UNIVERSITY OF TARTU
Institute of Computer Science
Software Engineering

Stanislav Belogrivov

Cogbug – A Mobile Tabletop Game

Master's Thesis (30 ECTS)

Supervisor:
Raimond-Hendrik Tunnel, MSc

Cogbug – A Mobile Tabletop Game

Abstract:

This thesis covers the design and development of a mobile version of the tabletop game application to play *Cogbug*. The thesis starts with a brief overview of mobile games and with background information regarding the physical tabletop version of *Cogbug*. The thesis continues with the digital Cogbug game specification that was discussed and formulated together with the author of the physical tabletop game (Customer) via requirements analysis. Theoretical research of the tools for development and testing as well as practical implementation and product testing form the main body of the thesis. Practical implementation focuses on delivering a Minimum Viable Product with a functioning multiplayer gamemode. The application went through platform and usability testing phases to ensure the quality and stability.

Keywords:

Cogbug, mobile application, multiplayer, Unity, mobile game, game development, tabletop game digitalization

CERCS: P170 Computer science, numerical analysis, systems, control

Keerdvärk - mobiili lauamäng

Lühikokkuvõte:

See lõputöö käsitleb lauamängu *Keerdvärk* mobiilversiooni disaini ja arendust. Lõputöö algab lühikese ülevaatega mobiilimängudest ja taustteabest mängu Cogbug füüsilise lauaversiooni kohta. Lõputöö jätkub digitaalse Keerdvärk mängu spetsifikatsiooniga, mida arutati ja formuleeriti koos füüsilise lauamängu (Klient) autoriga nõuete analüüsi kaudu. Töö põhiosa moodustavad arendus- ja testimisvahendite teoreetilised uuringud ning praktiline juurutamine ja toote testimine. Praktiline juurutamine keskendub minimaalse elujõulisuse toote tarnimisele koos töötava mitme mängijaga mängurežiimiga. Rakenduse kvaliteedi ja töökindlust uuriti kasutajatestimise ja platvormitestimise kaudu.

Võtmesõnad:

Cogbug, Keerdvärk, mobiilirakendus, multiplayer, Unity, mobiilimäng, mängude arendamine, lauaarvutimängude digitaliseerimine

CERCS: P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Table of Contents

1. Introduction

Nowadays digital devices allow us to get information from any place with network access[1]. Additionally, digital solutions allow us to connect with each other without regard for distance between individuals, making it possible to communicate with less effort. Because of that, more and more people tend to take interest in digital means of entertainment - playing digital versions of games or meeting up online instead of doing it physically. The result of this is that the digital market is becoming a solid and large addition of the gaming industry.

This trend is also related to the fact that the popularity of digital devices is increasing. Over the time, such devices have become more user-friendly and compact[2]. Furthermore, the decrease of costs and the increase of availability of internet services has also caused an interest in digital services [17, 18]. As one can access any library in the world at any place or time, the interest and trends would shift towards a faster, safer and simpler approach. The safety in this assumption is based on the fact that information is more accessible and is, in most cases, limitless. With the earlier example of a library, the physical one can run out of printed copies of a book, whereas an electronic one cannot.

It is important to note that digital solutions and trends do not completely take over their physical counterparts. For example, sports games like football and basketball do have digital counterparts (i.e. *FIFA* and *NBA 2K* franchises respectively), the interest in real-world sports is still growing[3].

This thesis covers the development process of the MVP for a digital strategy tabletop game *Cogbug*. The development was carried out for the authors of the physical version of *Cogbug*, Edulab OÜ. The development team size increased throughout the development process, consisting only of the author of the thesis during the early stages of development. After the work on the networking module was started, the team size increased to three, as Tarvo Metspalu, the Edulab representative, volunteered to join the process as a digital artist and

[1] Garrett, B., 2017. *How Technology Is Driving Us Toward Peak Globalization*. [online] Singularity Hub. Available at: https://singularityhub.com/2017/10/22/peak-globalization-is-the-path-to-a-sustainable-economy/ [Accessed 30 March 2020].
[2] Computerhistory.org. (n.d.). *Computers | Timeline of Computer History | Computer History Museum*. [online] Available at: https://www.computerhistory.org/timeline/computers/ [Accessed 9 Nov. 2019].
[3] Statista. (n.d.). *FIFA World Cup average & total attendance 1930-2018 | Statista*. [online] Available at: https://www.statista.com/statistics/264441/number-of-spectators-at-football-world-cups-since-1930/ [Accessed 9 Nov. 2019].

concept art designer. Additionally, Madis Kaspar Nigol has joined the team at the same time as a sound artist.

Chapter 2 provides background regarding the games and video games as means of entertainment. Afterwards, it provides reasoning why developing a digital mobile game is a reasonable investment.

Chapter 3 starts off with the description of the physical game of *Cogbug* and its rules. This is followed by requirements and functionality that was initially intended to be transferred to the digital version of the game. This was achieved by performing several discussions with the Edulab representative. The general roadmap of the development process focused on developing a single-player MVP first, then developing a multiplayer solution around it as the functionality for each player is the same, according to game rules.

Chapter 4 covers the development process. This chapter focuses on issues that arose during the development process as well as the solutions that were chosen in order to resolve them. This chapter covers the implementation of the multiplayer module and its integration with the previously developed components.

Chapter 5 covers the methods that were used to test the application. The testing was performed using two approaches - internal, which was done by the development team, and closed, which involved unbiased testers. Both approaches included platform testing, resulting in the application being run on 8 different devices during the testing, including both smartphones and tablets. Additionally, network connection stability and performance were observed to evaluate the quality of the developed multiplayer module. Closed testing also aimed to test the application's usability.

The Appendix contains documents that describe the initial requirements from the Customer, rules of the physical game of *Cogbug*, documented meetings. Additionally, links to recordings of testing sessions and testing form feedback are provided in Appendix V.

2. Motivation

Games, as a part of the entertainment industry and human culture, have been known to exist since around 2600 BC[4]. With the advances of technology and culture, games and their rules have also changed. By the 15th century, the first "modern" tabletop games had started to emerge. An example of that can be chess. While the game of chess itself has existed before the 15th century, the rules were being changed over time and only became similar to the current ones by that time[5]. Still, the game and the rules were and are getting adjusted afterwards as well. With the increased interest over chess, multiple popular variations like rapid chess have emerged, with their own sets of additional rules. Nowadays, with the advances of digital technologies, various digital versions of physical games are being created.

Most games require not only knowledge and skills, but also specific equipment specifically designed for a particular game. For example, for most people, a chess game is unplayable without the chess board and pieces[6]. This also partly applies to stationary computers, as they require peripheral devices to be properly usable but do not require much knowledge nowadays. Mobile devices are designed to be portable and simple to use. Therefore, they do not require additional devices or skills. As a result, it is possible to expand the target audience in this field by creating a mobile application.

On the other hand, such applications generally require less resources in both initial development and distribution than physical games. Distribution of digital solutions is significantly cheaper as it does not require any additional investments to produce new copies of the application. As a consequence of these two factors, game developers can increase their profits by, among other methods, increasing the size of their target audience to this market with less risk.

It is also important to note that even in cases when digital solutions games draw away demand from their physical counterparts, the interest in the game itself does not decrease.

[4] British Museum. *The Royal Game of Ur*. Available at: https://www.britishmuseum.org/research/collection_online/collection_object_details.aspx?objectId=8817&partId=1

[5] Encyclopedia Britannica. (2017). *Chess - History*. Available at: https://www.britannica.com/topic/chess/History

[6] Web.archive.org. (2010). *Sight Unseen-The Art Of Blindfold Chess*. Available at: https://web.archive.org/web/20120329111856/http://www.jerrywalldesigns.com/WhiteKnightSep10.pdf

Because of that, if a company that owns rights over a specific physical game were to develop a digital solution, they would still get the profits. However, with two largely different sources of income, the target audience and the profits would increase.

Therefore, globalization, which allows easier expansion to markets around the world, and the development of digital technologies have created a digital market for games as well. For example, the amount of shipped smartphone devices has already passed 1 billion mark[7]. Digital market is not dependent on physical equipment. Digital market also eliminates the risk of product copies getting damaged or not sold, becoming an expense. Instead, digital solutions make it possible to distribute copies of a product only when one is requested. While there is still a risk that the initial development will not be covered by sales, lack of necessity to pre-order materials and physical equipment still minimises the potential expenses.

Physical tabletop games receiving digital adaptations is not a unique idea. Many large and successful titles such as *Monopoly*[8], *Magic: The Gathering*[9], *Tsuro*[10] or *The aMAZEing Labyrinth*[11] have official digital versions developed for them. While there are many more tabletop games that could be selected as examples, these four fall into the similar category as *Cogbug*. All of these games can be classified as strategy turn-based games. In all of these games actions of one player may directly affect other players in both positive and negative ways. For example, buying a tile in *Monopoly* may result in other players stepping on it and paying the owner or placing a new tile on the game grid may disrupt or even eliminate other players in *Tsuro*. *Cogbug* is not as drastic towards players as *Tsuro* is - a player is capable of disrupting the strategy of other players, but can neither reset nor eliminate others from the game.

Magic: The Gathering is a card-based game and is, therefore, completely different from *Cogbug* in many aspects. However, it does have one similarity -players have access to active cards that can either be used to advance the player to victory (using these cards to attack the opponent) or to defend themselves from their opponent's active cards. In order to be able to

[7] Web.archive.org. (2014). *Worldwide Smartphone Shipments Top One Billion Units for the First Time, According to IDC - prUS24645514*. [online] Available at: https://web.archive.org/web/20140131071943/http://www.idc.com/getdoc.jsp?containerId=prUS24645514 [Accessed 8 Nov. 2019].
[8] https://monopoly.hasbro.com/
[9] https://magic.wizards.com/
[10] https://boardgamegeek.com/boardgame/16992/tsuro
[11] https://www.ravensburger.org/uk/products/games/family-games/labyrinth-26448/index.html

play such cards, players must play a special type of card called “Land” cards. This approach is also similar in *Cogbug* - while the goal is to advance to a specific score first by transferring buttons across the field (game rules are explained more in chapter 3.2), players must first lay a groundwork of cogs that would enable for buttons to move across the board.

2.1 **Mobile Games**

Mobile games have first appeared in 1994, when the phone Hagenuk MT-2000 first came out and had *Tetris* preinstalled[12]. Since then, the market share of games has been steadily increasing. According to a study performed by SensorTower, out of the 10 most earning mobile games of 2019 (including *PUBG Mobile*[13] by Tencent Games), 8 games (excluding *Candy Crush Saga*[14] by King and *Dragon Ball Z Dokkan Battle*[15] by Bandai Namco) include cooperative or player versus player (PvP) elements [19].

The reason behind such profit distribution is that online games tend to provide a better and more sustained level of user experience to the players, which in turn results in increased customer loyalty and, followingly, better revenues per customer. A study performed by Choi [10] has shown that online games are designed to trigger feelings of competition and achievement, which are formed via the personal interaction (interaction between the user and the system), alongside with social interaction (the user-to-user interaction). While singleplayer games also trigger such feelings with personal interactions, they cannot provide a sufficiently high level of social interactions. However, online games can do that as they are designed with game modes that rank the player against other players. On the other hand, mobile applications require the formalization of interaction between players, as informal interaction that is possible between players when playing a physical game cannot be achieved effectively. Additionally, smartphones are easily accessible, which makes it easier to engage with a game at any place or time. Schell [11, pp 102-106] has a similar opinion - competition and mastery are one of the main aspects that attract male players. Since female players have different interests, such as getting emotions or learning by example, these aspects are also included in modern mobile games by providing tutorials that explain what each button and

[12] T., Nick. "This Was The World's First Cell Phone With A Game Loaded On It". Phone Arena, 2014, https://www.phonearena.com/news/This-was-the-worlds-first-cell-phone-with-a-game-loaded-on-it_id62920
[13] "PUBG Mobile." https://www.pubgmobile.com/
[14] "Candy Crush Saga on the App Store." https://apps.apple.com/gb/app/candy-crush-saga/id553834731
[15] "DRAGON BALL Z DOKKAN BATTLE – Apps on Google Play." https://play.google.com/store/apps/details?id=com.bandainamcogames.dbzdokkanww&hl=en_GB

action does and adding a storyline setting to the game. The latter does not imply that only female players benefit from tutorials, but states out that they gain greater benefit from such. Since the demand for games is large, more competitors are attracted to the scene as a potential customer base. Therefore, it is logical that other new companies would be interested in introducing another game to the market as it would increase their customers and, ultimately, profits.

A study performed by Viennot et al. (2014) shows that up to 25% of applications displayed on the Google Play digital distribution platform are duplicative. However, a larger segment of mobile games is original in content and ideas which means that original content generates more value and tends to be more demanded among the consumers. Among those products there are also games that were originally developed for other platforms like *PUBG* (PUBG Corporation, 2017), *Minecraft* (Mojang, Microsoft, 2011), *Fortnite* (Epic Games, 2017) or *Hearthstone* (Blizzard Entertainment, 2014).

Porting an existing game to the mobile platform makes it possible to increase the sales by introducing the game to a new market. This action is also not rare for tabletop games - Uno (Mattel, 1992) already has an official mobile version and Monopoly (Hasbro, 1935) is scheduled to get an official one next year as well as many other games[16] [17] [18]. This increases the market share for games that require more than one player. For that reason, Edulab OÜ (Customer) has also decided to expand their target audience with a mobile app of their tabletop game *Cogbug*.

In order to organize the development process, it was negotiated and agreed with the Customer to split the development process into three parts:

1. Background work – settle legal arrangements, develop task requirements.
2. Development process – implementation of the MVP of *Cogbug*. MVP was required to contain an online multiplayer option. During this part, multiple internal testing sessions were held to evaluate the implementation of each developed module.

[16] Sonechkina, A. (2018). *The 10 Best Digital Board Games For Your Mobile - GameAnalytics*. GameAnalytics. Available at: https://gameanalytics.com/blog/10-best-digital-board-games.html
[17] "MONOPOLY is heading to iOS and Android later this year ...." 2 Oct. 2019, https://www.pocketgamer.com/articles/081227/monopoly-is-heading-to-ios-and-android-later-this-year/
[18] "The 15 Best Board Game Apps - Popular Mechanics." 19 Jan. 2018, https://www.popularmechanics.com/culture/gaming/g2210/the-10-best-board-games-apps/

3. Testing – after the MVP would be considered done, a closed testing session was scheduled. This part was designed to attract people that were not involved with the development process to ensure more objective results.

3. Goal Overview

This section describes what kind of a game *Cogbug* is, what the goal of the mobile version is. The section is concluded with the requirements that were discussed and agreed with the Customer representative.

3.1 Cogbug Tabletop Game

Considering the tendency of various tabletop games getting mobile adaptations developed, the Customer, who developed the physical version of *Cogbug*[19], is also interested in getting a mobile version of this game. The physical version of the game was officially released in November 2019.

Image 1 shows the tabletop game. In Image 3.1 a random situation from the game is simulated: multiple cogs have been placed on the board, several buttons are also deployed on cogs. Finally, the picture shows a player rotating a cog, which in turn, rotates other cogs connected to it.

Image 1. Cogbug physical tabletop version

3.2 Game Rules

The rules for the game can be found in Appendix II. This game is designed in such a way that it can be played by two to four players. An average game session lasts from 30 to 90 minutes. The game can be classified as a strategy game as it requires the players to plan out the strategy several turns ahead in order to gain advantage over other players.

It is important to note that the rules of the tabletop version of *Cogbug* do not cover all the possible cases that can happen during the game. The official website's Frequently Asked

[19] https://gearboardgame.com/cogbug/

Questions section[20] states that players are free to resolve issues that arise in any way they see fit as it would mimic the real life approaches, ie negotiating the possible solution between players.

3.2.1 **Assets**

As the game is intended to be played by two to four people, it is necessary to differentiate these players apart. Because of that, the Customer has chosen to apply one of the simplest ways to do so - the players are differentiated by color. In the game, all player-related assets come in four colors - red, green, yellow and blue. Appendix III demonstrates a full set of game assets of a physical game.

The physical game contains a set of assets that are mandatory for the game to be playable. These assets are:

- **Game board (board)** - in the physical version of the game, contains a 7x7 grid of *pins*, to which cogs are attached.
- **Bases** - bases are player identifiers. Every player has to pick a base at the beginning of the round. The base color dictates the color the player will be working with. Each physical game copy contains 4 bases, one of each color. Bases are designed in such a way that they can be easily attached to the edges of the board to represent the amount of players in the game.
- **Buttons** (further *victory buttons*, as the original name would conflict with the user interface elements, buttons) - similar to bases, are player assets. Each player has 3 victory buttons of their color at the start of the session. These victory buttons need to be transferred across the board in order to progress towards victory.
- **Regular cogs** - core element of the game. Cogs are pieces that can be placed on pins on the game grid. Each cog has 4 arrows that indicate a direction. Each of these arrows has a specific color to indicate which victory buttons of such color can enter or leave the specific cog from this direction. If 2 cogs are placed on nearby pins not in a diagonal way and they have arrows of the same color pointing at each other, a Button can be transferred by the player of this color. There are 28 regular cogs in total. Arrow colors combinations are predefined and fixed (see Appendix II), their combination set

[20] https://gearboardgame.com/cogbug/faq/

was developed by the Customer and described by rules of the physical desktop version.

- **Blocker cogs (blockers)** - special type of cogs, these can be attached to either a pin on the grid or to the other cog. These cogs can be rotated, but do not affect any connected cogs, as they do not form a chain. Players also cannot transfer buttons across the blocker or across cogs to which blockers are attached to. The physical game comes with 5 blockers.
- **Joker cogs (jokers)** - mechanics-wise are identical to regular cogs. The only difference is that jokers can accept and transfer any color in any direction. Therefore, jokers can be used by every player. Physical game offers 2 jokers.

All cogs (jokers, blockers, regular ones) are designed to have a hole in the middle. That way, they can be attached to the pins on the game board.

The size of each cog is such that its radius is equal to half the distance of between two neighboring pins (excluding diagonal neighbors). Each cog has multiple teeth, which are used to affect cogs that are attached to neighboring pins. That way, when a cog is rotated, it would rotate all its neighbors.

The number of game assets is not large. However, some of them could become a subject to adjustments or changes in numbers for the digital version. In order to account that, the implementation was scheduled to utilize techniques that would allow for scalability adjustments during the development process.

3.2.2 **Actions**

The game is turn-based. Each player can perform a specific set of actions during their turn. These actions are:

- **Drawing** a new cog - players can use this action to draw a new cog and place it on the grid. Players cannot choose which cog they will get. Instead it is a blind draw from a bag. Before the cog is placed onto an available pin on the grid, the player can choose it's orientation first (rotate it without using their rotate action). The latter does not use up the rotation action. This action is mutually exclusive with cog movement - only one of these can be made each turn.

- **Rotating** - the player can rotate the cog if it is placed on the grid. This will rotate all the cogs that are nearby and form a chain. The tabletop version allows to rotate any one cog (or cog chain) by any amount of degrees, but only once per turn.
- **Moving** a cog - the player can move the cog to an available pin. While moving, the player is free to rotate the cog as they wish, this does not use the rotation action. It is not allowed to move a cog if any victory pin or blocker is placed on it. It is also allowed to pick up a cog, rotating it in the "air", then placing it back on the same slot. This action only uses up the move action. Since the cog is lifted from the board and does not physically affect any other cog, the player can rotate only that cog, without the risk to affect the chain of cogs. If this action is performed, players can no longer draw a new cog during the same turn.
- **Move victory buttons** - the goal of the game is to deliver a fixed amount of victory buttons across the board. At the start of each game players agree on the amount of buttons required to achieve victory. By default, it is three. All player's victory buttons start at the player's starting position. Players can move the victory buttons only via the cogs if the two cogs have a connection of the same color as their own.

Each action is limited to no more than one per turn. This ensures that each turn does not last too long as well as brings some fairness into the game. However, it still allows a certain degree of freedom as the players are not obliged to use any actions at all or perform them in any particular order. For example, a player can just draw a new cog, place it on the grid then end their turn without using other actions permitted by rules. Another possible combination would be to move and rotate the cog(s), then move a victory button along the newly created trail.

Additionally, it is important to note that victory buttons, once transferred to the end of the board, are removed from the game field and are no longer interactable with. This is designed as a form of checkpoints. If a player has transferred two out of three required victory buttons across the board, they are guaranteed to have their score at 2. Neither the player nor their opponents can affect the buttons that have already been redeemed.

However, removing the victory buttons off the board has its drawbacks. While this does ensure that one's score is increased, a victory button placed on a cog also ensures that other

players cannot relocate the cog. Because of that, leaving victory buttons on critical positions will ensure that the opponents cannot disrupt the strategy laid out by the player.

3.3 **Mobile Version**

The digital version of the game is intended to resemble the physical game in both looks and functionality. Therefore, some discussions were conducted with the Customer representative to establish the details and main aspects of the product. Additionally, one on-spot observation of the game cycle was conducted. Image 2 below shows a visual comparison of the physical and digital versions of *Cogbug*.

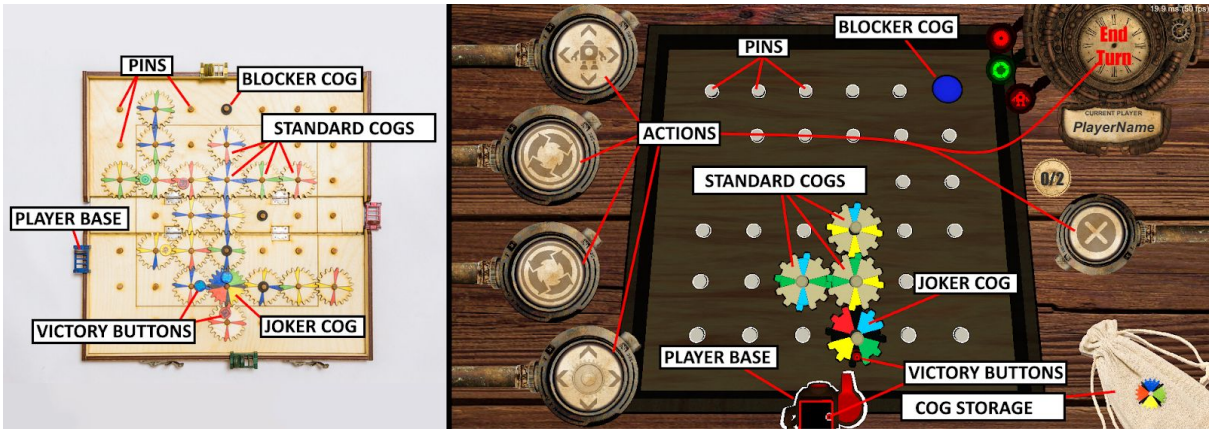


Image 2. *Cogbug* physical (left) and digital (right) game view comparison

3.3.1 **Requirements**

After discussing the requirements with the client, the following non-functional requirements were clarified:

- The game should support up to four players.
- The game should be playable on devices that run on Android 4.4+ and iOS 10+ systems.
- The average game session should be faster than the physical one. The physical game session can last up to 90 minutes, which is too long for a mobile version. Initial goal was agreed to have the game session last no longer than an hour (60 minutes) .

In discussion with Customer it was agreed that the focus of the mobile game development would be to implement the core mechanics of the tabletop game (see chapter 2.1). There were also additional mechanics discussed, which are documented in Appendix B / II. This was

done in order to be able to deliver the minimum viable product without developing a feature creep tendency[21].

The requirement regarding Android and iOS versions is based on the minimum requirements required to run a Unity project. These requirements were taken from the official Unity website[22].

For the game, the following game modes were planned to be implemented:

- **Singleplayer mode** - the game should have an option where only one player is required and others are controlled by the computer.
- **Pass and Play mode** - the game should be playable on a single device where human players take turns.
- **Online Multiplayer mode** - the game should be playable with other human players on different devices over the Internet.
- **Tutorial** - a game mode where the player would be taught how to play the game. The player would be explained the game rules, restrictions and logic. This game mode is linear and does not assume complex thinking.

However, after discussing the feasibility and impact of these game modes, it was agreed to implement online multiplayer mode first. Singleplayer mode, while being an interesting concept, would take a lot of time planning as well as significant investment regarding the development of propper opponents.

At present, the game does not have a defined linear victory strategy. This means that computer-controlled opponents would require some on-spot situation evaluation in order to make the most logical move. This would require a complex system that would most likely rely on machine learning algorithms. The latter requires a lot of fine-tuning in order to determine the best action (or set of actions). While it is completely possible to implement a machine learning algorithm, it would be better to get the data for it first. Such data can be obtained from observing multiplayer sessions.

[21] "What is feature creep? - Definition from WhatIs.com - SearchCIO." https://searchcio.techtarget.com/definition/feature-creep
[22] https://docs.unity3d.com/Manual/system-requirements.html

Alternatively, it is possible to develop an artificial intelligence that would not require machine learning. For example, Minimax[23] algorithm can be used to dynamically calculate the most optional action for the artificial opponent. Minimax works on the idea that each state that can be reached has a particular value. One player's goal is to reach the highest value state whereas the other's is opposite - to reach the lowest value state.

However, this approach is developed only for games where there are only two players. In case of *Cogbug*, up to 4 players can participate at the same time. It can be considered that the "opponent" player is a combination of up to three players that play against the one using the minimax method.

The main issue with Minimax is that this algorithm takes more and more processing time with the increase of the complexity of the game. This happens to be the case of *Cogbug*. *Cogbug* is a strategy game where momentary loss can be necessary to achieve a long-term gain. Additionally, During each turn, a player can make up to 3 decisions and each opponent can do the same. Because of that, it would be required for artificial intelligence to calculate the game several turns ahead, with end states changing every turn due to other players' actions. Each player action creates a separate branch that would need to be calculated on the run. Because of that, such an algorithm is unsuitable for *Cogbug* - each artificial opponent would require a considerable amount of time to make a turn that would have some logic behind it. As the game is designed for mobile platforms that can have less processing power available than stationary devices, multiple artificial intelligence instances can negatively impact the performance. Other more complex algorithms were not studied, as their implementation was considered too difficult to implement for the MVP.

Pass-and-Play mode shares similarity with the online multiplayer mode, but would be less comfortable when used on smartphones compared to tablets. Therefore, it was decided to implement a multiplayer solution first, test it, refine and then focus on other game modes, if that would still remain a priority goal.

The tutorial, while being extremely important for teaching the basics to the player, was also delayed. It was agreed that the player would need to receive some general instructions before playing the game itself. For earlier product versions, the tutorial would be replaced with a

[23] https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/

text-based ruleset. With these factors, it was decided to put the tutorial as the second highest priority game mode. While it was not planned for the MVP version of the application, it was set to be implemented first afterwards.

Image 3 and Image 4 depict the difference in the concept art and final design for the MVP. The singleplayer button, which would start the game mode, is disabled. The tutorial button, while being active, notifies players that this mode is not implemented. This was done in order to state that tutorial is a higher priority task than single player mode. Additionally, this prioritisation coincides with customer's market strategy priorities.

Image 3. Early stage main menu concept art

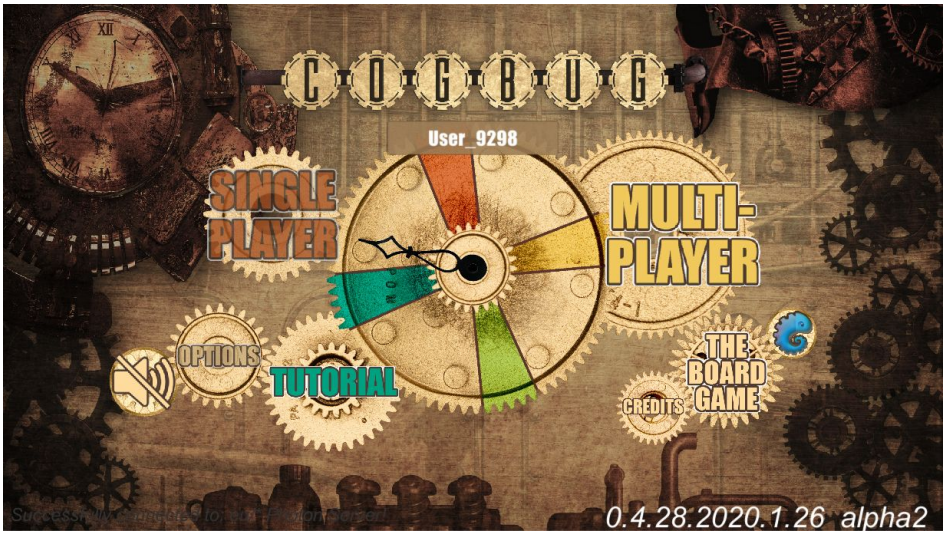


Image 4. Main menu MVP design

3.3.2 **Technologies Used**

The game was developed using the *Unity*[24] game engine. Initially, based on the requirements, it was decided that the most efficient way to develop the game would be to use a game engine. In order to achieve this goal and because of personal experience, *Unity* and *Unreal Engine 4* were considered. The main difference between the two tools is that *Unity* offers more built-in elements that can assist with the development of Android software [1, 21]. Unreal Engine offers higher quality graphical solutions such as lighting and other complex post-processing effects. However, this is not a primary requirement for *Cogbug* [2].

While both *Unity* and *Unreal Engine* allow the development for mobile platforms (both iOS and Android), *Unity* is usually the preferred choice. The reason behind that lies, as stated earlier, in the fact that *Unreal Engine* is designed for better graphics quality and, followingly, better hardware, which is the main limit of a smartphone. The benchmarking results on mobile platforms show that *Unity* approach yields a significantly higher frametime than the *Unreal Engine* one, which means that *Unreal Engine* performs faster on a mobile device than *Unity* [16, pp 41-48]. The latter should be considered as a strong side of *Unreal Engine*. Graphics-wise, *Unreal Engine* provides significantly more built-in features and more complex light calculation means [16, pp55-56].

However, such details are usually omitted or not implemented at all on mobile platforms as, while smartphones offer the same screen resolution as desktop devices, the actual screen size is considerably smaller. Because of that, small details or complex reflection and refraction on an object are not things that would benefit the product enough to be considered. Finally, personal experience shows that *Unreal Engine* projects tend to take more storage space and, therefore, more traffic when downloading and updating on a mobile device. The latter can become a problem, as smartphones have limited storage space and, since smartphones rely on mobile network connection, can take too much time or become too expensive when downloading.

[24] https://unity.com/

GameMaker Studio could be used, but it is limited to 2D game development only[25]. Because of this, it was discarded as Customer representative was initially interested in either 3D or 2.5D perspective for their mobile game.

For the graphical part, *Paint.NET* was chosen as a tool for editing. Its alternatives were *GIMP* and *Krita*. Additionally, *Adobe Photoshop* could also be considered an alternative, but since there was no need for too complex textures or graphics, the basic functionality provided by free tools was considered sufficient[26]. *Krita*, even though being a viable alternative, was discarded due to the lack of personal experience with the tool. Therefore *Paint.NET* and *GIMP* were chosen as alternatives for a more detailed review. *Paint.NET* is considered to be easier to learn and use, which was an important factor to account as previous experience was limited to basic knowledge and this thesis could require learning advanced techniques[27]. Both tools have plugin support, so some of the missing functionality can be replaced by community-made solutions. Paint.NET has less built-in functions than GIMP, but is faster and lighter. GIMP, being initially developed and released for Linux systems, has an interface that is more native and understandable towards users of that platform. For Windows and iOS users, GIMP's interface is more foreign and, therefore, requires more time to get accustomed to.

3.4 **Interaction with the Customer**

For this project, the agile product development methodology was chosen. Such an approach makes it possible to embrace the overtime changes in tasks and their priorities. According to the methodology, a series of meetings were conducted with the Customer. Most of the meetings and discussions were carried out in a web-based form via email and instant messaging tools (*Slack, Facebook*).

Firstly, it was required to establish the groundwork for the project, such as understanding what the product is and what is the expected result. Therefore, the following goals were scheduled for the first meetings:

[25] Yoyo Games. (2019). GameMaker | YoYo Games. Available at: https://www.yoyogames.com/gamemaker
[26] Wycislik-Wilson, M. (2019). GIMP vs Paint.NET: which is the best image editor for you?. TechRadar. Available at: https://www.techradar.com/news/gimp-vs-paintnet-which-is-the-best-free-photo-editor-for-you
[27] Slant. (2019). Slant - GIMP vs Paint.NET detailed comparison as of 2019. [online] Available at: https://www.slant.co/versus/3000/5472/~gimp_vs_paint-net [Accessed 9 Nov. 2019].

1. Understand what the client wants to receive as a product.
2. Understand how the physical (and already implemented) version of the game operates, how it is intended to be played.
3. Agreeing on the visual concept of the game.
4. Discussing the results of usability testing.
5. Scheduling the development milestones.

Goals 1 and 3 were defined so that the development process would stay strict and avoid inconsistencies in goals and expectations between the Customer and the work done during this thesis.

Goal 2, on the other hand, was established in order to get an understanding of the core elements of the physical game. As the Customer requested the game sessions of the digital version to be faster than those of a physical game, a deeper understanding of how this could be achieved was required (see Appendix II).

The first meeting that took place in October 2019 had the goals of establishing the groundwork for the collaboration. The first goal of this meeting was to agree on the future communications means. Initially, email-based communication was chosen with instant messaging services being added later. Secondly, the rules for the physical tabletop game version were analysed and discussed, in order to find out which aspects of the physical version can be optimised or sped up in the digital solution. Finally, legal questions, including the non-disclosure agreement, were discussed and settled in order to avoid issues related to exposing the information about the development process.

Meeting 2 (see Appendix IV) was intended to demonstrate a concept player perspective of the game as well as to discuss the future work with the Customer representative. Additionally, the created 3D model designs for cogs, the game board and player pawns were demonstrated. It was agreed that the player view should be adjusted, the next meeting was scheduled as well as the goals for it were settled. No adjustments for the 3D models were necessary.

In January 2020 the Customer representative expressed interest in expanding the development team by introducing another developer. Initially it was intended that the new team member would assist in the development process in terms of programming due to

having more experience. It was agreed that this would be discussed in more detail during later interactions, as it would allow all members to discuss the priorities, tasks and, based on that, make more weighted and logically structural decisions. With that, the team size increased to three, having two developers (Stanislav Belogrivov, Madis Kaspar Nigol), and one designer (Tarvo Metspalu, Customer representative). The latter was responsible for developing the concept art for the application as well as designing the user interface and model redesign at later stages of the development. The final goal of the meeting was to discuss the early version of a playable demo that was provided for the meeting in order to establish the direction of the future work. Image 5 depicts the user perspective of a build that was released slightly after the meeting. The main difference in the interface is that the meeting version did not have the white outline for indicating the player.

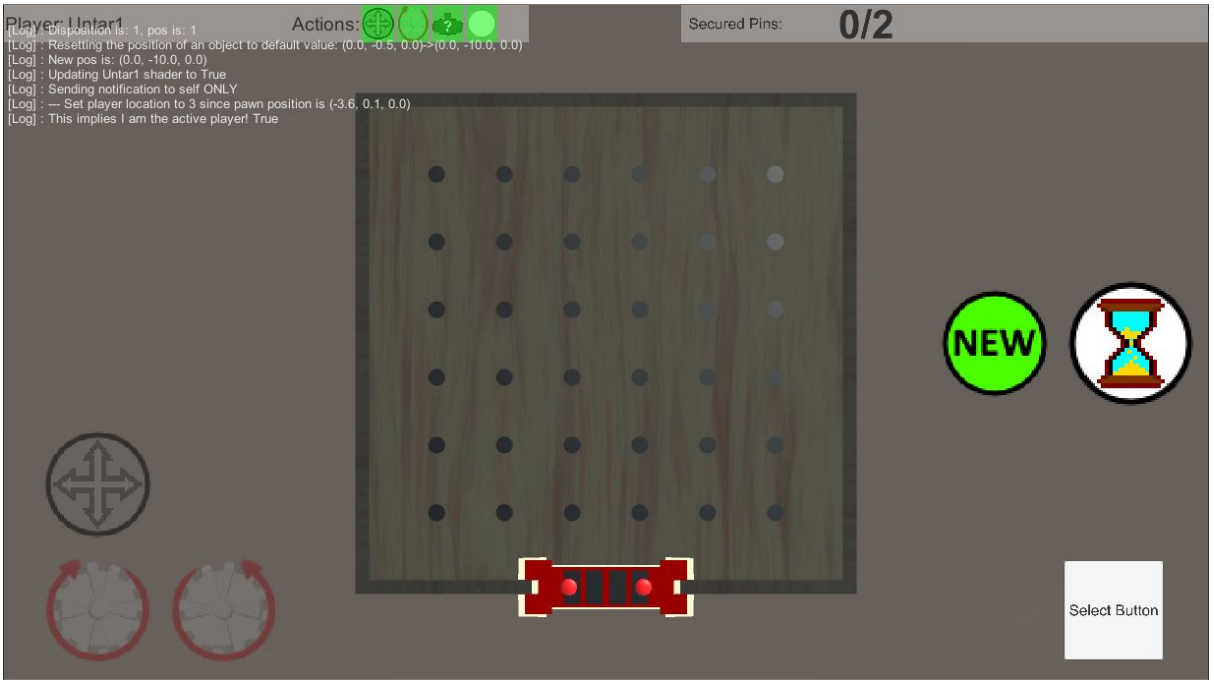


Image 5. Single player perspective, early build

With the increase of team size, it was also agreed to synchronize the project state using GitHub[28] software development platform. While there exist other alternatives (eg GitLab[29], BitBucket[30]), all team members were familiar with this solution. Therefore, GitHub was chosen.

[28] https://github.com/
[29] https://about.gitlab.com/
[30] https://bitbucket.org/product

During the meeting it was settled that the highest priority between the playable game modes would be the online multiplayer option. This was set as a milestone for the MVP. Additionally, the demo was discussed and a bug was found in the functionality that was set to be fixed as a high priority. Finally, several gameplay details were discussed and settled.

This meeting allowed splitting the future work between the members of the team. Customer representative, based on the demo, agreed to develop additional visual designs for the product, putting emphasis on the design of the graphical user interface (GUI). The additional developer's initial tasks were set to reviewing the existing code, working on sound and visual effects for the project and, for the later stages of development, assist in the multiplayer development. During the development process these tasks were shifted to only implementing sound effects for the application.

Additionally, the customer representative volunteered to join the development team as a digital artist. This is a reasonable approach, as, being a customer representative, they can provide both the concept art for the final product as well as implement actual models to be used in the application.

Image 6 depicts one of such concept arts. This image shows the perspective of a player with the nickname "ONETWO3". The UI contains the following elements:

- Top right corner contains the player name indicator, end turn button as well as 3 main action indicators (moving a cog, rotating a cog, moving a victory button). In this image the drawing a cog and moving a cog would have the same indicator. The reasoning behind that is that the player can perform only one of these actions during their turn.
- Center right side contains a single "cancel" button.
- Bottom right corner contains an image of a linen bag, which is intended to act as a button, by pressing which the players would perform the cog drawing action.
- Left side contains the buttons which allow the user to perform actions. The middle button has an overlaying check symbol. This was intended to demonstrate that the player would have to tap the button again to finalize the action. It was proposed to implement the same functionality for every other button as well as introduce animations regarding that.

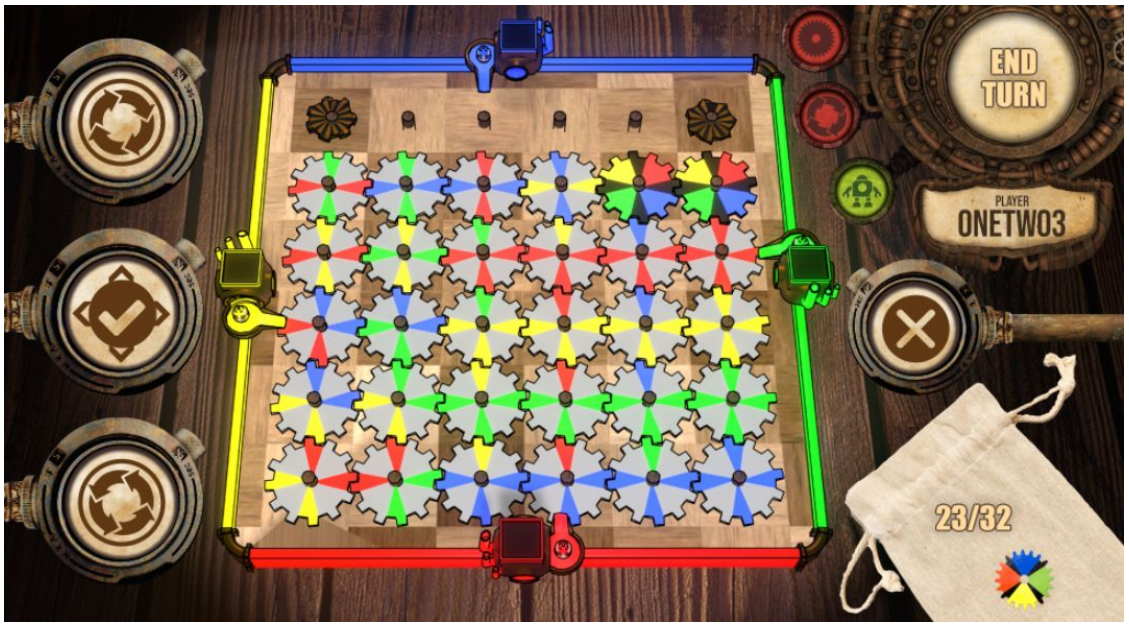


Image 6. Concept art of user perspective, provided by the customer

This image has highly saturated colors and shaders added to all the in-game objects. This looks more visually appealing than the MVP that was developed (Image 6). The reason behind that is that the highly saturated and more complex in-game objects take up considerably more processing power to render and cause a severe performance drop. When testing the application on a 2016 Samsung Galaxy Tab A tablet, the recorded performance drop was around 60% on average. As seen on Image 7, it was possible to rework most of the objects and implement the suggested design with the game board and blockers not being implemented changed.

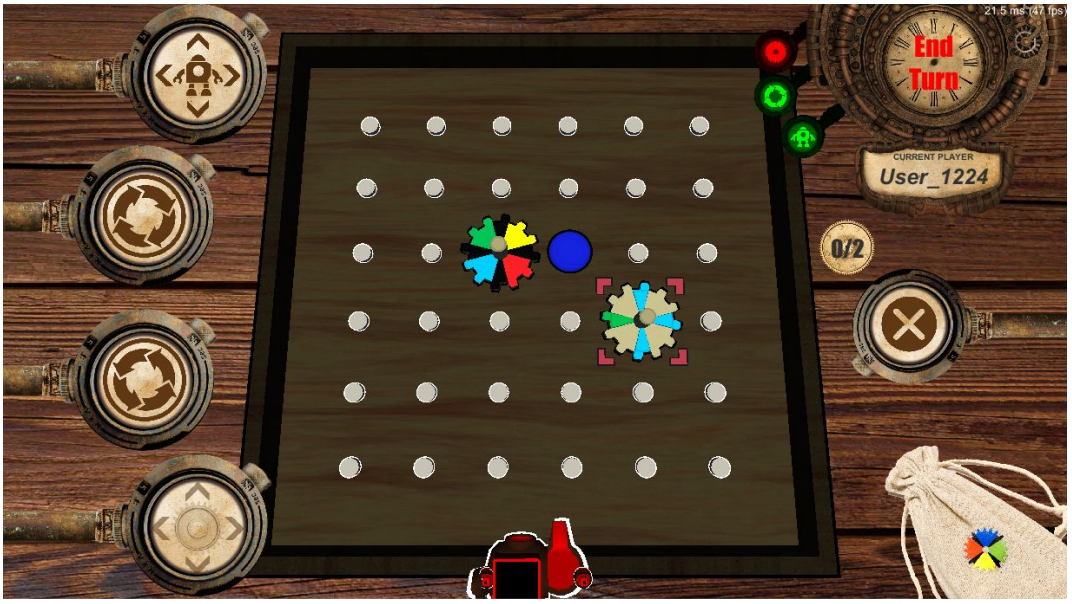


Image 7. Actual design from the MVP

3.5 **Gameplay Differences**

As mentioned before, the digital application is intended to have a set of differences from the physical game in order to speed up the game process. As a result of discussing with the Customer, the application was agreed to have the following specifics:

1. Game field size – in the physical version the game grid contains 49 (7x7) pins on which cogs and blockers can be placed. It was decided to reduce this amount to 36 (6x6) pins on the grid. Reducing the amount of the pins speeds up the game as the distance each player has to traverse reduces. As a result, the average amount of turns per game would reduce.
2. Undo action – in the physical game it is possible to undo any amounts of your actions if it is your turn and other players do not mind that. As a result, it is possible to prolong the game by making a lot of actions then undoing them. Digital solution introduces the undo action as a legitimate and separate action, since discussions with other players (the analogue of "other players do not mind" in the physical game) are difficult to implement effectively over the network. The undo action can only revert unfinished actions only (rotate cog, move cog, move victory button). Player is unable to undo the drawing action and cannot undo the actions that were already finalized.
3. Actions – the actions themselves are the same as in the game. However, it was decided to automate the actions that these actions would do. For example, moving the cog around was agreed to follow the point and click mechanic.
4. Cog rotation – the physical game suggests that cogs (and their chains) can be rotated by any degree. Digital solution allows the rotation only by 90 degrees per use, allowing multiple consecutive uses, however (eg the user can rotate the cog clockwise 2 times in a row per turn). Firstly, as the rotation is automated, the time for each rotation will remain fixed regardless of the amount of cogs in a chain as opposed by the physical version where friction applies. Secondly, such limitations will ensure the aesthetics and ease of understanding of the situation: all the cogs are always aligned in the same manner.
5. Victory Buttons – the amount of game buttons was agreed to be reduced from 3 to 2. These objects need to be traversed over the map to achieve the only victory condition.

Therefore, reducing their numbers will significantly reduce the amount of turn and decrease the late game.

6. Visuals – the digital solution allows more freedom in the graphical aspect as the 3D objects do not have to be affected by real-life physics and manufacturing costs do not apply to them more than once. However, as it was mentioned earlier, visual effects are limited only by the performance of mobile devices. Therefore, it was agreed that the visual design of digital game objects would differ from their physical counterparts.
7. Bases – the physical game offers each player to pick a color of their own, with each subsequent player having to choose from one less option. With the digital solution, it was decided to hardcode the color palette and color distribution. The decision is dictated by the fact that if in the offline or desktop version of the game communication between the players is possible, not restricted and can be non-formal. On the other hand, in the online version players' communication possibilities are usually limited and formalized for such types of games. Therefore it is difficult to effectively formalize such discussion of the distribution of colors. As a result, the free choice of colors and discussion of the distribution of colors would be complex to use and greatly delay the start of the game. Additionally, this would reduce the traffic and resolve the issue with having to synchronize specific custom colors between players or ensuring that two players do not pick similar colors. As a result, the first player would always be assigned red color, second green and so on.

4. The Implementation

The process of the implementation was split into three parts:

1. Implement all client-side features and mechanics as well as ensure that they work.
2. Modify the first part into a standalone player module.
3. Develop a networking solution, integrate the player module into it.

Such order was considered the most optimal, as it would allow to test the game mechanics and their behaviour first. Additionally, it was stipulated that after the development of the client side mechanics, further actions, such as network part development and the visual refinements, could occur in parallel (as it described above, appearance refinements were partially undertaken by the customer). Other modules could affect the application's behaviour (eg networking not working in a stable manner). However, if only the player-side elements are developed and tested, the development was planned to run in a more smooth manner.

The first stage was finished in January 2020 with the internal meeting. The player-side solution was presented, tested and was accepted by the client. Afterwards, this solution was put into a separate module and the development of the multiplayer module was started. Subchapter 4.9 describes the multiplayer logic in more detail.

As stated in the previous chapter, the development was following the agile practices. Because of that, after each interior milestone was achieved (eg victory buttons implementation), they would be tested and, if necessary, fixed or reworked. Such an approach was considered optimal by the team as it would allow the team and the Customer to observe each mechanic. If the client saw that the implementation would differ from what they originally intended, that feature could be discussed over.

The implementation phase of the development process was finished in April 2020 and was followed by the user testing phase. Chapter 5 covers the testing aspect in greater detail.

4.1 Cogs

As described in subchapter 3.2.1, the game rules define 3 main types of cogs. Because of that, their behaviour can be described as follows:

1. Regular cog. Can be moved and rotated. Can transfer and hold victory buttons of the colors that it has. For example, a red button can be transferred to and from the cog only if the cog has a red point in the respective direction. Regular cog can contain up to two different colors.
2. Joker cog. Can be moved and rotated. Can transfer and hold victory buttons of any color in any direction.
3. Blocker cog. Can be moved, cannot be rotated. Can neither transfer nor hold victory buttons. Is not affected by other cogs if those are rotated. The blocker cog does not prevent the chain of cogs from rotating, and does not transfer the "rotation" from the chain to other chains it can be connected to. Essentially can be considered an object that takes space, but has no impact on the surroundings.

Because the logic of the cog behaviour is similar, it was decided to implement them using the *type object behavioral pattern* [6]. Each cog contains an implementation of a class that defines the general logic for each cog. A cog contains information about its type, colors on each side and any attached victory buttons. Cogs do not contain any movement or processing logic. Instead, cogs provide information about themselves, their position or their stats.

Additionally, there is also a unique trait of a blocker cog: a blocker cog can be attached to any other cog, if they are not occupied by victory buttons. In that means, this makes a blocker cog similar to a victory button (can be attached to cogs). Because of this feature, the blocker cog has the movement properties of a regular cog (move to empty spaces) and the attachment properties of both cog and a victory button (can be attached to pins, can be attached to other cogs if there are no buttons attached). Class diagram 1 below illustrates the inheritance of all cog types from the main `Cog` superclass.

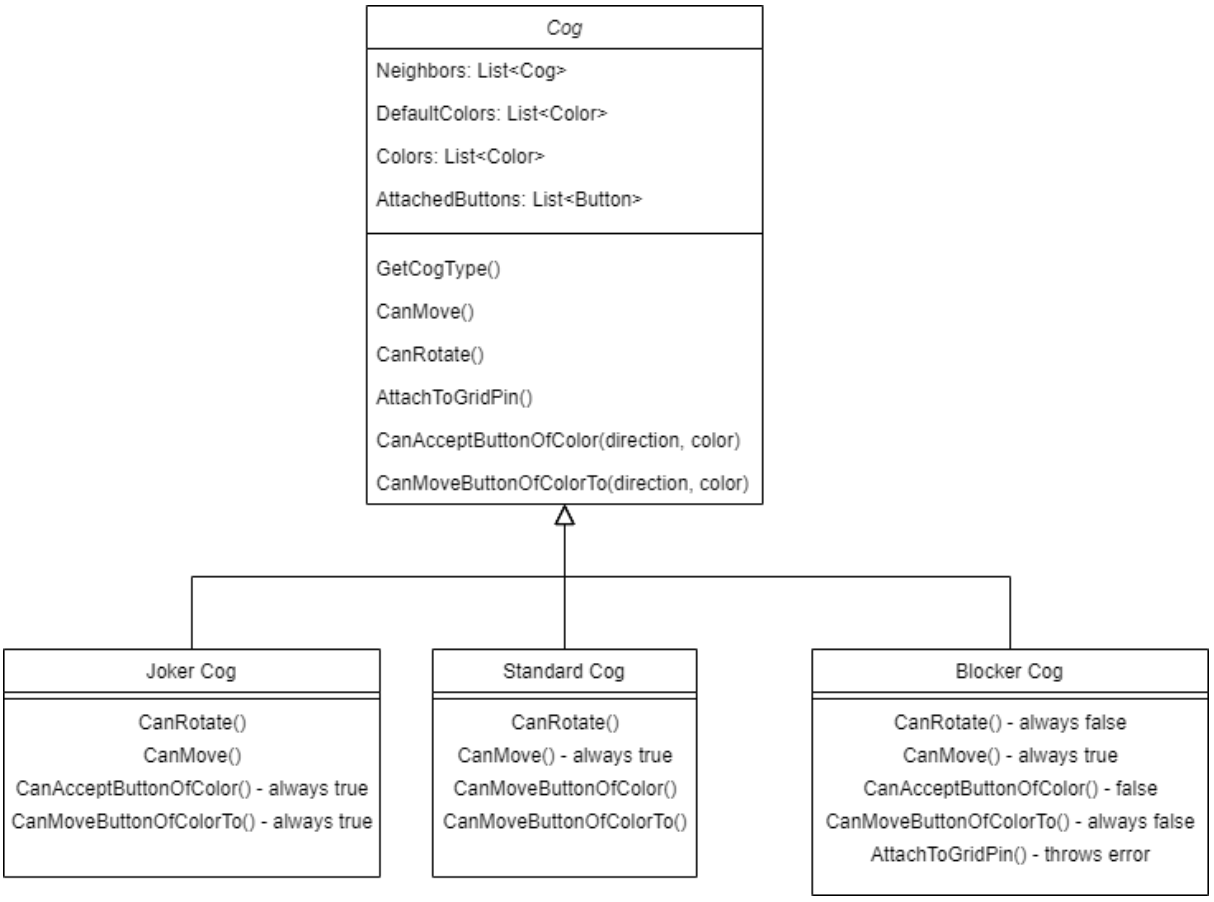


Diagram 1. Class diagram of the cog classes using type object pattern

4.2 **Victory Buttons**

Victory buttons are used as objectives. Each player gets the same amount of victory buttons at the start of the match. Each victory button is colored according to the color of the owning player. When the player delivers a victory button to the end of the board (respective of their initial position), the system gives that player a point towards the objective.

Destroying an object in Unity may lead to various issues, such as references to the non-existing object. Such references may also occur with the game engine itself. To avoid that as well as to accountability and traceability, victory buttons are never destroyed. When a button reads the end of the board, it gets disabled and is turned invisible and cannot be interacted with. This means that the button still exists in the game and, therefore, can be referenced in the code. The end player, however, does not see the button as it is not rendered. Since the amount of game buttons is not large (current rules assume up to 3 buttons per person, so up to 12 objects in total) and they are spawned from the start, the long-term impact on the performance can be neglected.

The approach described above does not tie the buttons to other game elements. This makes the solution more modular and allows it to be modified separately without requiring significant changes to other elements.

4.3 **Rotating Gears Effect**

When the game starts, an *M*x*N* grid is generated. Each grid element was designed to have as little logic as possible, since these elements are static and only hold 2 important values: their position on the game grid (x and y coordinates) and an attached object. The attached object can either be nothing, a cog or a blocker, which creates 3 possible states for future use. By themselves, grid elements do not perform any actions.

A cog, after being generated and placed on the grid, holds information about its four neighbors (above, below, left and right). When a request to rotate a cog is issued by the player, a list of all potentially affected cogs is collected. This is done by checking the neighbors of the cog and verifying that the neighbor can be rotated. Afterwards, each cog is stored with a value that indicates by how many degrees the rotation will be made. For each neighbor, this value is inverted (eg if we want to rotate a cog by 90 degrees clockwise, all its neighbors will be rotated by -90 degrees clockwise). This is done to simulate the transfer of force in the real world. Each scanned cog's status is updated, so that the recursive process does not process the same cog twice, entering a loop. After all cogs in the cog chain are obtained, all the cogs are rotated simultaneously. This is achieved by running the rotation for each cog as a separate *coroutine*, which are integrated in Unity. Finally, another coroutine is created that would wait for all cogs to finish rotating. Only after that the rotation action is considered finished and is updated accordingly.

Coroutines in Unity are special functions that can be run frame-by-frame similar to the `Update()` function[31]. However, these functions can run independently and are executed only when needed. Because of that, coroutines are a good way to execute simple animations, such as transferring objects from point A to point B or, in this case, rotating the cogs.

The C# code snippet below illustrates how the cogs rotation is processed. This code does not describe how the rotation itself is processed, only the logic behind how the cog chain is calculated and rotated.

[31] https://docs.unity3d.com/Manual/Coroutines.html

```
void PerformCogRotation(Cog startingCog, float byDegrees)
{
    List<Cog> cogsToBeRotated = FindAllConnectedCogsRecursively(startingCog);
    foreach (Cog cogToBeRotated in cogsToBeRotated)
    {
        StartCoroutine(RotateCog(cogToBeRotated, byDegrees));
    }
    StartCoroutine(WaitForRotationToFinish());
}
void FindAllConnectedCogsRecursively(Cog cogToBeRotated, float byDegrees) {
    List<Cog> cogChain = new List<Cog>();
    if (cogToBeRotated.WasAdded() == true)
        return cogChain;
    if (cogToBeRotated.CanRotate() == false)
        return cogChain;
    cogChain.Add(cogToBeRotated);
    cogToBeRotated.WasAdded() = true;
    foreach (Cog neighborCog in cogToBeRotated.GetNeighbors())
        cogChain.AddAll(FindAllConnectedCogsRecursively(neighborCog,
-byDegrees);
    return cogChain;
}
```

4.4 **Victory Button Transition**

According to the game rules, a player needs to transfer all buttons of their respective color from their side of the game board to the other via the cogs. In order to do that, the cogs were designed to hold information about colors on each of their sides. Therefore, when a player wants to transfer a button from cog A to cog B, the system will recursively check which edges of its own are of the same color as the pin, check if the adjacent cog has the same colored side connected. This process is repeated recursively until a list of all available locations is generated. When this process is done, the player can select the cog to which the button should be moved to by tapping on the highlighted area. The C# pseudocode at the end of this subchapter illustrates how the available paths are found. The special case regarding the starter or ending positions are omitted. These cases will be covered in subchapter 4.5.

```
List<Cog> FindPathForButton(VictoryButton victoryButton)
{
    if (victoryButton.isDeployed == false)
        return GetAvailableStarterPositions();
    elseif (CanTransferToVictory(victoryButton) == true)
```

```
        return GetAvailableFinishTiles();
    Cog currentlyAttachedTo = victoryButton.currentlyAttachedTo();
    return FindAvailableCogsRec(currentlyAttachedTo, victoryButton, new
List<Cog>());
}


List<Cog> FindAvailableCogsRecursively(Cog cog, VictoryButton victoryButton,
List<Cog> acceptedCogs)
{
       foreach (Cog neighborCog in cog.getNeighbors())
              if (acceptedCogs.Contains(neighborCog) == true)
                     continue; //skip to next neighbor
              if (CanMoveToCog(neighborCog, victoryButton) == true)
                     acceptedCogs.Add(neighborCog);
               FindAvailableCogsRec(neighborCog, victoryButton, acceptedCogs);
       return acceptedCogs;
}


bool CanMoveToCog(Cog targetCog, VictoryButton victoryButton)
{
    if (victoryButton.isDeployed == false)
        return false;
    else
        if (targetCog.CanMoveFrom(targetCog,victoryButton.PlayerColor()) ==
true)
            return true;
    return false;
}
```

4.5 **Victory Button Deployment**

From the game start, the button is not attached to anything. Same applies to the “find a victory tile” situation - there are no cogs or pins outside of the board. This makes the algorithm that was described earlier useless in these cases. Because of that, an additional algorithm was developed.

Firstly, each player stores information about the closest row or column to them. This is obtained by calculating the distance from the player base to 4 pins at the center of each grid side. The shortest distance implies that the row or column the pin is in is the closest one. This row is stored as a “starting row”. Additionally, the player's direction is saved for later use.

Direction is calculated relative to the world position of the player pawn. Image 8 illustrates this situation when the amount of rows or columns is even.

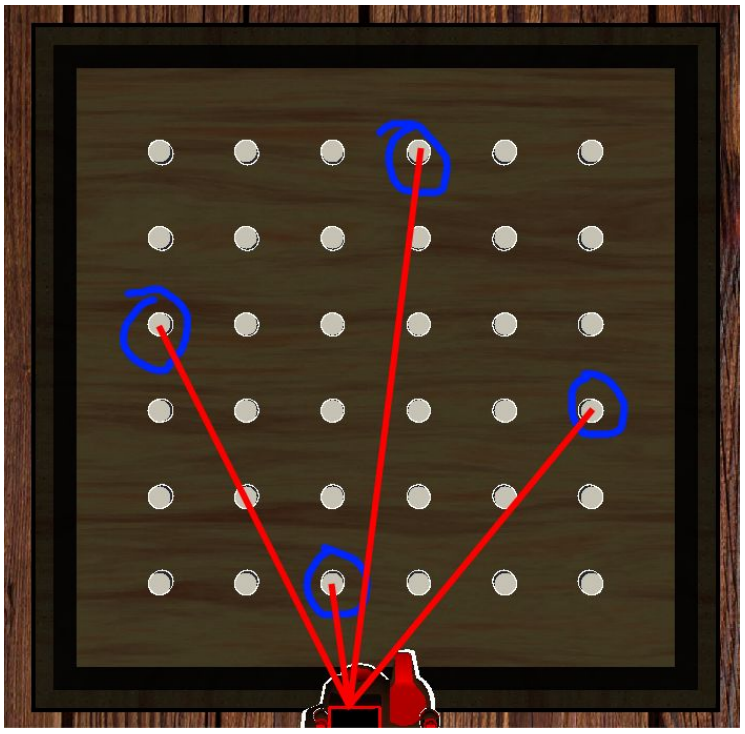


Image 8. Distance to the player from the middle of each edge row/column

Secondly, the starting row is used for first time deployment, if a button has not been moved yet, every pin in the starting row or column is checked for the placed cogs. If any are found, the process is similar to the standard flow - check if the player's direction and color match the opposite direction and color of the cog.

For generating the finishing tiles, the process is reversed - the button has to be attached to the cog that is in the last row for the player. If it is, it is checked if the cog has the player's color in the player's stored direction. With these conditions met, the ending tile is spawned on the outside of the board next to the button. The last row for each player is calculated by getting the most distant pin on the grid (opposite of the starting row).

4.6 **Object Spawning**

Cogbug is a strategy game that does not focus on having too many animations or frequent state changes. This means that the amount of total objects is not large and it does not change throughout the game progression considerably. The objects that are instantiated in the game are as follows:

- Game board – a static object without any logic. This object has no state changes and is more a visual element. Game board also has a background texture to make it look like it is placed on a table.
- Board pins – cogs and blockers are attached to these objects. The amount of the pins is defined by the board size (MxN, 6x6 by default).
- Cogs and blockers – while these objects have different logic, there is also a limited amount that is dictated by game rules (28 cogs + 2 joker cogs, the amount of blockers is equal to the number of players).
- Indicators – these objects are generated when a player tries to move a selected object. Depending on the type of the object to be moved, indicators can be spawned where needed. Due to the difference in usage, 2 types of indicators were used:
    - regular indicators – indicators that are utilized within the boundaries of the game board. The amount of regular indicators can not exceed the amount of pins (m*n, 6*6 by default).
    - victory tile indicators – was covered in more detail in the previous subchapter. These indicators spawn outside of the game board and are not tied to any other game object.

Since the number of objects described above is not large, it was decided to spawn all of these objects at the beginning of the game. This technique is called object pooling [6]. Instead of instantiating a series of identical objects dynamically when needed, these objects are pre-generated in advance and are brought to the scene when needed. With this approach, the objects are not destroyed when not needed, but are merely disabled or removed from the scene. This approach was implemented for cogs and indicators.

Cogs require object pooling to reduce the amount of traffic during the game itself and to ensure all objects are identical over every client. If all possible cogs are already generated, there is no need to make any additional verifications or synchronizations over the network to ensure that every client understands which particular object is to be spawned.

It is important to note that cogs still will need updates for their position, rotation and its internal states (ie color distribution, attached victory buttons or blocker, reference to what it is attached to). Object pooling only shifts the load of object creation to the start of the game.

For indicators, they can be called frequently (as fast as the player can tap the "move" buttons and cancel them), so generating up to 36 new objects each time a player presses a button is redundant. Instead, a number of tiles is pre-generated in advance and, when needed, they are called to the scene with new properties.

Image 9 illustrates the implementation of this approach. On the left side a list of pre-instantiated cogs can be seen. On the right, these same cogs are located outside of the player player view, so they are not rendered until needed. The same principle is applied to the availability indicators that show where an object (cog or victory button) can be moved.

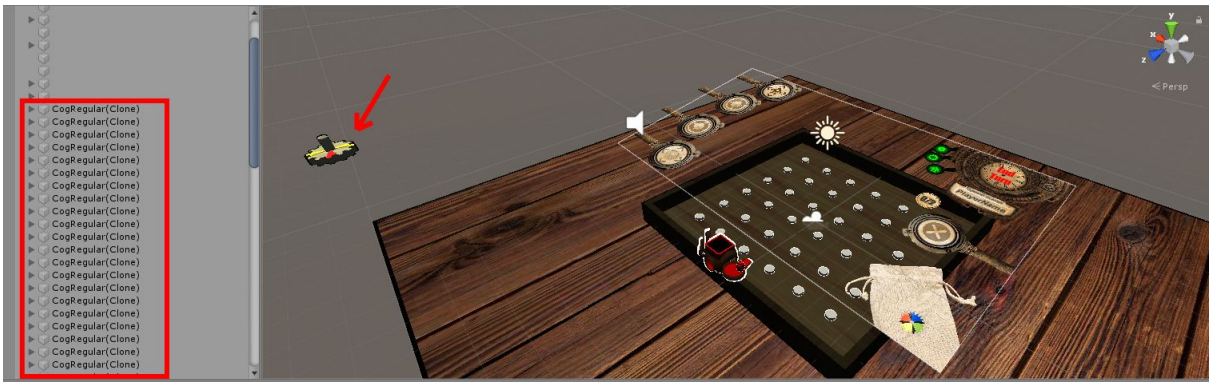


Image 9. A list of pre-instantiated cogs in the game, Unity Editor view

Alternatively, it would be possible to spawn indicators dynamically when needed. At first glance, it may seem more logical as these are not synchronized over the network and are used by each player directly. However, the game is intended to be used on various devices, including weaker or older models. Therefore, it would be better to avoid creating objects too often. Doing that would needlessly reallocate resources and could negatively impact weaker devices. Finally, the number of objects is not large in its current state. Even if the game scale were to be changed in the future, the number of objects will not increase much.

4.7 **User Interface**

In order to handle transitions to various UI views (eg opening the game browser, opening the "Credits" view), a multiple layer UI structure was used. By using multiple layers, it is possible to adjust each view separately as well as ensuring that one view would not accidentally affect another. Additionally, this approach follows the modularity method, as each view or window is set in its own canvas (Unity's element that contains a set of UI).

In this solution, the canvases were designed in such a way that they would be enabled over each other. This can lead to UI issues such as improper rendering. However, such issues are automatically resolved by Unity, as each canvas can either go in front or behind the other.

When not needed, canvases are disabled, thus saving up on both CPU and GPU usages. The drawback is, however, that in Unity, disabled elements and all the objects within their hierarchy are suspended, thus ensuring that the subcomponents are not run. As a result, certain canvases have to either be enabled at more times when they are actually needed, having other canvases overlay them. Alternatively, it is possible to provide all the data needed to the canvas when it is re enabled. The latter is a more reliable solution as it ensures that that old data is either cleared out or is overridden. Therefore, that approach was chosen.

Image 10 illustrates this approach. While the main screen remains active, the overlay canvas, when activated, completely covers the main screen, making it impossible to interact with. However, the main screen remains active, allowing it to keep its internal logic running.

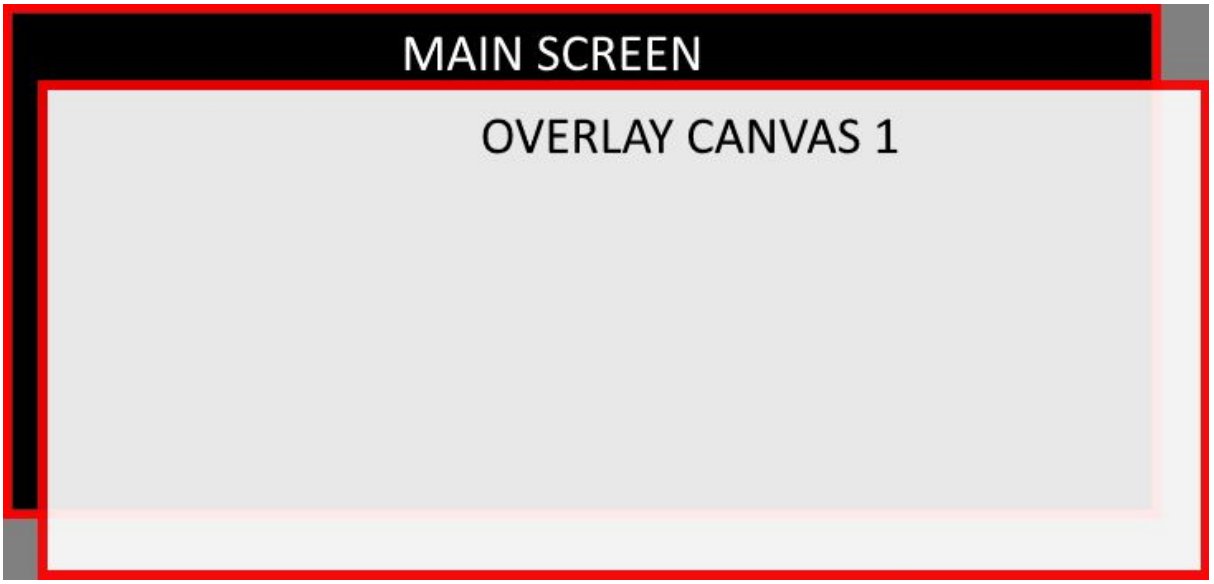


Image 10. “Overlay canvas 1” is above the main screen canvas

The main menu canvas deviates from the general strategy (disabling canvases when not needed). There are two reasons behind that:

- Main menu screen is always visible – most canvases do not completely cover the main menu one or have semi-transparent elements, it is always visible.
- Main menu contains animations – the screen contains running animations (ie cogs and clock hand rotating). Suspending these animations when the screen is disabled is

possible. However, in order to properly resume these animations, more data would be stored. While possible, this is not effective, as users are able to open and close various windows fast, forcing these animations to be suspended and resumed constantly.

These factors considered, it was decided to keep the main menu canvas enabled at all times until the user loads into the game session.

4.8 **Handling User Input**

Since the game is developed for the mobile platforms, the inputs were designed in such a way that the end user is unable to change them. Each input button is located on a specific place that cannot be changed manually. However, their location and scale will change depending on the hardware specifics. Because of that, it is safe to bind the logic directly to the input. Image 4.8.1 shows a comparison of the main menu screen simulated on screens with 10:10, 18:9 and 16:9 aspect ratios

In order to handle user input, it was decided that each input would handle itself directly. This approach attempts to follow the *component decoupling pattern* [6]. This approach creates a more modular solution which makes it possible to change the behaviour of each input without making the code too complicated or hard to read.

Each button utilizes its own instance of a "button template" type class. The template establishes necessary links to the external objects as well as contains a virtual method (a method that can be overridden in child classes) that would handle the "cancel" action. Each actual implementation relies on the basis made by the template as well as implements its own response to the cancel function. Additionally, the child scripts contain the respective logic that cannot be described universally in the template.

When an action button (rotate cog, move cog, draw new cog or move the victory button) is pressed, the button checks whether or not the owning player has performed the action. If an action has not been performed and the player is not performing any other action (unless explicitly defined), the button press logic is executed. Otherwise, the action is ignored.

When a button is pressed and the action is performed, all other buttons are disabled unless explicitly defined by the button logic. For example, if the player presses a "move cog" button, both "rotate cog" buttons stay enabled, since that is defined by the game rules, but "draw new

cog" and "move victory button" buttons are disabled. This ensures that the number of button combinations stays limited and explicitly defined.

Image 11 below illustrates this approach. It depicts a situation when all actions except the "move victory button" one cannot be used by the player. The reason behind that is that the player has not selected any cogs and, therefore, cannot perform any actions with them. This situation occurs most often at the start of the turn. In this case, however, a more narrow situation is depicted as the image shows the first turn in the game session. Therefore, there are no cogs placed on the board yet.

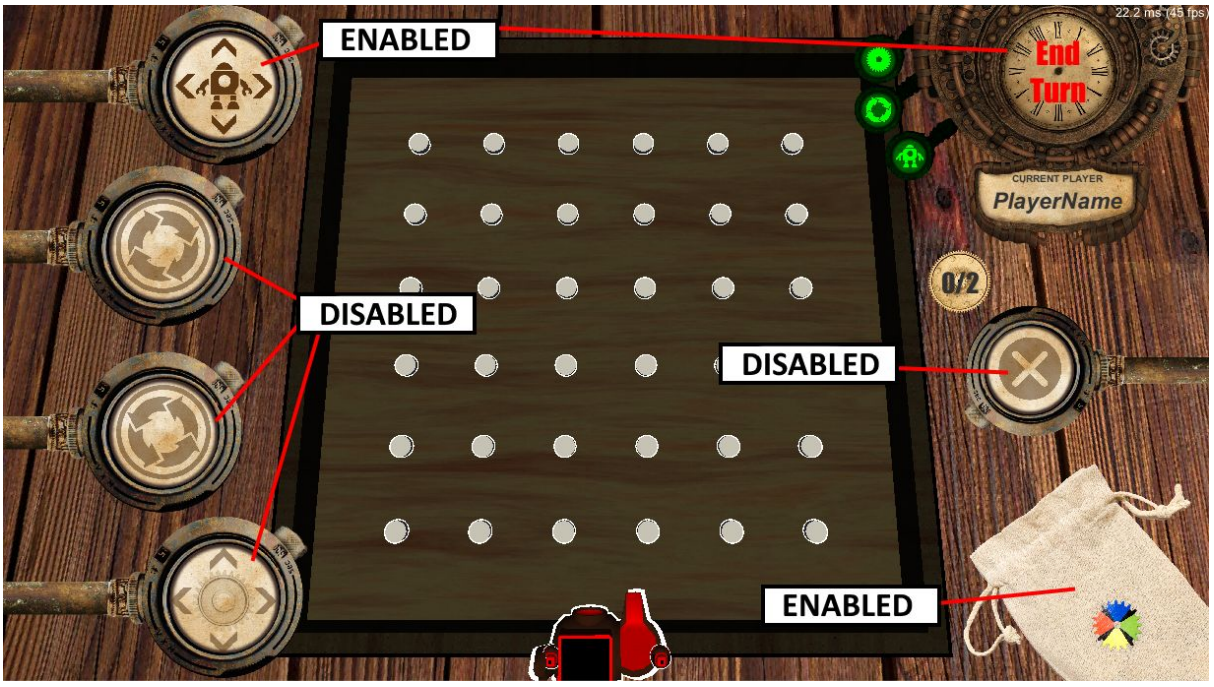


Image 11. MVP user interface in at the start of turn

Image 12 illustrates another situation. The player has selected a cog from the field and has pressed the "move cog" button (bottom left). According to the game rules, the player cannot draw new cogs or interact with victory buttons, so these buttons are disabled (the draw cog button has the linen bag texture which is changed to "closed" state when the button is disabled). However, the player is allowed to place the cog on the slot, rotate the cog or cancel the action - these buttons are enabled.

Image 12. Player tries to move a cog

Finally, there is a situation when there already are cogs on the game field. Image 13 shows a comparison in the UI changes in this situation.

On the left side it is visible that the buttons act in a similar way as at the start of the turn. The difference is that one of the turn indicators next to the “End Turn” button is now red. The reason behind that is that a cog drawing action has already been performed this turn. Additionally, the button responsible for drawing cogs is also disabled, as indicated by its changed texture.

On the right side of the image is a situation when the user has selected a cog. As the move action has already been used during the turn, the “move cog” and “draw new cog” buttons are both disabled. Additionally, an indicator appears around the selected cog to provide visual assistance to the player.

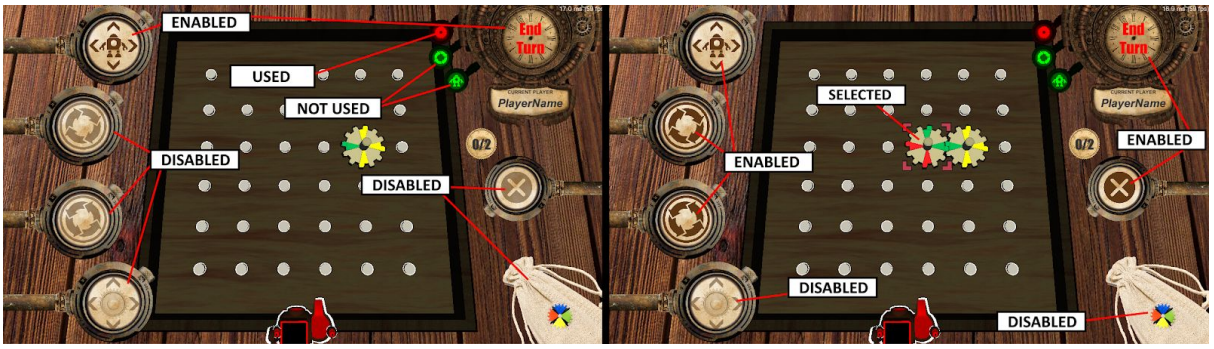


Image 13. Comparison of button states in the middle of turn: no cog selected (left), cog selected (right)

Such an approach was considered efficient. It visually explains the player what they can or cannot do during any particular situation. Additionally, it prevents the player from entering states that could be unaccounted for.

4.9 **Online Multiplayer**

There exist several premade solutions for Unity that enable networking an application [22]. For this project, Photon Unity Networking 2 (PUN 2)[32] SDK was chosen. PUN 2 is a big project and offers a lot of tutorials, examples and is simple to learn and use. As the application is intended to support up to 4 players per session, this solution is the most optimal one as its alternatives (eg Unity built-in networking[33], Mirror[34]) either are deprecated, have a smaller player limit per session or are new and, thus, undocumented or untested. Additionally, PUN 2 offers a free plan that supports up to 100 concurrent users, which is ideal for the development process and early stages of the project.

PUN 2 developers state that the API utilizes a *client-to-server* connection system[35]. However, the free version of PUN 2 does not allow the server to execute any log and instead forces the server into a relay mode. The latter makes the free version behave in a way that is similar to the *peer-to-peer* connection, but the messages still pass through the relay servers hosted by Photon, making it a client-to-server connection.

As a result, PUN 2 sessions have no host player. Instead, the player that creates the room is assigned as a *master client* and all game objects that do not belong to any other player are

[32] "Photon Unity 3D Networking Framework SDKs and Game Backend | Photon Engine" https://www.photonengine.com/en-US/PUN
[33] https://blogs.unity3d.com/2018/08/02/evolving-multiplayer-games-beyond-unet/
[34] https://assetstore.unity.com/packages/tools/network/mirror-129321
[35] https://www.photonengine.com/pun

assigned to this player, though are classified as “scene objects”. The difference between a master client and other clients is that PUN 2 offers built-in functionality to differentiate a master client from others, allowing certain pieces of application to be executed only once and only by one specific client.

Additionally, PUN 2 offers a master client migration functionality. If a master client were to leave the room by any means, a new master is chosen from the remaining players. This acts as a safety measure and is useful for mobile devices. Stationary devices (eg personal computers) use wired connection or, at least, do not change their location. On the other hand, mobile devices constantly change their position and orientation. This leads to mobile devices to frequently look for a better connection and, if possible, reconnect to them[36]. This, in turn, may result in a less stable connection.

Finally, PUN 2 automatically removes all objects belonging to a player should the player disconnect. This ensures that a game can go on since the other players and game logic are not obstructed by objects with no players. However, this mechanic can have an issue when combined with object ownership. In order to avoid that, two measures were implemented:

- If a player is not responding, the server would wait 1 minute for the response before considering the player “disconnected”. This negates minor connection issues.
- If the player is already disconnected, the game is cancelled and all remaining players are moved to the main screen. While such an approach is not the most effective one, it was considered sufficient and easy to implement.

As an alternative, it would be possible to reserve the place for the player until they reconnect. This approach has multiple drawbacks. For example, the following situations would need to be addressed:

- Player’s victory buttons do not allow for the underlying cog to be moved. If all player data is preserved until they return, other players would be unable to move certain cogs.
- If the player disconnects, a set of measures would need to be developed in order to make it possible for them to reconnect.

[36] https://www.electroschematics.com/mobile-phone-how-it-works/

- The player was performing an action when they lost connection. In this case, this action would either be required to finish or to cancel. Additionally, if the disconnect happened during the player's turn, that turn would be forced to end automatically, thus forcing the disconnected player to not gain any progress in a strategy game.
- When reconnecting, the player would need to load the updated game state. This would require all the networked modules to share all of their essential data with that player. While possible, this would be an impact on the network load.

Considering these factors as well as the initial requirements regarding the application, it was agreed by the team (see subchapter 3.4) to implement quick match finding game mode first. Additionally, a game room browser was implemented (described in subchapter 4.4.2) so that the player would be able to either choose a game, quickly join one or create a new game.

PUN 2 implements the following architecture within it: all cloud servers are split into regions. These servers are hosted by Photon. When a user launches the application, they are connected to either the best available server or the one that is predefined in the configuration files (depends on implementation). Each server can contain multiple *lobbies* and always has one by default. A lobby can be considered a specified list of rooms. Each lobby can contain multiple game rooms, which are essentially multiplayer game session instances. It is possible to completely avoid using the concept of lobbies, however that can yield unexpected results such as the player not always being able to see the existing rooms in the game server.

Image 14 shows the difference between how messages are exchanged between host and clients in the built-in Unity and Photon APIs. This image shows a situation when Client B needs to send a message to Client C. The general architecture is the same: the server does not not hold any game-related information, but simply acts as a relay. However, Photon allows a faster interaction between clients as it is possible to directly address the client instead of having all messages go through the host.

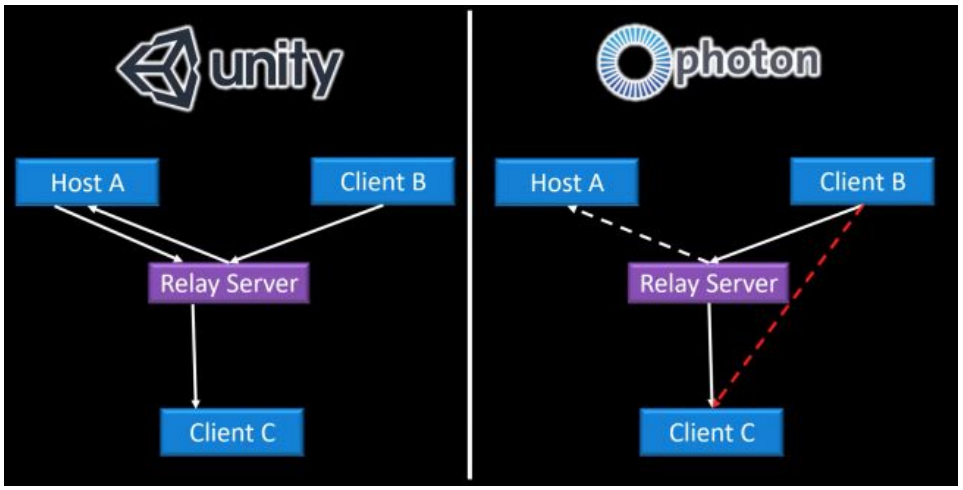


Image 14. Differences between Unity Networking and Photon PUN communication[37]

This approach still requires the data to be transferred through the relay server, instead of direct connection, but multiple testing sessions performed showed that such an approach is successful with mobile devices.

In order to reduce the amount of data that would be exchanged between the clients, it was decided to keep both the amount of networked objects and synchronized data to a minimum. For *Cogbug*, the most important elements are key objects' position (eg cogs, victory buttons) and rotation (cogs only). Other elements, such as graphical user interface state or various indicators that would spawn on the field, are less critical and do not require to be shared. Finally, core game logic and states should be synchronized as otherwise the players would not be able to effectively get information about the actual game progress.

In order to implement that, it was decided to use PUN 2 built-in functions. PUN 2 provides functionality to synchronize object position and rotation of networked objects[38]. However, other elements of the object require manual networking. For that, it was decided to call updates only when state changes would occur. The actual changes would be applied locally on every player. For example, if player 1 moved their victory button, other clients would only receive the updated position of the victory button. However, if the player were to finalize the movement action, all other clients would receive a call to update this particular button's state to "attached to another cog".

[37] https://www.youtube.com/watch?v=xLECRl1eyGk
[38] https://doc.photonengine.com/en-us/pun/v2/gameplay/synchronization-and-state

While such an approach is more vulnerable to client-side modifications, it is extremely simple and ensures that only the important data is sent and delivered to everyone. It is possible to implement an additional module that would verify the changes and compare these changes to the states of other players. However, the team decided to not implement this as it would require a significant increase in traffic between clients.

During this stage of development, the single-player module that was previously developed, had to undergo multiple reworks. This is due to the fact that certain behaviour that would work locally had to be addressed in a different manner in order to properly and effectively transfer data between clients.

4.4.1 Player Identification

To differentiate and identify players over the network, a nickname system was implemented. In the main screen the players are prompted to enter their nickname, which will be used in the matchmaking. Should a player submit their nickname, it is saved locally for subsequent game sessions. This is implemented using `UnityEngine.PlayerPrefs`[39]. This approach is commonly used in Unity to store any player data. In this project, audio volume preferences are also stored in a similar manner.

In case the player does not submit their nickname and tries to access any of the game modes, one is automatically generated for them. This ensures that a player gets at least some sort of a nickname and can be identified over the network. The automatic nickname generation was implemented in a simple manner, always starting with 'User_' and followed by a pseudo randomly generated number.

While this may lead to multiple people getting the same nickname (eg 'User_1234'), it is not a critical element. The nickname is used merely as a visual identifier of the player. When a user connects to PUN 2 servers, they are automatically assigned a unique identifier, which will be used by the API code for player identification and will not be displayed. The nickname is displayed in the main menu (Image 15), in the game room browser (Image 16) and in the main game, when it is the player's turn (Image 17).

As an additional means to identify the player whose turn it is, it was decided to highlight that player. Image 17 shows that the active player has a white outline. This identification is

[39] https://docs.unity3d.com/ScriptReference/PlayerPrefs.html

synchronized across all players, so every player will be able to see the same player with the white outline. When it is not the player's turn, the outline turns to black.

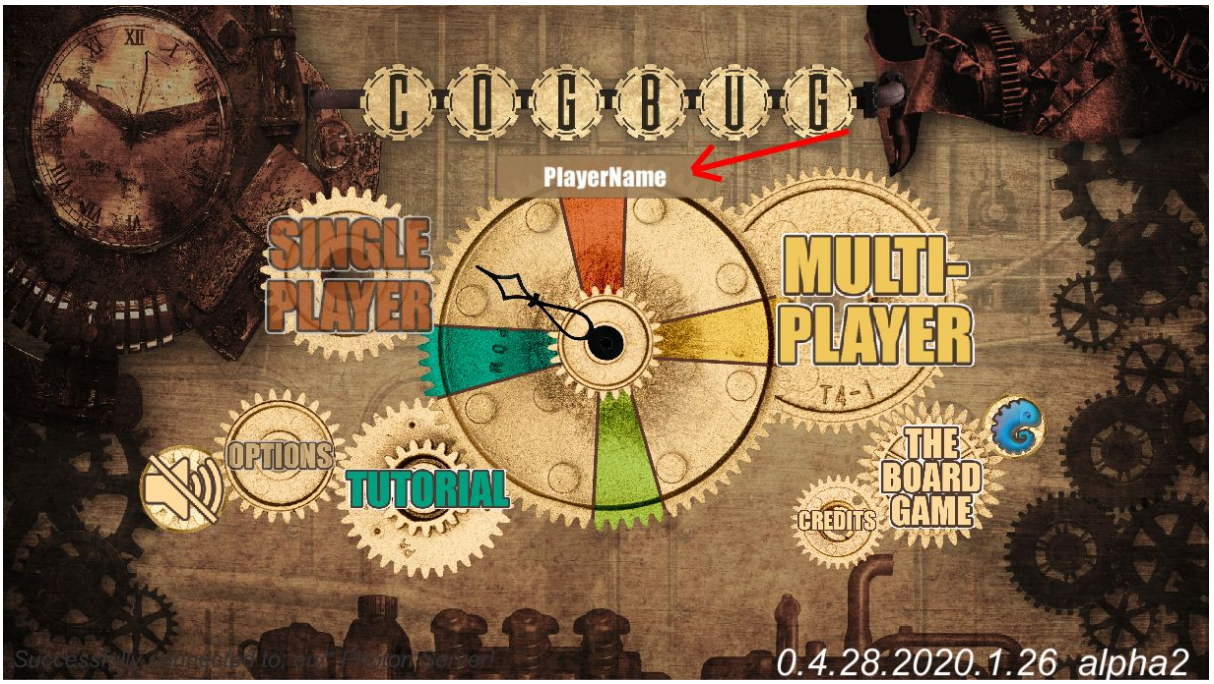


Image 15. Main menu, custom set player name "PlayerName" is displayed

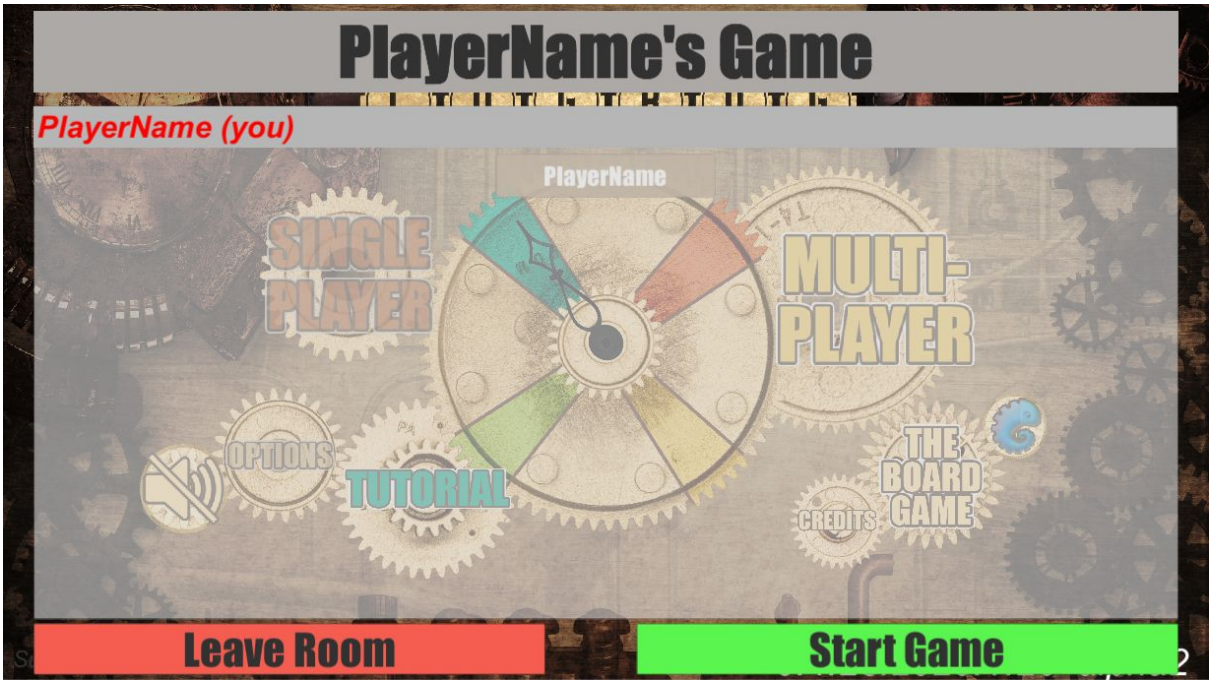


Image 16. Game Room info, player's nickname is on the list

Image 17 In-game, current player's nickname is below the "End Turn" button, active player model has a white outline

Alternatively, other means of player identification could be used. For example, introducing animations to indicate the current active player or adding unique sounds to each player that would be played when their turn starts. However, these were considered too complex for the MVP product. Additionally, it could negatively impact the performance on weaker devices. Due to the time limitation set for the MVP, it was considered to postpone the discussion regarding such means to a later date.

4.4.2 **Game Room Browser**

Since the core element of the application is multiplayer, a game room browser was implemented. This is an element of the user interface where players can see the list of existing games.

In PUN 2, it is possible to fetch a list of games using the built-in functions[40]. By doing that, a list of game rooms that are open and not full is retrieved. This means that such rooms (full or closed) will not be visible to other players by default.

40 https://doc-api.photonengine.com/en/pun/v2/class_photon_1_1_pun_1_1_mono_behaviour_pun_callbacks.html#aab1c0bf03b29543c85fba6f116da6986

The visual design of the game room was not changed from its early design for the MVP. Image 18 shows the simulated situation where multiple game rooms were created. Each available room is displayed, its name and the amount of players in the room are displayed.

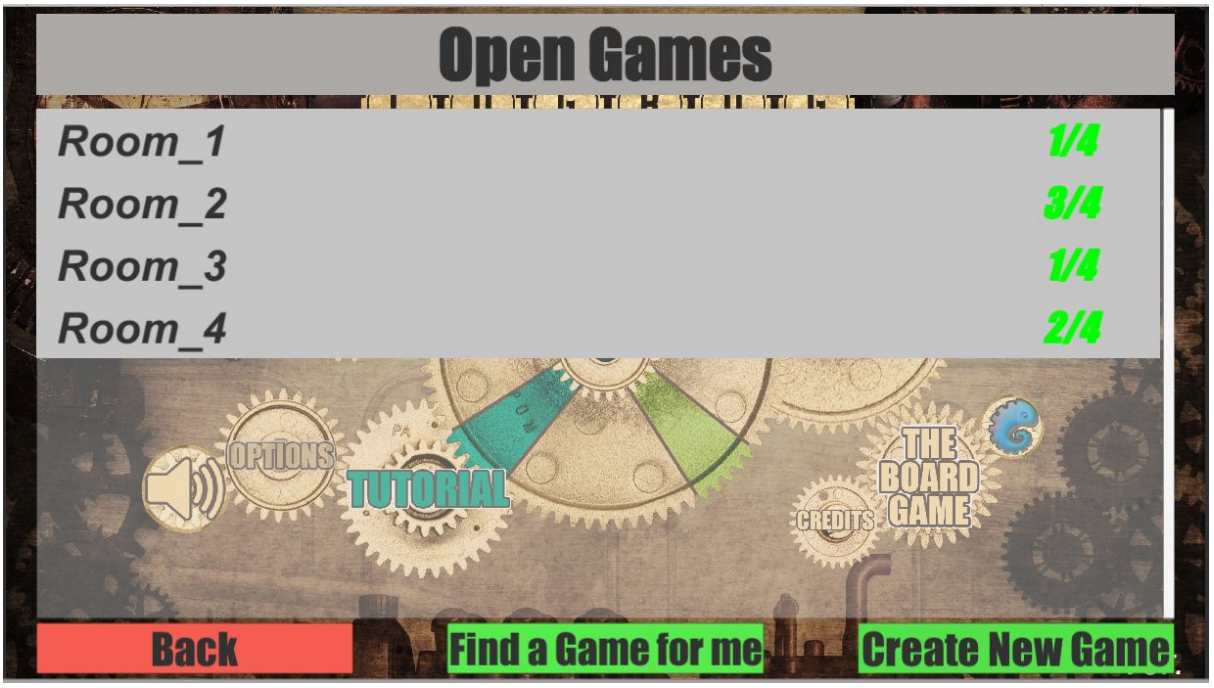


Image18. Simulated game room browser with 4 games available

It is also important to update the list of existing games. For that, there are two options. The first one is to make the update triggerable by the player. By triggering the update, a list of rooms would be fetched. This approach reduces the network usage, but could negatively impact the user experience, as a player could try to join a game that is already closed or has already started.

As an alternative, Photon offers a dynamically updating solution. When a server room's status is changed, it sends a message to all listeners notifying about the change of the specific room. This requires more network usage, but allows for a more reliable update method. Since the rooms that do not meet the criteria are removed as soon as possible, special cases (eg several players trying to join the same room that has only one spot, players trying to join a game that is no longer available) occur less often.

4.4.3 **Quickplay**

Quickplay is one of the options for finding a game room. This option relies on the idea that the player does not care who to play with and instead just wants to find an open game as quickly as possible.

PUN 2 provides means to implement such functionality. For this application, a built-in function `PhotonNetwork.JoinRandomRoom()`. This function, as follows from its name, looks for any game room that is not full and that is not locked and attempts to join it. This method can fail in the following situations:

- No available rooms were found – happens when either all game rooms in a lobby are full, locked or there are no rooms at all.
- The room is available but it is no longer joinable – happens when the host starts the game, locking the room while the player is joining. Alternatively, another player could join the last empty spot before this player's joining is finished, thus making the game full.
- The room is no longer available – happens when the host closes the room while the player tries to join it.

Regardless of any of these outcomes, this method calls back `OnJoinRandomFailed()`. This solution assumes that such a situation can occur. So, in order to work around that, a new game is created. In short, the logic behind quickplay can be described as "*find a game for me, but if there are none I can join, create one*". Diagram 2 illustrates the logic executed when the player presses the quickplay button.

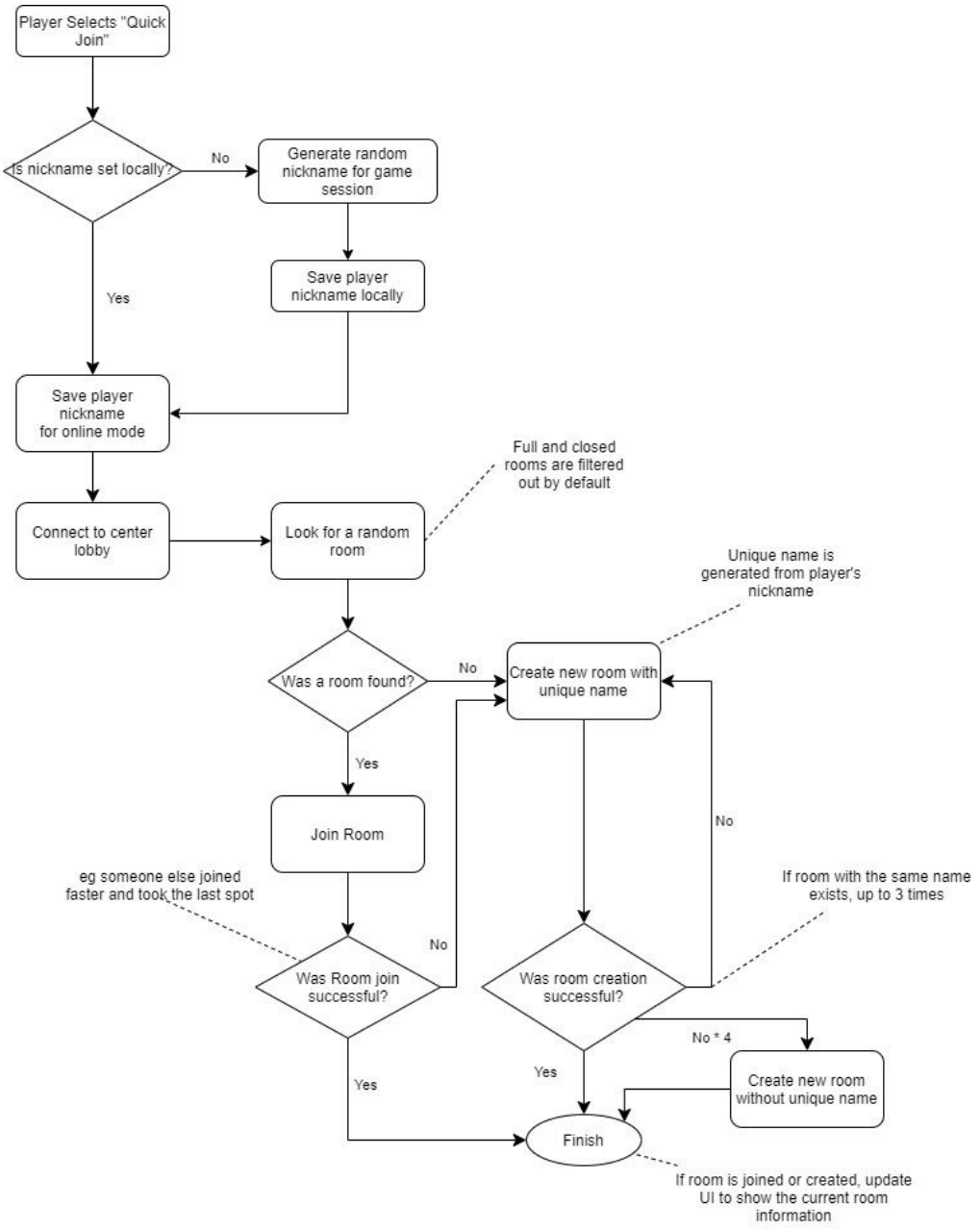


Diagram 2. Flowchart displaying quickplay match finding process

In case when a room creation fails, it is assumed that the process fails for any other reason than there already being a room with the same name. Therefore, when reattempting to create a room with a unique name, the room name is not changed. On the fourth attempt, a room without a unique name is created. This approach may be less efficient than looking at every possible server response during room creation. However, it was chosen as it is simpler to implement and was deemed sufficient for a small project.

4.4.3 **Room Creation**

PUN 2 offers two ways for creating a room:

- Creating a generic room
- Creating one with a unique identifier

Both of these room types provide the same functionality, but are slightly different in how they are made.

The identifier in PUN 2 is also referred to as *room name*. The latter originates from the fact that room ID is stored as a string and thus is automatically used as room name by PUN 2. Because of that, the difference in these room types is whether or not the client provides their own name for the room.

A room with a unique identifier has its restrictions. There can only be one room with such a name in the game lobby or server. Normally, that would not be an issue, as the users cannot directly affect the name of the room. However, the room name is taken based on the nickname the user provides, so it is possible for such an issue to occur.

To counter the problem with colliding room names, the client would attempt to create a named room only a set number of times. Should it fail, it will request to create a generic room instead. A generic room is one that is created without any identifier provided, therefore, does not have a unique name attached to it. Instead, the server automatically assigns one for them.

In general, most of the solutions chosen could be considered successful. Some of the solutions were chosen for their simplicity and are a subject for rework or discussion. The elements that would need to be revisited the most would be game room browser design and functionality and visual design. The goal of the development was to produce an MVP first. Because of that, elements that provide core functionality may be simplistic, but they deliver the results that they were intended to and passed the evaluation by the Customer.

Integration of the multiplayer module into the digital application of *Cogbug* was scheduled as a last major milestone where new features would be implemented. After the product was internally tested, the team agreed to perform a series of closed testing sessions with users.

5. The Testing

This section covers the testing approaches performed during the development of the application. During the whole process, two types of testing approaches were used. The first one was an internal testing type, which was carried out dynamically alongside with the development process. The other one was considered as the final stage of the MVP development – user testing.

5.1 Internal Testing

The Customer did not have previous experience in online game development. In order for them to better understand the differences in the mobile version of the game compared to its physical version, the agile development methodology was combined with test-driven methodology. Additionally, it would give them a possibility to adjust their wishes in accordance with the capabilities of mobile devices (demonstrated on the first internal tests).

During the development process, a series of tests were carried out. After every major change or update was made, the team would test the newly updated mechanics. If bugs or issues were discovered, they would be fixed. Testing sessions were designed in such a way that they would cover both product usability and accordance with requirements.

In a series of cases, these internal tests allowed the team to discover deviations from the original design. Such deviations occurred due to insufficient task specifications. As the digital game of *Cogbug* was agreed to differ from its physical counterpart, several aspects of game mechanics were left undiscussed. Such an approach allowed the team to experiment with mechanics that would differ from the physical game and could not be brought up by the Customer in advance.

For example, the blocker cogs were initially designed to be simply an alternative type of a regular cog: it was possible to place them only on empty pins, they would not transfer rotation or allow victory buttons to be placed. However, after the testing sessions with the Customer, it was agreed that such an approach would not work and would deviate too much from the original physical game. As a result, this type of cogs had to be reworked to behave in a manner that would resemble its physical analog.

Additionally, internal testing helped with game balancing or behaviour. As the team would test the game, they would spot out the mismatch in various elements of the game. Such cases would be discussed by the team, re-evaluated and eventually reworked. Below are the three most important changes that were made due to the internal testing. These changes are not put in any particular order, as the outcome of the balancing changes cannot be evaluated directly before extensive user testing:

- Victory button movement – initially it was possible for players to transfer their victory buttons to the victory tile that would progress the player towards victory. It was considered to speed up the game too much due to the reduced game grid size. After the discussion the team agreed that it would be more reasonable to move the transition to the victory tile to a separate turn. Currently, players can do so only if their victory button is already on the cog that is adjacent to the opposite board end.
- Victory buttons – this was mentioned in Chapter 4.5. The initial amount of victory buttons players would get would actually be 4. Later it was changed to match the physical game amount (3). As of the current version, it was decided to reduce it even further to 2. This still does not eliminate the tactical parameter of the game, but greatly shortens the late game stage, where the amount of placed cogs would be large.
- Game speed – during the earlier stage of development, the client representative stated that the actions the player would take were too slow. This was due to the fact that the object transitions would be too long. In order to speed up the general game pace, the animation times were reduced by 25% on average. Additionally, this decision brought the idea to store the game speed for all players in the session. This was implemented for a planned feature where master clients would be able to customize the game pace in the game room alongside other features.

After the last testing session, where multiplayer elements were tested, was completed, it was agreed that the product is ready for closed usability testing.

5.2 **Usability Testing**

Before releasing the application into the public access, it was agreed that it would require to undergo a closed testing phase. This phase would evaluate how well the application was designed and whether or not it appeals to the end user.

5.2.1 **Tools**

To properly record the testing session, an external solution was required. This solution needed to allow up to 4 mobile devices providing a simultaneous screen share translation to a fifth device. Most solutions that work on mobile platforms allow only one screen sharing device per session. For user testing, a number of such solutions were reviewed.

To observe the flow of the testing phase AirDroid[41] application was chosen. This application allows the user (in this case, the developers of the game) to monitor the activity of the mobile device via a web-based platform. Because of this, it is possible to monitor what each tester is doing during the testing phase. AirDroid does not support multiple screen shares per session, as it is more of a device-to-device solution. Because of that, up to 4 separate AirDroid sessions would be required when observing the testers. Image 19 shows a screenshot from the second testing session. The screenshot depicts the screens of each participant in the game at the same time.

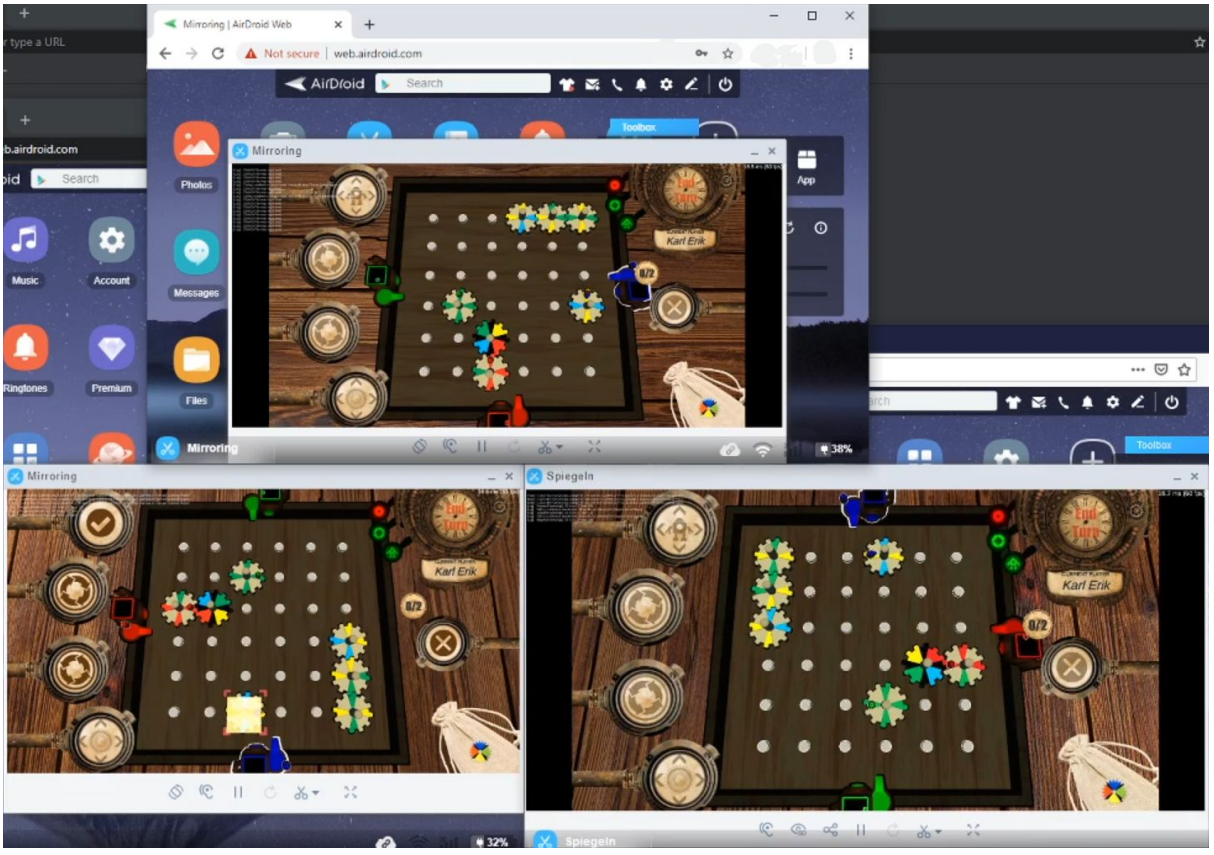


Image 19. Screenshot showing three concurrent sessions of AirDroid

[41] https://www.airdroid.com/

There exist alternative solutions that can be used for acceptance testing monitoring. For example, Zoom[42] allows creating a single meeting where everyone can use screen sharing. However, its free licence limits the meeting length to 40 minutes per session. Additionally, Zoom does not allow mobile devices to use the screen share feature if anyone else is doing so already. The latter makes Zoom inefficient for mobile application testing. On the other hand, by purchasing a licence it is possible to effectively use this solution for PC-based applications.

Another alternative that was reviewed was See[43]. This application is intended for spontaneous meetings and supports up to 9 people per session. See supports voice communication and either a screen share or camera feed. See utilizes a link system in order to allow people to join the sessions. So when someone creates a new call, See generates a link that they can share to anyone they want. This system makes See a cross-platform solution as it is possible to join the session using either a smartphone or a computer using a web browser. This means that this application is suitable for small testing sessions. The downside of this application is that it is unstable when used over an open network. As a result, this application would crash without notifying the user, which makes the observation of the testing progress more challenging. Additionally, when the application would crash, the tester would have to reopen the application and set up the screen share manually. This could potentially distract the testers and thus decrease the quality of the testing session. Therefore See was not considered as a solution stable enough for this case.

Third alternative was Slack[44]. Slack is designed as a communication platform for businesses and developers. This solution offers the possibility for multiple users to share their screens over the call. However, this functionality requires a subscription per each individual user.

Finally, Screenleap[45] was reviewed. This solution is similar to AirDroid as it allows screen sharing from a mobile device to a web browser on a PC. Screenleap provides a more stable connection than See, but the resulting image quality reduced considerably when the application was tested over the open network.

[42] https://zoom.us/
[43] https://play.google.com/store/apps/details?id=castcircle.see
[44] https://slack.com/
[45] https://www.screenleap.com/

After considering all the solutions and their drawbacks, AirDroid was chosen. This solution would require to run up to 4 browser sessions at the same time to be able to observe all the players. However, that is the only major drawback which can easily be resolved with a powerful enough stationary computer. Other options, however, either do not behave in a stable manner or limit the free license functionality.

5.2.2 **Testing Groups**

Acceptance testing took place from 30.04.2020 to 02.05.2020. The registration for invitation did not restrict anyone, but required a direct invitation. Initially, 3 sessions were scheduled, one for each day. However, due to the insufficient amount of registrations for the session on May 1 (1 person), it was decided to create a second schedule on May 2 instead. Table I below shows the finalized participant distribution throughout the sessions.

Table I: Distribution of members that signed up for the testing sessions. Red indicates those that did not actually participate in the registered session

Apr 30 17:00 - 18:45	**May 1 15:00 - 16:30**	**May 2 16:00 - 17:45**	**May 2 18:00 - 19:30**	**Notes**
			+	Signed up too late
			+	**Participated**
		+		**Participated**
		+		**Participated**
			+	**Participated**
		+		Was absent during testing session
		+		**Participated**
+				**Participated**
+				**Participated**
+				Technical difficulties
+				**Participated**

Participants for the testing sessions were requested from the *APT GameGenerator*[46] Facebook group and members of the Computer Graphics and Virtual Reality Lab[47] from the University of Tartu. The first choice was made as this particular group was created for game developers in Estonia. The second group was selected based on the similar interests as it includes students studying computer graphics who are already interested in the field.

The testers registered themselves via a *Doodle*[48] poll. This solution was chosen for the following reasons:

- **Transparency –** Doodle shows up every user that has submitted their choice to every other user that visits the poll. This allows those who register later to see the distribution of the participants and make decisions based on that.
- **Ease of setup** – Doodle poll creation is designed in such a way that a user does not need to spend much time. One specifies the poll topic with an optional description, the dates the poll should cover (times are optional) and sets up additional settings when needed.
- **Entry number restriction** – when creating a poll, Doodle allows limiting the amount of votes for each entry. This is ideal for scheduling testing sessions where the number of concurrent players is limited. In this case, the limit was set to no more than 4 votes per time slot. Additionally, it is possible to limit the amount of votes each user can make. This ensures that the participant will choose only one time slot that suits them the most and will not participate in the testing more than once. The latter follows proper usability testing procedure as well as ensures that as many people can participate as possible.

In total, there were 11 registrations, but 3 of them were discarded due to reasons described in Table 5 (above). As a result, 3 testing sessions, 2 with 3 users and one with 2, were held. Ideally, another testing session with a full 4-user team should have taken place. That way the testing with 2, 3 and 4 players would allow to test all possible setups that are allowed by the game.

[46] https://www.facebook.com/groups/GameGenerator/
[47] https://cgvr.cs.ut.ee/
[48] https://doodle.com/

5.2.3 **Testing Evaluation Methods**

In order to gather more feedback, it was decided to put the application to public access. However, public testing by itself would not generate enough information regarding the application. This means that information such as number of downloads or performance feedback (overall weakly formalized and generalized feedback) could be obtained from passive data analysis. Information about playability or user experience, however, cannot be reliably obtained this way. This would be possible if the application were released on Google Play or Apple Store as these have the option to prompt people to rate the applications. However, the testing at this stage did not assume releasing the application. Therefore, another approach was required.

In order to obtain the latter, testers were prompted to submit feedback via a questionnaire (see Appendix V). For this, a set of questions were developed. These allowed the player to rate various elements of the application.

The questions were split into the following groups:

- **Gameplay –** these questions gathered information about the mechanics and how they were implemented in the solution. Each question in this section consists of two parts:
    - Requesting the player to rate specific game mechanics.
    - Optionally, provide free text feedback regarding their rating.
- **User Interface –** this section asked the players to evaluate the overall visual design of the application. Similar to the previous section, the questions consist of two parts.
- **Conclusion –** questions put into this category would prompt the player to evaluate the potential of the application. Players are asked to rate the potential future features as well as add free text feedback about the features that were not covered in other questions.

Additionally, it was possible to include several general questions. These would provide information regarding each tester. However, as the device information was obtainable from AirDroid transmissions, it was decided to not add these questions. Information about personal knowledge about *Cogbug* was taken from players directly before the testing sessions began.

5.2.4 **Testing Results**

As mentioned in subchapter 5.2.2, there were a total of 3 testing sessions performed. For further analysis, the feed from the user screens was recorded (See Appendix V). The application's current design, like most mobile games, does not assume any in-game chat or other means of communication between players. Because of that, audio was not recorded.

Each testing session lasted for more than an hour and included at least 2 games each. During these sessions several problems, which were not spotted during the internal testing sessions by the development team, were discovered. Some of these issues were severe enough to block the end game state, which would not let the testers finish the game session by winning. However, after the first testing session, significant problems were eliminated and in subsequent sessions, testers were significantly more focused on the gameplay, and not hindered by the program's shortcomings.

The first testing session took place as scheduled. However, out of the four registered participants only three managed to attend the testing. The fourth participant encountered software issues with their mobile device that could not be resolved within reasonable time before the testing started. Appendix XI contains a video recording of the session of two out of three testers. The third participant was unable to maintain the stable recording due to hardware issues on their end.

This session lasted 1 hour and 15 minutes during which 2 game sessions took place. The feedback from the participants indicated that the application contained several issues that made the application hard to use.

As there was a gap of one day between the first and second sessions, it was possible to improve the application based on the feedback and observations retrieved from the first session. Table II (below) shows the issues that were discovered in the first session, their cause, value for the end user and their status before the second session.

Table II: Issues discovered during the first session

Issue #	Underlying Issue	Effect(s)	Value	Resolution
1	Client-side UI updates	Player was able to end	Medium	Resolved

		the turn while performing an action		
2	Same as Issue 3	Player was able to place an object outside the grid	**Very High**	**Indirectly Resolved**
3	Cross-client data synchronization issue	When moving an object, movement indicators would not show up correctly	**Very High**	Resolved
4	Incorrect calls to the object generation by non-master clients	Player could not draw new blocker cogs	Medium (testers stated blockers are not that useful)	Resolved
5	Photon PUN based object ownership issue	Cogs would finish rotating in a non-90 degree state, ruining the alignment	Low/Medium (affected only visuals)	Resolved
6	Incorrect user evaluation when ending turn on client's end (does not impact anything else)	When passing a turn to another player, the notification would occasionally display active player name incorrectly	Low/Medium (testers got slightly disoriented)	Resolved
7	Unknown	Ending a turn would occasionally skip one of the players	High	**Unknown**

Issues 2 and 7 were not resolved directly, as the underlying cause was unclear. Additionally, these issues were not constant during the session so replicating them and testing was difficult. However, the recording of the game session provided help with these issues. During the testing with the second and third groups, these issues did not reoccur. Therefore, the status of these issues is to be considered unknown.

The feedback of the first group was not analysed explicitly as it was decided to use the overall data from all the sessions to get more complete feedback.

Second and third sessions lasted 1 hour 16 minutes and 1 hour 8 minutes and had 3 and 2 participants respectively. Both of these sessions used the newer version of the application in order to avoid the same issues as during the first session. During the second session 5 game sessions were made and 3 during the third session.

During these sessions, the total amount of issues noted was 6. This is a slight improvement compared to the first session. Additionally, only 2 issues can be considered critical or difficult to resolve. Table III (below) describes the discovered issues.

Table III: Issues discovered during second and third sessions

Issue #	**Session**	**Effect(s)**	**Underlying Issue**	**Value**	**Resolution**
1	**2, 3**	Player was unable to end their turn	Incorrect UI update	**Very High**	**Requires testing**
2	2	Player disconnects when switching active windows for too long	Android system would suspend the application when in background	Medium	Unresolved
3	2	Player was able to perform actions outside of their turn	End Turn button would not correctly finalize unfinished turns	Low	**Requires testing**
4	2	Cog with a blocker placed by another player would not be rotated in a chain	Incorrect permission transfer on turn end by players	Low /Medium	Resolved
5	3	Cogs would occasionally move vertically	Possibly poorly timed rights transition over objects.	Very Low/ Low	**Requires testing**
6	**2,3**	Player cannot move their victory buttons to cogs	Incorrect exception handling when user would spam the move action	Medium/ High	Resolved

Issue 3 would seem like a serious one, as the player would be able to perform actions at any time. However, after analyzing the recording of the issue, the actual issue turned out to be that the player would get stuck with an unfinished action, which could be cancelled when they were to get the turn. This issue is marked as "Requires testing" because of the proper means of ending the turn, which were implemented to fix issue 1.

Issue 1 could be rated as the most critical one as it completely breaks down the flow of the game. Similar to issue 3, it was marked as "Requires testing" as the underlying cause for it was discovered and fixed, but was not sufficiently tested. An additional safeguard to this would be to make player turns limited in time. As a result, even if the player is unable to end the turn themselves, the client application would force the end of turn for them. As a bonus, this could increase the game pace. This solution has not been implemented yet as it is currently being reviewed by the development team in cooperation with the Customer.

While issues 1 and 6 can be considered game-breaking, the feedback received from the testers during and after the testing session was more positive than during the first session. Issue 2 can be resolved by enforcing the application to run in the background. This would increase the battery consumption on mobile devices, but will ensure that the connection is more stable. Issue 5 was considered a minor one as it would not interrupt the game flow. Additionally, it is difficult to replicate so it was marked as "Requires testing" similar to issues 1 and 3.

There is also one additional issue that occurred during the second testing session. During one of the games, an application for one of the participants crashed while they were performing an action. This issue was looked at and several attempts on different devices were made to replicate the issue. However, all the attempts were unsuccessful. Due to that, it was assumed that the issue was client-side.

5.2.5 **User Feedback**

As mentioned in subchapter 5.2.3, all participants were instructed to fill out a feedback form. The first block of the feedback is covered in the next subchapter 5.2.6 as it is an essential topic of its own. Here only the most interesting moments of the feedback form are provided. The file with complete responses can be found in Appendix V.

The section regarding the user experience and UI design holds mostly average reviews. The respondents stated out that the animations were insufficient, slow and too simple.

The lowest average rating was given to the design of the main menu room browser. The users considered the design of this element too similar to the game browser, often not noticing it. Image 20 below shows a visual comparison of the two elements.

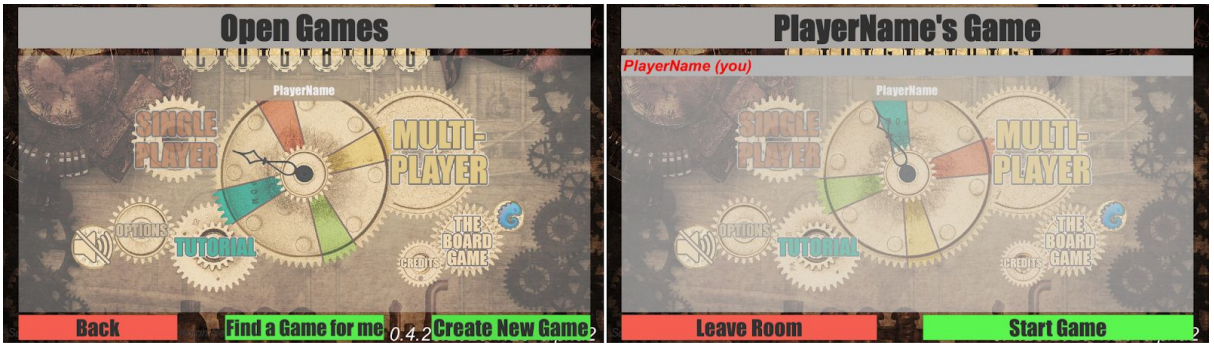


Image 20. Comparison of room browser (left) and game browser (right)

Finally, players stated that the feedback (both visual and audio) regarding their actions wasbe severely lacking. This is critical for the end product and release and should be corrected as soon as possible, in cooperation with the Customer.

In the free text submission, where respondents were asked to provide their own ideas about the product, the presence of issues described in the previous subchapter was also noted. Several suggestions about changing the game rules were also provided. These were scheduled for further review with the development team and the Customer as changing the game rules would deviate the digital game from its physical counterpart. Additionally, such changes would require significant changes both in the application itself. Finally, that would require the game balance to be reevaluated. With these factors accounted for, potential game rebalancing was postponed to a later date.

Generalising the feedback received, the users did not have many serious complaints about game mechanics or the application not working properly. While the animations and input feedback are extremely essential, these can be implemented gradually without having to rework the core elements of the application. Additionally, the graphical elements were not at the highest priority by the testing phase, as it was considered more important to show and test the game mechanics, their stability and behaviour.

5.2.6 **Network Stability**

Another important aspect that required evaluation was the networking module. In order to evaluate those, players were asked to rate various aspects of the application that utilized the networking module.

Charts below (Images 21 through 24) are taken from the Google Forms results of the testing form. These charts illustrate that none of the questioned testers saw any major issues with the connection speed. The last chart additionally shows that players were satisfied with the stability of object movement synchronization over the network.

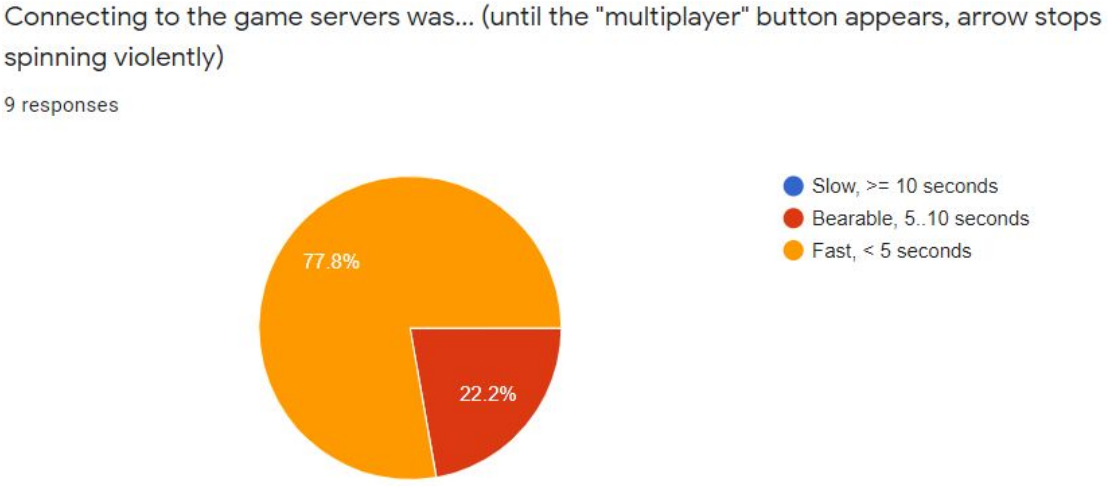


Image 21. Chart of testers' evaluation of initial game loading speed, no negative responses

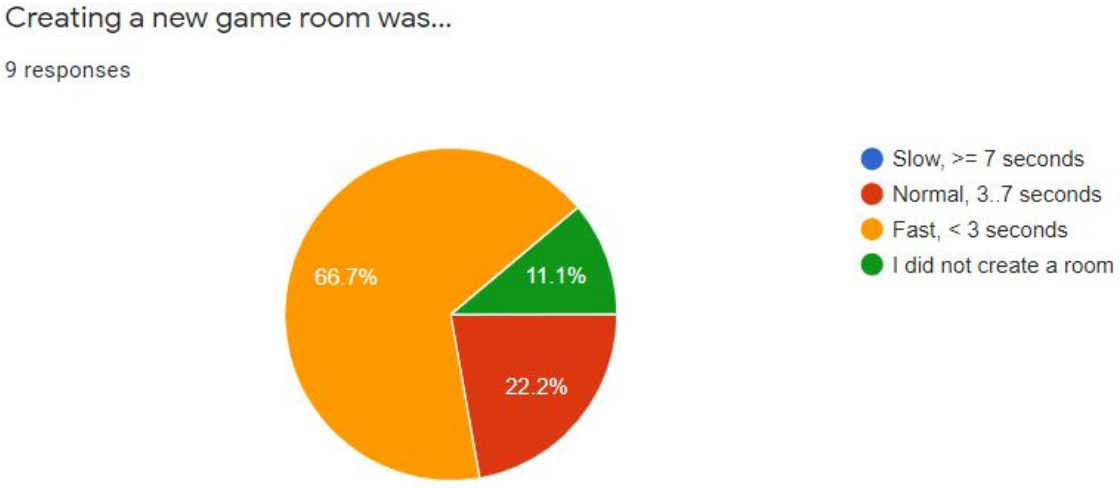


Image 22. Chart of testers' evaluation of game room creation speed, no negative responses

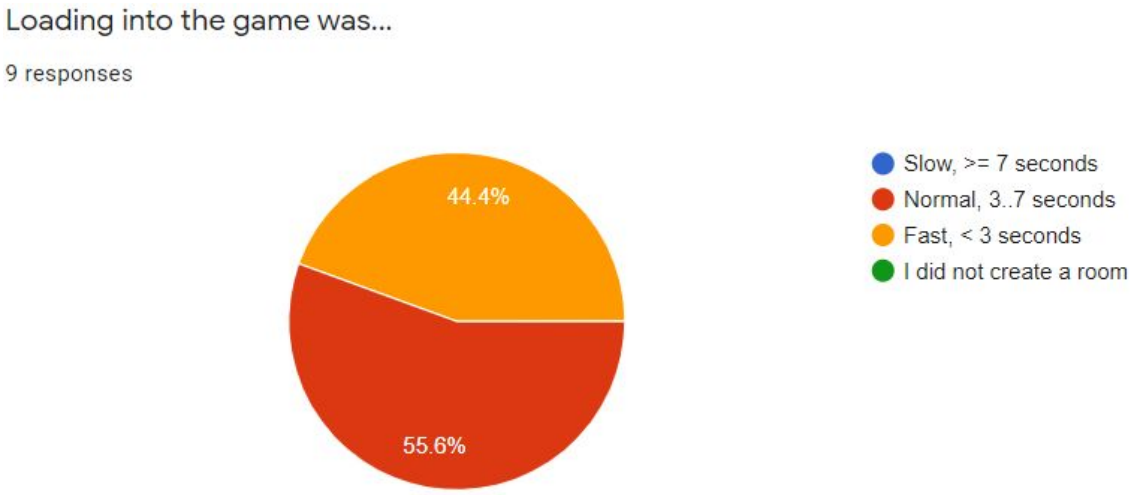


Image 23. Chart of testers' evaluation of game loading and asset instantiation speed, no negative responses

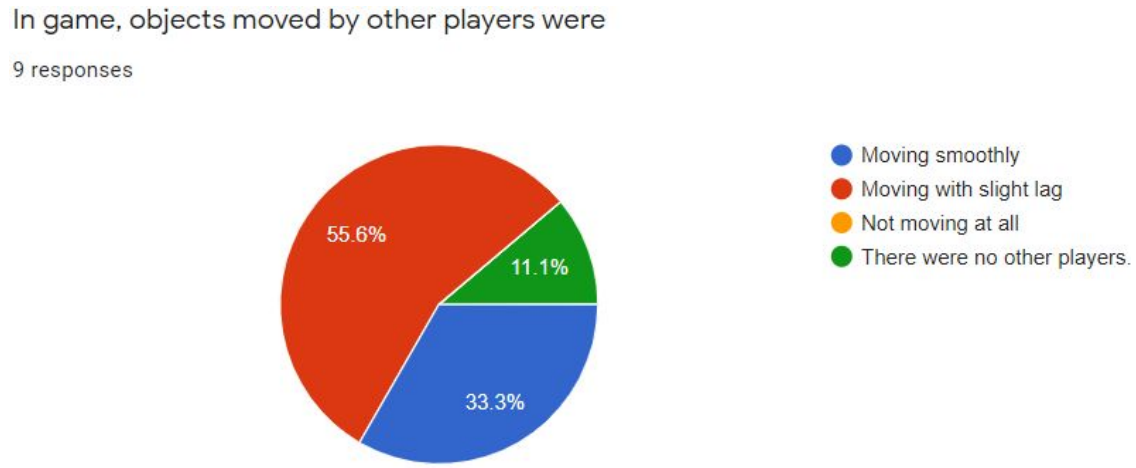


Image 24. Chart of testers' evaluation of networked movement stability, no negative responses

After analysing the testing session recordings as well as feedback, the following conclusion was made: the application networking module was implemented correctly and works in a stable manner, without any considerable interruptions, stutters or desynchronizations. This, in greater part, demonstrates that PUN 2 was a reasonable and correct choice of the networking API. Additionally, this also implies that the solution utilizes the module in a correct way and does neither overuse or misuse the tools provided by PUN 2. The feedback provided by the testers further supports that as none of the testers noted any issues in the network module on their end.

6. Conclusion

During this thesis a mobile application was developed. This application would be used as a digital alternative to a physical tabletop strategy game *Cogbug*. The tabletop version of *Cogbug* was developed in Estonia by Edulab OÜ. The physical game was publicly released on 18th November 2020.

The development process of the mobile version of *Cogbug* was carried out in a team of three members. These members were the author of this thesis, Tarvo Metspalu, who, being a Customer representative, dealt with visual design and later 3D modelling, and Madis Kaspar Nigol, responsible for audio design and its integration in the project.

In order to maintain a stable connection between the team members, it was decided to maintain connection using instant messaging platform Slack. To synchronize the development process, the project was maintained via the GitHub platform.

This project was intended to be a digital counterpart of a physical game, which already had all of its rules and mechanics developed and tested. However, certain elements that are essential in a physical game either can be avoided or reworked (eg object collision physics) or cannot be implemented reasonably in the digital solution. Alternatively, some elements have to be significantly limited and formalized in accordance with the genre of the game (eg communication between players). Additionally, such a type of project was new for the team. Because of that, it was agreed to follow the agile approach during the development process. This would allow a larger degree of flexibility when making decisions and would make it possible to review older efforts and rework them, if needed.

As the project was considered complicated, it was agreed to focus on delivering the MVP first. Doing so would allow the team to implement all the core mechanics first and be able to test them. In order to optimize that, the development process was split into three stages.

The first stage was dedicated to collecting information from the client such as system requirements for the game, technical aspects of the physical game, task specification. It was decided that the application should run on both iOS and Android platforms and support up to 4 players per game session. Since the average duration of game sessions of the physical game

was considered too long (90 minutes), it was decided to develop the digital game in such a way that the average game session would last no longer than 60 minutes.

This stage finished with a developed set of non-functional requirements, a list of essential game elements and assets. Multiple changes from the physical game were discussed and settled, as some aspects of real-world gameplay can either be implemented differently or cannot be implemented at all in a digital solution. Some new features were also introduced (a dedicated undo action).

Additionally, some concept arts of the digital solution were developed. For that, several 3D models were made, which were later used in the application itself. Iit was agreed that the application would be developed using Unity game engine. Paint.NET was used to design textures for 3D models and concept art development. Finally, it was decided that the first task would be to develop the MVP for the digital version of *Cogbug*. This MVP would contain an online multiplayer game mode.

The second stage was dedicated to the development process itself. The process was split into multiple sections where separate elements of the game would be implemented. After the implementation of every new function was complete, they were internally tested by the team. This ensured that the requirements were met. Additionally, each team member tested the solution on their own mobile device, providing a certain coverage of platform testing.

During the second stage, two big milestones were established. First one was to implement all essential game mechanics and make them work offline with one player. Second one would be to integrate this single player module into the multiplayer environment. Such approach was considered optimal as it would allow the team to test the game mechanics themselves without having to worry about the issues that the networking could bring (ie desynchronization, unstable connection).

The functionality of the single player mechanics was put into a separate module. It was planned that this module would then be replicated for each player that would join the game session. However, the process of networking showed that this approach was not the most effective, as certain elements that worked for one player on one device had to be rewritten completely in order to make them work over the network. After these mechanics were tested

and improved multiple times, the team agreed to start the integration of the multiplayer aspect of the game.

The structure that was chosen for the multiplayer was as follows - each game session would have a game logic module, which would store and exchange essential data between the players' local game instances. The players themselves would carry no game logic directly and would only be able to issue commands which would then be processed by the logic module.

In order to approach such a structure, Photon PUN 2 was chosen as a networking API. This solution offers a lot of built-in functionality which made the development process much faster. For example, connecting to game servers or maintaining connection processes and logic were already implemented within PUN 2.

PUN 2 offers several options for establishing connection between players. The one chosen for this project was the cloud-based approach. The benefit of such a solution is that no data is stored in the cloud directly, but having no host between the players either. Because of these factors, all application instances that join are considered clients, with one of them being the master client. This allows for easy "host migration", should the master client leave the game. Additionally, this solution allows a lot of game logic to be processed on the client's end, without having to relay too much data between players.

In this solution, it was decided to keep the amount of transferred data to a minimum where possible. This comes from the fact that the application was developed for mobile devices, so minimizing the traffic consumption for devices that may rely on mobile connection is reasonable. Because of that, it was decided to exchange the position of the essential game elements (ie all types of cogs and victory buttons) between the players and only share the information about the status update notification when necessary. For example, if a player were to move a cog to another spot, other players would (besides the changing position and rotation synchronization) receive a status update about the cog being attached, which they would then locally process and update local data. The data is not verified across all clients.

This approach is not ideal, as it allows data to be modified locally. However, as mentioned earlier, due to the specification that the application could be used over the mobile network, it was decided to accept such risks. Implementation of a cross-referencing module that would

constantly ensure that data stays the same among all connected clients was considered to be too traffic-intensive to implement.

The final stage of the development was testing the MVP. In order to do that, it was decided to perform several closed-access user testing sessions. In total, it was possible to organize 3 of such sessions with a total of 8 participants. There were 2 sessions with 3 testers and one with 2. Each session lasted over an hour and there were at least 2 games held in each session.

The testing sessions showed that the visual aspect of the application was on a level that is not yet acceptable for a full product. However, it was sufficient for the participants to be able to use the application.

The testing sessions uncovered several issues related to the game mechanics, which were not discovered during the internal testing sessions by the development team. Most of these issues were fixed between the sessions, resulting in the participants of the last session to give mostly positive feedback regarding the product.

As it is, the current state of the application that was developed for this thesis is considered to reach the status of an MVP. Because of that, the initial goal of the thesis is considered to be achieved. However, future work is also planned in order to successfully deliver the project to the open market.

I would like to thank Raimond-Hendrik Tunnel, my supervisor, who consulted me in various questions regarding both the development of the application as well as formulating the thesis. Thanks to Vladyslav Kupriienko and Aap Vare, students of the University of Tartu, who provided valuable feedback on my thesis. Special thanks to Tarvo Metspalu and his company (Edulab OÜ) for providing me with an opportunity to work on this project. Additional thanks to him and Madis Kaspar Nigol who helped with the development of the application. Finally, I would like to thank all the people from *APT GameGenerator* and Computer Graphics and Virtual Reality Lab that participated in the testing of *Cogbug*.

References

[1] Sinicki, A. (2018). Best Android developer tools for getting started or levelling up your dev skills. [online] Android Authority. Available at: https://www.androidauthority.com/best-android-developer-tools-671650/ [Accessed 9 Nov. 2019].

[2] Lahoti, S. (2018). Game Engine Wars: Unity vs Unreal Engine | Packt Hub. [online] Packt Hub. Available at: https://hub.packtpub.com/game-engine-wars-unity-vs-unreal-engine/ [Accessed 9 Nov. 2019].

[3] Docs.unity3d.com. (2019). Unity - Manual: Getting started with Android development. [online] Available at: https://docs.unity3d.com/Manual/android-GettingStarted.html [Accessed 10 Nov. 2019].

[4] Viennot N., Garcia E., Nieh J. A Measurement Study of Google Play. ACM Digital Library. 2014, pages 8–10. https://doi.org/10.1145/2637364.2592003 [Accessed 10 Nov. 2019].

[5] Medium. (2017). How to build mobile games with people in mind. [online] Available at: https://medium.com/googleplaydev/how-to-build-mobile-games-with-people-in-mind-cdc480967fcc [Accessed 10 Nov. 2019].

[6] Nystrom, R. (2014). Game programming patterns. Genever Benning.

[7] Tekinbaş, K. and Zimmerman, E. (2004). Rules of play. Cambridge, Mass.: MIT Press.

[8] SWINK, S. (2017). GAME FEEL. 1st ed. CRC Press.

[9] McShaffry, M. and Graham, D. (2013). Game coding complete. 4th ed. Boston, Mass.: Course Technology PTR.

[10] Choi, D. and Kim, J., 2004. Why people continue to play online games: In search of critical design factors to increase customer loyalty to online contents. CyberPsychology & behavior, 7(1), pp.11-24.

[11] Schell, J. (2019). The Art of Game Design: A book of lenses. AK Peters/CRC Press.

[12] Fang, Y., Chen, K. and Huang, Y. (2016). Emotional reactions of different interface formats: Comparing digital and traditional board games. Advances in Mechanical Engineering, [online] 8(3), p.168781401664190. Available at: https://journals.sagepub.com/doi/pdf/10.1177/1687814016641902 [Accessed 9 Nov. 2019].

[13] Cossu, S. (n.d.). Game Development with GameMaker Studio 2. Apress, Berkeley, CA. Available at: https://link.springer.com/book/10.1007%2F978-1-4842-5010-5 [Accessed 10 Nov. 2019].

[14] Buckland, M. (2010). Programming Game AI by Example. Burlington: Jones & Bartlett Learning, LLC. Available at: https://books.google.ee/books?hl=en&lr=&id=gDLpyWtFacYC&oi=fnd&pg=PR13&dq=game+ai&ots=v1xjbSWn0e&sig=-RzuLuxE69Z4r80gkCc_HkOVdzk&redir_esc=y#v=onepage&q=game%20ai&f=false [Accessed 9 Nov. 2019].

[15] Bakkes, S., Spronck, P. and Van den Herik, J. (2009). Opponent modelling for case-based adaptive game AI. Available at: https://www.sciencedirect.com/science/article/abs/pii/S1875952109000044 [Accessed 10 Nov. 2019].

[16] Šmíd, Antonín. "Comparison of Unity and Unreal Engine." Bachelor Thesis, DCGI/Faculty of Electrical Engineering, Praga, República Checa (2017).

[17] Kaleelezhicathu, R. "History of Internet Pricing." HUT, Finland (2003).

[18] Computerhistory.org. (n.d.). Computers | Timeline of Computer History | Computer History Museum. [online] Available at: https://www.computerhistory.org/timeline/computers/ [Accessed 9 Nov. 2019].

[19] Chapple, Craig. "Top Mobile Games By Worldwide Revenue For September 2019". Sensortower.Com, 2019, https://sensortower.com/blog/top-mobile-games-by-worldwide-revenue-september-2019. [Accessed 9 Nov 2019].

[20]Katzenbach, C., Herweg, S. and Van Roessel, L. (2016). [online] Hiig.de. Available at: https://www.hiig.de/wp-content/uploads/2016/01/4802-18724-1-PB.pdf [Accessed 10 Nov. 2019].

[21] Sinicki, A. (2018). Which is better? Unity vs Unreal Engine for Android game development. [online] Android Authority. Available at: https://www.androidauthority.com/unity-vs-unreal-engine-android-game-development-842045 [Accessed 10 Nov. 2019].

[22] Google Docs. 2019. Unity Networking Frameworks - Feature List. [online] Available at: https://docs.google.com/document/d/1ufcl6_ylbzthfmzzZ7HPiwDLR0sr1N3UAunUG9rnisY [Accessed 20 March 2020].

Appendix

I. Glossary

Minimum Viable Product (MVP) – a product with just enough features to satisfy early customers and provide feedback for future product development.

Multiplayer – a game mode where two or more human players can play the same game in the same session using multiple devices. The connection between the devices is established over a network.

Pass and Play – a game mode where two or more human players can play the same game in the same session using a single device. When one player finishes the turn, they confirm it on the device and then physically pass it to the next player.

Singleplayer – a game mode where only one human player is present. The player either has a set of goals that are meant to be achieved alone or has a set of artificial opponents that simulate other players and compete with each other and the human player to achieve a goal or a set of goals.

Player versus Player (PvP) – a game mode where two or more human players compete in order to achieve a goal or a set of goals. Victory can only be achieved by one team that can consist of a single player or a group of players. Sometimes, teams have means of preventing other teams from achieving the goals.

Cooperative (coop) – a game mode where two or more human players work together to achieve a common goal or a set of goals using teamwork.

Two-and-a-half-dimensional (2.5D) perspective – a game perspective when either the 2D images are designed in such a way to simulate a 3D perspective object. The opposite approach is used when a video game that is three-dimensional is limiting the perspective to two dimensions.

Object pooling – a pattern used in programming when a set of objects is initialized before they are actually needed. These objects are not removed and are kept until needed. When not needed, the objects are returned to the "pool" until needed again. This pattern is an alternative

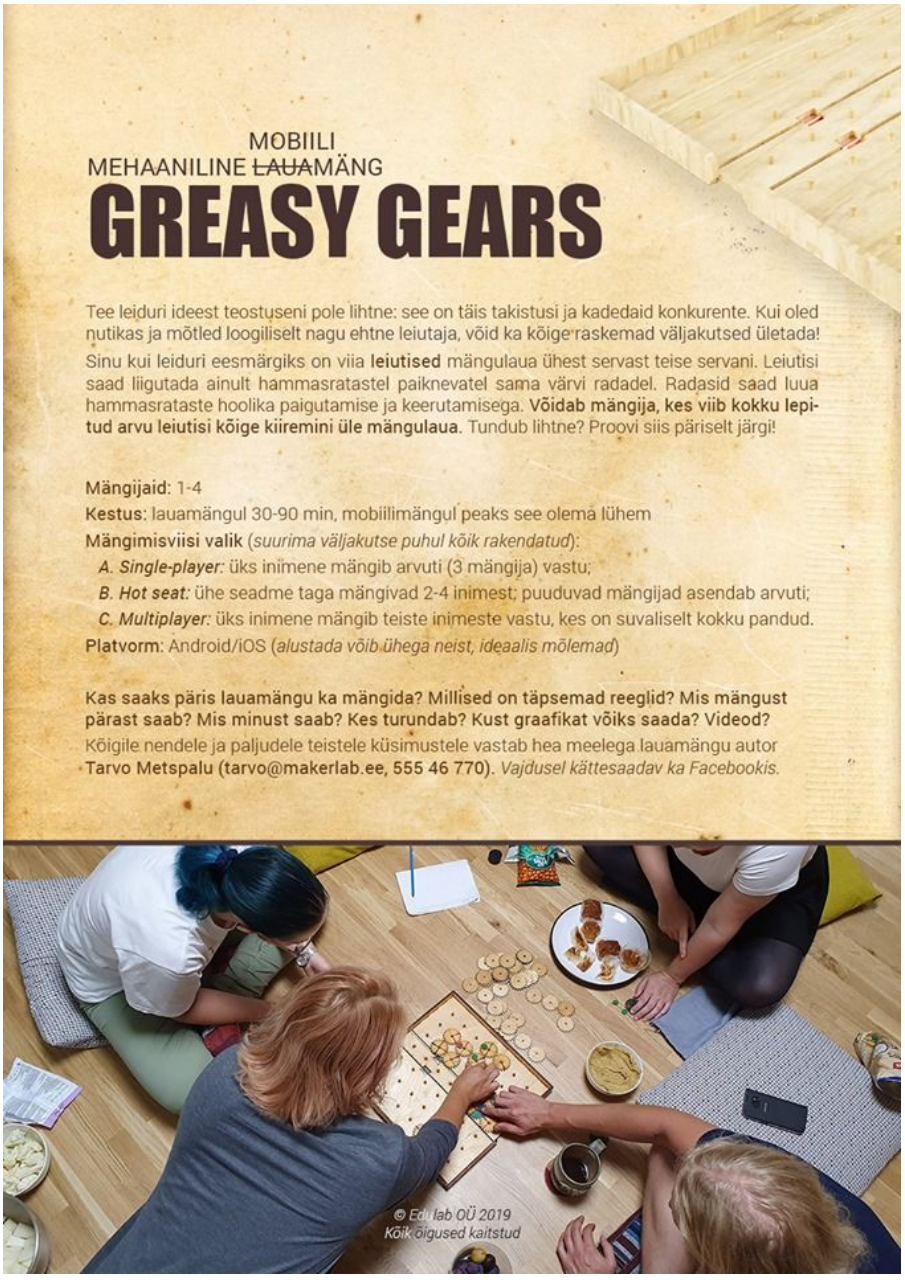
to creating and destroying objects on demand. Object pooling is a useful pattern when dealing with situations where multiple objects need to be used often, but are otherwise not needed.

Type object decoupling pattern – an object-oriented programming (OOP) pattern. It's main goal is to group objects by a set of features. These features are common for all object types of the group. These features are brought out into a separate class that is used or overridden by each particular object instance of the group.

Client-to-server architecture – a type of network architecture where each machine connected to the network is either a server or a client. In order to interact with other clients, every message is always sent through a server, which then redirects it to the target client or clients. This is the opposite of Peer-to-Peer architecture where there is no explicit role differentiation between a client and a server.

Master client – a type of client identification used by Photon PUN 2. A master client is chosen automatically when this client creates a new room. Master client functions in the same way as every other client. Photon PUN allows identifying the master client. This makes it possible to make certain processes or procedure calls go through a particular user. By default, if a master client leaves the game room, a new master client is chosen.

II. Initial Requirements



MOBIILI
MEHAANILINE ~~LAUA~~MÄNG

GREASY GEARS

Tee leiduri ideest teostuseni pole lihtne: see on täis takistusi ja kadedaid konkurente. Kui oled nutikas ja mõtled loogiliselt nagu ehtne leiutaja, võid ka kõige raskemad väljakutsed ületada!

Sinu kui leiduri eesmärgiks on viia **leiutised** mängulaua ühest servast teise servani. Leiutisi saad liigutada ainult hammasratastel paiknevatel sama värvi radadel. Radasid saad luua hammasrataste hoolika paigutamise ja keerutamisega. **Võidab mängija, kes viib kokku lepitud arvu leiutisi kõige kiiremini üle mängulaua**. Tundub lihtne? Proovi siis päriselt järgi!

Mängijaid: 1-4
Kestus: lauamängul 30-90 min, mobiilimängul peaks see olema lühem
Mängimisviisi valik (*suurima väljakutse puhul kõik rakendatud*):
A. Single-player: üks inimene mängib arvuti (3 mängija) vastu;
B. Hot seat: ühe seadme taga mängivad 2-4 inimest; puuduvad mängijad asendab arvuti;
C. Multiplayer: üks inimene mängib teiste inimeste vastu, kes on suvaliselt kokku pandud.
Platvorm: Android/iOS (*alustada võib ühega neist, ideaalis mõlemad*)

Kas saaks päris lauamängu ka mängida? Millised on täpsemad reeglid? Mis mängust pärast saab? Mis minust saab? Kes turundab? Kust graafikat võiks saada? Videod?
Kõigile nendele ja paljudele teistele küsimustele vastab hea meelega lauamängu autor **Tarvo Metspalu (tarvo@makerlab.ee, 555 46 770).** *Vajdusel kättesaadav ka Facebookis.*

III. Cogbug Rules

UNIQUE FIRST EDITION

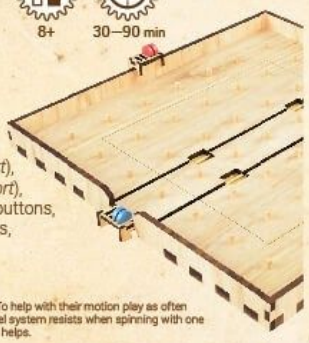
GROBS

RULES FOR THE MECHANICAL GAME

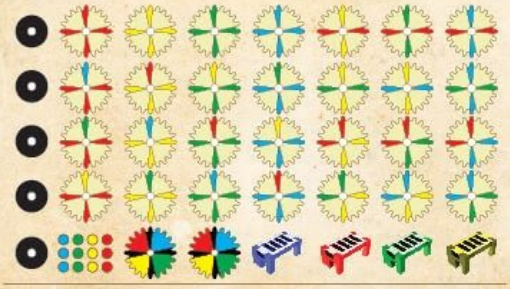
2 to 4 | 8+ | 30—90 min

1 - PARTS

- 7x7 gameboard,
- 28 different cogwheels (*cogs in short*),
- 2 Joker cogwheels (*Jokercogs in short*),
- 3 blue, red, green and yellow game buttons,
- 4 colored game button base stations,
- 5 black blockers,
- + spare game buttons



Cogwheels abrase and start spinning better over time. To help with their motion play as often as you can or simply spin the cogwheels. If the cogwheel system resists when spinning with one hand use bothhands. Regular mending with flax oil also helps.



2 - BACKGROUND, PRINCIPLE & WINNING

The world is full of clever inventors, who would do anything to be the best. But the road from an idea to implementation isn't easy: it's full of obstacles and very envious competitors. By being smart and thinking logically like a real inventor You can conquer even tough challenges and win the race!

Your goal is to take Your **game buttons (representing Your inventions)** from your side of the table to the opposite side. You can only move these buttons on the same coloured trail as is the colour of your chosen button base. The trails are created by adjusting and turning the cogwheels. **The player who takes the agreed amount of game buttons across the gameboard the fastest wins!**

3 - LET'S BEGIN!

Adjust the gameboard so that each player faces an edge. Place the cogs facing down or into the big bag next to the gameboard. Each player chooses a color by selecting a game button base station.

Agree on how many **inventions (game buttons)** one must take across the gameboard - the length of the game depends on it. Place the agreed amount of inventions to Your game button bases. You can put the base either on the edge of the gameboard, near it or simply in front of you.

Decide whether to play on the whole **7x7** game field or on the **5x5** midfield. When playing on midfield, 3 blockers are in use, one cogwheel spot must always stay empty, and game buttons are taken over the field one at a time.

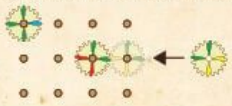
If there are less than 4 players, agree whether everybody can move on the unused trails or not. For example two players with red and yellow colours can decide that both can use green trails to move but blue if off-limits for both players. It is reasonable for beginners to stick to their colours and not use other coloured trails at all.

4 - NOW WHAT?

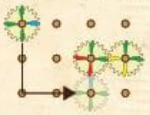
Each player gets to make up to 3 different actions during their turn. The order of the actions and whether to use them all is up to the player: **place, spin, move!**

1x PLACE!

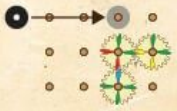
a) a cog onto the gameboard spot by blindly taking it from the bag or **the table**



b) a cog without buttons to a new location on the gameboard or adjust it on place



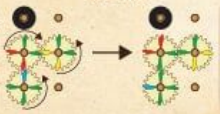
c) a blocker onto the gameboard to a new empty location or on top of a empty cog



- Make sure to always place cogwheels so that the trails align, including Jokers!
- Spinning the cog while moving it to a new location does not count as a separate action!
- Cogs & blockers that are placed on the field stay on the field. Blockers block placing cogs and moving buttons.

1x SPIN!

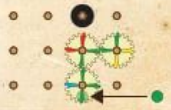
- **You can freely choose which cogwheel system aka connected cogs You spin. You can only spin one system during Your turn!**



- **On a beginner level the angle of turn is unlimited. 90º on an advanced level.**
- Turn calmly without disrupting the layout. If needed, use both hands.

1x MOVE!

a) move a button to the first row when Your colour on the cog is towards You



b) a button along Your trail only if Your button was already on the gameboard

- **Relocating a button between Your colours on the same cog is not an action!**
- When you make it across the gameboard and Your trail is directed toward Your opponent, move the invention to their base. **This is an action!**
- **By the end of Your turn, only different coloured buttons may be left on a cog!**
- The button placed on the gameboard can travel as far as your coloured trail goes.
- Jokers allow to move in all directions to all.

5 - ADDITIONAL RULES FOR YOUR NEXT GAMES

Making first steps is always the most difficult part in both life and fulfilling ideas. This much-heard saying confirms that learning a new board game is also difficult in the beginning and gets easier over time. It is important to our development process that a repeating game would offer fresh challenges for the brain. Here are rules that you can choose to try out during next games.

ADVANCED RULES

1. "Race"-like game
Every player start from one side of the gameboard and have to make it across. Place 2 invention bases to the start and 2 to the finish. Every player has up to 2 inventions to use.

2. Freeware
Before the game, place cogs on the table facing upwards. A better favored beginning soon turns out to be agonising when Your opponents start quickly building their trails.

3. Complete mind block
Place the blockers on the gameboard before the game. Symmetry is recommended. Do not remove them during the game.

4. One idea at a time
Game buttons can be moved over the field one at a time. In other words, every player may have one game button on the gameboard.

5. Limited perspective
During Your turn You can only turn the cogwheels 90 degrees.

6. Graveyard
Instead of the placing action You can remove a cog off the gameboard into a pile. This would be the one and only action during Your turn. The "Graveyard" only fits up to 12 cogs - these can no longer be used during this game.

7. Last minute idea aka 2+1 or 3+1 game
After successfully taken 2 or 3 game buttons across the gameboard, move just one more button across to win. It's well known that last-minute ideas are always the best ones!

8. Determinism aka destiny
The order of actions are fixed and not freely chosen. Place, turn and then move!

9. Spionage warfare
You must get Your game buttons across the middle cog of the 1st row of all other players. This means 2 or 3 buttons for 3 or 4 players.

10. Masterbuilder
You can place up to 2 cogs but can not take any more actions during a turn.

6 - FAQ

1. When I already have taken a cogwheel into my hand, can I spin the other cogs on the gameboard before placing it? Nope.

2. Can I retrieve my button from the gameboard during my turn? No. Also in life we may not be able to take back our actions.

When You face other questions, just turn to your co-players and make agreements. This way You will end up with unique rules. Use this to remind Yourself that there might even be an inventive spirit in You! When some things are still left muddy or seem complicated, please contact the author of the game who is also very thankful for all Your shared thoughts. Enjoy the game!

Author: Tarvo Metspalu (tarvo@makerlab.ee)

CAUTION!
The game is not suitable for kids under the age of 3 as it may cause a choking hazard. Also there is a slight danger of getting a splinter!

IV. Meetings

A series of meetings were held in order to clarify and adjust the flow of the development process.

Meeting 1

This meeting took place on 25 October 2019. The goals of this meeting were:

- Agree on communication means.
- Discuss the Cogbug physical tabletop version rules.
- Settle legal arrangements.

After discussing the goals and related questions, the following results were achieved:

- The ruleset for the physical version of the game has been analysed and discussed and agreed upon.
- Victory conditions have been discussed and agreed upon.
- Cog placement rules have been discussed and agreed upon.
- Physical meeting was scheduled for a live demonstration of a game session.

Meeting 2

This meeting took place on 12 December 2019. The goals of this meeting were:

- Discuss the mobile game details.

After discussing the goals and related questions, the following results were achieved:

- Scheduled the initial mechanics demo to be delivered by 20 January 2020.
- Made adjustments to the player camera angle.

Meeting 3

This meeting took place on 20 January 2020. The goals of this meeting were:

- Discuss cooperation with a new team member suggested by the Customer representative.
- Discuss the first demo.

- Discuss further steps in development.

After discussing the goals and related questions, the following results were achieved:

- Local multiplayer is a priority.
- Demo bugs discovered.
- New team member tasks clarified.
- Regular schedule for discussing project progress established.
- Expanded team coordination, communication issues resolved.
- Discussed the cog drag and drop mechanics changes.

V. Attachments

README.txt file contains a reference list to all documents similar to this section.

Cogbug usability testing feedback form: cogbug_form_questions.pdf

Cogbug usability testing feedback results: cogbug_form_responses.csv

Cogbug usability testing sessions:

- Session 1:
    - Online: https://youtu.be/9z8GWEuNdmA
- Session 2, part 1:
    - Online: https://youtu.be/CcMXNeWnlXw
- Session 2, part 2:
    - Online: https://youtu.be/i-CioJXsKHE
- Session 3:
    - Online: https://youtu.be/Jk43QLTRkvg

Cogbug MVP application version:

- Android version: cogbug_mvp.apk
- iOS version: iOS platform is more secure than Android, request version with installation instructions via 01.stass@gmail.com

Cogbug GitHub repository: project is in a private repository https://github.com/Untar1/GG, request access via 01.stass@gmail.com

License

Non-exclusive licence to reproduce thesis and make thesis public

I, Stanislav Belogrivov,

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

Cogbug – A Mobile Tabletop Game,

(*title of thesis*)

supervised by Raimond-Hendrik Tunnel.

(*supervisor's name*)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Stanislav Belogrivov

05/15/2020