Ville Kähärä

IMPLEMENTATION OF INTERACTIVE APPLICATION FOR MARETARIUM AQUARIUM

Bachelor's thesis
Information Technology / Game Programming

2019

Kaakkois-Suomen
ammattikorkeakoulu

Tekijä/Tekijät	Tutkinto	Aika
Ville Kähärä	Insinööri (AMK)	Toukokuu 2019
Opinnäytetyön nimi Interaktiivisen mobiilisovelluksen toteutus		39 sivua
Toimeksiantaja Digiverstas - XAMK		
Ohjaaja Niina Mässeli		

Tiivistelmä

Opinnäytetyön aiheena oli innovoida sekä tehdä tuote, jolla olisi mahdollista kasvattaa kävijämääriä akvaariotalo Maretariumille. Tuotteen tilaajana toimi Digiverstas. Digiverstas yhdistää Kaakkois-Suomen ammattikorkeakoulun opiskelijoita paikallisten yrityksien kanssa. Digiverstas tarjoaa yrityksille mahdollisuuden löytää uusia liiketoimintaratkaisuja sekä ideoita, jotka vauhdittavat yrityksen kasvua.

Opinnäytetyö tuo esille Maretarium-pelin kehitysprosessin sekä millaisia ominaisuuksia tehty tuote sisältää. Maretarium-peli tarjoaa erilaisen oppimistavan alakouluikäisille koululaisille, sillä heillä on mahdollisuus opetella uutta informaatiota, tietovisojen sekä minipelien avulla. Koululaiset voivat käyttää tätä peliä osana heidän opetussuunnitelmansa mukaisia biologian kurssejaan.

Tämä opinnäytetyö kertoo tarkasti pelin implementaationprosessin, samalla kuvaillen teknisesti mitä interaktiivisen mobiilipelisovelluksen toteutus vaatii. Implementaatioprosessi sisältää tekniikoita ja ominaispiirteitä, joita peli hyödyntää. Peli on toteutettu käyttäen Unity-pelimoottoria sen prototyyppikehityksen nopeuden ansiosta. Opinnäytetyön tuotoksen tuloksena julkaistiin peli, joka on aktiivisessa käytössä akvaariotalo Maretariumissa.

Asiasanat

mobiili, interaktiivinen sovellus, peli, Unity, Firebase

South-Eastern Finland University of Applied Sciences

Author (authors)	Degree	Time
Ville Kähärä	Bachelor of Engineering	May 2019

Thesis title

Implementation of interactive application for Maretarium aquarium

39 pages

Commissioned by

Digiverstas - XAMK

Supervisor

Niina Mässeli

Abstract

The main objective in this thesis was to innovate and implement a product to increase the number of visitors in Maretarium aquarium house. The thesis was commissioned by Digiverstas - Digital Innovation Ecosystem project which provides a connection between the students of Southern-Eastern University of Applied Sciences and local companies. Via Digiverstas, the companies are able to find new business solutions and ideas that correspond to the company growth and development.

This thesis provides insight into the development process and features of the Maretarium game. The Maretarium game provides a different learning method for elementary school students. The students can utilize this game as part of their biology curriculum. The game can be used as a part of class session by way of group activities.

This thesis describes the implementation process in detail and also examines technical aspects of the making of an interactive game application requires. The implementation process contains the techniques and characteristics that were used to develop the game. The game was developed in the Unity game engine and its features were used for rapid prototype development. As a result of this thesis study, the game was successfully created and is in use at the Maretarium aquarium house.

Keywords

mobile, interactive application, game, Unity, Firebase

TABLE OF CONTENTS

1 INTRODUCTION

The main objective in this thesis was to innovate and implement a product to increase the number of visitors in Maretarium aquarium house. The thesis was commissioned by Digiverstas - Digital Innovation Ecosystem project which provides a connection between the students of Southern-Eastern University of Applied Sciences and local companies. Via Digiverstas, the companies are able to find new business solutions and ideas that correspond to the company growth and development (Digiverstas 2019).

This thesis provides insight into the development process and features of the Maretarium game. The Maretarium game provides a different learning method for elementary school students. The students can utilize this game as part of their biology curriculum. The game can be used as a part of class session by way of group activities. Group activities improve learning experience and are an effective learning tool because students can improve their communal skills and learn self-expression better in groups (Perusopetuksen opetussuunnitelman perusteet 2014, 21). The Maretarium game targets elementary students as requested by Maretarium aquarium house staff.

Metrics that the game provides include for example player retention metrics, Game usage statistics and the trivia game specific statistics providing insights to the information on the individual fish tanks. By using the product, Maretarium staff can monitor user activity and observe their interests in specific fish tanks or for example knowledge about different species of fish. These insights boost business strategies and allow a multitude of marketing opportunities from in-game events to real world competitions.

2 DESIGN

The potential that a particular game is able to achieve in the development and design process immerses the players into the game and reveals the functionality of the game concept. The design process can vary on per game basis and reflects the game and its features. (Kramarzewski & De Nucci 2018, 9.) The Maretarium game production model was a mix of the traditional software development model and the Agile software design principles. The

Agile model allowed the testing, implementation and design phases of game to be developed in conjunction with Digiverstas and Maretarium staff.

During the implementation phase, a modern software design principle, Agile, was used. This allowed the game to develop and adapt to the changes based on the client's needs. The game was designed to adapt during the development in such way that the game satisfied the client. It was possible to change the game features as requested and as concluded during testing phases due to the applied software design principles. These included techniques from Agile space such as Kanban-boards and active weekly meetings. By applying a Kanban-board technique and organising regular meetings during the development process, the game developed with a steady rate and no major redesigns were needed for the game implementation. The scope of the game corresponded to what was defined in the project specification plan.

The daily work during the Maretarium game development process itself focused on milestones. The milestone workflow allowed the game to be developed as defined in the game design document and the game features were testable in different states of development. By reflecting on the milestones on a regular basis, it was possible to keep the game concept in check and up to date with respect to the game design document.

2.1 Traditional software development model

The Waterfall model is a simple and idealistic software development model. However, this software model is outdated as goals can change and affect the game in significant ways. The Waterfall model proceeds to the next phase when the previous is fully completed, and this introduce more challenges than might be expected. More appropriate development models for games are Agile and Scrum. The Agile model is described below, but to understand the Maretarium game and modern game development models, the most traditional model is inspected first. In this chapter, Pal (2018) introduces the Waterfall software model by following seven basic principles.

The first phase is to study the problem and determine solutions to solve the problem. The solutions are analysed in terms of benefits and drawbacks in order to recognize the problem. If the problem can be solved by the development team while keeping financial and technical aspects in mind, the development of the game can be managed efficiently. The game idea is formed during this stage and presented to the director board. The idea can be modified during the game life cycle and then an iterative Waterfall model is to be used because the traditional Waterfall modes does not allow for iteration. The iterative Waterfall model allows the game to go through the traditional Waterfall model phases repeatedly until requirements are met.

The second phase is to gather requirements concerning the game's development process and analyse them. The purpose is to form a consistent product from the aforementioned solutions and process them into a requirement specification document which is also called a game design document, and it is regarded as a good practice to gather all game related specifications in this document. (Ryan T. 1999.) This phase may have some overlapping characteristics with the next phase, or they can be implemented independently. The requirements can be divided into technical aspects and implementation techniques as well as guidelines for the design phase. As Pal (2018) states, the second phase describes the game requirements. These requirements consist of the device specifications that the game needs to run properly, the development funding, the scope of the game and the game idea explained in writing. Also. The customer needs must be defined, and it must be determined if the game is developed for a particular company. In this study, the Maretarium game has been implemented in accordance with the requests from the Maretarium aquarium house staff.

The design phase includes converting the game design document into implementation structure that can be developed into a game. The game engine is chosen usually in this phase as the game requirements are already known. As stated in the chapter above, this phase and the collection of requirements specification can overlap with the game development phase. This ensures that it is possible to implement the requirements suitable for the chosen game engine.

The implementation phase is to implement the game. Implementation phase requires cooperation as the game development progresses further. By following a game design document cooperation during the implementation phase can be more fluent and the development process can advance further. The implementation is based on the game design document. As changes occur, they are documented in the design document. The Maretarium game implementation phase is documented in Chapter 4.

As Pal (2018) states, the testing phase entails alpha and beta testing as well as acceptance testing. Some modern games also involve a soft launch phase, where a few countries and players can participate. In the basic Waterfall model, the alpha, beta and acceptance testing phases are in use. In alpha testing, the development team identifies the most critical bugs and forwards them to the beta testers. In the beta testing, a group of people play the game and report critical bugs that affect the game. The acceptance testing determines the adoption rate among the customers after the game has been delivered to the market. The testing phase of the Maretarium game is described later in Chapter 6.

The last and one of the most important phases is maintenance. As Pal (2018) states, the traditional maintenance phase consists of three modes. The corrective maintenance mode fixes errors and bugs not found during the testing phase. The perfective maintenance mode improves and modifies the game based on the needs of players. The third one is an adaptive maintenance mode where the game is ported to a new platform. However, by today’s industry standards, maintenance phase specifies a game as a service.

The phases of the traditional Waterfall model have advantages and disadvantages. The most beneficial advantage is that the model as a whole is simple and easy to understand. It includes clearly defined milestones that are well documented. While this model is suitable with small projects, larger projects might struggle with following it.

The drawbacks include challenges resulting from the absence of overlapping phases and difficulties in adapting features during the development as requirements change. By not setting a feedback loop, operators assume that

errors will not be made during the development phase and the developers will be successful in their tasks.

The basic waterfall model is suitable for small projects and provides clear milestones, this slightly outdated model is not used in its entirety as a guide but rather as a foundation for other game development models, as Pal (2018) states, giving to a basic understanding of these models. A modern version of the agile software design model is used in the Maretarium game. The Agile software design model allows the Maretarium game to better adapt to the players' needs and desires.

2.2 The Maretarium game design model

The Agile model was designed to be adaptable and suitable for the ever-changing goals and features of a game. As an example, in the Maretarium game, the mini-games have goals and scopes that vary from each other. Thus, a more agile model was needed. In the mini-games, for example, the scopes require such and also varying tasks that not one software model accommodate all of the mini-games. The Agile model that allow for changes during the implementation and testing phases. Architecting design and requirements are merged. This is illustrated by the variety of games developed and designed with the model. (Principles behind the Agile Manifesto. 2001.)

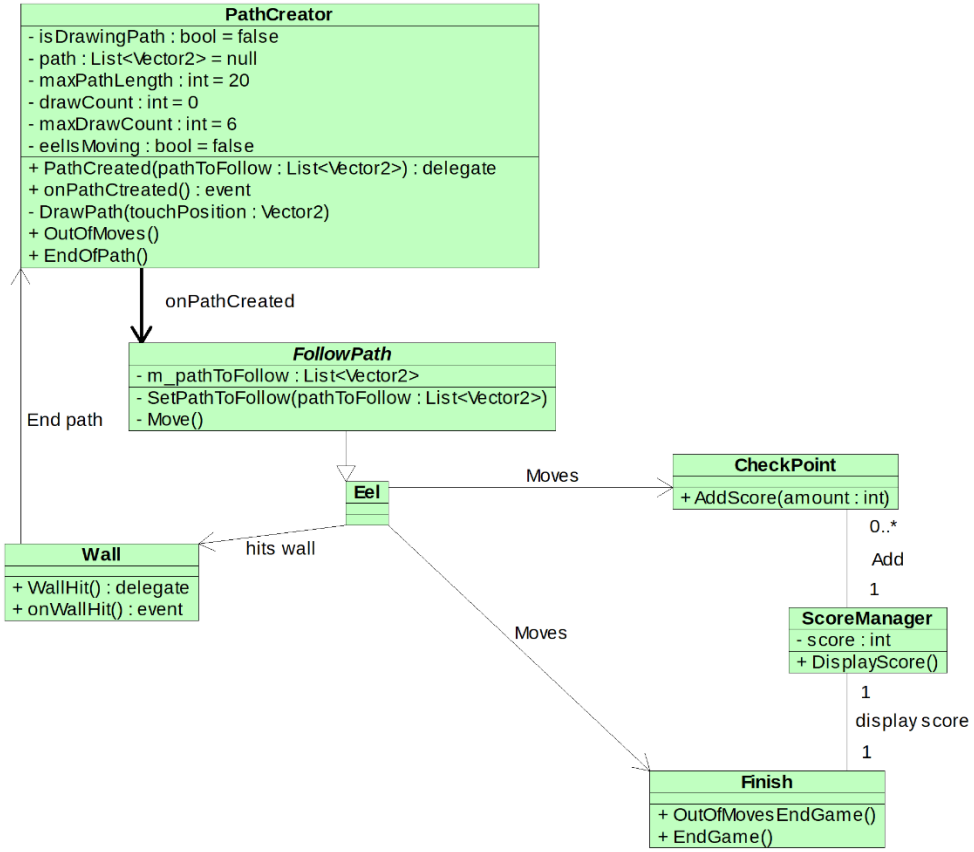


Figure 1: Eel game class diagram

The most common design principles allow the game to become more testable and, most importantly, satisfying to the players. As seen in Figure 1, even a simple idea can prove complex. Manageability means that new features can be added later in the game's lifecycle. The managing aspect aims to ensure a testable product that minimises the number of obstacles faced during game development. This lowers the development costs of the game, which means that development funds can be distributed according to the needs of the development company.

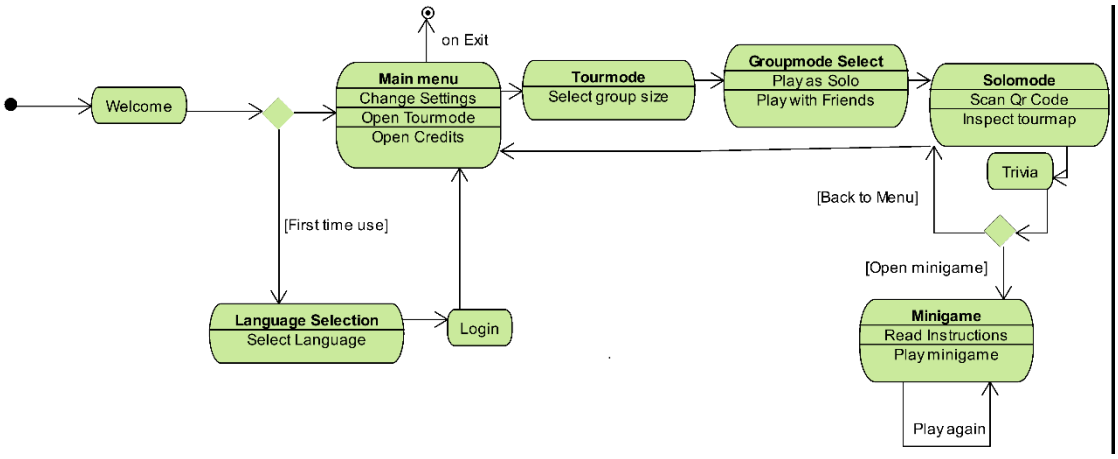


Figure 2. The Maretarium game flowchart, single-player experience.

In Figure 2, The Maretarium game flowchart displays transitions between screens and the flow of single-player experience. The game was originally designed to be playable in groups but due to time constraints, the development team focused on completing the core game loop of single-player experience. This change in the designed core game loop emphasises the group play between players in a real-world environment. The change brought positive outcomes during the testing period, and multiplayer features can be added at a later time.

3 TOOLS AND TECHNOLOGIES

Application development tools range from text editing software and integrated development environments to full featured game engines. Software for specific tasks aid in corresponding development. For example, 2D artists use drawing and animation software in their daily work. Game programmers use game engines in conjunction with integrated environment software. Software used by development team has a wide spectrum of application and is one of the key parts of the development process. For example, Kramarzewski & De Nucci (2018, 11) describe that “each Super cell development team is called a cell which is a small team consisting of a producer, a few artists and coders, at least one tester, a generalist game designer and potentially a monetization or live operation-oriented person”.

3.1 Unity Engine

Unity is a cross-platform and all-in-one editor featuring tools for rapid development. It includes user-friendly tools appropriate for developers and artists, and support for both 2D- and 3D-functionality specific to users’ needs. (Unity Features 2018.)

Unity game engine servers as a great starting point for the development. Quick concept prototyping is one of its advantages over other game engines that are available to general public. The process of setting up a project in Unity game engine can be initiated even by beginners. The project creation process is streamlined and easy to use because the process is guided with clear instructions. Every team member can focus on their specialities and start

prototyping a game right from the beginning. (Unity Game engines – How do they work 2018.)

Features of Unity are well documented and available online and also offline if downloaded during the engine installation process. Documentation is divided into categories based on different topics. The knowledge level required to understand the topics varies from beginner to advanced and professional. (Unity User Manual 2018.)

The documentation topics range from installing the engine on multiple platforms to creating the whole game in a live session. Generally, each topic consists of a heading, description, sub-heading, methods and variables to use with the topic and an example of how to use the topic in a game. These sections are labelled in a logical way and information can be found quickly by looking at the sub-heading of the topic.

Also, high-quality video games can be developed with the help of Unity game engine. These include but are not limited to; Hearthstone, Cities Skylines, Cuphead and Monument Valley. (Palmer & Williamson 2018.) In a professional environment, Unity is a proficient developing tool for a product. The aforementioned games vary in their scope. This proves that Unity game engine is able to provide high-quality tools and features for game developers to develop games in a large range of genres. (Unity Features 2018.)

The Maretarium game use Unity engine 2D features extensively to implement the game. Game included mini-games that features tools such as a 2D grid system, Unity UI event system and Unity 2D art tools. For example, the 2D grid system is used to place elements and obstacles in Eel mini-game and therefore faster level creation. Faster level creation made the creation of modular mini-games possible in a short time span.

3.2 Google Firebase technology

Firebase is an all-inclusive development platform for mobile devices. With Firebase, the developer can build better apps by utilizing for example authentication, Realtime Database, and Hosting services. The quality of an

app is improved by monitoring and testing its performance on a mobile device. Sending in-app notifications, predicting app growth and analysing app usage results can expand a company's business. (Google Firebase 2018.)

Firebase provides analytic tools for game developers including for example the ability to monitor how much app content is being purchased and the number of hours players have been playing the game. Sharing game content and game notifications enhance the player experience. The game can be tweaked from the cloud. (Google Firebase 2018.) These topics are discussed in further detail in Chapter 5.

The Maretarium game uses Google's Firebase Realtime Database technology to store question data. The Firebase Realtime Database allow the Maretarium game to scale according to the player base while providing Trivia dashboard ability to manage trivia data. By storing trivia data in an online database, data usage can be monitored and analysed accordingly with set parameters and usage metrics.

3.3 Vue.js

The online portal for managing trivia data contains a dashboard made with the Vue.js web framework. Vue.js allows the creation of user interfaces with a progressive JavaScript framework. Vue.js is scalable between the library and framework depending on the project. It is designed to be incrementally adoptable from the ground up. (Vue.js 2018.) Trivia dashboard utilized Vue.js progressive framework owing to its ease of use in production, extension library of material style and the responsive design choices that Vue.js offered from the start. Material style components allowed the dashboard to be designed and look fashionable while maintaining its core functionality. (Vuetify 2018.)

The development speed of Trivia dashboard is improved by using Vue.js as a website framework and Vuetify as material component framework. Usage of a progressive web framework allowed to select a particular development focus. Development was focused more on the content than the layout design and the technical aspects of the development process.

Vue.js was chosen because it has the capacity and features needed for the trivia dashboard. Vue.js as a framework is very adaptive. The single file components provide robust building blocks from which the developer can construct a website on demand. During the development process this allows a developer to focus more on the visual aspect of website. (Vue.js 2018.)

The single file components contain sections for a template, functionality and styling. The template uses HTML5 syntax for layout components. Functionality section uses modern versions of JavaScript language and ECMAScript standard to script commands and functions. This allows for a more modern and fast typing syntax; therefore, efficiency improvements are immediate. The ECMAScript standard is also more human-readable. (ECMAScript Language Specification 2015.)

The styling feature uses CSS version 3 to implement component styling. Component styling allows styles to be applied to specific components. The styling of components then becomes more extensive and flexible. (Copes 2018.)

Vue.js single file components can embed other single file components. The building blocks of the website can be divided into smaller easily manageable sections. The Trivia dashboard and the question data are discussed in further detail in the trivia dashboard section of Chapter 4.

3.4 Version control

Tracking files during the game development process can be a laborious task. Large files can be hard to debug, modify or share between the game versions. Game and its development can be kept under monitoring with a version control system (VCS). With a VCS, the developer can go back to previous files, merge changes, and even check the progress made during the development cycle. This helps the project in a significant way because the problems faced during the development phase are identified and can be tracked to the source of the problem. The version control is most commonly used in programming centric tasks. Version controlling can be applied to any file and is useful in all forms of digital work. (Chacon & Straub 2014.)

The Unity game engine has a built-in version control system called Unity Collaborate. The personal licence of Unity allows up to three developers to use version control features with minimal required setup. Developer teams with more than three developers that need collaborator access are required to purchase an upgrade to Unity Licence. The Collaborate features the version history of the project, reverting it to an earlier state and the basic version control system tools such as pulling, pushing and merging. Multiple developers are able to collaborate across the project and the aforementioned features can support multi-disciplinary teams. (Unity User Manual 2018.)

Unity Collaborate was used when creating the Maretarium game. The Collaborate features appealed to the development team. The most appealing factors were the ease of setup, ability to work on multiple workstations at the same time, ability to comment on specific commits and possibility of reverting the game back to its previous state. The team size allowed the use of Unity Collaborate under the Unity Personal license.

4 IMPLEMENTATION

Game development without a game engine is time consuming and challenging. Game design and development time is reduced by using a game engine and other software tools. The implementation of different game features and mechanics can be a daunting task. Create every component and detail of the game including the game and the engine it is built upon requires a great amount of work. Decision is made between whether to make an engine to develop a game or make games with an engine. The latter option in most cases is the most beneficial regarding team goals and desires. (Game engines – How do they work. 2018. Unity.)

4.1 Mini-game collection

The Maretarium game features a collection of mini-games. These mini-games feature different fish that can be seen in the physical fish tanks. Mini-games are playable during the tour mode. In tour mode, mini-games reward tour progression with points. The players acquire Maretarium points from the mini-games. These points accumulate across the different mini-games. The

Maretarium points can be acquired from the sets of goals completed in each of the mini-games. The players' progression level is displayed by Maretarium points which are given at each of the fish tanks. The progression level of the tour indicates the speed and the different skill types and paths taken during the tour mode.

The sets of goals vary on per game basis and are not the same in each of the mini-games. For example, the Feed the Eel and the LohiJump games are both based on the mini-game scoring system of collection of goods. They both represent the collecting aspect of gather-as-much-you-can but endorse different play styles and mindsets. In game of Feed the Eel, the players can plan and think about a strategy for a next move and take time to consider all of possible outcomes as they are progressing towards the finish line. This strengthens the players' ability to plan and think ahead. The LohiJump focuses more on reactive skills and quick thinking than slow-paced planning. This illustrates the different learning experiences that can be witnessed while playing the mini-games and offers a wide range of different learning methods by means of gaming. All of the mini-games feature many of fish in the fish tanks. The players can identify fish seen in the game at the physical fish tank. For example, at one of the locations, the Trivia about the Pike and the Eel game give the players valuable information about them and their habitat. The game is able to educate the players to better understand fish and their environment.

The mini-games are implemented using Unity game engine. Every mini-game has its own configuration and features as described in the above chapter. The mini-game implementations differ from each other. This means that the games are hand-crafted, unique and modified according to the feedback received from the testing group.

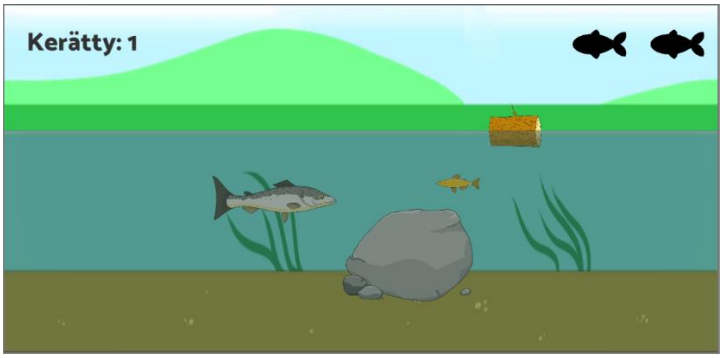


Figure 3. The LohiJump game view.

In every mini-game, the basis is constructed from Unity game objects. These game objects can have multiple component parts attached to them. For example, in the LohiJump, a salmon jumps around the playfield. In order to make the jumping function work as expected, the game object needs a Transform component for game world placement, Sprite renderer for displaying the salmon on screen, 2D-collider for colliding against other game objects and a script so the salmon can jump and act in correspondence to the environment.

```
40      if (Input.GetButton("Fire1") && IsInWater() && !Death)
41      {
42          Jump();
43      }
        1 reference
52      private void Jump()
53      {
54          fishRB2D.velocity = new Vector2(0, jumpPower);
55      }
56
```

Figure 4: Salmon jumping script.

The script containing the salmon functionality uses C# as a scripting language. In Figure 4, in line forty-two, the salmon jumps only when a specific button is pressed and certain conditions are met. The salmon can jump if it is still alive and in the water as seen in Figure 3. In this case, where the jumping logic is under the Input class, the GetButton input call is converted automatically to touch Input by the Unity game engine (Unity User Manual 2018). The salmon jump script is then attached to the salmon game object and placed in the game world.

4.2 Trivia game

The trivia game mode was designed as the core game idea which was that in each of the fish tank locations, a player was to give answers to a set of three questions. The game checks whether or not the given answer was correct as the player answers a question. After the player answers all three questions, the trivia round ends and the player is rewarded with Maretarium points. The player receives Maretarium points per correct answer while heading towards the next fish tank location.

Figure 5. Trivia Interface.

The trivia game displays the correct answer in a green box. If the player did not answer correctly, the chosen box turns red, and the trivia game displays the right answer in the green box. In Figure 5 above, the player has just answered a question incorrectly. The chosen wrong answer has turned red, and the correct answer lights up with a green filament.

The scoring of the trivia questions is based on the players' answer. Mathematically, the players have a twenty-five percent change of choosing the right answer. Every right answer gives a score of ten Maretarium points. Points are given in bulk at the end of each round. The calculated theoretical maximum score per round at a fish tank location is thirty points.

There are as many trivia rounds as fish tanks in the Maretarium aquarium house. Every fish tank location contains a different set of trivia questions. The

players can complete trivia locations in any order they prefer. The order of completed trivia locations does not affect the core game play loop. As was seen in the testing phase, most of the players completed the fish tanks in the order of the followed path. The trivia game location completion list checks and marks every completed fish tank location as the players visits the fish tanks and answers the trivia.

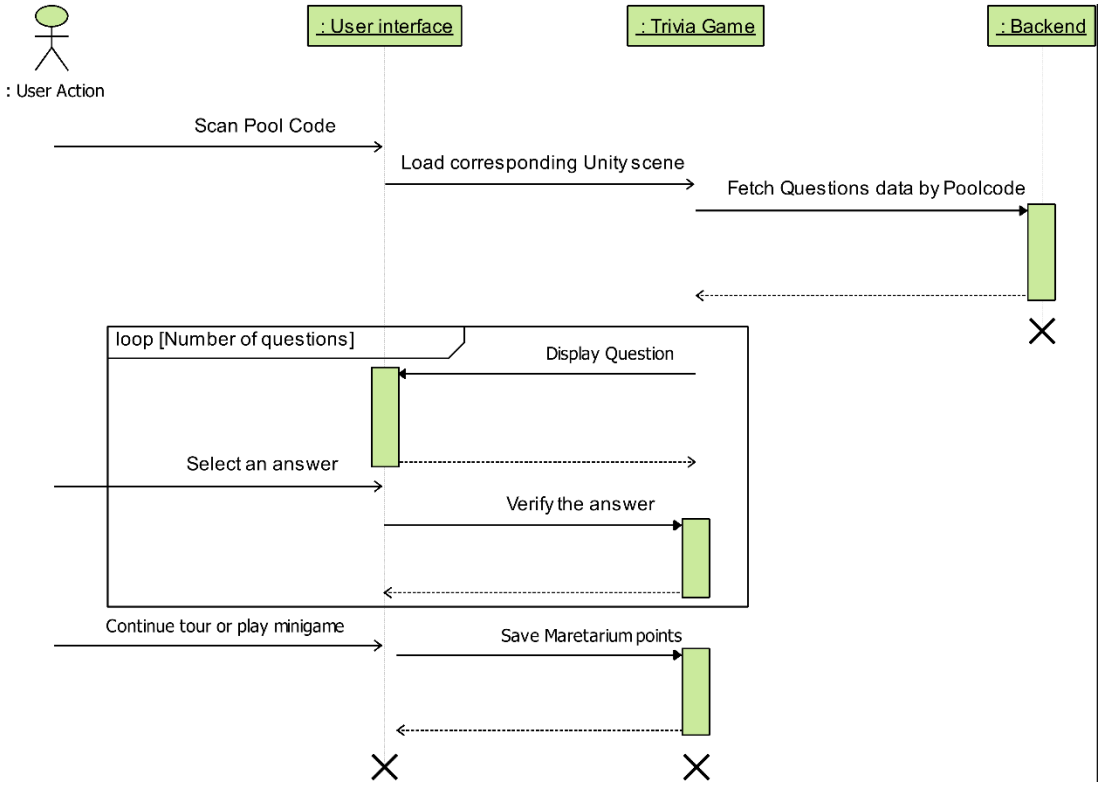


Figure 6. Trivia sequence diagram.

Figure 6 above illustrates how players can complete the trivia game mode with a minimal connection to the database. The number of database connections are minimized as three questions are fetched at once. This provides lower server costs and lower bandwidth usage on players' mobile devices. The trivia game mode fetches questions from the database and stores them temporarily in a selected private game question pool that are then fed into the trivia game mode.

The trivia game fetches the question data from the database and then converts it into a JSON string. Then a question collection is constructed from the received JSON strings. The JSON string is parsed trough a Unity's built-in JSON parser class, which is the preferred way of parsing according to Unity's user manual (Unity User Manual 2018). The speed and usability compared

with other solutions available on the market were significantly better. The parser that is built into Unity was able to convert the QuestionData class back and forth between the server and the client with no significant difficulties. After the client receives a data string from the server, the JSON string is parsed by key-value pairs as described in the JSON Data Interchange Syntax specification (2017, 3). The example of received data would be in form of key identifier followed by data value, as in, {“Question”: “what is the eel favourite food?”}. The received key-value pair data form the QuestionData class that has a property for the key and a value stored into the property. Figure 7 displays possible keys for the different data.

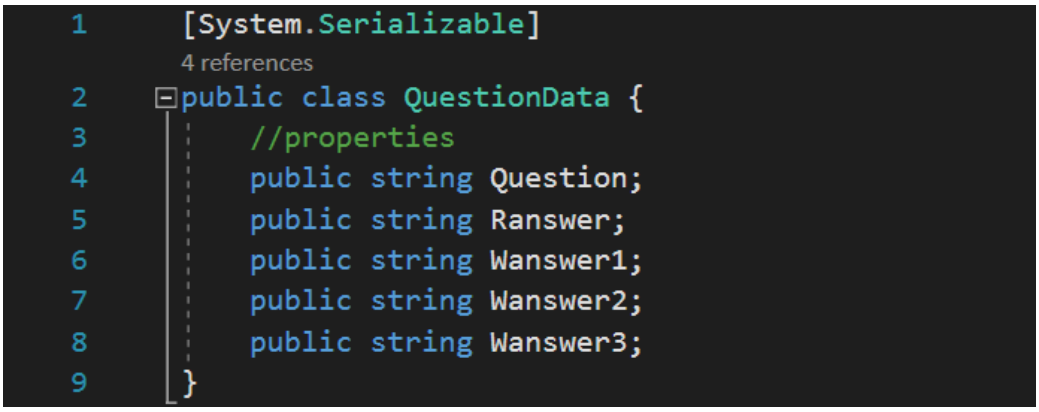


Figure 7. Question data class.

The QuestionData class contains all the information that a question can have. The string type for the question and the answers is also displayed in Figure 7. Possible extensions to this are pictures and other media. The extensions to questions are further discussed in Chapter 7. When all trivia game questions are fetched, the QuestionData objects are added to the list of the active question pool.

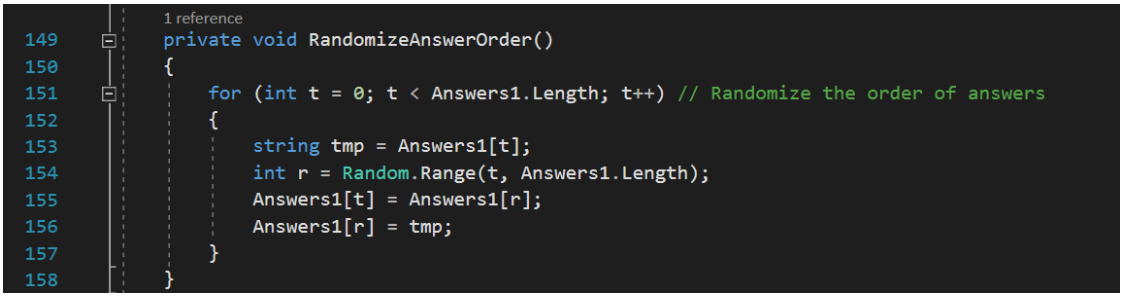


Figure 8. Randomization of answer order.

The active question pool is randomized so that the incoming question order is not the same every time players open the trivia game. This decision of randomizing questions was made to not only keep memorization aspect to the minimum, but also to increase the replayability of the game. As seen during the testing phase the players were not able to memorize the questions which engaged the players to play the game in intriguing and re-playable manner.

In Figure 8, the question pool is randomized and placed in the user-interface block. The randomization uses Unity C# function from Random namespace class to randomize the order of questions. Randomizing is based on the given question pool length. Randomizer method assigns pool's questions to different order at the start of trivia game.

```
1 reference
125     private void SetQ1()
126     {
127         if (NoData || ID1 == 100)
128         {
129             Answers1[0] = RAnswer1;
130             Answers1[1] = "Kalaa";
131             Answers1[2] = "Mätiä";
132             Answers1[3] = "Kaislaa";
133         }
134         else
135         {
136             RAnswer1 = questions.questionPool[ID1].Ranswer;
137             Answers1[0] = RAnswer1;
138             Answers1[1] = questions.questionPool[ID1].Wanswer1;
139             Answers1[2] = questions.questionPool[ID1].Wanswer2;
140             Answers1[3] = questions.questionPool[ID1].Wanswer3;
141             Question1 = questions.questionPool[ID1].Question;
142         }
143
144         RandomizeAnswerOrder();
145
146         Canv.GetComponent<Question>().Rndmz(RAnswer1, Answers1, Question1);
147     }
```

Figure 9. Question component constructed from the QuestionData class.

The data of question is checked and then placed upon the list of questions. The question data check allows the game to continue if no online data is fetched. Default data that replaces the question data of trivia game is integrated into the game client in case that no internet connection can be established. The question data is fetched from the game server if internet connection is established and the user scans the fish tank's QR-code. While the game requires an online connection for the full experience, it is not mandatory.

4.3 Trivia dashboard

Trivia dashboard allows the customization of trivia game mode. The dashboard makes the trivia game mode more interactable; therefore, the number of replays increased as was seen in the testing phase. The trivia dashboard enables the game to be fuller of events and react on seasonal changes. For example, the questions can be focused on seasons or events that are organized at the Maretarium aquarium house.

The trivia dashboard was implemented for Maretarium staff. The Maretarium staff requested an online portal where the questions can be managed. The dashboard provides a way of managing the questions without the need of Unity game implementation. The dashboard can be accessed from a desktop computer or a modern mobile device. For the dashboard access, the device requires an active internet connection to be established.

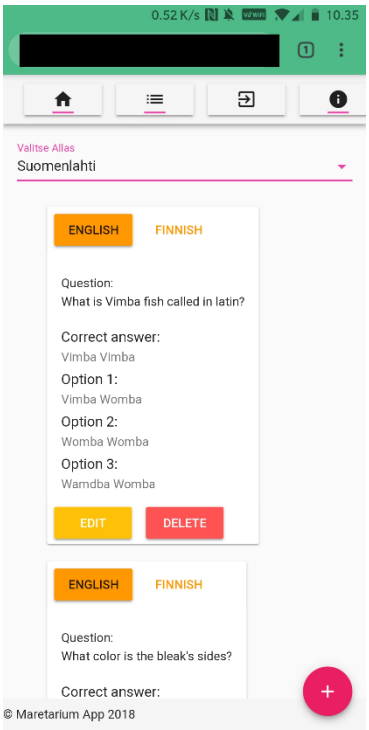


Figure 10. Mobile dashboard interface.

Trivia dashboard is a website where the trivia game data can be managed. The website was made with a responsive design in mind which means that layout of the website adapts to the screen resolution of the device in use. As seen in Figure 10, the website scales the content to mobile devices. For example, the question cards are stacked vertically on top of each other. The

website layout changes based on the resize command sent by the browser window. The add button is a floating action button. The floating action button is a specific Google material design button that floats on top of all the content (Google Material Design 2018). This button can be interacted with the standard Vue.js specific commands. The trivia dashboard has the building blocks for Question data, Question List and the application wrapper. The trivia questions information is retrieved from the Firebase Realtime Database.

A fetch request to receive the information is sent by the application and returned to the question information if correct data was found. If the data is not found on the server, the application returns and displays an error message to the player. Then the question information is parsed and constructed into a template.

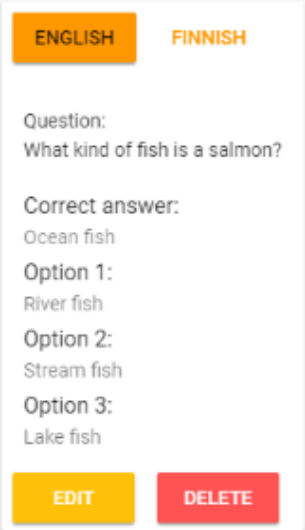


Figure 11. Question data information card.

Question cards are styled with Vuetify components that follow Google's material design (Vuetify 2018). Question cards are in two languages and language choices follow the design of the Maretarium game. The Maretarium game is supported by both English and Finnish. English language support allows wider audience and Finnish language support was mandatory for supporting the Finnish national core curriculum for education.

Question data contains a question text and the answers to the questions. Figure 11 displays a question card. Question cards are editable and removable via the trivia dashboard with authenticated user account. Every

collection of question data contains at least three questions to meet the minimum requirements of the Maretarium game trivia, and the maximum size is the size of the database storage space allowed by the Firebase Realtime Database technologies.

Via the trivia dashboard, the Maretarium game trivia questions can be managed. The operations that can be performed are adding, editing, saving and deleting. Questions can be added with a click of a button. Adding a question appends it to the list of questions. After the question is added, information can be edited. In editing phase, text labels are converted into text fields. Input restrictions are applied to the input fields by the Vue.js input validation library. The first restriction is that the question field needs to be at least four characters long. The second restriction is that all the fields need to be filled with information before the question can be saved. After all the fields are filled correctly, the question can be saved, that is, sent to the server via provided API calls. The delete function removes the question card from the list of questions and calls the API for deletion.

The API is a gateway bridge between the application and the database. API is also known as the connection between the client and the server. (Mozilla API 2018.) In the trivia dashboard, the client is the website and the server is the Firebase Realtime database. With the API, the client can communicate with the database via established internet connection. The implemented application programming interface has the same functions that correspond to the dashboard client functions. The trivia dashboard API uses JavaScript and the Firebase NodeJS package to send the authenticated calls to the database.

5 BACK-END SOLUTION

Implementing the back-end solution makes the game more interactive between the players and provides for example the ability of cloud saving and cross-platform play. The back-end in games allows the developer to make small adjustment to the gameplay without players needing to reinstall the game client every time that a change occurs.

The information stored in the database is publicly available at the Maretarium aquarium house. The deletion, editing and adding functions need an authenticated user account to be completed. The fact that data fetch calls are restricted by authentication was not a concern during the development phase for this particular reason. Due to the Firebase Realtime Database security rules, implementing the authentication of the Maretarium game accounts would be needed to further improve the security of the database (Google Firebase 2018). The need for more stringent database security rules are further discussed in Chapter 7.

Structuring a database can be a challenge for a small multi-disciplinary team. The design process affects the features of the game implemented. The process of managing the database also depends on the back-end solution. Providing a solution that corresponds to the features of the game can be a challenging task. A limiting factor for the game is that when the database is not constructed with a particular game in mind, it becomes very difficult to change the database afterwards. (Google Firebase 2018.)

The data structure of the data stored into database needs to provide a solution for the game implementation. This can include the ability to save scores, achievements and even player progress to the database. Different parts of the saved data can be downloaded to different devices. The online database allows the implementation of multiplayer game and cross-device play. The players of the game can communicate and interact with each other via the implemented back-end server.

The cross-platform play between devices can be enabled by using the online database. The saves, scores, player progression and achievements are transferred to the new device by registering it to the online account. The online account can be registered with a player’s email address, social account or with some other identifier that is specific to the game. For example, the login options for social login are provided by Google and Facebook. The online playability and cross-platform play are further discussed in Chapter 7.

The available database solutions for the Maretarium game database server were examined and assessed by the development team. Setting the cost

range and required set of features for the implementations and development helped in making of the database selection for the Maretarium game. The game required a database functionality for the trivia questions, scoreboard, saving of player progress and achievements earned while playing. All these had to be stated in the original game design document. Due to the time constraints, the development team concluded that only the database for the trivia question data was to be implemented.

5.1 Realtime Database overview

The Firebase Realtime Database was chosen for its ease of data storing in JSON format. The nature of scaling was based on the game's playerbase and the low set-up time, as the Firebase Realtime Database is hosted on the cloud.

The data format that the Firebase Realtime Database uses for its stored data is structured as a JSON-tree. This comes with its advantages and disadvantages. The Firebase Realtime Database is a NoSQL database technology. This allows different optimizations and functionalities compared with the traditional databases that are relational databases. The NoSQL database feature set allows fast execution of operations. This allows the developer to build a responsive and scalable app. JSON formatted data is easy to understand and read (Introducing JSON 2019). For example, the path to the user account can be similar to this example path: "ExampleDatabaseName/UserCollection/Username". The ease of readability allows developers to quickly understand how data is constructed, and the data can be managed even by new employees. (Google Firebase 2018.)

As for the disadvantages, data fetched from the database cannot be nested deeply into the JSON-tree. This is due to the fact that if one of the QuestionData objects are accessed and only one field needed from the question collection data, then all of the child data is also fetched. An example of this is when the developer accesses the path: "ExampleDatabaseName/QuestionCollection/Poolname/". This path contains all of the questions for a particular pool of trivia questions. The path accessed returns all questions. Another example of a data path is when the developer

fetches the data of question based on the question identification number, accessing path requires the line: “ExampleDatabaseName/QuestionCollection/Poolname/Question1/QuestionText”. The data fetched from a deeply nested path can cause performance problems and for this reason data structure should be flattened. (Google Firebase 2018.)

The flattened data allows quick and specific data fetches. It is good to flatten the data based on the needed functionality with regards to the game that is being developed. (Google Firebase 2018.) For example, when using the Maretarium database as an example of the flattened data structured, the questions are divided into languages and they are separated by key identifiers, and the question data is converted into the form of key-value pair collection. Questions are fetched from a collection that contains only the necessary information to the active trivia question fish tank. In the Maretarium game, only one language can be active at a time, so the data is sorted by language. This means that the first database contains questions separated by language. The second database contains the set of questions that are active at the fish tanks. The question activators are stored at different locations, and activated questions are gathered from the language-based database.

The Firebase Realtime Database is a cloud hosted database service. No physical set-up is required for the use of the database. The database service is accessible via the Firebase console and by using the included API calls from the Firebase SDK. The use of Firebase console requires an account. The player signs in by using a Google account and selects the account tier based on the available features of Firebase. The tiers include access to all the different Firebase features other than the Realtime Database, for example, Authentication, Storage and a Cloud Firestore which is an alternative to the Realtime Database. At the time of writing the thesis, the Firestore was in a beta stage and not ready for production use and, therefore, not an option because it provided no support for Unity platform. (Google Firebase 2018.)

The Pricing plans of Firebase are available in three price tiers. The Plans are named as a Spark, a Flame and Blaze. Firstly, the Spark plan is a free tier option. It has tight restrictions about the scope of usage and limited feature

set. This tier is good for non-professionals and for starter projects. The second is the Flame plan. The Flame plan is a pay-by-the-month tier option. The price is approximately twenty-five dollars per month and contains expanded limits compared to the Spark plan. The third tier is Blaze plan which contains the most recently updated features and is by nature a pay-as-you-go tier. The blaze plan can be customized to the needs of the developer and the project. (Google Firebase 2018.)

Realtime Database allows the developer to connect the product to the users. Features that enable to this behaviour are called scaling factors. The service reliability and the quick access to the database are the benefits of Realtime Database.

5.2 Realtime Database set up

The Firebase Realtime Database is set up by the Firebase console. In this chapter, the Firebase Realtime Database set-up is described by following an example and the set-up process is guided with instructions. The Firebase console is the administration panel for the Firebase products, and via the console the developer is able to manage the individual products that are being developed. (Google Firebase 2018.)

Setting up a project from the Firebase console requires no prior high-level technical knowledge, and the developer is guided through the set-up process. After the developer is logged into the Firebase console, the next step is to create a Firebase Realtime Database page.

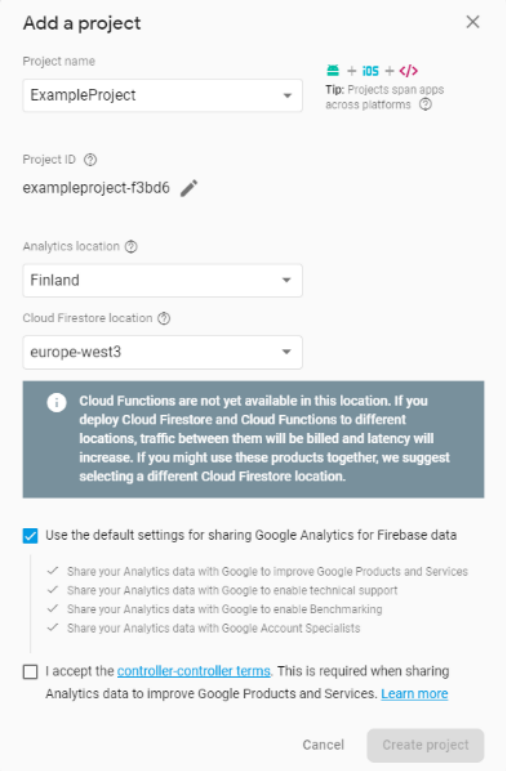


Figure 12. Adding new Firebase Project.

In Figure 12, a developer can define the database name and a project ID. The project ID can be customized to accommodate the project. The location of the cloud Firestore is defined when adding the project, the first time. After accepting the terms and continuing to the next step, a project is created by Firebase.

Figure 13. Basic Realtime Database rules for a new project.

Next, the Firebase asks the developer to set the basic security rules for the database. The Firebase recommends the use of the standard security rules

after the data structure for the database has been created. Enabling Firebase Realtime Database steps brings the developer to the database page where data can be managed.

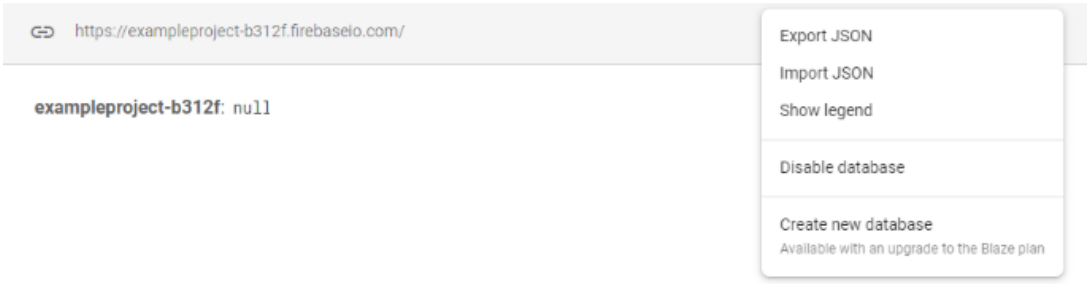


Figure 14. New empty database.

Data can be created from the Firebase console or imported as a JSON formatted file. The import process where the JSON objects are imported from the file into the Firebase Realtime Database can be made from a computer. As stated in the JSON Data Interchange Syntax (2017), the imported JSON data objects are assembled as key-value pairs. The imported data can be accessed from the Firebase Realtime Database by the user as defined in the Firebase security rules.

The Maretarium trivia game database was designed to support multilingual questions and multiple fish tanks. As illustrated in trivia implementation, the database is divided into fish tanks and localized data. The selection of which database section is active is based on the used language at currently active fish tanks. The database design was implemented based on the requirements provided by the Maretarium staff. The multilingual feature was one of the key requests that the development team received regarding the trivia questions. The multilanguage support was beneficial to the user base of the app and retaining customers at the Maretarium aquarium house. The trivia database reflects the data used in both the Maretarium game and in the trivia dashboard. The upkeep of the trivia database is processed through the trivia dashboard and by the Maretarium staff. Question data is categorized by fish tanks. Localizing question data is enabled by the multilanguage support.

5.3 Realtime Database security rules

The Firebase Realtime Database rules are always enforced. Rules determine the integrity of structured data and read and write access rights. The full read and write access rights to the data are most secure for the authenticated users. Data in the database is protected against abuse with these default rules which can be customized to accommodate the need of developers. (Google Firebase 2018.)

Security rules include user rights concerning reading, writing and validating. These rule types follow a JavaScript like syntax and can be set in the Rules panel on the Firebase Realtime Database console. Data content can be validated by the Security rules. Validation function verifies the data integrity based on the given parameters. These parameters can be set based on the data content. For example, the validation may be addressed by string type fields that need to be of a certain length or explicitly correspond to a particular string type. The following example illustrates the validation of the name of a user. "name": { ".validate": "newData.isString() && newData.val().length > 0 && newData.val().length < 20 && !newData.val().contains(HannaHauki)" }

The example of validation contains the string verification, the value of data and the required length of the data. Data integrity is enforced as the user writes data to the database. Write access can be allowed in the Rules hierarchy by the Boolean operator. Setting the data with write access rules containing the authenticated user access and data validation serves as a protection against abuse. Validation can be embedded to write access by providing the validation rules, but this is not recommended according to the Firebase Realtime Database security guidelines. (Google Firebase 2018.)

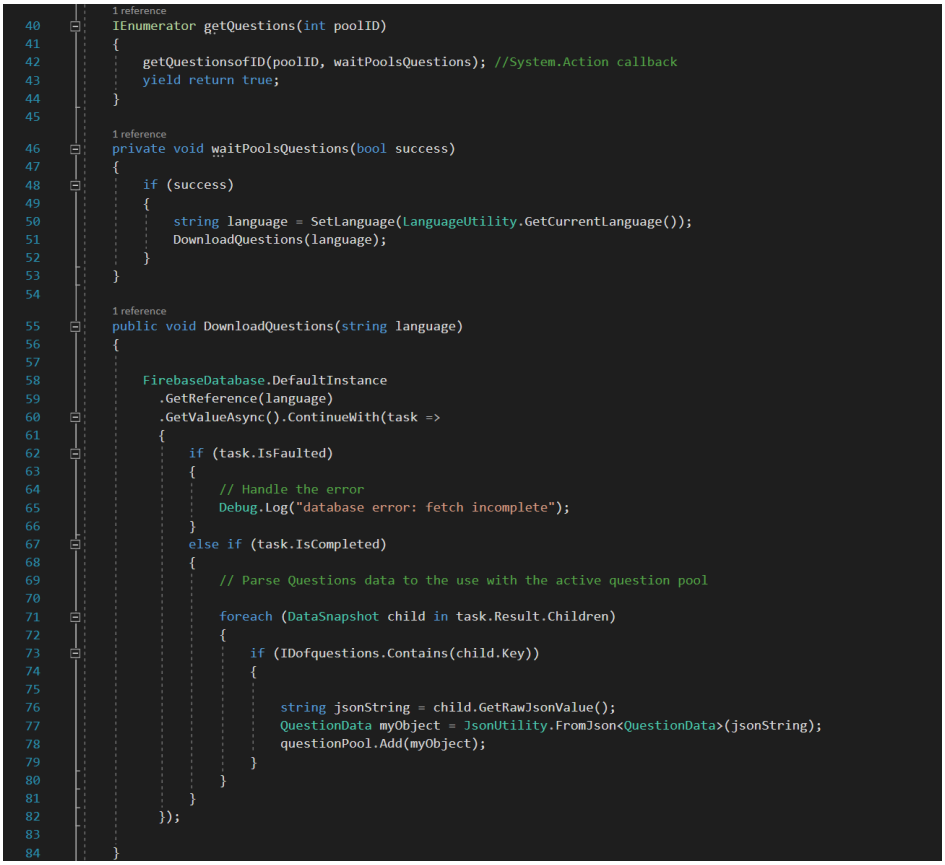
Read access includes the same restrictive capabilities as validation and write rules have. Read access can be set based on the data location and user access. One path can contain public data that can be accessed by everyone, the second path can contain data that requires an authenticated user and the third may only contain the last ten messages from the last fifteen minutes. (Google Firebase 2018.)

Back-end requirements for the Maretarium game have been set by the development team in conjunction with Maretarium staff. The back-end requirements were based on the growth of Maretarium aquarium house. The game features needed to be supported by the back-end. One special requirement was concluded during the development. The cost of the database was to be in the free-of-charge tier. This requirement narrowed the available back-end solutions. The growth of the company has been taken into consideration when choosing the back-end solution. The back-end would need to support the user base of the Maretarium game and have potential to grow in the future. As stated in the post launch reports of the Maretarium game, the traffic accumulated by the playerbase can be supported by the back-end with a Firebase Spark plan. The Spark plan is currently in use and can be upgraded if necessary to have more features or more capacity as the company grows. The growth of the company was also taken into consideration when choosing the back-end. The Spark plan can support the expected growth of the company and provided a solid ground to a small-scale app. (Google Firebase 2018.)

The game requirements for the database included access to the Trivia database. The access was provided by the Firebase SDK plugin, along with the necessary components for the communication between the Firebase Realtime Database and Unity client. The Firebase SDK contains multiple API calls wrapped in ready-to-use functions. Unity client uses these functions to fetch data from the server. This data includes trivia question data such as the QuestionData class that was introduced in Chapter 4.

5.4 Trivia game back-end functions

Trivia question data fetch establishes a connection to the internet and then tries to retrieve data from the database. After a successful fetch, a call sent by Unity client returns a JSON parsed string to the Maretarium game collection of questions and is parsed according to the received data.



```
1 reference
IEnumerator getQuestions(int poolID)
{
    getQuestionsofID(poolID, waitPoolsQuestions); //System.Action callback
    yield return true;
}

1 reference
private void waitPoolsQuestions(bool success)
{
    if (success)
    {
        string language = SetLanguage(LanguageUtility.GetCurrentLanguage());
        DownloadQuestions(language);
    }
}

1 reference
public void DownloadQuestions(string language)
{

    FirebaseDatabase.DefaultInstance
      .GetReference(language)
      .GetValueAsync().ContinueWith(task =>
      {
          if (task.IsFaulted)
          {
              // Handle the error
              Debug.Log("database error: fetch incomplete");
          }
          else if (task.IsCompleted)
          {
              // Parse Questions data to the use with the active question pool

              foreach (DataSnapshot child in task.Result.Children)
              {
                  if (IDofquestions.Contains(child.Key))
                  {

                      string jsonString = child.GetRawJsonValue();
                      QuestionData myObject = JsonUtility.FromJson<QuestionData>(jsonString);
                      questionPool.Add(myObject);
                  }
              }
          }
      });

}
```

Figure 15. Client-side Trivia game database functions.

In Figure 15, the questions are downloaded temporarily to the client device and removed when the client closes the trivia game mode. During the download, active questions are fetched from a Firebase call. If the database call is successful, a C# System Action call-back is executed, and active question ID's are returned from that call. The C# System Action call-back ensures that two Firebase instances are returned successfully, and the database connection does not fail during the trivia data preparation.

Download function utilizes the Firebase Realtime Database SDK to fetch the Question data from the Database. The GetValueAsync and GetReference functions in lines 59 and 60 in Figure 15 ensure that the data is fetched from the correct path and only once per call. This reduces the amount of usage and does not overload the database. The data parsing is made in the completed task block. The task system is asynchronous, which means the function result can be returned later by the executive method as the fetch is in progress. This allows the continuation of game while the questions are retrieved from the server. The task system requires a .NET Framework 4 to function and lambda

expression are used to determine the type of work that the task is performing based on the Status property. (Microsoft Documentation 2018.)

When a task is completed, the DataSnapshot is returned from a successful Firebase Database fetch containing the QuestionData object for the trivia game mode. The snapshot is cycled through where the active questions are parsed with a built-in JSON Utility class and added to the question pool; therefore, ready for the trivia game mode and passed to the interface as introduced in Chapter 4.

6 PLAYTESTING

Testing by organizing gameplay test sessions gives feedback and a more subjective perspective on the gameplay. Player feelings and experiences that emerge during gameplay are the core concept of the gameplay tests. (Schultz & Bryant 2012, 303.) Gameplay test sessions can help to identify the issues of how gameplay feels. Gameplay tests give players the ability to participate in development by giving feedback and suggestions to the developers. Feedback can be informal or very structured depending on the test case and testing methods used. Received feedback is used to improve the game.

This chapter describes what is included in testing sessions and who can participate. How the Maretarium game is used in production environment and how the feedback guided the production towards a more consumer-friendly direction.

The Maretarium game team measured the players' feels and the experiences they had while playing the game. The experiences during the gameplay test session were positive. The whole test group thought that the game was engaging. Overall, the fun factor and competitive aspect occurred beyond other feelings. These experiences were a highly desired outcome of the gameplay sessions. The atmosphere was light and joyful during the gameplay testing session that was held at the Maretarium aquarium house and participants eagerly took part in it.

6.1 Test group

The test group consisted of ten elementary school students who were familiar with mobile devices. All of the students that took part in testing were allowed to use mobile devices in class with their legal guardian's consent.

Information and communication technology is an essential part of all versatile learning environments. It helps students to strengthen their interpersonal skills and involvement in their personal learning paths. Media culture is taken into consideration in the development of learning environments. Information and communication technology support and enhance the learning process. The students' personal information technology devices can be used as part of the learning process in ways which has agreed on with the student's legal guardian. At the same time, it should ensure that all students have an opportunity to use information and communication technology. (Perusopetuksen opetussuunnitelman perusteet 2014, 29.)

6.2 Test feedback

Gameplay testing feedback is important and should be presented in organized, detailed and specific manner. Having detailed and specific feedback allows the game development team to adapt the game to a more customer-friendly product. Well-constructed feedback is informative and concise and advances the development cycle of product. Improving the game in a test environment that corresponds to real world usage allows narrowing the product scope based on statistics and organized feedback. (Schultz & Bryant 2012, 296.)

The development team gathered feedback with an online form which consisted of series of questions. The questions answer to what kind of a participant was involved. The students in the test group answered questions on gaming habits and familiarity with mobile gaming in general.

Ad hoc testing allows players to test the game in the suggested play mode. Giving players the freedom to test the game as they please, helps to discover bugs and unwanted features that might never surface in a structured and controlled test environment. Players are able to give feedback regarding the

features that are important to them. The flow of gameplay can be modified by processing player feedback and the gameplay sessions test results. (Schultz & Bryant 2012, 285.)

7 CONCLUSION

The purpose of this thesis was to describe the implementation of the Maretarium game. The game's core concepts were successfully created and at the time of writing this thesis the game is in use at the Maretarium aquarium house. The game can be improved in multiple areas and developed further. As stated in Chapter 2, the multiplayer features are an area where the game can be improved. In the future, multiplayer features will include account support, high score display, competition against one's friends in trivia and social features such as sharing personal high scores to an online platform and personalizing one's account profile. The social aspects of multiplayer mode can be implemented by having a proper back-end for the game. Implementing social features raises a database security concern and stricter database security rules may be necessary to ensure that information is not publicly available.

In the future, this game concept and the game itself are adaptable to other environments and tourist attractions. Game content can iterate across multiple scenarios and be tailored for specific locations and needs of customers. The trivia implementation could be improved by having support for multiple question types such as images in question text and answer options. These again require good back-end and proper examination of available solutions as a server can require significant amount of storage space for images and a great amount of funding to cover the expenses. Acquiring a complete server solution where it fulfils all aforementioned multiplayer features is challenging task for a small development team.

For the author, this study provided an excellent experience in making a game ready for production. While working on this thesis, the amount of game development experience has grown tremendously while improving the author's team skills. The Maretarium game is available in the Google Play store, and the process of publishing provided good experience on its own.

REFERENCES

API. 2018. Mozilla Developer Network. WWW document Available at: https://developer.mozilla.org/en-US/docs/Glossary/API [Accessed 1 Feb 2019].

Chacon, S. & Straub, B. 2014. Pro Git. E-Book. 2nd Edition. Apress. Available at: https://git-scm.com/book/en/v2 [Accessed 3 Dec 2018].

Copes, F. 2018. The Vue Handbook. A thorough introduction to Vue.js. E-Book. Available at: https://vuehandbook.com/ [Accessed 18 Dec 2018].

Digiverstas. 2019. XAMK. WWW document. Available at: https://www.xamk.fi/tutkimus-ja-kehitys/digiverstas-2/ [Accessed 13 May 2019].

Documentation. 2018. Microsoft. WWW document. Available at: https://docs.microsoft.com/ [Accessed 14 Dec 2018].

Doran, J.-P. & Casanova, M. 2017. Game Development Patterns and Best Practices. Birmingham: Packt Publishing Ltd.

ECMAScript 2015 Language Specification. 2015. 6th edition. Ecma International. PDF document. Available at: http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf [Accessed 18 Dec 2018].

Firebase. 2018. Google. WWW document. Available at: https://firebase.google.com/ [Accessed 27 Nov 2018].

Game engines – How do they work. 2018. Unity. WWW document. Available at: https://unity3d.com/what-is-a-game-engine [Accessed 3 Dec 2018].

Introduction. 2018. Vue.js. WWW document. Available at: https://vuejs.org/v2/guide/ [Accessed 28 Nov 2018].

JSON. 2019. Introducing JSON. WWW document. Available at: https://www.json.org/ [Accessed 3 Jan 2019].

The JSON Data Interchange Syntax. 2017. Second edition. Ecma International. PDF document. Available at: http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf [Accessed 3 Jan 2019].

Kramarzewski, A. & De Nucci, E. 2018. Practical Game Design. Birmingham: Packt Publishing Ltd.

Material Component Framework for Vue.js 2. 2018. Vuetify. WWW document. Available at: https://github.com/vuetifyjs/vuetify [Accessed 18 Dec 2018].

Material Design. 2018. Google. WWW document. Available at https://material.io/ [Accessed 18 Dec 2018].

Pal S. 2018. Software Engineering, Classical Waterfall. GeeksforGeeks. WWW document. Available at: https://www.geeksforgeeks.org/software-engineering-classical-waterfall-model/ [Accessed 14 Jan 2019].

Palmer, C. & Williamson, J. 2018. Virtual Reality Blueprints. Birmingham: Packt Publishing Ltd.

Perusopetuksen opetussuunnitelman perusteet. 2014. Opetusministeriö. PDF document. Available at: https://www.oph.fi/download/163777_perusopetuksen_opetussuunnitelman_perusteet_2014.pdf [Accessed 19 Nov 2018].

Principles behind the Agile Manifesto. 2001. WWW document. Available at: https://agilemanifesto.org/principles.html [Accessed 14 Jan 2019].

Ryan T. 1999. The Anatomy of a Design Document. WWW document. Available at: http://www.gamasutra.com/view/feature/3384/the_anatomy_of_a_design_document_.php [Accessed 14 Jan 2019].

Schultz, C. & Bryant, R. 2012. Game Testing All In One. Second Edition. Dulles, Virginia: David Pallai.

Unity Features. 2018. Unity. WWW document. Available at: https://unity3d.com/unity [Accessed 16 Nov 2018].

User Manual. 2018. Unity. WWW document. Available at: https://docs.unity3d.com/Manual/index.html [Accessed 27 Nov 2018].

FIGURE LIST