **Lights**: Players will be able to create and place lights in the world. Lights that are available are the flare that you can toss, and the permanent wall light.
  - **Game goal**: The game will feature a tangible goal for the players to reach, reaching this goal will allow the players to “win”.
  - **Scoring**: The scoring system will be implemented where players can see their score compared to other players’ score. Scores will be pushed to a remote database.
- Identify needed assets. (List for each actor/event in the game).

This milestone will have implemented most of the gameplay, and the product will be a playable gameplay demo showing the core mechanics, style and how the flow of the game will feel with menus.

Assets needed will be identified and listed in a document. Placeholder assets will be collected or created and put into the game.

Milestone 3 (Playable and testable game)

- Advanced (90-95%) gameplay complete:
    - **Player knockout**: Players need to be even more careful when traveling around the world. Crashing with the jetpack, getting hit by enemies or falling will have the players at the mercy of physics, trying to gain control of their character. Falling or crashing at high speeds will also damage the player.
    - **Weapon upgrading**: The player can upgrade and configure all their weapons' properties.
    - **Advanced weapon functionality**: All the advanced features of the weapons will be implemented in the game. Players will be able to zoom with the sniper.
    - **Sonar**: Sonar functionality will be fully implemented.
    - **AI enemies**: Ground based melee and ranged enemies will be implemented and spawned in the world.
    - **Biomes**: Biome placement system will be functioning, creating different cave formations with different items. Jungle and desert caves will be implemented.
    - **Plants**: Different plants will spawn throughout the world.
    - **Green Dust:** Green dust can be found.
    - **Tutorial:** The player will be introduced to the game by a tutorial.
- Playable Windows version, i.e. smooth framerate and no serious crashes on Windows..
- Windows installer letting users easily try the game.
- OS X app letting user easily try the game.
- User testing system up and running.
    - Collecting statistics about how the game is played.
    - Users can type a message wherever in the game.
    - Messages and game data regarding that message stored in database.
    - Simple web interface for reading the messages and their data.
- Distributed the game to users for testing, and receiving feedback.

This milestone will focus on finishing the advanced gameplay features, actually making the game interesting, and having a version ready for testing where users can give feedback.
In this state the game will be feature complete and ready for tweaking and performance optimization.

We will also consider adding other features not planned for any milestone, but described in the GDD, to this milestone, if there is enough available time.

After this milestone the game will have a feature freeze meaning that the game features must all be implemented by now.

Milestone 4 (Complete and bug-free game)

- Development done.
- Tweak game balancing against user experience.

- Crash-free game.
- Adjust to meet requirements. (Optimizing, documenting, etc.)

With this milestone the game will be complete and polished, ready for a user to install and run reliably. It will mainly consist of identifying bugs and performance issues, and fixing these issues.

Documentation

After milestone three the code documentation (doxygen), and system documentation will be reviewed and fixed to match the requirements.

The project report will be written over the whole project period, but focus will shift over to it during the last sprints. After the last sprint, three weeks will be dedicated to writing the report and documentation. A review will be done before the hand-in to make sure that all is included and that the quality is good.

Risk Analysis

Agile Development

We are using agile development in the form of scrum. This severely helps us in splitting up the issues into smaller and more manageable tasks. For each week of development we will easily see what tasks are causing issues as we develop them. This allows us to shift the focus towards the areas that are causing the problems and solve them as fast as possible.

Identified Risks

- Poor engine performance on different platforms**. (Probability: M.)**
- Gameplay not being equal on all platforms. **(Probability: L.)**
- Not getting the engine to work properly on windows. **(Probability: L.)**
- Creating setup/installation program for windows. **(Probability: L.)**
- Mac OS X installation that will not work for multiple systems. **(Probability: M.)**
- Mac users being unable to run due to c++ 11 support. **(Probability: H.)**
- Bad performance. **(Probability: M.)**
- Legal issues regarding libraries. (not being able to statically compile etc.) **(Probability: L.)**
- Not able to distribute on linux. **(Probability: L.)**
- Jira server crashing. **(Probability: L.)**
- Bitbucket crashing. **(Probability: L.)**
- Losing work/source code. **(Probability: L.)**

Consequences

- Poor engine performance on different platforms. **(Impact: M)**
    - More time needed to polish and optimize for the different platforms.
    - Limiting supported hardware, If the performance is too low on certain operating systems we might need to set higher system requirements for that particular system.
    - This will affect the users of the application severely. If the game does not perform well for the users, they will probably not enjoy it, and quit playing.
- Gameplay not being equal on all platforms. **(Impact: L)**
    - Randomly generate worlds not functioning equally on all platforms
    - Imbalanced highscore-lists for cross platform gameplay.
    - Unable to have cross-platform competitions, challenges from friends.
- Not getting the engine to work properly on Windows. **(Impact: M)**
    - More time needs to be spent configuring the engine for windows.
    - Needing to scrap windows support.
    - Game performing worse on windows.
- Difficulty creating setup/installation program for windows. **(Impact: M)**
    - Inability to get testers for windows.
    - Unable to release on windows.
- Ability to create Mac OS X installation that will work for multiple systems. **(Impact: M)**
    - Needing to build specific versions for each OS X version.
    - Unable to release for earlier versions of OS X.
- Mac users being unable to run due to c++ 11 support. **(Impact: L)**

    - Unable to support certain OS X versions
    - Needing to rebuild engine to support earlier OS X versions.
- Bad performance. **(Impact: M)**
    - More optimization needed.
    - Need to reduce the scale of the world.
    - Game scope being reduced
    - Gameplay becoming less interesting.
    - Need to reduce graphic fidelity
- Legal issues regarding libraries, not being able to statically compile etc. **(Impact: M)**
    - Need to drop functionality/remove libraries
    - Lots of work needing to be done when releasing final version.
- Not able to distribute on linux. **(Impact: M)**
    - Inability to get testers for the linux platforms.
- Jira server crashing. **(Impact: L)**
    - Inability to update the backlog while problem persists
- Bitbucket crashing. **(Impact: M)**
    - Inability to push code to the server, hampering cooperation.
- Losing work/source code. **(Impact: H)**
    - Needing to redo work that has previously been done.
    - Project Failure.

Risk/Probability	L	M	H
L	Very Low	Low	Medium
M	Low	Medium	High
H	Medium	High	Very High

(matrix used for calculating risk)

Risks by Rank

- Bad performance
    - Risk: Medium
    - This is probably the most pressing concern when developing the application. Pyroeis consists of numerous complex parts that all require a big chunk of the cpu time. the raycasting used for the lighting, Box2D's physics, the generation of the world and all the other subsystems will need to perform very well to be able to achieve the target frame rate of 60 frames per second. We will need to spend a considerable amount of time to make sure to utilize multiple cores of the processor whenever we are able to do this safely. We will also make sure that the users are able to configure the game's graphics and physics to make sure they can find a good mix of performance and effects.
- Poor engine performance on different platforms.
    - Risk: Medium
    - Since the game is a multiplatform game, we will need to make sure that the performance is good across the board. We are developing the game mainly on the

Linux and Mac OS X platforms on a day to day basis. Each week we will test the performance on Windows to make sure it meets the performance requirements we need. Towards the end of the project we will also shift our focus over to the windows platform.

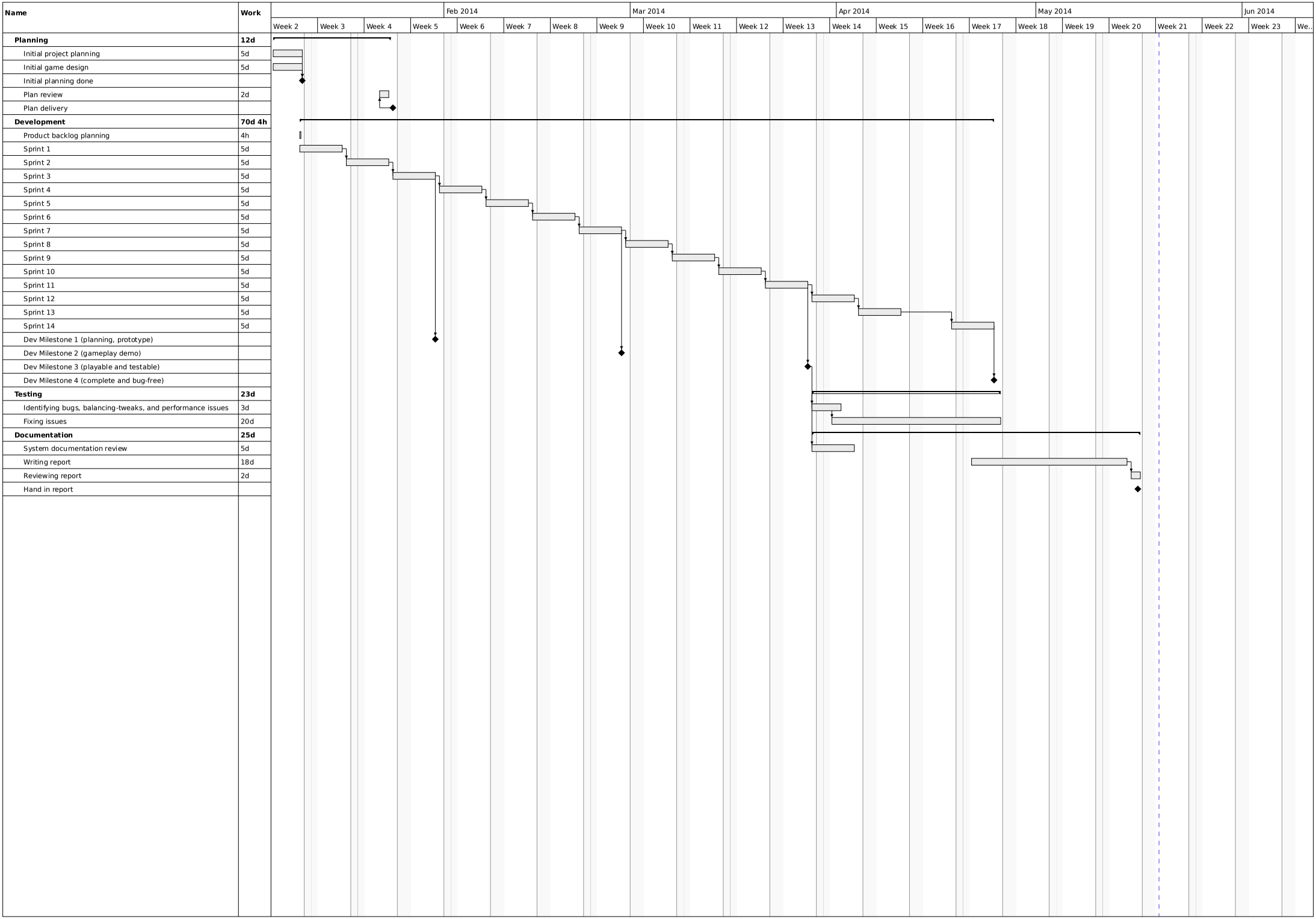
- Losing work/source code.
    - Risk: Medium
    - To make sure that this scenario does not happen the source code is backed up rigorously at multiple locations; locally on developers machines, offline servers, bitbucket. This ensures that in the off-chance that we were to lose some data, it would be isolated to a minimum amount.
- Unable to create Mac OS X installation that will work for multiple systems.
    - Risk: Medium
    - We are in fact able to create separate versions that will run for the operating system they are built on. The problem is that we are not able to build a one version that will be able to run on all the different operating systems at the moment.
- Mac users being unable to run due to c++ 11 support.
    - Risk: Medium
    - This risk is already a fact that we know we need to face. Users of OS X versions before 10.7 will not be able to run the program due to the fact that c++ 11 is not supported further back than 10.7. We will do nothing to alleviate this risk as it is more important to focus on the engine being forward compatible than having it support old platforms. The OS X user base is upgrading fast towards the newest release; Mavericks 10.9. Back in early December 2013 the 10.7 and above user base was already at around 75%
- Legal issues regarding libraries. (not being able to statically compile etc.)
    - Risk: Low
    - We try our best to use software libraries that are very flexible and allow us to use them the way we see fit. All software that will be used to will have the legal status of the checked to make sure they fit our goals.

Total Risk

3.8 / 12 = 0.3116666667 = Low

D Pyroeis: Gantt Chart

Gant chart for Pyroeis. Also part of project plan in Appendic C.


Name
Work
Feb 2014
Mar 2014
Apr 2014
May 2014
Jun 2014
Week 2
Week 3
Week 4
Week 5
Week 6
Week 7
Week 8
Week 9
Week 10
Week 11
Week 12
Week 13
Week 14
Week 15
Week 16
Week 17
Week 18
Week 19
Week 20
Week 21
Week 22
Week 23
We..
Planning 12d
Initial project planning 5d
Initial game design 5d
Initial planning done
Plan review 2d
Plan delivery
Development 70d 4h
Product backlog planning 4h
Sprint 1 5d
Sprint 2 5d
Sprint 3 5d
Sprint 4 5d
Sprint 5 5d
Sprint 6 5d
Sprint 7 5d
Sprint 8 5d
Sprint 9 5d
Sprint 10 5d
Sprint 11 5d
Sprint 12 5d
Sprint 13 5d
Sprint 14 5d
Dev Milestone 1 (planning, prototype)
Dev Milestone 2 (gameplay demo)
Dev Milestone 3 (playable and testable)
Dev Milestone 4 (complete and bug-free)
Testing 23d
Identifying bugs, balancing-tweaks, and performance issues 3d
Fixing issues 20d
Documentation 25d
System documentation review 5d
Writing report 18d
Reviewing report 2d
Hand in report

E Pyroeis: Game Design Document

Game Design Document (GDD) for Pyroeis. Minor changes were made during the project period, but most of it is exactly like the original.

Magnus Bjerke Vik
Asbjørn Sporaland
02/04 2014

Pyroeis

Game Design Document

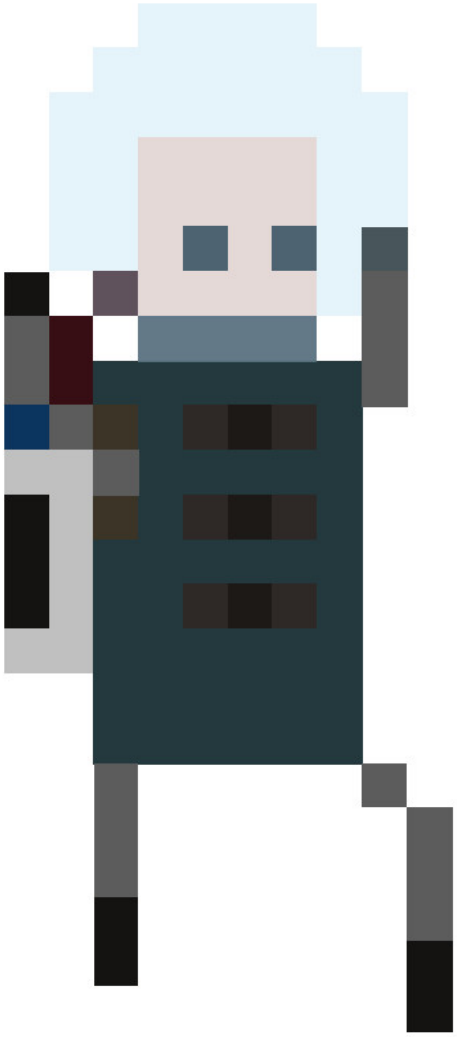


Table of Contents

Introduction

Pyroeis is a 2D action exploration game for PC that uses a custom built engine to send the player to Mars as an astronaut, where they will explore the mysterious caves and creatures below the planet's surface.

Story Background

You wake up in a dark cave, alone, with only your flashlight. You can't remember where you are, or what you are doing there. The only direction you can go is up.

After a while, you reach the surface where the ground is covered by red sand and stone. Not far from the cave entrance stands a landing pod. You read the log file on this pod and discover that you currently are on the planet Mars, and your mission is to explore the caves where recently a rover disappeared after receiving a mysterious signal from far below the surface.

Description

To complete your mission, you will have to descend below Mars' surface. A jetpack on your back lets you more easily get down or up steep hills, but it is a difficult task to master controlling it.

Some parts of the caverns might be blocked and will force you to either go another path or throw bombs that stick on walls and explode when you trigger them remotely. When the bombs you triggered explodes, an area around where the bomb exploded gets destructed and you can go there. But be careful, parts of the cave will fall apart and be thrusted by the explosion through the air, and they might hit you hard.

As you descend further down the caverns, the terrain changes and more wildlife appears. Alien, though somewhat earth-looking, plants grow, and dangerous animals attack you. If you go further, you might discover a completely different type of wildlife.

To defend yourself against the animals, you are equipped with four different guns with different properties. You can shoot animals or plants to kill or scare them away.

You will find special Martian dust with varying different colors if you kill animals and plants, destroy cave walls, or ice. You should always pick these up as you will need them to survive the harsh environment.

Dust collected can be merged into a Martian gem of the same color if you have enough dust. You use these to upgrade the different properties of your weapons. Some properties might be more important than other, so you can choose where to delegate the power. You also use dust to fuel your gear, such as the jetpack. So if you don't use your gear much, you will be able to upgrade your weapons much more.

The caves are dark, so you will have to light it up. Your flashlight helps a little, but not much. To really light the caves up, you use dust to create light sources that can either be placed on walls, or be thrown as a flare.

A sonar device can be used to help you find thin walls that might lead to another cave if you destroy them.

You will descend further and further down until you reach the core where the mysterious signal came from.

Key Features

- **Different world each time**: Don't get tired of the same world each time. Experience new and exciting worlds.
- **Destructible environment**: Destroy the caves wherever you want to get access to the stuff you want.
- **No world limits**: Travel as far as you want.
- **Varying wildlife**: Don't get bored over the same plants and animals.
- **Upgradable and configurable weapons**: Choose how you want to play.
- **Dynamic lighting**: Be challenged by dark caves and use lights strategically to your advantage. Objects in the world will cast shadow so enemies or other objects might be hidden from you.
- **Realistic physics**: Physics of the real world will challenge you when flying with jetpack or when debris is created from explosions.

Style and Feel

The game is set in the not so distant future in our real world on the planet mars. The human technology has advanced from our current time, but the look of space equipment is similar to what we use today.

Mars on the surface has the same looks as it has in the real world, but below the surface there is plenty of nature and wild life. Trees, flora, water. Different biomes provide different looks. Some might be rich on nature jungle-like, while others might be more dry like a desert.

The art will feature simple looking pixel graphics like the astronaut in the image.

Pyroeis will feature musical entries that match it's atmosphere . The music and sound will adapt to the environment the players find themselves in, ranging from the lushness of jungles to the gritty and harsh environments of a sub-surface dry ice-cave. Explosions echoing off walls, and strange creatures screaming in the dark will ensure a dark and bleak atmosphere.

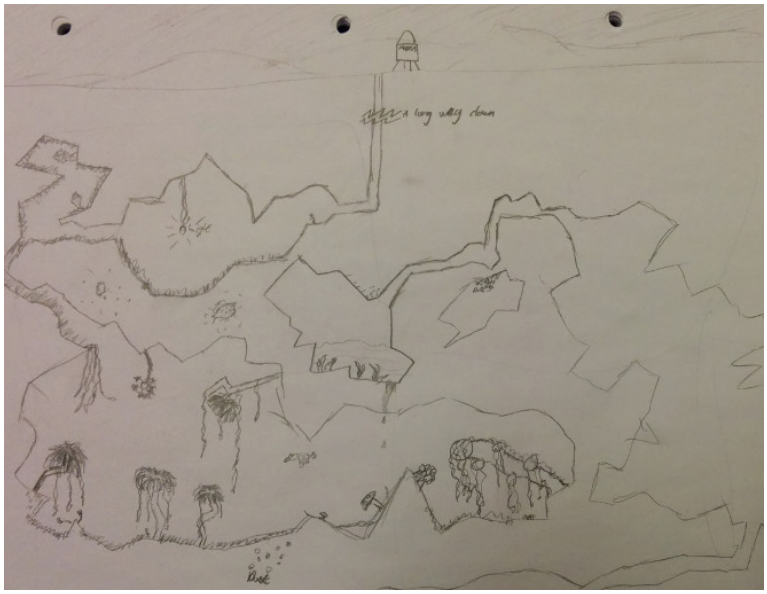
Gameplay

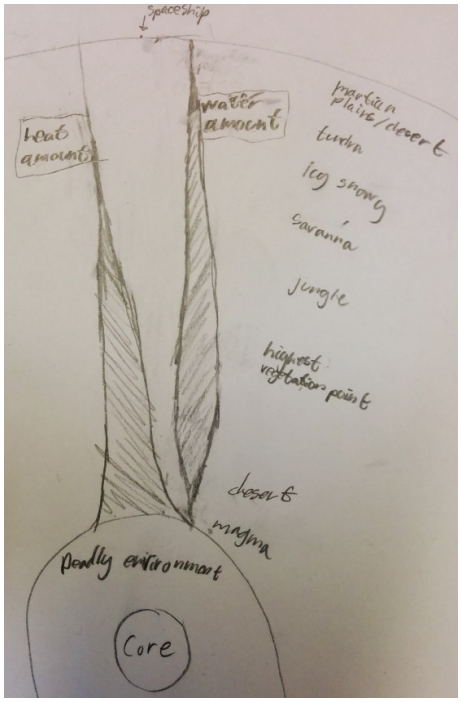
Mechanics

Core gameplay mechanics will focus on the player traversing the Martian environment and survive. The player will have the ability to move, jump and use a range of tools/weapons to help gather resources. It is a singleplayer game that will focus on exploration of the world and surviving on your own. Physics will affect the player as he travels the world, knocking him over and sending him flying if he is struck by heavy objects.

Cave System

The cave system below mars' surface will be procedurally generated and contain different biomes at different places (jungle, desert, etc.)





The cave system will have a finite top point which is the surface, and a finite bottom point which is the core of the planet. Left and right go to infinity.

The biome types will be based on the combination of humidity and heat. Close to the surface the caves are dry and cold. As you descend further down the landscape will get warmer and more humid. As there appears more water it will transform to tundra, ice and snow. Then the heat will increase with the water, and it will transform to savannas and jungle. Then the heat still increases, but the humidity decreases and it will transform to deserts and then finally to magma.

Destructible Environment

The environments of mars are destructible. Players will be able to forge their own paths through the caves, exploring new areas while making their way towards the goal. Destroying the environments will provide the player with different strategies when it comes to protecting himself and attacking the monsters. Materials that have been destroyed may also yield different resources that the player can use to further his goals.

Martian Dust and Gems

The Martian dust can be found in different places in the cave system. Collecting a certain amount of the dust will allow the player to create one Martian gem. These gems will allow the player to upgrade a specific property of a weapon. Some dust can also be used to power other items.

- Red dust can be found within the Martian earth. You will have to destroy the earth to get it.
- Green dust can be found within plants.
- Yellow dust will be dropped by animals when dead.
- Blue dust can be found by destroying ice.

Enemies

Flying

Melee

The flying melee enemy will flee at the first sight of the player. If the player manages to kill the enemy before it finds a safe place to hide, all is well. If the enemy manages to flee and find a hiding spot, she will start giving birth to her dangerous spawns. These small creatures will fly at you and damage you upon impact.

Ranged

Ranged flying enemies will be roaming around the caverns. If they see the player, they will immediately try to kill them by spitting poison. The ranged enemies will also move around to make it harder for the player to hit them when attacking.

Ground

Melee

Melee enemies on ground will walk around and jump. If they see the player, they will run after and strike them.

Ranged

Ranged ground enemies will always try to stay at a safe distance and shoot small exploding bombs at you.

Weapons

Players will be given the opportunity to wield a variety of different weapons that are useful for different scenarios. Ranged weapons will feature different projectiles with different speeds and effects on the targets.

Weapons have a power level that determine the total power available on that weapon. How to delegate the power to the different parts of the weapon is up to the player. There is a limit to what the power level of the different subsystems are. These may be upgraded using gems. The player may set up different settings and switch between these during gameplay using hotkeys.

By combining one gem of each color, the player will be able to add one more power to a weapon.

Handgun

The handgun is the simples weapon in the game, It fires one bullet at a time, semiautomatic. The fire rate of the handgun is only limited by how fast the player is able to pull the trigger. The handgun has the following parameters that can be tweaked by the player:

- *Damage (Red Gem)*
  The damage of the weapon determines how lethal the weapon is when it hits its target.
- *Recoil Reduction (Green Gem)*
  This increases the rate of fire that the gun is able to handle while still firing accurately.
- *Magazine Size (Blue Gem)*
  The number of bullets that the gun is able to fire before reloading.
- *Projectile Velocity (Yellow Gem)*
  Determines the speed of the projectile. Since all projectiles are affected by gravity this will also make the weapon fire straighter, lessening the need to calculate bullet drop.

Shotgun

The shotgun fires a massive number of bullets each time the trigger is pulled, the shotgun is semiautomatic. The total damage of the shotgun is divided by the number of bullets that are fired. so there is a tradeoff between having lots of bullets and high damage. The shotgun also has a spread that can be adjusted, as well as vary by the number of bullets fired per shot. The shotgun has the following parameters:

- *Damage (Red Gem)*
  The damage of the shotgun, together with the bullet this determines the damage inflicted by the bullets.
- *Spread (Green Gem)*
  The player is able to decrease the spread of the shots that are released by the weapon, this will allow the weapon to focus the damage more in one direction.
- *Bullet Count (Blue Gem)*
  The number of bullets that are fired by the weapon, the damage for each bullet is the damage of the shotgun divided by this number.

- *Fire Rate (Yellow Gem)*
  Determines the rate that the weapon is able to fire at.

Sniper Rifle

The sniper rifle will allow the player to engage targets at a greater distance than the other weapons. It will provide the player with the zoom ability, effectively moving the camera in the direction that the player is looking. It is heavier than the other weapons in the game, thus aiming will be less responsive.

- *Damage (Red Gem)*
  The damage of the weapon determines how lethal the weapon is when it hits its target.
- *Target Acquisition (Green Gem)*
  Increasing the target acquisition parameter will make the player more proficient in aiming with the sniper rifle, decreasing the time it takes before the player is able to fire accurately.
- *Zoom (Blue Gem)*
  The zoom determines how far the player is able to pan the camera when using the sniper rifle, further increasing this value will allow the player to acquire targets that are even further away. When the zoom ability is fully upgraded, the player will be given a trajectory of where the bullet will travel, this will allow him to make the adjustments that are necessary to hit the target.
- *Projectile Velocity (Yellow Gem)*
  Determines the speed of the projectile. Since all projectiles are affected by gravity this will also make the weapon fire straighter, lessening the need to calculate bullet drop.

Machine Gun

A fully automatic weapon that fires a large amount of bullets in a short amount of time, the weapon is more accurate when it is fired in short bursts.

- *Damage (Red Gem)*
  The damage of the weapon determines how lethal the weapon is when it hits its target.
- *Accuracy (Green Gem)*
  Determines how accurate the weapon is when fired consecutively, increasing this will allow the player to fire the weapon with more efficiency.
- *Magazine Size (Blue Gem)*
  The number of bullets that the gun is able to fire before reloading.
- *Fire Rate (Yellow Gem)*
  Determines the rate that the weapon is able to fire at.

Bomb

The player will have a bomb thrower that will throw bombs that stick on the walls and can be detonated remotely by a trigger. The bomb thrower requires red dust to create new bombs. It can also be refueled at the base.

Light Sources

The game will allow the player to use different items that will dynamically illuminate the world. This will allow the player to get a clearer overview of the world, allowing him/her to have a greater deal of control on his surroundings, lessening the threat from monsters and dangers such as falling.

Flashlight

This is available at all times, and is fastened to his shoulders. It will light up a cone where he is looking.

Wall Light

Will Illuminate around itself forever and sticks to the background wall. This light must be created with yellow dust.

Flare

Can be thrown and will burn for a small period of time, illuminating a large area. This light must be created with yellow dust.

Sonar

The sonar will allow the player to “scan” the area around the player, letting them discover hidden caves. This works by illuminating the world in a torus/donut around the player. This effect has a certain distance that it travels revealing the caves as it passes through them. The player will have to use green dust to refuel the sonar, but it can also be refueled at the base.

Jetpack

The jetpack will allow the player to more easily traverse the terrain of mars, but is difficult to control. The physics of the jetpack will be like a chopper where you control the upward force and the angle. If the player crashes while flying at a high speed, he will be damaged and loose control over the jetpack, resulting in the player being thrown through the air.

Implosion Device

The implosion device is a small device that can be thrown. After a few seconds it will apply a force to the objects around it, pulling them in towards its center. It lasts for a couple of seconds, until an explosive force is applied to the objects surrounding it.

Proximity Mine

Proximity mines will float mid air and Detonate when an enemy get close enough. The mine will spart Spawning fragments in a circular fashion. Credit: Nordling.

Goal

The player will be trying to explore the Martian subsurface environment to get to the goal, which is a valuable item near the core of the planet. Returning this item to the surface will be the ultimate goal in the game. If the player achieves this goal he is free to continue the exploration on Mars however long he wants to. The game concludes and a score is calculated based on the time that was used to collect the item. Restarting the game after this will allow the player to start a the game in the new

game+ mode. In this mode, the player to continue with the gear he previously acquired and do the mission once more, with harder enemies.

Progression

As the player descends deeper, the enemies will increase in numbers and difficulty. At certain depth levels, such as each 1000 meters, a boss that they have to survive will appear. If they manage to survive the fight, the place were they defeated the boss will be their new spawn point. After a certain number of such instances, they will reach the goal and complete the game.

User Experience

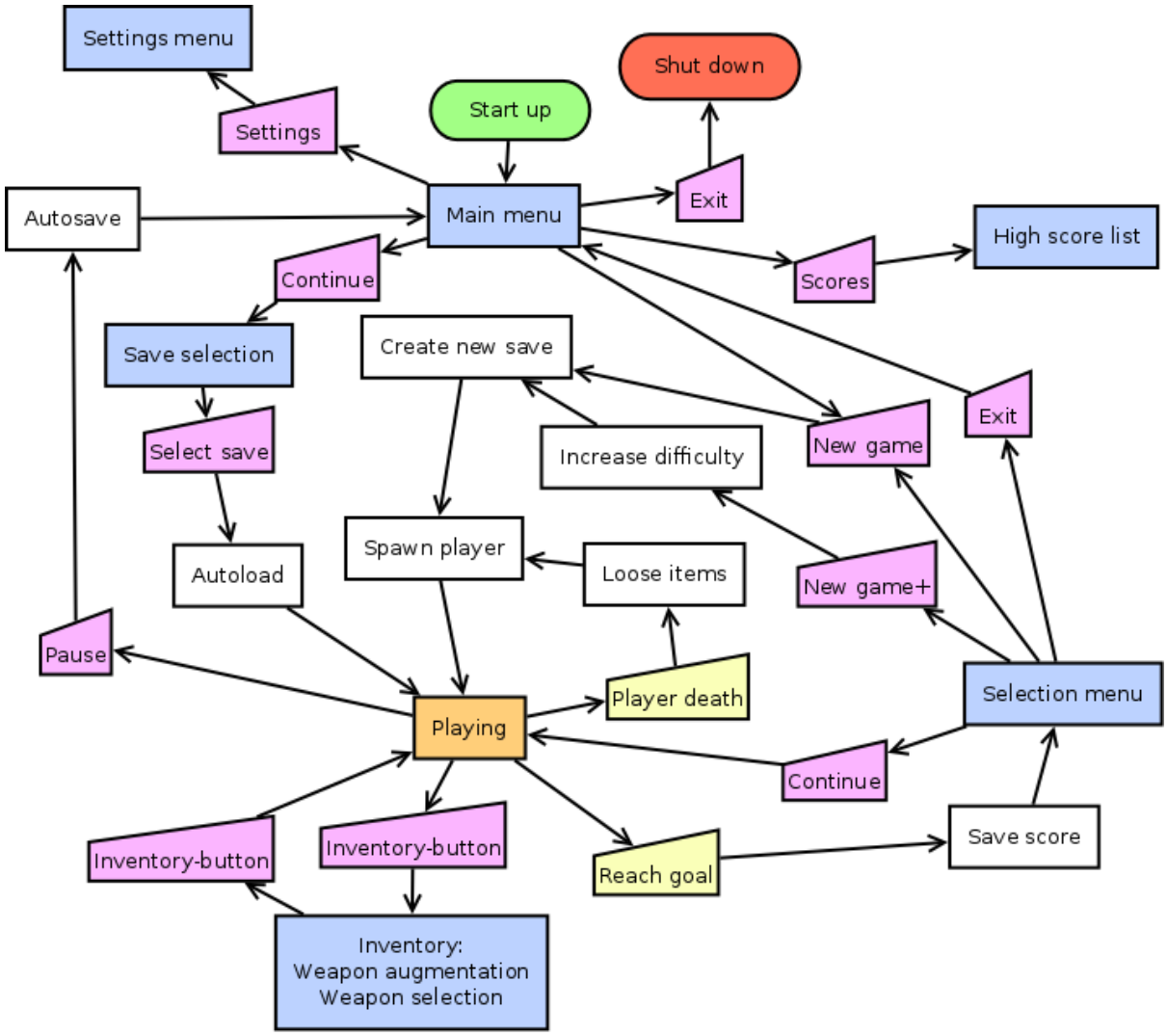
Players will be guided through a simple set of tutorials the first time they play the game. This will learn the basics of all the concepts like moving, jetpack, shooting, bombs, dust and gems. After this, the world is open for them to explore. When playing the game, pressing the E key will open the players inventory screen. This will give the player an overview of all the items he has collected, the weapon that is currently equipped and other stats.

Game Flow

The game will start by displaying a standard menu. This will allow the player to modify some options via the settings menu, start a new game, or continue a previously started game.

The chart below describes the flow of the game.

- Blue box: Menus that require user actions.
- Pink box: User actions.
- Orange box: Game play (user interacting with the game mechanics).
- White box: Internal process.
- Yellow box: Game events.
- Arrow: Flow from one state to another.



Game Controls

The game will be controlled by mouse and keyboard. It will allow players to change the control setup of the keyboard and mouse. The mouse positioning will determine the aiming direction of the player.

Default Controls

Core Controls

Action	**Keyboard**	**Mouse**
Move left	A	
Move right	D	
Jump	Spacebar	
Fire weapon / Use item		Left mouse button
Aim		Mouse cursor
Reload weapon	R	Mouse button 4
Weapon special		Right mouse button
Switch weapon configuration	1-9	
Toggle jetpack	C	
Open inventory	E	

Jetpack Controls

Action	**Keyboard**	**Mouse**
Tilt left	A	
Tilt right	D	
Increase thrust	W	Mouse scroll up
Decrease thrust	S	Mouse scroll down
Full thrust	Spacebar	

Platform

The game will use SDL 2.0 for window handling and user/system input, while OpenGL 3.0 will be used for rendering and OpenAL for audio.

This will make the game runnable on Windows, OS X, and Linux, making it available on the most popular PC platforms.

Since the game targets PC platforms, it will support keyboard and mouse input for controlling the player and using the GUI.

Sources used as GDD templates

- http://www.stemchallenge.org/students/game-design-documents/
- http://www.gamasutra.com/view/feature/130127/design_document_play_with_fire.php

F Pyroeis: Non-functional Requirements

Non-functional requirements for Pyroeis.

Pyroeis - Non-functional requirements

Performance

The program must run at at least 60 FPS on recent desktops and laptops.

Reliability

On average no more than one crash (unexpected shutdown, freeze) can happen during five hours of program uptime.

The game progress must not be corrupted by the program. If the program crashes it has to be able to restore the game progress with no more than five minutes of lost progress.

Documentation

It is required that the code is documented well enough for developers on the team or new developer can easily start working on a part that they have not worked on before.

The architecture must be documented well enough for a new developer to get an overview over how the different modules of the system are interrelated.

The documentation should not be so well done that a completely new team can continue work without a previous developer's help.

Maintainability

The program must be coded in a way that is easily maintainable by making it readable and loosely connected. Part of this requirement is to follow the coding and documentation guidelines.

G Pyroeis: Coding Convention

Coding conventions for Pyroeis. Derived from Project Nox. Now also includes format for Actor JSON documentation.

Coding conventions

- Never use **prefix or postfix** on names (variables, functions, everythings).
- **Interface classes** is an exception to the above rule. They must begin with a capital "I". e.g. class IOnEvent.
- Always use **this->** when you reference a member of a class (variables and functions).
- Use **CamelCase** on every **name** for classes, **camelBack** for variable/function names.
- **Filenames** must be identical to the name of the class it contains. If it does not contain a class, it must be lower case with underscores separating words.
- Use **printf** rather than cout.
- Never use "using namespace", always use "**namespace::thing**".
- For very long namespaces (e.g. some boost libraries), an alias can be set for it.
- Follow these guidelines in regard to pointers: http://stackoverflow.com/a/8706254
- Avoid using **global** objects.
- Avoid using **singletons** (which in practice is a global object).
- Always use **override** when overriding a base class method.
- Avoid public member variables, rather use **accessors and mutators** (get and set).
- Use **const** wherever sensible, as on methods (e.g. accessors), references, pointers, that shouldn't change the data.
- Pointers and references must be **declared with the asterisk or ampersand to the left** (e.g. int* a;)
- Headers must have include guards, rather than "pragma once". The include guards must be defined as the filename in all uppercase, with "_H_" at the end. Spaces should be ignored, and not replaced with underscore.

Everything not listed here will follow these guideline:
http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml

Documentation

Actor components should include documentation describing what values can be set in the json file for them.

Format

```
# JSON Description
- __<name>__:<type> - <description text>            // general type.

- __<name>__:object - <description text>            // object.
      + __<name>__:<type> - <description text>      // object member.

- __<name>__:array[<type>] - <description text>     // array.

- __<name>__:array[object] - <description text>     // array of objects.
      + __<name>__:<type> - <description text>      // member of object in array.
```

<type> can be object, array, string, real, bool, vec2.

vec2 is a custom defined object type with the format `{ "x": <x-length>, "y": <y-length> }`. Alternatively it can be a single numeric type. The number will be applied to both x and y when parsed.

Example

```
# JSON Description
- __property1Name__:type - Description...
- __property2Name__:object - Description...
      + __property2Property1Name__:type - Description...
      + __property2Property2Name__:type - Description...
- __property3Name__:array[string] - Description...
- __property4Name__:array[object] - Description...
      + __objectProperty1Name__:string - Description...
```

H Pyroeis: Assets List

List of assets for Pyroeis. Exported on 13.05.2014.

Asset Type	Category	Name	Animated	Status	Priority	Notes
Texture	Character	Running	Yes	Done	1	
Texture	Character	Idle	No	Done	1	
Texture	Character	Health indicator		Ready	1	
Texture	Dust	Red dust	No	Done	1	
Texture	Dust	Yellow dust	No	Done	1	
Texture	Flying Mother Enemy	Idle	Yes	Done	1	
Texture	Flying Ranged Enemy	Idle	Yes	In progress	1	
Texture	Handgun	Idle	No	Done	1	
Texture	Handgun	Reticle	No	Done	1	
Texture	Jetpack	Thrusting up	Yes	Ready	1	
Texture	Machine gun	Idle	No	Done	1	
Texture	Machine gun	Reticle	No	Done	1	
Texture	Shotgun	Idle	No	Done	1	
Texture	Shotgun	Reticle	No	Done	1	
Texture	Sniper	Idle	No	Done	1	
Texture	Sniper	Reticle	No	Done	1	
Sound	Handgun	Fire	-	Placeholder	1	
Sound	Jetpack	Thrusting up			1	
Sound	Machine gun	Fire	-	Placeholder	1	
Sound	Shotgun	Fire	-	Placeholder	1	
Sound	Sniper	Fire	-	Placeholder	1	
Texture	Bomb	Bomb	No	Done	2	
Texture	Character	Jumping up	No	Done	2	
Texture	Character	Falling down	No	Done	2	
Texture	Flare	Flare	No	Done	2	
Texture	Flying Child Enemy	Flying	Yes	Placeholder	2	Using the mother as the placeholder.
Texture	Flying Ranged Enemy	Flying	Yes	Done	2	
Texture	Flying Ranged Enemy	Attacking	Yes	Ready	2	
Texture	Jetpack	Tilt thrust	Yes		2	
Texture	Lamp	Lamp	Yes	Done	2	
Texture	Rocks	Rock 1	No	Done	2	
Texture	Rocks	Rock 2	No	Done	2	
Sound	Dust	Pick up	-		2	
Sound	Flying Child Enemy	Death	-	Placeholder	2	
Sound	Flying Child Enemy	Charging	-		2	
Sound	Flying Mother Enemy	Spawning minions	-	Placeholder	2	
Sound	Flying Mother Enemy	Death	-	Placeholder	2	
Sound	Jetpack	Tilt thrust			2	
Texture	Character	Taking damage	Yes		3	
Texture	Effect	Explosion	Yes	Ready	3	
Texture	Flying Child Enemy	Charging	Yes		3	
Texture	Flying Child Enemy	Dead	No	Placeholder	3	Using the mother as the placeholder.
Texture	Flying Mother Enemy	Dead	No	Done	3	
Texture	Flying Ranged Enemy	Dead	No	Done	3	
Texture	Flying Ranged Enemy	Taking Damage	Yes		3	
Texture	Rocks	Rock 3	No		3	
Sound	Character	Death		Placeholder	3	
Sound	Character	Running			3	
Sound	Character	Landing hard			3	
Sound	Effect	Explosion	-	Placeholder	3	
Sound	Flying Mother Enemy	Taking Damage	-	Placeholder	3	
Sound	Flying Ranged Enemy	Attacking	-	Placeholder	3	
Sound	Flying Ranged Enemy	Death	-	Placeholder	3	
Texture	Flying Child Enemy	Idle	Yes		4	
Texture	Flying Mother Enemy	Taking Damage	Yes		4	
Texture	Ground Melee Enemy	Running	Yes		4	
Texture	Ground Melee Enemy	Idle	Yes		4	
Texture	Ground Ranged Enemy	Idle	Yes		4	
Texture	Plant	Jungle tree	No	Done	4	Only one tree configuration is in-game
Sound	Character	Taking damage		Placeholder	4	
Sound	Character	Jumping up			4	
Sound	Flying Child Enemy	Taking Damage	-	Placeholder	4	
Texture	Flying Mother Enemy	Spawning minions	Yes		5	
Texture	Ground Melee Enemy	Attacking	Yes		5	
Texture	Ground Melee Enemy	Jumping	No		5	
Texture	Ground Ranged Enemy	Attacking	Yes		5	
Texture	Ground Ranged Enemy	Dead	No		5	
Texture	Handgun	Fire	Yes		5	
Texture	Machine gun	Fire	Yes		5	
Texture	Plant	Jungle grass	No		5	
Texture	Plant	Jungle flower	No	Done	5	
Texture	Shotgun	Fire	Yes		5	
Texture	Sniper	Fire	Yes		5	
Sound	Character	Falling down			5	

Sound	Flare	Burning	-		5	
Sound	Flying Ranged Enemy	Taking Damage	-	Placeholder	5	
Sound	Ground Ranged Enemy	Attacking	-		5	
Texture	Character	Death	Yes		6	
Texture	Flying Child Enemy	Taking Damage	Yes		6	
Texture	Flying Mother Enemy	Death	Yes		6	
Texture	Ground Melee Enemy	Dead	No		6	
Texture	Ground Ranged Enemy	Running	Yes		6	
Sound	Flying Mother Enemy	Fleeing	-		6	
Sound	Ground Ranged Enemy	Death	-		6	
Texture	Flying Child Enemy	Death	Yes		7	
Texture	Flying Mother Enemy	Fleeing	Yes		7	
Texture	Flying Ranged Enemy	Death	Yes		7	
Texture	Ground Ranged Enemy	Jumping	No		7	
Texture	Plant	Jungle vine	No		7	
Sound	Ground Melee Enemy	Attacking	-		7	
Sound	Ground Melee Enemy	Running	-		7	
Texture	Flare	Burning	Yes		8	
Texture	Flying Ranged Enemy	Fleeing	Yes		8	
Texture	Rocks	Rock 4	No		8	
Texture	Rocks	Rock 5	No		8	
Sound	Flying Ranged Enemy	Fleeing	-		8	
Texture	Ground Melee Enemy	Death	Yes		9	
Sound	Ground Melee Enemy	Death	-		9	
Sound	Ground Ranged Enemy	Fleeing	-		9	
Texture	Character	Jumping up while running	Yes		10	
Texture	Character	Walking	Yes		10	
Texture	Dust	Green dust	No	Done	10	
Texture	Dust	Blue dust	No	Done	10	
Texture	Effect	Core	No	Ready	10	
Texture	Effect	Smoke	No	Ready	10	
Texture	Effect	Flame	No	Ready	10	
Texture	Ground Ranged Enemy	Fleeing	Yes		10	
Texture	Ground Ranged Enemy	Death	Yes		10	
Texture	Handgun	Reload	Yes		10	
Texture	Machine gun	Reload	Yes		10	
Texture	Shotgun	Reload	Yes		10	
Texture	Sniper	Reload	Yes		10	
Sound	Character	Walking			10	
Sound	Ground Melee Enemy	Jumping	-		10	
Sound	Ground Ranged Enemy	Jumping	-		10	
Sound	Handgun	Reload	-		10	
Sound	Machine gun	Reload	-		10	
Sound	Shotgun	Reload	-		10	
Sound	Sniper	Reload	-		10	

I Pyroeis: Meeting Log

Log of meetings with external persons such as the supervisor and employer.

09/01 Jayson

Milestone for each month - break up into sprints.
One week code review with previous engine to see if it needs refactoring.

Milestone 0: Refactor, research, prototype world, and complete design.
Milestone 1: All of the basic gameplay, world generation. About 75% of gameplay.
Milestone 2: Advanced features. 90-95% gameplay complete.
Milestone 3: Decide what to do based on what time is remaining. Feature lock? Only bugs?

Document decision we make on how things are implemented.
Three weeks on report.
Milestone review after each milestone - forming a chapter in the report.
One milestone could be to make it playable on Windows.
When to have placeholder assets, when to have production assets.
Asset list.
Documenting everything.
Use tags in atlassian tools.
Tag what task you do in toggl.
Technical brief about what (doesn't) need to be refactored.
Keep a broad implementation of features.

Gems for boosting items.

Use the same technique for both cave creation and destruction (e.g. cutouts).
Talked about how to generate caves.
Use the xy coordinate as a string to identify the chunk.
Parameterize cave generation based on water etc.

Clickable installer.
No crashes.

Talk about how game design affects the system engineering.
Analyzing game balancing (cost, weight).
User testing with different parameters (num gems, etc.)
User feedback message at a position in the world.

A sonar device where thin walls light up.

Refactoring:

- Height in Box2D wrapper - make it depth?
- AI pathing.

10/01 Ådne (employer)

Signing papers.

One point have to be rewritten.

15/01 Mariusz (supervisor)

- Try to get Simon as “product nagger”.
- Ådne could check product against specs.
- Think about the structure of the thesis and work backwards.
- Sell your story to the examinator. (video, images etc)
- Make big deal of the stuff.
- Reference someone elses work, especially academic papers.
- All about how you approach the problem.

17/01 Mariusz (supervisor)

- Quantify things. Do not just say “short”, say an actual period.
- Testing: unit testing.
- Libraries for user analytics?
- Use bibtex in latex.
- Google Scholar for citing with bibtex.
- Write about each sprint. What was planned, what was completed.
- Section about risk assessment in project plan.
- Describe that because of the agile methodology we mitigate risks by decomposing tasks.
- Let the user save the seed for a world?
- Describe the overview of the game in a non-technical way.
- Decouple technical from non-technical.
- Spend time describing the jetpack mode since other games don’t have it.
- Not side-scroller.
- Have to have progression against goal and give feedback on this.
- Tell the longer version of the story. What happens after he wakes up?
- Begin GDD with the vision of the game as if we would have two years for it. Then describe what we actually will do during the project.
- Expand GDD first sections to be more broad.
- Tell exactly what features (weapon, enemy) we are planning for each milestone.
- How do the player get to the core when they are down there?
- Game structure/flow should not be under user experience.
- Vision up front. Technical detailed down.
- Section about aesthetics (feel, mood). Probably in user experience.
- More on performance requirements.
- Plan what assets to use before the last minute so that we don’t rush it to the end.
- Discuss about time vs story points estimation.

24/01 Mariusz (supervisor)

- Say what bitbucket is.
- Combine or specialize the performance risks.

- Legal issues with libraries is dealt with before you pick them, so not an issue.
- Might be an issue that libraries can suddenly be closed source.
- Can have some smaller risk like the art guy quitting.
- Using scrum is a risk.
- A little bit more detail of the expectation of milestones.
- Remember to reflect on the process in the thesis.
- User testing milestone goal is ambiguous.
- Measure the world/chunk generation/loading in technical detail in the thesis. How much storage needed?
- Trail of where the player has been.

03/02 Mariusz (supervisor)

- Mariusz thinks it is okay to write about things implemented in game programming course. He will check with Simon.
- Write about multiple platforms in the thesis.

Mariusz (supervisor)

- Crash in dynamic tree on removeBody…
- Reticle parts disappear.
- Measure performance before and after and write the numbers down. Important for the thesis.
- Prioritize making it a game. Move milestone 3 stuff like goal to milestone 2, and features in milestone 2 to 3. With this we minimize the risk of the gameplay being good.
- Maybe prioritize creating the tutorial.
- DO RECORD TIMINGS.
- Discuss changes in plan.
- Match all the features to the experience. Describe why certain decisions were made, why that enhances the goal experience.

14/02 Mariusz (supervisor)

- Should maybe make jetpack controls easier.
- Implement goal and progress.
- Keep the jetpack advanced and realistic, but make it more accessible to new users.
- Jetpack stabilization mode.
- Bomb detonation timing might feel unnatural. Should maybe detonate one after one in the order they are placed. Or having different registers that detonate a group of bombs in the same register. Each register could have a matching color on bomb and trigger.
- Maybe weak ai’s can flock around the caves splitting up and grouping together.
- Bombs should scare the ai’s.
- Do analytics on the website and analyze what was working and not working in regards to sharing and promoting.

- Provide services to subscribe to where we post updates, like twitter. Better with channels specific for the game (a Pyroeis twitter account).
- Start collecting data as soon as we have the actual gameplay.

21/02 Mariusz (supervisor)

- Line of enemies flying towards you seems dumb.
- The terrain should encourage use of bombs. Just having more horizontal caves might help. More narrow corridors.
- Plan how the game should feel. Strategic? Action shooter? Decide on this and then test that gameplay.
- Have video of game walkthrough for the thesis.
- In the thesis we should show how the game is planned to be even though we haven't implemented it. Like other types of caves, and bosses.
- Describe changes to the system in bulkier chunks in the thesis.

28/02 Mariusz (supervisor)

- Timestamping events from user tests.
- How much time players spend in places (inventory, menu, etc.)
- Ability to regain health.
- Describe workflow and tools in the thesis. Reflections on how we worked (good, bad).
- Healthbar on player.

15/02 Mariusz (supervisor)

- Easier beginning, hook players.
- Players might not realized that child enemies are spawned by mothers.
- Controls are good.
- Might not need to automatically stand up when crashing.
- Vsyncing not working on mac when not on screen might be a good research topic. http://wiki.libsdl.org/SDL_WindowEventID
- Describe multithreading handling in the thesis.

28/03 Mariusz (supervisor)

- Friendly monster would make it more "alive".
- Balancing exploration and games.
- Snapshot tests and write about them (like lighting blur).

12/04 Mariusz (supervisor)

- Images of the cave generation improvement will be good for the thesis.
- Shame that we still are coding and not completely focusing on the thesis.
- Not completely rewinding the progress when dead could improve experience.

25/04 Mariusz (supervisor)

- Focus completely on report for a block of time. Don't be distracted by coding urges.
- Might be some problems with the actor extend system when extending from more than one.
- Old caves with target depth felt like progressing. With the new caves this feel isn't there. Maybe show a "score" that you can compare with your previous when you die.
- Death with progression is a good idea.
- Oppdragsgiver helps us with the business. Pyroeis game is a way of making the features of the engine. The engine will be used for the company.

29/04 Simon Bachelor Update

- Presentation more content and topic, not a story about our work. Content rather than narrative.
- Understanding limitation: Understand why the product isn't as good as you wanted it to be. Missing stuff, bugs, why.
- Should probably do some more feedback gathering. Friends, supervisor, anyone. Can probably write something about previous feedback and what we did we it.
- Information presentation: Useful and interesting. Structured with diagrams showing flow and structure. Graphical figures. Class diagram. Flow diagram.
- Expect your reader to be a third year programmer. Write for the people around you.
- Don't write narrative, write in a logical structure.
- Better to mention other work that you got inspiration/ideas from.

09/05 Mariusz (supervisor)

- Maybe change development process to something like project process.
- Consistent font size in figures.
- Not references inside lines. It breaks context.
- Use tilde after e.g. and the likes.
- Deployment section.
- First letter capital when referring to appendix, fixture and such. e.g. Appendix, Figure.

16/05 Mariusz (supervisor)

- Don't reference SO. Reference the source of information. Could reference it as a private communication with the person that wrote it. Same goes for wikipedia.
- Create a test for the smart pointer performance. Test with and without optimization.
- Try to only inline the virtual stuff, and not optimize.
- Document the CPU boost experience.
- Write gotchas in the conclusion section.
- Maybe write about tutorial in testing? I don't remember doing it.
- Buttons more button-like.
- Describe how to make the engine more modular.

- For further development talk about improving engine, supporting mobile, deal with controls, multiplayer, tutorials.
- Curly braces around reference titles prevents
- \tt can be used to make text monospace.
- \- can be used to split links.

J Pyroeis: Daily Scrum Log

Notes from each daily scrum meeting.

13/01 Daily Scrum

Asbjørn

Did:

- Set up some debug rendering mode for the lighting.
- Fixed edit actor check.
- Lighting working (increased length to 100).

WIll do:

- Start designing wheel component.

Magnus

Did:

- PYRO-1
- Setting up JIRA.
- Researching user stories

Will Do:

- Cave destruction design.

14/01 Daily Scrum

Asbjørn

Did:

- Wheel component.
- Problems with functional (std::bind).

Will Do:

- Callback on sensor giving information if on ground or not.
- Stop moving while not on ground.
- Jumping.
- Start refactoring code.

Magnus

Did:

- Destruction of caves working.
- Buggy destruction when having a very small area.

Will Do:

- Fix bugs regarding destruction.
- Check performance with light.

15/01 Daily Scrum

Asbjørn

Did:

- Finished jumping.
- Created jetpack component.

Will Do:
- Finish jetpack.

Magnus
Did:
- Finished destructible environment.
- Started movement review.

Will Do:
- Begin cave generation.

16/01 Daily Scrum

Asbjørn
Did:
- Completed jetpack features.
- Fixed problems with player movement.

Will Do:
- Talk with Mariusz about mac app.

Magnus
Did:
- Started generating random caves.
- Revised project plan.

Will Do:
- Continue cave generation.
- Several cave paths with splits.
- Test how to attach segments.

17/01 Daily Scrum

Asbjørn
Did:
- Fixed jetpack.
- Documented jetpack.
- Reviewing code.
- Testing app on older OS X.
- Command line input for logger.

Will Do:
- Complete documentation and qa.
- Work on getting OS X app to work.
- Tes XCode 5.1

Magnus
Did:
- Dig caves.

- Reviewed jetpack.
- Fixed Box2D area assertion.
- Disabled polygon simplifying which caused polygons being buggy.

Will Do:
- Complete documentation and qa.
- Fix inconsistency in generation on different platforms.

20/01 Daily Scrum

Asbjørn

Did:
- Worked on OS X app.

Will Do:
- Handgun story.
- Controller configuration.

Problems:
- Problems with different versions of system libraries on different OS X version. Can only get it to work on Mavericks.

Magnus

Did:
- Nothing.

Will Do:
- Infinite caves.
- Look at integrating terrain with the rest of the system.
- If there is time

21/01 Daily Scrum

Asbjørn

Did:
- Keyboard mapping. Complete, but a bug is remaining.

Will Do:
- Fix keyboard bug.
- Implement the handgun.

Magnus

Did:
- Halfway integrated world generation with actors.
- Changed physics interface to support requirements of world generation.
- Worked on Box2D area assertion.

Problems:
- Assertion is super weird.

Will Do:
- Fix area assertion. Finish merging physics interface changes.

- Merge in stashed physics interface.
- Structure should be done today.
- Maybe infinite caves.

22/01 Daily Scrum

Asbjørn

Did:

- Fixed keyboard mapping feature.
- Started on handgun. Ability to shoot in any direction.
- Tested new sprite sheet.

Will Do:

- Risk analysis.
- Handle bullet hit world/enemies.

Magnus

Did:

- Infinite generation of worlds are now working in a basic form. Fixed bug with weird box spawning at the local origin of each chunk. Started looking at despawning chunks. Reviewed Keyboard Config. Changed boost points to glm::vec2 this will match our system more.

Will Do:

- Manage despawning chunks.
- Fix geometry sets, Two modes.
- Saving the world to a file.
- Split chunks into more bodies
- Cleaning.
- Managing/cleaning game design document.

24/01 Daily Scrum

Asbjørn

Did:

- Shooting works.
- Wrote risk analysis.

Will Do:

- Fix crashing.
- Document weapon classes.
- Polishing project plan (risk and milestones).
- Write about music and sound in GDD.

Magnus

Did:

- Optimizing geometry
- Unloading chunks

- Cave Generation refactored.
- Seams are nicer.
- Plan/GDD
- Generator equal on Mac, Windows and Linux.

Will Do:

- Document world generator
- Reviewing handgun
- GDD restructuring/Writing.

27/01 Daily Scrum

Asbjørn

Did:

- Looked at lighting.
- GDD and plan.

Will Do:

- Fix jetpack thrusting bug.
- Start work with bombs.

Magnus

Did:

- GDD, Plan, delivered.
- Designing improvements for the debug rendering of the world generator.

Will Do:

- Image view, Button these will be done
- Text rendering will be started on.

28/01 Daily Scrum

Asbjørn

Did:

- Bombs destruct terrain and are sticky.
- Looked at performance (union of polygons).
- Fixed jetpack bug.

Will Do:

- Complete bomb implementation.
- Debris.

Magnus

Did:

- Image View Done
- Button functionality Done
- Text rendering and label view are nearing completion, not efficient at this point
- Refactoring view classes, cleaning, documentation.
- Fixed Flipping bug when landing upside down.

Will Do:

- Document List view, the rest of the view classes
- Review Text rendering.
- Complete View Classes
- Fix PYRO-40 Adding the same fixtures to the debug rendering.
- Spawn at the surface.

Discussion

A discussion was had on the issue of the problems of how to solve the issue of collision filtering, Box2D offers up the masking, group and contact filtering methods. Allowing us to filter the collisions ourselves is probably the most convenient system, but also requires some system to handle it. This will be figured out at a later time.

29/01 Daily Scrum

Asbjørn

Did:

- Reviewed image and button view functionality.
- Bombs are loaded from file.
- Completed bomb documentation.
- Almost complete with debris.

Will Do:

- Reviewing rest of the view branches.
- Finding and adding assets to the game.
- Working on report.

Magnus

Did:

- GUI done, ready for review.
- Started looking at players spawning
- Reviewing bomb throwing branch
- Helping out with debris generation.
- Writing document for proposals to system changes.

Will Do:

- Make sure the Start report for world generation
- Set up Latex document and repo.
- Perhaps review debris.
- Player spawns at the surface.

30/01 Daily Scrum

Asbjørn

Did:

- Reviewed all GUI branches.

- Small enhancements to debris branch. Fixed some bugs. Documented.
- Reviewed player spawn.
- Reviewed physics_debug_fixes.
- Started adding textures: bomb, pistol, mars rock 2.

Will Do:

- Look at bachelor thesis.
- Further design enemies.
- Latex.
- Adding more assets.

Magnus

Did:

- Fixed small issues and optimization for the debug rendering.
- Player is spawning at the surface.
- Event manager is now thread-safe.
- Set up Latex document.

Will Do:

- Reviewing Debris branch
- Look at bachelor thesis.
- Further design enemies.
- Latex.
- Writing up world generation and destruction report.

31/01 Daily Scrum

Asbjørn

Did:

- Found what was wrong with the lighting.
- Read some bachelor thesis.
- Looked at latex.
- Random size debris.
- Fixed lighting this morning.

Will Do:

- Meetings.
- Adding some sounds.

Magnus

Did:

- Setting up webpage.
- Looked at old thesis.
- Work on thesis (front page and caves).
- Merged debris.
- Helped out changing the way scaling works in the game.

Will Do:

- Meetings.
- Start looking at splitting the chunks into smaller bodies.

03/02 Daily Scrum

Asbjørn

Did:

- Minor tweaks (textures).
- Making Windows installer.
- Making Mac installer.
- Recording video.

Will Do:

- Look at proposals.
- Start working on weapons (shotgun and smg).

Magnus

Did:

- Making Linux package.
- Website blog post with download.
- Fixed json-parsing.

Will Do:

- Splitting terrain into bodies.
- GUI.

04/02 Daily Scrum

Asbjørn

Did:

- Ability to switch weapon.
- Bullets won't collide with each other or the wielder.
- Shotgun done.
- SMG done.
- Bullets' velocity are affected by the wielder's velocity.

Will Do:

- Review of terrain split.
- Start working on sniper.

Magnus

Did:

- Terrain is now split into the number of quads we want, this improves performance a lot.

Will Do:

- GUI, starting with system for scenes, able to switch between the states. adding buttons. Start/exit.
- Some more writing in the report about generation.
- Reviewing guns.

05/02 Daily Scrum

Asbjørn

Did:

- Fixed spawning of bullets in update (smg).
- Bombs with player speed.
- Performance/crash issues with terrain split.
- Began looking at reticle.

Will Do:

- Work on reticle. Add weaponslots in Json
- Write report (movement and player).

Magnus

Did:

- Terrain splitting is now implemented, increasing performance improvement a lot.
- Helped out with crashing/performance increasing.
- Reviewing new weapons and the system.
- Started working on the scene system for menus/playing, almost complete.

Will Do:

- Complete scene system.
- Adding start button/menu with the start button
- Hopefully exit and pause as well.

10/02 Daily Scrum

Asbjørn

Did:

- Completed reticle feature.
- Reviewed main menu branches.
- Wrote about player movement.

Will Do:

- Reviewing physics disabling.
- Sprint review / planning.

Magnus

Did:

- Implemented scene system.
- Created main menu.
- Fixed reticle jittering.
- Reviewing sniper branch
- Fixed exit/pause/new game
- CMake script is now able to activate all optimizations fast.
- Disabling of bodies nearly done.
- Changed actor find/has

Will Do:

- Disabling of bodies completion
- Sprint review.
- Document.

11/02 Daily Scrum

Asbjørn

Did:

- Testing body disabling and found a bug that we fixed.
- Reviewed body disabling.
- Review headlight and light lag fix.
- Filmed sprint 4.
- Build sprint 4 version for Windows and OS X.
- List assets.

Will Do:

- Start working on enemies.

Magnus

Did:

- Disabling bodies out of range.
- Fixed crash where bodies were not initialized before being used.
- Flashlight, fixing light.
- Assets.
- Blogpost
- Linux build.
- Started working on flares.

Will Do:

- Finish blogpost
- Finish flares
- create wall lights

12/02 Daily Scrum

Asbjørn

Did:

- Merged actor template functions.
- Worked on enemies.
- Worked on the art with Jakob.
- Tested flare feature.

Will Do:

- Complete monster children attacking.
- Mother spawners.
- Reviewing light branches.

Magnus

Did:

- Fixing Blogpost
- Flares and bombs
- Wall lights

Will Do:

- Saving the states of everything
- Check lighting algorithm.
- Enable mouse when in edit mode.

18/02 Daily Scrum

Asbjørn

Did:

- Added child monsters.
- Optimized AI.
- Optimized overall performance (smart pointers and inlining).
- Worked on art with Jakob.
- Merged save/load features.
- Did some playtesting (the game is a tad hardcore, especially controls and enemies).
- Implemented HUD with num kills, target depth, weapons and equipment.

Will Do:

- Add mother monster that spawns children.
- Make mother monster hide when seen.

Magnus

Did:

- Optimizing the lighting algorithm
- Testing the AI
- Saving and loading the world.
- Helping Nordling with the rendering.
- Merging HUD, AI and Optimization branches.

Will Do:

- Fix the deltatime bug.
- Create boss encounter.

19/02 Daily Scrum

Asbjørn

Did:

- Mother monster working.
- Merged delta time fix.
- Started looking at monster fleeing.

Will Do:
- Complete mother fleeing.
- Work on children finding their way to the player.
- Work on children flocking.

Magnus
Did:
- Fixed physics delta time.
- Worked on the boss cave, finished generation of the cave. Timer is working. Started researching/implementing monsters spawning.
- Fixed gui rendering performance.
- Review AI mother spawner.

Will Do:
- Finish the boss fight.
- Add endgame scenes.
- Start checking out the scoring system.

20/02 Daily Scrum

Asbjørn
Did:
- Completed mother fleeing feature.
- Worked on mother spawning children when hidden.
- Added new sprites.

Will Do:
- Scale sprites properly.
- Make children flock.

Magnus
Did:
- Reviewed child spawning branch
- Boss battle.
- Red dust is spawning
- Designing pickupable

Will Do:
- Review AI hiding branch.
- Pick up dust, attraction.
- Boss battle finished.
- Yellow dust.

25/02 Daily Scrum

Asbjørn
Did:
- Completed mother fleeing and spawning children.

- Added new sprites.
- Merged branches.
- Fixed AI.

Will Do:

- Implement ranged enemies spitting goo.

Magnus

Did:

- Reviewing Spawning mothers.
- Red and yellow dust is pickupable by the player.
- Boss battle progression.
- Boss battle finished,
- Textures and light and dust.
- Fixed a lot of issues for the release.
- fixed alot of small issues with the rendering of circles, physics behaving weirdly. etc.

Will Do:

- Inventory and inventory screen.

26/02 Daily Scrum

Asbjørn

Did:

- Almost finished ranged flying.

Will Do:

- Make ranged flying spawn together with mother.
- Review inventory branch.
- Work on roaming enemies.

Magnus

Did:

- Finished inventory system.
- Discussing design of UI, inventory
- Upgrading Jira

Will Do:

- Make actual inventory screen.

27/02 Daily Scrum

Asbjørn

Did:

- Completed goo spitting monsters.
- Made monsters roam.
- Reviewed inventory.

Will Do:

- Explosion and collision damage.

Magnus

Did:

- Inventory screen is now done.
- Reviewing AI branches.
- Discussing inventory design
- Discussing AI design.

Will Do:

- Create the crafting system for the inventory.
- Add the assets to the inventory screen.

28/02 Daily Scrum

Asbjørn

Did:

- Merged tweaks.
- Implemented that explosions apply damage.
- Implement some sounds.

Will Do:

- Adding more sounds.

Magnus

Did:

- Inventory screen is almost complete

Will Do:

- Finish the crafting of the items and possibly the gems.

13/02 Daily Scrum

Asbjørn

Did:

- Multisampling, mipmapping, and supersampling.
- Fixed release builds.
- Started work on terrain texturizing.
- Merged pull requests.

Will Do:

- Complete terrain texture.

Magnus

Did:

- Dust attraction implemented.
- Linux build.
- Web page.
- Actors that are outside the active area will be deactivated.
- High performance light (no shadows) .

- Merged dreki terrain results.]
- Supersampling help.
- Fixed AI crashing. broke guns.
- Fixed guns.
- Biomes

Will Do:

- Biomes

14/02 Daily Scrum

Asbjørn

Did:

- Reviewed pull requests.
- Worked on terrain texturing. All OpenGL stuff ready.

Will Do:

- Continuing with texturizing.
- Create new texture atlas with plant sprites

Magnus

Did:

- Fixed issues in pullreques branches.
- Worked and planned the biome caves.
- No plants are in the game yet.

Will Do:

- Plants will hopefully be placed in the game using the proper temperature and water parameters.
- Figure out the proper way to set the water and temperature parameters.

17/02 Daily Scrum

Asbjørn

Did:

- Reviewed pull requests.

Will Do:

- Render background texture.

Magnus

Did:

- Merged antialiasing branch.
- Created pull request for light aabb check.
- Testing inline vs virtual vs normal performance.
- Writing document, world generation.

Will Do:

- Write about world generation.
- Make plants spawn in the world.

19/02 Daily Scrum

Asbjørn

Did:

- Textures on walls are rendered.

Will Do:

- Render texture on the actual terrain bodies.
- Might start implementing weapon upgrading.

Magnus

Did:

- Planting plants.
- Plants on ground and plants on roof.
- Green dust is spawning.
- Writing on the document, about procedurally generated terrain.

Will Do:

- Might write document.
- Fix plant spawning, currently only use active bodies.
- Fix lag when generating a chunk.
- Make plants spawn by the parameters.

25/03 Daily Scrum

Asbjørn

Did:

- Finished world texturization.
- Reviewed pull requests.
- Balance weapons.

Will Do:

- Start implementing weapon upgrade system.
- Help with crash on plant branch.

Magnus

Did:

- Implementing plant spawner
- Actor creation is now multithreaded, this increases the performance but has caused some bugs that need to be fixed.
- Linux package is now easier to use.
- Reviewing texturizing branch.

Will Do:

- Fix crashes caused by multithreaded actor creation.
- Make plants vary with heat and moisture.

26/03 Daily Scrum

Asbjørn

Did:

- Finished world texturization.
- Reviewed pull requests.
- Balance weapons.

Will Do:

- Start implementing weapon upgrade system.
- Help with crash on plant branch.

Magnus

Did:

- Implementing plant spawner
- Actor creation is now multithreaded, this increases the performance but has caused some bugs that need to be fixed.
- Linux package is now easier to use.
- Reviewing texturizing branch.

Will Do:

- Fix crashes caused by multithreaded actor creation.
- Make plants vary with heat and moisture.

26/03 Daily Scrum

Asbjørn

Did:

- Helped out fixing feature/biome crash.
- Researched performance issues.

Will Do:

- Work on weapon upgrade system.

Magnus

Did:

- Fixed crash in biome branch
- Improve performance in biome branch.
- Tweak amount of spawned actors, and plant health.

Will Do:

- Document the rest of the branch.
- Make the biome and plants vary by heat and amount of water.
- Might start working on varying terrain.

27/03 Daily Scrum

Asbjørn

Did:

- Created weapon customization scene accessed by clicking weapon.
- Started work on weapon power component.
- Reviewed plant spawning branch and fixed performance.
- Experimented with blurring light.

Will Do:

- Finish weapon power component.
- Work weapon screen.

Magnus

Did:

- Finished and created pull request for biome branch.
- Plants are now spawned based on heat and moisture, this is purely based on the depth atm.

Will Do:

- Create some dry plants, and test the creation of different types.
- Make the heat and moisture levels vary by the longitude.

02/04 Daily Scrum

Asbjørn

Did:

- Experimented with blurring light.
- Tested performance and Windows.
- Weapon customization.
- Filmed video for update.
- Split Mac build into libs.

Will Do:

- Finish weapon customization (use gems and power).

Magnus

Did:

- More plants are added
- Terrain is now varying with heat and moisture, this is specified in the world file.
- Fixed issue where the plants were casting the ray to short.
- Started working on plants despawning when not attached to terrain, this was too complex.
- Bombs now have an exponential falloff on their damage, they damage more in the core and less at the edges of the explosion.

Will Do:

- Fix the issue with destruction not being applied to other chunks.

23/04 After-easter update

Asbjørn

Did:

- Worked with getting it to work on older macs.
- Discussed with Stian about the music.

Will do until next week:

- Focus on writing the report.
- Review saving branch.

Magnus

Did:

- Fixing the saving issues, not yet complete.

Will do until next week:

- Finish fixing the saving issues.
- Focus on the report.

K Pyroeis: Progress Reviews

Sprint and milestone reviews.

Sprint 1

Goal

- Prepare code.
- Walk and fly around in random generated caves.

The Plan

- Refactoring.
- Reviewing the code.
- As a player I want to move the character.
- As a player I want to fly with a jet pack so that I can more easily explore the caves.
- As a player I want to experience randomly generated caves.

The Result

Most of the tasks we set for sprint one where completed. Randomly generated caves can now be explored with the Jetpack and running around in them is also possible. We also reviewed some of the engine code and documented parts of it. This is something we wished we had more time for.

Basic features of the cave generation issue were completed, but still some of the acceptance criteria was not met. Missing parts are that the world should be infinite in size, that they are naturally formed and reproducible. Another problem is that the generation is different on the various platforms, due to the random seed being different.

The reason that the randomly generated caves feature was not completed is that it actually didn't fit into one sprint. We estimated one week and three days for this story, but the sprint only lasted for one week.

Although we did not manage to complete all the tasks set for this sprint, the goal was achieved.

Retrospective

- Splitting tasks into smaller tasks worked well. Easier to estimate and to see progress.
- For some tasks (Documentation/Cave generation) the splitting did not work as well)

Sprint 2

Goal

- Prepare GUI.
- Interact with the world and be challenged.

The Plan

- Support images, buttons, and labels in GUI system
- As a player I want to experience randomly generated caves

- As a player I want to throw bombs that explode walls so the area becomes available to explore
- As a player I want to shoot with a handgun
- As a player I want configure the controller mapping from a configuration file so I can control the character the way I prefer.
- As a player I want to be challenged by flying ranged enemies.

The Result

For this sprint we were met with alot of setbacks and tasks being alot more time consuming than we initially thought. The monday started of pretty badly, due to the mathematics course being in the start of the day and oversleeping. This day was also cut short due to the fact that we ended up having a meeting with the rest of the guys at Suttung Digital, discussing the future of the company and our game idea for the next year. Refining the project plan and GDD also took some time.

The GUI system was not started this sprint. Randomly generate caves are now working, and generating an infinite range of caves. These will spawn and despawn from the rendering and physics to allow the game to perform well even when the player has traveled great distances. Bomb throwing was not started this sprint. Players are now able to fire bullets in the world, the bullets will affect actors and physical objects in the world. The bullets will disappear when they hit something.

Creating the firing mechanism took us a lot more time than we originally estimated (16 hours instead of 4). Most of the development time for this particular task was spent on the development itself and on the necessary research. The delay was not caused by problems when implementing, the problem was just bigger and more complex than I had anticipated. On a personal note, this was not the best week I've had as a programmer (-Asbjoern)

Mapping your prefered controls is now possible by using the configuration file, all controls are now loaded by this. The AI was not started this sprint.

As is evident we barely finished half the tasks we set out to achieve for this sprint. The GUI is not implemented and the player is not challenged by the world in any real sense. What we did achieve is quite interesting though, the player is able to fly around in infinitely generated caves that are random and varying in size and shape. The randomness is also equal on all the three platforms we are targeting.

Retrospective

- Better estimates, we need to be more realistic in our measurement of how much time we will need for each task.
- Lots of other work than development also take our time, like meetings and planning/design. Therefore we can't expect to be able to complete more than four days of development during a sprint.

Sprint 3

Goal

- Prepare GUI (again).
- Destroy world.
- Playable world.

The Plan

- Complete implementation of basic GUI views (image, button, label).
- The player can throw bombs and trigger these to destroy the terrain.
- Debris will spawn when terrain is destroyed.
- Spawn the player at the surface of mars with one entrance.
- Fix jetpack thrusting on low fps.

The Result

All the issues were completed, so we achieved the goal!

The view implementations took a lot less time than estimated, one day less than the estimated four days. The label and text rendering implementation was what took most of the time, but was still below the estimate. Some more time was used on refactoring the view classes than anticipated. With all this, we have a basic GUI system that is ready to be used.

The player can now throw bombs by right clicking the mouse, and trigger them with the middle mouse button. The bombs will destroy the terrain when triggered. This task was completed faster than estimated.

Debris will spawn when the terrain is destroyed. This feature took exactly the estimated time, and works as expected. The debris even got a texture!

The player will now spawn at the surface of Mars, rather inside walls. This also implies that the world has an upper limit, which is the surface. This also took less time than estimated, and works really well.

For this sprint we even managed to do some non scheduled tweaking, allowing the debris to spawn in different sizes and the explosions affecting the physical world. We also added some assets to the world. The player's bombs and shots have textures that make them visible even when not in debug mode. rocks that spawn as debris also have textures that make sure we are closing in on the game being more of an actual game, with some simple gameplay. You are now able to run around in the world and destroy it. The physics simulation is quite fun, and engaging.

Retrospective

For this sprint we were very decent with our scheduling and timing of how much time things would actually take us, some tasks where completed faster than expected, but this is a lot more realistic scheduling than the previous sprints.

The conclusion we reached in the previous retrospective – expecting no more than four days of work – worked perfectly. We removed one day of work to fit in the thesis, website, and more design. We also estimated one extra day of work to have some stretch tasks. This ended up in 4 days of work estimated.

For the next sprint we will estimate 4 days of work, plus 1 day of stretch tasks that are not expected to be completed. This is of course per person.

Milestone 1

Goal

- Complete project plan.
- Complete GDD.
- Complete non-functional requirements specification.
- Core graphical user interface implementation.
- Engine refactoring.
- Simple (10-25%) gameplay complete:
    - **World generation prototype**: Basic world generation and destruction will be fully implemented. This includes a world that is playable, both in terms of performance and movability, and infinite in size. The player will be able to access the caves from the surface of Mars.
    - **Simple movement**: Players will be able to move around the world by running, jumping and using the jetpack.
    - **Destructible world**: The player can destruct the world with explosions.

The Result

Project plan, with requirements, and initial GDD is completed and delivered. The core graphical user interface is ready to be used. Game engine has gone through a quick quality check and documentation has been added to most of the parts missing.

For the gameplay, everything was completed. In addition to this, we added some assets, and a few extra features and enhancements.

The world generation prototype didn't take as long as we estimated, and the results are much better than what we expected. It is an actual playable world. With this, we know that the world generation and destruction, which are core features, won't be halting our progress.

There are some performance issues regarding world destruction, physics simulation, and debug rendering, but we have fixes for this planned that won't take more than 1-2 days of work.

Movement is also working perfectly. It features both a classical side-view movement where the player is standing with a fixed rotation, and a fully physics affected mode where you hover with a jetpack. The classic mode required some "hacks" that aren't physically simulating movement of legs. A wheel is used instead of legs. With the right friction and motor torque, this works pretty well, and

the player will naturally have problems walking up steep hills. A sensor is also attached to the wheel so we know when we are able to jump or move with the wheel.

In addition to the planned features we were also able to complete the following task: shooting. Players are able to fire a handgun, the bullets will affect the world and have a callback function that is used to control what happens when they hit an object. The shooting is not complete but at least more progress than we anticipated is made.

We were also able to add some assets to the game. This makes it a whole lot more game-like than just using the debug rendering.

Retrospective

The milestone was planned out to me smaller than what we actually have achieved. We have also started doing some of the tasks that were planned for milestone number two. Although this is not a bad thing it means that our planning is off when it comes to the time that is actually needed in order to complete the project. Hopefully this means that the next milestones does not turn out to be too large for us to complete in the given time. Overall we are quite happy with the amount of work we were able to complete for this milestone.

Sprint 4

Goal

- Better user experience with main menu.
- Be challenged by flying enemies, and challenge them.

The Plan

- Usable main menu with start/pause/exit.
- All four weapon types available and working.
- Reticle for the weapons.
- Minion-spawning mother monster attacking you.
- Split terrain into smaller bodies.
- Disable physics bodies outside the “active area”.

The Result

The sprint was severely hampered by family issues (funeral). Asbjørn was away both thursday and friday during the actual work time, and was not able to contribute anything of significance these days.

We managed to complete the menu system, the reticles for weapons and get all the weapons working.

Both the scene system needed to handle the menu, and the menu itself with the buttons took longer than expected to implement. Moving the HumanView logic over to the GameScene was the thing that took the longest time. Also getting the button list to work properly with our fantastic (sarcastic) view system did take a lot of time. Pausing the game also took some time, since we had to get the

mouse to re-appear, and implement the functionality for resetting the game when pressing “New Game”.

The reticle system is currently working very well. Weapons can have different reticles with different parts that can all be customized very easily from each weapons Json files. We have not yet settled on how we are going to do the actual rendering of reticles and other HUD (Heads up display) elements. Currently the reticles are rendered like any object in the world, meaning that they are affected and require lighting to be seen by the player. This is not an ideal situation. A discussion is needed on how we are planning to solve this problem.

Our epic task for the enemies is not yet even started.

The terrain is now split into several bodies, rather than on large. This improves the destruction performance a lot since it only has to apply the destruction on the polygons that overlap the destruction polygon. This took double the time estimated.

We were also able to disable the physical bodies that are outside of the current “active area”. This results in smoother performance all over. Getting this to work took almost three times the estimated time. The biggest problem was the when bodies are deactivated, they are removed from Box2D’s dynamic tree, and therefore we won’t be able to do AABB queries on them to re-enable them. We tried some solutions like checking the bodies’ position rather than their AABB. With this we had to make each terrain body be positioned individually with their position in the center. This was not really a good solution and didn’t work properly. The final solution we got to was to have another b2DynamicTree with the AABB of all the disabled bodies. With this we could query both the active and inactive bodies. We also had a lot of problems with this solution. The biggest problem was that bodies were not removed from the inactive-tree when removed from the physics. The other problem was caused by bad synchronization handling leading to crashes.

Retrospective

We should not estimate work for the hours we are away, even though we think we will have some time available. We can always use the time to write in the report. Due to the fact that Asbjørn was away for a large portion of this sprint and was not able to do the required amount of work this week. We still feel that we made reasonable progress. Alot of performance issues have been fixed and the game now has better performance than the milestone 1 release.

We should double the estimation for issues requiring a lot of refactoring like the terrains split and body disabling.

Sprint 5

Goal

- Make the game challenging.

The Plan

- Let the player light the world with flares and wall lights.

- Challenge the player with flying melee enemies.
- Let the player continue from where they left of.

The Result

With this increment, the game is a lot more challenging.

All the lighting features were completed. The player have a flashlight on their shoulders, and can place wall lights and throw flares. All the light features were implemented faster than estimated, but with the flares, we added more features and created a system for equipment, so this took a while longer. While creating the flares, we changed how the bomb triggering works. This had to be done because of the change in the equipment system. Bombs are now triggered based on their distance from the player. Bombs far away are not triggered. Bombs closer are triggered faster. They also have a light that will shrink until detonation.

The player is now able to save their progress and continue after quitting the game. They can also start a completely new game. This took much longer than estimated. The biggest reason to this is that there was a lot of things that had to be done to properly implement saving and loading. For instance all the views controlling the AI's also had to be saved.

Weak flying monsters are spawned as the chunks of the world are created. They will attack and chase the player when they see him, harming and killing him when they hit him. AI introduced a whole bunch of performance related issues that were also dealt with. The game is now able to run at good framerates (>60fps) even with extremely many (>6000) AI entities spawned. We were unable to complete the more advanced AI tasks. The Mother type that will hatch the smaller ones was not implemented.

Retrospective

Although we were not nearly able to complete all the tasks that were set up for this sprint, we still managed to meet some of the criteria for the game aspects that were set as the goal. The sprint was slowed due to Asbjørn being sick and somewhat reduced in programming/development ability.

Having Jakob around doing art greatly increased the fun of developing the game. We implemented some assets that he created and were able to give direct feedback on the work he did.

Sprint 6

Goal

- Make the game a game, i.e. a goal and win/lose condition.

The Plan

- Make monsters that flee from the player and spawn melee flying monsters.
- Let the player reach a goal and win the game.
- Let the player collect dust from dead monsters and cave destructions.
- Let the player get a score when they win the game.
- Give the player the implosion device when they win the game.

- Make explosions and collisions apply damage to actors.

The Result

The AI has been made much smarter by doing a raycast when traveling the world. this allows them to search for a clear path away from the player when they see him. The mothers will flee from the player and start spawning children that attack the player.

We also added animations and proper sprites to most of the game, the player is now fully animated for all the basic movement. The animation works correctly when the player is moving both left and right, jumping, falling and using the jetpack. The player also wields and uses all the weapons that we have in the game currently.

As the player kills the enemies they will drop dust that the player is able to pick up.
Destroying the walls of the world will also generate dust.

There is now a goal in the game. At a certain depth, currently 1100 meters below the surface, the caves are much larger, and the player will encounter a bos fight. If the player survives for 80 seconds fighting off the monster, they will win the game. If they die, they will lose the game, and have to restart from the surface.

Adding the boss “biome”, with the large caves, was no trouble. We just made the destruction scale larger, and added more paths. This was easy by just parameterizing the TerrainGenerator.

Adding the boss fight was more trouble. It took us almost three days rather than one. We ended up adding the boss triggering logic to the GameScene, which is not ideal, but works. When the player reaches the depth, a series of spawners, that keep their distance to the player, are spawned in a circle around the player.

We were not able to implement actors being damaged by explosions and falling debris. The payer is not given a reward for finishing the game either.

Retrospective

The major focus point for this sprint was the gameplay elements themselves. We feel that we really achieved this. Although the gameplay is fairly limited and not balanced as well as we would like it to be, we feel that the result is pleasing. As we are getting closer to milestone 3, where we will focus more on the actual gameplay, balancing and other issues that are part of the development, we feel that the game is in a lot more testable form the way it is now.

Sprint 7

Goal

- Completing feature for milestone 2.

The Plan

- Add ranged flying enemies.
- Make flying enemies roam.
- Have an inventory where you can craft items and gems from dust.

The Result

For this sprint we managed to finish most of the tasks. Ranged enemies are flying around the caves attacking the player and fleeing if to close. When the enemies are not engaged with the player they will roam around the cavern system trying to find him. The roaming is done using a ray casting algorithm that checks for the best path to travel next at certain intervals and when the agent reaches its target. In most of the cases this works out really well. Some cases of the enemies finding the player if he is being idle for too long are helping setting the tone that the cave system is dangerous.

The inventory is fully implemented. All items are shown with their count, and the player can craft all types of equipment from dust, and craft gems from dust of the corresponding color. Proper assets are used for the inventory, rather than placeholders.

Implementing the inventory took almost as much time as estimated. No big problems were encountered. Placing the inventory items properly with the new assets was more problematic. We planned to have arrows from dust to items, but placing them properly was harder than expected, so we removed them and made some small changes to the inventory design.

We also started added some sound assets and the system for handling them to the engine. This adds a lot of immersion and fun to the gameplay.

Some extra bug and improvement tasks were planned as extra task that we would do if we completed the other tasks too quickly. These were not implemented as we didn't have the time.

Retrospective

From a planning perspective this sprint was very successful. We did not manage to finish all the issues, but all of the important were completed. When planning the sprint we made sure to think about how much time we would actually be able to put into development. The tasks where more realistically estimated and fit well for the time we had.

Milestone 2

Goal

- Basic (70-75%) gameplay complete:

    - **Shoot**: Players will be able to use weapons to damage the enemies he encounters in the world. This milestone will feature the all types of weapons. All [placeholder] assets for the current elements will be present; sound, effects, textures.
    - **AI enemies**: Enemies will spawn at random throughout the world, attacking players on sight. For this milestone we will focus on the flying enemies. Both the ranged and the melee enemies will be implemented and attack players on sight. Enemies will be killable and be able to kill the player presenting a danger towards the player as he travels the world.
    - **Resource pickup**: Dust will be present in the world, allowing the player to gather these. This stage will only include the red and yellow dust.
    - **GUI menus**: Main menu / pause menu with continue, exit, new game working.
    - **Inventory**: Players will have an in-game inventory at their disposal allowing them to see what items they have in their possession, and create gems from dust.
    - **Lights**: Players will be able to create and place lights in the world. Lights that are available are the flare that you can toss, and the permanent wall light.
    - **Game goal**: The game will feature a tangible goal for the players to reach, reaching this goal will allow the players to "win".
    - **Scoring**: The scoring system will be implemented where players can see their score compared to other players' score. Scores will be pushed to a remote database.
- Identify needed assets. (List for each actor/event in the game).

The Result

Some milestone goals were shifted between milestone 2 and milestone 3 halfway in. This was done to focus more on making it a "game", i.e. having a goal and a win/lose condition, as early as possible. The reason for this decision was that having a goal in the game would attract more players to test the game and have them more interested in actually playing the game. Without a goal they won't know what to do, and might therefore just quit the game after a few seconds. If they they have a known goal, they will have a reason to actually play the game. Even if the goal isn't very exciting, it helps a lot in making more players play the game.

The goals shifted to milestone 3 was adding biomes, plants, green dust, and the sonar. These are all more advanced features and fit well into that milestone. Goals shifted to milestone 2 were having a game goal and scoring system.

The player can now shoot with all the four types of weapons (handgun, shotgun, machine gun, sniper rifle), and they all have textures and sound.

Flying ranged and melee enemies are spawned around the world, and will roam around until they see a player and attack them. They are both complete with textures, except for the ranged enemy's tail, and some placeholder sounds are in place.

Red and yellow dust are spawned from world destruction and dead enemies respectively. The player picks these up when touching them. We planned to make dust be pulled towards the player by a force, but did not have the time to implement this.

The main menu is working properly with all functionality planned. Inventory menu is also fully implemented where you can see all your items and their count. You can also craft items and gems from dust.

Lamps, flares, and the player's flash light is working as planned.

As we were shifting more towards focusing on the gameplay aspects, we found that we no longer wanted to add the score functionality. A scoring system is something that doesn't match the exploration feel we're after for the game, so it is not something that will add much to the game. The scoring system is something that would fit the game in its current state, but not the vision of the complete game.

Retrospective

Milestone 2 was, as we expected, larger than what we had time for. We focused on the most important aspects of the development and managed to reach the main goal of the milestone: Having a playable gameplay demo of the game. Having the ability to move the plans around, and shift our focus towards what we find more important is great.

Sprint 8

Goal

- Small fixes.

The Plan

- Implement system for setting actors as inactive/active.
- Make dust be pulled toward the player.
- Render most lights without shadow casting.
- Add more sounds.
- Experiment with multisampling and other antialiasing techniques.

The Result

This sprint was much shorter than the others since we did a game jam on another game from tuesday to thursday. We lost a lot of time because of this, and therefore also planned a lot less than normal

Alot of time was spent researching various methods of antialiasing the pictures that were rendered. We had the most success using multisampling to achieve a smooth effect when rendering the lines of the program. This applies mostly to the light and the objects being rendered as pure colored triangles in the scene. Since we are rendering the light to a texture, a special technique was used, we first had to render the texture to a multisample buffer, then blit that buffer to the normal buffer that we then blended normally with the textured world.

Some effort was also put into using supersampling to render the textures smoother, this results in textures that are rendered are now much smoother. What we are doing is rendering the textures to a resolution that is much higher than the resolution of the screen.

The only features we got to implement was more sounds and dust attraction. Dust in the world will be pulled towards the player if they are close enough.

Some other enhancements was added. Lights that don't need to cast shadows, like those on dust and bombs, won't ray cast, and will therefore give a large performance boost. The actors have new inactive/active system, where they will be set as inactive when outside the viewable area. All the actor's components will get a callback when their owning actor is set to active or inactive.

Retrospective

Since this sprint was severely hampered by the game jam we had this week we were not able to schedule as many tasks. We completed all the tasks we planned, and we are fairly happy with the results. Since the game jam occupied a lot of time we are now slightly behind our schedule for this milestone.

Sprint 9

Goal

- Add nature to the world.

The Plan

- Add biomes with varying plantlife and terrain.
- Put texture on the cave walls and background

The Result

None of the issues planned were completed. We got about halfway done with both the main issues above, so we did absolutely not reach the goal. We also had two extra minor issues that we did not get to start on.

We did get to design how the biome system will work, and enhanced the terrain generator to support more parameters, like making curvy cave paths. The new curvy terrain is much more interesting and varying. The biome issue is about 30-40% complete.

For the task with the terrain we are 60-70% complete with what we wanted to do. Tiles are created when the view changes enough. What is left is to render the stencil and then the texture to the world.

Retrospective

During this sprint we were constantly disturbed by finishing the project we did last week. We are also currently in the process of establishing our company Suttung Digital AS. This required us to take

lots of time of Pyroeis to have meetings with the rest of the board members. We did estimate an appropriate amount of hours for what we expected, so the plan was not wrong.

Sprint 10

Goal

Make the world more interesting to explore.

The Plan

- Implement textures in the world
- Implement Plants.
- Implement new caves

The Result

Both ground and hanging plants are spawned in the world. They are not affected by heat or water levels. They spawn green dust when dead and can be killed like any other thing with guns and bombs.

To be able to spawn 3000-4000 plants the actor system had to be rewritten to spawn them on multiple threads. The spawning methods were also optimized to use less time. Because of this new system, it sometimes crashes. This is the reason it is not yet merged, and that we didn't get as far as we wanted. One whole day was used trying to fix the crashes.

The cave generation system has been improved a lot. It is much more parameterized. It can use midpoint displacement to make the caves curve. This makes the caves much more interesting. It is not yet affected by heat or water level. Because of the displacement, they can go outside the current chunk and over to other chunks where they do not apply the destructions. Because of this some paths may be blocked. This feature is also not merged yet.

The world is now textured using two different tiled textures. These are generated to always cover the screen. The tiles are updated only when it is required because the camera has moved enough . One of the tiled textured is rendered using a stencil that represents the world. This is to ensure that the texture is only rendered on top of the area that is actually covered by terrain.

We did not start work on the weapon customizations, but the weapons were tweaked.

Retrospective

A lot of the planned features were not completed. This was mostly because of bad estimation and other work.

The biome feature with plants and caves was poorly planned as we did not know exactly how it would act. We should have looked at this more before we started the sprint and split the task into smaller pieces. Making the plants and terrain be affected by heat and water could be two separate task. This would make our estimations more accurate.

Rendering the tiles caused the framerate of Magnus' computer to drop by a significant amount. This is probably cause by the blending that is being done by the renderer. Although the drop in frames is bad it is not at the point where we are hitting below 60 fps. (At least with the high performance gfx cards.)

Sprint 11

Goal

- Make the player want to explore the world by adding varying plants and terrain.
- Let the player customize their weapons.

The Plan

- Make the world have a heat and moisture value that changes with depth.
- Make plants spawn where heat and moisture requirements are met.
- Make terrain change with the heat and moisture.
- Let players upgrade and customize their weapons' power.

The Result

The world has both a heat and moisture value that change with the depth. Both are independent and ranges are defined for each in a json file.

Plants are randomly picked just as earlier, but only plants that can live at the spawn point (based on heat and moisture) are spawned. A cactus and dry bush was added to make the plant life vary more. More plants with different requirements must be added later to make the plant life even more interesting and unique, but the current set of plants satisfies this sprint's goal.

The terrain generation changes a lot of different parameters based on heat and moisture levels (roughness, cave size, number of caves, etc).

We were not able to finish the weapon customization branch. The weapons each have their own scene that will control the upgrading and powering of their stats. The remaining tasks are to make the scenes load the stats that will be tweaked for each weapon and create the gui for them. we need to call the functions that are necessary in order to tweak them in the callback for each button. The branch is at 70-80% completion.

We also did some testing for blurring the lights to create a softer shadow effect on the world. The result is that we are not completely happy with the new blurred shadows. The edges of the light that are close to the world get blurred into the darkness causing a unfocused effect that looks bad. Although the shadows are nice we would still like to polish them some more before we push them into the master branch. Ideally the lights would blur a small amount into the terrain itself but not vice versa.

We did reach the first goal, but the second goal wasn't successful.

Retrospective

As of the last few sprints we have been very busy with various tasks surrounding the founding of our company, Suttung Digital. Thus we have not been able to work as efficiently on Pyroeis lately. As this last sprint focused on some of the last important features that we feel are necessary for the game to become what we want it to be, we will still need to push the necessary into the polishing part of the development.

Milestone 3

Goal

- Advanced (90-95%) gameplay complete:
    - **Player knockout**: Players need to be even more careful when traveling around the world. Crashing with the jetpack, getting hit by enemies or falling will have the players at the mercy of physics, trying to gain control of their character. Falling or crashing at high speeds will also damage the player.
    - **Weapon upgrading**: The player can upgrade and configure all their weapons' properties.
    - **Advanced weapon functionality**: All the advanced features of the weapons will be implemented in the game. Players will be able to zoom with the sniper.
    - **Sonar**: Sonar functionality will be fully implemented.
    - **AI enemies**: Ground based melee and ranged enemies will be implemented and spawned in the world.
    - **Biomes**: Biome placement system will be functioning, creating different cave formations with different items. Jungle and desert caves will be implemented.
    - **Plants**: Different plants will spawn throughout the world.
    - **Green Dust:** Green dust can be found.
    - **Tutorial:** The player will be introduced to the game by a tutorial.
- Playable Windows version, i.e. smooth framerate and no serious crashes on Windows..
- Windows installer letting users easily try the game.
- OS X app letting user easily try the game.
- User testing system up and running.
    - Collecting statistics about how the game is played.
    - Users can type a message wherever in the game.
    - Messages and game data regarding that message stored in database.
    - Simple web interface for reading the messages and their data.
- Distributed the game to users for testing, and receiving feedback.

The Result

All the functionality for this milestone could not be completed, so some tasks were pushed out for later consideration. The largest reason to this was that almost one complete sprint (week) got lost because of a game jam that we did. This means we only had about ¾ time of the planned amount to complete the tasks. Some of the reason was also that too much was planned for one milestone.

Plants are spawned throughout the world and they drop green dust.

The biome system planned was changed to actually not be a biome system. Instead of defining biomes like “jungle” or “desert”, plants and terrain are dynamically parameterized with the world heat and moisture. These values have defined ranges for the depth, but will vary to the sides in a later version. Having the world dynamically parameterized makes it feel more realistic and creates infinite possibilities of unique experiences.

The player knockout feature was down prioritized as it is not an important feature compared to the other issues, and as such it was not completed for this Milestone. This feature is also something that is highly uncertain if it would add any actual fun factor to the game, it would require extensive testing.

The weapon upgrading feature is nearing completion (70%-80%) and will be prioritized and finished as soon as possible.

The advanced functionality for weapons will also be pushed to the polishing phase.

The sonar has not been implemented as we struggled more than we planned to make the terrain and world textured, which was necessary in order for the sonar to be implemented. The sonar has also been moved to the polishing phase but is not highly prioritized.

The AI ground enemies have been down prioritized as well. Due to the difficulty of implementing ground enemies and the little impact we predict this would have on the gameplay we have chosen to focus on the more essential elements of the game.

The windows version has been tested fairly thoroughly every week and as such we have not found many issues that are specific to this version over the others. the performance is around the same and there are no stability issues that we have noticed. Rather than waiting to properly test the Windows version until this milestone, we decided to test it underways make make sure it works well.

The OS X app has been available for the Mavericks version since the start of the project. It has been tested on various different Macs and has been running decently on almost everyone. We have had some issues with very Old mac (2006-2007) machines in the later builds.

As we have not been able to reach the part of the development that we wanted we have not been able to push out a system for testing the application in a wide scale. We have not tested the game using any system but we have released a number of different versions to the public and received some feedback. We would like to be able to do this in a larger scale soon.

Retrospective

This results of this milestone wasn’t as good as expected, but still the result represents a much more complete and entertaining game.

The reason to the bad result wasn’t planned too much, but that we got less time to work than estimated.

Sprint 12

Goal

- Finish remaining features and enhance plant spawning.

The Plan

- As a player I can customize weapon attributes with gems.
- Spawn plants on inactive terrain.
- Make plants fall when ground disappears.
- Apply destructions on intersecting chunks.

The Result

The weapon customization is done, Players are now able to spend their resources to permanently increase the power of their weapons. The UI for the weapon customization is set up using a separate view for each of the weapons the player has. The views now have a new type of hover action, the tooltip. Views can have a view assigned to them as a tooltip, to make the adjustment of the tooltips size and position scale nicely with the view itself, we have decided that the tooltips size and position are specified as a percentage and an offset of the views size. Tooltips are very handy for displaying information to the player while keeping the user interface clean.

We also added health to the rocks/debris in the game, this removes some of the problems we have been having with the players getting stuck while traversing the world using explosives.

Earlier plants would not spawn on inactive terrain, resulting lots of missing plants. This now works properly.

Destructions intersecting other chunks (from the one generated) are now applied to these as well. It is currently in review since it is waiting for the feature where plants fall when having no ground, but it is otherwise complete.

Detaching plants when the ground is gone was not completed. It took much longer than estimated (3d instead of 2h) and it is still not complete. Most problems has to do with the deactivation and activation of bodies, and changes in body shapes.

Retrospective

This sprint went fairly okay. We worked at a steady pace and completed most of the tasks we set. We hit some major issues with the plant spawning branch that we have not been able to figure out.

Milestone 4

Goal

- Tweak game balancing against user experience.
- Crash-free game.
- Adjust to meet requirements. (Optimizing, documenting, etc.)

The Result

For our final Milestone we focused on the issues that were the most important to get done for this version of Pyroeis. The planned focus was to remove bugs and enhance the features that would make it as playable as possible.

Because of some issues that were not completed before previous milestones, we couldn't only focus on removing bugs and tweaking the game experience, we had to implement the missing features (like weapon customization) since they are part of the core functionality. These missing features were completed.

Some important bugs were fixed like the saving functionality no longer working, and that some actor states were not properly saved.

Plants can now spawn on inactive terrain, which results in that all plants are in place.

Destructions applied outside a chunk is now applied properly to all intersecting chunks. This fix resulted in another bug where some plants hover. This issue was prioritized, but because of the unexpected complexity it was not finished.

Other minor bugs like the flare's light fading being wrong was fixed.

Even though a lot of bugs were fixed, there are still some, but most of these are minor. The game still crashes a few times, but this happens very seldom and most crashes have to do with the event manager which is not safe in many ways yet. We still struggle making the event manager completely safe.

One new not planned feature was added: Healing.

Game balance tweaking was not done since we did not finish the feedback system previously and had more pressing issues that had to be completed.

We were also able to fix some issues that have affected the mac specific versions of the game. The game should now run on all macs that have support for OpenGL 3.0 and C++11. This bug was due to Apples new SDK 10.9 in OS X 10.9 Mavericks not being backwards compatible with the previous versions of OS X.

Another Issue that was fixed was hardware specific issues for GPUs. OpenGL 3.0 does not require the implementation to support pure stencil buffers. since we were using those the framebuffers

could not be created for certain GPUs, In our case we had trouble with a couple of 2009 MacBook Pros. The issue was solved by using the correct format, which is a combined stencil and depth buffer.

Although we did not finish all the issues planned for the development phase we are happy with the result and are left with a game and an engine that should run on most platforms fairly well.

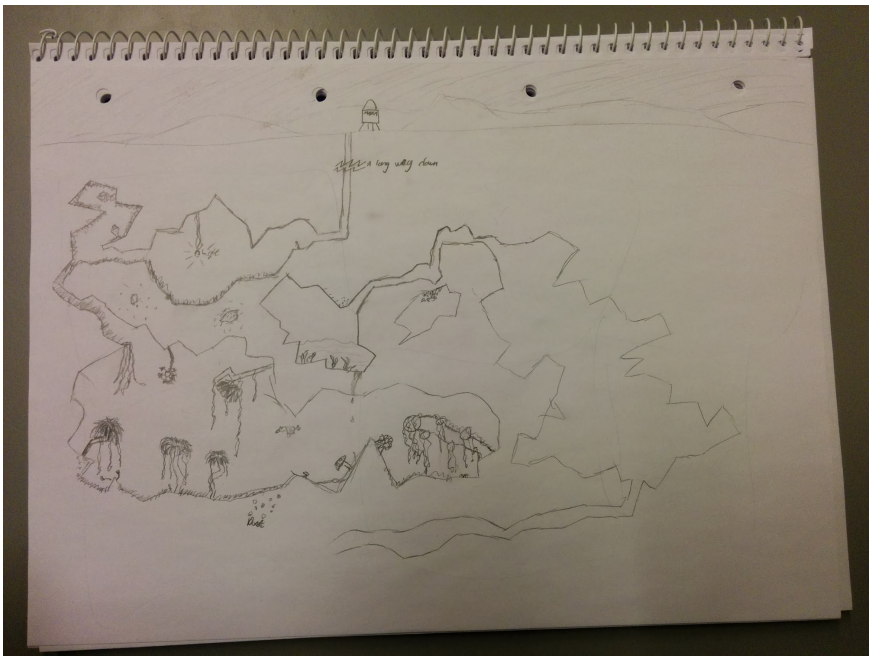
Retrospective

During this final Milestone we have been less strict in the development process. As the easter break cut the milestone in half and severely limited the time we had available for pure development, we were left with a much more patchy workflow than what we have usually had. We have also had a lot of other tasks that needed to be done during this milestone, for our company, Suttung Digital. We have sent applications for funding to both NFI and Nordic Game Program, and this has taken some full days and nights out of the schedule for our bachelors. We have also started creating the application needed to get funding from Innovasjon Norge. Still having issues from earlier phases gave us less time to focus on bugs and user experience.

L Pyroeis: Work Log

Work log for Pyroeis.

Monday 06/01

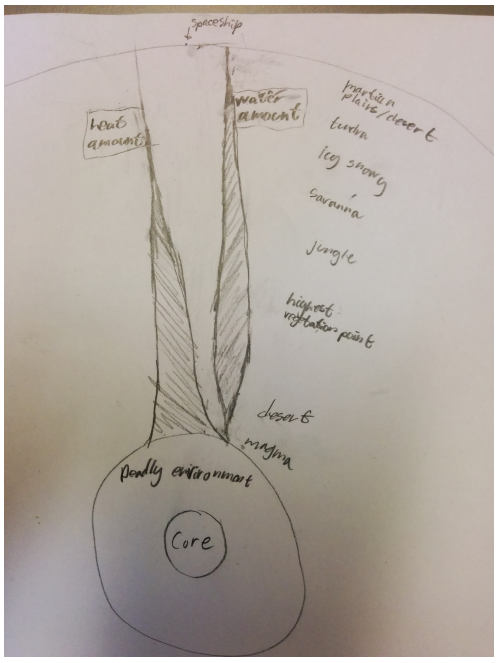


The first day of the bachelor project. We discussed what should be done before starting the development, wrote group rules, and started working on the project plan. In the project plan we wrote the goals of the project and the participants with their responsibilities. Some research into scrum planning were done. We also wrote a lot about the game in our game design document which will be included with the project plan. A small sketch was drawn to get a feel of how the caves would look and feel.

Sources:

- GDD: http://www.stemchallenge.org/students/game-design-documents/
- GDD: http://www.cs.tufts.edu/comp/150CIS/AnAntsLife/AnAntsLife-GameDesignDocument.pdf

Tuesday 07/01



We finished most of the design document with a lot of work done on the weapons and their augmentations. We also explored how the world generation should act and what biomes should be placed where. The drawing to the left shows the basis for the biome system.

We research how to properly implement and use the scrum methodology and modified the project plan by adding a section about the development process, where we decided on how to implement the scrum framework.

Simon gave us access to Atlassian's Confluence where we explored some of the possibilities. We are still waiting for access to the jira software.

Research was done on the various ways of using Box2D to handle player movement and collision, ensuring that the player can move about freely while the world is generated randomly might prove to be an issue.

Some work was put into fixing issues with the lighting in the engine, The issue is that the current lighting algorithm is currently highly dependent on certain factors being true for the polygons that are created when generating the world. Fixing this would allow greater flexibility when implementing the randomly generated caverns. This issue is still in progress.

Sources:

- Scrum: https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/Scrum-Guide.pdf#zoom=100
- Box2D: http://www.iforce2d.net/b2dtut/

Wednesday 08/01

Issues with the lighting are now resolved. The fix ended up being simply casting the ray some angle of the target if the target was a min or a max vertex. This resolves the issue concerning the world needing to be generated with a specific type of polygons. the new algorithm should be able to handle all cases.

Merged the UI system into master in the repository for Project Nox, this readies the engine for a UI system.

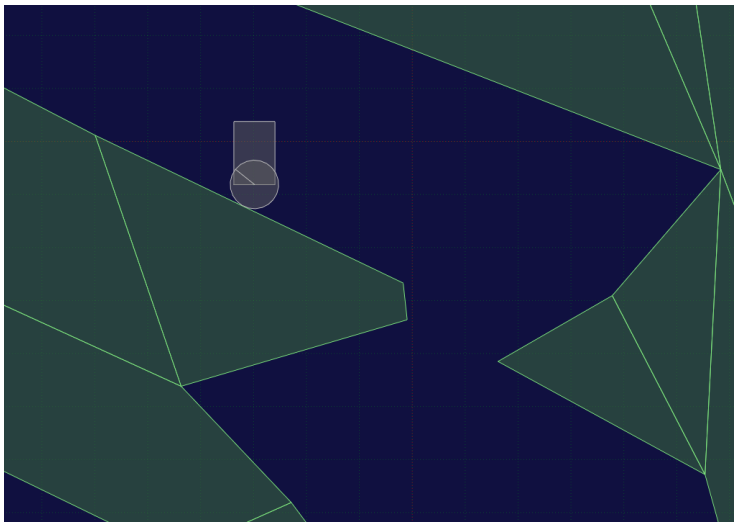
Wrote more in the project plan. All that is missing now is the time and resource plan. Started developing in Planner.

Thursday 09/01

More work was put into the design document, Highlighting some of the items that the player will be able to use, like the light sources and the bombs. A high level flow chart was made for the game's flow.

We also had a meeting with Jayson where we discussed all elements of the project, and especially the destruction and random generation of the terrain. He also advised us to implement a simple system for users to report bugs to us and then deploy this to installers that are sent out to testers, this would allow us to get easy feedback from the users and use this to refine the game.

Research was done towards player movement, The following test scene was developed in R.U.B.E



Here you can see the player as the grey box with the “wheel”, the wheel will act as the players feet, ensuring that we can have different friction on different materials when moving the player. Using this method also enables us to take advantage of momentum on the character, thus giving us the ability to have more interesting movement. This was tested in the R.U.B.E simulator that enables us to quickly test different setting on the friction etc.

Some time was spent on researching methods for decomposing non convex polygons as this is needed for the physics engine to cope with the world. This is something that will be used when generating the random world. We found three different solutions to the problem: Bayazit, Poly2Tri and the b2Separator class By Antoan Angelov, we opted for the last.

The gantt diagram was completed and some time was put into researching how to write good user stories.

Friday 10/01

- A revised contract was made for use with the bachelors project.
- Revised project plan and gantt chart to include some of Jayson’s suggestions.
- Meeting held with Ko-Aks regarding contract.
- Researched making installers on Mac.
- Revised GDD to better specify how the player’s items work.
- Wrote some non-functional requirements.
- Created user stories for the product backlog with all tasks planned for milestone 1 and some for milestone 2.
- Planned sprint starting now (estimated hours, added to sprint backlog).
- Still waiting for jira access…

Saturday 11/01

Asbjørn

Researched creating executables for Mac OSX. We are now able to create .app files that contain all the frameworks, dylibs and assets that are needed for the app to be run on any

mac that meets the requirements. This also eases the development process for mac as we no longer need to handle the assests manually every time they change.

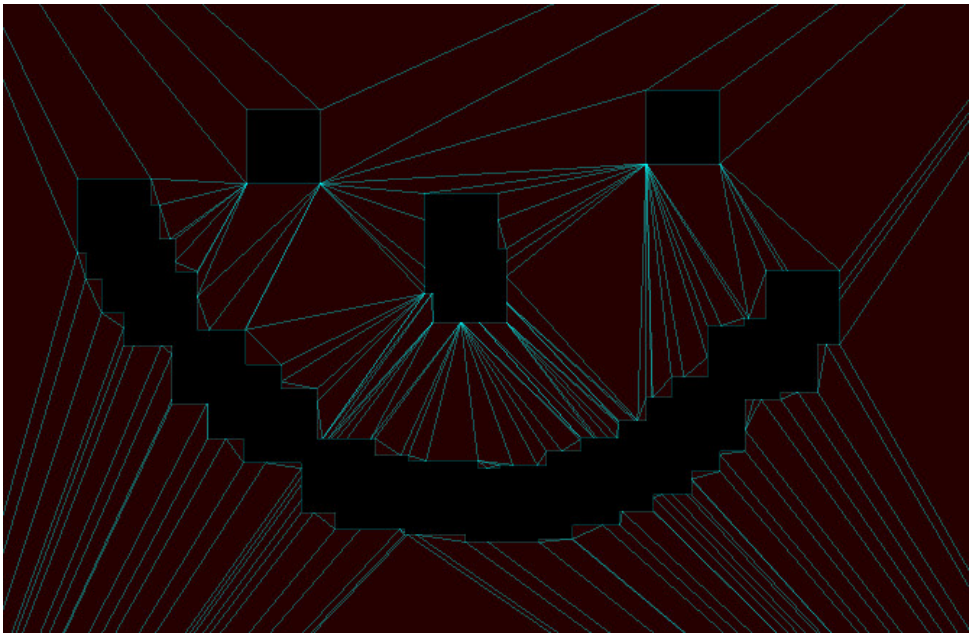
Magnus
Installing and learning JIRA

Sunday 12/01

Asbjørn
Put lot's of time and effort into researching post-build scriping, this eases the development process further by allowing certain scripts to be run for the created .app file on mac, making it even easier to create files that work for all apple computers. Also set up the base level for testing the player movement, this makes it easy to test the player in some set, cavelike scenarios

Magnus
- Installing and learning JIRA.
- Researching user stories and scrum.

Monday 13/01



Magnus
- Designed solution for destructible environment.
- Lots of ideas from http://www.emanueleferonato.com/2013/10/17/how-to-create-destructible-terrain-using-box2d-step-2/
- Partially implemented destructible environment (sometimes buggy when clipping). Two cases: completely inside fixture (boost::geometry::within) and overlapping an edge. Poly2tri is used for the first case (create a hole), while b2dSeparator is used for

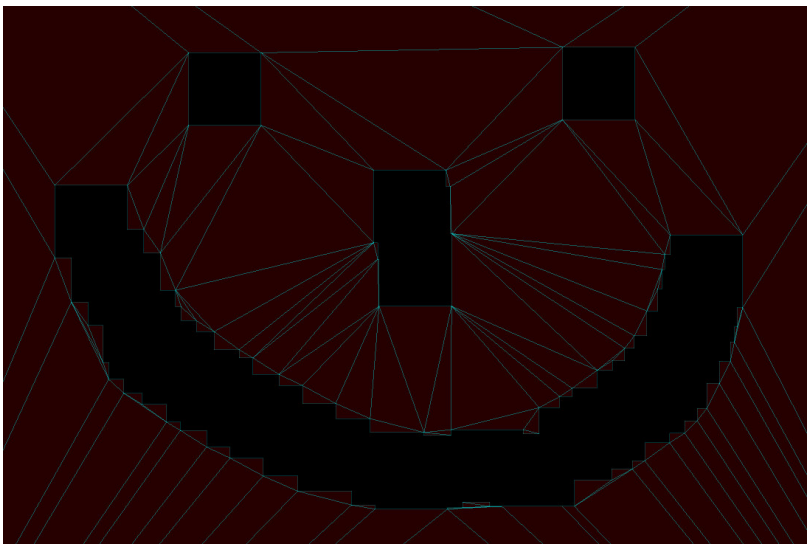
the second (not quietly working yet). Lots of horrible converting between b2d, p2t, b2dSeparator, and boost :(

- Destruction working everywhere (though kinda buggy). Using p2t instead of b2dSeparator.

Asbjørn

Implementing player movement, in particular the system for handling the feet of the player “WheelComponent” have tested this in the test level, it seems to work fairly well, also working on making the player only able to run and jump while in the air. Some of the codebase needs to be changed to accomodate this, (actors need to have more than one body etc.

Tuesday 14/01



Asbjørn

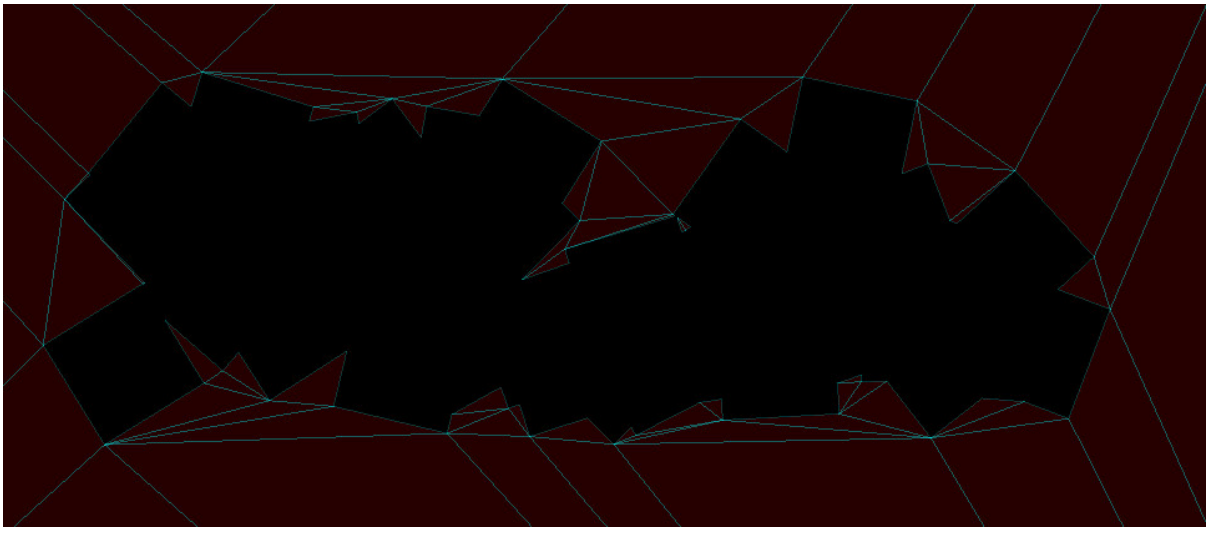
Risk: Chose to use system_clock for timing between jumps, is this a risk?
The player is now able to jump and run around, and only when on the ground.
Designed and implemented JetpackComponent, the player will be able to controll the jetpack soon.

Magnus

Worked on making the destruction free of crashes. Ended up changing over to letting the terrain be represented in boost::geometry and when a destruction happens, the original terrain part is removed from the set, and difference from destruction is calculated and added to the terrain set, then it is split by poly2tri and broken down into fixtures.

Lots of problems with boost::geometry::difference. Used half the day on this. Apparently boost wants the vertices clockwise by default while I used them counterclockwise. A template argument had to be set to false to use counterclockwise. Also all polygons had to be closed (last vertex == first).

Wednesday 15/01



Met with Mariusz (see meeting log).

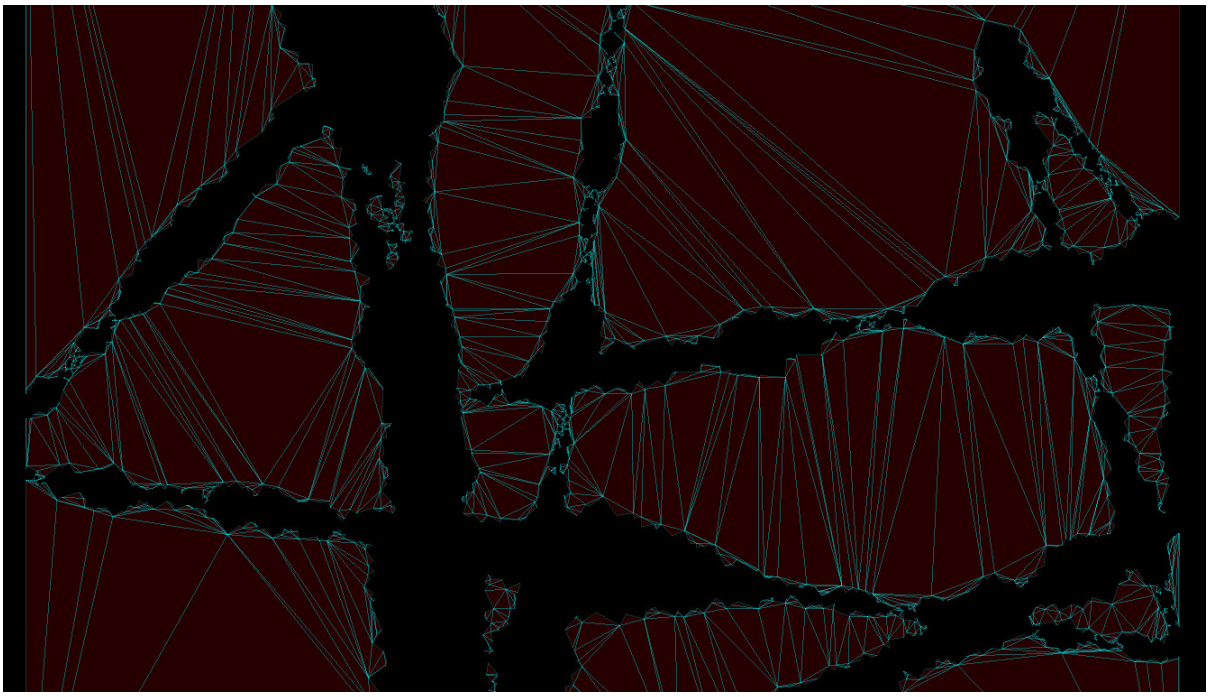
Magnus

Finished work on destructible terrain, and started working on random generation of world. Currently only does a few explosions along a path in a random direction.
Finished reviewing and fixing player movement and merged with master.

Asbjørn

The jetpack component is now finished, the player is able to fly around the world, controlling thrust and the tilt of the jetpack. I've tried to focus on keeping the gameplay of the jetpack difficult and rewarding to master.

Thursday 16/01



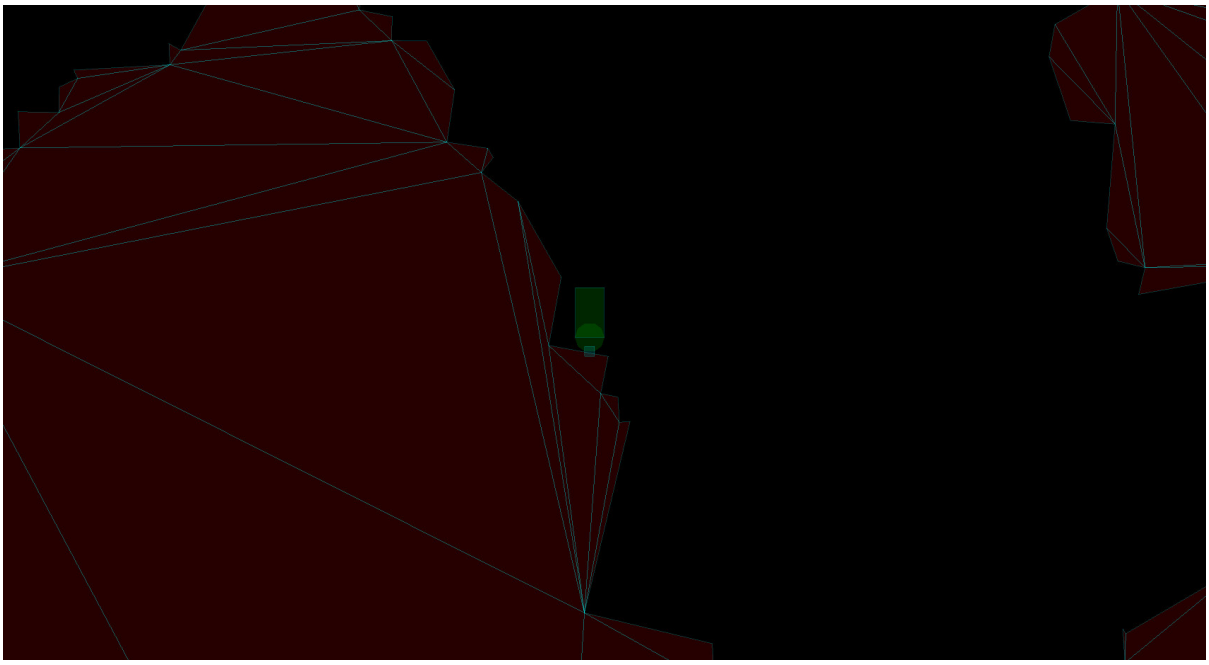
Magnus

- Worked on the random generation of the terrain.
- The terrain will now generate one segment with caves connected to other segmnets when these are implemented.
- Used a lot of time with a destruction translation bug which was my fault. glm::translate and friends should not multiply the matrix since they already do that. This resulted in double multiplying.
- Used a lot of time with cave digging bugs that made it super slow, spawning segmentation faults, and sometimes running infinitely.
- All these bugs are resolved now.
- Cave generation was not consistent between Mac and Linux. Probably because of the std::hash<std::string> function object giving different results.
- Reviewed jetpack.

Asbjørn

Some adjustments and “bug fixes” were made to the jetpack controls, this resulted in a easier to control, and more fun jetpack. Also made the change that the player can no longer jump when the jetpack is equipped. I also spent a lot of time enhancing and reviewing the engine code. Added the possibility to have command line arguments that affect the debugging levels of the app.

Friday 17/01



Asbjørn

The first part of the day was spent in researching how to build for OS X 10.8, I have made some progress in figuring out what the actual issue is. Solving the issue would require me to edit the path that the libvorbisfile uses to check for it's dependencies. I am currently unable to do this because of some bug or something similar with the command-line tools, The rest of the day was spent documenting parts of the previous code and planning the next sprint.

Magnus

- Meeting with Mariusz.
- Documenting and checking code. This took longer than anticipated so we couldn't do this as throughout as we wanted.
- Merged master into the world generation so that we can move around in a generated world.
- Tested build on Windows and fixed issues.

Monday 20/01

Asbjørn

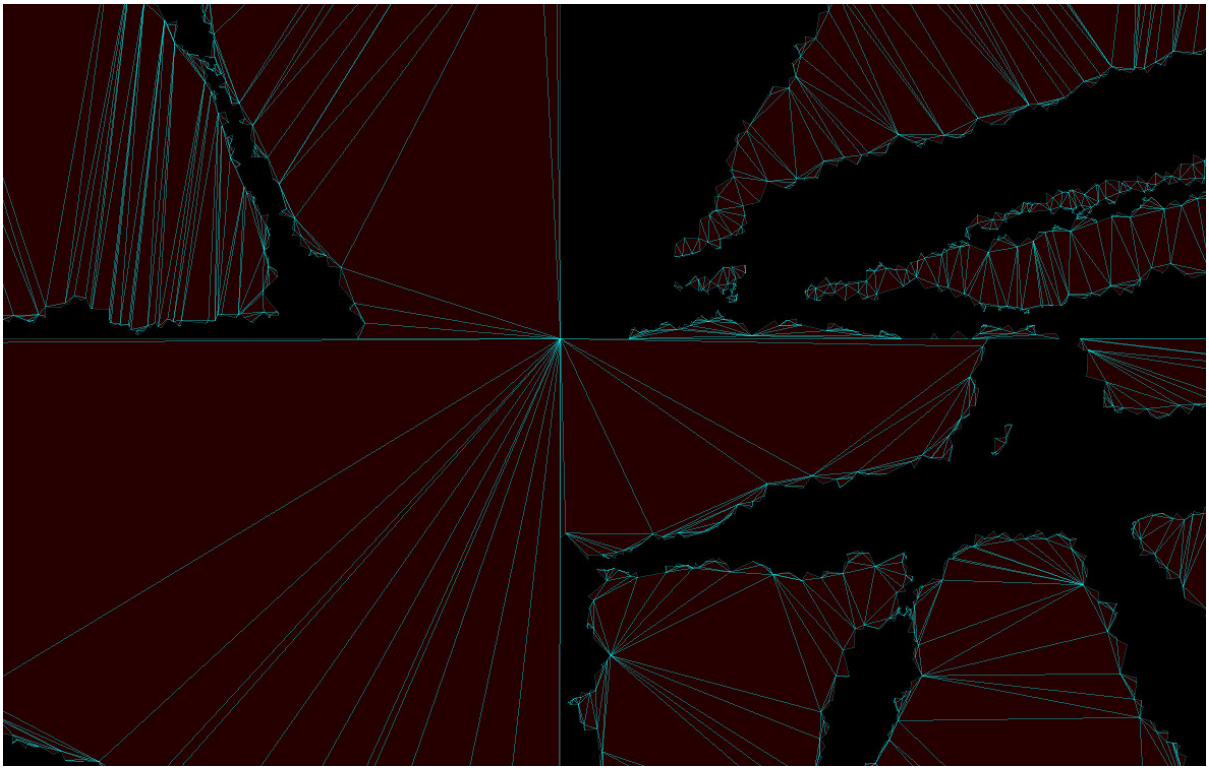
Most of the day was spent implementing the keyboard feature, allowing the player to customize the layout of his or her keyboard. We then took some time of the day to have a meeting with the rest of the guys at Suttung Digital regarding future plans.

Magnus

- Started work on infinite caves.
- Realized that if the terrain should be decoupled from the physics, the physics interface would need new functionality. So work was started on this and completed.

- Began merging physics interface changes into world generation. Had a problem that Box2D asserted on the polygons.

Tuesday 21/01



Asbjørn

Today I finished up the keyboard implementation and did a good chunk of work on the weapon firing. The player is now able to fire and aim weapon creating bullets that fly into the world, damage and other stuff is still missing. I also tested out the new sprite created by Jakob, it looks great.

Magnus

- Fixed Box2D assertion on polygons. Some problem with either my or Box2D's area calculation. Solution was to copy the Box2D area calculation from the centroid function.
- Made the world an actor.
- Generate chunks infinitely (only those in view).
- Reviewed controller mapping feature.

Wednesday 22/01

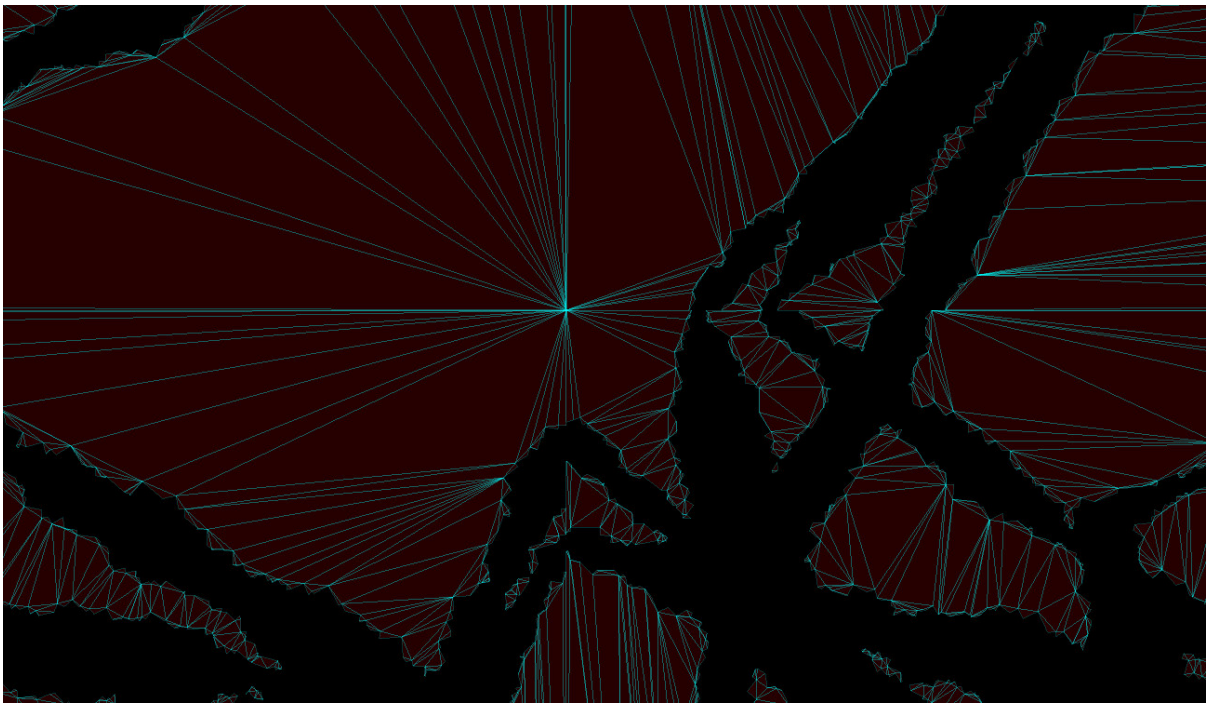
Asbjørn

The first part of the day was focused on doing the risk analysis for the project plan, this is nearly done, but still needs the risk mitigation and contingency plans for it to be ready. The rest of the day was spent researching and implementing the bullet response mechanism. The bullet firing mechanism is something that has taken a large amount of time at this stage.

Magnus

- Optimized geometry set so that removal of geometry would not kill the fps. Some other optimization changes as well.
- Merged the optimization changes with the world generation. Now unloading chunks works properly without any noticeable fps drop.
- Fixed problems regarding when to unload and load chunks.

Thursday 23/01



Asbjoern

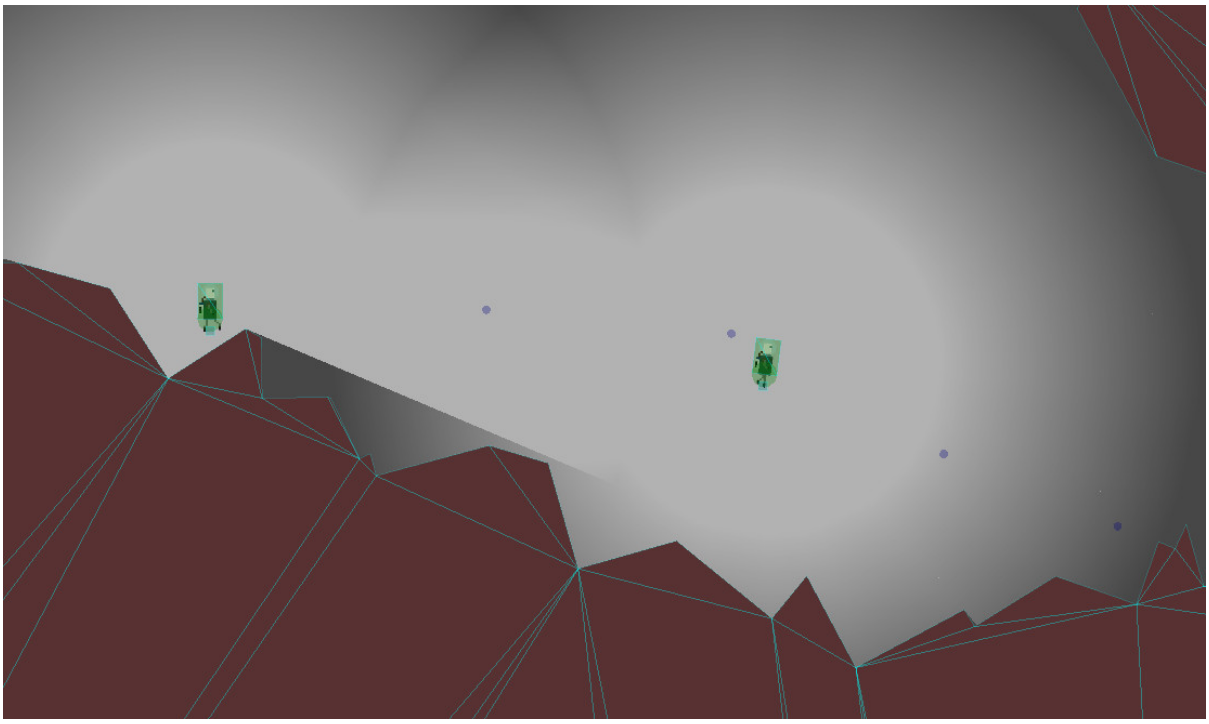
Today I finished up most of the gun firing task, the player is able to fire in any direction, the bullets will hit the objects and affect them but will then disappear. I also finished up the risk analysis and started listing all the assets that are going to be needed for the project.

Magnus

- Refactored all the world generation code with better naming and split it into more separate classes.
- Made chunk entrances match between chunks so no caves would abruptly stop (image above shows four chunks connected).
- Converted to using radians instead of degrees everywhere.
- Created a custom uniform int and real distribution that produces the same result on every platform. This results in the caves being equal on all platforms with the same seed.

Friday 24/01

Goof time



For a special first time ever we goofed around with network play for Pyroeis, this worked fairly well and was a lot of fun. We will probably try to do this every friday just for fun. One problem with this was that the collision event was not sent over the network causing the “shots” to stay suspended in air on the client.

Asbjoern

- Reviewed and Merged the world generator
- Meeting with Mariusz regarding progress
- Fixed a bunch of bugs and errors caused by both the world generator and the handgun branches.
- Sprint Review
- Planning Sprint

Magnus

- Fixed bugs in handgun branch caused by the world generation (multithreading and geometry changes).
- Met with Mariusz.
- Documented world generation clases.
- Reviewed and merged handgun branch.
- Reviewing sprint.
- Planning sprint.

Saturday 25/01

Revising plan and gdd.

Asbjoern

Today I started researching and looking at how to implement a better and more suitable lighting engine. The new proposal for how I think the engine should work is the following. Note that the static lighting is not necessarily static. The static is only in reference to the terrain it will take into account.

Static

- We should have one lighting algorithm for the static parts of the world, this means the static terrain that is generated using the boost geometry libraries.
- The static lighting only needs to be updated if the geometry near a light, and/or the light changes. This would save a lot of time on the cpu, as the light could be cached to memory and redrawn every frame it is not used.
- We should only use the boos geometry information when rendering this lighting, this greatly simplyfies the creation of an algorithm allowing us to very easily adapt our current method to this, even increasing performance.
- This allows us to easily track what edges have been seen by the player. Allowing us to display this information to the player.
- see http://www.redblobgames.com/articles/visibility/ for a overview of the method to be used here.

Dynamic

- Another algorithm would be needed for the dynamic objects. The lighting used here would be primarily focused on adding shadows back to the scene, from the dynamic objects.
- The algorithm used for this still needs to be figured out.
- Something like this could be useful http://www.gamedev.net/page/reference/index.html/_/technical/graphics-programming-and-theory/dynamic-2d-soft-shadows-r2032

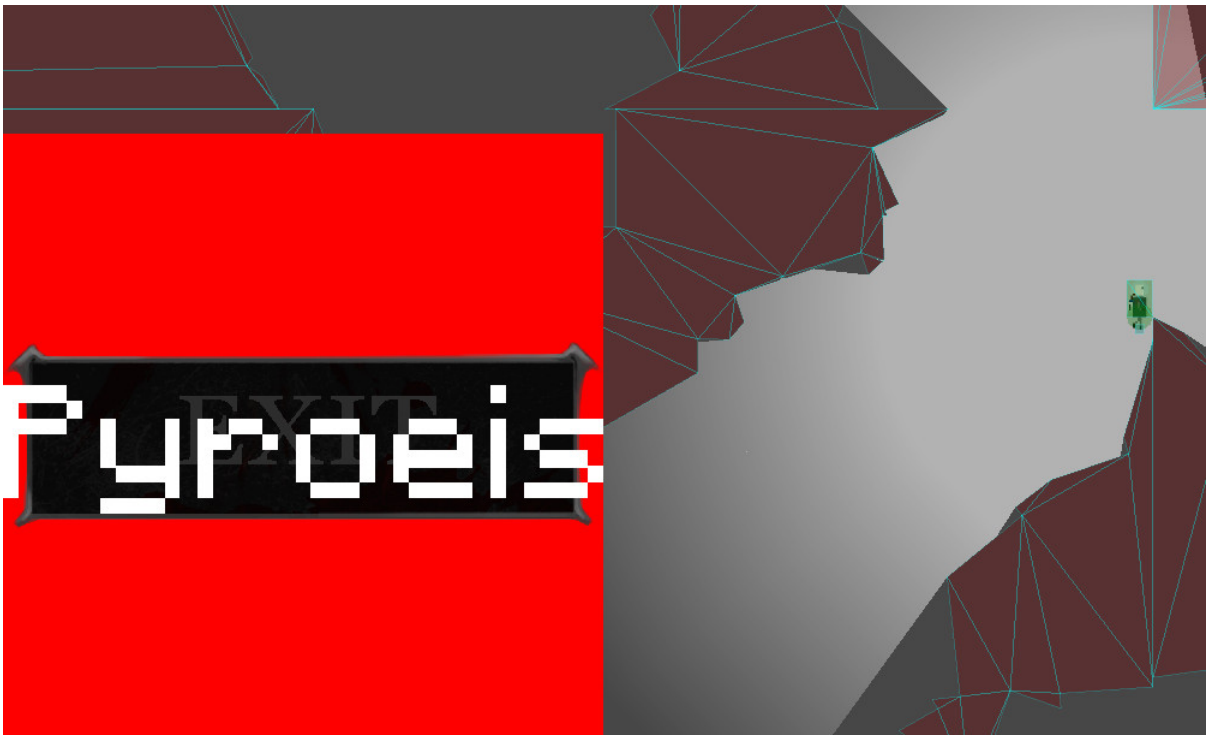
On another note: worldSeed -182818391 is good

Sunday 26/01

Magnus

Revised plan documents and handed them in.

Monday 27/01



(label over image, over basic view)

Magnus

- Implemented image view.
- Implemented button functionality.
- Refactored view class.
- Implemented text rendering and label.

Asbjoern

- Fixed Bug where jetpack would act very sluggish when the frame rate was low. PYRO-39.
- The player is now able to throw bombs in the environment, they will stick to the walls and can be detonated by the player by pressing the middle mouse button, The destruction is very slow at the moment.

Tuesday 28/01

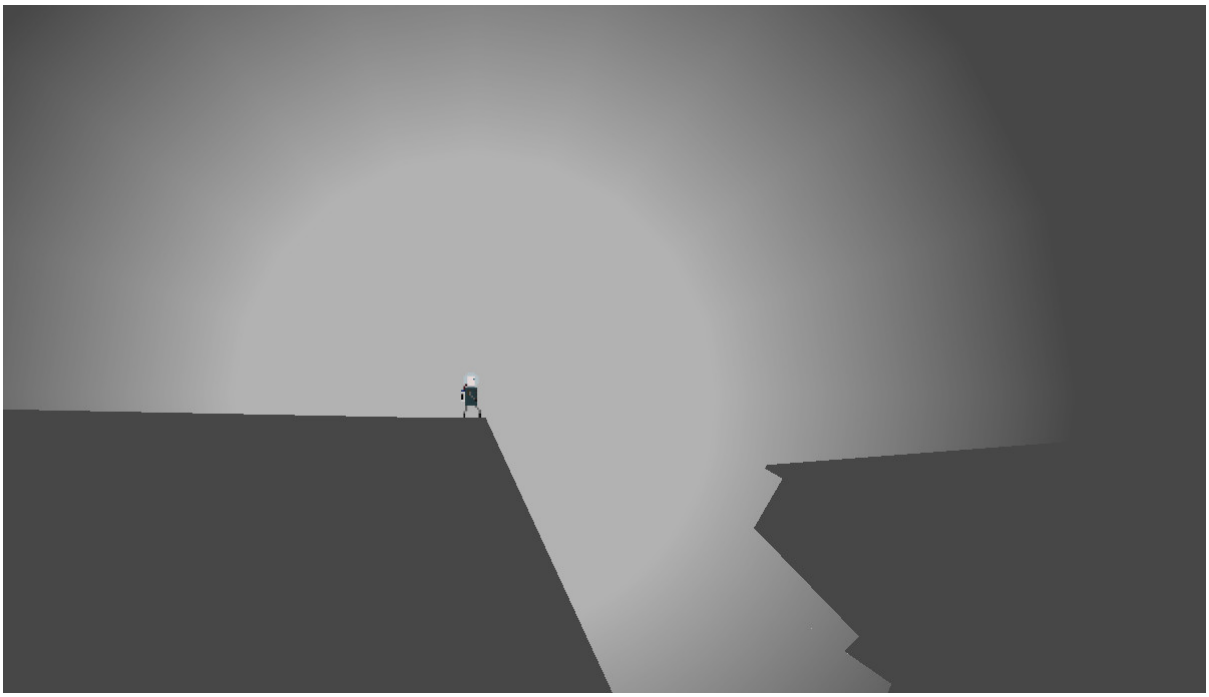
Magnus

- Finished view documentation and cleanup. All view classes are ready to be used, and reviewed.
- Reviewing of bomb feature. Some name changes, and changing how the stickiness value works.
- Helping with debris spawning.
- Wrote som system change proposals.

Asbjoern

- Debris is now working, The performance is not vert good but the player is able to destroy the world.
- They will also spawn debris based on the ammount of destruction.
- Reviewed Parts of UI class

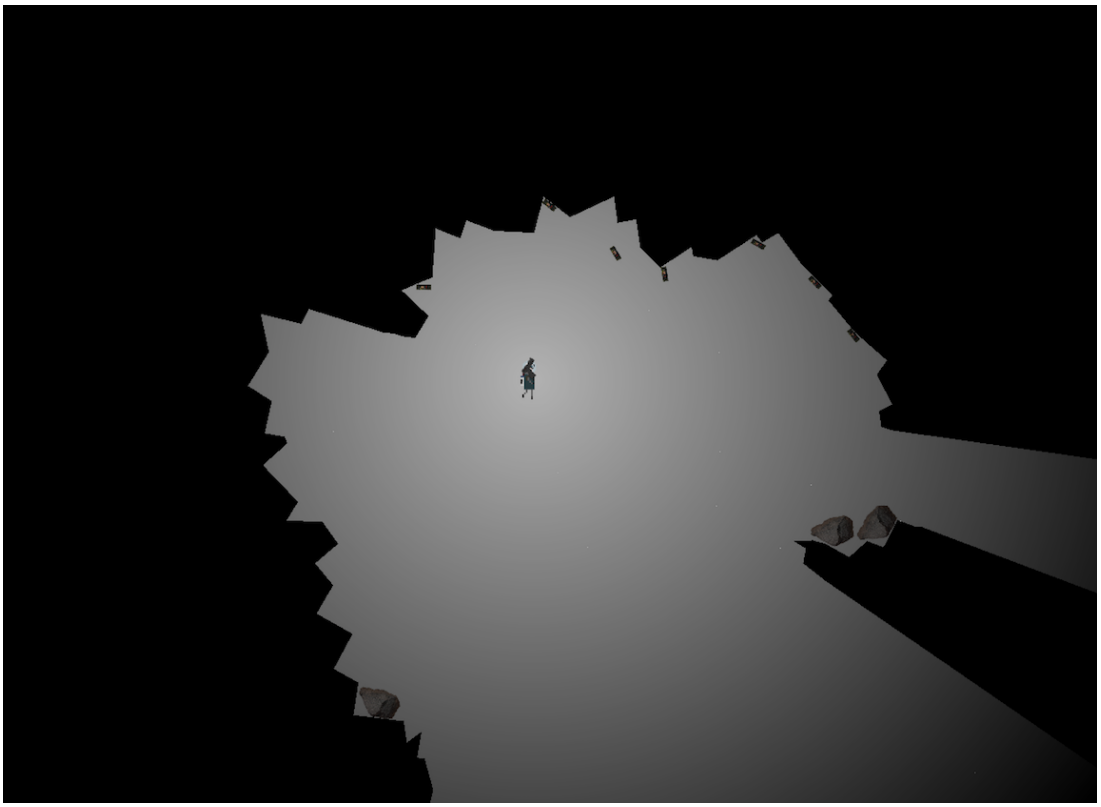
Wednesday 29/01

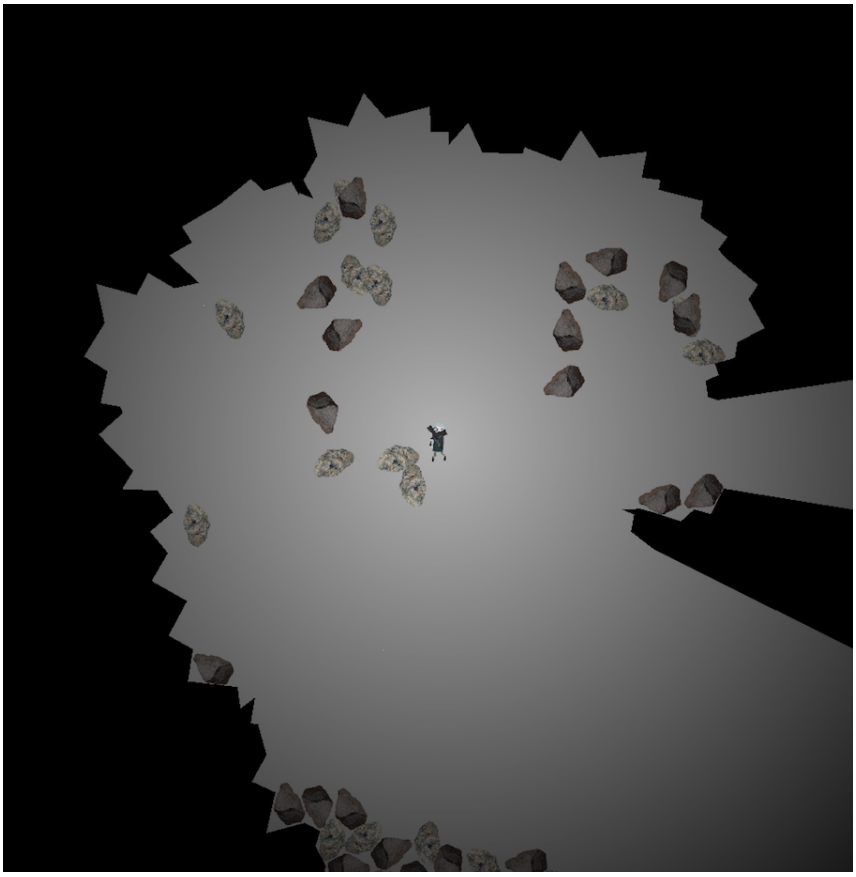


(the surface)

Asbojern

- Reviewed GUI system
- Tweaked and fixed the Debris system,
- Started adding placeholder assets to the game.
- added different types of debris that will be created by explosions.





(player exploring and destroying environment, debris is spawned from destruction.)

Magnus

- Started and finished story that the player should spawn above ground.
- Fixed an issue where physics debug object would be rendered at the reu-used polygon's position once.
- Optimized debug rendering, and physics, by not splitting convex polygons, and by not splitting triangle polygons, and splitting rectangle polygons only once.
- Made event manager thread safe (hopefully).

Thursday 30/01

Asbjoern

Isolated the issue we have had with the lighting on the randomly generated world. The issue seems to be isolated to the fixtures that are a lot longer than normal. This probably has something to do with the current way we figure out what vertex is the max and the min of the fixture.

Did some performance testing, specifically on the destruction method used in the DestructibleTerrain class. Some interesting finds are that the intersects function used to check if the polygon even should be tested actually uses a lot of time through the boost::geometry::detail::distjoint::distjoint_linear function. Removing this test will make that specific function over 30% faster. We will not remove this test for now, this will be done once the actual chunks that are used by the physics are reduced/split into a set of smaller ones.

Magnus

- Reviewed the debris branch, and merged it. With this, all sprint tasks are complete.
- Set up project wordpress website with description about the project.
- Created a new thesis front page based on this (the basic GUC one was horrible): http://www.latextemplates.com/template/university-assignment-title-page
- Read some old bachelor thesis to get an overview of how to write ours.
- Began writing about the world generation and destruction.
- Helped Asbjørn out with scaling debris randomly.
- Fixed the bomb detonate crash (PYRO-45).
- Started work on splitting chunks into bodies.
- Looked at texture packing algorithms. This one seems ok: https://github.com/lukaszdk/texture-atlas-generator

Friday 31/01

We reviewed both the sprint and milestone. Then we planned the next sprint. We also tested the networked play again, which didn't work perfectly with the new features.

Asbjoern

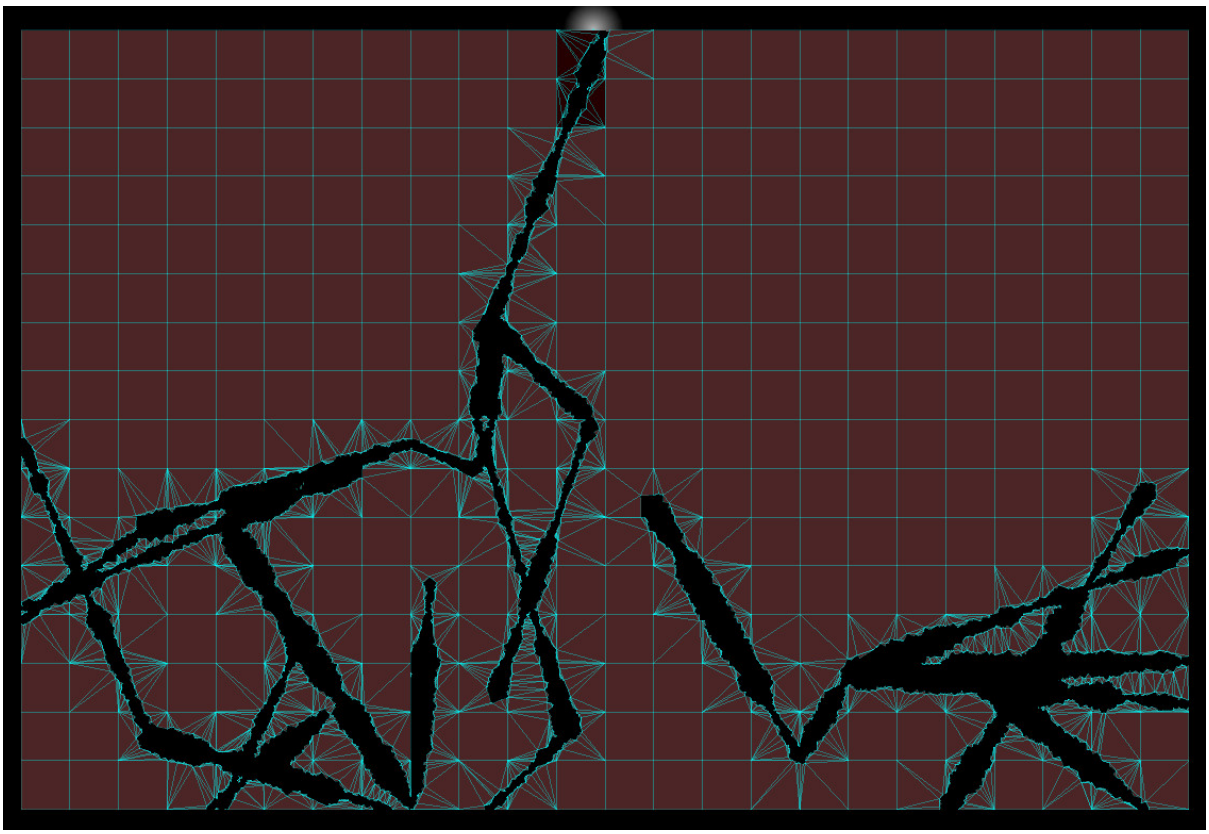
As this was the last day of the milestone and the sprint a lot of work was put into polishing the game elements for the first milestone release. This work continued over the weekend, a

video was created and uploaded to youtube, as well as windows, mac and linux builds of the game.

Magnus

- Reviewed the assets branch, and merged it.
- Wrote more about the world generation.

Monday 03/01



Magnus

- Meeting with Mariusz.
- Finished most of the splitting chunks into bodies. Some minor refactoring and documentation needed.
- Fixed parsing of json files (stupid constructing of string from char* without \0).

Asbjoern

- Setting up youtube channel and tweaking it.
- Creating placeholder assets for use until we are able to find the ones that we actually want to use.
- The shotgun was implemented in it’s basic form.
- Reworked entire weapon system. Players are now able to switch between weapons using the 1, 2, 3, etc. keys. Machinegun was also implemented. this was very easy due to the work put into the weaponsystem. The change did introduce a bug in that there was no way of creating the bullets while the game was updating the actors, this caused weird behaviour.

- We had a long meeting with Mariusz discussing progress and reviewed some of the milestone we had just completed.

Tuesday 04/01

Magnus

- Finished splitting chunks into bodies (documentation and refactoring).
- Worked on implementing scene system.
- Used a lot of time moving HumanView logic over to GameScene.
- Reviewed the shotgun and smg features. Fixed some minor bugs and refactored some things.
- Helped out with optimizations and crash fixing in terrain body split branch.
- Made CMake enable LTO on gcc if available.

Asbjoern

- The bug in the weapon system has now been fixed, the main change was that when adding actors they are simply added to a queue in the logics, when the system is ready for them they will be added to the update and rendering loop.
- I also fixed the problem with the bullets and bombs not conserving the momentum of the player when thrown. they are now spawned with the same speed as the player, making the gameplay much more natural.
- I also spent a lot of time optimizing and fixing various performance issues. The performance gain by inlining a function that was being called very often was immense. The main bulk of the performance testing was done to ensure that the new improvement in splitting the large chunks into smaller bodies was working correctly and not introducing too many bugs.
- Some work was also put into the reticle system for the sniper, but I was to tired to get anything done in the end.

Wednesday 05/02

Asbjoern

Most of the day was spent making the Reticle system. The weapon system is shaping up to be really nice and very easy to modify using the json files. I also did some performance

testing using various methods for timing the application. There i currently a bug causing the Reticle to sometimes jitter on the screen. this is really ugly and disturbing.

Magnus

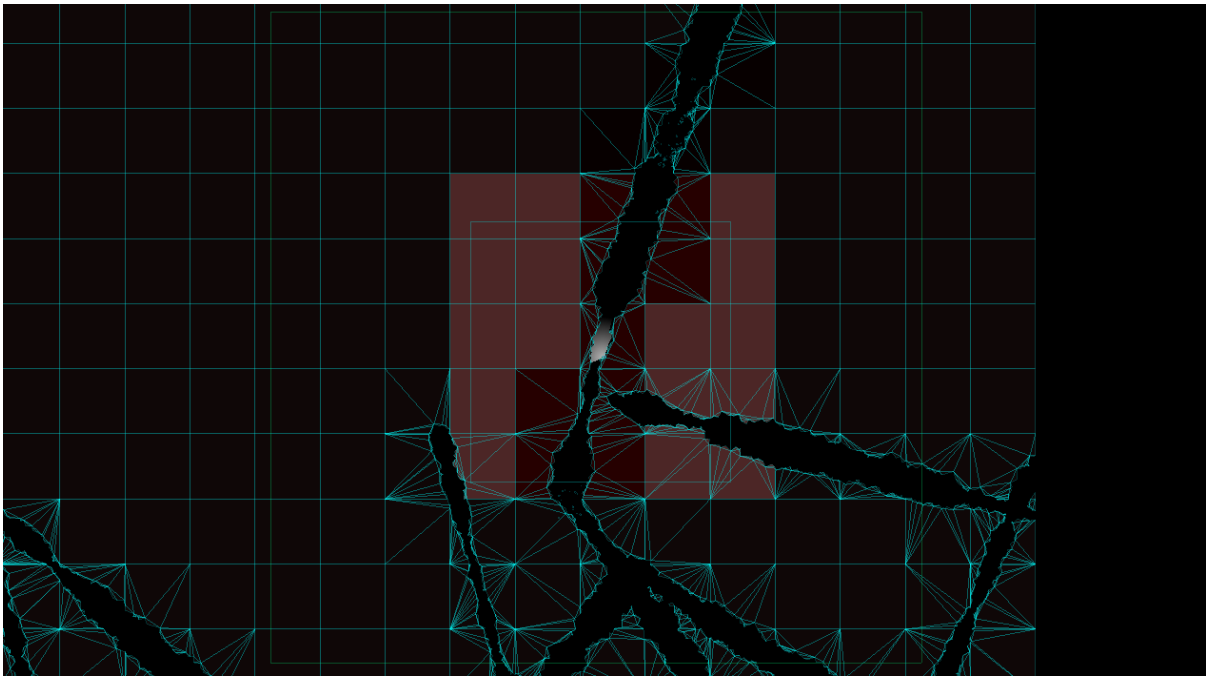
- Finally finished the scene implementation.
- Main menu with start button successfully implemented.
- Fixed some problems in the view system. Especially ListView.
- Reticle lag bug was fixed (made sure reticle render position is update immediately when mouse event is received).

Thursday 06/02

Magnus

- Cleaned up the reticle branch.
- Fixed broken reticle rendering when switching weapons.
- Weapon slots can be specified in json.
- Merged the sniper branch.
- Updated the scene code to work with the new reticles and mouse “disappearing”.
- Exit button.
- Pause menu halting the game.
- “New Game” button that resets the logic.

Friday 07/02



Magnus

- Disable bodies outside physics area.
- Tweaked the active chunk area.
- Render the area as rectangle wireframes.

- Log the time with the logger.

Saturday 08/02

Magnus

- Updated Linux package script to automatically find version information.
- Fix strftime problems in Windows.

Sunday 09/02

Magnus

- Fixed occasional loading hang.
- Tweaks to active area.
- Cleaned up and fixed physics code.
- Added template functions to Actor for finding components.

Monday 10/02

Magnus

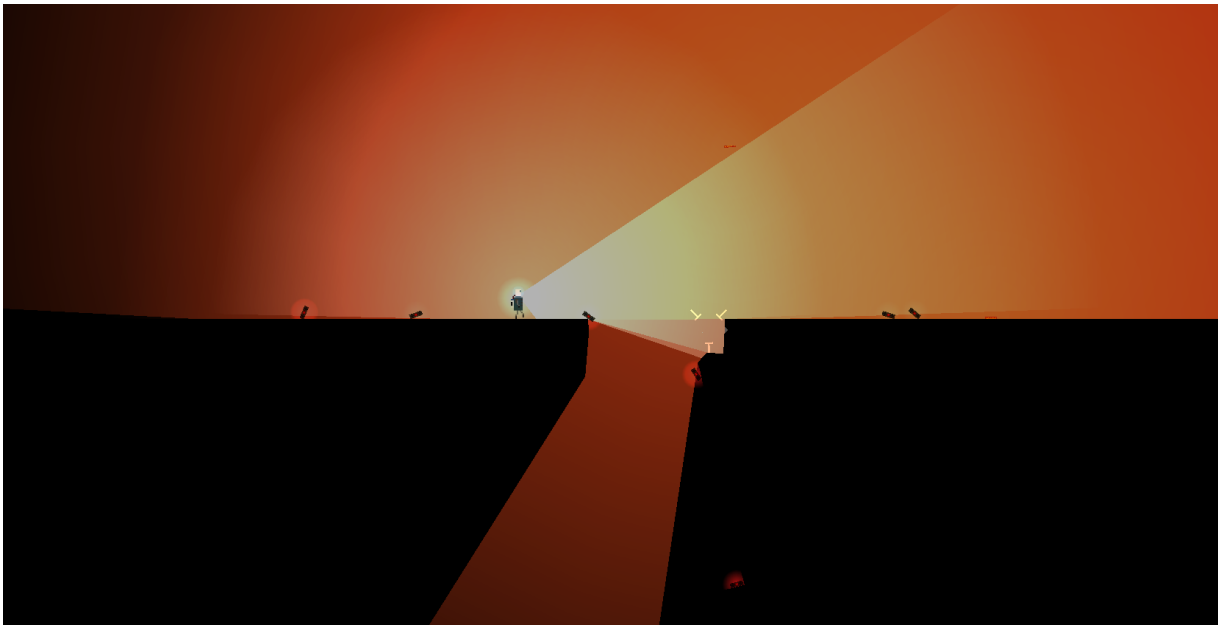
- Performance fixes.
- Reviewed and planned sprint.
- Implemented flashlight feature.
- Fixed light lag.
- Met with Mariusz.
- Made sprint_4 Linux build.
- Wrote sprint_4 blog post.
- Worked on listing assets.
- Started work on flare.

Asbjoern

I was back home for a funeral this weekend from 06/12-09/12, so I didn't do much work during these last few days. I did review some of the work that was done by magnus.

Today we put alot of work into releasing a new version of the game, and a video to go with that, We also had a meeting with mariusz discussing how we should progress. We agreed that we needed to focus more on the gameplay elements of the game, to make it testable for other users. This would allow us to use testers and get feedback on the game. We also reviewed the previous sprint and planned the next as I was away both the thursday and friday.

Tuesday 11/02



Magnus

- Finished flare feature.
- Finished and published blog post.
- Enhancements to the bomb triggering (timed with light).

Asbjoern

I started working on the implementation of PYRO-51 the simple AI for the weak enemies. Jakob, the artist was here and worked on the animation of the mother mob, the melee flying. I helped in designing and making decisions on how this would be implemented. Jakob also created guns for the player and reworked some of the art for the player to be more suitable to the game engine.

Wednesday 12/02

Magnus

- Implement wall lights.
- Fix cursor gone in edit mode.
- Optimize light calculation.
- Implement data storage provide in preparation for saving/loading.

Asbjoern

Today I worked from home as I was not feeling well. I continued to progress with the AI, but progress was slow as my head was not in the best shape of its life.

Thursday 13/02

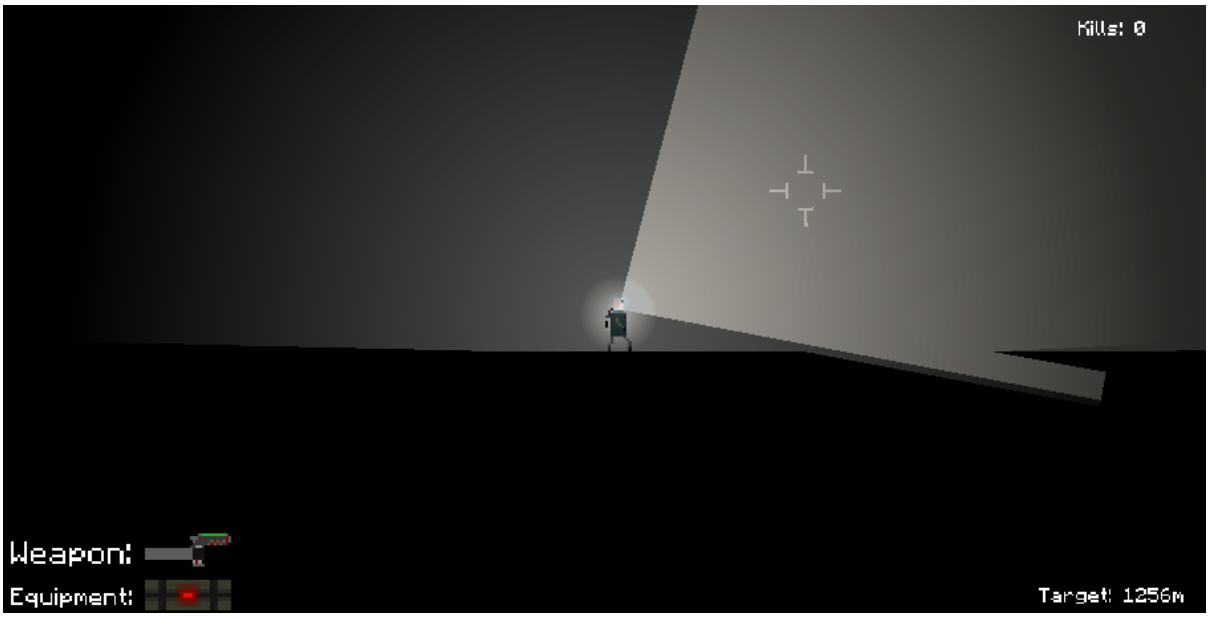
Magnus

- Tested the AI branch.
- Made chunks save and load.
- Helped the other “Hatchling” group with some geometry rendering.

Asbjoern

I continued to work from home. I finished up the weak simple enemies and started working on the mother enemies.

Friday 14/02



Magnus

- Met with Mariusz.
- Developed the saving/loading further.
- Reviewed AI branch.

Asbjoern

We started the day with a meeting with mariusz where we discussed progress. We decided to focus more on the game-elements. I also created the hud today, This gives the player a lot of information on his progress and status.

Sunday 16/02

Magnus

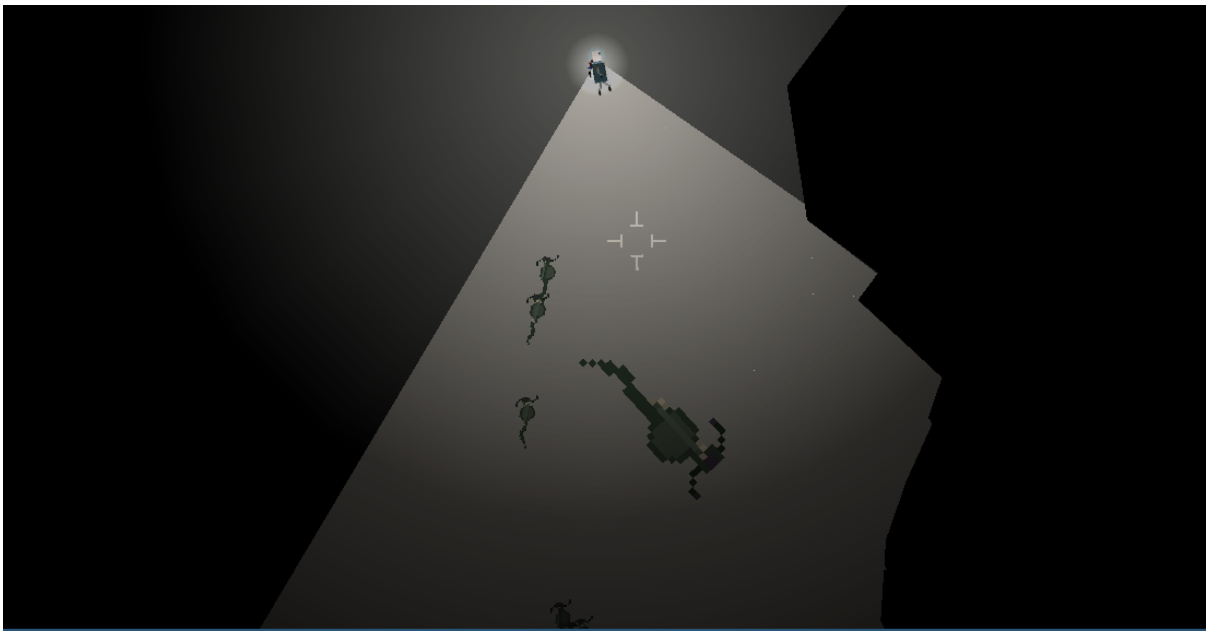
- Fixed the world saving/loading so that the AI’s are properly saved.

Asbjoern
I focused on improving the hud and increasing the performance of the AI in the game.

Monday 17/02
We did the scrum review and planning for the current week. The sprint lengths have been messed up the two previous weeks, but we're getting back on track now.

Magnus
- Tested and reviewed the AI enhancements.
- Added more items to the assets list.
- Merged in all the AI and optimization branches.

Asbjoern
I continued to work with the performance increase branch and fixed some issues with the hud. I also tested the saving and loading system and merged it. I also did some gameplay testing with a few friends and got feedback that the control mechanism was too difficult.

Tuesday 18/02


Magnus
- Fixed incorrect delta time. Easy fix. Added another deltatime that is the actual frame time, rather than the fixed physics step.
- Used the rest of the day implementing the boss fight.

Asbjoern
Today most of the time was spent implementing/finishing the implementation of the mother spawning. The mothers will now start spawning enemies when the see the player, the spawn, that are the simple agents will then attack the player. I also started to implement the mothers fleeing while spawning to avoid getting killed by the player.

Wednesday 19/02

Magnus

- Reviewed AI mother.
- Adde red dust spawning when destroying terrain.
- Began adding yellow dust.
- Discussed a bit regarding the dust pickup mechanism.
- Worked some more on the boss.

Asbjoern

Most of the day was spent making sure the Spawner AI started fleeing when it sees the player, I used a simple raycast method to determine the best path away from the player, by simply sending a bunch of rays in the oposite-ish direction from the player and selecting the one that went furthest without hitting anything. I also started implementing the spawned enemies seeking out the player using a path from the mother. Also added some new texture assets.

Thursday 20/02

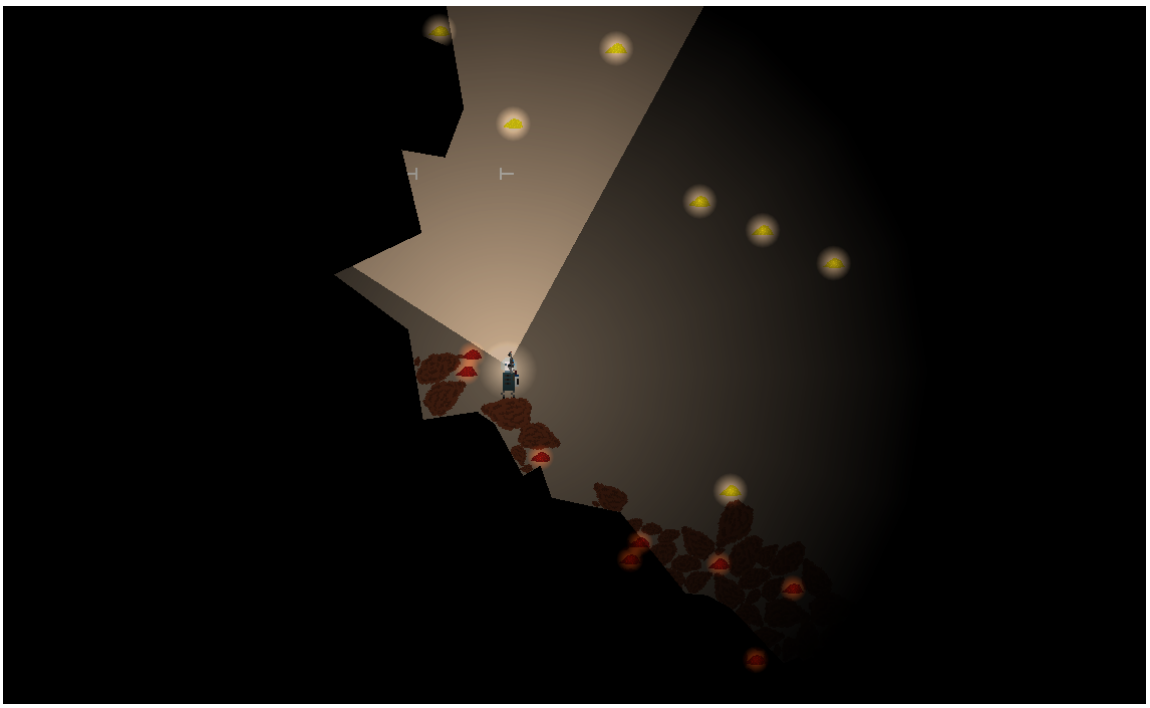
Magnus

- Reviewed AI fleeing.
- Made yellow dust spawn from dead enemies. Had to reconfigure the debris component so that the json can take the impulse and other data for the debris.
- Pick up dust. Took a lot longer time than estimated. Most of the reason is that the sensor, used for picking up, I attached to the player with a joint acted weird. It would some times only be jointed when the player flew with the jetpack, and it always locked the player's orientation. In the end I added the sensor as another fixture to the player body. Worked well.
- Boss spawning is working. Had lot of trouble with EventManager::triggerEvent crashing when calling onEvent on a listener. Problem ended up being that the triggerEvent called itself recursively through someone handling the event, so that it would modify the listener list while iterating through it. Resolved it by locking the triggering, and quing any events that are triggered while locked, so that the function never is called recursively.

Asbjoern

Almost the entire day was spent making sure the Animation of the player and his weapons was better. The player will now turn in the direction he is aiming with the weapon and have the proper stance based on wether he is running, standing, jumping, falling or using the jetpack. Weapons will also be held properly by the player when He aims and turns around. Death frames have been added to indicate when the enemies where killed.

Friday 21/02



Magnus

- Completed boss fight implementation.
- Reviewed the big assets branch, and tweaked some stuff.
- Added textures and glow to the dust.

Asbjoern

As I was leaving early today for a Dream Theater concert. I Spent most of my time fixing some small issues for the release we planned for the weekend. Some fixes for the AI and a lot of merging the branches that were completed was the main focus.

Sunday 23/02

Magnus

- Made it running on Windows.
- Fixed a lot of bugs regarding saving/loading and multithreading. They became more apparent on windows.

Asbjoern

As we prepaired for the release some issues with the way the character felt when moving was fixed, I added a dampening value to the character to allow for greater control when flying around the world.

Monday 24/02

We reviewed the previous sprint, and planned the next.

Magnus

- Fixed a few bugs for the version we planned to release.
- Looked at some issues regarding older Ubuntu version (12.04 LTS).
- Wrote the blog update, posted it, and shared it everywhere.

- Built linux release package.

Asbjoern

Today Also the day we release the new version of the game, There was a lot of work put in ensuring that no major bugs were causing issues with the released executables. We also put a lot of time into sharing the release on various social media platforms.

Tuesday 25/02

We had a meeting as Sutting Digital that took most of the day, so not much work was on Pyroeis.

Magnus

- Implemented the inventory system (actor component storing item count).
- Upgraded Jira during the evening.

Asbjoern

I started implementing the goo spitting enemies (ranged flying) and continued with that until the meeting.

Wednesday 26/02

Magnus

- Reviewed the ranged flying AI.
- Implemented a working inventory screen displaying all items and their count.

Asbjoern

I finished up the GooSpitting enemies, and made sure that the world would spawn them as well as the mothers. I also cleaned up some of the code to remove warnings that where specific to XCode for OSX. The enemies are also roaming around the caves system looking

for the player when they are not in combat. This was easily implemented by extending the usage of the ray casting algorithm that the mothers were using when fleeing.

Thursday 27/02

Magnus

- Remade the inventory screen to fit the design made by Jakob. A lot of work went into this as the items had a more complex positioning. We planned to have arrows from dust to items, but this was too hard to place with the gui system, so we dropped them and made the buttons more obvious.

Asbjoern

I started the day by reviewing some pullrequests with enhances for various elements of the dust spawing system. The bombs will now damage the player and enemies if they are too close to the explosion. I also started implementing and finding some placeholder sounds for the game.

Friday 28/02

This was a normal review/planning day with a meeting with Mariusz, review of the sprint and milestone, and planning the next milestone and sprint.

We also had some extra fun testing the multiplayer version of the game with three persons, this resulted in a surprisingly playable and great experience.

Magnus

- Added both item and gem crafting to the inventory.
- Reviewed explosion damage feature.

- Reviewed the new sounds.
- Implemented the dust attraction during the evening. Still needs some minor work.

Asbjoern

I continued to find and implement more sound assets for the game. I also tried to get Antialiasing working, but with no luck this far.

Saturday 01/03

We built a new version to be released, made a demo video of the features, and posted the update on youtube, our blog, facebook, twitter, and google plus.

Magnus

- Tested the performance improvements of using AABB query on light ray casting rather than checking every existing fixture.
- Made the linux package used shared objects rather than statically link. It now includes libstdc++ and should therefore be able to run on older systems.

Asbjoern

A lot of sound assets were added to the game, I also tried to add several similar sounds to the various effects, this is to ensure that the same sound is not played at the same time, or very close to each other, causing very ugly effects. I also built and tested the windows and mac versions of the game.

Sunday 03/03

Asbjoern

Worked on the multisampling, The lights are now rendered with this, the effect causes the light to appear much more smooth than previously.

Monday 03/03

We had a meeting at Suttung Digital, so a lot of hours went into that.

Magnus

- Made so that lights that don't need to cast shadow, won't ray cast. This improves performance a lot since the light calculation algorithm is slow. Now it will only update the vertices each time some light data has changed, which is a fast process.
- Enhanced the actor system with an active/inactive state. The actor is set to inactive when outside the "active area", and active when inside. All the components can override a method to change their processing when active, or inactive.

Asbjoern

Most of the day was spent improving and properly implementing the multisampling. I put a lot of effort into researching the need for mipmapping and how to implement it in our game. The result is that mipmapping will not help smooth our textures. The reason for this is that we are

using pixel graphics, and since mipmapping downscales the texture this does not make sense in our case.

Tuesday 04/03 - Thursday 06/03

We did a game jam on another game Dreki based on the Pyroeis code as it currently is. Some new functionality could be merged back into Pyroeis, especially the new terrain generation with the hills terrain.

Monday 10/03

Time was spent on discussing the Dreki game jam, and we reviewed the previous “sprint” and planned the next.

Magnus

- Cleaned up the merged Dreki terrain.
- Fixed issue with the actor json extend code where members that had conflicting types would crash the program.
- Fixed so that the chunk generators will stop executing when the chunk loader is halted.
- Looked at supersampling and helped Asbjørn with this. Updated laptop drivers to support OpenGL 3.3.

Asbjoern

I put work into the multisampling and supersampling branch, the supersampling is used to render the textures more smoothly to the screen when the pixels of the art is not aligned with the pixels of the screen.

Tuesday 11/03

After 12 we worked on Suttung Digital and Dreki, rather than Pyroeis, so we lost a lot of time.

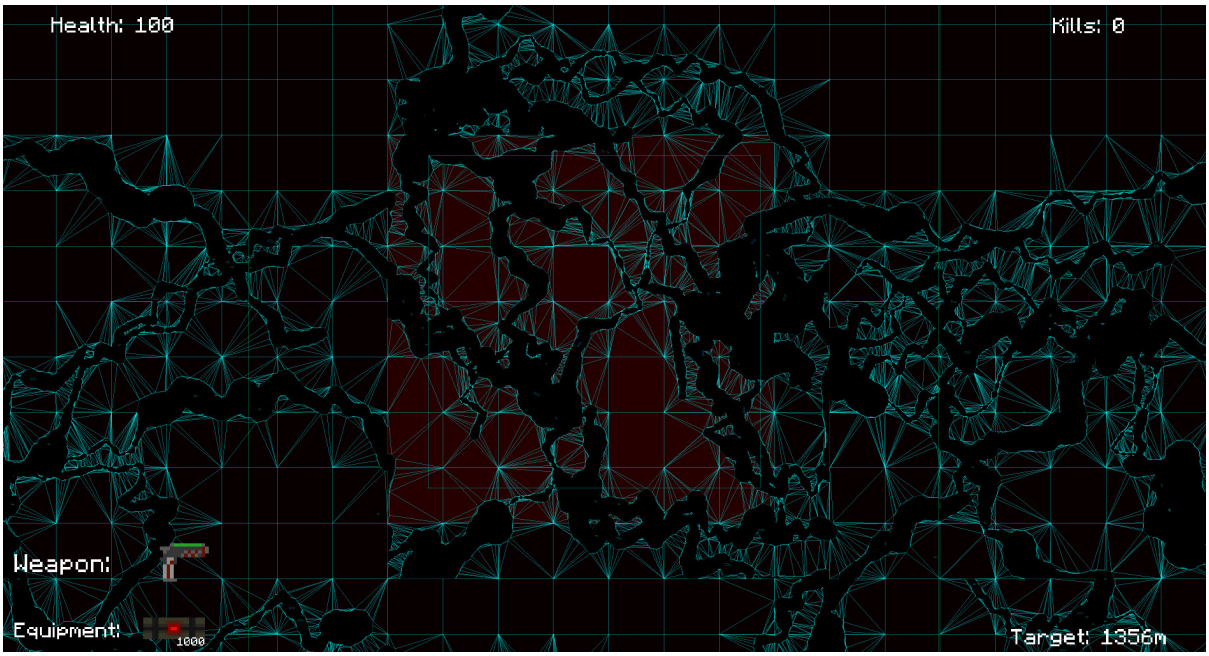
Magnus

- Documented new terrain code pulled from Dreki.
- Fixed fatal crash sometimes happening when spawning AI from the chunk loader thread. Did not happen until I upgraded my graphics driver. Probably shifted the timings so that the threads crashed more often.

Asbjoern

Today I worked on supersampling the texture that we use to render the text. Supersampling is a very costly way of achieving what we are looking for, and the results are also somewhat blurring the crispness of the texture. Research shows that we should be looking into aligning the text with the pixels of the screen.

Wednesday 12/03



Asbjoern

I overslept badly and worked from home not feeling my best. I worked with the implementation of texturing the world.

Magnus

- Looked how the biome system should act.
- Enhanced the terrain generation with more parameters like displacing the paths to make them curve.
- Helped Hatchling with their geometry rendering.

Thursday 13/03

The day was cut a little short by a meeting with Suttung Digital.

Magnus

- Fixed problems with my pull requests that Asbjørn was going to merge.
- Discussed the world texturing with Asbjørn.
- Discussed the biome system with Asbjørn. The terrain generation will also be parameterized by the world heat/humidity.

Asbjoern

At the start of the day I reviewed two branches that Magnus had created, I found a few bugs that needed to be sorted with each of them. After Magnus fixed these I merged them into the master branch.
I continued with the texturing of the world. I have now implemented most of the needed OpenGL functionalty for rendering the textured quads to the framebuffers stencil bit.

Friday 14/03

We met with our Supervisor, did the sprint review, and planned the next.

Magnus

- Research publishing services like Steam and Desura.
- Worked on making the game pause and drop fps when out of focus.

Asbjoern

Most of the day was spent doing non-development related stuff like researching the posibilities for releasing our game on steam and other platforms. We also planned out the next week.

Saturday 15/03

Asbjoern

Today I reviewed some pull requests, The AABB testing for the raycasting and the program becoming Idle when in the background. I also tested some different ways to render the lighting.

Monday 17/03

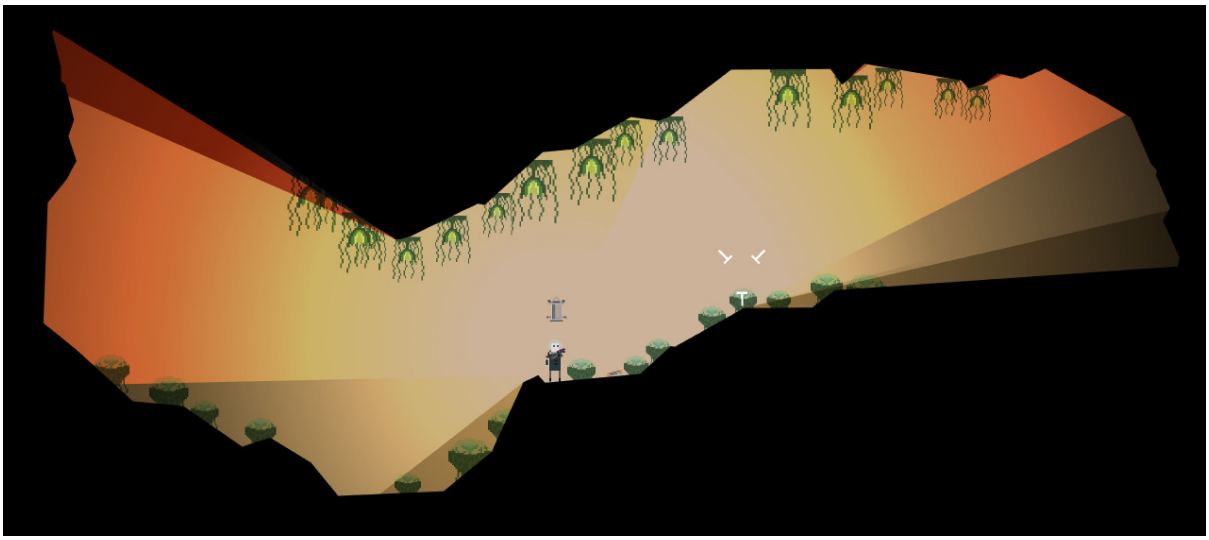
Magnus

- Wrote more about the world random generation and chunk handling.
- Started spawning plants in world and made them sit on the ground.
- Made plants die and spawn dust.
- Lost a lot of time because of car troubles.

Asbjoern

I had to go to the bank to create an account for Suttung Digital and so I lost some time from the development. I worked on the textures.

Tuesday 18/03



Magnus

- Started work on selecting plants based on their requirements.
- Added hanging plants that will stick to the roof.
- Lost a lot of time because of car troubles again.

Asbjoern

Today I made a lot of progress on the texturizing of the world. I'm using the stencil to render only the terrain to a specific part of the screen. It's now working okay.

Wednesday 19/03

Magnus

- Fixed cone lights without shadow casting acting like radial light.
- Helped Asbjørn with terrain texture.
- Optimized actor creation.

Asbjoern

The texturing of the world in nearly complete, You can now texturize the terrain and the background of the world.

Thursday 20/03

Magnus

- Optimized actor creation and json handling.
- Made an actor manager that handles multithreaded creation of actors. This removes the horrible lag spike.

Asbjoern

I made some improvements to the way the texturizing is done. You can have a separate renderer for each tileset you want to render, like the background.

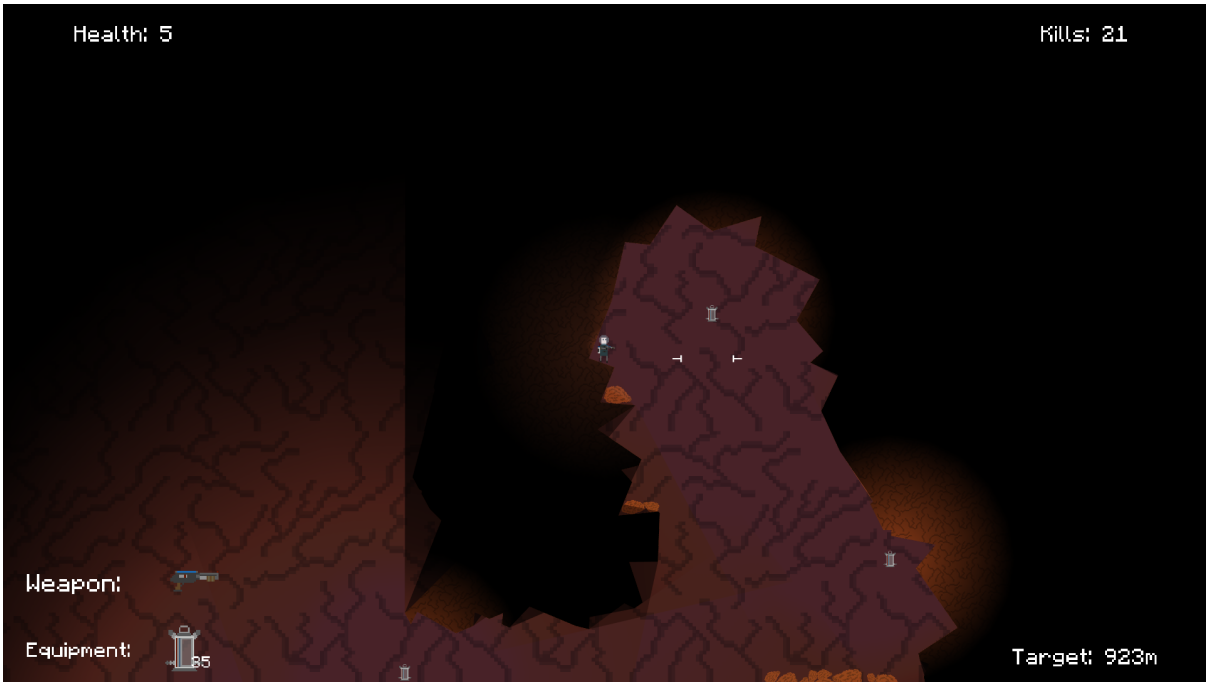
Friday 21/03

Magnus

Worked all day trying to fix horrible crashes in actor spawning. Fixed the most serious crashes, but still some crashes remain. Varies with platform and compiler optimizations.

Asbjoern

I worked from home and completed the texture rendering.



I also did alot of bug-finding and performance testing on the branch Magnus is working on implementing. the biomes.

Saturday 22/03

Magnus

Improved the Linux package with an installer script that installs a desktop file.

Asbjoern

I fixed some issues with the windows specific implementation of the texture renderer. Also improved performance for some issues.

Monday 24/03

We did the sprint review and planning today.

Magnus

- Reviewed and merged terrain texturizing.
- Worked more on fixing and tweaking plant spawning.

Asbjoern

Small improvements to the texturizing. Balanced the weapons somewhat.

Tuesday 25/03

Magnus

Worked with Asbjørn on fixing the actor spawning crash in feature/biome.

Asbjoern

I helped Magnus find the reasons for the frequent crashes in the biome branch. I also did some general performance testing for the new method of generating terrain. The color of the terrain was tweaked with the help of the artist, Jakob.

Wednesday 26/03

Magnus

- Finished up plant spawning branch to be reviewed.
- Made heat and moisture vary with terrain and base plant spawning on this.

Asbjoern

I spent a lot of time reviewing the code for the feature/biome branch. After this I started working on the weapon tweaking branch, making the UI for the weapon tweaking system and setting up the general functionality.

Thursday 27/03

Magnus

- Added cactus, dry bush, and tree to the world.
- Started work on varying terrain with world environment.

Asbjoern

I tested some various methods of blurring the light that we have in the game. we are not happy with the result. I continued the work on the branch for the weapon tweaking menu, most of the system is ready, I still need to do the actual implementation of connecting the actual elements. I also helped out Nordling a bit with his project on the AI for their bachelor Hatchling as the design is similar and I did parts of the AI for our engine during the Game Programming course.

Friday 28/03

Magnus

- Looked at light blurring with Asbjørn.
- Fixed actors sometimes being deactivated for ever.
- Restructured code directory.

Asbjoern

Today we both spend some time looking at different ways of doing the blurring of the light for the game. We found some results that were pleasing but none that were the ideal solution to our problem. The issue is that we want the shadows to be blurred but not the edges of the world, as this creates a fuzzy/unsharp effect that is not desirable.

Saturday 29/03

Magnus

Continued restructuring and enhanced CMake build script to build libraries.

Asbjoern

I continued setting up the tweaking menu for the weapons.

Sunday 30/03

Asbjoern

I tested the performance of the current master on my windows machine. The issues I have been having with the performance on my mac are not present here, probably due to the cpu being way more powerful.

Monday 31/03

We did our sprint review and planning, and reviewed milestone 3.

Magnus

- Tweaked terrain variation.
- Looked at bug where plants would often not spawn.

Asbjoern

The weapon tweaking system is nearing completion. the UI system is setup and ready to take the input from the user. I still need to hook the buttons callback functions up to the actual weapons.

Tuesday 01/04

Magnus

- Fixed bug where plants often would not spawn.
- Tweaked bombs to use exponential damage and impulse based on the distance (stronger close, weaker far away).

Asbjoern

I made some good progress for the weapon tweaking system, the weapons now have their own weapon power component that handles the extra stats that the weapon will have added to its base stats once the user edits the weapon.

Wednesday 02/04

Magnus

- Made destructions intersecting surrounding chunks apply on these as well. Works well, but some plants hover.
- Worked on making plants fall when not touching terrain. More troublesome that I thought. Made the plants dynamic with a sensor and a joint to the world to accomplish this. They often fall even though the terrain is still there.

Asbjoern

I finished the basics of the weapon tweaking system, The weapons are not balanced yet but the tweaking is working as intended and lets the player spend gems to upgrade various stats of the weapons.

Thursday 03/04

Magnus

- Worked more on making plants fall. Progressed some, but new problems appeared. Crashed a lot.
- Tried to trender tooltip views on top of others. Failed.

Asbjoern

I started the balancing of the weapons to make sure they all have stats that are working and legit. I also cleaned up the UI and started implementing tooltips that will appear when the user hovers the mouse over an element. I encountered an issue with the UI system that makes the tooltip appear under other elements of the UI due to the way the hierarchy is handled. The issue is being investigated by Magnus as he is the creator of the UI system and thus more familiar with it.

Friday 04/04

Magnus

- Made the tooltip render on top.
- Worked on fixing falling plants crash. Fixed crash, but plants still fall.
- Reviewed enhancements on rocks with health.

Asbjoern

Magnus found a fix for the tooltip bug, I implemented it and finished the implementation of the branch. I also spent some time testing the terrain generation fix that magnus has been working on, it seems like the crash is gone.I also implemented the destructible rocks branch that ensures that the player is able to destroy the debris that is created when an explosive is

used on the world. This helps avoid the player being unable to move further due to being stuck behind a bunch of rocks.

Saturday 05/04

Magnus

- Reviewed and enhanced weapon customization.

Sunday 06/04

Magnus

- Reviewed and enhanced weapon customization.

Monday 07/04

Magnus

- Finished reviewing weapon customization and merged it after requested documentation was added.
- Tried to fix falling plants. Again some progress, but still not successful.
- Made some minor enhancements to the terrain destructions on other chunks branch. Created pull request.

Asbjoern

I started writing up the report on the light rendering, especially the methods we have used in order to improve and adapt the lighting to the specific criterias introduced by Pyroeis. I also started doing some research on what methods that could be done in order to improve the performance of the lighting.

Tuesday 08/04

Magnus

- Did some code review.
- Made actors store components in std::vector rather than std::unordered_map.

Asbjoern

I fixed and did some timing tests to test some of the different methods we have implemented in order to make the lighting calculation more efficient. results are varying and The improvements may be non-existent due to the checks taking more time than the actual calculation. These are the results when we are checking the points we raycast towards are equal (enough, due to float variance). Blue dust was created and added to the game, the plants now have an equal chance to drop each of them. Reviewed and merged actor vector iteration branch.

Wednesday 09/04

Magnus

- Started work on re-enabling saving.
- Worked on falling plants, but did not progress. Decided to pause this issue and focus on saving.

Asbjoern

Implemented the ability to create and use health packs. The player can create health packs using blue dust, and then use the health pack to heal if he has lost health.

Thursday 10/04

Magnus

- Made basic saving work again (save button that saves the whole world on main thread).
- Reviewed health kit feature.
- Reviewed flare fade fix.
- Prepared the Linux release and blog update. Fixed some crashes when tweaking weapons.

Asbjoern

I have created a fix for the lighting fading bug when the lights of the flare start to shrink. instead of the light fade component shrinking all the lights you are now able to specify what lights, by name, that you want to fade. This was done by giving a name to every light and changing the light specific functions to access the lights by their names. I also prepared the release versions for Windows and Mac, and created a video, highlighting the major improvements in the new version.

Friday 11/04

Magnus

- Reviewed collision damage feature. Changed the way the damage is calculated.
- Split WorldHandler so that the world handler is part og the game logic rather than an actor, and a world generator that generates the terrain and plants will be part of the actor.

Asbjoern

I implemented damage for the actors when they collide with a great enough force, This allows for interesting gameplay when the player can for example plant bombs that create debris that fall and crush enemies. This also adds another difficulty level to navigating the world. I started discussing the musical feel we want for the game with Stian.

Saturday 12/04

Asbjoern

I had a discussion with Stian who is going to create the music for our game about the implementation requirements with regards to the music. We have decided on a model where the music has three stages, explore, danger and battle, these will require separate tracks that are able to loop and switch between each other dynamically. There will also be the need for one extra track of base that loops.

Sunday 13/04

Asbjoern

I did some work to try and get the game working for older macs. I have narrowed the issues down to two separate problems; There is an issue when creating the framebuffer on the newer versions, and an issue when using the libvorbis for doing the sound. I was not able to fix any of the issues.

Monday 14/04

Continued the work of figuring out the issues with some macs.

Saturday 12/04 - Monday 21/04

Magnus

Worked during Easter to re-enable the saving. Had to fix terrain generation and a lot of other bugs. Details in git log. At the end of Easter the saving was working properly again. Some minor issues were still there though.

I removed the threaded actor creation from ActorManager because the normal actor creation is now thread safe. Doing this actually improved the performance a lot and almost eliminated the fps drop when loading a chunk. This is most likely because more processing is put onto the thread that creates the actor. The only things blocking the main thread is the locking when pushing an actor on to the vector and broadcasting of the creation.

Tuesday 22/04

Magnus

- Changed the terrain config so that you specify how many bodies each chunk is split into (in each direction) rather than how large each body is. This makes the body size, which is now calculated, naturally match the chunk size.
- Changed how the ListView's children placement is decided. Previously enum, now class for each. This was to more easily add new placement options, and to make data specific to each placement only part of that placement.

- Let the player load the save when dead.

Asbjoern

Today we fixed the issues with the old macs, the issue was narrowed down to a hardware specific issue with OpenGL. The pure stencil buffer is not required in OpenGL 3.0 and the nvidia cards of the two 2009 macs we've had trouble with does not support this. A google search resulted in this: http://stackoverflow.com/questions/11084961/binding-a-stencil-render-buffer-to-a-frame-buffer-in-opengl We should use the combined depth/stencil buffer. We changed this out and the issues were resolved.

Wednesday 23/04

Magnus

- Reviewed the old mac support.
- Fixed collision impulses sometimes being too large. Reason to the problem was that we misunderstood how the normal impulse passed as argument was meant. It is not x and y, but one or two contact points. If only one, then the second, what we thought was y, is uninitialized and therefore random garbage. Merged the branch.
- Hide load button when no save exists.
- Enhance the chunk generation in several ways.
- Removed 'new game' from game won scene since it might be accidentally clicked and there is no real need for it since you can go to the menu and start a new game.

Asbjoern

Today I tested the saving branch extensively, it has a lot of smaller issues that need to be dealt with before merging. I also reviewed the bomb tweak branch, the difference was not that big.

Thursday 24/04

Magnus

- Continued tweaking bombs in bomb tweak branch. Fixed a wrong calculation.
- Fixed deadlock in saving branch.
- Helped Asbjørn with fixing xcode and sdl problems, fixing the astronaught animation after loading, and fixing saving problems.
- Discussed some of the multiplayer possibilities in the game.
- Used some time to look at published bachelor thesis to get a better idea on how to write ours.

Asbjoern

I discussed the state of the bomb tweak branch with magnus. He had a look at it and fixed some bugs and tweaked the balance a bit more. I also tested the bombs and tweaked the health of the rocks a bit to avoid them being too difficult to destroy. I Also worked on fixing the issues we've been having with the saving branch. I also had a look at fixing the sound delay issues, this would involve having different delays for types of sounds or the individual sounds themselves.

Friday 25/04

Reviewed milestone 4.
Meeting with Mariusz.
We also did some fun multiplayer testing, It's getting really buggy now. In order to support it we'd need an extensive amount of work done.

Magnus

- More code documentation for the saving branch.
- Fixed menu appearing over saving window.
- Helped Asbjørn fixing actor state saving problems.
- Updated network branch with the latest changes.

Asbjoern

I continued to fix the issues with the saving and loading. The more fatal issues are gone, the only known issue that remains a real problem is the "weapon saving problem". Weapons are saved, but new ones are created and given to the player everytime he loads.

Saturday 26/04

Magnus

- Completed documentation of saving branch.
- Reviewed actor state save fix.

Asbjoern

Reviewed the save_button branch.

Monday 28/04

Magnus

- Wrote about how the world is destroyed. Completed first iteration.
- Helped Tomas with pausing the game and the ai crashing.

Asbjoern

Today all day was spent writing the report. I wrote some of the basics on the Lighting, backward compatibility, Rendering. I also tested the difference when calculating the lighting with or without the test for equal points.

Tuesday 29/04

Simon had a presentation on the new marking scheme for the bachelors. Seems more suited than the old one.

Magnus

- Wrote about how the world is destroyed. Completed second and third iteration.
- Started putting destruction writings into the latex document.

Asbjoern
Report, Difficulty, OpenGL rendering, alternatives to OpenGL. Multiplatform specific benefits to OpenGL.

Wednesday 30/04

Magnus
- Continued putting destruction writings into the latex document. All destruction writings is there now.
- Enhanced the writings about destruction and generation.

Asbjoern
Today most of the day was spent writing about the rendering pipeline we currently have in our engine. some restructuring of the previously written material.

Thursday 01/05

Magnus
Wrote about world handling, terrain generation and saving. Restructured some of the world sections.

Asbjoern
I finished most of the rendering documentation, and also wrote some on the new AI system for Pyroeis.

Friday 02/05

Magnus
- Wrote and finished writing about the varying environment (fauna, flora, and terrain).
- Looked at lighting blur with Asbjørn.
- Looked through the parts written by Asbjørn.
- Looked through and fixed some comments in the thesis.

Asbjoern
I continued the write up on the lighting calculation, I had magnus test the rendering performance on linux to see if they matched my findings. I also improved the light rendering by introducing a different shader for the final rendering. This makes the rendered light appear more natural by allowing some cross bleeding of colors that are strong. I read through the world handling Magnus has written.

Saturday 03/05

Magnus
- Reviewed lighting bloom.
- Fixing thesis comments.

- Went over the world handling and improved sections based on feedback and my thoughts.
- Enhanced the world handler so that the active area is handled by it rather than the logic. Areas are now based on the screen size and scale with the camera size in play mode. Better structuring of the area handling code.

Asbjoern

Today I started porting the engine so it would run on iOS. I got the engine running, but there are still problems with the rendering.

Sunday 04/05

Asbjoern

I continued the porting of the engine to iOS. The major problems are still the rendering. I got the rendering working when I'm not using any FBO.

Monday 05/05

Magnus

- Reviewed sections about world and refactored some of the text.
- Wrote about the system architecture and upgraded the diagram from Project Nox.
- Helped with the introduction chapter.
- Wrote about systems, especially actor.

Asbjoern

I wrote the introduction and some future improvements we are looking to do with pyroeis in the future. I also figured out and fixed the issues with the iOS rendering. Turns out iOS does not use the OpenGL standard specified default FBO and Renderbuffer 0 but instead use the 1. This was solved with the help of the SDL forum.

Tuesday 06/05

Short meeting with Mariusz about the thesis.

Magnus

- Wrote more about the systems.
- Created sequence diagrams.
- Wrote about program flow.

Asbjoern

We met with Mariusz and discussed the progression of the thesis, also showed him the mobile port.

Wednesday 07/05

Magnus

- Writing about the remaining systems.
- Reviewed weapon and inventory sections.

Asbjoern

I read through the entire bachelors and commented some minor issues. I wrote the weapon and inventory implementation explanation. I also reviewed some pending pull requests that had some port friendly enhancements. And fixed the enhanced lighting fix. I also did some major improvements to the iOS port's performance. And enabled anti aliasing on iOS.

Thursday 08/05

Magnus

- Wrote about workflow and development process.
- Wrote on lighting optimization.
- Reviewed optimization and further development.
- Wrote some other stuff.

Asbjoern

I wrote about the performance optimizations done to Pyoreis to keep the game running great. I also wrote about rendering improvements, like blur and AA.

Friday 09/05

We had a meeting with marius where we discussed the written style of the bachelors

Magnus

Moved stuff to LaTeX.

Asbjoern

The day was spent creating the pdf using LaTex. 30-40% complete

Saturday 10/05

Asbjoern

Report wrote about iOS port, OS X and windows tools. Read through much of the bachelors. Also continued working on the iOS port, and started an implementation of a particle system.

Sunday 11/05

Magnus

- Researched user testing.
- Planned user testing session.
- Did a user testing session with Tomas, Edvin and Nordling.
- Wrote about the test process and results.
- Wrote about linking on Linux.

Asbjoern

Continued working on the report, getting parts into the LaTex. wrote about the rendering process.

Monday 12/05

Magnus

- Tested and tweaked new LaTeX class from Simon.
- Merged light blooming.
- Packaged pyroeis for Linux release.
- Finished user testing writings.
- Finished Linux deployment.

Asbjoern

Built and thoroughly tested OS X and Windows versions of the application. Finished putting the remaining parts into the LaTex document.

Tuesday 13/05

Magnus

- Fixed various latex things.
- Reviewed light and rendering sections.
- Reviewing thesis, and enhancing architecture section.

Asbjoern

Report, refining lighthing and rendering sections. Created playthrough for release.

Wednesday 14/05

Magnus

- Created image for AI sight.
- Enhance TD with diagrams and text.
- Created more AI images.
- Wrote about planned feedback system.

Asbjoern

Report, Artificial intelligence, lighting, rendering, Testing.

Thursday 15/05

Magnus

- Looked at feedback.
- Reviewed comments on thesis.
- Worked on game design.
- Experimented with raycast inlining based on Asbjørns observations. Ended up being Asbjørn's CPU boost creating false results.

Asbjoern

Providing feedback for the report, fixing small mistakes. Creating playthrough. Tested inlining raycast code. Massive gain turned out to be false positive that was created by the boost feature of my CPU. Creating proper citations.

Friday 16/05

Met with supervisor about thesis, and planned important tasks for the thesis.

Magnus

- Worked on game design.

Asbjoern

Fixing various issues and reviewing parts of the report, Trying to get feedback through various public forums, TIGSource, reddit, twitter.

Saturday 17/05

Magnus

- Making the inline/virtual test better and made nice table, added listings, and wrote more details about results.
- Fixed some comments from supervisor.
- Fixed build errors and crash in Hatchling on Linux/OS X.
- Fixed various things in thesis.
- Added sequence for weapon firing, and described it.

Asbjoern

Reading and fixing parts of the report Items, Equpment. trying to get feedback for the game on reddit and TIGSource.

Sunday 18/05

Magnus

- Fixed various things in the thesis.
- Finished game design chapter.
- Wrote about requirements.
- Made some improvements to the appendix.
- Reviewed crafting and equipment written by Asbjørn.
- Wrote about sharing the game.

Asbjoern

Creating discussion and conclusion parts, equipment and Items.

M Pyroeis: Feedback and Publicity

List of some of the written feedback, and links to posts about the game.

Feedback

Facebook 1 - 14/05:
Etter å ha dødd 5-6 ganger av å sprenge meg selv så satte jeg meg fast under bakken xD Vet ikke om det er noe bra tilbakemelding... Men sykt dette var artig!
http://i.imgur.com/etHixDi.png

Facebook 1 - 14/05:
Er en ting da. Siktet blir mørkt i skyggene så det er vanskelig å se hvor jeg sikter hen

Facebook 2 - 15/05:
Jeg skvatt da “musikken” starta! XD
Elsker lys og skygge animasjonen
Vanskelig aa skyte noen ganger, man maa staa stille? De store monsterne spawner babisser fortere en man rekker aa skyte med haggle.
Ble fanget mellom to kaktuser, tok litt tid aa finne ut at man kan skyte de.
Bakke teksturen gir en ikke helt gjavn gange, maa hoppe over smaa kanter.
Fikk muligheten til aa fortsette aa spille etter jeg dode, med minus liv X)
Tingene I inventory er bare lys?
Endte opp I en dal og kom ikke opp igjen
Potesiale til aa bli veldig bra!!

Facebook 2 - 15/05:
Noen ganger blei jeg skada og andre ikke naar jeg falt langt

Facebook 2 - 15/05:
kei, saa jetpack hjelper litt da -.- Hvem sier at jenter leser bruksanvisningene... Og jeg vet at det er en som er en bombe av itimsene i inventory. Mikke provde spillet og syns det var kjekt Jet packen var vanskelig aa styre sa han, ogsaa var veien ganske lang ned til bunnen eller hva det er der nede (har ikke kommet saa langt enda, men skal gjore et nytt forsok naa

Facebook 2 - 15/05:
Yey! Fordi jeg trykte paa utsiden av spillet naar jeg dode gikk den i pause, og da kunne jeg fortsette aa spille med minus liv. Litt mye bomber og ignorering av fiender ga meg litt over -6000 i health og jeg vant! XD

Facebook 3 - 04-02:
græla kult
jetpacken fungerte bra!
moro å fly rundt
kult lys

Mail 1 - 14/04:
Hi guys,

Just did some gaming on your exploration extravaganza called Pyroeis and have some feedback for you. These are first impressions. I just downloaded the game without looking into the story and background.

I liked the overall look of the game. Loved the explosions and the possibility to just blow your way to where ever you want. It reminded me of Red Faction which I loved for that by the way. After some playing I got confused. I had dropped down for a while but didn't get anywhere. The target got closer but that was it. I felt kind of lost. And then suddenly an enemy appeared and the action was on.

I liked the game you guys got something here. After looking a little bit more into the projects page I found out that you tested multiplayer. That gold guys. I really think that multiplayer would make this kind of exploration experience an really awesome one.

Keep at it and good luck.

Indrek Vändrik,
Creator of ZombieRun

TIGForums:
http://forums.tigsource.com/index.php?topic=40672.0

Twitter 1 - 13/04
@Asbjoern1337 Just took @Pyroeis for a spin. Looking good. Liked the destructable world and explosions. Ok I'll continue playing now.
https://twitter.com/ZombieRunGame/status/455599500338675712

Twitter 2 - 13/04:
@Pyroeis Just played a bit, great atmosphere with the music! The gameplay is really fun with the explosives ;)
https://twitter.com/ian_snyder/status/455551626195906560

Twitter 3 - 14/04:
@Pyroeis I'm really liking Pyroeis so far!
https://twitter.com/AlexanderTeaH/status/455710156723322881

Publicity

The Gaming Ground:
http://thegaminground.com/2014/march/pyroeis-a-2d-action-exploration-game.html
Four tweets:

- https://twitter.com/TheGamingGround/status/440794686886670336
- https://twitter.com/TheGamingGround/status/441244460320432130
- https://twitter.com/TheGamingGround/status/442662038062772224
- https://twitter.com/TheGamingGround/status/451424768903094272

N Pyroeis: Website statistics

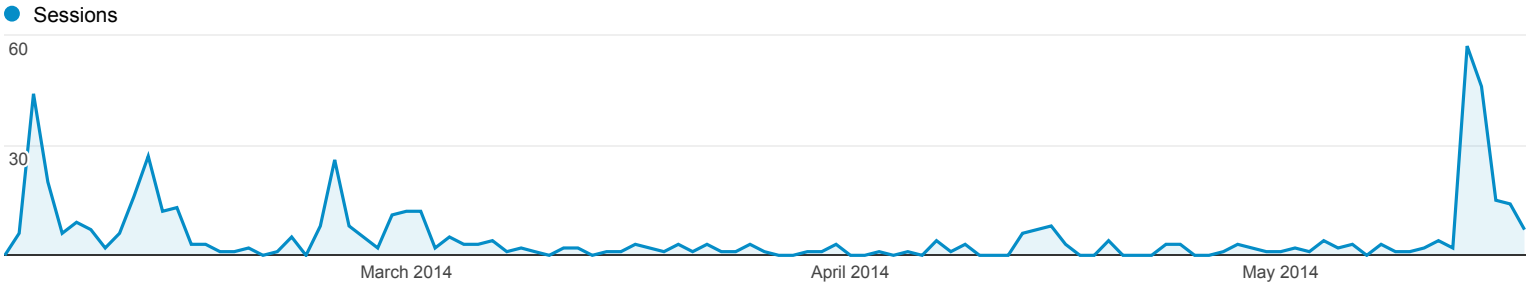
Statistics retrieved from Google Analytics.

Audience Overview

Feb 1, 2014 - May 18, 2014

All Sessions
100.00%

Overview



Sessions
535

Users
279

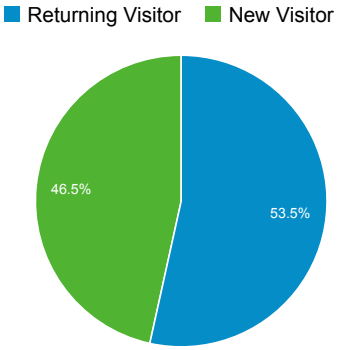
Pageviews
1,147

Pages / Session
2.14

Avg. Session Duration
00:02:05

Bounce Rate
54.58%

% New Sessions
46.54%



	Country / Territory	Sessions	% Sessions
1.	Norway	446	83.36%
2.	United States	21	3.93%
3.	Denmark	10	1.87%
4.	United Kingdom	10	1.87%
5.	Australia	6	1.12%
6.	Germany	6	1.12%
7.	Netherlands	5	0.93%
8.	Canada	4	0.75%
9.	France	4	0.75%
10.	Poland	4	0.75%

O Pyroeis: Source Code

Source code is hosted as a private git repository at Bitbucket. To request access, send an email to mbvett@gmail.com. The source code will be made public in three years from this thesis' publication date.

P Pyroeis: Demonstration Videos

Game playthrough https://www.youtube.com/watch?v=GDXDLinRn4E

Milestone 1 https://www.youtube.com/watch?v=UvA1Su0uTQo

Sprint 4 https://www.youtube.com/watch?v=5QshY8P78Wo

Sprint 6 https://www.youtube.com/watch?v=ILKXjomL9YM

Milestone 2 https://www.youtube.com/watch?v=aM0HrtSlKrE

Sprint 12 https://www.youtube.com/watch?v=Q2ckQRnMYbg

Multiplayer test https://www.youtube.com/watch?v=xeA02d-WCX0