Bachelor's project

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Artūrs Umbraško, Kacper Lewandowski, Daniel Dahl

OS Runner

Educative multiplayer rogue-like deck-building gaming experience in Cybersecurity

Bachelor's project in Programming
Supervisor: Aland Mendoza
Co-supervisor: Marius Pedersen
May 2021

NTNU
Kunnskap for en bedre verden

Artūrs Umbraško, Kacper Lewandowski, Daniel Dahl

OS Runner

Educative multiplayer rogue-like deck-building gaming experience in Cybersecurity

Bachelor's project in Programming
Supervisor: Aland Mendoza
Co-supervisor: Marius Pedersen
May 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

OS Runner

Educative multiplayer rogue-like deck-building gaming experience in Cybersecurity

Kacper Lewandowski Artūrs Umbraško Daniel Dahl

Abstract

Summary of Bachelor thesis

Title: OS Runner
Date: May 20, 2021
Authors: Artūrs Umbraško, Kacper Lewandowski, Daniel Dahl
Supervisor: Aland Mendoza
Employer: Danny Lopez Murillo
Pages: 71
Attachments: 5
Availability: Open
Summary:

Games can be used as a very effective education medium. However, the market of educational cybersecurity games has a very small and limited supply. Danny Lopez Murillo, a cybersecurity specialist, has come forward to us with a proposition to develop a new game to help and change this stance, as well as to be among the first innovators in this field.

This is a relatively niche market, and still, some of the games focusing on the cybersecurity are already there. However, most of them fail to combine the fun game aspect with the education aspect, and end up being undiscovered, with only a few being relatively successful. OS Runner tries to fill the gap, and be serious about learning while having advanced and immersive game elements.

During this bachelor, the development of OS Runner was kick-started. What was produced is not a complete game, but a proof of a perspective concept that can be developed further.

Our game combines both the fun and the educative elements. It can be used as an education tool for high school students and undergraduates during lectures, lessons, seminars and other events. Even with only limited prior knowledge of cybersecurity in general, this game can be an interesting and captivating experience, making sure that players will learn the basic principles of cybersecurity, cyberattack and -defense mechanisms that are used in real life.

The game was developed by us, using Scrumban methodology, in tight continuous cooperation with our employer.

Utdrag

Sammendrag av Bacheloroppgave

Tittel: OS Runner
Dato: May 20, 2021
Deltakere: Artūrs Umbraško, Kacper Lewandowski, Daniel Dahl
Veileder: Aland Mendoza
Oppdragsgiver: Danny Lopez Murillo
Antall sider: 71
Antall vedlegg: 5
Tilgjengelighet: Åpen
Sammendrag:

Spill kan bli brukt som et veldig effektivt læremiddel. Imidlertid er markedet for pedagogiske spill med fokus på cybersikkerhet lite og meget begrenset. Danny Lopes Murillo, en cybersikkerhet spesialist, tilnærmet oss med et forslag om å utvikle et nytt spill for å hjelpe med å endre på denne holdningen, i tillegg til å være noen av de første innovatørene innenfor dette feltet.

Dette er et relativt nisje marked, men det finnes fremdeles noen spill med et fokus på cybersikkerhet. Imidlertid, feiler mange av dem på å kombinere moro med læring og ender opp med å være uoppdaget, med kun noen få som er relativt vellykkede. OS Runner prøver å fylle dette hullet og å være seriøs i å videreformidle kunnskap, i tillegg til å ha avansert og innlevende spill elementer.

I løpet av dette bachelor prosjektet så ble utviklingen av OS Runner satt i gang. Det som er produsert er ikke et komplett spill, men et bevis på et konsept på noe som er tilrettelagt for videre arbeid. Spillet vårt kombinerer både morsomme og lærerike elementer. Det kan bli brukt som et læremiddel for elever på videregående, studenter på universitet/høyskole, seminarer osv. Selv med bare begrenset forkunnskap om cybersikkerhet generelt, så vil dette spillet være en høyst interessent og fengende opplevelse, som forsikrer at spillere vil lære de grunnleggende prinsippene om cybersikkerhet, cyber angrep og forsvarsmekanismer som brukes i det virkelige liv.

Spillet er utviklet av oss, ved bruk av Scrumban metodikken, i tett kontinuerlig samarbeid med vår oppdragsgiver.

Preface

We want to thank Danny Lopez Murillo for his task proposal and giving us the opportunity to work on this very nice task. His continuous motivation and willingness to cooperate with us allowed us to reach a prototype we can be proud of and gave us an enjoyable experience when working on a larger project for the first time. Discussing the game with him also helped us to widen our knowledge of cybersecurity.

We would also like to thank our supervisor Aland Mendoza, who, even while being in a different part of the world, continued to provide us with advice and cheerful spirit. The material he provided for us also helped us immensely while writing this thesis.

Last but not least, we want thank Marius Pedersen for valuable feedback during writing of the thesis and guiding us on how it can be improved.

Contents

Figures

Tables

Code Listings

Acronyms

API Application Programming Interface. 3

GUI Graphical User Interface. 2, 27, 53

HP Health Points. 36

JSON JavaScript Object Notation. 30

LMB Left Mouse Button. 41

PC Personal Computer. 2

SFX Sound effects. 53

UI User Interface. 67

UX User Experience. 8

Glossary

asset A gameplay element representing a network asset that is represented by an interactive object in the topology map. 2

Bitcoin Bitcoin is the in-game currency used for all things shop related. It can be acquired by playing cards which gives bitcoin. 2, 47, 48

BPROG Study code: Bachelor in Programming. 4

buff A positive effect given to the player. 60

codebase Codebase is a collection of source code used to build a particular software system, application, or software component. 2

deck-building game A game in which the player starts with a basic cards and through the game he acquires stronger cards. 3

development velocity development velocity is a metric for work done and is often used in agile software development. 16

discard pile Discard pile contains all of the cards that the player has already played. 32

draw pile Draw pile contains all of the cards that the player will draw if he plays his current available cards. 32

game mechanics Rules that dictate player's possible actions and game's responses to those actions. 3, 7

game settings A set of rules that change the technical behavior of the game. 8

gamification The process of adding games or gamelike elements to something (such as a task) so as to encourage participation. 1

GDScript GDScript is a high-level, dynamically typed programming language and its goal is to be optimized for and tightly integrated with Godot Engine. 51

instance Adding a scene into already existing scene is called instancing. A scene that you play to add into main scene is called instance. If a scene already exists, it will be overwritten instead.. 30

network topology map A set of interconnected assets in the game that the player can interact with. 20

Neutral Asset Conquering One of the main stages (phases) of the gameplay, consisting of the players conquering neutral assets, preceding the enemy asset conquering phase. 60

nickname A nickname is a pseudonym that players can choose in the game. 36, 55

node A "building block" in the Godot engine. A node can be made a child of a different node. A single node can have multiple children. 30, 35

phase One of three main multiplayer game stages, explained in detail in appendix B. 60

player profile A customize set of data portraying the player. 8

playerbase Population of active players of a video game. 58

playtest Process of testing the game for bugs and flaws by playing it. 16

roguelike A subgenre of games which feature procedurally generated levels and permanent death of the player character. 4

scene A tree of nodes is called a scene. 18, 27, 53, 54

state machine It is a behavior model which consists of finite amount of states. The state machine must be in a state at all times.. 60

status effect Status effects are either beneficial or harmful effects that can be inflicted on a player. They usually last multiple turns, affecting the player on the start of each one. 36

unobtainable cards Cards that can not be purchased in the shop. These cards are received through other cards. 32

Chapter 1

Introduction

1 The topic

Currently, the market for educational cybersecurity games is quite shallow. Out of the not so many games that exist in this field, even fewer are combining the educational aspect with being actually fun to play. This project is an attempt to change that, to improve and accelerate the cybersecurity education by introducing a way for students to engage in tangential learning.

Based on a different card game - *Intrusion Attempt* - also in development by our employer for three years now, the game correlates real-life cybersecurity practices and concepts with common video game and card game mechanics and concepts. This way, by engaging in the gameplay, the students will also increase their understanding of the real underlying topics.

The gamification of the learning process with the goal of improving high school and undergraduate students' understanding of the cybersecurity topic are considered by us to be the main benefits of this project. The gamified tangential learning experience can also mean an easier introduction into the topic for those previously not at all familiar with it. This can in turn lead to improved performance in cybersecurity-related classes, greater interest in cybersecurity-related studies, as well as many other positive education effects and applications.

2 The task

The game will be based on a card game project by the employer. The main mechanics will be revolving around a deck-building and card battle gameplay. The

cards will be based off of the MITRE ATT&CK framework [1]. As it is stated on MITRE website: "ATT&CK is a knowledge base of cyber adversary behavior and taxonomy for adversarial actions across their lifecycle." It is being constantly updated and it describes how to prevent cyberthreats, as well as detailed description of different tactics and techniques in cybersecurity. However, we will be working on the technical details of the game, and the cards are supposed to be supplied by the employer.

Initially, we have considered developing both the single-player and multiplayer aspects, but during the requirements design phase, the single-player aspect was cancelled by us limiting the scope of the project.

Cybersecurity will be the main theme of the game. The cards will be based on different methods and techniques used by malicious thread actors and suitable for teaching courses at university level related to penetration testing and red teaming.

The game will teach players some basic aspects and ideas of data security, as well as in general raise data security awareness and interest in high school students, undergraduates and graduates.

The gameplay will consist of two main parts: a map, that will resemble a network topology where players will be able to roam and compete for in-game resource - Bitcoin - and assets, and a card battle screen, where the actual card battles will take place.

Initially, we have planned to develop for the Android platform. However, after discussing this further between each other, and with the employer, we have concluded that we will primarily focus on the Windows platform, and then port the game to Android. This was done for several reasons.

First of all, our team has a much bigger expertise on desktop platforms, such as Windows and Linux. Therefore, we could expect the final product to be of higher quality. We wouldn't have to spend much time researching the intricacies of the platform, but instead we could exploit more in-depth features, and have a clearer vision of the development process from the start, which helps to avoid refactoring and remaking of the codebase during the development.

We also considered the Windows platform to be more suitable for our project and the resources that we have. Desktop platforms are more convenient for class and lecture environments, than mobile platforms. According to some studies [2], PCs have some advantages as a learning medium. First of all, mobile platforms have smaller screen sizes, which is a disadvantage in our text- and GUI-heavy game. Also, a desktop computer is more suitable for a game, since it provides a more immersive game experience, which will improve the quality of the gameplay.

Finally, the desktop platform has a better support of networking features that

we could utilise. We don't have resources to host our own servers, but we can use such frameworks as the Steam API to give players access to multiplayer features without having them to be on the same network.

3 The inspirations

Since we are gamers, we have had the experience with deckbuilders before. Consequently, the OS Runner also takes some inspirations from notable games already on the market.

The idea of the game is greatly inspired by a game called Slay the Spire [3]. Slay the Spire is also a deck-building game, featuring card battles. Some aspects of our battle and card mechanics were inspired by the analogous game mechanics in Slay the Spire. We have considered the way the player battles enemies using cards as a "weapon" something that would work really well with the player-versus-player card battle element in our game.

Slay the Spire was not the only inspiration. Another great example of a deck-builder is a game called Monster Train [4]. The inspirations came in form of the visual design for the game. We believe that the game has a very clear and concise look and user interface, and we have drawn some ideas about how our visuals should look like from it.

4 The audience

Report
This report is written for the examiner, fellow game programming students and the product owner.

Product
The product is created for the product owner, Danny Lopez Murillo.

End-user
As briefly mentioned in Section 1 and Section 2, the end-users will be high school students, undergraduates and graduates. We can therefore safely assume they are mostly young with a basic knowledge of computers and an innate sense of intuitiveness in regards to games.

5 The team

We all are students from the Bachelor in Programming (BPROG) course at the Norwegian University of Science and Technology. We specialize in software development and have worked with projects like a ticket finding and music API, a deck-building roguelike action game, some smaller mobile apps. Since we all study the same thing our expertise is virtually the same.

All three of us are interested in programming and games. This is one of the main reasons as to why we chose to take this task: it seemed like a perfect fit considering our interests in various games, but also our knowledge in programming and game design. The genre of rogue-like deck-builders is also not new to us, and we have played such notable examples of it as: Slay the Spire, Monster Train, Ascension and Poker Quest RPG. During our previous courses we have already worked on a deck-building game, and that is some additional experience that we could use for this task.

Our employer is Danny Lopez Murillo, a former NTNU educator, with a master degree in Cybersecurity, specializing in Ethical Hacking. Currently, he is involved in digital forensics. However, previously he was a teacher at the Norwegian University of Science and Technology for the Ethical Hacking course [5], as well as a teachers assistant for the Network Security course. He admits that he has a great passion for gamification in cybersecurity. He wants the game to be used by students and undergraduates during their courses as an educational tool, as well as make it serve as a knowledge base with links and references to further materials about the topics.

6 The goals

The goal of **OS Runner** is to have a fun strategical deck-building card game trying to facilitate and stimulate cybersecurity education through tangential learning and associations. The game is intended to create an engaging gameplay experience that will also widen the players' understanding of real-life hacking, penetration and red teaming scenarios.

As a group our goal during the project is to learn more about cybersecurity as well as game design. Considering we've also made a game during the "Game Programming" course [6] and also during the "Graphics Programming" course [7]. We also want to know what it's like working in a multidisciplinary team on a bigger project. Improving our game programming, design knowledge and skills is also a very beneficial and interesting experience for us.

We also have no previous experience with making multiplayer. This could be helpful in future projects as well, considering that adding multiplayer to a game opens up a plethora of possibilities in regards to gameplay. Having experience with making multiplayer games will also be professionally useful.

7 The thesis

In this thesis we will be discussing the following:

- **Introduction**
  In this part we will describe the project, the goal, the tasks and the team.
- **Requirements**
  In this part we will discuss the requirements for this project, and how we designed them.
- **Implementation**
  In this part we will discuss how we approached the technical and graphical implementation and design of the game.
- **Discussion**
  In this part we will discuss how much has been accomplished, as well as give the project, the team, and the working process some praise and critique.
- **Further work**
  In this part we are going to discuss how the project can be used and improved after the thesis is over.
- **Conclusion**

Chapter 2

Requirements

1 Initial requirements development

1.1 Planning the requirements

Initially, the task description did not feature too many actual requirements. The employer explained to us what kind of game he had envisioned, what elements he wanted to include, which of these elements were to be stressed out, and what we should avoid. He also gave us a lot of liberty in regards to the development process and the resulting product. In terms of technical details we had the full control: we were free to choose any platforms and frameworks that suited us. In terms of gameplay and UX the employer had a lot of ideas and designs, but was still interested in discussing them and cooperating with us to ensure the most engaging and educative gaming experience.

The absolute majority of the requirements that we have designed was derived from discussions with the employer. We have actually spent a significant portion of time in the beginning of the project to develop a common vision of the game, and consequently develop documented requirements.

1.2 Scope

Since the beginning of the project, we aimed to have a fully functional game with the core gameplay features implemented and a server for multiplayer. Since none of the members specialize in creating game art, our team had primarily focused on creating code for the game mechanics with the visuals being a lesser focus.

1.3 Gameplay flowchart designs

At the end of our planning phase, we have developed design documents that contained a detailed description of how the gameplay had to look like. We then used these documents as a basis for more detailed requirement specifications, such as user stories and tasks.

One of the first designed flowcharts was the main menu flowchart, that explained how the players would navigate the main menu to play the game, change game settingss and edit their player profile. This flowchart can be found on the figure 2.1.

The design document, developed by us, that describes in detail the gameplay flow of a multiplayer game of OS Runner, can be found in appendix B.

1.4 MoSCoW

We have also used the MoSCoW method[8] to classify the most important features that have to be implemented for the game to fully function. MoSCoW method is a requirement prioritization method to classify the requirements by how important it is to include them in the current delivery. According to the method, we have selected a collection of broad, general requirements for our project, and classified them into four groups.

Must haves

This group includes the core game requirements. They were absolutely crucial, and the game could not function without them. As such, they were prioritized first. We mostly worked on developing them in the first iterations of the development cycle.

This category included such base elements as the platform support, the multiplayer, as well as some of the main gameplay aspects, such as cards, the map, and the battles.

Should haves

This group consisted mostly of elements that added the most additional gameplay and educative value, as well as some important UX components. It is important to note that some of the requirements had qualitative description, specifically the visual and sound design requirements, as well as the network performance

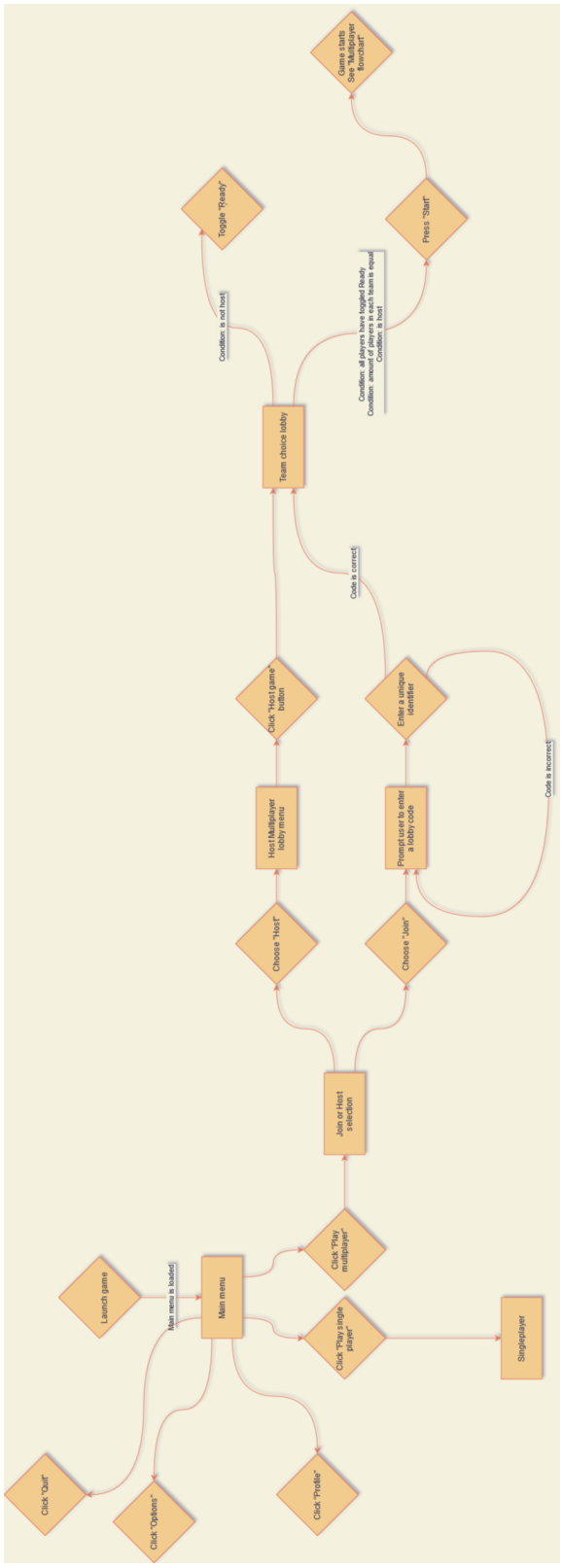


Figure 2.1: Initial flowchart of menu navigation

requirement. Since these descriptions are qualitative, they are not especially descriptive by themselves, but the qualitative descriptors will be more useful when compared to respective requirements in the **Won't have** group. These descriptions help to put emphasis on how much work should be put into the specific element, and not how exactly the element should function.

The listed requirements were such mechanics as the in-game shop and the encyclopedia, support for player-added cards, the persistent player profile, options and more platform supports.

This group was still considered very important, but unlike the **Must haves**, the elements in this group had to be build atop of the already existing core elements. Therefore, they couldn't be prioritized from the get-go, so they were moved into this section.

Could haves

This group consisted mostly of more complex gameplay elements that were proposed by and discussed with the employer, but were from the very beginning considered less important at this moment. Employer was aware of that and agreed with us.

They were moved to this group because they required extensive game base to be built upon, which meant that they will probably be out of scope for the time frame of this project, but if there are no other features to work on, the team members could implement them.

Won't haves

This group consisted of requirements that were not to be expected by the end of this project. They were added to the list in order to help the team to focus on the more important features.

It mostly consists of qualitative descriptions of visual and sound design. None of the team members specialize in those, so that was not the focus of the project.

The requirements list

Below is the complete list of requirements classified using the MoSCoW method. In Chapter 4, we will go through the list again, and take a look at what progress we've achieved.

Must have:

- We must have a card base
- We must have multiple card types
- We must have card battles
- We must have functionality to add cards to the deck
- We must have Local Network multiplayer
- We must have map topology
- We must have map navigation
- We must have a main menu
- We must have Windows support

Should have:

- We should have a persistent player profile
- We should have good network performance
- We should have working single-player mode
- We should have working Bitcoin shop
- We should have options menu
- We should have all of the card types
- We should have in-game card encyclopedia
- We should have nice visuals in the game
- We should have nice audio design in the game
- We should have functionality to easily add new cards into the game
- We should have Linux and Android support

Could have:

- We could have majority of the cards
- We could have a more complex scoring system
- We could have a team-up functionality
- We could have a reputation system
- We could have specializations
- We could have assisting of other players in multiplayer
- We could have working high score menu
- We could have iOS support

Won't have (this time):

- We will not have ALL of the cards
- We will not have a story
- We will not have good visuals in the game
- We will not have good audio design in the game

2 Continuous requirements development

2.1 Weekly meetings

During the initial planning phases, we decided that for our development management methodology we will be using Scrumban[9]. One of the benefits of this agile methodology was that it allowed us to continuously change and generate requirements. The methodology will be discussed in more detail in Chapter 3.

Employer meetings

Every working week we followed a schedule that enabled us to quickly react to issues and design decision that came up. Every Friday during the development we met up with the employer and discussed the progress. During these meetings we would show him the latest prototype of the game, talk about what was added and changed, and listen to his feedback. We would also discuss the direction in which the game is heading, the aspects where our visions of the game disagree, and, most importantly, on what elements we should focus at that moment.

Whenever we and the employer had any disagreeing views, we would have a discussion, during which we would find the best solution together.

Team meetings

On the following Monday, we would then have a closed team discussion. One of the main topics in these team meetings would be generating new user stories, which we would incorporate into our Scrumban board and focus on in the following week.

2.2 User stories

Some of the user stories were developed immediately after creating the gameplay flow design. But most of them we have developed later on, during the course of development and during the weekly meetings. We have kept the user stories short and atomic, so that they are easier for us as developers to work on. We have also used acceptance criteria parameter to define when a user story should be considered finished. The user stories document can be found in appendix A.

A typical user story

A typical user story used in our documentation can be seen on the figure 2.2, and it consisted of multiple elements.

Each issue had an internal number for easier conversion into GitHub issues. The number just continuously increased with each new issue, and was prefixed at the beginning when a corresponding issue was created.

Each user story also had a description, that functioned as a name. This description always followed the same formula: it described the actor, and what kind of action that actor wants to perform. This description field was used together with the internal ID in GitHub issues for easy reference.

The final field is the acceptance criteria. It describes the issue in a greater detail by adding specific sub-requirements, that have to be fulfilled in order for the issue to be considered finished. This helped us unify our understanding and vision of every user story, without adding too much complexity, which could have resulted in having to break down many issues into smaller tasks.

Designing user stories

The layout and fields that we chose, as well as the way we wrote the issues was not arbitrary. We spent quite a bit of time in the beginning of the project discussing and researching different methods of efficient writing and use of user stories.

Many of the user story guidelines that we have followed were inspired by a user story guideline, published by the Agile community of the Government of the United Kingdom[10].

What we found most appealing with these guidelines, is that they allow us to be specific about what kind of problem we are solving, because they offer a detailed overview of it, while also helping to make sure that it is the right problem, since the user - the player in the absolute majority of cases - is a part of that user story and is always considered. We also found the acceptance criteria very appealing, because they help us make sure that every team member sees the problem the same way, which minimizes the risk of diverging visions and incompatible solutions.

User Story number	9
Description	As a player, I want to have my own profile in the game
Acceptance criteria	I. It's done, when the player can access the profile from the main menu II. It's done, when the profile is persistent III. It's done, when all the profile data is displayed in the profile menu IV. It's done, when the player can go back to the previous screen after accessing the profile

Figure 2.2: A layout of a typical user story

Tasks

Even though we have tried to make the user stories short and concise, some of them still required to be broken down into smaller components: tasks. These tasks focused more on small technical elements that had to be implemented in order to finish a larger user story, as well as to specify the amount of technical that had to be put into a user story, in order to make them more granular and smooth. They also followed a specific structure: they had a number, which was assigned based on the related area of development, such as GUI, multiplayer or map topology; they also had a field for specifying the connected user story for easy reference; and they had a small description, which specified an atomic technical detail to be implemented. A typical task can be seen on figure 2.3.

Tasks were also added as GitHub issues, prefixed with a letter 'T', task number and connected user story number.

Task number	103
Connected User Story number	26
Description	Develop a data structure to keep votes from each team
Acceptance criteria	

Figure 2.3: A layout of a typical task

Chapter 3

Implementation

1 Introduction

This Chapter will explain different sides regarding the implementation of our end-product. We will start with the development process and explain how we used Scrumban methodology when developing. Furthermore, we will talk about the technical and graphical design as well as coding practices to ensure quality code. Thereafter, we will take a deep dive into the crucial parts about our product.

2 Development process

2.1 Scrumban development model

During the planning stage, we spent quite some time arguing and discussing what kind of model to use. The two options, that we could not decide between, were Scrum and Kanban. The Scrum approach offers better time management and control. It is more decisive and strict and implements more internal deadlines. Kanban on the other hand offers more flexibility, and the tasks can be selected by each member personally; which allows for easy task distribution. Kanban also offers us to plan continuously, which is definitely going to be helpful in our project, since we expected the plans to change relatively often.

We have then decided to read and research more about development methodology, and found a model that combines both advantages - the Scrumban[9] model. The main elements of our model were **the iterations** and **the project board**.

We had initially divided the project timeline into multiple week-long itera-

tions. At the start of each iteration we would discuss and choose a development focus for that iteration, then we would collect user stories, and use them to populate the project board, with issues. The project board consisted of five columns: "To do", "In progress", "To review", "Blocked" and "Done". Initially, all issues were moved to "To do", and any team member was free to choose any issue and move it to "In progress", and start working on it. One member could only have one task in the "In progress" column, in order to avoid losing focus. After completing their task, the team members would move them to the "To review" column. Tasks, that could not be completed due to some unfulfilled requirements, were to be moved to "Blocked". This ensures that essential tasks could be prioritized.

During our team meetings every Monday we would go through the "To review" column. We would test and discuss implementations, and moved the issues into "Done" if they have passed our playtests. Otherwise, they would be sent back to "To do", with necessary changes discussed.

We also had two additional project boards, one for bugfixing, and one for tests. If a feature had bugs, or if a bug was found, new issues would be added to the former board. If a feature required more testing, then we would add issues to the latter board. We would go through these boards on our Monday meetings as well.

Initially, we have planned to have week-long iterations. However, after assessing our development velocity, we have switched to two-week iterations, with the very first iteration being a whole three weeks. The reason behind this was that we found out, that one week is too little for an effective development focus. And to lay the base of the game took so much time, that even a two-week iteration was not sufficient. Still, since we used the Scrumban model, our development was dynamic, and we assessed our progress on a weekly basis anyway, adding new bugs, testing, and discussing our progress and feature implementations between ourselves and the employer.

2.2 Mapping the requirements

At the start of the development cycle, we went through our list of user stories at that moment, and put them onto a flowchart to make it easier to monitor the development cycle, as well as to see the extent of the current iteration.

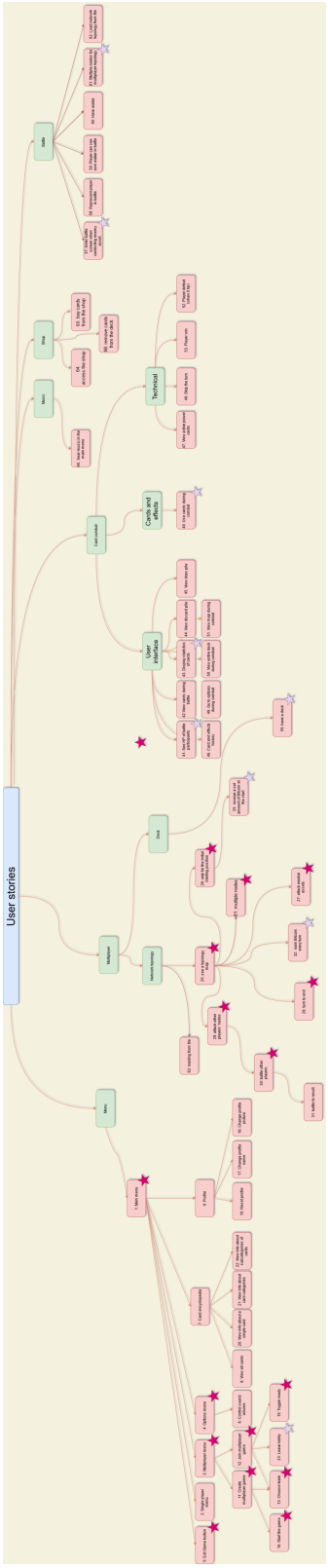


Figure 3.1: Flowchart of user stories

3 Initial technical and graphical choices

3.1 Game Engine choice

Considering there were no requirements as to which technologies we had to use and that learning how to efficiently use a game engine takes a lot of time and practice, we decided to use the Godot[11] game engine as we already had experience with it from the "Game Programming" course[6]. Other reasons for using the Godot game engine were also the following:

- It is open source
- It is lightweight, executable is around 70MB and requires no installation
- Has a dedicated scripting language with high readability, better editor integration and more straightforward optimizations for speed
- It is cross platform and exports to multiple platforms
- Good and extensive documentation
- Version control friendly, scenes are stored as text in friendly and human readable format

3.2 Version Control

As our version control tool we have used git together with GitHub [12]. We have created a private repository, and worked on the code there.

Branches

For various aspects of the game, that we were working on, we have created special branches, to make sure that the team members can work independently on a stable version without having to deal with merge conflicts too often. The branches included such aspects of the game as the GUI, the multiplayer server, the cards, special branches for the release and others. Anyone could be working on any branch, depending on what that specific team member was working on, however, after some time into the development, some branches were used more often by specific team members, than others.

Kanban Bot

To make it easier to follow the schedule, keep track of the overall progress and the progress of individual team members and individual aspects of the game, a

webhook-based bot was developed by us. The bot monitored any changes to issues in the repository, as well as any changes to the Scrumban board, and notified about them in our team Discord server. This also helped us to quickly react to any issues the team members had during the development.

3.3 System design

We have decided to build our product from the ground up using the Godot engine. We have not used external libraries or frameworks in our development.

This allows us to avoid relying on third party maintenance and support, so our product is much easier to work on in the future. It also means that the user will have an easier time running our project, since they don't have to install any third-party software.

It also is beneficial for us as programmers, since we can experience building our software completely y ourselves, which widens our expertise in different fields, such as networking, file management, and others.

3.4 Coding practices

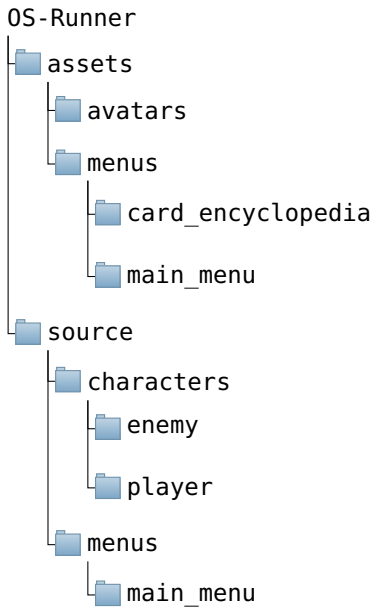
To ensure good code quality and project organization, we decided it would be very beneficial to agree on a set of rules regarding the code and file system structure at the beginning phase of the development. Since Godot has no restrictions on project or file system usage, we decided to follow the suggested workflow in the official Godot documentation [13]. In regards to coding this involves the following:

- **snake_case** for file, folder, functions and all variables except variables containing nodes
    - If a function or variable is private, an underscore will be prepended
- **PascalCase** for node names as well as class names and variables containing nodes

3.5 File structure

For the folder structure we decided on having a **source** folder and an **assets** folder. Everything that needs to be compiled into the game goes into the **source** folder and can be further sorted into their own subfolders. All assets on the other hand, are placed inside of the assets folder and furthermore sorted inside their

own subfolders. An example of how the file structure looks like would be the following:



3.6 Graphical design

In terms of graphical design, we have decided to go forward with a relatively simplistic, clean user interface. The game should look elegant, but minimalistic.

At the same time, it should be easily viewable on mobile deices, not only on desktop platforms. This is especially important in text-heavy parts of the game, such as the encyclopedia. Therefore, in the main menu and the encyclopedia we have decided to go with big buttons bright buttons and a big font. For the font choice, we have decided to us the Ubuntu font. It is a clean, modern font, that is easy to read and fits the hacking theme perfectly.

When designing cards, card battles, and the network topology map, we decided to invest into the hacking theme heavily. Still, even though as a result we went with a darker palette with more contrasting elements, we still tried to keep everything clean and relatively simple. This way, the new players will have an easier time learning the game, its elements and mechanics.

Also, when designing gameplay elements, we have decided to go with more dynamic elements. This creates a more engaging gameplay, that will draw players attention to the game more easily.

4 Server

4.1 Functionality

The server is, arguably, the main element of the code. It is used to connect the players together, allowing them to engage in a multiplayer game. It also transfers all the game data between players, making sure that everyone has the same game state at all times.

A more complete list of what the server is responsible for includes:

- Store data about the current game phase, and update this information for clients
- Store information about players, their profiles, teams, whether they are available for battle and such
- Stores and manipulates voting data during the voting phase
- Manages the shop timer
- Stores the map and asset data

4.2 Basis

The game server is built using the Godot built-in high-level networking API. This feature of the engine helped us a lot, since we could avoid having to build our own networking from ground up. The server is a scene, consisting of *Server.gd* and its child, *BattleServer.gd*. As soon as the player tries to join or host a game, the scene is created. The scene is created regardless of whether the player is the host or a client, despite what the scene name might suggest. The reason for this is how Godot API works: nodes between different players can only interact, if their names and paths are the same.

The players' local server nodes communicate between each other by using remote procedure calls. Some of the function in the server node can be executed only by the master (host) on puppets (clients), some can only be executed by puppets on master, and some can be executed by everyone on anyone. This means that we can write the same code for everyone, regardless of whether they are a server or a client. However, it also adds extra complexity, because often the node has to execute an action differently, based on whether it is a server or a client.

4.3 Structure and flow

The server follows the principle of client-server model, but with some Godot-specific intricacies. When a player hosts a game, their local server node is instantiated. It creates all the necessary data structures, that are needed for gameplay, such as the players dictionary. It also immediately registers the hosting player to that list.

When a player wants to join a game, the node is also created. It tries to connect to a server on the specified IP address. In case of a successful connection, it executes a remote procedure call to register the player on a server. From this point, most of the client-server relations follow a specific pattern.

Client-server communication

Whenever a player initiates almost any action, such as switching the team, attacking an asset, or toggling ready state[reference required], it is completed in four steps:

1. **The server node checks whether it was initiated as the server or the client**
   The server node has a *mode* variable, that stores information about this.

Code listing 3.1: Checking what mode the server is set to

```
if mode == BootMode.SERVER:
    # I am the server, I can do stuff myself

else:
    # I am the client, I have to ask the server to do stuff for me
```

2. **If the node is instantiated in server mode, it completes the action and notifies clients about it**

Code listing 3.2: Server completes the action on its own and notifies other players

```
func switch_team():
    if mode == BootMode.SERVER:
        _switch_player_team(get_own_id())

        print("The␣host␣switched␣teams,␣sending␣an␣updated␣players␣list")
        rpc("update_player_list", players)

    else:
        # This is done by the clients
```

3. **If the node is instantiated in client mode, it asks the server to complete the action in its stead**

Code listing 3.3: Client asks the server to complete an action

```
func switch_team():
    if mode == BootMode.SERVER:
        # This is done by the server
    else:
        # Do a remote procedure call to server and ask
        # him to switch the team for me
        rpc_id(SERVER_ID, "request_team_switch")
```

4. **If one of the clients asked the server to complete an action, it will do so**

Code listing 3.4: Server completes the requested action, and notifies players of it

```
# Switch player team
master func request_team_switch():

    var sender_id = get_tree().get_rpc_sender_id()
    print ("Received␣team␣switch␣call␣from:␣" + str(sender_id))

    _switch_player_team(sender_id)

    print ("Team␣switched,␣sending␣updated␣player␣list")
    rpc("update_player_list", players)
```

This approach allows us to use one and the same node as both the server and the client, without having to write any additional code. It also helps us to keep the hierarchy and make the server responsible for completing actions and notifying the players instead of having the clients notify each other. This prevents the data from being desynchronized, since only the server has the authority to execute most of the logic.

This approach, however, has one notable shortcoming. It forces us to repeat most of the code twice: once in the if statement for the server and once in the remote procedure call.

5 Cards

5.1 Introduction

As explained earlier in Section 2 of Chapter I, the main mechanics revolve around card battle gameplay and deck-building as well as red teaming and penetration testing being the main theme of the game. The cards are therefore the

cornerstone of OS Runners gameplay, but also the educational aspect of OS Runner. The cards are based off of tactics and techniques from the Mitre ATT&CK framework [1], but their functionality have been simplified to make for easier learning.

The player starts the game with a default deck and throughout the game they are able to change it however they like. Changes to the starting deck can occur in the form of buying new cards or removing cards already in the deck (see Section 7.1). This forces the player to themselves evaluate the strengths and weaknesses of the different cards and figure out what works best with the cards they currently have. After having read and understood the card, the player should also have an underlying idea of the what the card is based on in real life.

5.2 Card base

Every card in the game has the same common attributes. Those attributes are presented in table 3.1:

Categories and subcategories are based on a list provided by our employer. He used cybersecurity frameworks to classify different techniques and therefore cards. Here is the full list of categories and subcategories that we have:

- **Attack**
  - Web
  - Networking
  - Wireless
  - System
  - Social Engineering
  - Physical
- **Malware**
  - Trojan
  - Backdoor
  - Ransomware
  - Adware
  - Cryptominers
  - Exploit-kit
  - Worm
  - Virus
  - Spyware
  - Scareware
  - Rootkit
  - Logic Bomb

Property	Description
ID	the id of the card. This is only used internally, as all the current game cards are stored in a hash map with the ID being the primary attribute
Name	name of the card. In our case the card names will be the equivalent of the real life hacking and tricking techniques
Description	a very short text telling the user slightly more about the technique. This attribute isn't strictly necessary for the cards, but we believed it might add flavor to the game and make players more interested the techniques themselves. This description is shown on the cards themselves, so the player will always be able to read them
Long description	detailed descriptions of the real life techniques. This description will only be shown in the encyclopedia (see Section 8). It is supposed to educate players on how the techniques work in real life and redirect to sites with even more information
Card rarity	used to estimate the power of a card. The rarer the card, the more powerful it is supposed to be. Rarity also dictates how frequently a card will be displayed in the shop
Price	decides how much the card will cost in the shop. More about the shop in Section 7.1
Category	determines from what field this card originates from. If the card for example generates money for the player, it could be placed in the *social engineering* category, while other more harmful cards could be placed in the *malware* category
Subcategory	a more specific type of a category. Malware is too general term since there are very many types of malware. If the category is malware, the subcategory could for example be *trojan*
Image	a visual representation that will help players easily identify the card and give a better idea of how the technique of the card is performed
Effect	card will do in the game. Effects will be more discussed in the subsection 5.5

Table 3.1: All properties for cards.

- **Defense**
  - Defense

5.3 Design

Old design

During the development we had two major designs for how the cards should look like. Figure 3.2 is our first design of the card. This design was very simple but after some play testing we realized that it didn't display what the card actually does. Even though the first design was lacking, it helped a lot in getting a feel of how the displaying and dragging of the cards will work later in the development. This design was expanded and improved upon later in the development.

Figure 3.2: Initial design of the cards

Description of the figure 3.2:

1. Category of the card
2. Image of the card

3. Name of the card
4. Description of the card

Current design

Our current design focuses more on how the card looks like visually. We added the border to the cards to make them more visually appealing. This design also includes what effects the card contains, more about effects in Section 5.5. We wanted the effects to be separated from the rest of the text with a white background to highlight the most important part of the card. Figure 3.3 shows the result.

Figure 3.3: How a card currently looks like in the game

5.4 Technical implementation

Card is a scene which contains all of the GUI elements of the card. To make the text look clean we use a *VBoxContainer* that equally separates text from each other making the text easy to read.

The card's most important role is to make sure that the correct data is being displayed at all times. Every time a card is being played, placed, displayed or

changed, the card will update its visual data. The card also include several borders which change based on the category of the card. Listing 3.5 displays the updating of the cards.

Code listing 3.5: Updating of the card

```
func update_card_gui():
        name_label.text = card_name
        type_label.text = category
        subcategorylabel.text = subcategory

        match category:
                "Attack":
                        $Background.texture =
                        preload("res://assets/card/background_attack.png")
                "Malware":
                        $Background.texture =
                        preload("res://assets/card/background_attack.png")
                "Defence":
                        $Background.texture =
                        preload("res://assets/card/background_defence.png")
                "Normal":
                        $Background.texture =
                        preload("res://assets/card/background_normal.png")

        description_label.text = card_description
        card_image.texture = saved_texture
        var effects_string: String = ""
        for effect in effect_container.get_children():
                effects_string += effect.effect_name + "␣:␣" +
                str(effect.power) + '\n'
        $VBoxContainer/VBoxContainer/CardEffects/
        MarginContainer/CardEffectsLabel.text = effects_string
        show_on_top = true
```

5.5 Effects

Each card in the game can deal damage, apply status or heal the player. We have decided to call every ability of a card an *effect*. There are no two effects that are the same.

All cards have a node called *effect_container*. Every effect of a card will be a child of this node. If a card is supposed to have an effect that deals damage to the enemy player, the effect_container would have one child called *damage_effect*.

Code listing 3.6: Adding effects to effects container

```
func add_effect(effect_name : String, effect_power : int,
    effect_parameters: Dictionary, new_effect_texture: Texture):
    var effect = load("res://source/cards/effects/" + effect_name + ".tscn")
    effect = effect.instance() as BaseEffect

    effect.effect_texture = new_effect_texture

    if effect_power:
        effect.set_power(effect_power)

    if effect_parameters.size():
        effect.add_additional_parameters(effect_parameters)

    effect_container.add_child(effect)

    update_card_gui()
```

Each effect in the game inherits from the abstract class *base_effect*. This abstract class has three variables: power, type and name.

- Name dictates what will be shown on the card itself in the effects field.
- Power dictates how powerful an effect will be. Damage dealing effect with power five will deal five damage to the opponent.
- Type dictates whether the effect will be positive or negative. Effects of type *offensive* will hurt the enemy and effects of type *supportive* will benefit the player.

The effects exist as classes. When adding a new effect to the card, it must be a predefined effect. There is currently no way for the player to change or create new effects in the game. All of the effects are described in table 3.2.

Effect	**Description**
Deal damage	Deals damage to the opponent
Bomb	Deals damage after some amount of turns to the opponent
Heal	Heals the player
Add bitcoin	Adds bitcoin to player's account
Poison	Deals damage every turn
Protection	Protects you from specified type of cards
Spam	Gives the opponent player cards that do nothing
Corruption	Replaces some of the opponents cards with cards that do nothing. Once played the card returns to its original state
Reveal	Shows some of the cards in the opponents deck

Table 3.2: All effects for cards.

5.6 Card manager

Card manager is an essential node which enables the game to have playable cards. The primary role of this node is to store all of the cards in the game, distribute it between players and shop. Another important role is to keep track of the cards the player has. More about this in Section 6.2.

Reading cards from JSON

During planning phase, the employer put emphasis on importance of easily adding, removing and changing cards without needing any coding experience. To fulfill this requirement, we decided to place all of the cards in the game in a JSON file. This way we can easily add, remove and change cards from the game. As mentioned earlier, players can only add already existing effects to the cards through the JSON file. An example of a card in the JSON file is shown in listing 3.7.

Code listing 3.7: Example of card stored in JSON file

```
{
        "name": "Power outage",
        "description": "Just plug your competitor’s computer out of the outlet",
        "card_rarity": 3,
        "effects": [{
                "name": "damage_effect",
                "power": 25
        }],
        "price": 1000,
        "category": "Attack",
        "subcategory": "Physical Attack",
        "image": "no_power.png"
        "long_description": "Historically, the most efficient way of hacking."
}
```

When creating a new card, we first instance a new empty card scene. Basic information like name and rarity is copied directly into the card. Fields that require more attention are *image* and *effects* fields. Listing 3.8 shows how effects and image are being added to a card. Adding of the basic information is not included in the listing.

Code listing 3.8: Adding a card to the game

```
var base_card = preload("res://source/cards/card_base/card_base.tscn")
base_card = base_card.instance()
add_child(base_card)

var texture = load("res://assets/card/" + card["image"])
base_card.saved_texture = texture

var effect_parameters = {}

for effect in card["effects"]:

        var effect_name = effect["name"]
        var effect_power

        effect.erase("name")

        if effect.has("power"):
                effect_power = effect["power"]

        effect.erase("power")

        base_card.add_effect(effect_name, effect_power, effect, texture)

base_card.update_card_gui()

all_game_cards[base_card.id] = base_card
total_amount_of_cards += 1
remove_child(base_card)
```

In listing 3.8 it is important that we are first adding the card as a child to the main root scene. To apply any changes to properties of an object in Godot, we need to add them to the scene tree first. Trying to interact with a texture of a node, while this node was not yet added to a scene tree, will result in an error, because the texture will be *null*. When we read the textures for an image of the card, we assume that the image is already included in the files and can be easily accessed.

We add the effects to the cards through JSON. Looking back at code listing 3.7, we can see field called "effects". This field is an array because a card can have more than one effect. First we write the name of the effect and follow it with the power of the effect. There are effects that require more than just name and power to function properly. Listing 3.9 shows how an effect with more than power would be parsed into the JSON file. The only difference in this example is the duration field. Other specialized effects might swap duration field with for another keyword more fitting for the effect.

Code listing 3.9: A unique effect with more than power and name

```
"effects": [{
        "name": "bomb_effect",
        "power": 40,
        "duration": 3
}],
```

Cardmanager has a dictionary called *all_game_cards*. As the cards are being read from the JSON file, they are being added to this container. It will store all the original cards in the game, including the unobtainable cards. None of the cards in this dictionary should be changed throughout the entirety of the playthrough. Changing them will affect all of the future cards of the same type in the game.

5.7 Deck

This section will focus on the *deck* nodes and their purpose of handling the flow of cards between each other. These nodes include: *deck*, *hand*, *drawPile*, *discardPile* and *deck base*, with the latter being an abstract class for the others. The *deck* class acts as a controller and handles the interactions between the draw pile, discard pile and hand, as well as keeping track of the cards in the player's deck.

Deck_base

As mentioned earlier, the classes for draw pile, discard pile, hand and deck inherits from the abstract *deck_base* class. This is because they share the same functionality for adding cards, removing cards, returning cards, resetting and shuffling of the *current_cards* array. This array keeps track of cards in deck or the piles. In figure 3.4 we visualized the inheritance of the abstract class.

The deck class

To better understand the reason for our implementation, we need to explain how playing cards works in the game. When the battle starts, the player will receive cards from *the deck* into their *hand*. The player will then play cards from their hand. When a card is being played, it is removed from the hand pile and is being sent into the discard pile. The player immediately draws a card from the draw pile to their hand pile. This process continues until the player's draw pile is empty and they can no longer draw any cards. When that happens, the cards from the discard pile are being shuffled back into the players draw pile and the player draws a card from the draw pile.

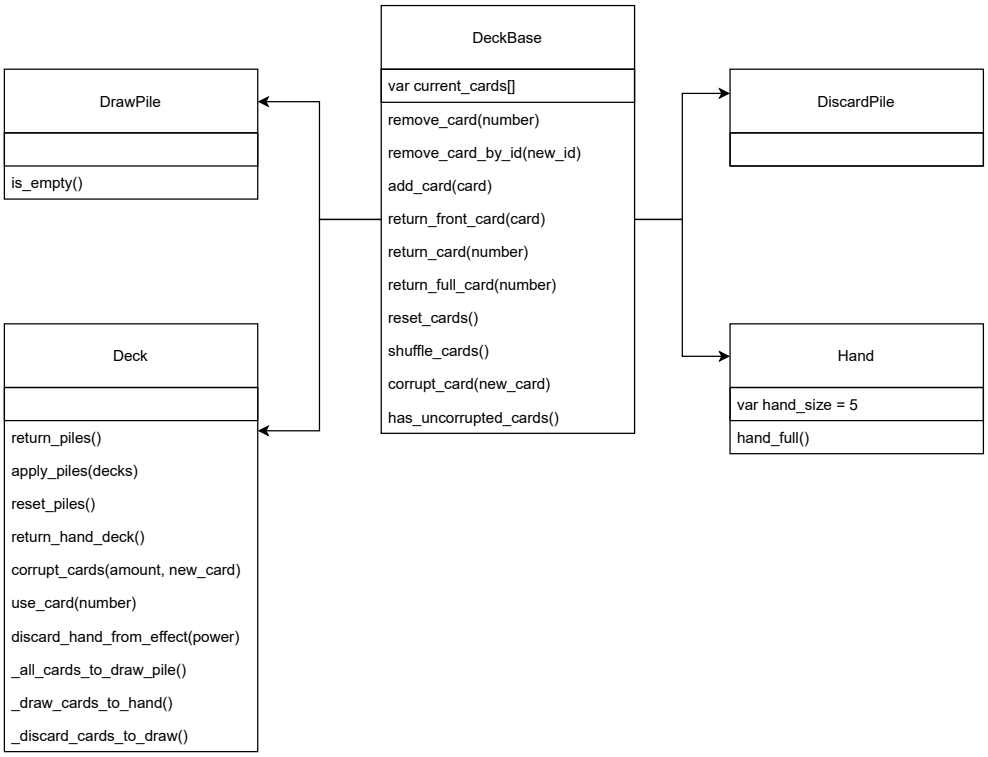


Figure 3.4: Inheritance from deck_base

With the description of how the card flow looks like, we can now go into more detail what each component does, starting with the *deck*. The deck has two primary roles, the first one being a manager for the flow of the cards. To manage the flow of the cards between piles, the piles must be children of the deck. When player uses a card, the hand sends the played card to the deck, and the deck then sends the card to the discard pile. Figure 3.5 shows how the deck and the piles are related

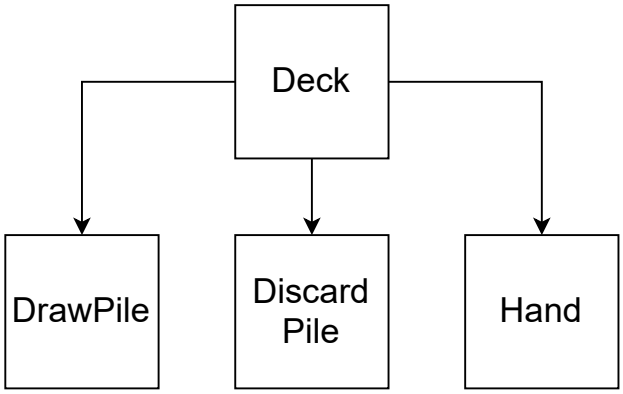


Figure 3.5: Relation between the deck and piles

The second primary role of the deck is to keep track of the cards that the player currently has in the game. At the start of the game, the player starts with basic cards and is supposed to add and remove cards from their deck to make them stronger. Every time a battle starts, the cards from the deck will be sent to the piles and the player will be able to use them. When the battle ends, cards in the piles will be removed. This means that any changes to the cards in the deck applied between the battles will apply to player's next combat.

Another feature for the deck is the ability to return and apply custom piles. This is is primarily used to show opponent's deck during a battle when a revealing effect has been played. More about it in Section 6.2.

Pile classes

As previously mentioned, there are three piles in the game, each has it's own role. The draw piles primary role is to constantly check if it's empty. If it is empty, it will request the deck node to give cards from the discard pile. The discard pile in itself has no extra functions. It is just supposed to store all of the cards that the player has used. The hand pile's special attribute is the maximum amount of cards in the container. Having more available cards to play gives for more opportunities to win against the opponent.

5.8 Pile display

During the game, it is important to preview all of the cards that the player has in their deck. For this reason we have decided to create a new node and called it *PileDisplay*. Its role is to display all of the cards in a current deck class node.

At first we had no idea what its design should be, we therefore decided to look to other sources. Most of the team members have played *Monster Train* (Shiny Shoe, 2020) [4], which is also a rogue like deck building game. We believed the deck preview they have created is simple yet intuitive and decided to take inspiration from it. Figure 3.6 illustrates how PileDisplay looks like currently in the game.

During the process of creating pile display, the shop has been worked upon at the same time, and we stumbled upon the problem: "How do we display the cards that we want to remove from our deck in the shop?". This is when we returned to Monster Train and noticed that they are using the same display for removing cards in the shop as when displaying cards in a pile. This led us to the conclusion that PileDisplay should display any deck class pile in the game.

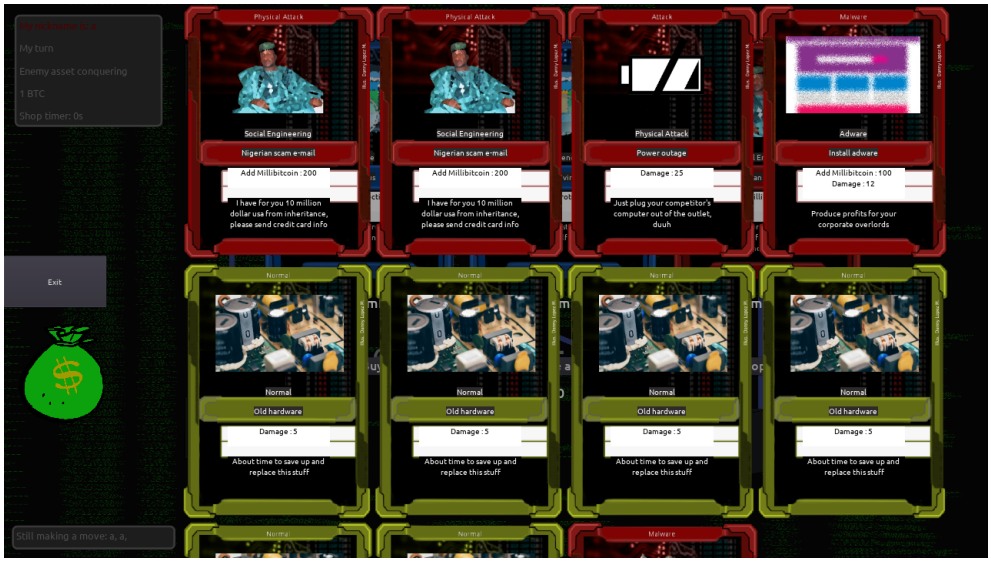


Figure 3.6: PileDisplay in the shop

6 Battle Manager

The battle manager is a node that is the most essential part of the battles between players. It controls the cards the player can use during a combat, how much health points each player has during combat and most importantly, who is

going to win the game. Battle manager also has the important role of showing the user all the essential information about the battle itself.

In this section, we will be talking about how it evolved design and implementation wise, describe all the things it does by itself and with combination of other nodes.

6.1 Design

Throughout the planning phase, battle manager went through many iterations of visual designs. While creating the design, we were focusing on what the player actually has to see in order to properly understand the battle situation. A player must be able to see the following:

- Player's current amount of HP
- Enemy's current amount of HP
- Enemy's nickname
- Player's cards
- Player's draw pile
- Player's discard pile
- Enemy's cards
- Enemy's draw pile
- Enemy's discard pile
- status effect
- Protection effects

These properties have to be displayed to the player during a combat, and yet it can not be cluttered. We wanted to make it intuitive for the player where everything is placed and how to read what is shown on the screen. During our brainstorming sessions we came up with a design that we believed would fit all of the criteria (see figure 3.7).

1. Discard pile of the enemy
2. Draw pile of the enemy
3. The cards of the enemy
4. Powers of the enemy
5. Avatar and health of the enemy

In this design, the GUI for the player is mirrored with the GUI of the enemy, that means the the player's cards, piles and health is at the bottom of the screen.

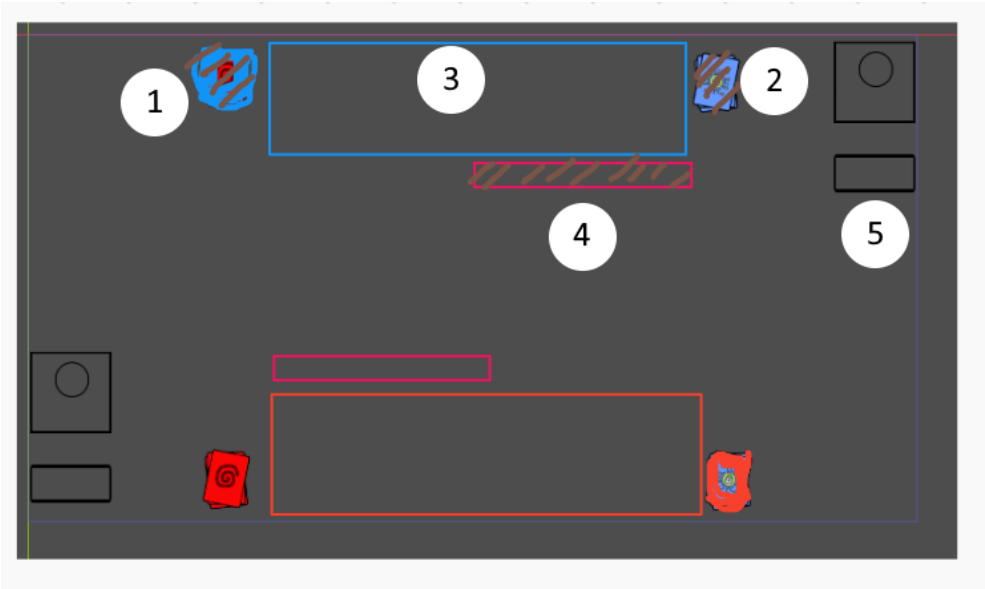


Figure 3.7: Original design of battle screen

Current design

As simple as this visual design is, we struggled to recreate it and during the development, some visuals have changed too. The result was figure 3.8. We made everything much bigger. We kept the positions of most elements intact. A huge change was the card frames in the middle of the screen. From those card frames you could see the latest card the player and his enemy has played.

Another difference is the lack of proper power implementation. The powers were supposed to play an important role in making the player stronger, but the design of them was never fully discussed. The game currently features protections against certain types of attacks. Initially they were supposed to be placed in the same field as powers, but due lack of time the visual representation is just a small texture of the card image placed at the top left of the screen.

Minor bug

During the development of map topology (more in Section 7), we have added background to make the game feel more interesting. It was supposed to stay only within the map topology screen, but due to some error it wasn't removed properly. This caused the background to be visible during battle as well. We have decided to let it stay this way as it was more interesting than a blank screen. Figure 3.9 shows the result.

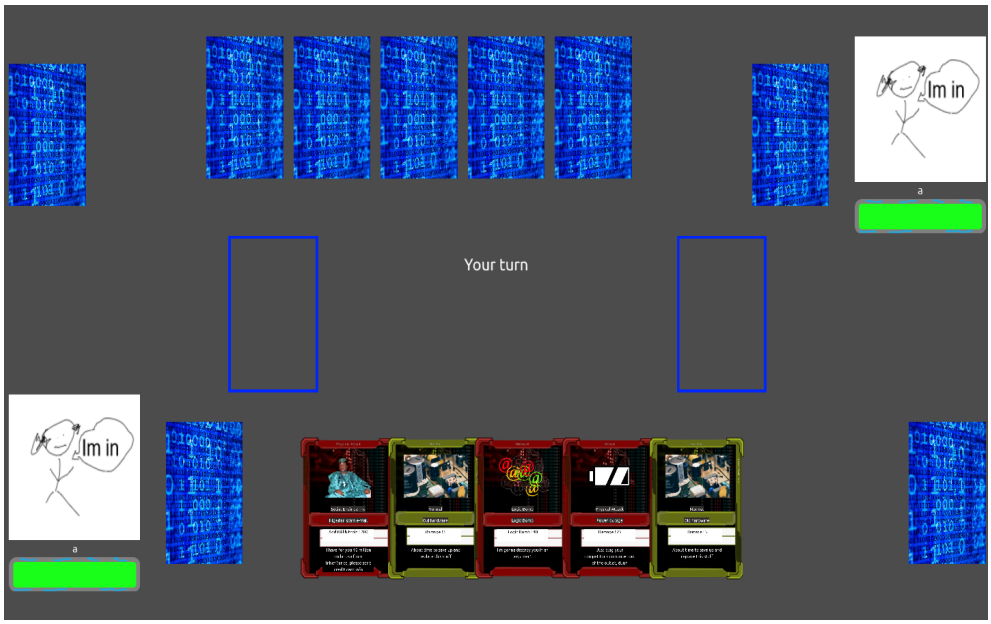


Figure 3.8: Current design of the BattleManager

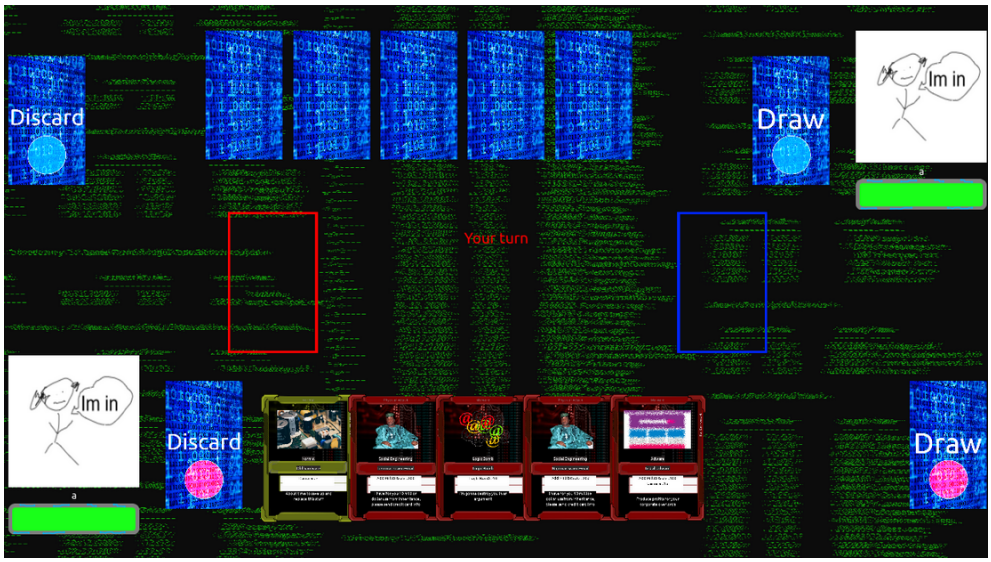


Figure 3.9: Battle screen with the background bug

Future designs

There is still a lot of possible improvements for how the battle screen could look like, and we are constantly coming with more ideas. The latest design proposed by our employer showed in figure 3.10 is how the game could potentially look in the future. It builds on top of the current design and improves it without altering it too much.

Figure 3.10: How battle screen could look like in future

The immediate visible changes are the sizes of most elements. The cards are much smaller. The health bars of players are smaller too and player's health bar is now placed at the top left instead of bottom left. At the middle of the screen there is a clean looking background indicating where the player is supposed to place cards.

A new feature included in this design is a new preview of the cards. Instead of hovering over the card and enlarging it to make it readable, we place a bigger version of the cards on the left side of the screen. This way the player always knows where to look if he wants to read a card and its properties.

In this design there is also a field for powers. There are arrows pointing from player's powers to the opponent's, this is not indicative of how it will look like, rather it's a sketch on how the powers could interact with each other in the future. Same applies to having multiple of the same powers.

6.2 Player

Our battle manager contains a node called *Player* that was originally supposed to store all the information like cards in player's deck and his health points. We quickly noticed that while in the map screen, we would remove the battle manager node from the main tree scene because it was unnecessary. This caused the player's deck to be constantly deleted, removing the deck building part of the game. To counteract this, we have decided to put the player's deck inside of the card manager in the node called *Deck*, more about deck in Section 5.7.

Currently, the *Player* node keeps track of player's status effects, protections and most importantly health. Whenever the opponent plays a card, they will directly affect that node. This node also decides how powerful the poison should be. Whenever the player has a protection against some type of card, this node will stop its effects.

Battle manager contains a node called *Enemy*. Initially in the development every card played by the player would affect this node and then overwrite the information in the opponent's *Player* node. During further discussion, we came to the conclusion that it's not good because this allows for cheating to occur. If player would edit the cards before the game started, they could instantly win all the battles without the opposing player knowing what just happened. Therefore we have switched to a new method that will be described in Section 6.4.

6.3 Cards

Displaying cards

During the development, to make the cards look clean, we wanted to separate them using a *HBoxContainer*. This didn't work well because *HBoxContainer* automatically resizes itself to the smallest possible size. Children of that node could move only within the boundaries of its parent, thus limiting the amount of space where the cards could be dragged. We circumvented this by creating an empty node called *_hand_pile_GUI* which was of the size of whole screen. This way we could drag the children across the whole screen. This method also gives us the freedom of choosing the exact position where the cards should be placed and how much space there should be between each card. Potential problem with this approach is the difficulty of changing how many cards can be displayed at the same time. Currently it is meant to display five cards at any time. Showing six or more cards could make some of the buttons unclickable.

Because the cards can be displayed in multiple places, we had to find a solution for when the cards are supposed to have certain properties. The most important

one for displaying the cards in the battle manager was the enlarging. When player hovers over a card, the card is supposed to be enlarged so that the player can read it. When we add the cards to *_hand_pile_GUI*, we set a boolean called *enlarging* and specify its original position. When player hovers over a card with enlarging property, it will scale up to double the size. When the cursor leaves the cards, it will go back to its original size and position. Code listing 3.10 shows the entire process of showing a card in battle manager.

Code listing 3.10: Displaying of cards

```
func display_cards():
        var i = 0
        for card in hand:
                card = card.duplicate()
                card.enlarging = true
                card.dragable = true

                var additional_space = i * 170

                var screen_size = Vector2(1920, 1080)

                card.set_position(Vector2(screen_size.x * 0.3
                + additional_space, screen_size.y * 0.7))

                card.initial_position = Vector2(card.get_position())
                card.set_scale(Vector2(0.5, 0.5))
                card.initial_scale = card.get_scale()
                card.connect("card_dragged_to_play",
                self, "_on_card_dragged_to_play")

                _hand_pile_GUI.add_child(card)
                card.update_card_gui()

            i += 1
```

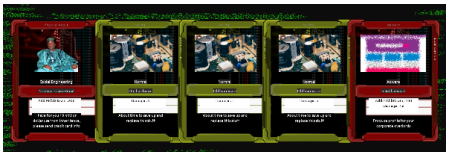
The enemy also has cards that are playable. During the planning we knew that there will be some cards in the game that will enable player to look at the cards of the enemy. We have decided to just insert textures for the enemy's card. While the cards are not revealed, the player will see backside of the cards. When a reveal card has been played, it will reveal a card that the enemy has in their hand. Figure 3.11c shows how it is visualized in the game.

Dragging cards

Like with previous approach, dragging cards works similarly to the enlarging of cards. We have a boolean called *dragable* which decides whether the player should be able to drag a card or not. This has been used only in battle manager.

When the player presses and holds LMB, he will be able to drag the cards. The card will be centered on the middle of the mouse. The player can freely move

(a) Before the cards are revealed



(b) Enemy's current cards at hand

(c) Enemy's cards revealed for the player

Figure 3.11: Revealing of enemy's cards

the card across the whole screen. Upon releasing LMB, the card will return to its original position, unless the card is higher than the middle of the screen. If the card is placed somewhere above the middle of the screen, the card will be played.

6.4 Effects in battle

Battle server

Before talking about how the effects are applied to player, it is important to talk about the communication between two clients. Battle server is the connector between the players battling each other. Its primary role is to make sure that the data is being updated for both players. Battle server makes sure that it is the correct persons turn, sends the id's of the cards that are being played and ends the battles when one of the players is inflicted lethal damage.

Using of the cards

This part went through two major iterations. The first iteration as mentioned earlier in Section 6.2 was supposed to make changes to the enemy node within the battle manager. That means every effect in the game would affect that node and all the data about the enemy node would be sent and applied to the other player's *player* node. We found it more secure to simply send the cards themselves through the battle server to the other player and make that change directly on the other client. This would also make cheating through editing cards impossible. If the player would do that, the edited cards would work against them.

The next two listings and paragraphs will try to explain in detail the process behind using cards.

Code listing 3.11: Using a card

```
func use_card(number : int):

        if !_BattleServer.my_turn:
                return

        var received_id = CardManager.Deck.use_card(number)
        var card = CardManager.return_card(received_id)

        _PlayerCardFrame.put_card(card)

        var supportive_effects = card.return_supportive_effects()

        for effect in supportive_effects:
                effect.run(_player)}]

        _BattleServer.send_health(_player.health)
        _BattleServer.send_card_id(received_id) # Important

        update_health_bars()
        display_cards()
        _display_enemy_cards()

        var players_deck = _return_piles()
        _BattleServer.get_decks(players_deck)
        _BattleServer.switch_turn()
        _TurnLabel.toggle_state()
```

At first we check if it's players turn. If it is, we get the id of a card and the card itself. Then we display the image of a card in the appropriate card frame. As mentioned earlier in Section 5.5, every effect has a type accompanied by it. If the effects are of the supportive type, they will not be trigger for the enemy, they will only trigger for the player using that card. Next is when the networking happens. We send the HP of the player to the enemy so in case he used any healing cards, those changes will be visible for the other player too. The line with important as a comment will be described in more detail soon. When the effect is used and the card ID sent, the battle manager will update all of the GUI elements on the screen and send player's cards to the enemy's battle manager. The turn will switch so the enemy can make his own turn.

Code listing 3.12: Receiving a card

```
func _on_card_id_received(card_id):

    var card: CardBase = CardManager.return_card(card_id)

    if card.category == "Defence":
            card.flip()

    _EnemyCardFrame.put_card(card)

    if _player.PManager.has_active_by_name(card.category):
            return

    if _player.PManager.has_active_by_name(card.category + "␣Protection"):
            return

    var offensive_effects = card.return_offensive_effects()

    for effect in offensive_effects:
            effect.run(_player)


    update_health_bars()
    _BattleServer.send_health(_player.health)
    display_cards()

    if _player.is_dead():
            _BattleServer.lose_battle()
            _EndBattlePanel.battle_won = false
            _end_battle()
```

The line with important in listing 3.11 triggers this code for the enemy's battle manger. Important to note that we are sending just the ID of the card and not the entire object. This is because Godot doesn't allow to send objects through network. At first we get the real card from the ID we got. If the card is supposed to give protection to the original player, the card will be hidden from the opponent. We place the card that was sent to its right card frame to show the card enemy has played. After that we check for protections against a certain type of attack. If the enemy player is not protected then we will run all of the effects power.

To make implementation of effects easier, we have decided to let every effect run by itself rather than let battle manager have access to all of the effects. Each effect when ran triggers method in another class. Listing 3.13 shows how this process will approximately look like for most of the effects.

Code listing 3.13: Bomb effect

```
# This runs in effect
func run(player: Player):
    player.plant_bomb(duration, power)

    # This runs in player
    func plant_bomb(duration, power):
    _logic_bombs.append([duration, power])
```

The entire process of playing cards can be summarized in figure 3.12.

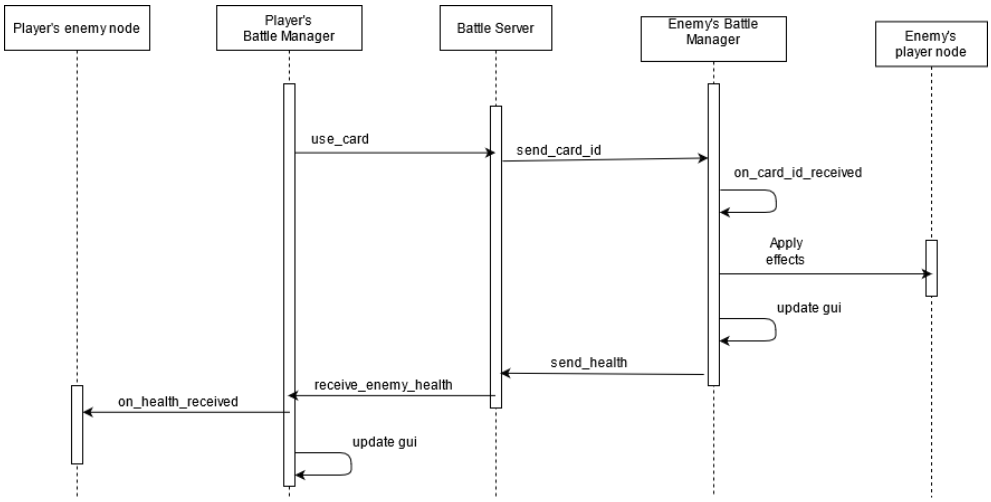


Figure 3.12: Flow of using cards

GUI

Every time a player plays a card, there is some change in the battle, and those changes must be visible to the player himself. The most obvious change is to player's hp. Whenever a player takes damage his health bar is updated accordingly.

(a) Player's health bar when he is healthy

(b) Player's health bar when he is damaged

(c) Player's health bar when he is in critical state

Figure 3.13: Representation of different states of player's health

6.5 Bugs during development

Creating battle manager required a lot of effort and learning. There were many things we didn't know when we first made it. One such situation would be how we have to handle the transactions of cards.

Overwriting cards

The corruption effect in the game is supposed to replace the card in the enemy's deck with a useless one that the enemy has to play to get his original card back. When we first started working on it, we didn't duplicate the cards properly. When one of the players used that effect, the changes were applied to the card in the opponents deck and to the cards in the card manager. That means all of the cards in card manager were replaced with copies of useless cards rendering the game unplayable.

7 Map Topology

Where the player starts and which assets they can conquer is all dependent on the map topology. The idea behind the map topology was to have it resemble a network which consists of a number of assets that are connected to each other. A subset of assets are what we consider as starting assets and they are valid starting points at the beginning of the game. The map topology is also where the players will work together as a team and strategically choose and conquer their desired assets.

Currently OS Runner features *one* map which can be seen in figure 3.14. In the map there are a total of 14 assets which needs to be conquered entirely by **one** team in order to win. The current map has a total of *six* starting assets which are placed on the edges. As mentioned earlier, some of the assets are connected to each other. The connection between assets is represented by a blue line which is animated with binary numbers which move from one asset to another connected asset.

After starting assets has been decided and the two teams begin conquering assets (as seen in figure 3.15), the conquered assets will be painted with the team's color and assets in the process of being conquered will be painted with both teams colors and have an animation to further signify this. Neutral assets on the other hand will remain uncolored.

When looking at the map topology, in the bottom left, all players can see which

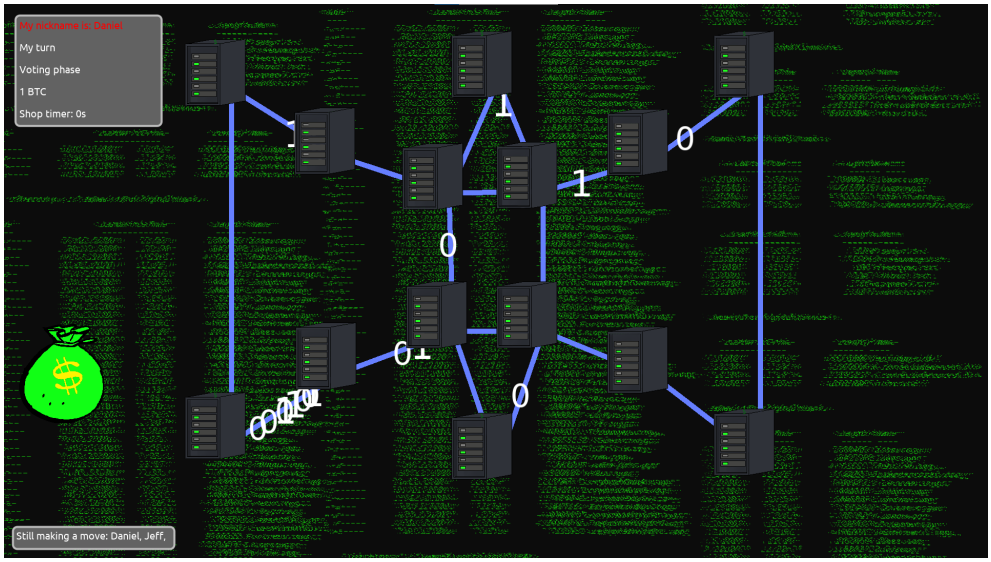


Figure 3.14: Map at start of game

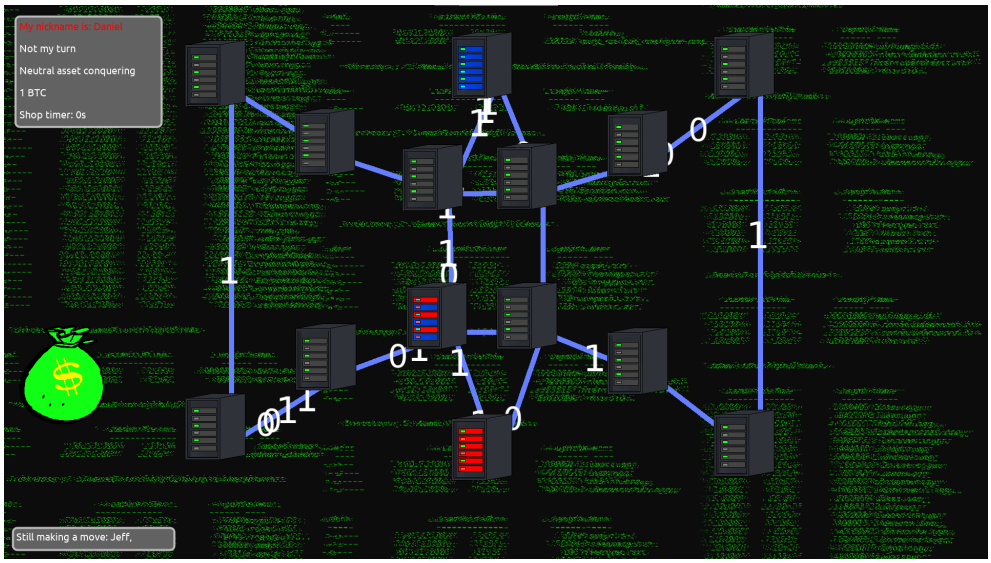


Figure 3.15: Map when conquering assets

player(s) that are still making a move during the current turn. At the top left however, all players are able to see their name and which team they are on. They can also see if it is their turn as well as which phase the game is in, their current amount of Bitcoin and the shop timer which prevents the players from denying other players the ability to update their decks in the shop by disallowing them to attack each other for some period of time.

Controller

The Controller node is there to make the map actually function correctly and respond to actions made by the player, or to new information about the game state sent by the server, and display them on the map.

The main idea behind this node is to ease the process of creating new maps. Instead of programming the logic in the map node itself, we instead decided to move it into a separate node. This way, if one more map was to be created, the developer wouldn't have to program all the interactions with the server again, but instead would just add this node.

This node is responsible for such aspects of the map as responding to player clicks, keeping track of what team is controlling each asset, hiding or updating the map when necessary, and managing the game over and error screens by responding to signals. The node also manages some visual aspects, such as the lines between assets and the moving numbers on them.

7.1 Shop

Currently the only way of acquiring new cards is through the shop when out of battle and in the map. When in the shop, the player also has the option of removing cards from deck and restocking the shop (see figure 3.16). All of these options cost Bitcoin and can only be done if the player possesses a sufficient amount of Bitcoin. However, the player is restricted to only be able to buy a new card or remove a card once per turn.

When entering the shop, by clicking the money bag in map, the player will be presented with *three* cards. These cards are randomly chosen based on their rarity. Beneath the card itself is the price of the card in Bitcoin. To buy a card the player has to select the card they want to buy, by clicking it, and press the **Buy a new card** button in the shop. The player will thereafter be charged Bitcoin equal to the price of the card and the card is removed from shop and added to the player's deck.

To remove a card from their deck, the player has to press the **Remove a card** button in the shop menu. After the button has been pressed, the player will see their own deck on screen. To remove a card, the player simply has to click on the card they want to remove. The chosen card is then removed, Bitcoin is charged and the price for removing cards is increased.

Restocking of the shop, allows the player to spend Bitcoin in order to view a new set of cards in the shop. These new cards are also randomly chosen based on rarity and they are not guaranteed to be different than the previous set of cards

Figure 3.16: Shop window in game

in the shop. The player can restock as many times as they want, this includes per turn as well, but the price also increases each time.

8 Card Encyclopedia

A natural part of all card games is being able to discover and explore the different cards available. The idea of having an in-game encyclopedia was therefore quite enticing as it further improves on the educational aspect by allowing us to further describe the cards. It allows for full exploration of all cards in the game, even those that are unobtainable, and the ability of filtering based on rarity, sort by price or simply searching. The in-game encyclopedia also includes a longer more in-depth description of the card and it's effect in the real world.

The in-game card encyclopedia can be accessed from the main menu through the "card encyclopedia" button. When entering the card encyclopedia, the user will be greeted with what can be seen in figure 3.17.

To the right in the card encyclopedia, the currently chosen card is displayed (defaults to first in list) exactly like it is whilst in battle. In the middle, the name of the card along with it's longer description is shown. The purpose of the longer description is to give a more in-depth explanation as to what the card is derived and based on in real life and educate the user.

To the left in figure 3.17 is a scrollable list which contains all cards based on

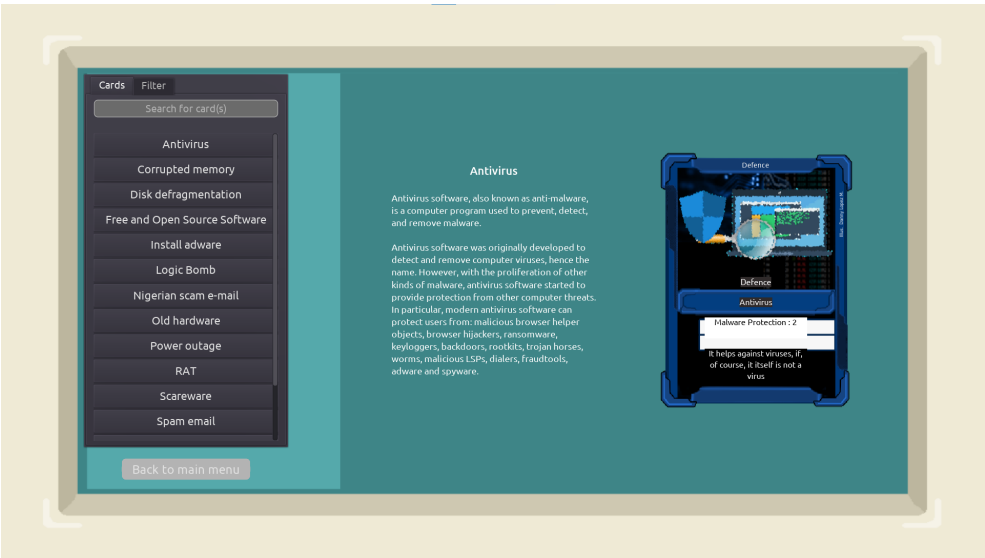


Figure 3.17: Card encyclopedia on cards tab

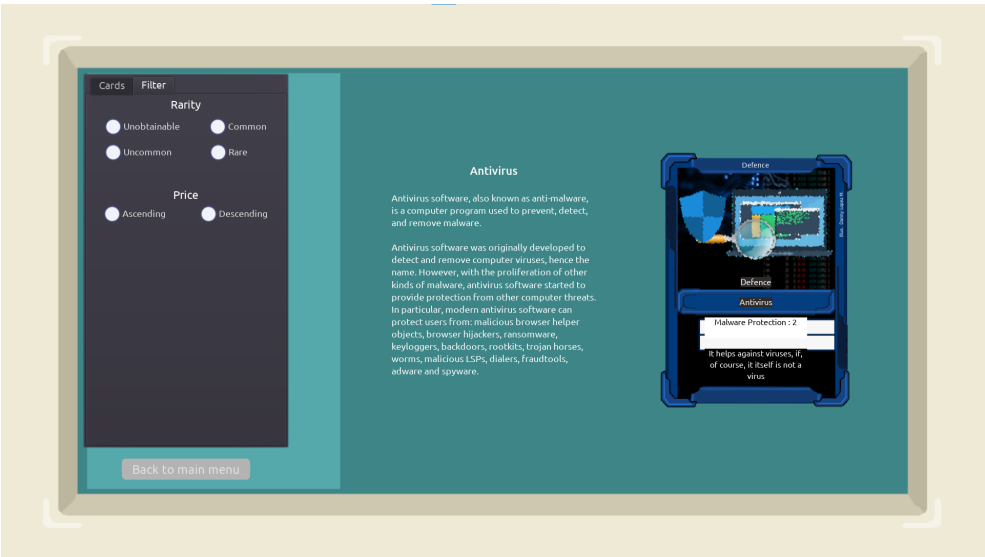


Figure 3.18: Card encyclopedia on filter tab

current filters. Above the scrollable list there is a search field for ease of access to a specific card in mind. For access to filters the user has to click on the "filter" tab which leads them to what can be seen in figure 3.18.

In figure 3.18 we can to the left, where the scrollable list was, see the available filters. These filters lets the user as mentioned earlier only have cards of chosen rarity shown as well as sorting them either ascending or descending based on price which is done using the mergesort[14] algorithm (see code listing 3.14 and

3.15). If no rarity filter is set, all cards will be shown and if neither ascending or descending is checked, they will be sorted alphabetically using GDScript's own sort method[15] for arrays.

Code listing 3.14: Implementation of mergesort algorithm used in card encyclopedia

```
# Performs mergesort based on price
func _mergesort_price(array):
	if array.size() == 1:
		return array

	var a = array.slice(0, array.size()/2-1)
	var b = array.slice(array.size()/2, array.size())

	a = _mergesort_price(a)
	b = _mergesort_price(b)

	return _merge(a, b)
```

Code listing 3.15: Merge function used in implemented mergesort algorithm

```
# Merges arrays "a" and "b" based on price
# and returns the merged resulting array
func _merge(a, b):
	var c = []

	while not a.empty() and not b.empty():
		if CardManager.return_card_by_name(a.front()).price <
	    CardManager.return_card_by_name(b.front()).price:
			c.append(b.pop_front())
		else:
			c.append(a.pop_front())

	while not a.empty():
		c.append(a.pop_front())

	while not b.empty():
		c.append(b.pop_front())

	return c
```

The reasoning behind the choice for implementing mergesort[14] was to have an efficient, consistently fast and stable sorting algorithm. The only drawback being the high use of n memory. Considering it uses the *divide and conquer* concept, it is highly capable of parallelism and has the potential for further optimization in terms of compute speed. In terms of optimizing the memory usage, the implementation can be changed to instead of using an *external* sort method to have an *in-place* sort method.

9 Music Manager

The music manager is a node, that runs as soon as the game starts. This node controls the music that plays during the gameplay. It uses Godot's built-in audio system to play the tracks.

A special element of our Music Manager is the activity system. Tracks, that the Music Manager will be playing, are listed in a JSON file, where they also are divided into activities. Currently, there are three of them:

- **Menu music**
  These tracks are supposed to be played in menus
- **Map music**
  These tracks are supposed to be played in the map topology
- **Battle music**
  These tracks are supposed to be played in a battle

By using the *switch_activity()* function, the activity can be changed. The Music Manager will then automatically choose a random track belonging to that activity, and play it. When the track ends, it automatically restarts.

10 Customization

Customization in a game is about altering features in an attempt to let players personalize the experience and bring themselves into the game. In OS Runner this is done through the profile, where the player can themselves personalize their in-game appearance. OS Runner also lets the player change their settings to their preference in regards to display and audio.

10.1 Settings

Options menu

The ability of being able to change settings is always of a great convenience and it was something we deemed as a "should have" in the MoSCoW presented in Section 1.4. The currently implemented settings that can be changed are: window type, window resolution as well as audio volumes. To change these settings, the user has to access the options menu from either the main menu or the pause menu. When entering the options menu, the user will be greeted with what can be seen in figure 3.19.

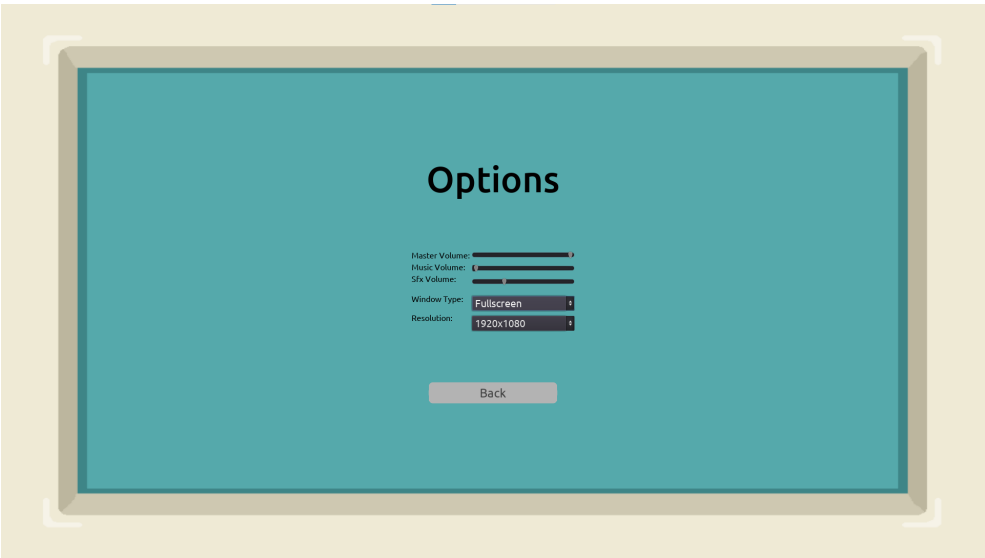


Figure 3.19: Options menu

The implemented window types are fullscreen, borderless fullscreen and windowed mode. If window type is set to either fullscreen or borderless fullscreen, the window will cover the entire screen and the ability to change resolution will be disabled. However if window type is set to windowed, the user has the possibility to change resolution from a predefined list of most common resolutions in 16:9, 4:3 or 16:10 aspect ratio.

The audio volume is split into three different categories which are: master, music and SFX volume. Master volume controls the overall volume of all audio, whilst music volume only controls the music and SFX volume only controls the sound effects. This gives the player more control over which part of the audio they want to adjust to their liking.

Settings

As mentioned earlier the options menu can be accessed either from the main menu or the pause menu. With the pause menu only being accessible whilst ingame. However, the options menu only acts as the visual interface for the player to change the settings. The changes themselves are being updated by the Settings scene which is a singleton[16]. Having the Settings scene as a singleton allows for abstracting the code that stores and applies changes to settings as well as differentiating it from the GUI, which the player sees and interacts with in the options menu.

Persistent storage of settings

In the Settings scene the settings data is being stored in a GDScript dictionary. Upon exiting the options menu, the settings scene writes the settings data dictionary as JSON to a file called "settings.cfg" (see code listing 3.16 and 3.17). The settings data is read from file at game launch and applied accordingly (see code listing 3.18). The reasoning behind this was to have the settings be stored persistently, so the player does not have to re-apply the desired settings each time after launching the game.

Code listing 3.16: Function for writing settings data as JSON to file

```
# Writes settings_data dictionary to file in JSON format
func write_settings_to_file():
	var file = File.new()
	if file.open(_SETTINGS_FILE_PATH, File.WRITE) != 0:
		print("Error opening file: settings.cfg")
		return

	file.store_line(to_json(settings_data))
	file.close()
```

Code listing 3.17: Settings.cfg example

```
{"master_volume":100,"music_volume":80,"resolution":"1366x768","sfx_volume":40,"window_type":2}
```

Code listing 3.18: Function for reading settings JSON data from settings.cfg

```
# Reads settings from file and updates settings_data dictionary accordingly
# If file is missing or unable to open, default settings are set instead
func read_settings_from_file():
	var file = File.new()

	if file.open(_SETTINGS_FILE_PATH, File.READ) != OK:
		print("Unable to find file settings.cfg")

		# Setting default settings if file isn't found
		settings_data["window_type"] = 0
		settings_data["resolution"] = "1366x768"
		settings_data["master_volume"] = 100
		settings_data["music_volume"] = 70
		settings_data["sfx_volume"] = 30

		update_settings()

		MusicManager.switch_activity(MusicManager.Activity.MAIN_MENU)
		return

	settings_data = parse_json(file.get_line())
	print(settings_data)

	update_settings()
	MusicManager.switch_activity(MusicManager.Activity.MAIN_MENU)
```

10.2 Profile

To make it easier to differentiate between players, having a profile is very convenient as it allows them to customize their in-game appearance. Currently each player is free to change their nickname and avatar, although nickname has a max length of *twelve* characters. Regarding avatar, players are able to change it to their liking between a predefined set of avatars.

Profile menu

In OS Runner, the profile menu can be accessed from the main menu by clicking on the avatar with nickname underneath at the top right of the screen. The player will then be greeted with what can be seen in figure 3.20.

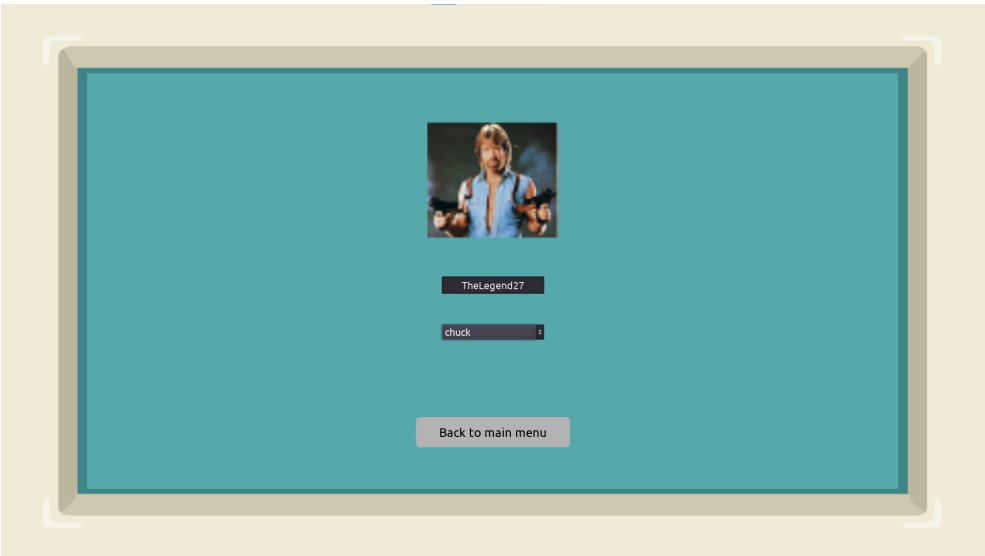


Figure 3.20: Profile menu

In the profile menu there is a text field where the player can freely change their nickname. Besides the character limit, players are free to enter whatever they want. This obviously also implies that we are not checking for any profanity or obscenities. Such a check was something we chose not to implement because of our limited time on this project.

To change the avatar, the player has to select a different one from the drop-down list located beneath the nickname field. The player can see the avatar in the profile menu after it has been selected. Currently there are a total of *three* avatars to choose from. The reason for avatars being predefined is to ensure that they are of proper size, but also to avoid the problem of them being changed to something

inappropriate or of such nature. But, also to make sure other players also have that avatar, to avoid crashes and or bugs related to missing texture.

Profile persistence

In order to avoid having the user change their profile after each time they launch the game, functionality for storing the profile data to a file has been implemented. This is done by writing the information to a data file in JSON format. In code listing 3.19 we can see an example of how the profile data file looks like stored as JSON in file. We can see the path to the selected avatar as well as the chosen nickname.

Code listing 3.19: Example of profile data file

```
{"avatar":"res://assets/avatars/jimmy.png","nickname":"CoolGuyJeff"}
```

Chapter 4

Discussion

1 Reflections

Now that we have discussed the implementation details of our project, we want to have a look at the finished product, and talk about it.

At first we are going to talk about the good and the bad about our end product and the development process. Additionally, we will go though the requirements from Chapter 2, and see how well we have fulfilled them. We will also look into where our game stands in relation to other similar games, as well as the employer's opinion of our achievements. After that, we will be critiquing our original task, and talk some more about our work process and work organization.

1.1 The good

During the development, we worked hard to ensure that at the end of every iteration, we had a working version of the game. Currently, the game features a lot of what we were aiming for.

The game has a working multiplayer mode, where players can participate in battles against each other, compete for assets and Bitcoin on a network topology map, manage their decks, buy and remove cards, and make use of various effects to attack opponents in a battle, or defend themselves against such attacks. The in-game shop makes sure that everyone's deck will be different, and they will have to use the resources and cards available to them in the best way they can come up with.

User interface

Although we have never focused our development on graphics of the game, we have tried to make the game look somewhat adequately good. We spent some time developing the graphical assets to be more fitting to the theme of the game. We have put effort into creating and organizing a clean and understandable UI.

Dynamic visual elements

In the earlier versions of the game, it appeared to look more like a presentation, because it was too static. We have addressed this issue, by adding more dynamic elements, which drastically improved the visual look. We also have tried to make the visuals and the visual theme appeal to the target audience - IT students. However, the game is also still lightweight, and will run even on older or less powerful devices.

Audio

The game also features a dynamic music system, that changes tracks automatically, depending on which activity the player is doing. We have chosen more calm and slow tracks for the menus and map interactions, and more fast and intense tracks for the battles. Adding music had also a huge impact on the user experience and game immersiveness, as we have concluded after doing our playtesting.

Card encyclopedia

While designing the card encyclopedia, we have aimed to make the learning material as easily accessible to the players, as possible. This would help to join the experience of playing the game and learning together. Encyclopedia would contain more information about every card, as well as explain how the cards relate to real-world concepts. Atop of that, by making each card represent a real world cybersecurity technique, we aim to stimulate the players to learn about the cards that they are playing with.

Card update system

The easy system for adding cards into the game means that players and the playerbase, as well as the teaching staff, can easily add new cards to the game, making sure that its easy to keep the game up-to-date with newest cybersecurity

techniques that appear in the professional cybersecurity world for the players to learn about. We will discuss more in Chapter 5 about how this system can be improved and automatized even more.

Code standards

The modularity of the system that we have designed will help future developers to easily maintain and update the game, while our coding practices will help them to easily understand the code.

1.2 The bad

However, our product is not perfect. It's only an alpha version, so there is still so much to work on.

Playtesting

During the development, we have only conducted gross playtesting on our product. This means we have primarily played through the game to find bugs and unexpected behavior in the game. We did not look for feedback whether the game is actually fun to play or not. We didn't focus on wider playtests since the employer wanted the project to be as closed as possible. During our weekly meetings with the employer, we would also do live playtests, in order to showcase the progress.

Cards

Currently, we consider this one of the biggest issues in the current iteration of the game. Right now, the game only includes about a dozen of cards. The main reason for this is that we have not received a full list of cards that can be used in the game from the employer. We as the developers team are not responsible for creating the cards themselves, that was not our task. It was initially agreed in the project planning phase that the employer will provide us with a list of cards. However, it turned out to be much more difficult to generate such a list. One of the main issues with this, as the employer explained, was to correlate the in-game effects to real cybersecurity techniques. Another important issue had to do with applying MITRE ATT&CK framework to in-game card categories.

Therefore, the cards that are currently in the game were created by us, and they serve more as a placeholder, visualizing all the available properties and uses

of a card, and all the different effects that are in the game.

Battle Manager

Battle manager can be quite difficult to navigate through if you don't understand the order in which the functions are running in. Currently every function in the battle manager is running independently from each other. This makes the entire battle manager hard to monitor and get a good grasp of. To make the battle manager more concise, it would be wise to implement a state machine that would keep track of what's going on through the entire process of a battle from start to end. This would ensure that if someone was to take the project from us, they would have easier time interpreting the code and understand the programmatic flow of battling.

Graphical design

While we tried to make the visuals clean, simple and on par with game theme, we are not really content with how they actually turned out. Since none of the team members specialize in creating and using art, we considered it of a lesser importance. In the end, we consider the graphics to look somewhat cheap and bland. Even though dynamic elements enhance the look, we still would have to make use of more complex and powerful shading and lighting effects, as well as in general improve the quality of the art. The game has to look better, to compete with modern indie games on the market.

Neutral Asset Conquering phase

Neutral asset conquering, the gameplay phase that comes after the initial voting but before the players can battle each other, turned out to be problematic. Initially, this part of gameplay was supposed to add a strategic element into the game. The players would choose which assets they would conquer, potentially having to pay for this action, or would be forced to choose between different types of assets that would provide the owners with different buffs, and adapt their game strategy accordingly. However, due to time limitations and the amount of work that had to be put into other game elements, we have neither fully designed nor implemented this part of the game. As a result, this part turned out to be boring, with virtually no decision making available to the player.

In the future iterations, we would overhaul this system by adding more elements to it, that would transform it from barely a repetitive buffer before the

actual gameplay, into a complete gameplay phase. This was discussed with the employer previously, and possible solutions could be adding different types of assets or adding a cost to conquering assets.

1.3 Other games

Our game is not the only game about education in the cybersecurity. It's a small and niche market, but there still are other games out there. However, we still think that our game has a place in it, and that our game bring some novel elements as well.

Let us examine some other games and see what our game tries to be more successful at, compared to them.

A notable game on the market would Maelstrom Defcon24 [17]. The main mechanic of the game includes an attacker, that tries to compromise a server while the defender tries to protect it. The players use cards to progress on a physical board. Each card has a certain amount of moves that the player will progress with. We consider this an unsuccessful design, that fails at providing an interesting gameplay, since the game does not provide many additional mechanics. Additionally, it also fails to provide an educative gaming experience, since the game does not go in-depth into technical aspects of the cards.

Maelstrom Defcon24 is only one example, but there are other similar games, like Backdoors & Breaches [18] or Control-Alt-Hack [19], however, these games are board games, which limits their use and distribution to physical copies, so they are in a slightly different category from our game.

Another, this time digital game, that is there on the market is called Cyber Threat Defender [20]. This game looks much more like our game, having card battles, where cards are used as the direct means to attack the enemy player. However, the big difference that our game, unlike Cyber Threat Defender, support multiplayer. We consider this a very important aspect, that adds a lot of gameplay value.

One more game that we as a team consider a very notable example is Threat Gen[21]. It is a quality digital game, investing heavily into the cybersecurity theme, and even supporting multiplayer. However, this investment also brings a downside - the game is very information-heavy and therefore difficult to get into and learn. This might also explain the reason for the low popularity of the game on Steam[22]. Our game, on the other hand, tries to be simple and easy to start playing even for those, who know little about computer games or cybersecurity. Also unlike our game, Threat Gen does not aim to use MITRE ATT&CK as a basis.

1.4 Fulfilling requirements

We put a lot of emphasis on making sure that the requirements that we initially set were fulfilled as much as possible, especially the ones that we considered the most important. If you remember from Chapter 2, we had drafted a list of requirements using the MoSCoW method. We have used that list to work our way bottom-up through the requirements, making sure that the **Must haves** and **Should haves** are prioritized the most. Let's have a look at the list again, and see which of the requirements have been fulfilled, and to what extent.

Different groups of requirements, as per MoSCoW method, had different levels of completion at the end of the project

Since the **Must have** requirements were so important, they have been effectively fully completed. The **Should have** group turned out a bit more complicated than initially assessed. Many of these requirements were complete, but others were either only partially done, or moved out of scope. This will be discussed more below, when we get to the actual list. The prognosis that the **Could have** requirements would be left out of scope turned out to be true, and effectively none of those features were implemented in the end. Finally, the **Won't have** group was there to not be implemented at all.

Must have:

- We must have a card base
  Fully implemented. Card base is the base element upon each card is generated.
- We must have multiple card types
  Fully implemented. Even though the concept of card types underwent multiple redesigns, cards are categorized with each type having its own mechanics.
- We must have card battles
  Fully implemented. Players use cards to battle for assets on the map.
- We must have functionality to add cards to the deck
  Fully implemented. Players can add cards to their decks and opponents' decks using various mechanics.
- We must have Local Network multiplayer
  Fully implemented. The game is multiplayer-based and players can host and join servers on Local Network as well as over the Internet.
- We must have map topology
  Fully implemented. This is one of the main gameplay mechanics, and lets the players battle for control of network assets on the map topology.
- We must have map navigation
  Fully implemented. Players can navigate on the map through an intercon-

nected network of assets.
- We must have a main menu
  Fully implemented. A main menu hosts ways to change game settings, edit player profile, access the Encyclopedia, and start or join a game.
- We must have Windows support
  Fully implemented. The game supports 64-bit Microsoft Windows releases with at least OpenGL 2.0 support.

Should have:

- We should have a persistent player profile
  Fully implemented. The players can edit their picture and nickname, which is saved to a file and displayed during battles.
- We should have good network performance
  Fully implemented. During playtests, no performance issues were discovered.
- We should have working single-player mode
  Not implemented. While this feature was initially planned and described, and even a full design documentation was written, this feature was quickly moved out of scope of this project, because implementing the multiplayer gameplay took more time than initially assessed.
- We should have working Bitcoin shop
  Fully implemented. Players can buy and remove cards using the in-game currency.
- We should have options menu
  Fully implemented. Persistent music and screen mode settings are implemented.
- We should have all of the card types
  Partially implemented. Initially it was discussed with the employer that he will provide us with a list of cards, that should be added to the game, with all the necessary parameters, such as the type. However, the existing card designs turned out to be incomplete and ill-suited, so after multiple redesigns, it was decided during discussions with the employer, that some of the cards and card types will be implemented, and some will be left for future.
- We should have in-game card encyclopedia
  Fully implemented. The game hosts an in-game Encyclopedia, where the player can find all the in-game cards, along with a more detailed description of their real-life analogues.
- We should have nice visuals in the game
  Implemented. While the visuals were not the focus of the game, we spent considerate amount of time working on them, with many of the graphical assets being crafted by the team. We have also worked on making the game look intuitive and easy to pick up for improved user experience.
- We should have nice audio design in the game

Implemented. Also not the focus, but the game still has royalty- and copyright-free soundtrack, that dynamically changes, based on what the player is doing.

- We should have functionality to easily add new cards into the game
  Implemented. The users can easily added their own cards into the game by simply editing a JSON file. While new card effects can only be added by editing the code, all of the other properties of a card, including use of already existing effects, can be added into the game without touching the code and requiring only basic understanding of the structure of a JSON file.
- We should have Linux and Android support
  Partially implemented. 64-bit Linux distributions with at least OpenGL [23] 2.0 support are supported, Android version turned out to be more difficult to finish. The game is exported and runs on Android, but some UI elements have to be changed for the game to be playable on this platform.

Could have:

- We could have majority of the cards
  Not implemented. The previously mentioned issues with card designs provided by the employer prevented us from implementing them.
- We could have a more complex scoring system
  Not implemented. Left out of scope. No specific designs were discussed.
- We could have a team-up functionality
  Fully implemented. The game supports two teams of 1, 2 or 3 players in each.
- We could have a reputation system
  Not implemented. Left out of scope. No specific designs were discussed.
- We could have specializations
  Not implemented. Left out of scope. No specific designs were discussed.
- We could have assisting of other players in multiplayer
  Not implemented. Left out of scope. No specific designs were discussed.
- We could have working high score menu
  Not implemented. Left out of scope. No specific designs were discussed.
- We could have iOS support
  Partially implemented. The iOS version is in the same state as the Android version, plus distribution difficulties imposed by the platform: an app can only be distributed on their own app catalog [24], and to get it there it must be approved.

Won't have (this time):

- We will not have ALL of the cards
  Not implemented.

- We will not have a story
  Not implemented.
- We will not have good visuals in the game
  Not implemented.
- We will not have good audio design in the game
  Not implemented.

By the end of the project, we have a working product, that can be played by multiple players at once, with functioning battle and map systems, and multiple cards and effects. There is still a lot more to do, which we will discuss in Chapter 5, but in general, we have made a good progress.

1.5 The employer's opinion

We have worked in close cooperation with the employer, and he was constantly monitoring our progress. The employer called the our final product, as well as the development process in general, an interesting experience. He is glad to finally see his ideas come to manifest in a physical product, as well as he is content with the way that we have approached the development and realized the game.

Further development

The employer is quite interested in proceeding with the development of the game. He wants to work on it further, make it into a fully-fledged game, and use it in his own scientific research. He has, however, admitted that the process of designing and developing a game is not easy, and that in the future he wants to change the course that the game is taking. He would like the game to focus more on realism, and add more elements to better represent the real world cybersecurity.

Many more features

Our employer always had a lot of ideas in mind. We were constantly discussing them during our meetings, and considering which ones we could add. However, often we had to explain to the employer that many of these features are beyond the scope. So, the employer had told us that he has a list of new features that he would like to add in further iterations of the game.

Improved graphics

None of the team members specialize in graphics and art. Therefore, we have not focused too much on that, even though we tried to keep the graphics at least somewhat adequate. Nonetheless, the employer had expressed interest in finding people who specialize in this to improve the looks of the game.

2 Critique of the original task

The task itself was lackluster in regards to actual requirements as mentioned earlier in Chapter 2 subsection 1.1. As such, the employer gave us a lot freedom in designing and developing the game and its mechanics. In the end of the development, we have discussed the results with our employer, and even though he was happy with how the game looks like, we have noticed that the experience that we have created is actually somewhat different from the experience that the employer was expecting.

We have then discussed, that if the game was developed further, we would like to consider changing the direction of the development. There were many aspects in the cybersecurity world that weren't properly introduced to us, therefore we can't precisely pinpoint the direction in which the game is supposed to go in in terms of the educational aspect. Since the game is very tightly connected to cybersecurity, this problem accelerated the difference between what the employer expected, and what the game turned out to be like.

It is important to note, that all of this was not due to unfulfilled requirements or a lack of communication. During the initial design phase and discussions, we have discovered, that our employer does not have a very specific vision of the game, its elements and mechanics. Therefore, even though we have thoroughly gone through the ideas and propositions, documented the designs and the requirements, it was really difficult for us to deliver the experience that the employer really wanted. The employer, as the team had concluded, himself did not really have a clear understanding of the experience that he wanted.

It is also worth mentioning that our employer was quite open minded and enjoyed sparring different suggestions about gameplay and gameplay mechanics as well as card ideas. Additionally the original task mentioned creating a mobile game, however after a meeting with our employer we were told that it was okay to focus on the platform we felt most comfortable developing for.

3 Evaluation of group work

We started the bachelor project by creating a project plan (see appendix D) where we discussed and disclosed the organization of the project and the schedule. This entailed creating and assigning of roles, choosing of development methodology, creating group rules and prioritization of features. This section is therefore dedicated to in retrospect evaluate and reflect over the decisions made in regards to the initial project plan.

3.1 Group organization

Roles

In the project plan, we agreed on a set of roles for each individual on the group. These roles were: communications manager, secretary and lead designer in addition to the developer role which everyone were given. The role of communications manager remained unchanged throughout the project, the same goes for the secretary role. Last, but not least there was the role of lead designer, which ended up becoming more of a lesser role since everyone aimed at creating a pleasant and meaningful design.

As mentioned, everyone had the role of developer and worked on the games code. However, as a group, we agreed to go for a divide and conquer approach in regards to developing features. As a result we decided that every member of our group should specialize in different areas of the game. The areas we decided on were: "everything related to cards", "everything related to UI" and "all things related to networking and map topology".

Group rules

In total, we decided on *5* rules which can be seen in the project plan. These rules were created to establish a proper working routine as well as a procedure in case a problem occurred. All group members upheld the rules regarding the working hours and if not a notice was issued to let the rest of the group know. When a problem arose, members of the group engaged in the following discussion to resolve the problem at hand.

Evaluation

All in all, this group setup and the general workflow of us as a group we consider a success. We have almost never had any communication issues, we made sure that all the members work hard and work consistently. All the rules and roles that we have applied and used, turned out to be healthy and useful for our progress.

As a team, we are proud of the time we have spent on working on the project, and we are proud of the final product.

Considering the lack of requirements, as mentioned in Section 2, we spent a lot of time in the planning phase. However as a group this is not something we regret as everyone agrees on it being necessary in terms of forging the path to fulfilling the goal of a proof of concept. We had to spend all this time on planning, to make sure that the final product will be something well thought-out and something we can be proud of. It would have been more beneficial to us if the task was formulated and described more precisely by our employer from the beginning.

3.2 Work organization

MoSCoW and prioritization

Since the bachelor task was about creating a proof of concept of the game built from the ground up in a game engine of our choice, we decided in the planning phase that it would be in our best interest to create an overview over the features based on the priority set by us. Which is why we decided on using the MoSCoW method[8]. We found this to be very beneficial and of great convenience as it was very clear to us, our employer and our supervisor as to which features we were going to focus all of our attention on to implement into OS Runner. The MoSCoW list helped us to focus on the areas that required the most attention, and to avoid getting distracted by less important side features. It also aided us by creating a set of informal milestones for the development.

Development methodology

As mentioned in the project plan in appendix D, we spent a decent amount of time discussing the different development methodologies. We had a productive discussion and ended up with a combination of two methodologies we deemed as the most effective and viable for this project. The final decision therefore ended up being the Scrumban methodology.

In hindsight using Scrumban seems to have been the correct choice for a multitude of reasons.

First of all, the prioritization of tasks ended up having a helpful symbiotic relationship with the MoSCoW prioritization technique we chose to utilize. Creating new user stories to complete the requirements prioritized in MoSCoW was a simple process because we knew precisely what we had to work on and deemed the most crucial.

Additionally we were working in short iterations, which was of great convenience in terms of always being able to showcase our progress during our weekly meetings with our employer. The adaptability these iterations gave us was also key if there was a requested feature or change from our employer. The progress from our iterations were also neatly kept track of as a result of using a Scrumban board.

One more very useful aspect of Scrumban that we came to utilize the most is the board. GitHub, that we have used as the version control system, has a built-in board system, that we used as our Scrumban board. This board helped us immensely in organizing the workflow, adding user stories and tasks, as well as bugs, and letting the team members choose the ones they want to work on. As well as the entries in the board being issues, we could easily reference them in our commits and associate them with the feature they contributed to.

Gantt chart

When creating the project plan, we made a Gantt chart for planning our activities and getting an overview of how much work could be done during our limited time span of this bachelor project. The Gantt chart we made can be seen in appendix D and it consists of a planning phase in the beginning and followed by a total of *twelve* iterations. We also decided that only the first iteration should be *two* weeks long.

During the initial first iteration, we noticed that the two week time frame to create a backbone for the project was to little and instead spent three weeks. During our future iterations, we couldn't quite decide on the length of each iteration, and we took an unanimous vote of having an iteration last two weeks. But even though the duration of our iterations increased, we still had weekly meetings with our employer where we showcased our new progress.

In general, the initial Gantt chart was more to guide us through the timeline. We did not expect nor try to adhere to it too much, instead we have decided to focus more on the progress itself, on the Scrumban board, and the discussions with the employer. The employer himself also did not require any hard deadlines

or timeframes for features, so the strongly sticking to Gantt made little sense. The revised Gantt chart can be seen in figure 4.1

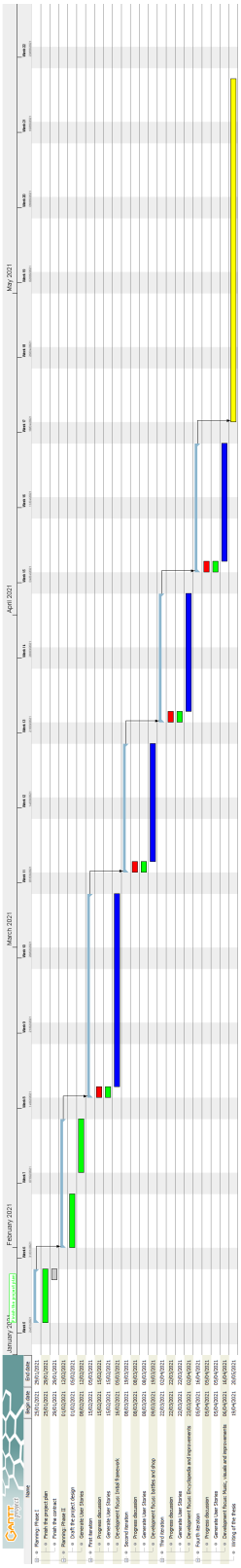


Figure 4.1: Revised Gantt chart

Chapter 5

Further work

1 Single-player

Single-player mode of the game was only seriously discussed in the beginning. It was mentioned in the task proposal, as well as the employer made us aware that he is interested in developing a single-player as well as multiplayer. He was, however, also clear that the latter is of much higher importance.

When we were writing the initial project plan, we had included the single-player as a **Should have** requirement in our MoSCoW list, but very soon this proved to be an impossible task. The effort required to develop the multiplayer mode, as well as all of the game foundations around it, turned out to be way too big. Consequently, the single-player mode was moved out of the project scope very early in the development, as to avoid losing focus on the rest of the game and on more important crucial multiplayer gameplay features. We as a team, together with the employer, have decided that we would much rather put as much effort as possible into the multiplayer and have it as fleshed out as possible.

Nonetheless, the design for a single-player mode, that would have similar game mechanics, as the multiplayer mode, in addition to a story revolving around a whistle-blower, was documented in a big design document, available in the appendix C.

2 Use in education institutions

As we have mentioned before, our employer is a former teacher with a huge passion for gamifying of learning in cybersecurity. Consequently, one of the most

important applications for our game is the use as an education tool in educational institutions.

In the current iteration, the game is not yet ready to be used as an educational tool, because of all the shortcomings, that we have mentioned in Chapter 4. However, the proof of concept is there, and in future iterations the game can be used in tangential learning.

As proposed by our employer, the game would not be used as a substitute to conventional learning methods and materials, but as an extracurricular element, such as organizing a game evening for the students.

During the following iterations of the game, when it will be deemed ready for big scale player review, it is planned that multiple universities worldwide would be reached out to with a proposition to include the game into their curricula.

3 Digital distribution platforms

When the development process finishes and we will be satisfied with the product we have completed, we might consider the possibility of distributing the game onto other platforms like Steam or itch.io [25]. Steam could also enable us to host the servers for the players, enabling for centralized server. This way the players wouldn't have to worry about network or port forwarding issues when hosting a game. This, however, would only be done when the game went through extensive improvements, and is deemed ready for the consumer.

4 Going Open Source

Another option that we have discussed and considered with the employer is making the game be sponsored through crowdfunding, and making the code open source. This way, we could get the financial resources to finish and improve the, while also making an educational game free for the consumers. The team members all agree that education should be free, so we consider this an option worth considering as well.

5 Cybersecurity wiki

The developer had discussed with us the creation of an open-source cybersecurity wiki. This wiki will be updated by the community, so it would always be

up-to-date and everyone would have a chance to communicate. The current JSON system of adding cards will be then substituted with a system that will automatically update add new cards from the wiki to the game.

6 Colorblind accessibility

As mentioned in Chapter 1 Section 6, the goal of OS Runner is to stimulate cybersecurity education through tangential learning and associations. It is therefore important to ensure the accessibility to colorblind players considering the prevalence of 5%-8% in males and 0,5%-1% in females[26]. The higher prevalence in males is especially important considering the high amount of men in IT studies. As of 2016, only 19% of computer science bachelor's degrees were awarded to women [27], although this statistic does not accurately represent the number of women or men enrolled in IT studies, it should still give an estimate.

The most common way of colorblind accessibility in video games, has been to include a whole-screen filter for the different types of colorblindness[28]. Examples of games where this is the case would be: Call of Duty: Ghosts, Call of Duty: Advanced Warfare and DOOM. However, in these games the filter does not alter the problematic colors, but instead tends to oversaturate the color palette, which results in a mix of undesirable colors. Which can in turn ruin the games immersive feeling and aesthetic.

Considering the less optimal way of handling colorblind accessibility using a whole-screen filter, a more optimal way of handling it would be to let the player themselves customize the colors for vital information. This would ensure that the player, regardless of colorblind type, can choose the colors they deem as most suitable. Another important design decision to further ensure accessibility to colorblind would be to avoid relying on color alone. Ergo, represent things by adding different icons/symbols.

This would help the colorblind players that are having trouble differentiating some of the elements in OS Runner. Currently these elements would include the information displaying which team has conquered which asset, health bars and card borders. The settings for changing the colors to these elements can be implemented into the options menu mentioned in Chapter 3 Section 10.1.

Chapter 6

Conclusion

We have made a proof of concept of a game which intention was to combine the aspect of cybersecurity and fun. When planning the process for creating the game, we started with carefully planning what to do and organized what we wanted to accomplished throughout our limited time for this project. We started from scratch and developed what we think is a functional prototype with such gameplay mechanics as contesting the network map and battles for assets on the network map using our implemented card system. All this is made possible with a client-server system that binds players together making the entire game possible to be played by multiple players together.

We as a team also consider that, even though what we have now is only a alpha version of the game, it has a lot of perspective given more time and resources. Our team is proud of what we have developed, and we hope for the opportunity to work more on it in the future.

The game is not the first of its kind, but nevertheless it brings novel elements to the field of educative cybersecurity games, already now having a solid foundation to provide both a fun and engaging gameplay and a useful learning experience.

Our employer is content with the end result as well. He also sees the potential in the game, and will be working on it further, seeking to distribute the game to numerous universities to test its effectiveness in teaching, as well as to use it in his own scientific research. We also believe that OS Runner could in the future be used for various research into gamification of cybersecurity by abstracting cybersecurity tactics and techniques into playable cards used to battle other players.

Bibliography

[1] *Mitre att&ck*. [Online]. Available: https://attack.mitre.org/, (Last visited 19.05.2021).

[2] *Computers in the classroom: Desktop vs. laptop vs. tablet*. [Online]. Available: https://www.stonegroup.co.uk/insights/computers-in-the-classroom/, (Last visited 13.05.2021).

[3] MegaCrit, *Slay the spire*. [Online]. Available: https://store.steampowered.com/app/646570/Slay_the_Spire/, (Last visited 19.05.2021).

[4] S. Shoe, *Monster train*, 2020. [Online]. Available: https://store.steampowered.com/app/1102190/Monster_Train/, (Last visited 19.05.2021).

[5] *Imt3004 incident response, ethical hacking and forensics*. [Online]. Available: https://www.ntnu.edu/studies/courses/IMT3004#tab=omEmnet, (Last visited 19.5.2021).

[6] *Imt3601 game programming*. [Online]. Available: https://www.ntnu.edu/studies/courses/IMT3601/2020#tab=omEmnet, (Last visited 10.5.2021).

[7] *Imt2531 graphics programming*. [Online]. Available: https://www.ntnu.edu/studies/courses/IMT2531/2019#tab=omEmnet, (Last visited 19.5.2021).

[8] *Moscow method*. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MoSCoW_method&oldid=1005313791, (Last visited 26.4.2021).

[9] *Scrumban – a hybrid agile framework with long-term planning*. [Online]. Available: https://teamhood.com/agile/scrumban-a-hybrid-agile-framework-with-long-term-planning/, (Last visited 29.4.2021).

[10] *Writing user stories*. [Online]. Available: https://www.gov.uk/service-manual/agile-delivery/writing-user-stories, (Last visited 29.4.2021).

[11] *Godot game engine*. [Online]. Available: https://godotengine.org/, (Last visited 10.5.2021).

[12] *Github*. [Online]. Available: https://github.com/, (Last visited 19.05.2021).

[13] *Godot project organization documentation*. [Online]. Available: https://docs.godotengine.org/en/stable/getting_started/workflow/project_setup/project_organization.html, (Last visited 3.5.2021).

[14] *Mergesort algorithm*. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Merge_sort&oldid=1020266621, (Last visited 28.4.2021).

[15] *Gdscript's array sort method*. [Online]. Available: https://docs.godotengine.org/en/stable/classes/class_array.html#class-array-method-sort, (Last visited 27.4.2021).

[16] *Godot singletons (autoload) documentation*. [Online]. Available: https://docs.godotengine.org/en/stable/getting_started/step_by_step/singletons_autoload.html, (Last visited 29.4.2021).

[17] *Maelstrom defcon24*. [Online]. Available: https://media.defcon.org/DEF%5C%20CON%5C%2024/DEF%5C%20CON%5C%2024%5C%20presentations/DEF%5C%20CON%5C%2024%5C%20-%5C%20Shane-Steiger-Maelstrom-Rules-V10.pdf, (Last visited 19.05.2021).

[18] B. H. I. Security, *Backdoors and breaches*. [Online]. Available: https://www.blackhillsinfosec.com/projects/backdoorsandbreaches/, (Last visited 19.05.2021).

[19] U. of Washington, *Control-alt-hack*. [Online]. Available: http://www.controlalthack.com/index.php, (Last visited 19.05.2021).

[20] *Cyber threat defender*. [Online]. Available: https://cias.utsa.edu/ctd_cards.php, (Last visited 19.5.2021).

[21] *Threatgen: Red vs. blue*. [Online]. Available: https://store.steampowered.com/app/994670/ThreatGEN_Red_vs_Blue, (Last visited 20.5.2021).

[22] *Steam*. [Online]. Available: https://store.steampowered.com/, (Last visited 20.5.2021).

[23] *Opengl*. [Online]. Available: https://www.opengl.org/, (Last visited 20.5.2021).

[24] *App store*. [Online]. Available: https://www.apple.com/app-store/, (Last visited 20.5.2021).

[25] *Itch.io*. [Online]. Available: https://itch.io/, (Last visited 20.5.2021).

[26] *Color blindness prevalence*. [Online]. Available: https://www.news-medical.net/health/Color-Blindness-Prevalence.aspx, (Last visited 18.05.2021).

[27] *Women, minorities, and persons with disabilities in science and engineering*. [Online]. Available: https://ncses.nsf.gov/pubs/nsf19304/digest/field-of-degree-women#computer-sciences, (Last visited 19.05.2021).

[28] B. Hardin. (2016). "Colorblind accessibility in video games – is the industry heading in the right direction?" [Online]. Available: https://www.gamersexperience.com/colorblind-accessibility-in-video-games-is-the-industry-heading-in-the-right-direction/. (Last visited 19.05.2021).

Appendix A

User Stories

Menu oriented user stories:
1 - 2 - 3 - 4 - 5 - 6 - 7 - 8 - 9 - 10 - 11 - 12 - 13 - 15 - 16 - 17 - 18 - 23

Server oriented user stories:
12 - 13 - 14 - 15 - 16

Cards oriented user stories:
19 - 20 - 21 - 22

Gameplay oriented user stories:

Battle related user stories:
40 - 41 - 42 - 43 - 44 - 45 - 46 - 47 - 48 - 49 - 50 - 51 - 52 - 53

Scrapped user stories: 14 - 24

User Story number	1
Description	As a player, I want to be able to access the main menu when I boot up the game, so that I can choose what to do next
Acceptance criteria	I. It's done, when the main menu shows up at the start of the game

User Story number	2
Description	As a player, I want to be able to access the single player starting screen from the main menu
Acceptance criteria	I. It's done when the main menu has a Single Player Mode button that moves the player to single player screen when clicked II. It's done, when the player can go back to the previous screen after accessing the singleplayer

User Story number	3
Description	As a player, I want to be able to access multiplayer starting screen from the main menu

Acceptance criteria	I. It's done when the main menus has a Multiplayer button which leads to multiplayer menu screen when clicked II. It's done, when the player can go back to the previous screen after accessing the multiplayer screen

User Story number	4
Description	As a player, I want to be able to access the options screen from the main menu
Acceptance criteria	I. It's done when the main menu has a button that moves the player to options screen when clicked II. It's done, when the player can go back to the previous screen after accessing the options

User Story number	5
Description	As a player, I want an "exit game" button in the main menu so that I can exit the game via the main menu
Acceptance criteria	I. It's done when the main menu has an "exit game" button that closes the game when pressed

User Story number	6
Description	As a player, i want to be able to control the sound volume from the options screen
Acceptance criteria	I. It's done, when the options screen makes it possible to change the volume II. It's done, when the player can change the music volume, the sound volume and the master volume separately III. It's done, when the changes are saved IV. It's done, when the user does not have to make them again on the

	next startup

User Story number	7
Description	As a player, I want to be able to access the card encyclopedia from the main menu
Acceptance criteria	I. It's done, when there is a way to go to the card encyclopedia that leads the player to the card encyclopedia screen II. It's done, when the player can go back to the previous screen after accessing the encyclopedia

User Story number	8
Description	As a player, I want to be able to view all cards in the game in the card encyclopedia
Acceptance criteria	I. It's done when the player can view all the cards in the card encyclopedia II. It's done, when new cards added to the game are automatically shown in the encyclopedia

User Story number	9
Description	As a player, I want to have my own profile in the game
Acceptance criteria	I. It's done, when the player can access the profile from the main menu II. It's done, when the profile is persistent III. It's done, when all the profile data is displayed in the profile menu IV. It's done, when the player can go back to the previous screen after accessing the profile

User Story number	10

Description	As a player, I want to be able to reset my profile, so that I can start the game over
Acceptance criteria	I. It's done, when the reset option asks the user again to prompt the action II. It's done when the user can easily find this button III. It's done, when all user achievements are reset by doing this

User Story number	11
Description	As a player, I want to be able to create a new multiplayer game
Acceptance criteria	I. It's done, when the player can create a room to invite other players II. It's done when the player can share some unique identifier with other players, which they can use to join

User Story number	12
Description	As a player, I want to be able to join an already created game
Acceptance criteria	I. It's done, when the user can connect to other players' rooms by using an unique identifier

User Story number	13
Description	As a player, after joining a game, I want to be able to choose a team, so that I can play on the same team with the people I want
Acceptance criteria	I. It's done, when the player can easily switch teams before the game starts II. It's done, when the game can only start if team sizes are as equal as possible

~~**User Story number**~~	~~14~~

~~**Description**~~	~~As a player, after pressing the start button, the game starts.~~
~~**Acceptance criteria**~~	~~I. It's done, when everybody else is ready in lobby~~

User Story number	15
Description	As a joined player, I want to be able to notify the host that I am ready to play
Acceptance criteria	I. It's done, when the joined player can easily toggle readiness II. It's done when the host player can immediately see the readiness changes for all players

User Story number	16
Description	As a host player of a multiplayer lobby, I should be able to start the multiplayer game
Acceptance criteria	I. It's done when the host player can start the multiplayer game by pressing a start game button in the multiplayer lobby II. It's done when the host can only start the game when all the players are ready

User Story number	17
Description	As a player, I want to be able to change my profile name
Acceptance criteria	I. It's done when the nickname is persistent

User Story number	18
Description	As a player, I want to be able to change my profile picture
Acceptance criteria	I. It's done when the player can change profile picture to another

	profile picture from a predefined list of profile pictures II. It's done when the chosen profile picture is persistent

User Story number	19
Description	As a modder, I want to be able to easily add new cards to the game, so that new stuff can be added to game not just by the developers
Acceptance criteria	I. It's done when new cards can be added without having to recompile the game II. It's done when the new cards can be added by users without extensive programming skills

User Story number	20
Description	As a player, I want to be able to see all information about a single card in the card encyclopedia
Acceptance criteria	I. It's done when the player can inspect a card in the encyclopedia and see as much information about that card as possible

User Story number	21
Description	As a player, I want to be able to see all information about different categories of cards
Acceptance criteria	I. It's done, when the player can do this in the card encyclopedia

User Story number	22
Description	As a player, I want to be able to see all

	information about different subcategories of cards
Acceptance criteria	I. It's done, when the player can do this in the card encyclopedia

User Story number	23
Description	As a player, I want to be able to leave the lobby before the game starts
Acceptance criteria	I. It's done when upon leaving the lobby, the player is returned to the previous screen II. It's done when upon the host player leaving the game, the lobby is destroyed III. It's done, when upon the lobby destruction all the remaining players are returned to the previous screen IV. It's done, when the player can not leave the lobby after the host player starts the game

~~**User Story number**~~	~~24~~
~~**Description**~~	~~As a player, I want to be able to join teams with other players~~
~~**Acceptance criteria**~~	~~I. It is done, when teams can consist of 1, 2 or 3 members~~ ~~II. It is done, when teams are chosen during in the game lobby~~ ~~III. It is done, when there can only be 2 teams~~

User Story number	25
Description	As a player, I want to be able to see a topology map when the game starts
Acceptance criteria	I. It is done, when players see this first immediately when the game starts

User Story number	26
Description	As a player, I want to be able to vote for the initial starting position with my team members
Acceptance criteria	I. It is done when all the team members can vote II. It is done when players can choose a predefined node as the starting position III. It is done when a draw in votes results in a coin flip IV. It is done, when a node with most votes wins

User Story number	27
Description	As a player, I want to be able to attack neutral assets around the network
Acceptance criteria	I. It is done, when attacking a neutral asset instantly makes it “contested” II. It is done, when attacking a neutral asset costs Bitcoins III. It is done, when a neutral asset can only be attacked by one player at a time IV. It is done when attacking a neutral asset is only possible when there are still neutral assets

User Story number	28
Description	As a player, I want the turn to end when all team players have chosen an asset to attack on the network topology screen
Acceptance criteria	I. It is done when ending a turn is automatic II. It is done when ending a turn makes all “contested” nodes conquered

User Story number	29
Description	As a player, I want to be able to attack other players’ nodes

Acceptance criteria	I. It is done, when attacking other player's node initiates battle combat II. It is done when a player can only be attacked by one player at once III. It is done, when players can only be attacked when there are no neutral nodes

User Story number	30
Description	As a player, I want to be able to battle other players
Acceptance criteria	I. It is done when after attacking another player's node, battle between the attacking player and a random defending player from the opponents team commences II. It is done when the winning player's team, gets control of the relevant node

User Story number	31
Description	As a player, I want a card battle to result in a victory or a defeat
Acceptance criteria	I. It is done, when one player's victory means a defeat for the other player II. It is done, when a player, that lowers opponent's health points to zero is victorious

User Story number	32
Description	As a player, I want to be able to earn Bitcoin every turn
Acceptance criteria	I. It is done, when each turn on the topology map the player receives a set amount of Bitcoin

User Story number	33
Description	As a team, I want to be able to receive a set amount of Bitcoin at the start of the game

Acceptance criteria	I. It's done, when at the beginning of the game each team receives enough Bitcoins to conquer the neutral assets

User Story number	34
Description	As a player, I want to be able to spend money in the shop
Acceptance criteria	I. It is done when the player can buy cards in the shop II. It is done, when the player can only buy a card once III. It is done when the player can remove cards in the shop IV. It is done, when the player can only remove a card once V. It is done when the player can refresh the stock in the shop

User Story number	35
Description	As a player, I want to be able to enter a shop
Acceptance criteria	I. It is done when there is a GUI they can access from the topology. II. It is done when this button will show a shop screen upon pressing.

User Story number	36
Description	As a player, I want to be able to leave the shop
Acceptance criteria	I. It is done when there is a GUI button within the window button. II. It is done when the shop window closes upon pressing the button.

User Story number	37
Description	As a player, I want to have time to make

	decisions in the shop.
Acceptance criteria	I. It is done when a timer is created when user enters a store A. It is done when the timer starts when the opponent attacks the you while in the shop B. It is done when the player must leave the store after certain amount of time II.

User Story number	40
Description	As a player, I want to be able to use cards during combat
Acceptance criteria	I. It is done when the player can drag his cards. II. It is done when the cards have an effect. III. It is done when a visual effect plays after playing a card

User Story number	41
Description	As a player, I want to see how much HP battle participants have
Acceptance criteria	I. It is done when there is a GUI displaying current health points II. It is done when the GUI updates after a card has been used.

User Story number	42
Description	As a player, I want to be able to view my hand of cards during battle
Acceptance criteria	I. It is done when there is a GUI displaying current hand of cards II. It is done when the GUI updates after a card has been drawn or used

User Story number	43
Description	As a player, I want to display the statistics of my cards
Acceptance criteria	I. It is done when you hover over the cards (PC) II. It is done when you hold your finger on the card (Phone) III. It is done when the card is big enough to read it

User Story number	44
Description	As a player, I want to be able to view my own discard pile during battle
Acceptance criteria	I. It is done when there is a GUI that can be clicked to view player's cards in the discard pile

User Story number	45
Description	As a player I want to be able to view my draw pile during battle
Acceptance criteria	I. It is done when there is a GUI that can be clicked to view player's cards in the draw pile

User Story number	46
Description	As a player, I want to see what cards have been used in the past five turns.
Acceptance criteria	I. It is done when there is a GUI displaying what cards have been played. II. It is done when there is a GUI displaying what effects have activated.

User Story number	47
Description	As a player, I want to be able to view active

	power cards during battle
Acceptance criteria	I. It is done when there is a GUI during battle that displays the active **power** cards for player and opponent II. It is done when on PC, player can hover mouse over active **power** cards to view its effect III. It is done when on mobile, player can hold finger over active **power** card to view its effect

User Story number	48
Description	As a player, I want to be able to skip a turn
Acceptance criteria	I. It is done when there is a GUI with text “end turn”. II. It is done when you press the button, the player's opponent gains a turn and.

User Story number	49
Description	As a player, I want to be able to go to options during a battle
Acceptance criteria	I. It is done when there is a GUI button that looks like a cog. II. It is done when you press on that button and a pop-up shows with options.

User Story number	50
Description	As a player I want to be able to view my deck, as it was before entering battle, during battle
Acceptance criteria	I. It is done when there is a GUI button during battle, which when clicked will display the deck as it was before entering the current battle

User Story number	51
Description	As a player I want to be able to view map during a battle
Acceptance criteria	I. It is done when there is GUI in the battle screen. II. It is done when you press on the button and it displays the current state of the map.

User Story number	52
Description	As a player, I want to lose when I reach zero health points.
Acceptance criteria	I. It is done when health points are updated after a card is played II. It is done when the player is notified he lost a battle. III. It is done when the player can not play any more cards IV. It is done when the player is send to the map topology (Multiplayer)

User Story number	53
Description	As a player, I want to win when my opponent reaches zero health points
Acceptance criteria	I. It is done when health points are updated after a card is played II. It is done when the player is notified he won a battle. III. It is done when the player can not play any more cards IV. It is done when the player is send to the map topology (Multiplayer)

User Story number	54
Description	As a player, I want to be able to view my current amount of bitcoin
Acceptance criteria	I. It is done when the current amount of bitcoin is displayed in the GUI

	II. It is done when the amount of bitcoin displayed is being updated to always display current amount of bitcoin

User Story number	55
Description	As a player, I want to be able to have a deck
Acceptance criteria	I. It is done when I can add cards to the deck. II. It is done when I can remove cards from the deck. III. It is done when I can view my deck.

User Story number	56
Description	As a player, I want to hear music in the main menu
Acceptance criteria	I. It is done, when the music volume can be controlled from options

User Story number	57
Description	As a player, I want to be able to enter a battle screen when I contest an enemy network asset
Acceptance criteria	I. It is done, when attacking an enemy network asset starts a battle screen II. It is done, when the battle screen displays: A. Both players' health B. Both players' hands C. Both players' powers D. Both players' avatars

User Story number	58
Description	As a player, I want to have my player

	represented in the battle
Acceptance criteria	I. It is done, when the player representation has health points II. It is done, when the player representation has a card of hands III. It is done, when the player representation has a deck of cards IV. It is done, when the player representation has powers V. It is done, when the player representation has an avatar

User Story number	59
Description	As a player, I want to to be able to see my own avatar in the combat
Acceptance criteria	I. It is done when there is a GUI displaying players avatar II. It is done when

User Story number	60
Description	As a player, I want to have an avatar
Acceptance criteria	I. It is done when the player can choose an avatar II. It is done when the avatar chosen is saved.

User Story number	61
Description	As a player, I want the multiplayer topology map to have multiple nodes
Acceptance criteria	I. It is done, when the nodes are conquered on click, when they are neutral II. It is done, when the nodes send the player to a battle screen on click, when they belong to an enemy

User Story number	62
Description	As a modder, I want the multiplayer network topology to be loaded from a text file
Acceptance criteria	I. It is done, when a new multiplayer map can be added without editing the code

User Story number	63
Description	As a player, I want to make moves in a team-based but still player-independent manner
Acceptance criteria	I. It is done, when during the voting phase, players move in teams II. It is done, when during the NAC phase, players move in teams III. It is done, when during the EAC phase, players still move in teams, but the order might change freely depending on which player is not in a battle

User Story number	64
Description	As a player, I want to be able to access the shop
Acceptance criteria	I. It is done when there is a button in the topology map that redirects you to a screen with a shop

User Story number	65
Description	As a player, I want to be able to buy cards from the shop
Acceptance criteria	I. It is done when there is GUI in the shop allowing the player to buy a card II. It is done when the cost of the card

	is displayed III. It is done when the card is added to the players deck IV. It is done when the exact amount of bitcoins is removed from the players inventory

User Story number	66
Description	As a player, I want to be able to remove cards from the deck via shop
Acceptance criteria	I. It is done when there is GUI in the shop allowing the player to remove cards II. It is done when the price of removing a card is displayed III. It is done when the price increases with each removal IV. It is done when the exact amount of bitcoins is removed from the players inventory

User Story number	67
Description	As a player, I want to be able to restock the store with new cards
Acceptance criteria	I. It is done when there is GUI in the shop allowing for restocking of the store. II. It is when there are new cards in the store with new prices. III. It is done when the exact amount of bitcoins is removed from the players inventory

User Story number	68
Description	As a player, I want to have an in-game currency called “Bitcoin”
Acceptance criteria	I. It is done when you can see the amount of bitcoin you possess (User story 54)

	II. It is done when you can spend the bitcoin (User stories 64-66) III. It is done when you can earn Bitcoin (User stories 32-33)

Appendix B

Gameplay flow multiplayer

Main concepts

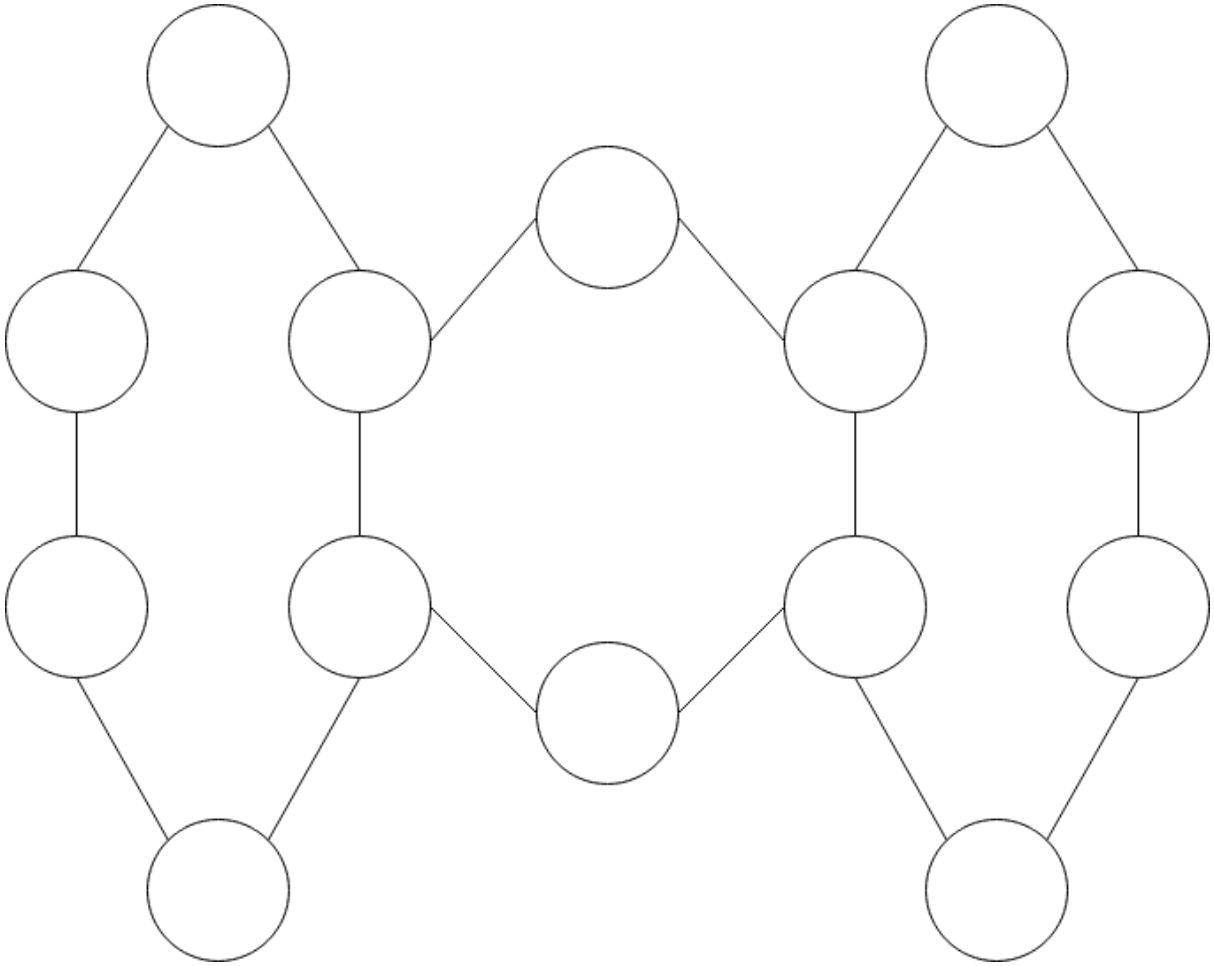
Gameplay flow

Multiplayer

Game starts when all players have toogled “Ready” and the host has started the game. See “Main Menu flowchart” for more details.

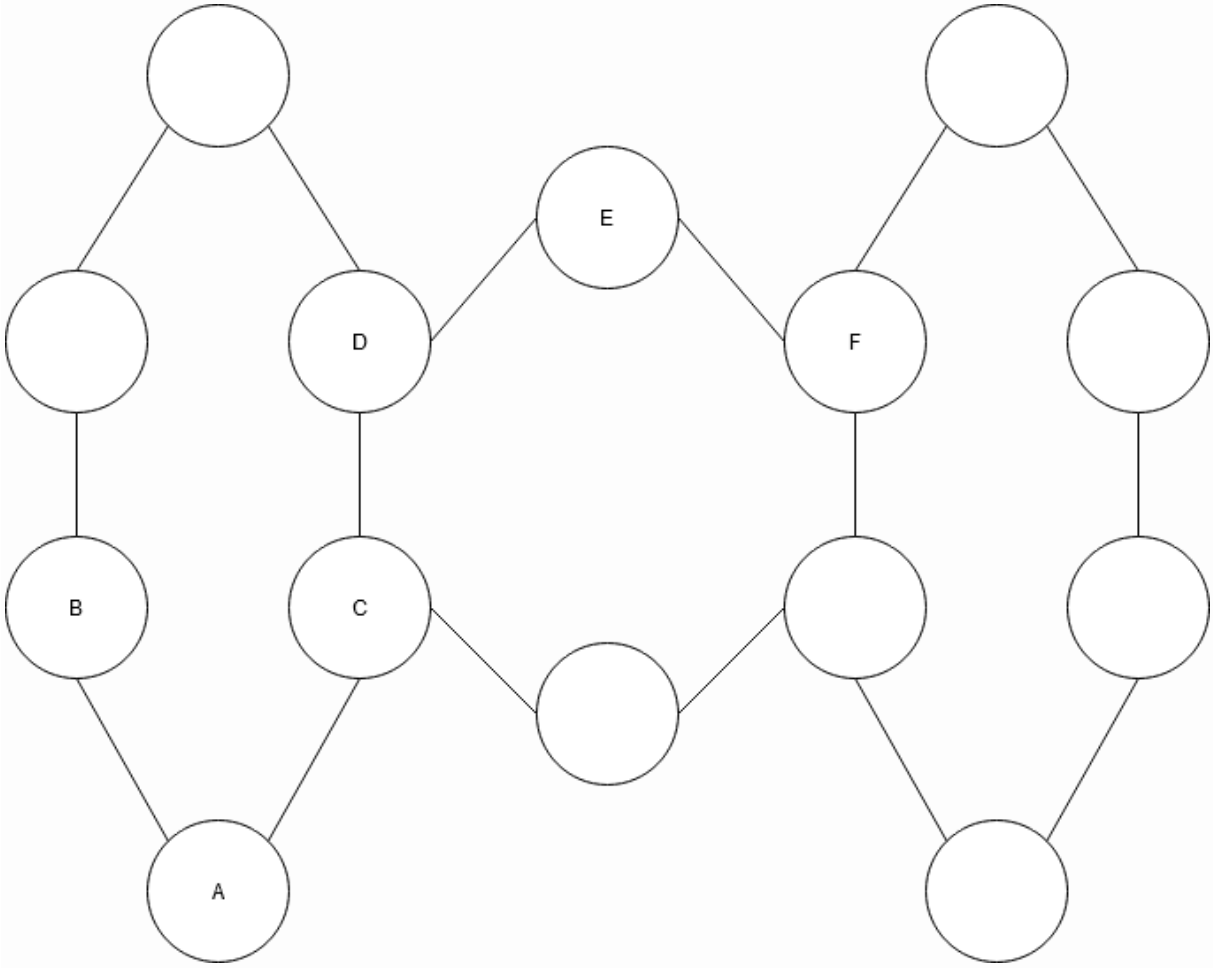
Topology map

Upon the initial game start, both teams are presented with a map, resembling a network topology. This is a schematic example of how it might look like:



The circles depict the assets – the nodes – that the players may conquer. The lines between them depict connections, and only two nodes that are interconnected can be traversed normally. This might have some exceptions, *exempli gratia* players can use special

cards or abilities to traverse to nodes that are not connected.



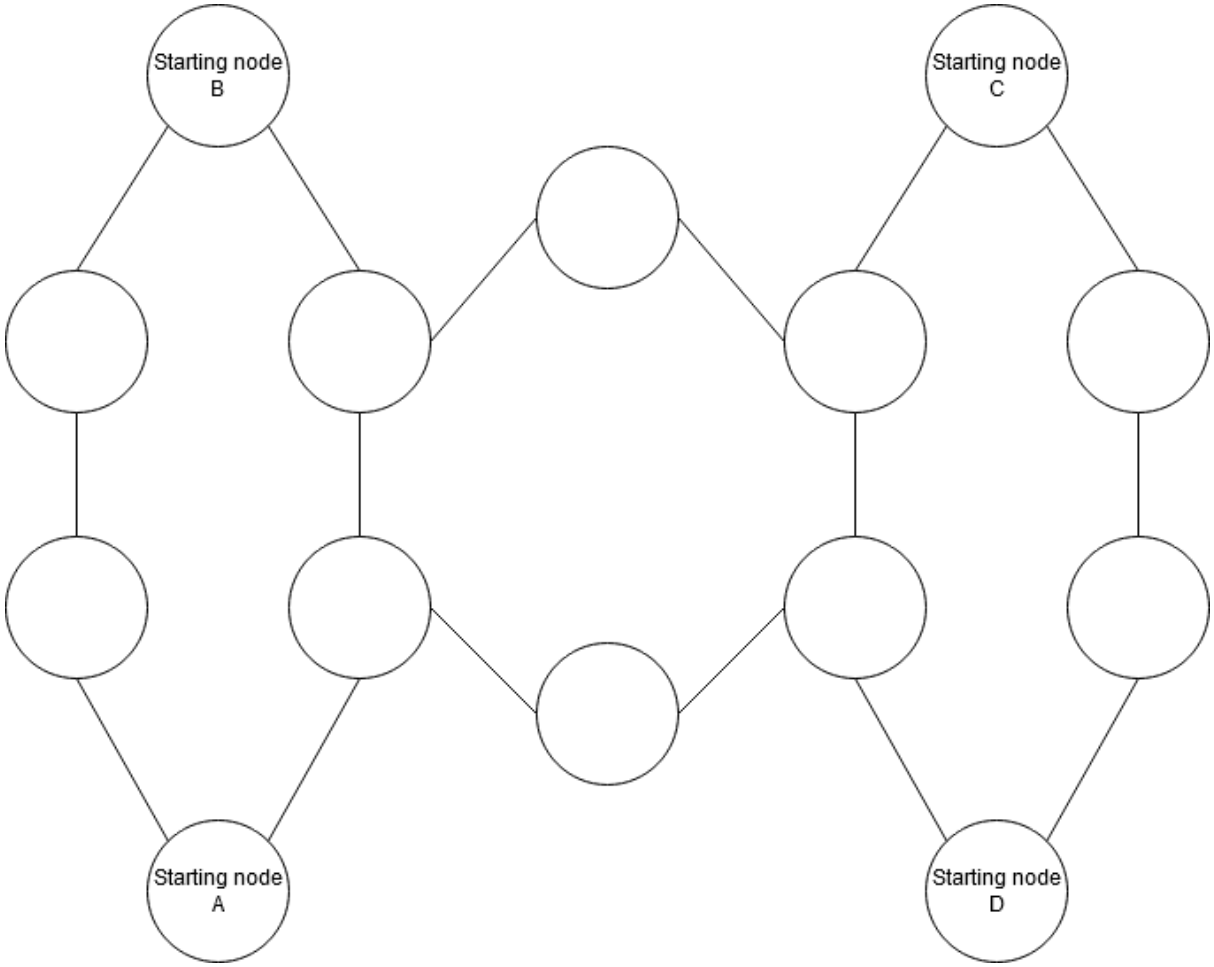
From node A, only nodes B and C can be travelled to. From node E, only nodes D and F can be travelled to, and so on. From node A, nodes D, E, F and the rest of the unnamed nodes can not be travelled to.

Initial node selection

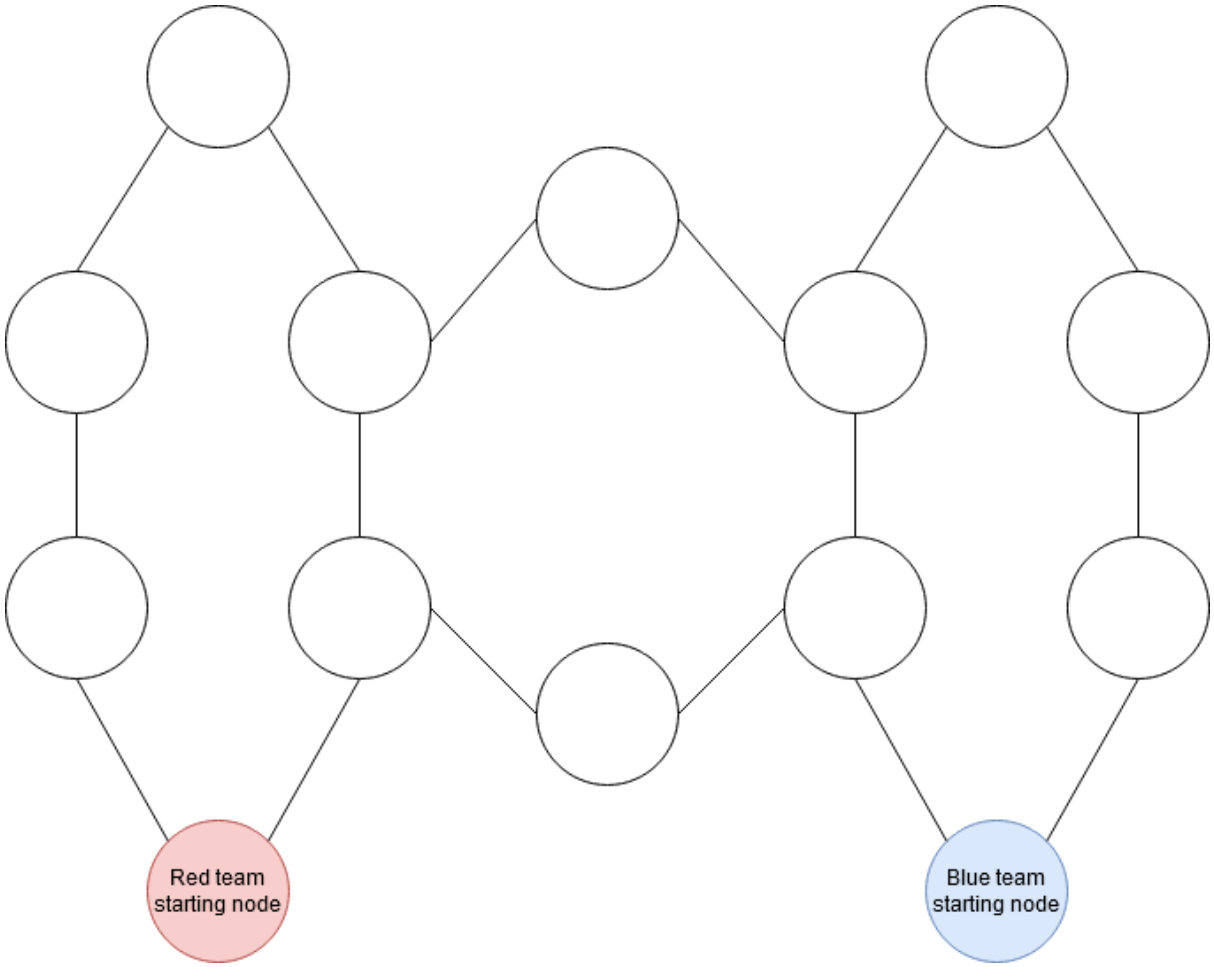
At first, the first team (out of 2) to make a turn will be decided by a coin flip.

The members of the winning team will then choose, which asset will be their starting node. They can choose a starting node from a list of predefined starting nodes. The node with most votes from team members will be selected. If multiple nodes have the same

amount of votes, a random one of them will be chosen.



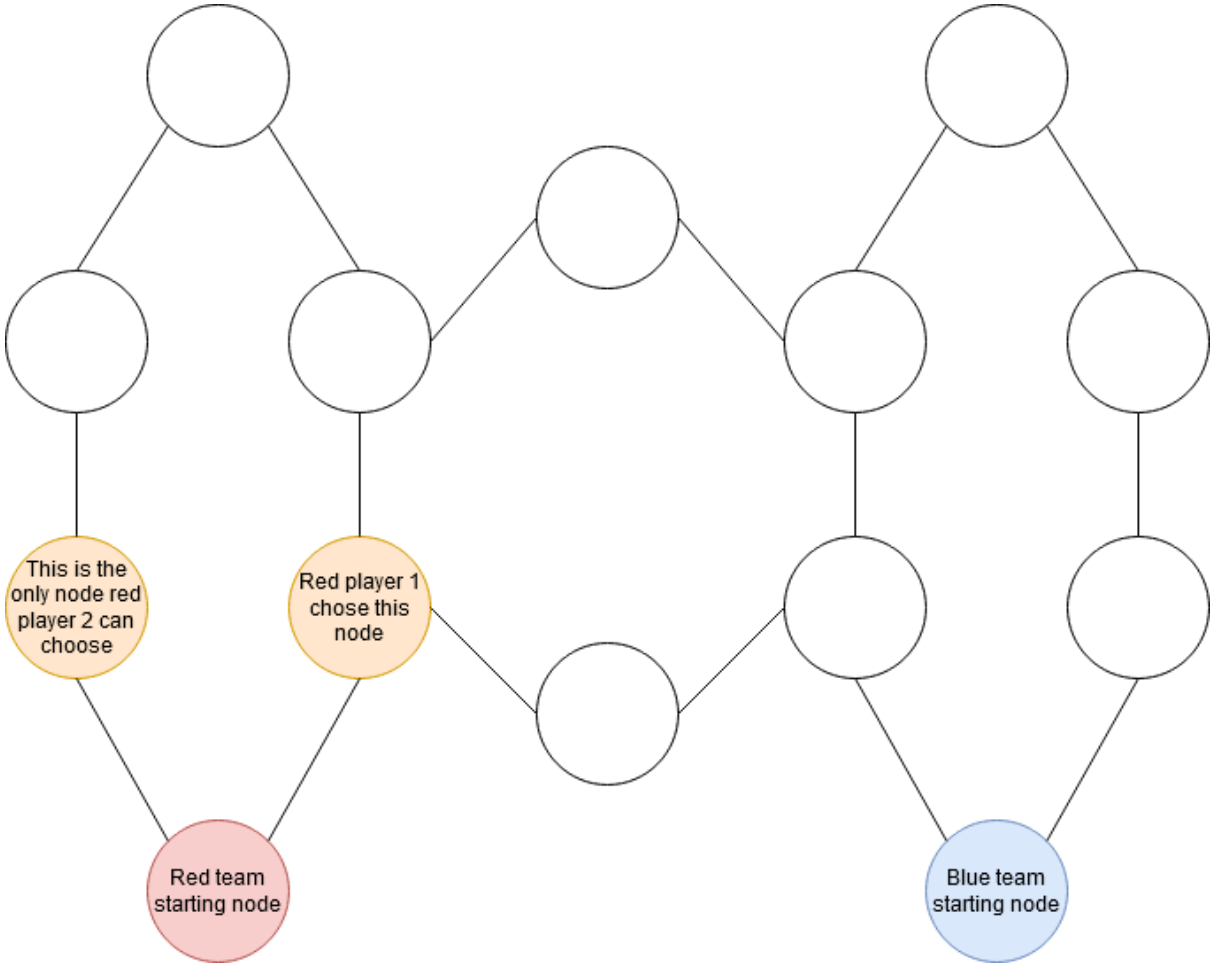
When this is done, a starting node is chosen by the other team. After the starting nodes have been chosen, one of the possible layouts might look like this:



Neutral asset conquering (NAC) phase

After the starting nodes have been decided, the neutral asset conquering phase of the game begins. The team that won the coin flip in the previous round makes the first move. In our case, let's imagine the red team won the coin flip. Each member of the red team will now choose an asset to attack. Players can only choose to attack nodes that are connected to already conquered nodes, in this case it's the starting node only. One asset can only be attacked by one player, that is, two members can not choose the same node, and each consecutive player has to choose a node from all the connected nodes that are not being

attacked.



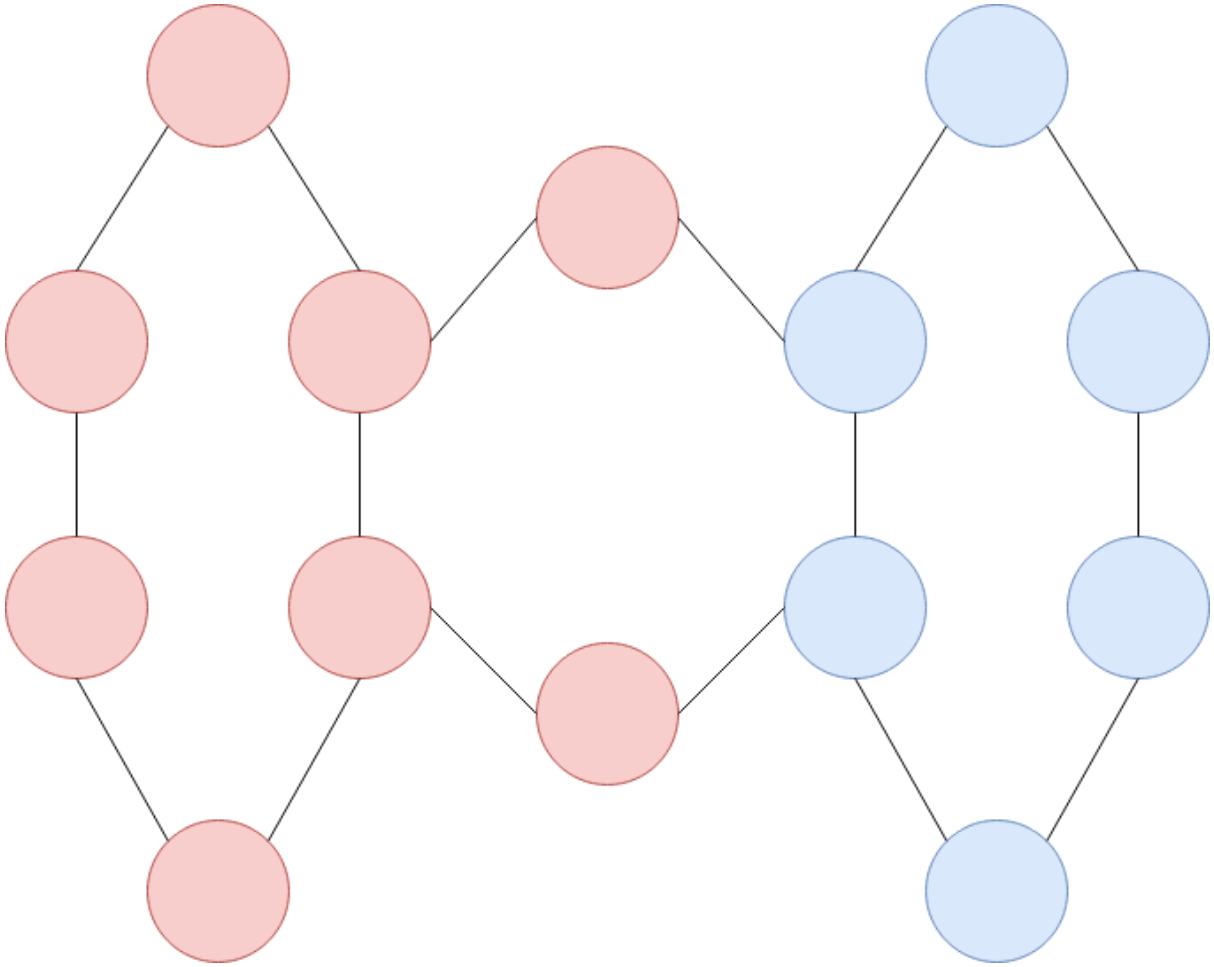
In this graph, only one red node is considered as “conquered”, and thus only that one node can be used by players as a starting point. That means that regardless of which node player 1 chose, player 2 can still only attack nodes adjacent to the starting node. Orange nodes on the graph are the ones that are considered “under attack”. When all the members of the red team have made their choices, the nodes that they attacked become conquered, and the other team, the blue in our case, gets to make a turn and repeats the same sequence of actions. This phase goes on as long as there are neutral nodes on the map.

While there are neutral nodes on the map, the players can not attack other players’ nodes. Players also can not interact with already conquered nodes.

Enemy asset conquering (EAC) phase

When there are no longer any neutral nodes left and thus the neutral asset conquering phase is over, the players must attack each other’s nodes. Let’s imagine this

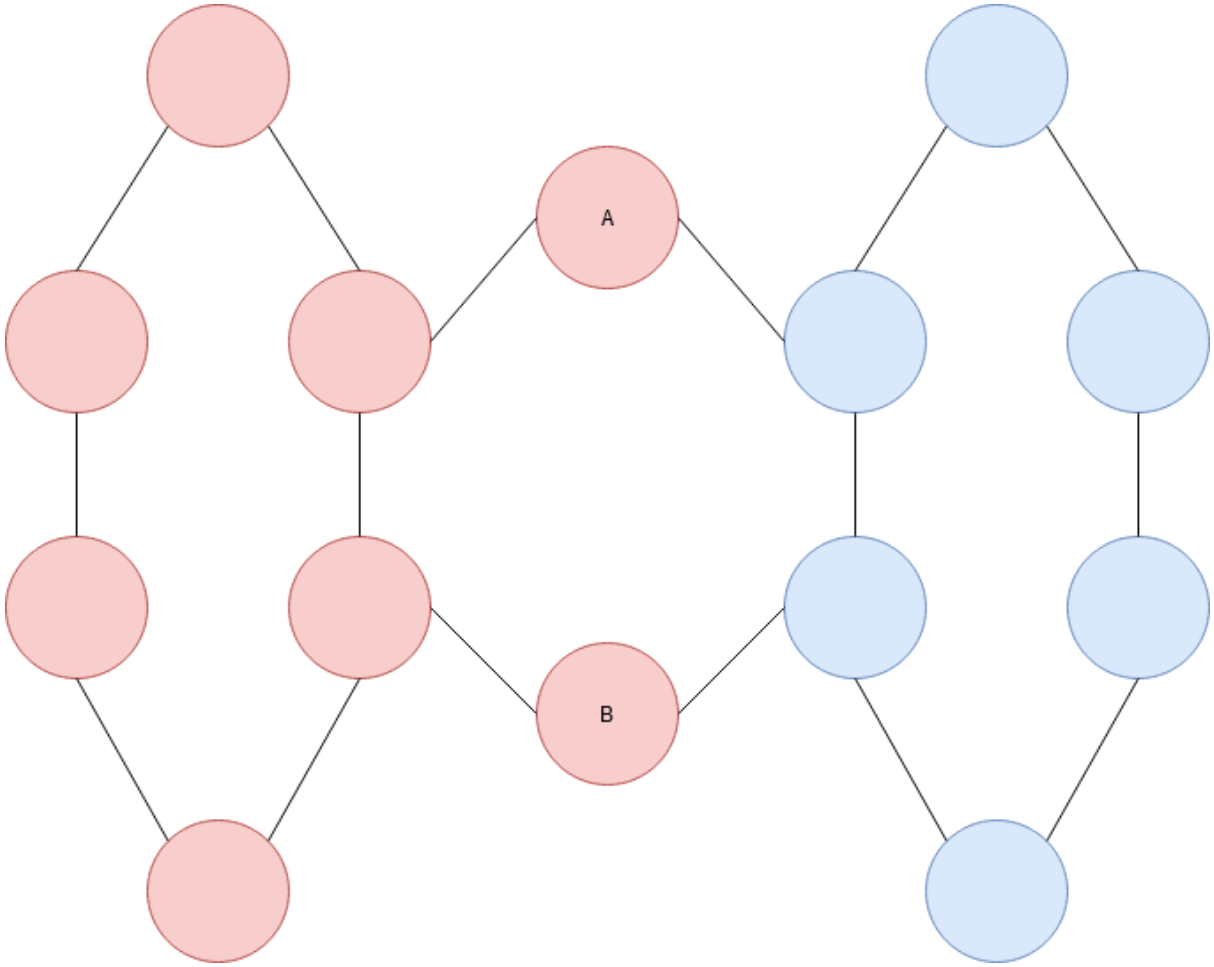
hypothetical case:



As we can see, there are no longer any neutral assets to attack, and therefore the only option to gain more nodes is to attack the other team. We can also notice that the red team was more aggressive, and conquered more nodes.

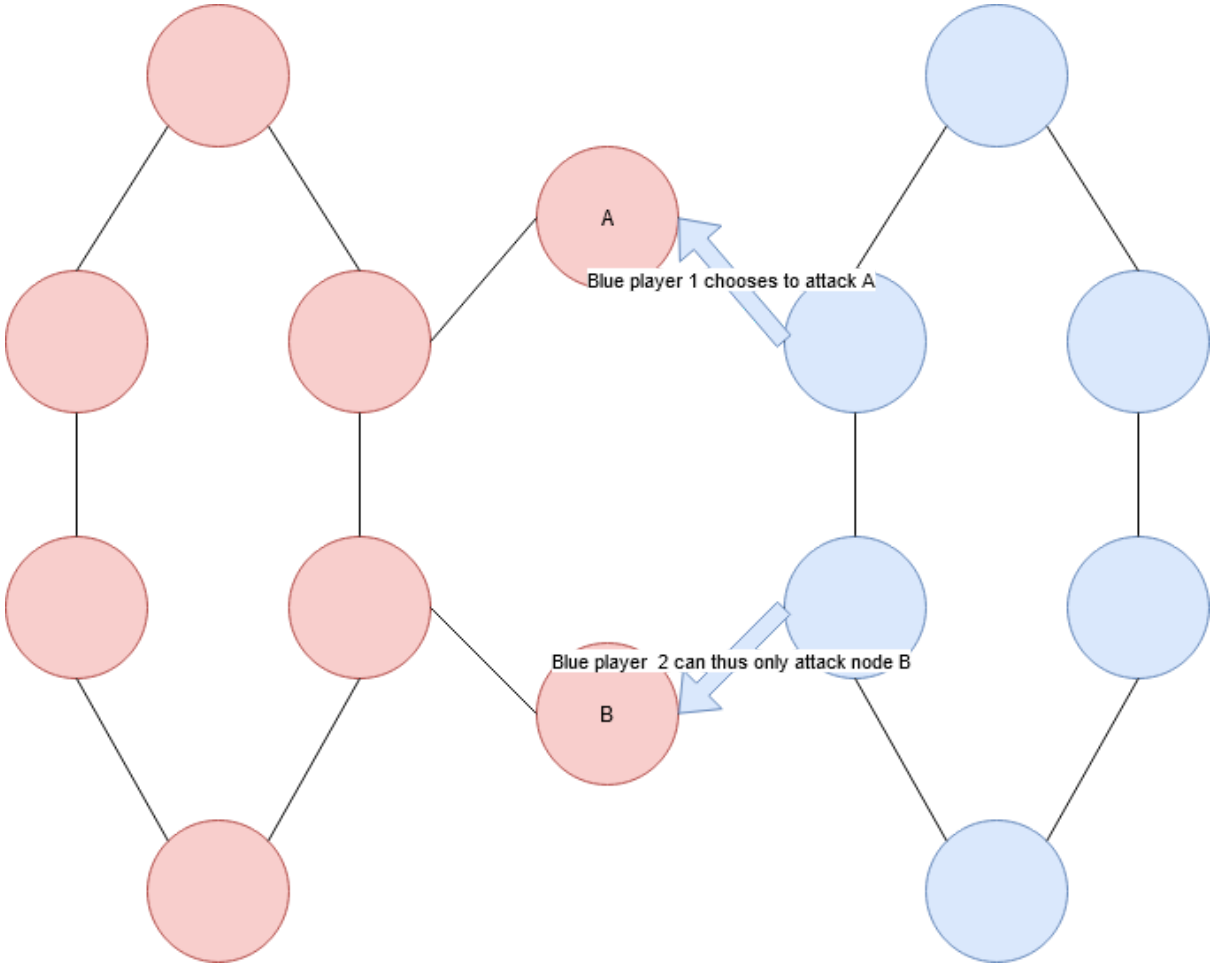
Now, the red team was the last team to conquer a neutral node, and therefore the blue team is now going to choose which asset to conquer. The rules for choosing a node to attack are the same as in the previous phase: players can only attack nodes that are connected to

already conquered nodes, and only one player can attack only one node.



Since it's now the blue team's turn, the first blue player can now choose either A or B, since these are the only nodes connected to the blue network. The first blue player chooses a node to attack, and a battle between him and a random red team member starts. The

second blue player can only choose the other node.



Before we talk about card battles themselves, we need to talk about one more aspect of the game: manipulating the card deck.

Bitcoins and deck management

The in-game currency is called Bitcoin.

At the beginning of the game, after the teams have chosen their starting nodes, both teams receive an equal amount of currency. During the neutral asset conquering phase, each player spends a share of their team's money to conquer an asset. This represents real-life spending money and resources to penetrate the security measures and gain access to the asset, and will be explained this way to the player. Gameplay-wise, this allows for team communication and cooperation, as well as different playstyles and strategies. One team could either spend more Bitcoin and conquer more assets in the initial NAC phase, or adopt a less aggressive, but also less expensive approach. Still, since each turn the players must attack an asset, at the end of the NAC phase both teams will be left with approximately the same amount of money, which helps us to avoid imbalanced techniques, for instance, where players amass too big amounts of money.

When the EAC phase begins, what's left of the team budget is split evenly between all team's members. The players now gain access to the main Bitcoin spending venue: the shop. During the NAC phase the player can not access the shop and any of its functionalities. The shop has three main functionalities:

- **Purchasing cards.** The players may purchase new cards. There is a small set of about three semi-random cards that the players might purchase, and each turn the cards on sale are changed. Cards vary in price based on their rarity and other parameters. The stock is not shared between players, and is generated separately for each player. Players might only buy one card per turn.
- **Removing cards from the deck.** The players may use the shop to remove any card from their deck. Players might only remove one card per turn.
- **Refreshing the stock.** The players may pay Bitcoin to immediately get a new set of cards for sale, without having to take a turn. This can be done as many times as the player wishes, as long as he has enough Bitcoin. With each consecutive stock refresh, the cost of this action increases.This increased refresh cost is not shared between players.

During NAC, the players will not have any ways to earn money. During EAC though, each asset owned by the team earns them a set amount of money, and before a player makes their turn, each player receives $\frac{total\ team\ income}{amount\ of\ players\ in\ that\ team}$amount of Bitcoin.

If the players are idle in the shop for a set amount of time, they are kicked out. This is to prevent players from staying in the shop and avoiding battles. This will be explained in more details below.

Card battles

When, during EAC, a player attacks an enemy asset, they become the attacker. Then, a random player from the opposing team joins the battle as the defender.

Each player must use the cards in their deck to defeat the opponent. Main card types are the attack cards, that directly deal damage to the opponent's health points, and the defence cards, that provide defence against some types of attacks. Each player has six cards in their hand at all times, they can only use 1 card each turn, and after a card is used, its effects are applied and it is moved into the discard pile. The player then takes a new card from the draw pile. If the draw pile is empty, the discard pile is shuffled and all cards from the discard pile are moved back into the draw pile. Players may choose to end their turn without using a card.

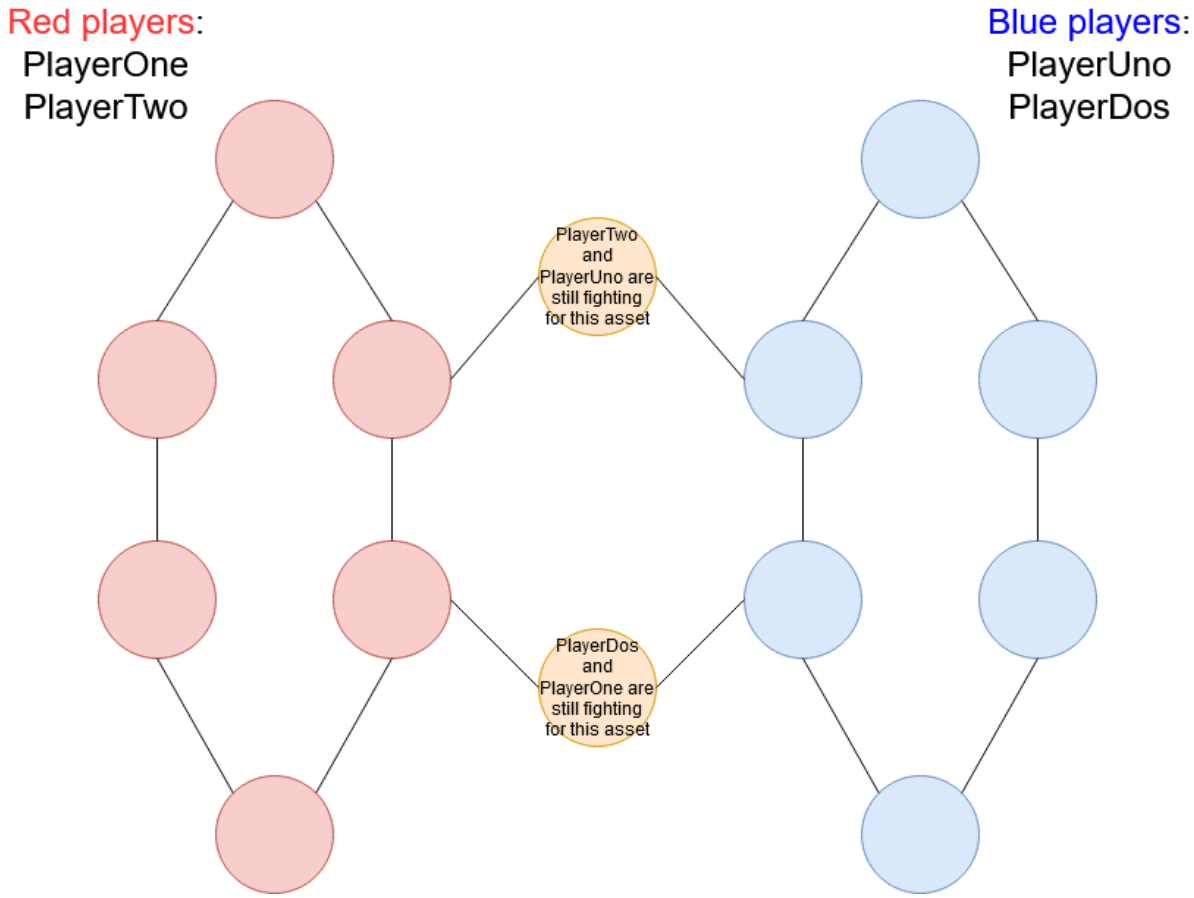
The defender makes the first move. Each player's goal is to lower the opponent's health point to 0 or less. If a player is idle for a set amount of time, his turn is ended automatically without using any cards. Should the player be idle for 3 consecutive turns, he automatically loses the battle.

Enemy asset conquering phase, continued

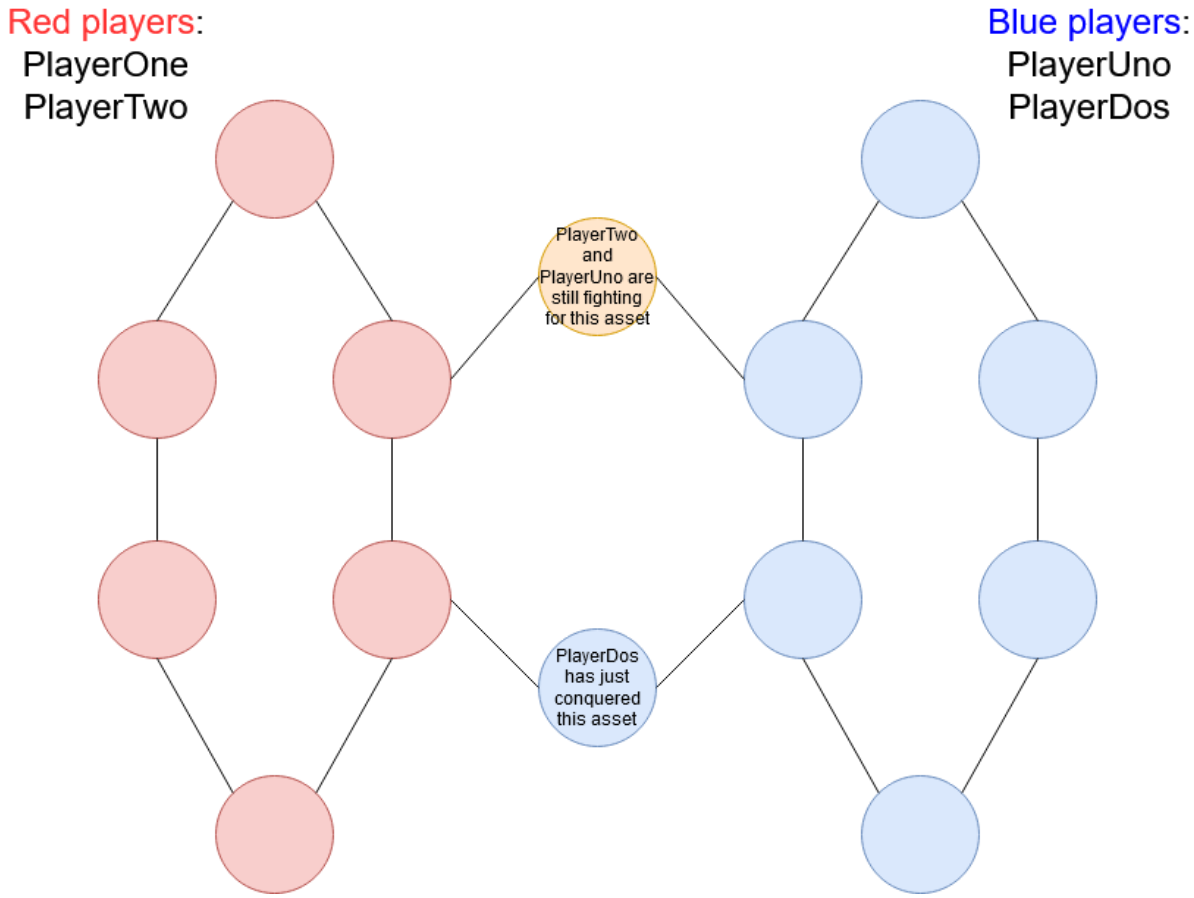
Whenever a card battle between two players ends, there is always a winner and a loser. If the attacker wins and the defender loses, the node that the teams were fighting for becomes owned by the attacking team. If the attacker loses and the defender wins, then the node does not change ownership.

EAC phase is also the phase where the players become more independent of each other, even in their own teams. It is inevitable that, as long as the total number of players is more than two (it must also always be an even number), some pairs of players will finish their battles earlier than others, and thus turns will no longer be synchronised between

players. Let’s take a look at this scenario, where the neutral asset conquering phase just ended, and both players from the blue team attack red team’s assets:



At some point, one of the player pairs will finish their round, and the other team will most likely still be in a battle. This is where the turns for players become separate.



For PlayerUno, it is still technically his turn, even though he is currently in a battle and that plays no role for him for now. For PlayerDos though, who just finished a battle with PlayerOne, the turn is over, and now PlayerOne will choose an asset to attack[1]. Even though both players can access the shop right now, only PlayerOne can attack. All of this just means that starting with the EAC phase, turns are more player-based than team-based.

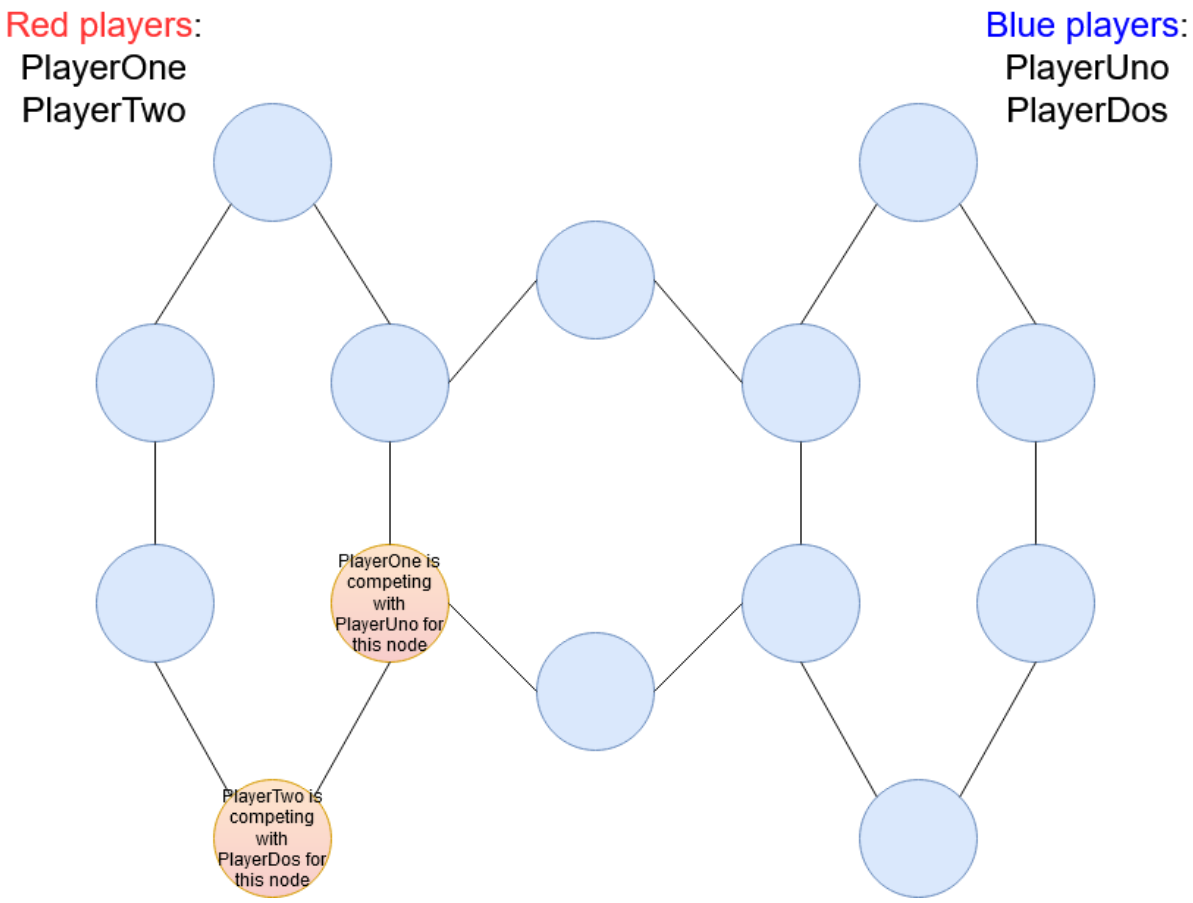
If PlayerOne attacks an asset, while PlayerDos is still in the shop, he will have to wait for his opponent to leave the shop. Otherwise players may prevent their opponents from managing their deck by starting a battle too quickly. At the same time, a player may not stay in the shop for too long to prevent players from avoiding battles this way.

Victory conditions

When the EAC phase starts and the teams start competing with each other, they have two main ways to win the game:

I. Conquer most enemy assets. If one team controls less assets than there are players in that team, it loses.

[1] In this case, he does not have much choice in what to attack though. The only asset he might attack is the same PlayerDos had just conquered. But this is only a problem on this simple example map, and will not be the case in the game, since the maps will be bigger and more interconnected.



In this situation, let's assume that both blue players are attacking. If any of the red players lose their battle, their team loses and the blue team wins. A more obvious victory condition would be to make a team conquer <u>all</u> enemy assets, but that is not possible to do. The reason is that if, for example, PlayerTwo loses, then at some point PlayerDos might have to make a move and the only asset he can attack at that point will be an asset that is already being fought for by PlayerUno and PlayerOne.

II. If the game goes on for too long, it will end even if both teams still own nodes. In this case, the players will receive a notification if the game will end soon (for example, 8 minutes before the end of the game), and when it ends, a score will be counted for both teams based on nodes owned and Bitcoin in posessment. This victory condition prevents the games from being too long and taking too much time.

Appendix C

Gameplay flow singleplayer

Gameplay flow

Single-player

After pressing the single player button, the player is being presented with an act choice menu.

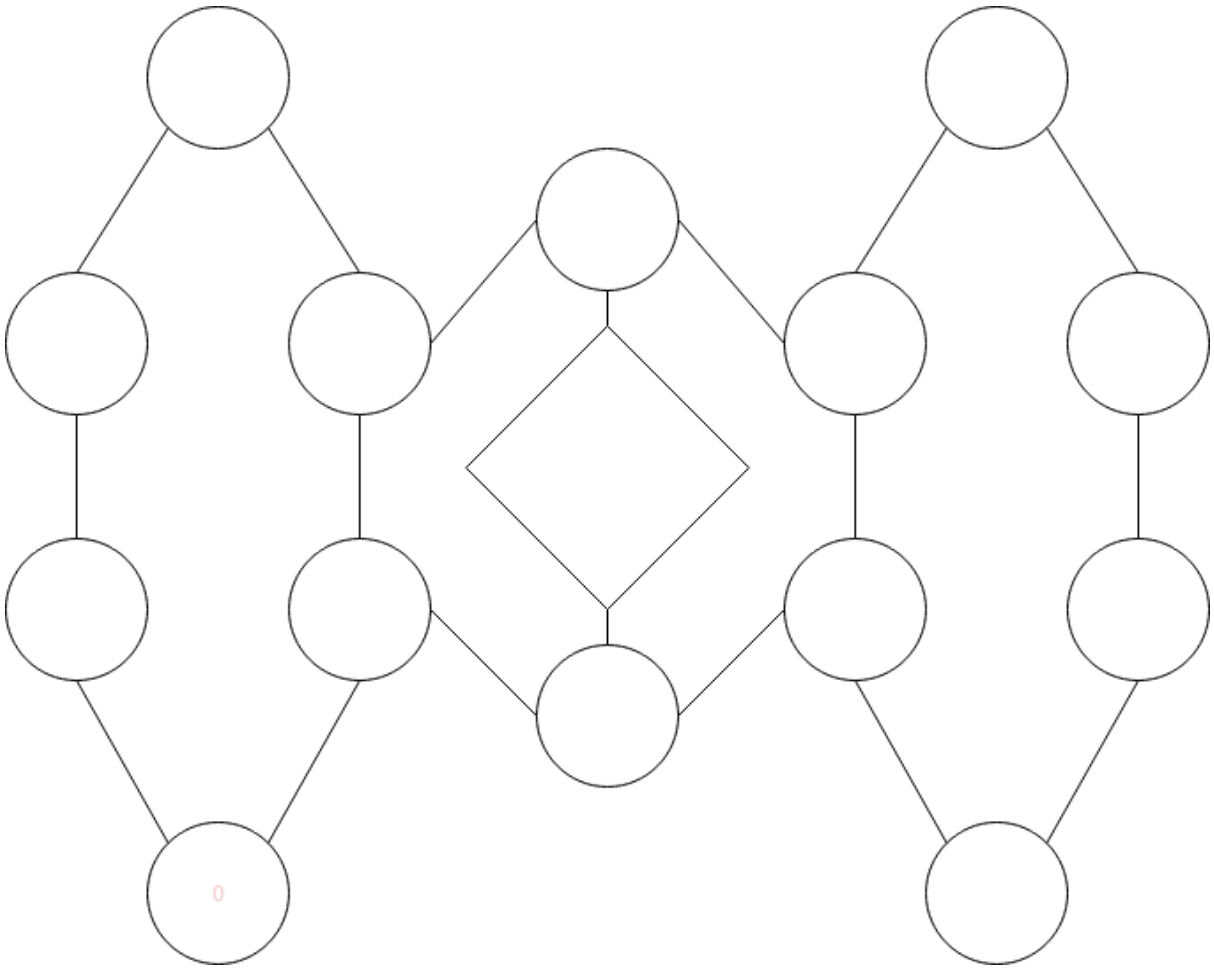
Game setting

In single-player, the player is going to be playing as a whistleblower, trying to collect revealing information about shady dealings of big corporations, corrupt government officials and institutions, as well as criminal organisations hidden in the Web, in order to expose them. Every country, institution, person *et cetera* in the game will be fictional.

Acts

The game will be divided into sections, which we will call acts. The acts will be represented as network topology maps, consisting of regular nodes, which will represent network assets, and special nodes. One special node that is alway going to be present is the boss node. In order to complete an act, the player will have to some amount of network assets – nodes – on the map. The player only gets access to the boss node after conquering the required amount of assets. There will be two types of acts:

- **A regular act.** The player will have to complete a set amount of such acts to proceed in the game. These acts will represent penetrating security measures of various organisations, as explained in the setting paragraph, in order to expose their wrongdoings.
- **The Final Act.** After the player has completed a set amount of regular acts, the Final Act will be unlocked. If the player completes this act, he will win the game.



Above is the representation of what an act might look like. Notice the big diamond in the middle. It represents the boss node.

There will be a predefined starting node in each act. The player will start the act there, and then will move along the lines connecting the nodes to conquer other assets.

Different acts will have different difficulty setting, and it will change the size of the act and the difficulty of battles.

Act choice menu

The act choice menu will be one of the main aspects of the single-player mode. As it has been described previously, in order to progress in the game, the player will have to complete acts.

Each time the player enters this menu, the player will be able to choose between four different options. The three first options will be regular acts. The fourth option will be the Final Act. In the beginning, the Final Act option will be locked, as the player has to complete a set amount of regular acts to unlock it.

The player has to choose an act to start it. Each act gives you a varying amount of score points with the top option giving you the least score and the bottom one giving you the most score. The bigger the amount of score gained from act is, the higher is the act's difficulty setting.

The score points belong to three different categories:

- Corporations
- Governments

- Criminal organisations

Each act may give a score in one or multiple categories.

This score can be used to unlock debuffs for enemies or buffs for the player. This means that the player may risk it for the biscuit and choose more difficult acts that will be beneficial in the long run, or choose easier acts but potentially sacrifice being able to purchase buffs and debuffs.

Combat

When an act is chosen, the player finally enters the topology of an act. The player is then able to conquer the nodes in that act. Whenever the player chooses to attack an unconquered node, a card battle combat with an AI opponent starts. This is more closely described in the combat document.

Winning a game

After defeating a boss node in an act, the act will be counted as finished, and the score in the act choice menu will be updated. After completing a set amount of acts, the option to start the Final Act will become available in the act choice menu and the regular acts will be locked. The Final Act will be considerably more difficult than the regular acts. If the player finishes the Final Act, he is considered to have completed the game.

Should the player lose a battle during any of the acts they will receive a temporary debuff until the end of the act, as well as they will have to fight for the node attack. If the player loses a battle a set amount of times, they lose the game permanently and have to start from scratch

When the player defeats an enemy, he gains Bitcoins.

- This can not be finished currently. Something is missing.
- This is a design choice that was not planned/not discussed
- This is the same but of slightly more importance
- This is wording/phrasing problem
- This is a problem of wording, a substitute word that is more descriptive should be used
- This is stuff that is no precisely discussed/conceptualised, and thus is suggested for removal to keep this document as precise as possible

Appendix D

Project plan

OS Runner

Kacper Lewandowski, Artūrs Umbraško, Daniel Dahl

January 2021

I Goals and limitations

I.I Background

Currently, the market for educational cybersecurity games is quite shallow. Out of the not so many games that exist in this field, even fewer are combining the educational aspect with being actually fun to play. The project is an attempt to change that, to improve and accelerate the cybersecurity education by introducing a way for students to engage in tangential learning.

Based on a different card game - *Intrusion Attempt* - also in development by our employer for three years now, the game correlates real-life cybersecurity practises and concepts with common video game and card game mechanics and concepts. This way, by engaging in the gameplay, the students will also increase their understanding of the real underlying topics.

Gamification of the learning process with the goal of improving high school and undergraduate students' understanding of the cybersecurity topic are considered by us to be the main benefits of this project. The gamified tangential learning experience can also also mean an easier introduction into the topic for those previously not at all familiar with it. This can in turn lead to improved performance in cybersecurity-related classes, greater interest in cybersecurity-related studies, as well as many other positive education effects and applications.

Our employer is a former NTNU educator himself, and the game is going to be used as an educational tool as well. It shall also serve as a knowledge base with links and references to further materials about the topics.

All three of us are interested in programming and games, which is why we chose to study game programming at NTNU in Gjøvik. This is also why we

chose to do this task as it seemed like a perfect fit considering our interests in various games, but also our knowledge in programming and game design. We also have some experience with the genre and are familiar with it, since we have played games like: Slay the Spire, Ascension and Poker Quest RPG, and during our studies we have already made a 2D deck-bulding roguelike game.

I.II Goals

The goal of **OS Runner** is to have a fun strategical deck-building card game trying to facilitate and stimulate cybersecurity education through tangential learning and associations. The game is intended to create an engaging gameplay experience that will also widen the players' understanding of real-life hacking, penetration and red teaming scenarios.

As a group our goal during the project is to learn more about cybersecurity as well as game design. Considering we've also made a game during a previous course, we also want to know what it's like working in a multidisciplinary team on a bigger project. Improving our game programming and design knowledge and skills is too a very beneficial and interesting experience for us.

We also have no previous experience with making multiplayer or mobile games. This could be helpful in future projects as well, considering that adding multiplayer to a game opens up a plethora of possibilities in regards to gameplay. Having experience with making mobile games will also be professionally useful.

I.III Limitations

While we have worked on Android software development previously, not all team members have experience with developing games on Android before. We consider learning new technology to be valuable experience.

We will be developing the game for the latest Android, but we will do our best to ensure a good level of backwards compatibility as well, if the opportunity arises. The game is supposed to have multiplayer support, and we have never worked with multiplayer games before, especially on mobile devices.

II Description and scope

II.I Description

The game will be based on a card game project by the employer. The main mechanics will be revolving around a deck-building and card battle gameplay. We are going to develop both single-player and multiplayer aspects.

Cybersecurity will be the main theme of the game. The cards will be based on different methods and techniques used by malicious thread actors and suitable for teaching penetration testing and red teaming

The game will teach players some basic aspects and ideas of data security, as well as in general raise data security awareness and interest in high school students, undergraduates and graduates.

Two main screens will be included: a map, that will resemble a network topology where players will be able to roam and compete for score points, and a card battle screen, where the actual card battles will take place.

The development will be first and foremost done for mobile devices, and thus the mechanics and controls will be optimised accordingly. The game could potentially be ported to PC as well.

II.II Scope

At the end of the project, we aim to have a fully functional game with the core gameplay features implemented and a server for multiplayer. None of the members have experience in creating game art, thus our team will primarily focus on creating code for the game mechanics and with the visuals will be a lesser focus.

We also have to consider the limited time we have been given for the project. We will be working with new technologies and have to take into consideration the time needed to learn them. We have previously learned how to create games on PC and never made a game for Android.

II.III MoSCoW

We have also used the MoSCoW method to classify the most important features that have to be implemented for the game to fully function. Here is the MoSCoW:

Must have:

- We must have a card base
- We must have multiple card types
- We must have card battles
- We must have functionality to add cards to the deck
- We must have Local Network multiplayer
- We must have map topology
- We must have map navigation
- We must have a main menu
- We must have Windows support

Should have:

- We should have a persistent player profile
- We should have good network performance
- We should have working single-player mode
- We should have working Bitcoin shop
- We should have options menu
- We should have all of the card types
- We should have in-game card encyclopedia
- We should have nice visuals in the game
- We should have nice audio design in the game

- We should have functionality to easily add new cards into the game
- We should have Linux and Android support

Could have:

- We could have majority of the cards
- We could have a more complex scoring system
- We could have a team-up functionality
- We could have a reputation system
- We could have specialisations
- We could have assisting of other players in multiplayer
- We could have working high score menu
- We could have iOS support

Won't have:

- We will not have ALL of the cards
- We will not have a story
- We will not have good visuals in the game
- We will not have good audio design in the game

III Organisation

III.I Roles

Artūrs Umbraško: Communications Manager, Developer
Kacper Lewandowski: Secretary, Developer
Daniel Dahl: Lead Designer, Developer

III.II Responsibilities

Each of us will also have special tasks and specialisation, but, since for every member of the team this is a programming student, everyone will be also working on creating the code. The Communications Manager is responsible for team communications, weekly meetings, as well as contacting the supervisor and the employer.

The Secretary will be responsible for taking notes and making transcripts of all meetings, both simply between team members, as well as between team members and the supervisor and the employer.

The Lead Designer will be, in addition to programming tasks, responsible for the artistic choices made during the development, such as graphical and audial design.

III.III Group rules

After some discussion, we decided on the following rules:

- Work time will be from 08:00 to 15:00 Oslo time during the weekdays.
- Every week we will have a meeting with the product owner and the supervisor.
- If a problem arises, a discussion with all group members will commence to resolve conflict.
- If the problem can not be resolved by the group members, the group will contact the supervisor to discuss the problem at hand.
- The group must be present at the decided timestamp, if not a notice must be issued.

IV Development model and strategies

IV.I Model

During the planning stage, we spent quite some time arguing and discussing what kind of model to use. The two final picks, that we could not decide between, were Scrum and Kanban. The Scrum approach offers better

time management and control. It is more decisive and strict and implements more internal deadlines. Kanban on the other hand offers more flexibility, and the tasks can be selected by each member personally; which allows for easy task distribution. Kanban also offers us to plan continuously, which is definitely going to be helpful in our project, since we expect the plans to change relatively often.

After some research and some more discussions, we decided to combine both advantages and use the Scrumban model. We will have week-long iterations, during which we will use a Kanban board and a Work-In-Progress limit to complete tasks and implement features. The very first iteration will be two weeks long, since the initial prototype will need a bit more time to develop. At the beginning of every week we will be having a meeting to create User Stories for features that will be added to the Scrumban board. First couple of iterations will be used to determine the team velocity. We also aim to have a working high fidelity prototype to show to the employer at the end of each iteration.

V Quality assurance

The Scrumban Work-In-Progress and Scrumban board allows for flexibility when choosing what task you want to work on. This makes changing, reworking or adding features to the game according to the owners vision a much easier task. Each member of the group can only work on one task at a time. This helps to assures that each member knows what they are working on and also helps organise a steady workflow. The steady workflow helps us in estimating the number of tasks that can be done during each iteration. All in all this ensures a higher quality product and prototype.

V.I Weekly checks

Every finished iteration, we will send the prototype to the employer to discuss the results and the general progress. The progress will also be discussed with our supervisor. With him we will also be discussing how effective our strategies are.

V.II Documentation and syntax/standard use

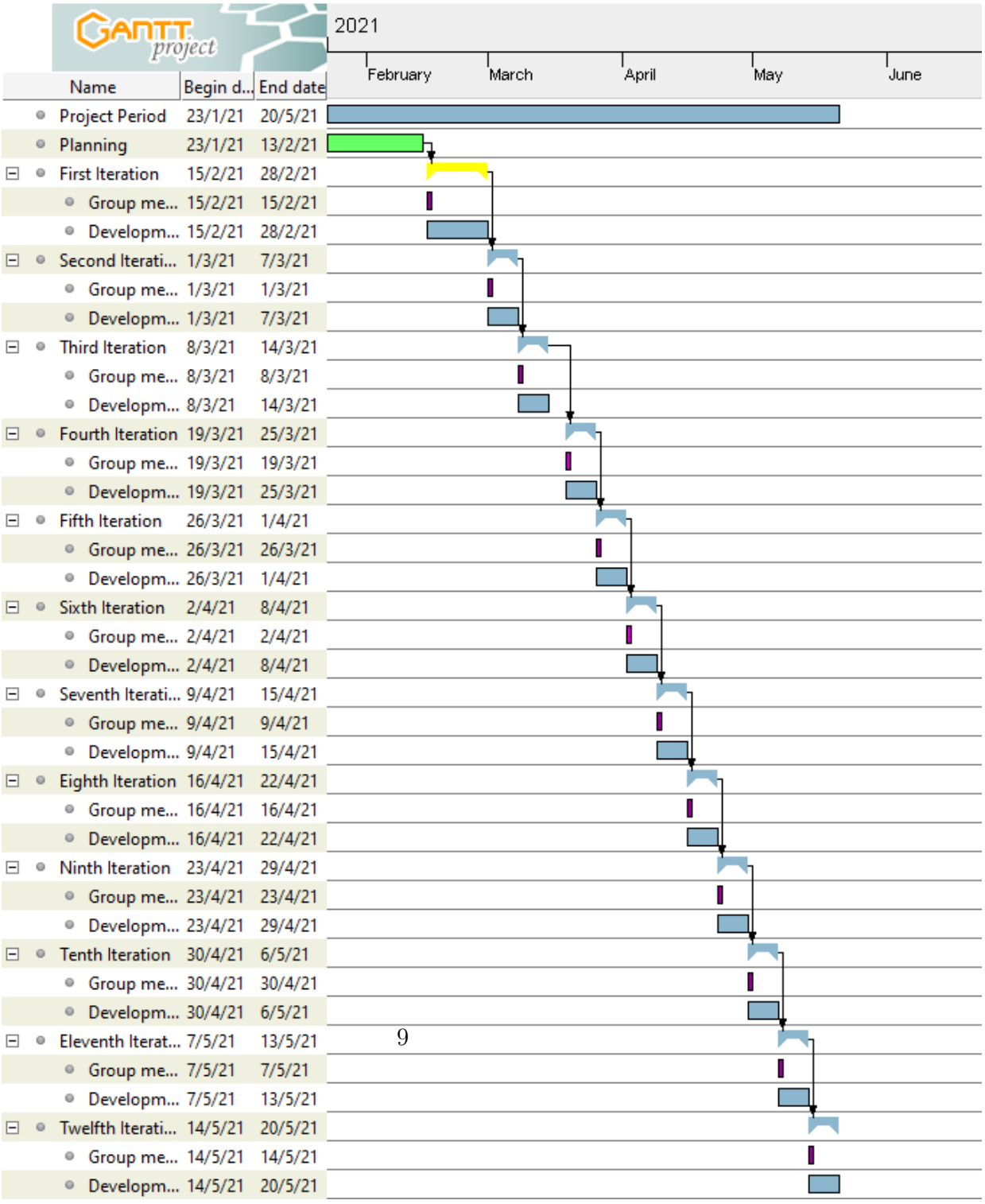
To ensure our code is readable, we will be using common practices and style guides. We will comment what is necessary parts of the code to make it easily understandable for everyone on the group, the employer and the supervisor. We will also include a documentation and readme files in our repository to make it easy for the employer or the future maintainers to use and work with the product.

V.III Version control

Our team will use git version control to make the development easier, to make the collaboration more efficient, to make it simple to deploy and present the software for the employer, as well as for security reasons. Version control will also allow us to make mistakes without the consequences for the rest of the team, and make it easier to work on multiple things at once.

The other reason for version control is due to its simplicity. We will be able to seamlessly share our code with each other with very little effort. This will allow us to work without worrying about the technical aspect of sharing the code with each other.

VI Gantt


GANTT project
2021
February
March
April
May
June
Name
Begin d...
End date
Project Period 23/1/21 20/5/21
Planning 23/1/21 13/2/21
First Iteration 15/2/21 28/2/21
Group me... 15/2/21 15/2/21
Developm... 15/2/21 28/2/21
Second Iterati... 1/3/21 7/3/21
Group me... 1/3/21 1/3/21
Developm... 1/3/21 7/3/21
Third Iteration 8/3/21 14/3/21
Group me... 8/3/21 8/3/21
Developm... 8/3/21 14/3/21
Fourth Iteration 19/3/21 25/3/21
Group me... 19/3/21 19/3/21
Developm... 19/3/21 25/3/21
Fifth Iteration 26/3/21 1/4/21
Group me... 26/3/21 26/3/21
Developm... 26/3/21 1/4/21
Sixth Iteration 2/4/21 8/4/21
Group me... 2/4/21 2/4/21
Developm... 2/4/21 8/4/21
Seventh Iterati... 9/4/21 15/4/21
Group me... 9/4/21 9/4/21
Developm... 9/4/21 15/4/21
Eighth Iteration 16/4/21 22/4/21
Group me... 16/4/21 16/4/21
Developm... 16/4/21 22/4/21
Ninth Iteration 23/4/21 29/4/21
Group me... 23/4/21 23/4/21
Developm... 23/4/21 29/4/21
Tenth Iteration 30/4/21 6/5/21
Group me... 30/4/21 30/4/21
Developm... 30/4/21 6/5/21
Eleventh Iterat... 7/5/21 13/5/21
Group me... 7/5/21 7/5/21
Developm... 7/5/21 13/5/21
Twelfth Iterati... 14/5/21 20/5/21
Group me... 14/5/21 14/5/21
Developm... 14/5/21 20/5/21

Appendix E

Contract

NTNU

Norwegian University of
Science and Technology

PROJECT AGREEMENT

between NTNU Faculty of Information Technology and Electrical Engineering (IE) at Gjøvik (education institution), and

Danny Lopez

(employer), and

Artūrs Umbraško, Kacper Lewandowski, Daniel Dahl

(student(s))

The agreement specifies obligations of the contracting parties concerning the completion of the project and the rights to use the results that the project produces:

1. The student(s) shall complete the project in the period from 11.01.21 to 20.05.21.

The students shall in this period follow a set schedule where NTNU gives academic supervision. The employer contributes with project assistance as agreed upon at set times. The employer puts knowledge and materials at disposal necessary to complete the project. It is assumed that given problems in the project are adapted to a suitable level for the students' academic knowledge. It is the employer's duty to evaluate the project for free on enquiry from NTNU.

2. The costs of completion of the project are covered as follows:
- Employer covers completion of the project such as materials, phone/fax, travelling and necessary accommodation on places far from NTNU. Students cover the expenses for printing and completion of the written assignment of the project.
- The right of ownership to potential prototypes falls to those who have paid the components and materials and so on used to make the prototype. If it is necessary with larger or specific investments to complete the project, it has to be made an own agreement between parties about potential cost allocation and right of ownership.

3. NTNU is no guarantor that what employer has ordered works after intentions, nor that the project will be completed. The project must be considered as an exam related assignment that will be evaluated by lecturer/supervisor and examiner. Nevertheless it is an obligation for the performer of the project to complete it according to specifications, function level and times as agreed upon.

4. All passed assignments will be registered and published in NTNU Open, which is NTNUs open archive.

   This depends on that the students sign a separate agreement where they give the library rights to make their main project available both on print and on Internet (ck. The Copyright Act). Employer and supervisor accept this kind of disclosure when they sign this project agreement, and they must possibly give a written message to students and head of Department if they during the project period change view on this kind of disclosure.

   The total assignment with drawings, models and apparatus as well as program listing, source codes and so on included as a part of or as an appendix to the assignment, is handed over as a copy to NTNU who free of charge can use it in lessons and in research purpose. The assignment or appendix cannot be used by NTNU for other purposes, and will not be handed over to an outsider without an agreement with the rest of the parties in this agreement. This applies as well to companies where employees at NTNU and/or students have interests.

5. The assignment's specifications and results can be used by the employer's own work. If the student(s) in its assignment or while working with it, makes a patentable invention, relations between employer and student(s) applies as described in Act respecting the right to employees' inventions of 17th of April 1970, §§ 4-10.

6. Beyond the publishing mentioned in item 4, the student(s) have no right to publish his/hers/theirs assignment, fully or partly or as a part of another work, without consensus from the employer. Equivalent consent must be made between student(s) and lecturer/supervisor regarding the material placed at disposal by the lecturer/supervisor.

7. The students shall hand in the assignment with attachments electronic (PDF) in NTNU's digital exam system. In addition the students shall hand in a copy to the employer.

8. This agreement is drawn up with one copy to each party. On behalf of NTNU it is the head of the Department/Group that approves the agreement.

9. In each case it is possible to enter separate agreement between employer, student(s) and NTNU who closer regulate conditions regarding issues such as ownership, further use, confidentiality, cost coverage, and economic utilization of the results.

If employer and student(s) wish an additional or new agreement, this will occur without NTNU as a partner.

10. When NTNU also act as employer, NTNU accede to the agreement both as education institution and as employer.

11. Possible disagreements concerning understanding of this agreement are solved by negotiations between the parties. If consensus is not achieved, the parties agree that the disagreement is solved by arbitration, according to provision in Civil Procedure Act of 13th of August 1915, no 6, chapter 32.

12. Participants by project implementation:

NTNUs supervisor (name): Aland Mendoza

Employers contact person (name): Danny Lopez

Student(s) (signature): Artūrs Umbraško A.UM date 21.01.21

Kacper Lewandowski date 21.01.21

Daniel Dahl date 21.01.21

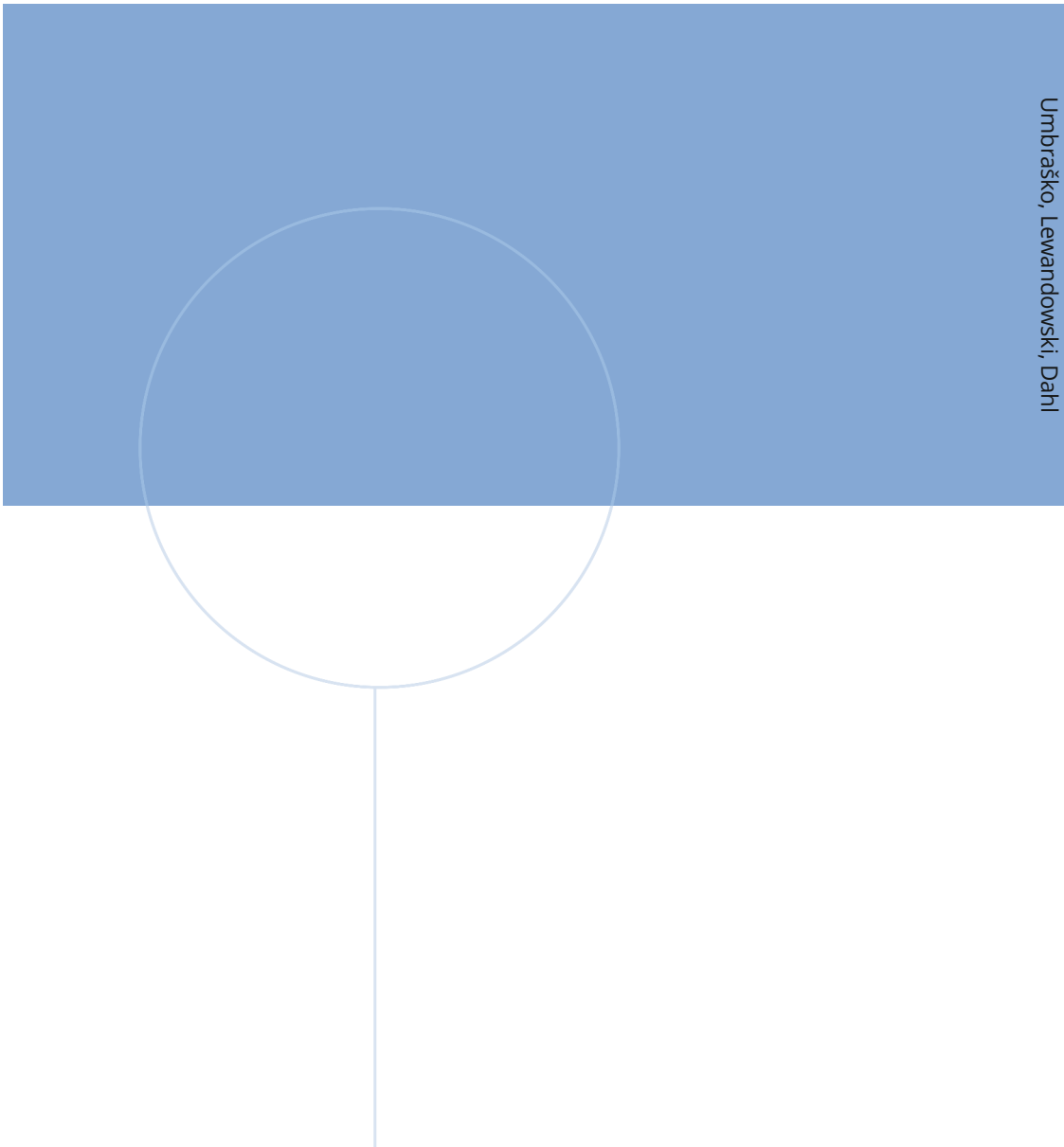
date

Employer (signature): Danny López M. date 27-01-2021

The Project Agreement is to be handed in in digital version in Blackboard. Digital approval by head of the Department/Group.

If a paper version of the Agreement is needed, is must be handed in at the Department in additional.

Head of Department/Group (signature): date



Umbraško, Lewandowski, Dahl

NTNU

Kunnskap for en bedre verden