Andreas Blakli
Vegard Opkvitne Årnes
Theo Camille Gascogne
Jesper Ulsrud

Gamification of Curricula for Primary, Lower Secondary, and Upper Secondary Schools

Bachelor's project in Programming
Supervisor: Mariusz Nowostawski
May 2021

Bachelor's project

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

Andreas Blakli
Vegard Opkvitne Årnes
Theo Camille Gascogne
Jesper Ulsrud

Gamification of Curricula for Primary, Lower Secondary, and Upper Secondary Schools

Bachelor's project in Programming
Supervisor: Mariusz Nowostawski
May 2021

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

Abstract

Through the years, teaching methods have developed further from their traditional ways in order to stay up to date with the modernisation of multimedia. NTNU Gjøvik wishes to explore the possibilities with providing a platform for the course coordinators to present their curriculum for their students through the interface of a game, optimising the experience for interaction and visual feedback with intent to provide the students with a greater, more consistent sense of accomplishment than traditional means of studying. The implementation consists of an Kotlin Android front-end with LibGDX/LibKTX for the cross-platform game engine, and Firebase for storage. The service provides an interface for structuring classes into classrooms consisting of one or multiple modules with either self made content for mini games, or imported ones created by fellow teachers. Students are able to join these classrooms and launch specific mini games, or access the mini game through an open world view. In the open world the students can interact with their teachers and be given quests. Upon completing the games, students are rewarded with points, and possibly achievements for their overall performance. In addition, users play as their configured avatar to make the player feel more involved in the game.

The game engine developed for this project is structured into an Entity Component System (ECS), with entities composed of components and systems that control how the data types and values in the components are manipulated and used. This combined with the LibKTX and LibGDX allows for great customizability, lightweightness and scalability for the engine. This simplifies adding new mini games as already existing components and systems can be reused.

The application as it is now gives any course coordinator, teacher or teachers assistant the opportunity to deliver any teaching material in a more interactable and rewarding way for the students. The way the quizzes are played differs from the usual way of just clicking the answers and lets the players navigate around an open world to handle the challenges.

Sammendrag

Oppigjennom årene har læringsmetoder utviklet seg mer og mer fra de tradisjonelle metodene for å holde følge med moderniseringen av multimedia. NTNU Gjøvik ønsker å utforske mulighetene til å gi lærerne en plattform til å presentere undervisningspensumet gjennom et spillgrensesnitt, for å optimalisere opplevelsen med interaksjon og visuelle tilbakemeldinger med den hensikt å gi et forbedret læringsutbytte, med bedre og mer konsistent mestringsfølelse. Implementeringen består av en Kotlin Android front-end med LibGDX/LibKTX for kryssplattform spillmotor, og Firebase til lagring. Systemet gir et grensesnitt for lærerne til å strukturere klassene inn i klasserom bestående av en eller flere moduler med enten selvlaget innhold for minispill, eller importert fra andre lærere. Studenter kan bli med i klasserom og starte spesifikke minispill, eller få adgang til minispillene gjennom en åpen spillverden. I den åpne spillverdenen kan studentene samhandle med lærerne og bli gitt utfordringer. Når spillene eller utfordringene blir utført, belønnes studenten med poeng, og mulige oppnåelsesmedaljer for innsatsen. I tillegg kan alle brukere spille med en egendefinert avatar for å gi spillerne en mer involvert følelse i spillverdenen.

Spillmotoren som er utviklet for dette prosjektet er strukturert inn i et Entity Component System (ECS, eller Enhets Komponent System på Norsk), enhetene består av komponenter. Systemer kontrollerer hvordan data typene og verdiene inne i komponentene blir manipulert og brukt. Dette kombinert med LibGDX/LibKTX gir god tilpassbarhet, lettvekt og skalerbarhet for spillmotoren. På grunn av hvordan man kan bruke opp igjen komponenter og systemer forenkler dette prosessen for å lage nye minispill.

Med måten motoren er satt opp er det enkelt å legge til nye minispill ved hjelp av alle de samme komponentene som allerede er i systemet. Applikasjonen slik den nå er, gir en hvilken som helst kurskoordinator, lærer eller lærerassistent muligheten til å levere undervisningsmateriell på en mer interaksjonell og givende måte for studentene. Måten quizene spilles på, skiller seg fra den vanlige måten å bare klikke på svarene og la spillerne navigere rundt i en åpen verden for å håndtere utfordringene.

Contents

Figures

Tables

Chapter 1

Preface

This Bachelor project was provided by the Department of Information Security and Communication Technology at the Faculty of Information Technology and Electrical Engineering by the Norwegian University of Science and Technology (NTNU IIK). This project was one of the top choices our group had put down, and it instantly caught our attention and curiosity. Every group member had previous experience with Android mobile development and game development using different technologies. The project allowed us a lot of freedom to choose the direction of the design in the game and application itself, which all group members appreciated immensely. In addition, the potential to further develop our skills and experience in both application and game development from this project was great, and we all want to thank NTNU IIK for the opportunity. We also want to give a big thanks to Espen Torseth, Basel Katt as the clients, and Mariusz Nowostawski as our study supervisor for all advice, guidance, feedback and encouragement. It has helped us improve the quality of the project and thesis tremendously.

Chapter 2

Abbreviations and Definitions

Android	Mobile operating system based on a modified version of Linux
Android Studio	The official integrated environment for developing Android applications
Android SDK	SDK stands for Software Development Kit, it is developed by Google for the Android platform
Kotlin	Programming language
LibGDX/LibKTX	LibGDX is an open source game development framework, LibKTX is the Kotlin extension for LibGDX
LibGDX Ashley	An entity framework, for creating different entities in games
GDXLiftoff	Setup configuration tool for LibGDX projects
OS	Operating system
iOS	Mobile operating system developed by Apple
Scrum	Agile development framework, often used for software development
Scrum team	All individuals working together using the scrum method, a team has different roles for the members
Scrum master	Role in a scrum team. A leader for the team, responsible for the other team members
Product owner	Role in a scrum team, the person representing the stakeholders
Sprint	Repeatable fixed length events used in scrum, with a concrete development goal to be reached
Product Backlog	List of all things that must be done to complete a project
Sprint Backlog	Subset of a Product Backlog for items to be completed during a specific sprint

Table 2.1: Abbreviations and Definitions part 1

Firebase	Platform for creating mobile and web applications
Firestore	NoSQL database from Firebase
ECS	Entity Component System used in this application, the main architecture of the game is structured into this system
NPC	A non-playable character in a game
Unreal Engine	Game engine developed by Epic Games
UI	Means User Interface, anything a user may interact with while using a software or any digital product
Gimp	Digital image manipulation software
Aseprite	Pixel-art tool
Adobe XD	User experience design tool
Git	Version control system used for software development
Git Bash	Application providing a command line experience for Git in Windows
Github	Web platform hosting software development version control and online repository using Git
Trello	Web application for organizing and prioritizing projects or tasks
Discord	Online communication platform for voice and text chat, and sharing content
Zoom	Online communication platform for video, voice or text chat
Prototype	A prototype is an early sample, model, or release of a product built to test a concept or process
Coroutine	Concurrency design pattern that you can use on Android to simplify code that executes asynchronously
API	Application Programming Interface, is an interface that defines interactions between multiple software applications or mixed hardware-software intermediaries
Alpha version	The first phase of the software release cycle
Beta version	The second phase of the software software release cycle
Gradle files	Gradle files are the main script files for automating the tasks in an android project and are used by the Gradle for generating the APK from the source files
APK	Android Package File, used to distribute applications on Google's Android operating system

Table 2.2: Abbreviations and Definitions part 2

Chapter 3

Document Structure

- **Introduction:** This chapter contains a short introduction of the project work.
- **System Specification:** This chapter contains the system requirements, user stories, use case diagrams and product backlog for the project.
- **System Architecture Design:** This chapter explains the system architectural design choices made regarding the application, game engine, game design, story, mechanics and game modes.
- **System Graphical Design:** This chapter explains how the graphical design process, game art style and layout of user interface was done.
- **The System Development Process:** This chapter explains the system development process of this project.
- **System Implementation:** This chapter explains how the planned features were implemented in the final product.
- **Quality Assurance and Testing:** This chapter describes how the quality assurance process was done with the testing process of the application.
- **Installation:** This chapter describes the installation process for the application on a PC using Android Studio with Emulator and installation on a physical/real Android device.
- **Result:** This chapter contains the final results of the accomplished goals for the finished application.
- **Discussion:** This chapter contains the discussion concerning the project work, what could have been done differently, further work for the application, summary and evaluation of the group's work.
- **Conclusion:** Conclusion of the whole project, based on the results in Chapter 12 and reflections around the solution

Chapter 4

Introduction

The Norwegian Directorate for Education has prepared a whole new curriculum for digital security for all ages in the school system (Primary, Lower Secondary, and Upper Secondary School). The newly prepared curriculum states that digital security will not be a subject by itself, but instead be a part of every individual subject. This is going to be a big challenge for a lot of teachers, especially coming up with a complete approach to the subject area, as digital security rarely will be part of the teacher's core competence.

This is why NTNU IIK wants to support the teachers so their work is simplified by developing an application that would make it easier to teach the Digital Security curriculum. The application would be a game-teaching app, that focuses on the gamification of the curriculum to make it more fun, interactive and help give the students a high learning outcome. Experience from other similar apps is that they can often be perceived as boring, irrelevant and not engaging enough to students, see the bachelor project task in the appendix A.0.3. The goal is to develop an engaging application that changes this so that it feels relevant to both the students and teachers.

Due to the size of the project it was decided with the clients Espen Torseth, Basel Katt and study supervisor Mariusz Nowostawski to scale down the project's requirements and focus on some key aspects of the task. The clients' expectations were a running prototype of the principal functionality and not a complete running system. And this project's results would be used as input for improving the specification of a planned system that will start development later in 2021.

The research question this project focuses on is the game and classroom modules for teachers. Modules will be highly customizable, allowing teachers to fill them with the required content at any given time, while giving them the ability to configure their own mini games by using the curriculum. One of the primary goals is to make an app that can give teachers a way to provide fun and interactive ways for students to learn the new curriculum. The application is supposed to make it possible to use in any given course, not just for the Digital Security curriculum.

4.1 Group Background and Competence

The group consists of four students, three of which are on the game programming track and one on the application programming track of the study. All four have prior experience in Android Studio and Android Mobile programming from the Mobile/Wearable Programming course. The application will be focused on the Android platform, and will be developed in the programming language Kotlin, using Android Studio as the IDE. For the game development it was decided to use LibGDX and LibKTX to create the game engine, libraries which the group members had no prior experience in. It was chosen as it would allow for high customizability, low requirements from users device, and easy adaptation for other platforms.

4.2 Project Audience

The application itself is aimed at all students and all teachers in the Norwegian school system, all the way from Primary School, to Lower Secondary and Upper Secondary School.

4.3 Thesis Audience

The thesis is aimed at an academic audience which understands the English language and has experience in data technology and informatics, with an interest in application/game design and development for mobile devices. It is primarily for NTNU's clients of this project, but is also suitable for potential investors or other people who want to support this project further after the initial proof of concept has been developed.

4.4 Roles

A hierarchy was created to organize group work, this was done to improve workflow so everybody always had something they could work on and would not sit idle and do nothing. One group leader and one secretary. The group leader's responsibilities was to organize the group members e.g. when to meet, what to prioritize in the development process, contact with the stakeholders, study supervisor and scheduling of guidance sessions with said stakeholders and study supervisor. The secretary's responsibilities were to take minutes of meetings with the stakeholders, study supervisor and contact with the stakeholders, study supervisor.

Group leader	Andreas Blakli
Group secretary	Vegard O. Årnes

Table 4.1: The group members' roles

Project work was organized to support the individual group member's skill set to maximise productivity, note: this was a guideline, not a rule written in stone.

Game Engine	Andreas Blakli
Application	Vegard O. Årnes
Design	Theo Camille Gascogne
Database	Jesper Ulsrud

Table 4.2: The group members' main responsibilities

Realistically, all of the group members' have had the roles laid out in the table above as everybody to some extent has been working on all main modules of the system, but the main responsibility of a given system belongs to the person which has been given that designated role.

4.5 Software Development Methodology

Scrum was chosen as the development process for the project. The sprints duration was decided to last for a week. Sprint planning and sprint retrospective meetings were held each Monday, and daily meetings were held throughout each workday. Each Monday at 11am there was also a scheduled meeting with the clients and the supervisor. See Section 8.2 for a more detailed description of how the Scrum method was used.

4.6 Progress Plan

In the project plan, all the way in the beginning while developing the foundations of the applications, a Gantt chart was set up for the work process of the project. See the Project Plan in A.0.2 chapter 6.2 for full Gantt chart. Each week had its individual concrete theme and goal. Some of these goals had to be changed throughout the process, because of how the scoping of the project unfolded during development. These changes are discussed in different sections in the thesis, depending on the area of the project that had to be changed.

Chapter 5

System Specification

5.1 Requirements

The main project requirements laid out by the clients and stakeholders was an application that had the goal to gamify the new upcoming digital security curriculum in the Norwegian school system. The user base for the application would primarily be teachers and students, at any grade in any course. As the application needs to be suitable for any context in any class, for all ages, it should be easy for the teacher to create and implement a gamification of any curriculum with as much creative freedom as possible. See Chapter 4 for further explanation of what the application seeks to accomplish.

5.1.1 Functional Requirements

1. **Gamification** - The app needs to offer a gamified experience to the user.
2. **Creative freedom** - The teachers should be able to create their own classrooms to host self-made or imported modules with self-made or imported mini games.
3. **Editable** - The course coordinators need to be able to add new content in to their classrooms modules at any time.
4. **Classrooms** - Students should be able to join a classroom that a teacher has set up.
5. **Android platform** - The app's primary platform should be on android phones and tablets.
6. **Cross-platform** - The app should be easy to port to other devices running different operating systems e.g. iOS and windows 10.
7. **Profile** - The app needs to have a user profile where users can edit their in-game avatar.

5.1.2 Non-Functional Requirements

1. **Scalable** - The app should be able to scale well to any screen with small or big resolutions.
2. **Modular** - The app should be modular and dynamic to allow for content updates, e.g. adding new mini games, achievements, game maps, and other new features.
3. **Pedagogically viable** - The game experience should help the students learn the curriculum.
4. **Performance** - The application should be lightweight to allow weaker and older devices to run the program. Support Android 8.0 and newer.
5. **Reliability** - Service uptime 99.5%.

5.1.3 Timetable

During early planning, a timetable was made to allocate time and human resource to the systems that were to be implemented in the application, which the group members would follow. The initial timetable can be seen in the Gantt-schema in the Appendix A.0.2. The changes made to the timetable is discussed in Section 5.2 and the product backlog can be seen in Section 5.7.

5.2 Initial Changes from the Project Plan

In the prototyping phase, the requirements engineering process was a continuous task revisited with every iteration. In discussions with the clients and study supervisor ideas were scrapped or adjusted to make sure the application's solutions stayed relevant and within scope, see the project plan found in appendix A.0.2. In the requirements, a multiplayer gameplay option was specified as being necessary for our intended system, due to miscommunication on what was optional and what was necessary. The multiplayer feature was therefore removed from the requirement, but is a part of the possible future extensions to the application. The service was also intended to be functioning on web browsers as to not discriminate against people with sub-standard phones or tablets, and while the game is playable on both desktop and in browser, the work needed to translate the application UI to work on desktop as well, was considered redundant and was therefore scrapped. In addition, creating an entire forum website environment for teacher was a goal specified in the Gantt-schema as it would make it easier for teachers to share their modules or games with each other. Instead, it was made a requirement to simply implement the functionality necessary for teachers to import each other's modules and games without a UI dedicated to making it shareable.

5.3 User Stories

Actor	Story
Teacher	As a teacher I need to register an account so that I can use the application with teacher permissions
Student	As a student I need to register an account so that I can use the application with student permissions
All	As a user I need to login to the application
All	As a user I need to logout of the application
All	As a user I need to be able to configure my avatar
Teacher	As a teacher I need to create a classroom
Teacher	As a teacher I need to create a module
Teacher	As a teacher I need to create a quiz
Teacher	As a teacher I need to play the quiz I created
Teacher	As a teacher I need to play the open world game
Teacher	As a teacher I need to delete the classroom I created
Teacher	As a teacher I need to delete the module I created
Teacher	As a teacher I need to delete the quiz I created
Teacher	As a teacher I need to import another teacher's module
Teacher	As a teacher I need to import another teacher's specific quiz
Student	As a student I need to join my teacher(s) classroom(s) and get access to the module(s) and stored quiz(zes) my teacher(s) has created
Student	As a student I need to be able to choose between the quizzes my teacher(s) has created and play the specific game I chose
Student	As a student I need to play the open world game where I can interact with my teacher(s) and see all the quizzes they have created
Student	As a student when I play the open world game I need to be able to chose between and play the quiz(zes) my teacher(s) have created
Student	As a student I need to check my total score and achievements so that I can see my progress

Table 5.1: User Stories

5.4 Use Cases Teacher

Use Case 1	Create Quiz
Actor	Teacher
Use Case Overview	The teacher logs into his/her account, then navigates to the classroom and module they want to add a quiz to.

Table 5.2: Use Case 1

Use Case 2	Create Classroom
Actor	Teacher
Use Case Overview	The teacher logs into his/her account, then navigates to the classroom list and adds a classroom

Table 5.3: Use Case 2

Use Case 3	Create module
Actor	Teacher
Use Case Overview	The teacher logs into his/her account, then navigates to the classroom and enters said classroom and creates a new module.

Table 5.4: Use Case 3

5.5 Use Cases Student

Use Case 4	Play specific quiz
Actor	Student
Use Case Overview	The student logs into his/her account, then navigates to the classroom chooses a module and then chooses a quiz to play

Table 5.5: Use Case 4

Use Case 5	Play in the open world game
Actor	Student
Use Case Overview	The student logs into his/her account, then navigates to the home screen and enters the game

Table 5.6: Use Case 5

Use Case 6	Change avatar
Actor	Student
Use Case Overview	The student logs into his/her account, then navigates to their user profile then chooses edit profile

Table 5.7: Use Case 6

5.6 Use Case Diagram

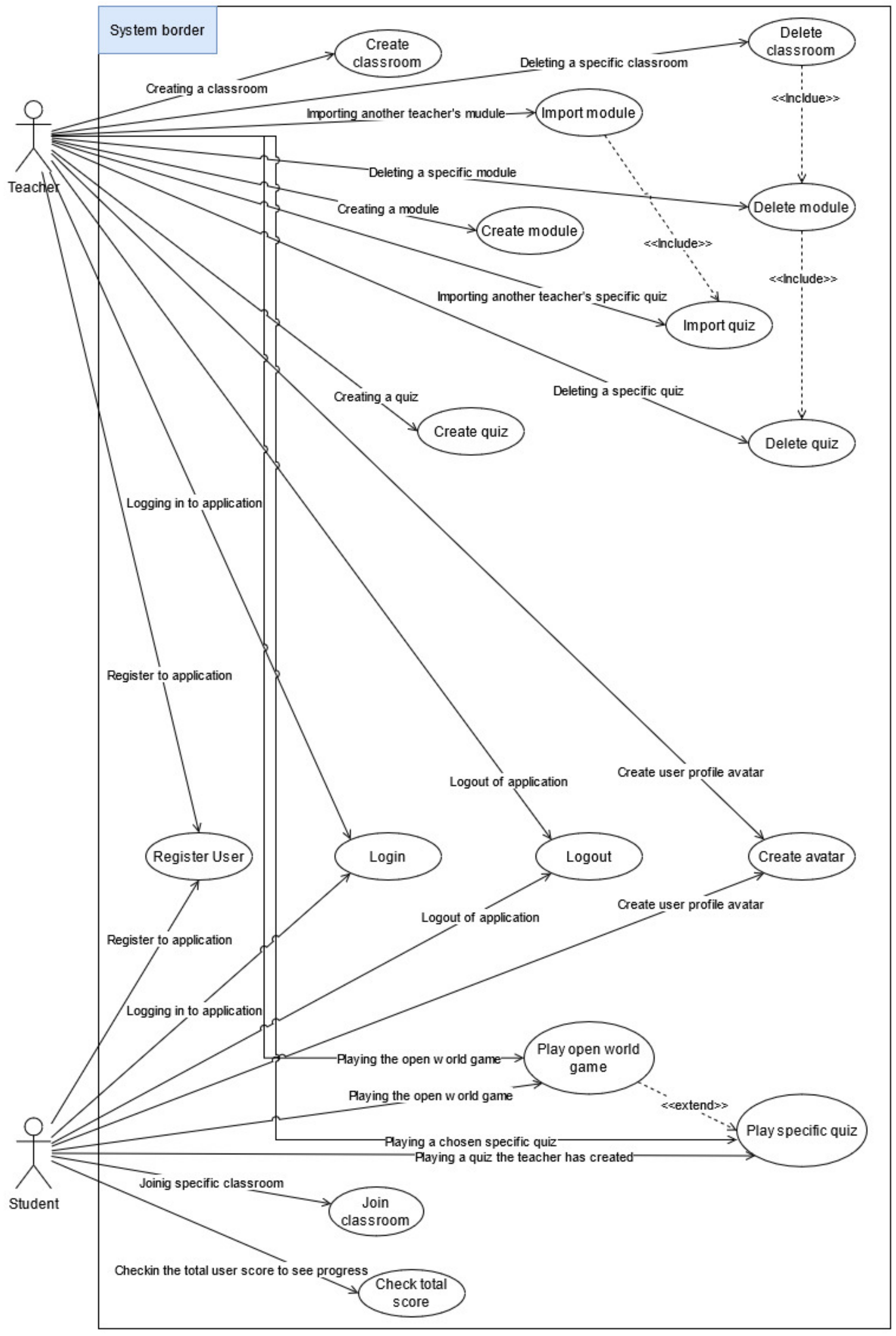
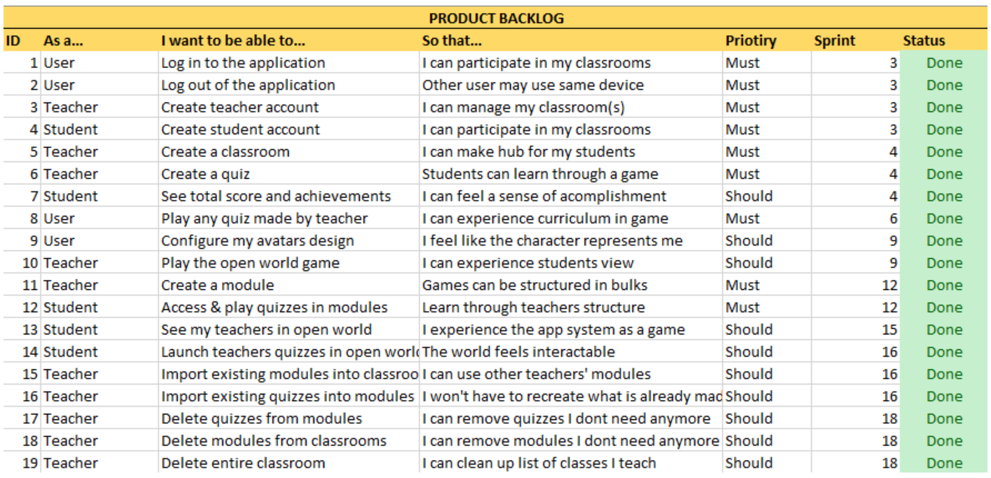


Figure 5.1: Use Case Diagram

The use case diagram Figure 5.1 shows how the system is intended to work and the many actions a user can do with it. A user wishing to use the app may first log in to the app. For the teacher, they may create a new classroom and new modules for that classroom or, if they wish, import modules to that classroom. For these modules, quizzes can be made or importing pre-made quizzes into it. Importing a module is marked with an include to importing another teacher's quiz since an imported module may come with pre-made quizzes. The teacher can also delete classrooms, modules, and quizzes. Deleting a classroom includes deleting all modules inside classroom, which will also delete every quiz inside those modules. Both the teacher and the student may register users and create an avatar and join the open world game. The students, once they have registered and created an avatar, may join specific classrooms start any quiz they wish to do inside it or simply play the open world game which by extension lets quizzes be accessed. Between quiz sessions, the students may check their accumulated total score. Once a user has finished using the app, they may log out.

5.7 Sprint Backlog



PRODUCT BACKLOG						
ID	As a...	I want to be able to...	So that...	Priotiry	Sprint	Status
1	User	Log in to the application	I can participate in my classrooms	Must	3	Done
2	User	Log out of the application	Other user may use same device	Must	3	Done
3	Teacher	Create teacher account	I can manage my classroom(s)	Must	3	Done
4	Student	Create student account	I can participate in my classrooms	Must	3	Done
5	Teacher	Create a classroom	I can make hub for my students	Must	4	Done
6	Teacher	Create a quiz	Students can learn through a game	Must	4	Done
7	Student	See total score and achievements	I can feel a sense of acomplishment	Should	4	Done
8	User	Play any quiz made by teacher	I can experience curriculum in game	Must	6	Done
9	User	Configure my avatars design	I feel like the character represents me	Should	9	Done
10	Teacher	Play the open world game	I can experience students view	Should	9	Done
11	Teacher	Create a module	Games can be structured in bulks	Must	12	Done
12	Student	Access & play quizzes in modules	Learn through teachers structure	Must	12	Done
13	Student	See my teachers in open world	I experience the app system as a game	Should	15	Done
14	Student	Launch teachers quizzes in open worl	The world feels interactable	Should	16	Done
15	Teacher	Import existing modules into classroo	I can use other teachers' modules	Should	16	Done
16	Teacher	Import existing quizzes into modules	I won't have to recreate what is already mad	Should	16	Done
17	Teacher	Delete quizzes from modules	I can remove quizzes I dont need anymore	Should	18	Done
18	Teacher	Delete modules from classrooms	I can remove modules I dont need anymore	Should	18	Done
19	Teacher	Delete entire classroom	I can clean up list of classes I teach	Should	18	Done

Figure 5.2: Sprint Backlog

The sprint backlog is the structure in which the tasks were divided and conquered. The structure for sprints were inspired by plans made in project plan Gantt chart visible in chapter 6.2 within A.0.2, though it saw some adjustments during development time as tasks required more sprints to complete. The actual content of the backlog is based on the user stories prepared in Table 5.1. Then the gap in sprint number signify either time taken to refactor, bug fix, or complete previously unfinished functionality from earlier sprints, sprints in which the goal was to learn more about necessary technologies, or sprints dedicated to writing the thesis. Further information about the version history of the application can be seen in appendix under section A.0.4

Chapter 6

System Architecture Design

6.1 Technologies and Frameworks Used

LibGDX

LibGDX is a framework for creating game engines. It is based on OpenGL (ES) and it is written in the Java programming language. It is free and open source and provides the option to be cross-platform between android, iOS, Linux, Mac OS X, Windows, BlackBerry, and web browsers with the use of WebGL. LibGDX was primarily chosen for this project because of the customizability and scalability it would give the game engine and android application at the cost of a longer and more intensive development process, because libGDX only provides a framework, all the game engine architecture (the entity component system) and logic must be made from scratch [1].

LibKTX

LibKTX is a framework extension of LibGDX which allows for a better development process as a lot of functionality from LibGDX is converted to Kotlin and expanded upon. It is written in the Kotlin programming language [2].

LibGDX Ashley

LibGDX Ashley is a framework extension of LibGDX which allows for better creation of the Entity Component System and is written in Java [3].

Cloud Firestore

Cloud Firestore is a NoSQL based database service created by Google which is mainly used for storing, querying, and synchronizing data from mobile devices and web applications. Cloud Firestore was created to be performance focused/-optimized and fast. It is used in this project for its tightly integrated functionality with Android and Kotlin which made database operations easier than going with a database framework from another service provider. In addition, the project was made knowing that in a final official distribution, the application would use other established database and authentication systems verified and already used by educational institutions which makes cloud Firestore (Firebase) a placeholder until further development. It is describes later, in Section 9.2 how this is implemented, and in Section 6.5 how it was designed.

Firebase Authentication

Firebase was chosen as the storage solution. The whole group had previously worked with Firebase which was a great advantage. Firebase has its own authentication system which was one of the reasons why it was selected. With its authentication system the user can choose what kind of login credentials to be used. For an application that is going to be used for education in the school system there would most likely be login credentials provided from the school. With Firebase authentication it is possible to update the means by which users login at any point. That means the application could be developed with just using anonymous usernames or emails. And then when it is used by a school or some institution, it can be upgraded to use their credentials at that time, for example, the Feide login system NTNU uses. It can also be upgraded to use phone, Google, Facebook, Github or some of the other Firebase approved login methods. Firebase authentication also works across all platforms and it is very secure. It is developed by the same team that created the Google Sign-in, Smart Lock and Chrome Password Manager. Firebase applies Google's internal expertise of managing one of the largest account databases in the whole world. The application needs different roles for their users, as there are going to be teachers and students using the app. Firebase has a system built in for that, but it is a part of their Cloud Functions, and they are behind a paywall. And for that reason it was decided to have an attribute in the user database that distinguishes the user as either a teacher or a student.

6.2 Game Engine Architecture Design

The main architecture of the game engine is structured into an entity component system (ECS). An entity can be thought of as an object (player, NPC, building, wall, rock etc.) placed in the game world and every entity is composed of one or multiple components. A great inspiration for the ECS and learning source for the GDX/KTX library was the Youtube channel Quillraven [4] and the *Libgdx Cross-platform Game Development Cookbook* by Marquez and Sanchez [5]. See Figure A.1 and Figure A.2 in the appendix for how a player entity is created. A component contains various data types and values specific to that component. The systems control which and how the different components data is used and manipulated. The goal of an ECS is to simplify inheritance between classes so that an entity's components can be added or removed dynamically during runtime, which improves readability of the code and scalability of the program.

Creation of the game engine is done by pooling the different systems from the ECS into the game engine. LibGDX provides pooling as a way of adding one or multiple systems to the game engine which is a performance friendly way of making the game engine for Android mobile devices as destroying and creating entities usually is a memory intensive/expensive operation if the systems are not pooled, but with pooling the memory allocation is more efficient.
See the API documentation LibGDX [6]

Using an ECS with LibKTX allows for great customizability, light weightiness, and size of the completed program because only the systems needed for a specific game and or game mode need to be created, instead of relying on huge frameworks e.g. Unreal Engine 4 which provides a ton of unnecessary functionality for this project's size and undertaking. It is also worth noting that LibKTX is royalty free which means that there are no licensing fees that need to be payed with a complete and commercialized product. This was considered when making the decision between which game engine to use for this project, as this thesis and its results would potentially be viewed by investors, and inspected in terms of how feasible it would be to fully release a commercialized product of the "gamification of digital security curriculum" idea.

6.3 Game Design

The main idea for this application was to have multiple mini games the teacher could choose between and input the necessary curriculum the students needed to learn. The player would also be able to play as his/her own configured avatar in the game worlds.

The game design process was inspired by *Using Design Games* by McMullin [7] and the game design document from Grand Theft Auto by Dailly [8] Some key design principles taken from each of these were **Objectives**, **Constraints**,**Success criteria**, **Reward** and **Play** from Using Design Games. From the Grand Theft Auto design document: **Story**, **Game structure** and **Players**.

- **Story** - The game will be set in a fantasy world
- **Game structure** - There will be a separate game world for each game mode. These game worlds are quiz and open world.
- **Players** - Primarily single player, but players must be able to interact with their teachers, so some online futures are expected.
- **Objectives** - The objectives will vary depending on the game mode chosen by the player.
  The quiz game mode will feature a quiz where the player is given a question

and will be presented with potential correct or wrong answers.
The open game world game mode will feature all of a player's teachers in the world which the player can interact with and be given quest objectives to complete.

- **Constraints** - In quiz game mode the player can not choose multiple answers, maximum 4 choices, question and answer text is limited to a reasonable amount.
- **Success criteria** - The quiz game mode success criteria are met when the player answers a question correctly.
  The open world game mode success criteria are met when the player completes a quest objective.
- **Reward** - The player will be rewarded with a score that increases when the success criteria are met. When the player meets a certain score criteria he/she is rewarded with an achievement.
- **Play** - The player will play as their own configured avatar in the game world corresponding with the player's game mode choice. The player will control his/her avatar with touchscreen based input.

6.4 Game Modes

The game is composed of two minigames/game modes: quiz and open world.

Quiz Game Mode

The main game mode is a quiz game. The game mechanics follow the logic of a quiz, where a player will be asked a question in form of text on the screen and presented with different potential answers as world objects they can interact with and choose between. The answer objects also have a text field above them with the answer text. All answers have the potential to be correct or wrong, to give teachers as much creative freedom as possible, they themselves can choose which answers are correct or wrong when they create the quiz. To give the player a feeling of achievement when they answer a question correctly, the player is given a score counter in the top left corner of the screen which increases with the maximum potential amount the teacher has set that question to be. When the player finishes the entire quiz, he/she is given complete feedback on their performance in the form of a list containing the score achieved for each answered question.

The quiz game mode doubles up as two game modes. If the teacher so wishes he/she can write a question as a piece of learning material, then set the max potential score to be 0 and set the answer text as "proceed". This can be combined into a quiz where a player is presented with a piece of information from the curriculum, then the player interacts with the "proceed" object to proceed to the next piece of information or is asked a question related to the previous presented information.

Open World Game Mode

The open world game mode is a world containing all of the teachers belonging to the specific player playing the game. Each teacher appear in the game world with their own unique avatar and name above their head. A teacher act as quest giver whereas the player interacts with a specific teacher and all quizzes made by that teacher are displayed as sign posts in the game world which are quest objects. When a player interacts with one of the quests, the chosen quest is activated and the player are teleported to the quiz game world. This means that the game modes have been switched and the Quiz Game mode is the one being used.

The game engine has been set up to be modular with system pooling. Expanding the game and creating new game modes have been set up to make the process as easy as possible. For every new game mode that is to be made, a new system and new component for that system is added and pooled to the engine.

6.5 Database Design

This section contains the designed structure of what the database for the application needed to consist of.

6.5.1 Define the Objective of the Database

For the application it was necessary to have a database that could store all the data needed about all the students, teachers, courses, games and personal avatars. It therefore needed to contain a list of all users, teachers and students. A list of all classrooms, linked to the teachers, students, and courses. And for the game data, it needed to store all information about the different teaching material input by the teachers, so that when the different minigames are to be played by the students it displays all the appropriate questions and tasks for the specific situation.

6.5.2 Locate and Consolidate the Necessary Data

The database needed to keep track of a specific set of data for all users. Users are stored by their user ID, which is randomly generated as a user is registered. In addition a user also has a name and an email connected to their account. Data needed to be stored, so the application could see if a user was a teacher or a student, and also see what courses each user is a part of, whether it being a teacher or student. There is an achievement system in the application, so each user has a list of achievements, these are also different from students to teachers. Every user has an individual in-game avatar that can be customized. Therefore, the user document had to store data that could tell the program what sort of outfit or appearance the user chose to have in the game.

The classrooms are divided into one document for each classroom. Classrooms are created by a teacher, and the students can join classrooms themselves. Therefore, there needed to be a field storing the teacher's name, and a list storing all the students taking part in the class. The data the document is consisting of needed to include what course it is for, and what grade. Since the same course can be run in different grades, like for example both 2nd and 4th grade can have English or Math, so there needed to be one database entry for each individual classroom. The entry also needed to store data on what year the classroom is active, because the students move grades each year, and the course may change curriculum or some of the contents. Each classroom has its own list of quizzes and minigames in it. So when the teacher makes a new quiz or minigame for the students of that class, it is easily obtainable from the game. The classrooms are divided into modules. The modules are supposed to be like different chapters in each course. In the modules, minigames and theory for the curriculum could be stored.

The minigames and quizzes have their own database collection for each minigame. There needed to be stored data so each minigame can display the right content from the curriculum to the specified students. For example, if a teacher is going to create a quiz for their students, the quiz entry needs to contain information about what course, what grade and what year, so it can be collected from the right classroom. It needed to hold all the questions, and all the answers and also all the data about the scoring of the different questions.

6.6 App-Game Cross Communication

As LibKTX is a cross-platform library, it runs and exists separately from the android application, thus, means to communicate between the app and game had to be implemented in a seamless manner. Three separate methods of communication between application and game are used to establish connection, namely shared preferences, pass through parameter, and file reading/writing. Though this project only utilises an android application for launching games, creating an IOS, or Windows application to launch the game through will work in a similar manner.

For sharing small, consistent segments of data between game and application, LibKTX can communicate with android's shared preferences, a form of reading and writing to a specific file which is easy to access in both application and game engine. Through shared preferences the application and game are able to persistently store 3 particular datatypes, the user's chosen head, the user's chosen body, and the score received during game play. The design of the player is stored in this manner as it needs to always be consistent with the player's design and will change rarely. Adapting this shared preferences system for IOS is also possible using NSUserDefaults, which is important for functional requirement 6 of enabling easy porting to other platform as shown in Section 5.1.1

When loading the open world, the teachers in each of the users classrooms are passed through the parameter used to launch the game, that is done by passing the necessary strings as arguments which can then be accessed in each screen. The argument passing is used for teacher design, and name, as well as game type and game name if the game is launched directly from the classroom. Finally, passing through read/write to file is done for the specific game data, as the game data is fetched from the database and written to file locally, then accessed by the file from local storage using file name passed through the parameter.

Chapter 7

System Graphical Design

7.1 Game Graphical Design

The design and art style of the game is retro-inspired from old 2D games. Players have their own customizable avatar which they can play as in-game. Assets like grass, bushes, trees, rocks, and other objects are low resolution pixel based.

Going for a 2D game in this project makes sense as a 2D game is less performance heavy than going with a game in 3D. It is unreasonable to presume and expect that the majority of the user base for this application will have expensive high-end devices which can handle 3D rendering, not to mention battery life as an application which requires a lot of processing power will eat up more resources i.e. more energy from the battery has to be used which shortens the potential play time. Making the game in 2D will also make the asset creation process less time consuming compared to creating an asset in 3D, i.e. more time can be spent elsewhere in the program development process. See Figure 7.1 for how the potential planned art style of the game will be.

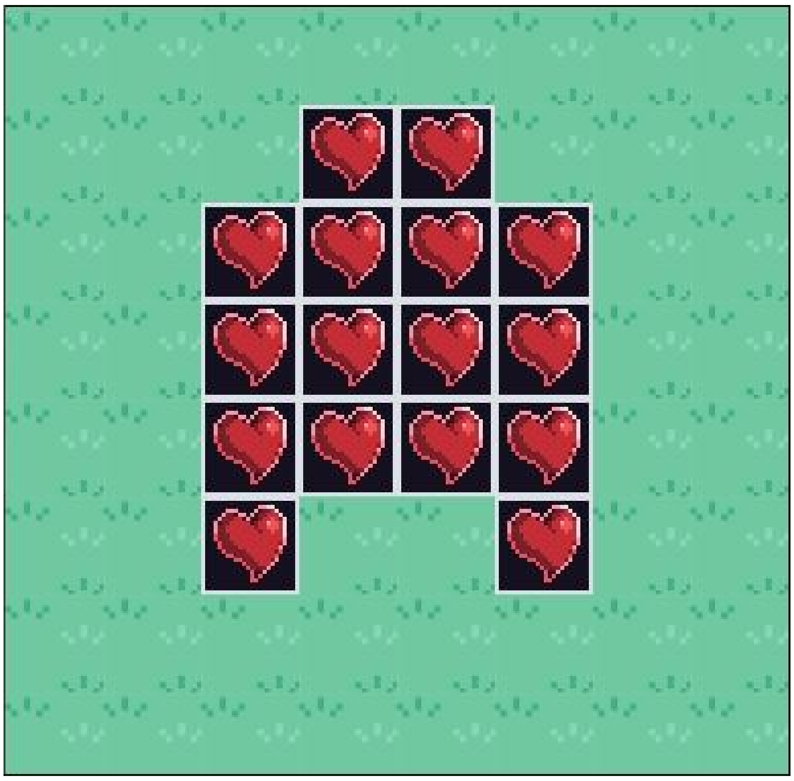


Figure 7.1: Early render of what the game potentially would look like.

7.2 UI Design

At the beginning of the project, some of the first activities to do were to make prototypes of the UI. The early prototypes revolved around the requirements of the app: to offload some of the teacher's work and make learning fun and engaging for students through gamification, so the UI needed to be simple, organized and user friendly. Naturally, there were a lot of proposals to what it would look like, all of which revolved around the main requirements which were taken into consideration.

7.2.1 Profile UI Prototype

To gamify the app, the prototypes of the user profile featured a profile picture, which would be the user's avatar, a total score display to give students direct feedback on their progression and a level display to accompany the user's total score. The idea was to give students a rewarding feedback in the form of an easy to digest summary of a user's progression and achievements. To make the app efficient and easier to use, the prototypes featured several functions in one fragment such as a search bar and editing inside the profile fragment and a friends list for social interaction and a group list for assignments.

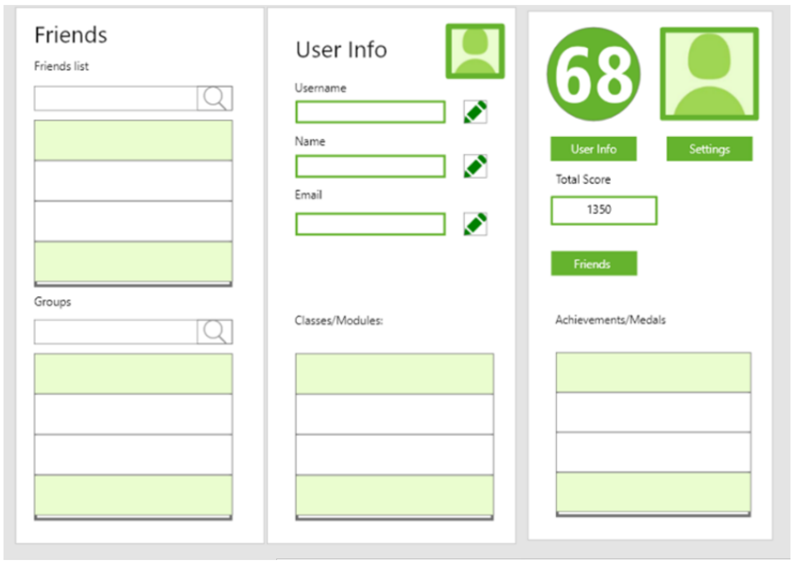


Figure 7.2: This figure shows an early concept of the User info, User Profile, and the discontinued Friends fragment.

7.2.2 Classroom and Module System Prototype

As the classroom and module system were the core of the application, the prototyping followed an iterative design process over the course of one work week, in which the important design principles were compounded into the main idea. The idea was to have a classroom section on the bottom menu that would navigate users to a page containing a list of classrooms as well as the option to join an existing classroom, or create a new classroom if the user was a teacher. Upon clicking the classroom, details from the classroom are loaded into a page with a menu to select a specific view, one would contain announcements from the teacher, another with class leader board, and finally one containing the module list. The module list was intended to be designed more as a scrollable page with graphical elements, with modules represented as nodes in a tree, where users can choose the path they wish to take when working to complete all modules. There were plans to have module-types which would require previous modules to already have been finished, as well as an option to skip past modules if the student felt comfortable enough with the previously unfinished modules. The skip module would fetch random games from each unfinished module in a test with low tolerance for failure, upon passing the skip module, the student would not have to complete the unfinished modules to advance past the skip. The teacher would be able to create new or import existing modules and place them freely on the node tree, and would be able to modify the content inside each module whether created or imported. The modules can be filled with different games that can be launched individually, or played in the sequence designated by the teacher.

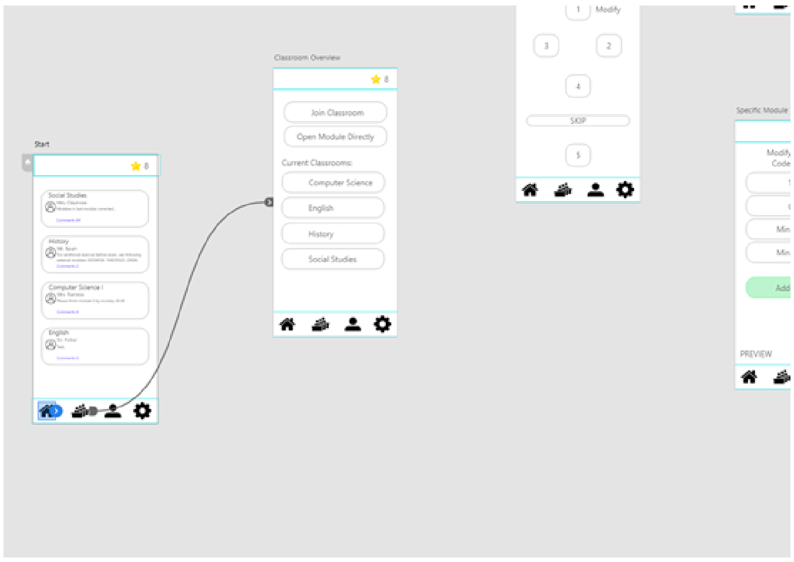


Figure 7.3: Screenshot of first prototype, made in Adobe XD

7.2.3 Navigation

Navigation plays a key role in making the app easy and efficient to use for the end users. Thus, the navigation had to be simple and organized. At the start of development, how the app's navigation should be, was amongst the first things to be considered. A drop-down menu was considered for the app so every fragment of the app would be accessible from one menu which would stay hidden until the user needed it, but ultimately a navigation bar was ideally suited for the task at hand as it is a simplified navigation menu which could be accessed without extra gesturing to make it appear. Additionally, the navigation bar always presents the user with four options. Initially, the navigation bar was planned to be used as an all-in-one navigator between fragments, but it proved to impede the user experience due to frequent miss clicks if the navigation bar was to have more than five options so later prototypes and early alpha would feature three to four buttons for the essential fragments and settings.

Chapter 8

The System Development Process

8.1 Technology and Methods Used

Android SDK

Android SDK is a system development kit for developing applications on the Android platform. The SDK provides functionality for programming with Java, Kotlin, C++, xml and others with extension packages to the SDK e.g. GoLang. It also provides tools for program debugging. The Android SDK supports Linux, Window 7 or later and Mac OS X operating systems. For this project Kotlin was chosen as the primary programming language over Java because Kotlin is trying to solve a lot of problems Java inherently has with its somewhat chaotic nature, its intuitive way of writing code and the null exception problem. Kotlin is being heavily pushed onto developers by Google, and other software companies are switching over. Pinterest and Square are two examples of companies which are currently using Kotlin [9], [10].

Android Studio IDE

Android Studio IDE is an Integrated Development Environment provided by Google and is based on JetBrains IDEA IntelliJ. Android Studio provides a text editor for programming, Gradle building, debugging tools, UI layout editor with Google's standard UI elements e.g. buttons, layouts, and an Emulator which can be configured to emulate different android devices running different OS version of Android. It supports Linux, Window 7 or later and Mac OS X operating systems. Android Studio was the only IDE which was used as it provided all the functionality needed for the making of this project [10].

Gimp

Gimp is a digital image manipulation program, it is free and open source. It supports Linux, Windows and macOS. Gimp was used in this project for editing sprites [11].

Aseprite

Aseprite is a raster graphics editor for Windows, macOS, and Linux. It is free and an excellent tool for making sprites from scratch. Aseprite was used to make custom sprites for the app [12].

Adobe XD

Adobe XD is a vector-based user experience design tool for web apps and mobile apps. It supports macOS and Windows. There are versions for iOS and Android to help preview the result of work directly on mobile devices. Adobe XD was used for its ability to create click-through prototypes which would simulate how the app navigation would work [13].

GDXLiftoff

GDXLiftoff is a setup configuration tool used for LibGDX Gradle projects, which helps to simplify the setup of the initial project files, extensions, and its dependencies [14].

Git

Git is a version control system which is actively used by many developers throughout a system development process. It allows for multiple people to work on the same project at the same time and distribute work done into a repository. People can push the changes they have done to the repository and others can then pull the updated repository to their own local system and work with the updated program code. Git holds a history of all the pushes that are made into the repository which allows for easy backtracking if an unknown error occurs in the program, so that the error can be backtracked to which push it came from and then removed/reverted to a previous version [15].

Git Bash

Git Bash provides an emulation layer of the Git command line for windows operating systems. It was used for pushing, pulling, branching, merging project files and version control in this project [16].

GitHub

GitHub is a web-based platform which is a free service provided for users with a registered account, with optional, paid, premium functionalities. Its primary purpose is to host program code in a repository. It allows for collaboration between people/developers all over the world given that they have an internet connection to use Git for sharing work. Git Bash and GitHub were heavily used throughout the development process of this project [17].

Trello

Trello is a web-based application which provides a hub for organizing project work. It does this by having a card system which members of a Trello project can manipulate i.e. add, move, remove cards and then set which person is currently working on a specific or multiple cards. The cards are organized in lists which makes converting the card system to SCRUM backlog easy. Trello was used in this project for having a SCRUM overview with a product backlog, current sprint log i.e. in progress/currently worked on and completed sprint backlog. This was done to keep track and overview of over which functionality needed to be created and who was working on different parts of the program [18].

Discord

Discord is a Voice over IP (VoIP) communication platform, which is available on Android, iOS, Linux, Windows, Mac OS, and web-browsers. Discord provides options for streaming, sharing content, talking, and writing text live. Discord was used as the main communication platform between group members through the development process of this project. Due to COVID-19 and lockdowns, physical group sessions were deemed too difficult and risky. With the Norwegian government recommending people to work from home if possible, it was the decided by the group to communicate digitally through Discord [19].

Zoom

Zoom is a digital communication platform which allows users to hold video conferences and online voice meetings. It supports Android, iOS, Linux, Windows 7 (and above) and Mac OS X. Zoom was used to communicate with the study supervisor and stakeholders [20].

Microsoft Whiteboard

Microsoft Whiteboard is a digitized whiteboard which allows for one or multiple people to cooperate remotely from their own PC. It is available for Windows 10 and Xbox One. This application was used to draw diagrams and share ideas between the group members of this project [21].

8.2 Development Process and Agile Methods

Scrum

Scrum is an agile system development method; Scrum can be explained by splitting it up into components (note: this explanation is focused on Scrum with software development, the Scrum process itself can be used for other product development processes as well) [22].

Product Owner

The product owner is a person responsible for representing stakeholder(s) who owns the product idea and presents what the stakeholders want in terms of realizing the product idea into a fully developed software product. Throughout the development process the product owner acts as an intermediary between the Scrum team and stakeholders to show what has been developed. This is to make sure the Scrum team develops a solution the stakeholders want in the finished product. The project ideas and problems that need to be solved are put into a product backlog.

Scrum Team

The Scrum team consists of developers, a product owner and a Scrum master. The team is usually built up with three to nine team members.

Scrum Master

A Scrum master is responsible for helping Scrum team members to stay on track and solve problems the product owner has laid out in the product backlog. In addition, the Scrum master helps the team with being successful by preventing distractions, ensuring good team cooperation, and educating team members on the Scrum process throughout the system development process.

Sprint

A sprint is where what work will be solved/completed from the product backlog by the Scrum team. A sprint usually lasts from one day and up to four weeks. Before starting a sprint, the Scrum team picks out tasks from the product backlog and discusses what will be done in the upcoming sprint. When a sprint is completed, a card holding all information on what has been worked on in the sprint is created, and added to the sprint backlog to keep track of which problems/tasks from the product backlog have been solved/completed. See Figure A.26 in the appendix for a Trello sprint backlog overview from an early point in the development process.

Project Work

For this project, the group decided Scrum would be the most agile development method to use. Kanban and Scrumban were considered as optional agile development methods, but the group decided against these because all group members had previous experience using the Scrum development method, and that Scrum would fit the workflow of the development process better than Kanban and Scrumban.

After some time in the development process, the Scrum method was modified to fit better with the group needs. Realistically, the Scrum team consisted of two product owners and two Scrum masters, Andreas Blakli and Vegard O. Årnes took on these roles. The Scrum method generally recommends against this approach with multiple product owners and Scrum masters, but since the group size was small this approached worked out very well. Daily Scrum sessions were held in Discord every workday to discuss current tasks and who was working on a particular problem from the product backlog, to coordinate workflow between team members. Since all team members were in Discord, problem solving, debugging and clarification of a scheduled sprint's log were easy to coordinate and solve. Work hours were scheduled between 09:00 to 16:00, Monday to Friday. Every Monday, a 30 minute session with the clients and study supervisor was held, to discuss progress, priorities, and product backlog.

8.3 Debugging Methods Used

Rubber Ducking

Rubber ducking is a debugging method that seeks to solve logical errors in the program code by explaining how an entire/section of code from the program works, what it seeks to accomplish and what the problem is to a rubber duck. The process of explaining a problem out loud to a rubber duck often helps a developer to understand what and where the problem is located [23].

Rubber ducking was extensively used throughout the development process and group members found this an efficient way of debugging.

Pair Debugging

Pair debugging is much the same as pair programming, it involves two programmers that work together to correct a logical error in the program code.

Pair debugging was less commonly used, as pair debugging required another person to stop working on their own code and help out with the problem code.

Chapter 9

System Implementation

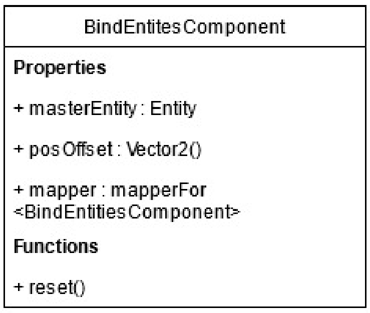
9.1 Game Engine Implemented System Architecture

9.1.1 The Components of the ECS

This section covers how the component's in the entity component system were implemented from the previously discussed game engine architecture in Section 6.2. How and what a specific component is used/does is covered in Section 9.1.2

BindEntitiesComponent

Contains vector for position offset for slave entity from master entity. See Figure 9.1


BindEntitesComponent

Properties

+ masterEntity : Entity

+ posOffset : Vector2()

+ mapper : mapperFor <BindEntitiesComponent>

Functions

+ reset()

Figure 9.1: UML class BindEntitiesComponent

InteractableComponent

Contains variables related to the victory condition of the game, the score value for a question, variables related for telling the program what sort of intractable entity type the entity is e.g. is it a teacher or quest entity. And name of the quiz if it is a quest entity. See Figure A.3 in the appendix for figure of the Interactable-Component class.

MovementComponent

Contains velocity vector (speed) for entities with this component. See Figure A.4 in the appendix for figure of the InteractableComponent class.

OrientationComponent

For orientation of entities i.e. up, down, left, right, e.g. player is moving left, then the sprite must be adjusted so that the sprite is facing in the same direction as the player is moving. Note: this component is used but has no visual effect in-game because all of the used sprites have only one direction, but it is fully possible to later expand upon this so sprites can be oriented. See Figure A.5 in the appendix for figure of the OrientationComponent class.

PlayerComponent

Contains data relevant to the player, total score gained, player health, player user-name etc. See Figure A.6 in the appendix for figure of the PlayerComponent class.

QuizComponent

Contains data relevant for the quiz game. See Figure A.7 in the appendix for figure of the QuizComponent class.

QuizQuestComponent

Contains data relevant to the quest and teacher entities, name of teacher and whether to show all available quizzes that belongs to a given teacher. See Figure A.8 in the appendix for figure of the QuizQuestComponent class.

SpriteComponent

Contains the sprite texture and its color values i.e. RGB and alpha. See Figure A.9 in the appendix for figure of the SpriteComponent class.

TextComponent

Contains data for text rendering. See Figure A.10 in the appendix for figure of the TextComponent class.

TransformComponent

For transforming entities with position vector and size vector i.e. moving the entities on screen. See Figure A.11 in the appendix for figure of the InteractableComponent class.

9.1.2 The Systems of the ECS

This section covers the implemented system's of the entity component system. See Section 6.2 for how the architecture of the ECS were designed and Section 9.1.1 for what a specific component contains.

BindEntitiesSystem

BindEntitiesSystem is responsible for binding an assigned entity (slave) to a master entity and transforming the transform pos (position) vector of the slave entity to match the master entity's pos vector plus the offset value so the entity is correctly offset from its master. This system is used primarily for the player entity as the player entity is built up by two entities which are the head and body entities. Using this system for other entities is not a problem as it is designed to work with all entities that have a sprite and a pos vector. It interacts with the BindEntitiesComponent and TransformComponent. See Figure A.12 in the appendix for figure of the BindEntitiesSystem class.

InteractableSystem

InteractableSystem is responsible for collision detection, if necessary, move player back away from the object it collided with, destroying specific entities i.e. quest, teacher, question, answer entities on collision, updating the player score based on correct/wrong answer. The components this system interacts with are the InteractableComponent, TransformComponent, PlayerComponent, TextComponent, QuizComponent and QuizQuestComponent. See Figure A.13 in the appendix for figure of the InteractableSystem class.

MovementSystem

MovementSystem is responsible for moving all the created entities with the MovementComponent and setting the world bounds so the player cannot move outside the game world. The components this system interacts with are the TransformComponent and MovementComponent. See Figure A.14 in the appendix for figure of the MovementSystem class.

PlayerInputSystem

PlayerInputSystem is responsible for handling player input. Based on player input it will manipulate the direction vector and orientation direction from the OrientationComponent. Which is then used in the MovementSystem to move the player entity. The components this system interacts with are the TransformComponent, OrientationComponent and PlayerComponent. See Figure A.15 in the appendix for figure of the PlayerInputSystem class.

QuizSystem

QuizSystem is responsible for main game logic of the quiz game, see Section 6.4 for how the quiz game mode was designed. It reads a specified quiz from file, then creates the quiz entities for the question and answers dynamically based on which questions and answers that belong together. When a player has given an answer, the InteractableSystem will change the playerHasAnswered bool to true in the QuizComponent and the next set of entities for question and answers is created. Once the player finishes the quiz, the score is saved and written to disk for use in other parts of the application and the game is exited. The component this system interacts with are the QuizComponent. See Figure A.16 in the appendix for figure of the QuizSystem class.

QuizQuestSystem

QuizQuestSystem is responsible for main game logic of the open world game, see Section 6.4 for how the open world game mode was designed. When a player interacts with a teacher entity, the system creates quest entities which are displayed as a signpost in-game for all quizzes that belong to that specific teacher the player interacted with. These signpost quests are dynamically removed by the InteractableSystem when a player interacts with another teacher entity to avoid overlapping entities. When a quest entity is interacted with by a player, the name of the quiz is passed to the quiz game and then the game screen is switched to the quiz game. This system interact with the QuizQuestComponent. See Figure A.17 in the appendix for figure of the QuizQuestSystem class.

RenderSystem2D

RenderSystem2D is responsible for rendering all entities with a 2D sprite texture, the viewport from the game engine is passed to the system and set inside the batch. The camera position is updated based on the player entity transform vector position. This ensures that the camera will follow the player entity and always stay centered. This system interacts with the SpriteComponent and TransformComponent. See Figure A.18 in the appendix for figure of the RenderSystem2D class.

RenderSystemText2D

RenderSystemText2D is responsible for rendering text to the screen. It has its own custom viewport so that characters in the text are sized appropriately. This is necessary because the default game viewport is scaled to the game world size, which is 9x16 units. Trying to render text with this viewport size makes it huge and blurry, so the custom viewport is set to 1080 x 1920 which makes the text a lot smaller and clearer because of the higher viewport size. To account for the offset between the different viewports sizes, the vector position coordinates are scaled up so that text entities will be positioned correctly according to the game world coordinates. The viewport camera will follow the player entity based on the player entity's vector coordinates. To keep the text entities placement static, their position is updated and offset by the camera's current position so they will always stay in their original set position. This system interacts with the TextComponent. See Figure 9.2 for RenderSystemText2D class figure.

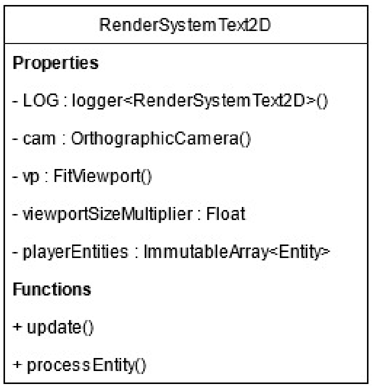

RenderSystemText2D
Properties
- LOG : logger<RenderSystemText2D>()
- cam : OrthographicCamera()
- vp : FitViewport()
- viewportSizeMultiplier : Float
- playerEntities : ImmutableArray<Entity>
Functions
+ update()
+ processEntity()

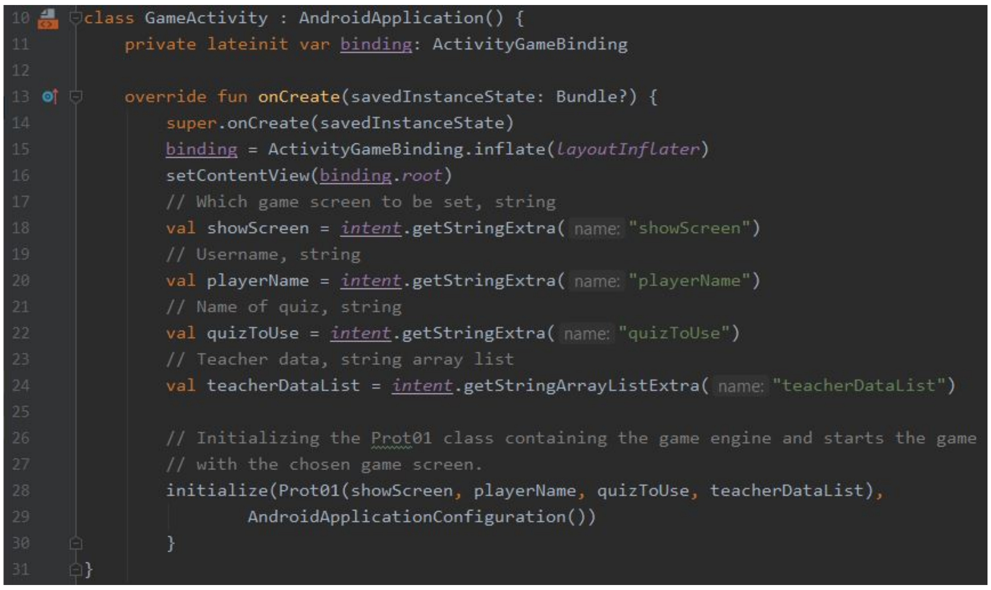
Figure 9.2: UML class RenderSystemText2D

9.1.3 Game Screens and Classes

This section addresses the game screens and relevant classes to fully create the game engine from the systems explained in Section 9.1.2 and game modes individual design explained in Section 6.4.

GameActivity

GameActivity class creates a new Android Activity and initializes the Prot01 class which is where the game engine is created and the chosen game is started. See Figure 9.3 for program code.

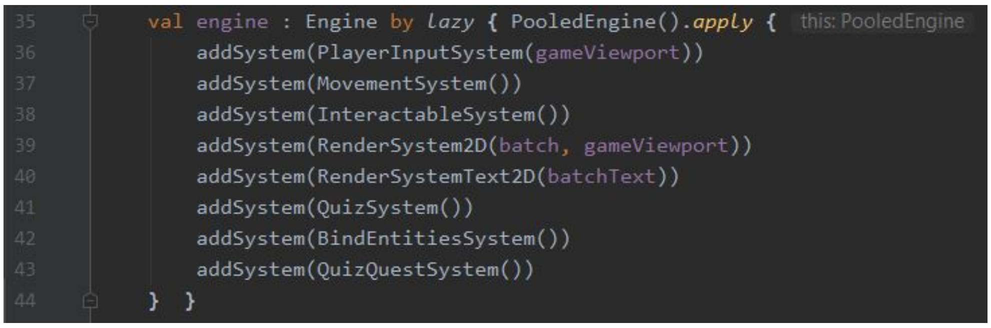


```
class GameActivity : AndroidApplication() {
    private lateinit var binding: ActivityGameBinding

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        binding = ActivityGameBinding.inflate(layoutInflater)
        setContentView(binding.root)
        // Which game screen to be set, string
        val showScreen = intent.getStringExtra( name: "showScreen")
        // Username, string
        val playerName = intent.getStringExtra( name: "playerName")
        // Name of quiz, string
        val quizToUse = intent.getStringExtra( name: "quizToUse")
        // Teacher data, string array list
        val teacherDataList = intent.getStringArrayListExtra( name: "teacherDataList")

        // Initializing the Prot01 class containing the game engine and starts the game
        // with the chosen game screen.
        initialize(Prot01(showScreen, playerName, quizToUse, teacherDataList),
                AndroidApplicationConfiguration())
    }
}
```

Figure 9.3: GameActivity program code

Prot01

Prot01 is the class responsible for creating the game engine with its pooled systems from the ECS, adding and setting the correct game screen based on which game the user chose in the main Android application. It takes three strings, and a list of strings in the constructor. These are: which game screen to create, username of player, name of the locally stored quiz and a list containing teacher data. The create() method adds and sets the game screen based on the showScreen string in a when statement, it defaults to OpenWorldScreen and passes on the respective values each game screen requires in their constructor. See Figure 9.4 for how the systems are pooled and Figure A.19 in the appendix for class figure.



```
    val engine : Engine by lazy { PooledEngine().apply {  this: PooledEngine
        addSystem(PlayerInputSystem(gameViewport))
        addSystem(MovementSystem())
        addSystem(InteractableSystem())
        addSystem(RenderSystem2D(batch, gameViewport))
        addSystem(RenderSystemText2D(batchText))
        addSystem(QuizSystem())
        addSystem(BindEntitiesSystem())
        addSystem(QuizQuestSystem())
    } }
```

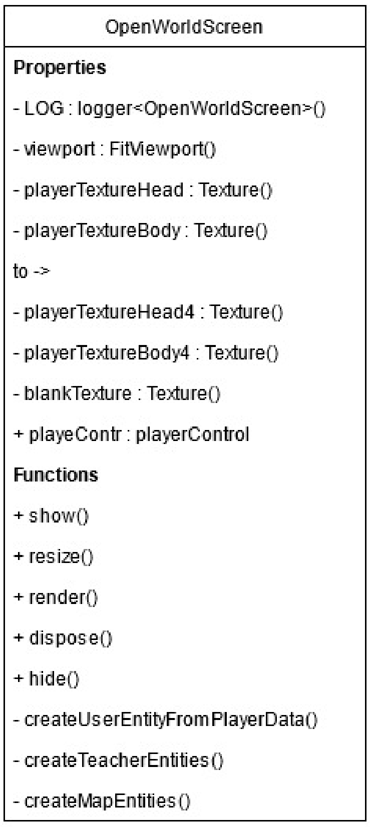
Figure 9.4: Engine pooled systems

Abstract screen

Abstract screen: Is an abstract class that holds the game Prot01 class and its pooled systems, game engine and sprite batch, this abstract class is used for the OpenWorldScreen and QuizScreen since each of these game screens requires all of the previously mentioned variables, and values. See Figure A.20 in the appendix for figure of the class.

OpenWorldScreen

OpenWorldScreen class constructor takes game class Prot01, a list of strings (of teacherData), username and inherits from the Abstract screen class. When the screen is shown the show() method is used to activate three functions, these are: createMapEntities(), createUserEntityFromPlayerData(), and createTeacherEntities(teacherDataList). createMapEntities() reads a text file from disk. This file contains the map layout in which each row and column are iterated through. Based on what value each element has, the appropriate entity is created with its corresponding components. createUserEntityFromPlayerData() creates the player entity with the corresponding sprites the user configured their avatar to look like in the app. It does this by reading the playerData+username.xml file containing the strings with which sprites the user has chosen. The player is built up by two entities: playerEntityBody and playerEntityHead. The playerEntityBody is composed of six components, these components are: TransformComponent, MovementComponent, GraphicComponent, PlayerComponent, OrientationComponent, and TextComponent. The playerEntityHead is composed of the TransformComponent, MovementComponent, GraphicComponent, BindEntitiesComponent, and OrientationComponent. createTeacherEntities(teacherDataList) function creates the teacher entities with the teacher(s) configured avatar(s) from the data provided in the teacherDataList. A teacher is built up by two entities, the teacherEntityHead and teacherEntityBody. The teacherEntityHead is composed of the TransformComponent, GraphicComponent, TextComponent, InteractableComponent, and QuizQuestComponent. The teacherEntityBody is composed of the TransformComponent, GraphicComponent, InteractableComponent, and QuizQuestComponent. See Figure 9.5 for figure of the class.


OpenWorldScreen

Properties

- LOG : logger<OpenWorldScreen>()
- viewport : FitViewport()
- playerTextureHead : Texture()
- playerTextureBody : Texture()

to ->

- playerTextureHead4 : Texture()
- playerTextureBody4 : Texture()
- blankTexture : Texture()

+ playeContr : playerControl

Functions

+ show()
+ resize()
+ render()
+ dispose()
+ hide()

- createUserEntityFromPlayerData()
- createTeacherEntities()
- createMapEntities()

Figure 9.5: UML class OpenWorldScreen

QuizScreen

QuizScreen class constructor takes game class Prot01 username and inherits from the Abstract screen class. The QuizScreen is built up like the OpenWorldScreen, when the show() method is used there are two functions which are called, the createMapEntities() and createPlayerEntity(). The createMapEntities() does the same as explained above in the OpenWorldScreen, the same goes for createPlayerEntity(), but the key difference here is that the player also has the QuizComponent. See Figure A.21 for figure of the class.

HelperFunctions

HelperFunctions is a class created to help against code duplication. Functions that are used multiple places throughout the game code's logic is to be placed in this class. For now it contains chopString(str: String, maxLength: Int) which takes a string "str" and an integer "maxLength" as input and returns the chopped string and how many times it has been chopped. This function is used by text components and is needed because when using LibGDX draw method for text, the text will be drawn from the starting position vector, to the right and down. So the string is chopped so it will not go out of the horizontal screen boundary and offset vertically so it moves upwards each time it has been chopped. See Figure A.22 in the appendix for figure of the class.

PlayerControl

The playerControl class was created to give the player a way to interact with the game world trough an action button. The class uses a stage, that handles the viewport, which in turn controls the coordinates used within the stage and sets up the camera used to convert between stage coordinates and screen coordinates. This is needed to render the action button on the screen. The class uses two near identical functions, touchUp and touchDown, which checks if the button is pressed or not and sets the correct boolean value accordingly. The draw() function will draw the button on the screen when called.

QuizInfo

The QuizInfo class purpose is to display the results after the quiz has been created as well as to give a UI element which allows user to traverse back to the application from the game. This is done through receiving the mutable list of points gathered from answering the questions within the quiz. The list is passed from the quiz component into the Quiz info constructor upon completing the game. Upon creation the class renders a forest green background that fills the window, then populates a table inside a stage with the point list and the exit button.

9.1.4 Systems Interaction

When a system in the pooled engine sees an entity containing a specific component that system will start using its implemented logic e.g. when the QuizSystem sees an entity with the QuizComponent it will activate and start the logic for the quiz game mode. See the figure Figure 9.6 for a collapsed class diagram of the ECS. See Figure A.25 in the appendix for the fully exploded class diagram.

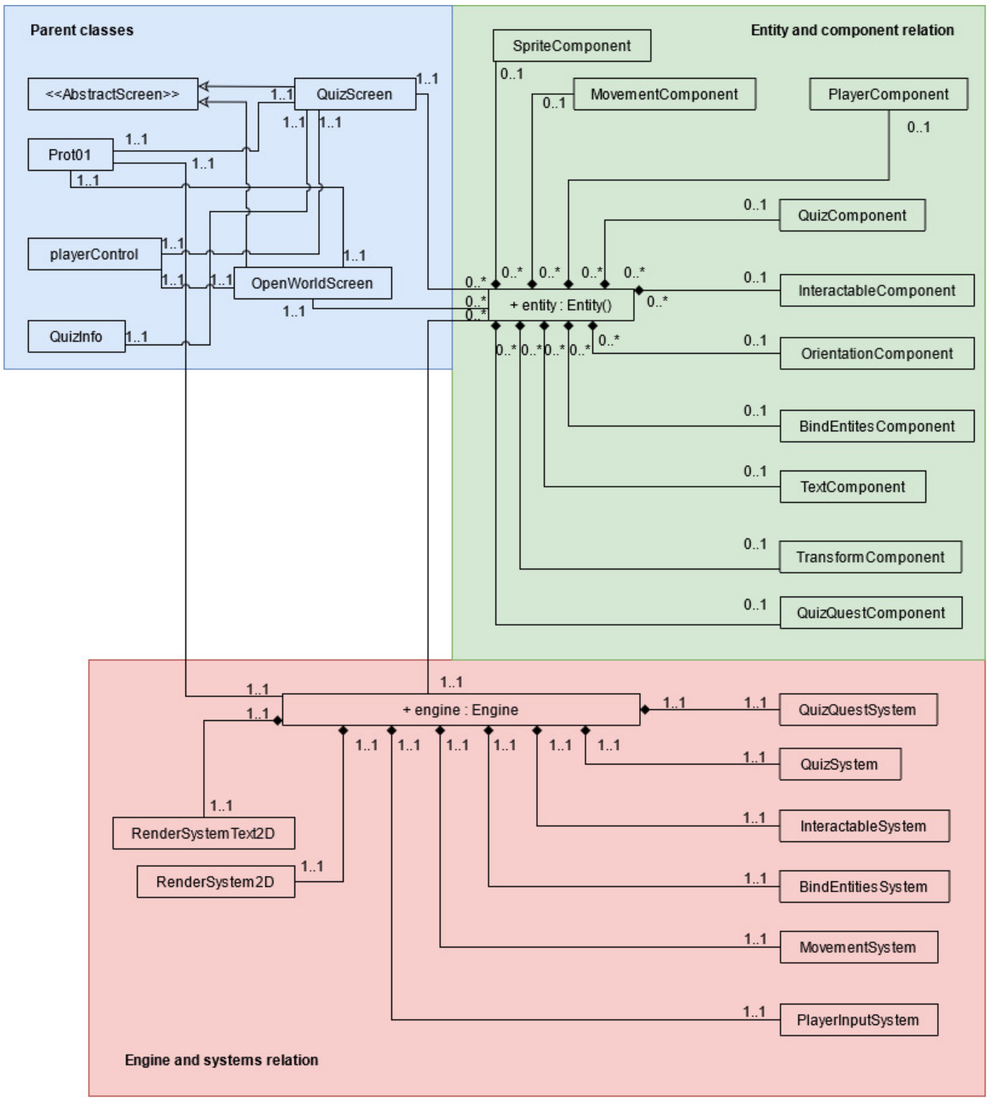


Figure 9.6: Collapsed UML Class Diagram of the ECS and parent classes

The Figure 9.7 is a system sequence diagram which represents the involved systems when a player interacts with a teacher in the open world. When a specific teacher is interacted with, the quest entities/quizzes belonging to that teacher are created and displayed in the world. Descriptive text is used instead of function names, the main function call is processEntity() for the systems, as this function processes all of the components belonging to that specific system on every tick.

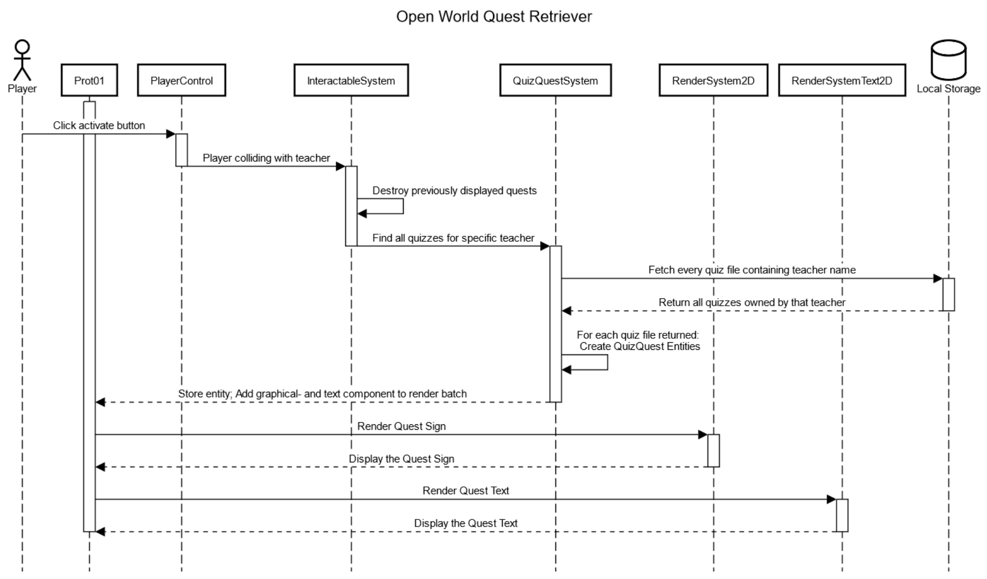

Open World Quest Retriever
Player
Prot01
PlayerControl
InteractableSystem
QuizQuestSystem
RenderSystem2D
RenderSystemText2D
Local Storage
Click activate button
Player colliding with teacher
Destroy previously displayed quests
Find all quizzes for specific teacher
Fetch every quiz file containing teacher name
Return all quizzes owned by that teacher
For each quiz file returned:
Create QuizQuest Entities
Store entity; Add graphical- and text component to render batch
Render Quest Sign
Display the Quest Sign
Render Quest Text
Display the Quest Text

Figure 9.7: System sequence diagram of open world quest retrieval

9.2 Database Implementation

The figure below, Figure 9.8, shows the structure of the database as it is in the project currently. It is not very large, with only four tables. As stated in Section 6.5, there was going to be a database entity for each of the mini games in the application, but as the quiz was implemented into the open world game, it was not needed to have more than one table for the quiz. The classroom entity can have zero to many quizzes stored in it, and individual quizzes can exist in multiple classrooms simultaneously. The classrooms also have two relationships with the user's entity because a user can either be student or teacher. A student user can be in zero or many classrooms, and a classroom can have zero to many students. A teacher user can be the teacher for zero to many classrooms, and a classroom can only have one teacher. The modules ended up being an entity of its own, as that made more sense than including it somewhere into the classroom's table. It is also connected to the quiz table, as quizzes are to be created into their own modules, and the classrooms are storing all the quizzes for all the modules in the classroom. There can be zero to many modules in each classroom, and the modules can exist in multiple classrooms. Each module can have zero to many quizzes, and each quiz can exist in multiple modules. Even though the database ended up containing only one type of game-data table, namely quizzes, the system has been designed so it is very easy implementing new mini games into the application and database. They would all communicate with the database in the same manner so it would only need to be added a new collection, and the database functionality in the app would be easy to manipulate to communicate with the new collection.

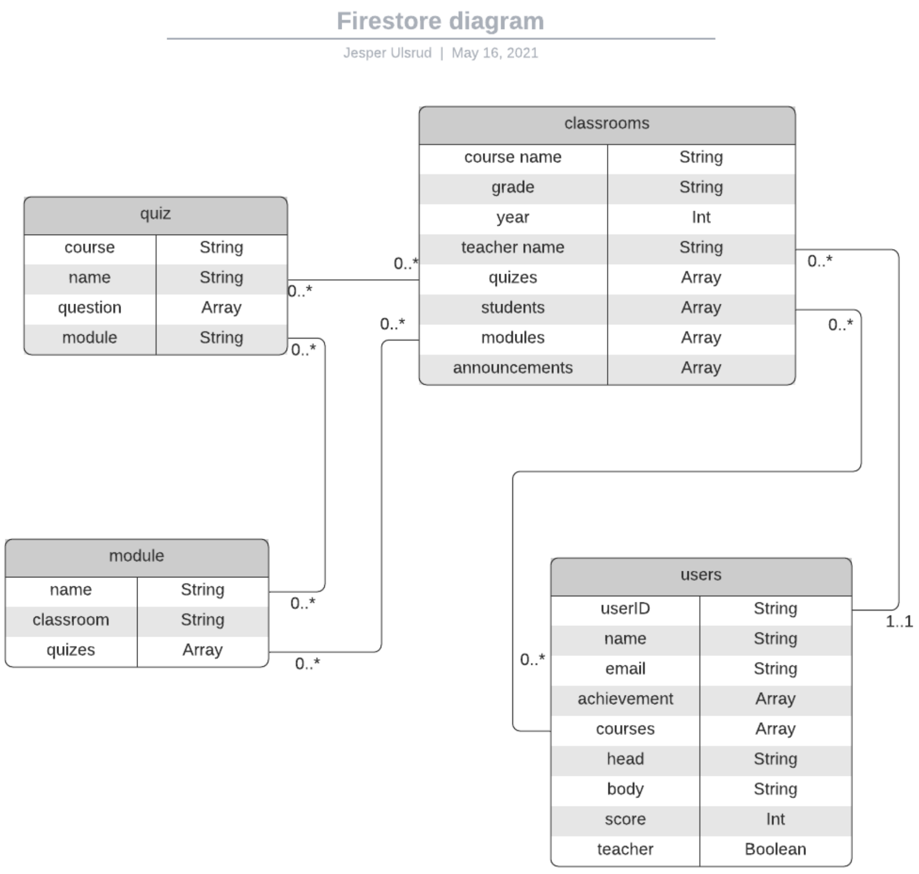


Figure 9.8: Diagram showing the structure of the database

9.2.1 Getting User Data from Firestore

For getting the data of a logged in user from Firestore there were some complications. Since making the queries and actually getting that data from the database takes some time, it needs to be done using coroutines in Kotlin. A coroutine is a concurrency design pattern that can be used on Android to simplify code that executes asynchronously. Since Kotlin does not let the data be fetched and returned instantly, coroutines is what is supposed to be used. The problem is that to be able to use the coroutines, use of a suspended function is required, and to use a suspended function the whole class needs to inherit the suspended functionality. And the way the navigation system is set up in the application itself regarding all the fragments, it needs to inherit from the Fragment class. Unfortunately, an effective solution for this was not found, it proved to be difficult to solve and it caused additional obstacles. But one of the main goals of the application was to make the classrooms and modules as interactive, modular and modifiable as possible for both the users. Therefore, it was decided to keep the fragments the way they were and find another way of getting the data from Firestore.

The solution ended up being that once logged into the application, all necessary data about the user like for example its courses, classrooms, teacher, students, and score, is read into a dedicated user object. So instead of having to make a query to Firestore every time data from a user was needed, it is made one query once the user is logged in, and the data is stored in an object which can easily pass out the data for the user as needed wherever in the application. Passwords or anything that could prove to be a potential threat, are not stored either in the Firebase user database or in the application [24].

9.3 UI Implementation

Much like early prototypes of the app, the user interface has been designed with large buttons labelled appropriately for easy navigation and provides lists in fragments where appropriate. The fragments appear with less functions than initially planned to avoid overwhelming the user with too many options than needed. For each user, lists have been implemented in their profile to give them quick access to the modules they have responsibility of if they are a teacher, or a list of classrooms to attend for students. The use of lists keeps the interface simple and clearer as the list was implemented with scroll function so it keeps the interface from displaying information that may not be needed at the time while also keeping all their modules and classrooms in one place at the bottom of the screen. The UI was planned to have a fragment dedicated to friends and colleagues with a way to send messages or notify the students of a new classroom/module. The fragment would have featured a list of friends and groups the user is currently in. However, this was scrapped as it was not needed to fulfil the requirements for the app, and the app itself has been developed around a 'single-player' experience so a friends list or a group list would not be useful. The App was also supposed to have no edit fragment for their profile as the edit function would be accessed through pressing buttons next to the user information and avatar, this was later removed as having its own fragment for editing profiles would remove any unnecessary miss-clicks and organize the app's functions so it would only be accessed when needed. See the figures: A.32 A.30

9.3.1 Gamification

To fulfil the requirement of making learning fun and engaging for students, the student's user profile fragment features an achievement list, a text view that displays their total accumulated score which is saved to the database, and a level display that increases depending on the user's score. Achievements functions as unlocks after certain criteria are met and will be added to a user's achievement list once those criteria are fulfilled. This serves to give the students feedback on their performance in the modules as well as reward for performing well. The inclusion of a customizable avatar functions as another layer of reward, where being able to show one's achievements through their avatar is a reward. Therefore, the avatar is constructed with two parts: a head and a body, which can be used to mix and match different head and body parts to the user's desire. This can also serve as a goal for users to strive for to unlock new parts for their avatar or for completion of achievements. Teachers are also given an avatar which is displayed within the open world game mode, but it does not function as gamification of their side of the app, rather as a representation of the responsible teacher of a classroom which the students may interact with in the game world. See the figure: A.33

9.3.2 Navigation

The implementation of the app's navigation is split into two parts: one for the user's profile, information, edit, and the other part is for creation of modules and quizzes. The home screen features a button to the open game world. Both parts of the app are always accessible through the navigation bar which has four buttons navigating to home, modules/classrooms, profile, and settings, respectively. The two parts are also further divided into a sub-home fragment which is where the navigation bar buttons lead to. The user's profile fragment includes buttons to directly access key features of the app; the user's information, a way to edit their avatar which is also their in-game representation. To avoid confusion, these buttons are labelled with the name of the fragment they navigate to. For the classroom part of the app, the teachers can make new classrooms with modules containing quizzes, and students can join these classrooms. The navigation bar has been implemented with icons clearly indicating where it navigates to. The app also has a side menu which appears when swiping to the right. See the figure: A.34.

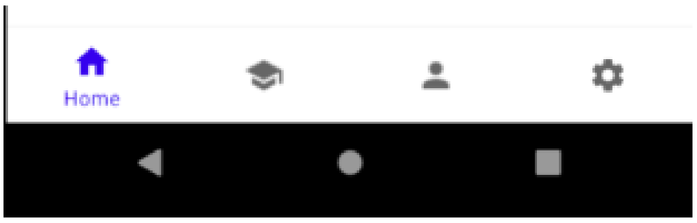


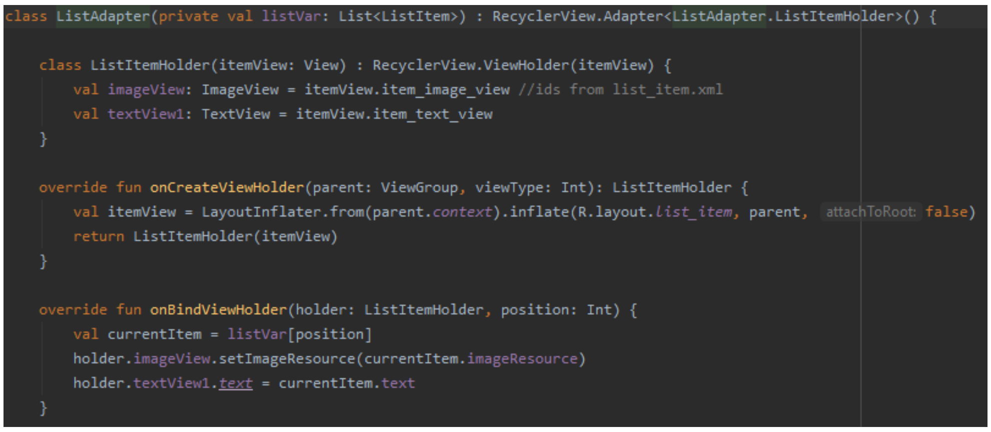
Figure 9.9: Application navigation bar

9.3.3 Database Querying from Fragment

Implementation of the UI fragments is mostly straightforward. Generally, the app's buttons use setOnClickListeners to preform actions when pressed. A lot of them simply serve as navigation to different fragments, whose names are labelled on the respective buttons. However, the UI fragments also use Firebase queries to gather and display user relevant information on screen. This is accomplished by functions inside a User object which returns saved variables inside said object, and these variables are queried from the Firebase database using the DBObject. Thus, username, score, relevant modules, and achievements will change depending on which user is logged on the app.

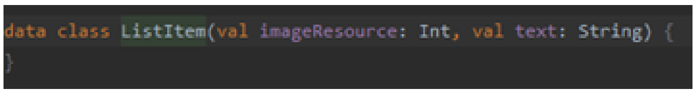
9.3.4 Lists

The lists are implemented as RecyclerViews which use an adapter class to display the items of the list onto it, as well as a class for list items that acts as a template for which the adapter can put in information. This class does this through a function, onBindViewHolder, which assigns text and image to the cardview of an object in a list depending on its position inside that list and onCreateViewHolder which uses an xml file. What the adapter can put in the cardview is decided by the listItem class which the class uses in its constructor. This makes it simple for developers to edit what type of information a list needs to have as editing on the listItem class will make lists that use it display new information depending on what was changed on it. Another function will use get functions from User object to get the user's classrooms, achievements or any other array from the database and dynamically create a list to display the queried information. This keeps the interfaces from being hard coded for each user and will instead always display information that is relevant for the current fragment.



```kotlin
class ListAdapter(private val listVar: List<ListItem>) : RecyclerView.Adapter<ListAdapter.ListItemHolder>() {

    class ListItemHolder(itemView: View) : RecyclerView.ViewHolder(itemView) {
        val imageView: ImageView = itemView.item_image_view //ids from list_item.xml
        val textView1: TextView = itemView.item_text_view
    }

    override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ListItemHolder {
        val itemView = LayoutInflater.from(parent.context).inflate(R.layout.list_item, parent,  attachToRoot: false)
        return ListItemHolder(itemView)
    }

    override fun onBindViewHolder(holder: ListItemHolder, position: Int) {
        val currentItem = listVar[position]
        holder.imageView.setImageResource(currentItem.imageResource)
        holder.textView1.text = currentItem.text
    }
```

Figure 9.10: Adapter class

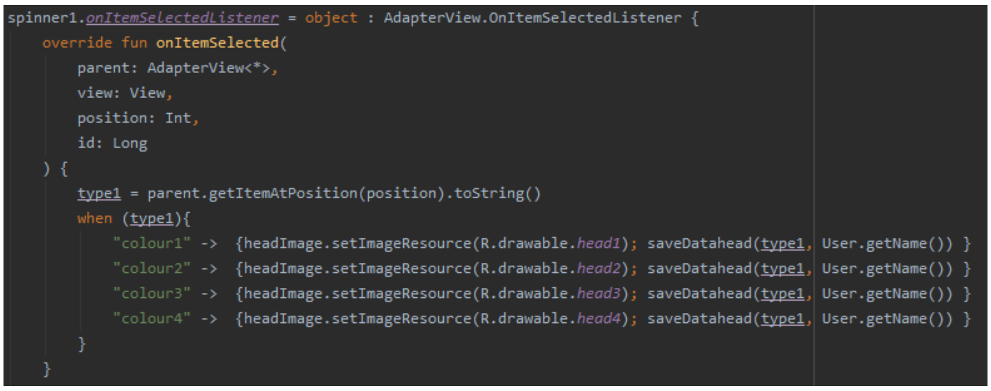


```kotlin
data class ListItem(val imageResource: Int, val text: String) {
}
```

Figure 9.11: ListItem class

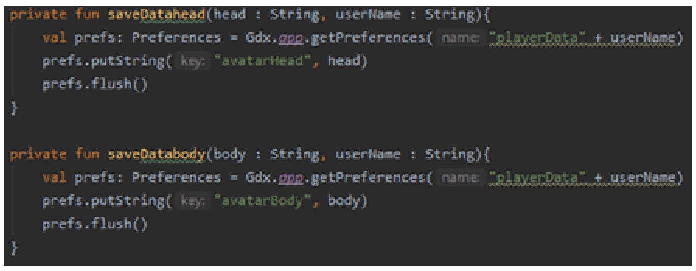
9.3.5 User Avatar

The avatar selection however is done locally via GDX files. The implementation of a GDX file system, which acts as a write to and read from the shared preferences files, is used to store local data such as which avatar parts the user had picked. The selection itself is done with spinners in the editProfileFragment. Once the desired avatar has been finished, the editProfileFragment will use save functions to write to the shared preferences files, through LibGDX, which parts were used under a keyword. The files created are labelled with the username of a user which distinguishes what files to use when using the get functions for the avatar selection, as the username becomes an argument in the get function. These functions are used for other fragments that also display the user's avatar. To avoid having the app rely on locally created files, which becomes a problem if a different user logs into the same device and their avatar selection has not been saved on said local files, the avatar selection is also saved to the database, so the necessary information is queried and written to the shared preferences files on log-in.



```
spinner1.onItemSelectedListener = object : AdapterView.OnItemSelectedListener {
    override fun onItemSelected(
        parent: AdapterView<*>,
        view: View,
        position: Int,
        id: Long
    ) {
        type1 = parent.getItemAtPosition(position).toString()
        when (type1){
            "colour1" -> {headImage.setImageResource(R.drawable.head1); saveDatahead(type1, User.getName()) }
            "colour2" -> {headImage.setImageResource(R.drawable.head2); saveDatahead(type1, User.getName()) }
            "colour3" -> {headImage.setImageResource(R.drawable.head3); saveDatahead(type1, User.getName()) }
            "colour4" -> {headImage.setImageResource(R.drawable.head4); saveDatahead(type1, User.getName()) }
        }
    }
```

Figure 9.12: Spinner functions

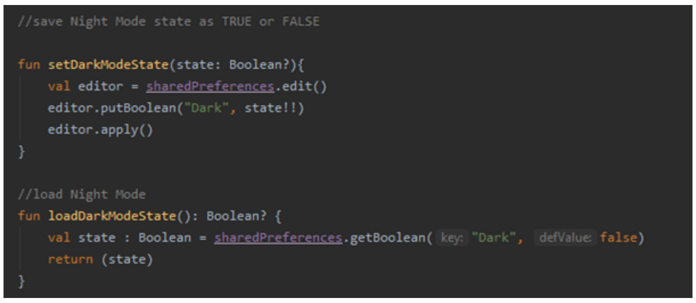


```
private fun saveDatahead(head : String, userName : String){
    val prefs: Preferences = Gdx.app.getPreferences( name: "playerData" + userName)
    prefs.putString( key: "avatarHead", head)
    prefs.flush()
}

private fun saveDatabody(body : String, userName : String){
    val prefs: Preferences = Gdx.app.getPreferences( name: "playerData" + userName)
    prefs.putString( key: "avatarBody", body)
    prefs.flush()
}
```

Figure 9.13: Avatar save and get functions

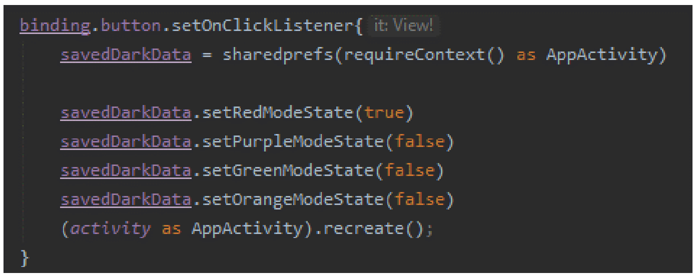
9.3.6 Settings

The settings fragment is used to change the colour and light mode of the app itself. The user may change between green, red, purple orange or default (blue) themes, and this will persist into dark mode should the user want to use that. The settings fragment uses a different class called sharedprefs which, like its name implies, uses shared preferences files to change the theme of the app. This is much like how the usage of LibGDX works since both uses shared preference files, but the class does it via the editor interface which is a built-in method used to modify values in a shared preferences object. Once a theme has been chosen, the settings class uses the editor to write Boolean true or false to the shared preferences files. Upon starting the app, the AppActivity class will check the shared preference file for which theme to use and apply the theme. Admittedly, this was challenging to achieve because the app cannot change themes during run-time. This is solved through restarting the app so every button will restart the app once it is pressed, which will then make AppActivity go through the shared preference file and check which theme to use.



```
//save Night Mode state as TRUE or FALSE

fun setDarkModeState(state: Boolean?){
    val editor = sharedPreferences.edit()
    editor.putBoolean("Dark", state!!)
    editor.apply()
}

//load Night Mode
fun loadDarkModeState(): Boolean? {
    val state : Boolean = sharedPreferences.getBoolean( key: "Dark",  defValue: false)
    return (state)
}
```

Figure 9.14: Usage of shared preferences files



```
binding.button.setOnClickListener{ it: View!
    savedDarkData = sharedprefs(requireContext() as AppActivity)

    savedDarkData.setRedModeState(true)
    savedDarkData.setPurpleModeState(false)
    savedDarkData.setGreenModeState(false)
    savedDarkData.setOrangeModeState(false)
    (activity as AppActivity).recreate();
}
```

Figure 9.15: Settings functionality

9.4 Classroom and Module System

9.4.1 Versatility of Gamification

As previously stated in Section 5.1, the application needed to provide the means for teachers to portray the content of their curriculum regardless of student age or course type. However, the classroom module system achieves this albeit in a limited fashion. The finished product of the module system allows teachers to create their classrooms with games that can be grouped into groups called "modules", and allows users to launch each separate game. The limitation of the service is the lack of variation in gamification. The application classroom system only provides one game type in which the teacher can provide an unlimited amount of title text and up to four answers for each title, as well as points gained for answering the designated correct answer. The finished game works primarily as a quiz but can also function as a means for teachers to provide informational text students can read in the game world. Nonetheless, the implemented method for launching games within the module has the potential to launch any game type as long as the game mode is its own LibGDX screen. The only adjustment necessary for the application to launch any newly created game mode would be to create a means for the teacher to input the data which is to be gamified, and thereafter create a collection for the new game mode within the database and allow the new datatype to be passed from application to game similarly to solution already present in the quiz game mode system. This aspect of the application was therefore developed in accordance with non-functional requirement 2: modularity, defined under Section 5.1.2

9.4.2 Decisions Surrounding the Data

The architecture for the classroom and module system within the application layout relies heavily on the RecyclerView functionality present within the Android API. It was chosen due to its performance- and battery efficiency when handling a potentially endless amount of frequently updated items in a list, while also being flexible in design opportunities [25]. In addition, this feature is compatible with the chosen system of storage as the RecyclerViews are filled with data real-time from the external database upon accessing the relevant fragments. This direct database solution was favoured as there would not be of any need for any persistent local SQLite storage unless the app was designed to be available for offline play, in which case the hit to application performance, though minor, would outweigh the minuscule network requirements from keeping viewed data synchronised with external database. As such, the requirements to run the application are more geared towards requiring stable connection rather than on the user's device.

9.4.3 Classrooms and Modules

The RecyclerViews were nested within each other by having list elements redirect the user to a new relevant fragment on click. The new fragment would then contain specific information to the particular item, and if that information happen to be a list it would be displayed again as a RecyclerView, which might also have the possibility to be clicked. The relevant RecyclerViews are for classrooms, announcements, modules and games. Each user has their own classroom list present in their UI. Upon clicking the classroom, the navigation system redirects them to the classroom fragment which is then filled real-time with classroom data from the database. The classrooms have a top menu for announcements, modules, and participant list showing all students. The announcement page is set to be the default start screen of the classroom and loads a RecyclerView with an array of announcement strings from the database. It allows for teachers to add new announcements through a standard text input field and publish button, which is not present for the students participating. Functionality which was intended to only be accessible for teachers are all registered as UI elements in the layout XML file that are only shown if the user viewing the fragment is a teacher. This assumes that all teachers participating in the classrooms should all have administrator rights for said classroom. Students can join any classroom by pressing the join button and writing the ID of the class, if it exists in the database, it is added to their list. As such functional requirement 4 as defined in Section 5.1.1 is fulfilled.

9.4.4 Modules and Games

The module tab within the classroom fragment contains a RecyclerView with a list for the modules, a button to create a new module, and one for importing an existing module. Modules are as previously mentioned a collection of topic specific games and the responsibility for creating or importing them falls on the teacher. Upon creating a module, it is added to the module array within the classroom document in the database classroom collection, while a new empty module document is created within the database module collection. The list of module ids within the classroom document allows the application to query the database for those specific module documents from the "modules" collection within the database, whose data it then uses to fill the RecyclerView with modules. Importing an existing module adds the module id to the module list in classroom document similarly to what happens when adding a new module, as it only needs to reference an already existing module document.

When clicking one of the modules in the module RecyclerView, the user is redirected to the fragment for specific module, where the module id is used to fetch relevant module data from the database. The module consists of a RecyclerView for the games, and input fields for the teacher to create or import a new game. The option for creating a game leads the user to a new fragment for filling the necessary data for creating the game, while importing lets user reference an existing game id within the database. Upon creating or importing, the game id is added to the database module document array responsible for storing the relevant games. The RecyclerView within the specific module is then updated to contain the games registered in the database. When the game in said RecyclerView is clicked, the game engine is launched, and the specific game type is started and loaded with relevant data which was written to file on click. The teachers ability to configure the games was the main functionality fulfilling the functional requirement of giving teacher creative freedom in displaying their curricula described in Section 5.1.1, as well as the fourth non-functional requirement of being pedagogically viable described in Section 5.1.2.

Games could also be launched directly from within the open world game by interacting with the teacher, specifics for this functionality can be found it Section 9.1.3. However the functionality allowing teachers to decide which mini games are accessible for students within the open world is located in the classroom system. Teachers can press the "World" button in the ClassroomModuleFragment to access the WorldEditFragment, containing an "Add" button and a RecyclerView. Upon pressing the "Add" button, teacher is prompted to input the ID of a mini game, if it exists its added to the classroom documents "quizes" array within the database and displayed in the RecyclerView. Thereafter students of the class can open the particular mini games in the RecyclerView by interacting with the teacher of the classroom in the open world game and selecting it. Launching the mini games from within the open world is possible because all of the mini games the teacher made available in open world are written to file in local storage and named in the format of "gameName-TeacherName". The locally stored files are updated whenever the user enters their profile fragment.

The third functional requirement of having the service be editable, as seen in Section 5.1.1, was achieved as every module and game can be deleted from their respective list. If entire modules were deemed unfitting for the classroom, the course coordinator could press the delete icon on the RecyclerView entry to remove the reference to the module in the database. Similarly individual mini games could be removed from modules or from the list of mini games available in open world in the same manner. Removed modules and mini games continue to exist within the database as the delete functionality only de-references their id within the array of IDs, as such if course coordinator wants to re-add them later, they can simply be imported again. The delete button is only available for users with teacher rights.

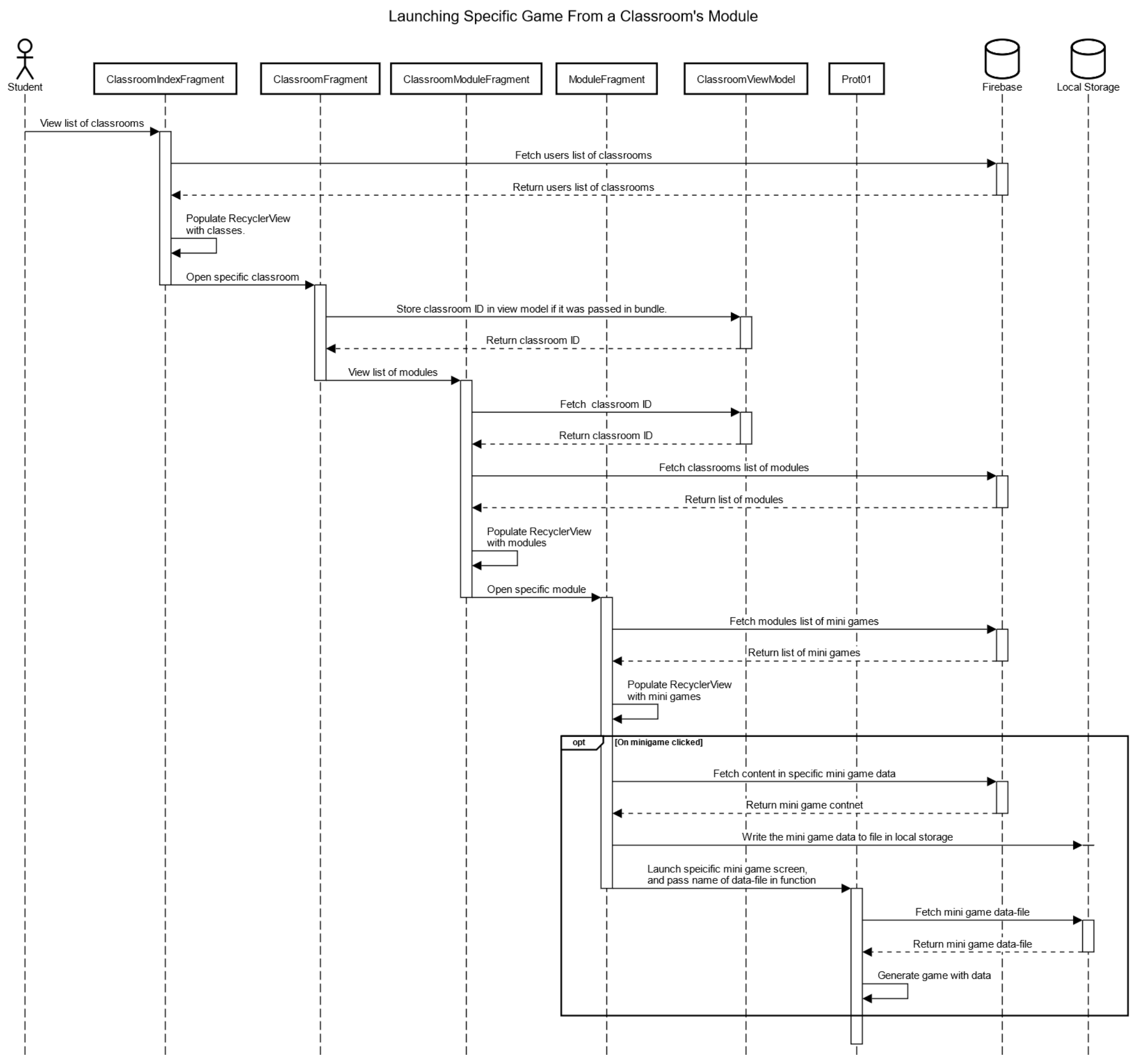

Launching Specific Game From a Classroom's Module
Student
ClassroomIndexFragment
ClassroomFragment
ClassroomModuleFragment
ModuleFragment
ClassroomViewModel
Prot01
Firebase
Local Storage
View list of classrooms
Fetch users list of classrooms
Return users list of classrooms
Populate RecyclerView with classes.
Open specific classroom
Store classroom ID in view model if it was passed in bundle.
Return classroom ID
View list of modules
Fetch classroom ID
Return classroom ID
Fetch classrooms list of modules
Return list of modules
Populate RecyclerView with modules
Open specific module
Fetch modules list of mini games
Return list of mini games
Populate RecyclerView with mini games
opt
[On minigame clicked]
Fetch content in specific mini game data
Return mini game contnet
Write the mini game data to file in local storage
Launch speicific mini game screen, and pass name of data-file in function
Fetch mini game data-file
Return mini game data-file
Generate game with data

Figure 9.16: System sequence diagram of launching a specific mini game from the application classroom/module system

Chapter 10

Quality Assurance and Testing

10.1 Quality Assurance

For the quality assurance for the application, the team decided on not having a single test phase. But instead having continuous individual testing on every git-push that was made, to any branch. This is in accordance with the scrum development model, as each individual developer has a constant responsibility for the quality of the product [26]. Test routines were set up, and updated as the application was developed to ensure that no pushes broke or invalidated any of the previous work. Every person who made a push had to go through all test points, and only if every aspect of the application were still running as intended, a merge request could be made to merge branches into main.

For pushing new or changed functionality into the repository it was the individual developer's main responsibility to check that the push would not conflict with any existing systems, but since this could be a rather large job for one person alone, multiple developers would check that the push did not break any of the systems they had the responsibility for. See the table: 4.2.

When a branch was to be merged into main, at least two members of the team needed to be present during the merging and follow the agreed upon testing routine. As responsibility for different sections of the product was split between the developers, the team members attending were dependent on which of the application's areas were affected. This was necessary to ensure that relevant parts of the system were tested before the merge.

Quality assurance during development was also an important aspect of the process, implementing proper error handling became key in detecting and narrowing down possible issues in the system. This was done using both user feedback in the form of toasts, and developer feedback by using the separate logging functionalities the android API and LibGDX/LibKTX.

10.2 User Testing

There were four test phases planned for the application. The first one at the end of sprint one, with the alpha prototype. This was to get feedback on the initial design and what direction the design should take from there.

The second test was planned at the end of sprint two. There the testers were supposed to test a more interactable version of the application, to see and give feedback on how the application behaved and felt while navigating through the different sections.

The third planned test was meant to take place in the early stages of sprint four. At this stage a Beta version of the application was to be presented to the testers, still with no game functionality, but with most of the applications functionality and design in place. The users were going to test the full experience using the application, from registering and logging in, to navigating through, opening classrooms and modules, checking out their profiles and all the way up to the point of starting a mini game. The feedback was supposed to help finalize the Beta version.

The fourth and final planned test phase revolved around the gameplay aspects of the application. The phase was meant to gather teacher feedback on their experience of managing classrooms and the configuration of games within it, as well as questioning students using the application regarding what could be done to improve the gameplay aspect and make it more entertaining and rewarding.

Unfortunately, there ended up being no actual user testing for the application. There were several reasons for this, the main one being the situation around the pandemic. The campus and schools were mostly closed, so this made the testing process extremely difficult, the potential time and work invested to make it work were deemed not to be worth it. The development of the game engine itself for the application took more time than expected. It was planned a concrete phase for the testing, but since the engine development prolonged the development for the application and the mini games it was continuously delayed.

Feedback was given from the clients and the study supervisor through the whole process, so they were the closest thing to a testing audience. They were kept updated through the Alpha and Beta stages, and provided some feedback and advice, but of course not as much as would have been gathered from actual user testing. The fact that no opportunity for testing opened up was quite a set-back. All potential feedback on the application from students and teachers were lost. The feedback would have been immensely helpful in finalizing the applications design, and center it around the user's needs and demands. The lost potential to find bugs and missing functionality that the team itself would not have been able to pick up on, is also something that might end up having an effect on the finalized product. In the end, the lack of proper testing became an obstacle which impeded progress and stunted the finalized version of the app.

Chapter 11

Installation

11.1 Requirements

To be able to run the application an Android phone or emulator with at least Android 8.0 and a network connection is required.

11.2 How to Run with Android Studio and Emulator on a PC

1. Install Android Studio on the PC, select the repository location and open the project.
2. Let all the Gradle files finish synchronizing before doing anything else. On the initial set up this will most likely take some time to complete.
3. Make sure there is an emulator installed with at least Android 8.0.
4. Before the application can be run make sure the right launch option is selected. The right one should be the one called just “android”. The options called "Android Launcher" or "Lwjgl3LauncherKit" will not work. Your emulator with the right Android version also needs to be selected. It should look like the picture below.

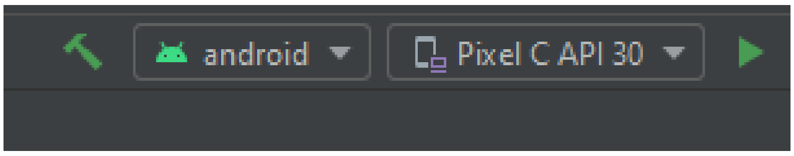


Figure 11.1: How the launcher options should look before running the application in Android Studio

5. The application is now ready to run and use.
6. An existing user can be used to test the application with the email: "student9@test.com" and password: "123456".
   Or email: "teacher1@test.com" and password: "123456"

7. Note! When using the application it is important to visit the user profile. Why this has to happen is discussed in Section 13.2.

11.3 How to Run on Phone with APK File

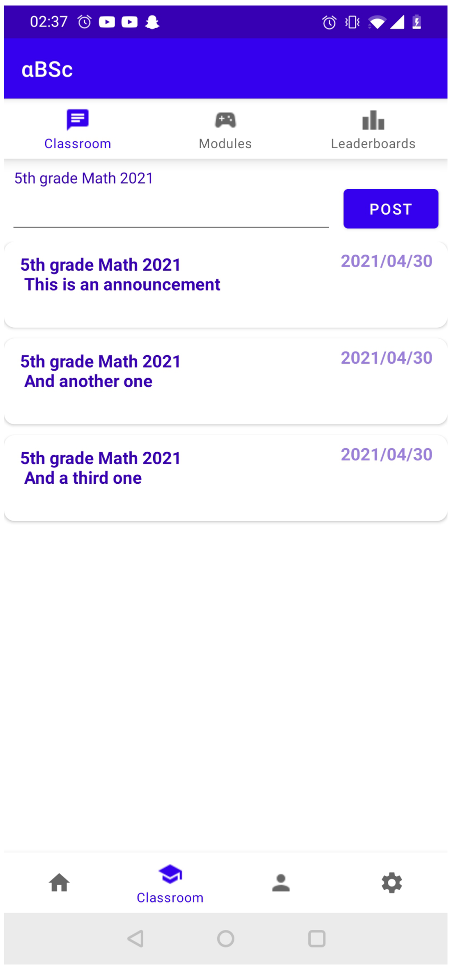
1. APK file needs to be downloaded or transferred onto the phone. The APK file is located inside the repository with the folder path:
   "gamePrototype01\android\release", if unforeseen problems arise in the release version of the application try using the debug release of the APK instead located in the folder path:
   "gamePrototype01\android\debug"
2. In the phone settings permissions to install applications from unknown sources need to be granted.
3. Once permissions have been granted the APK file needs to be found in the phone's file explorer. Once found just tap the file to install.
4. There might be more permissions or access you need to grant the application for it to install properly. These will most likely show up as pop-ups when installing.
5. Once installed the application is ready to use (For a couple of the phones that were used to test this, it would not work while connected to WiFi, so if there is no reaction while trying to log in or create a new user try switching from WiFi to mobile network).
6. An existing user can be used to test the application with the email: "student9@test.com" and password: "123456".
   Or email: "teacher1@test.com" and password: "123456"
7. Note! When using the application it is important to visit the user profile. Why this has to happen is discussed in Section 13.2.

Chapter 12

Result

The goals of the application were to develop an application that would make it much easier for teachers to teach any new curriculum to the students. It would give them an opportunity to create interactive ways to learn, using classroom modules and mini games. The teachers needed to be able to use any teaching material they would want to create mini games. The classroom modules in the application needed to be highly modifiable for the teachers so that it would be easy to create new ones, and it should be possible to share modules with other teachers so they can use mini games created by others.

All of these goals were met to some extent for the application that was developed. Any teacher can create a classroom, and inside the classrooms they can create new modules. The classroom acts like most classrooms on any teaching application where the teacher can post announcements and information about the course, see Figure 12.1. Where the application stands out from other learning platforms is the module system, see Figure 12.2 and Figure 12.3. The teacher can input any appropriate teaching material and use it to create a mini game. The mini game that was developed for the application was a quiz game, with a separate open world for the student to navigate. The player spawns into the open world, and there they can see the available teachers, see Figure 12.4. The player can walk over to a teacher to start a quiz the teacher has created, see Figure 12.5. The player then needs to navigate around the world to answer the quiz, see Figure 12.6. The way the application is interacting with the mini game makes it easy to pass the same teaching material into another mini game if it is created further down the line.



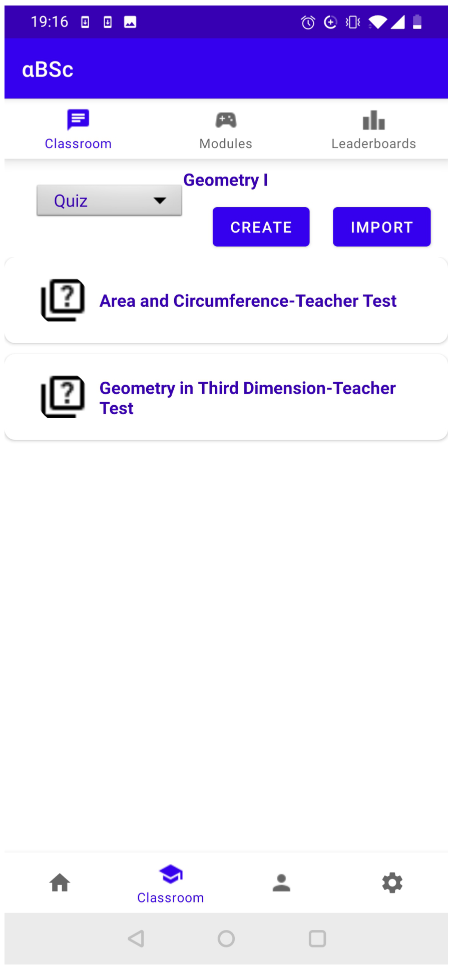
5th grade Math 2021

5th grade Math 2021 2021/04/30
This is an announcement

5th grade Math 2021 2021/04/30
And another one

5th grade Math 2021 2021/04/30
And a third one

Figure 12.1: Screenshot from app - this is the list of announcements inside a classroom. Accessed once a classroom has been picked

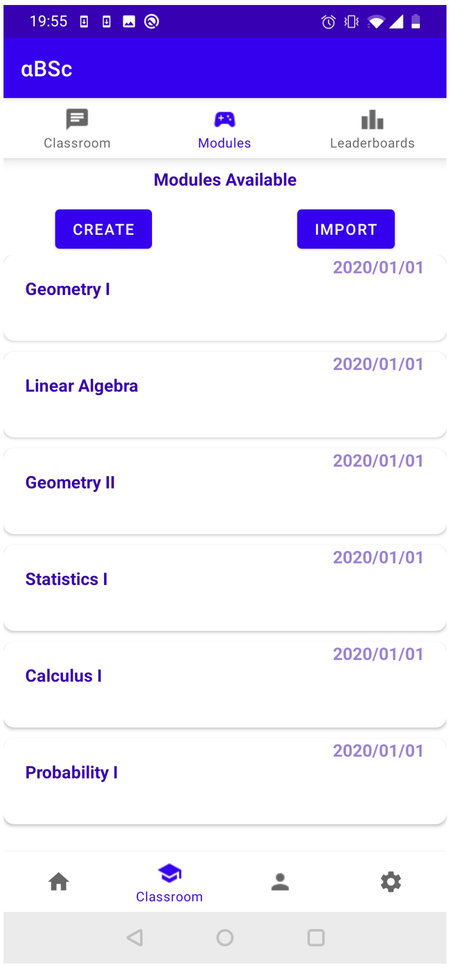


Geometry I

Area and Circumference-Teacher Test

Geometry in Third Dimension-Teacher Test

Figure 12.2: Screenshot from app 2 - Inside a module is a list of quizzes which the user may access


19:55

aBSc

Classroom | Modules | Leaderboards

Modules Available

CREATE | IMPORT

Geometry I	2020/01/01
Linear Algebra	2020/01/01
Geometry II	2020/01/01
Statistics I	2020/01/01
Calculus I	2020/01/01
Probability I	2020/01/01

Figure 12.3: Screenshot from app 3 - Inside the classroom, where a list of modules is displayed. The user may click to access a module

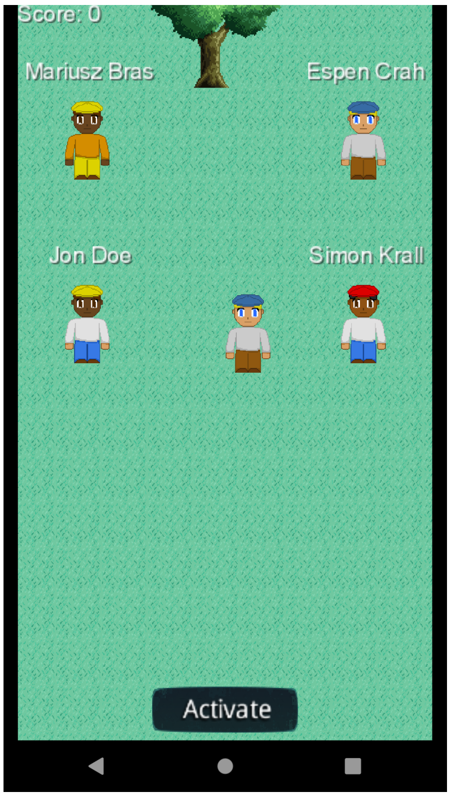


Figure 12.4: Screenshot from game - In the open world game, a student's teachers may be found and interacted with

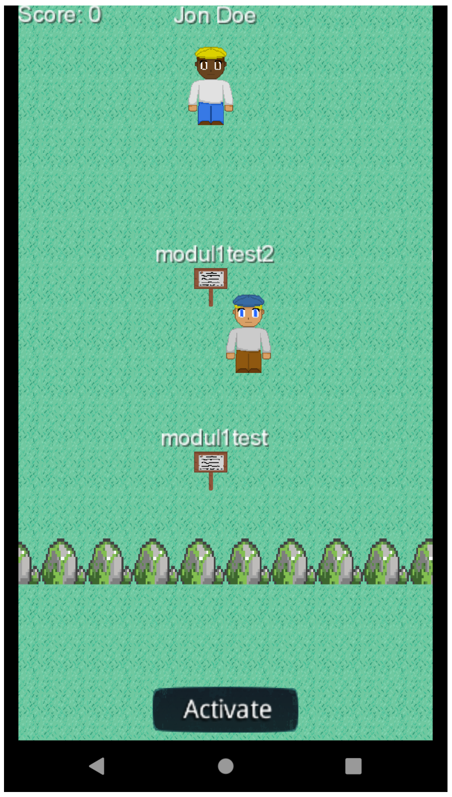


Figure 12.5: Screenshot from game - After interacting with the teacher, the student may select a quiz to do

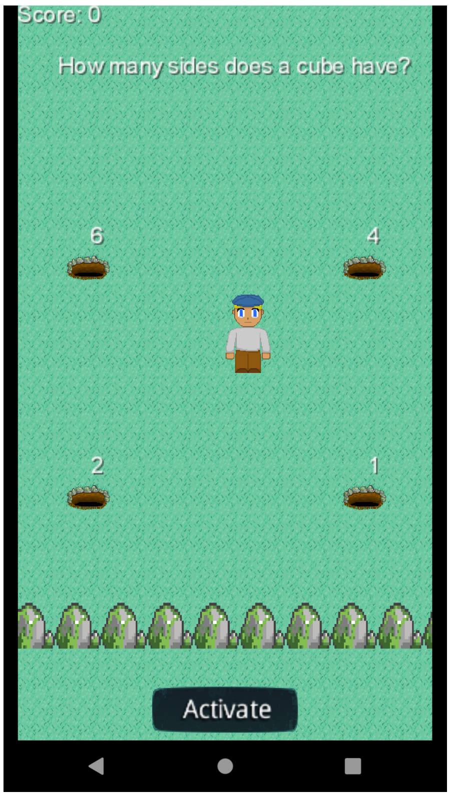


Figure 12.6: Screenshot from game - The quiz in action inside the game world

Chapter 13

Discussion

13.1 Evaluation

The finished product distinguishes itself from applications of similar purposes, for example *Quizlet* [27], by basing the gamification on a heavily graphical user experience in which they control a character representing themselves to interact with the world. It seeks to utilise this increased graphical feedback and intractability to further involve the students in the content presented to them. The open world game option has been designed to provide a digital environment where students can explore a world with teachers as quest givers, it is inspired by retro 2D roleplaying games and will be the passive and relaxed method for students to learn while having fun. For cramming, and increasing knowledge on specific topics, the application UI module system provides the same gamification from a more systematic interface. The application UI will allow students to play the same minigames available in the open world game, and potentially more, by accessing them directly within modules inside the classroom. Traditionally, gamification of learning experience follows minimalistic design principles, such as quizzes with black text on white backgrounds and clickable buttons for answers, or plain flashcards. This solution breaks that tradition in its attempt to make the gamification immersive, while still maintaining similar levels of customizability for teachers. All this while ease of scaling the service up and limitless creative freedom for further development.

13.2 What Could Have Been Done Differently?

13.2.1 User Testing

The lack of user testing became inevitable during the development of this application due to the situation around the pandemic as previously mentioned in Section 10.2. Nevertheless, it was a setback as potential valuable feedback from the public could not be acquired which would undoubtedly have been useful to gain insight in the practical uses of the application. The feedback would have helped with designing the app around the user experience instead of our own, and possibly point out issues or concerns some users might have with the application, so the finalized product could be as good as possible.

13.2.2 Database

There are some problems with the database that, while it did not impact development, makes it seem unfinished. The main issue being how it is not asynchronous. This, however, is no fault of the database itself, rather when the app sends and retrieves information to/from it. Instead of asynchronously getting the data needed at the time it is actually needed, it is all read as you log into the application, see Section 9.2.1 for more details on how it was implemented. From there it is put into a user object from which anyone can retrieve data about a user at any time. When a user logs in they have to visit the user profile for the quizzes and teachers to be read from the database, so that the lists used for the open world game can be generated. The original plan was to have all database functionality in a separate database class or object, and keep that functionality away from the fragments. This did not work because the fragments needed to inherit from the Fragment class, and therefore could not inherit the suspended functionality needed to run asynchronous functions. More error handling when getting data should be added to ensure no object are retrieved as null.

13.2.3 Workload

Initially, the project looked manageable at the start, but once the basis of the project were made, we quickly realized how much work was needed to get the main classes and functions working as specified. Due to the lack of proper documentation of Kotlin with e.g Firebase, a lot of time was put into research and we were left with less time than planned once the main classes and functions were added to the application. Simple bug fixing usually ended up with taking more time than expected. Other aspects of Android Studio did make some of our work more tedious than it should have been. Any work involving navigation would at times make Android Studio run at one frame per minute or crash program entirely due to the navigation file consuming too much memory.

13.2.4 Kotlin

Despite being a coding language made for mobile application, there was a lack of documentation for the use of Kotlin with specific components. Code examples in Java existed and converting the Java code to Kotlin took time. There is an automatic Java to Kotlin converter built in to Android Studio, but this would more often than not convert the code incorrectly so it was faster to manually convert the code to Kotlin. Since only one group member knew how to write Java, the rest of us were in theory learning a second programming language while we were expanding our knowledge with Kotlin. In reflection of this we could probably have achieved more with the application if we wrote it in C++ with Unreal Engine 4. That being said, this project has been a huge learning experience for all of the involved group members, and we all value this learning experience as it has expanded our knowledge and made us better developers. With the knowledge we now have we could make a similar application in Kotlin with LibKTX in a much shorter time span.

13.3 Further Work

Continuation of this bachelor project is definitively possible with how the game engine is set up. The code would have to receive minor changes to the internal pathing so the system paths would work correctly on other devices than Android, since LibGDX uses different methods for system paths between the devices it supports. The main application would need to be ported if it is to work with e.g. iOS (Apple devices) to the Swift programming language. The database functions should get asynchronous coroutines, this would greatly increase the application's data flow, because as of writing this, the user has to manually visit the different fragments so the database functions are called to synchronize the data between the device and database. The collision system needs a little bit of a tune up because if a player is persistent enough they can go through objects with a collision box, a "handy" feature for speedrunners, but it is something that should be fixed. Audio should be added further to the game as it adds a new "dimension" to game and it would make it more engaging for the players, currently there is only one audio feedback which activates once a mini game is completed. An animation system should be added to the game to animate the sprites e.g. when a player moves in game his/her avatar's feet should move with a walking animation. Potentially a new system should be created to help improve performance on older and weaker devices to dynamically add and remove entities as they go in and out of the screen boundaries. More game modes should be created to help increase the variety games the player can choose between and play. Regular new game mode additions is something that would help keep the playerbase of the application continuously interested as new content is refreshing and "new". The game engine architecture is designed to be modular so new game modes can easily be added. General bugfixes throughout the program is also something that should be done.

13.4 The Group Work

Mentioned in Section 4.4, distribution of work was divided among us and prioritized in regards to an individual's skill set. This was never meant to be a "hard" rule and was never used as that either. All members of the group have been working on all aspects of the program to some extent. A person was given a main responsibility of a system and had to keep track of how that system developed throughout the development process e.g. status updates. Did they need any help to get certain functionality working, what did other team members need to do to integrate their system with that system, etc. This approach worked out well, it allowed us to mix up our daily programming task so the development process did not become tedious and boring. It also allowed for different approaches to programming problems others had not thought about, which helped increase the final quality of the implemented program code.

Communication was done through Discord and Zoom due to the COVID-19 pandemic mentioned in Section 8.1. Communicating digitally worked out well; we were able to distribute work amongst ourselves and give updates on our workload. Discord made it easier to contact group members to work with and request assistance with problems that required more than one person to solve such as merging or combining the work of two to finish a function. To the group, this became a daily routine of checking to-do lists, notifying group members or simply establish work times. This daily routine served to ensure progress was made everyday and kept each other updated on how far other aspects of development were progressing, mentioned in Section 8.2.

Chapter 14

Conclusion

The plan for the project was to create an easily scalable application in which any form of curricula could be adapted into a graphical game, allowing teachers to gamify any content which students can access through an application UI, or a game UI. The requirements were all fulfilled to some extent, and the application is capable of portraying the intended user experience for teachers and students. The game engine and application achieved the goal of being developed with scalability as the primary focus, as further extending the service with new game modes will require minimal work due to the modularity of the engine. The cost of this was the limited presented end result which, although minimal, shows that adding new game modes is easily done by creating it as a new LibGDX screen. The scalability is also possible due to the entity component system, as adding entirely new functionality can be done through adding a new component with a system that executes for all entities with said new component. This also means that adding new functionality to existing systems can also be done easily by editing the systems themselves to update specific entities in a particular way if right conditions are met. This can be seen in the intractable system where the logic for both colliding with teacher and colliding with walls is implemented. The drawbacks of using LibGDX/LibKTX for the custom game engine is the learning curve for the libraries, which made the time investment for developing the engine greater than expected. As a result, more features of the service were put on hold as more people participated on the development of game engine, stifling productivity on other aspects of the application. The benefit of this however is the ability to optimize for performance and increased accessibility for all devices by avoiding the clutter which might follow when using commercial game engines. LibGDX/LibKTX provided everything necessary to develop the application, the only limitation is time. With the way things were developed, the time consumption of further developing all aspects of the game experience was shortened significantly, and the final product achieved the level of versatility wanted.

Bibliography

[1] LibGDX, *Libgdx*, Last visited 22.04.21. [Online]. Available: https://libgdx.com/.

[2] LibKTX, *Libktx*, Last visited 22.04.21. [Online]. Available: https://github.com/libktx/ktx.

[3] LibGDX, *Libgdx ashley*, Last visited 22.04.21. [Online]. Available: https://github.com/libktx/ktx.

[4] Quillraven, *Youtube channel*, Last visited 19.04.21, 2020. [Online]. Available: https://www.youtube.com/channel/UCe02AjMmzRwrChKaQ7mkv8g.

[5] D. S. Marquez and A. C. Sanchez, *Libgdx Cross-platform Game Development Cookbook*. 2014, Last visited 19.04.21. [Online]. Available: https://www.amazon.com/Libgdx-Cross-platform-Game-Development-Cookbook/dp/1783287292.

[6] LibGDX, *Api documentation: Class pooledengine*, Last visited 22.04.21. [Online]. Available: https://libgdx.badlogicgames.com/ci/ashley/docs/com/badlogic/ashley/core/PooledEngine.html.

[7] J. McMullin, 'Using design games,' 2007, Last visited 09.05.21. [Online]. Available: https://boxesandarrows.com/using-design-games/.

[8] M. Dailly, *Grand Theft Auto Game Design Document*. 1995, Last visited 09.05.21. [Online]. Available: https://www.gamedevs.org/uploads/grand-theft-auto.pdf.

[9] JetBrains, Last visited 22.04.21. [Online]. Available: https://kotlinlang.org/docs/faq.html#what-companies-are-using-kotlin.

[10] *Android studio*, Last visited 22.04.21. [Online]. Available: https://developer.android.com/studio.

[11] *Gimp*, Last visited 09.05.21. [Online]. Available: https://www.gimp.org/.

[12] *Aseprite*, Last visited 10.05.21. [Online]. Available: https://www.aseprite.org/.

[13] *Adobe xd*, Last visited 22.04.21. [Online]. Available: https://www.adobe.com/no/products/xd.html.

[14] T. Ettinger, *Gdx-liftoff*, Last visited 22.04.21. [Online]. Available: https://github.com/tommyettinger/gdx-liftoff.

[15] *Git*, Last visited 22.04.21. [Online]. Available: `https://git-scm.com/`.

[16] *Git-bash*, Last visited 22.04.21. [Online]. Available: `https://gitforwindows.org/`.

[17] *Github*, Last visited 22.04.21. [Online]. Available: `https://github.com/`.

[18] *Trello*, Last visited 22.04.21. [Online]. Available: `https://trello.com/`.

[19] *Discord*, Last visited 22.04.21. [Online]. Available: `https://discord.com/`.

[20] *Zoom*, Last visited 22.04.21. [Online]. Available: `https://zoom.us/`.

[21] *Microsoft whiteboard*, Last visited 13.05.21. [Online]. Available: `https://zoom.us/`.

[22] J. Sutherland, *Scrum Handbook*. 2010, Last visited 22.04.21. [Online]. Available: `https://www.researchgate.net/publication/301685699_Jeff_Sutherland%27s_Scrum_Handbook`.

[23] *Rubber duck debugging*, Last visited 13.05.21. [Online]. Available: `https://rubberduckdebugging.com/`.

[24] *Kotlin coroutines on android*, Last visited 29.04.21. [Online]. Available: `https://developer.android.com/kotlin/coroutines?gclid=Cj0KCQ%5Cnewline%20jwp86EBhD7A%20RIsAFkgakiJI8gqeUk6FxiT8%5Cnewline%20AhfceUp_4xdt84BtDmFZ7tJaptwaDDO4M65bQMaAqN3EALw_wcB&gclsrc=aw.ds`.

[25] *Create dynamic lists with recyclerview*, Last visited 05.05.21. [Online]. Available: `https://developer.android.com/guide/topics/ui/layout/recyclerview`.

[26] D. Pereira, 'How does qa fit with scrum?,' 2020, Last visited 09.05.21. [Online]. Available: `https://medium.com/serious-scrum/how-does-qa-fit-with-scrum-4a92f86bec5b`.

[27] *Quizlet*, Last visited 19.05.21. [Online]. Available: `https://quizlet.com/`.

Appendix A

Additional Material

A.0.1 Project Agreement

NTNU

Vår dato Vår referanse

Norges teknisk-naturvitenskapelige universitet

Prosjektavtale

mellom NTNU Fakultet for informasjonsteknologi og elektroteknikk (IE) på Gjøvik (utdanningsinstitusjon), og

NTNU Fakultet for informasjonsteknologi og elektroteknikk(IE) på Gjøvik

______________________________ (oppdragsgiver), og

Andreas Blakli, Vegard Opktivtne Årnes, Theo Camille Gascogne, Jesper Ulsrud

____

______________________________ (student(er))

Avtalen angir avtalepartenes plikter vedrørende gjennomføring av prosjektet og rettigheter til anvendelse av de resultater som prosjektet frembringer:

1. Studenten(e) skal gjennomføre prosjektet i perioden fra 11.01.2021 til 20.05.2021 .

   Studentene skal i denne perioden følge en oppsatt fremdriftsplan der NTNU IE på Gjøvik yter veiledning. Oppdragsgiver yter avtalt prosjektbistand til fastsatte tider. Oppdragsgiver stiller til rådighet kunnskap og materiale som er nødvendig for å få gjennomført prosjektet. Det forutsettes at de gitte problemstillinger det arbeides med er aktuelle og på et nivå tilpasset studentenes faglige kunnskaper. Oppdragsgiver plikter på forespørsel fra NTNU å gi en vurdering av prosjektet vederlagsfritt.

2. Kostnadene ved gjennomføringen av prosjektet dekkes på følgende måte:
   - Oppdragsgiver dekker selv gjennomføring av prosjektet når det gjelder f.eks. materiell, telefon, reiser og nødvendig overnatting på steder langt fra NTNU i Gjøvik. Studentene dekker utgifter for ferdigstillelse av prosjektmateriell.
   - Eiendomsretten til eventuell prototyp tilfaller den som har betalt komponenter og materiell mv. som er brukt til prototypen. Dersom det er nødvendig med større og/eller spesielle investeringer for å få gjennomført prosjektet, må det gjøres en egen avtale mellom partene om eventuell kostnadsfordeling og eiendomsrett.

3. NTNU IE på Gjøvik står ikke som garantist for at det oppdragsgiver har bestilt fungerer etter hensikten, ei heller at prosjektet blir fullført. Prosjektet må anses som en eksamensrelatert oppgave som blir bedømt av intern og ekstern sensor. Likevel er det en forpliktelse for utøverne av prosjektet å fullføre dette til avtalte spesifikasjoner, funksjonsnivå og tider.

4. Alle beståtte bacheloroppgaver som ikke er klausulert og hvor forfatteren(e) har gitt sitt samtykke til publisering, kan gjøres tilgjengelig via NTNUs institusjonelle arkiv NTNU Open.

Tilgjengeliggjøring i det åpne arkivet forutsetter avtale om delvis overdragelse av opphavsrett, se «avtale om publisering» (jfr Lov om opphavsrett). Oppdragsgiver og veileder godtar slik offentliggjøring når de signerer denne prosjektavtalen, og må evt. gi skriftlig melding til studenter og instituttleder/fagenhetsleder om de i løpet av prosjektet endrer syn på slik offentliggjøring.

Den totale besvarelsen med tegninger, modeller og apparatur så vel som programlisting, kildekode mv. som inngår som del av eller vedlegg til besvarelsen, kan vederlagsfritt benyttes til undervisnings- og forskningsformål. Besvarelsen, eller vedlegg til den, må ikke nyttes av NTNU til andre formål, og ikke overlates til utenforstående uten etter avtale med de øvrige parter i denne avtalen. Dette gjelder også firmaer hvor ansatte ved NTNU og/eller studenter har interesser.

5. Besvarelsens spesifikasjoner og resultat kan anvendes i oppdragsgivers egen virksomhet. Gjør studenten(e) i sin besvarelse, eller under arbeidet med den, en patentbar oppfinnelse, gjelder i forholdet mellom oppdragsgiver og student(er) bestemmelsene i Lov om retten til oppfinnelser av 17. april 1970, §§ 4-10.

6. Ut over den offentliggjøring som er nevnt i punkt 4 har studenten(e) ikke rett til å publisere sin besvarelse, det være seg helt eller delvis eller som del i annet arbeide, uten samtykke fra oppdragsgiver. Tilsvarende samtykke må foreligge i forholdet mellom student(er) og faglærer/veileder for det materialet som faglærer/veileder stiller til disposisjon.

7. Studenten(e) leverer oppgavebesvarelsen med vedlegg (pdf) i NTNUs elektroniske eksamenssystem. I tillegg leveres ett eksemplar til oppdragsgiver.

8. Denne avtalen utferdiges med ett eksemplar til hver av partene. På vegne av NTNU, IE er det instituttleder/faggruppeleder som godkjenner avtalen.

9. I det enkelte tilfelle kan det inngås egen avtale mellom oppdragsgiver, student(er) og NTNU som regulerer nærmere forhold vedrørende bl.a. eiendomsrett, videre bruk, konfidensialitet, kostnadsdekning og økonomisk utnyttelse av resultatene. Dersom oppdragsgiver og student(er) ønsker en videre eller ny avtale med oppdragsgiver, skjer dette uten NTNU som partner.

10. Når NTNU også opptrer som oppdragsgiver, trer NTNU inn i kontrakten både som utdanningsinstitusjon og som oppdragsgiver.

11. Eventuell uenighet vedrørende forståelse av denne avtale løses ved forhandlinger avtalepartene imellom. Dersom det ikke oppnås enighet, er partene enige om at tvisten løses av voldgift, etter bestemmelsene i tvistemålsloven av 13.8.1915 nr. 6, kapittel 32.

12. Deltakende personer ved prosjektgjennomføringen:

NTNUs veileder (navn): Mariusz Nowostawski

Oppdragsgivers kontaktperson (navn): Espen Torseth

Student(er) (signatur): Andreas Blakli dato 18.01.2021

Vegard O. Årnes dato 18.01.2021

Theo C Gascoyne dato 18.01.2021

Jesper Ulsrud dato 18.01.2021

Oppdragsgiver (signatur): [illegible] dato 18.01.2021

Signert avtale leveres digitalt i Blackboard, rom for bacheloroppgaven.
Godkjennes digitalt av instituttleder/faggruppeleder.

Om papirversjon med signatur er ønskelig, må papirversjon leveres til instituttet i tillegg.
Plass for evt sign:

Instituttleder/faggruppeleder (signatur): ______________________________ dato __________

A.0.2 Project Plan

Project Plan

Andreas Blakli Vegard Opkvitne Årnes
Theo Camille Gascogne Jesper Ulsrud

31.01.21

Contents

1 Aims and Borders

1.1 Background

The projects goal is to make an app for phones, tablets and web browsers which is a gamification of the digital security curriculum for Norwegian students from primary school, junior high school and high school.

1.2 Project aims

The project aims to make the teaching process for students fun, interactive and to give the students a high learning outcome and a sense of accomplishment when solving a task. The app also seeks to create a better tool for teachers to use in their workday, so their workflow becomes more efficient, and help them offload some of their daily workload.

1.3 Borders

1.3.1 Requirements

Needs to work on phones, tablets and web browsers.
The app must be designed in a way so it scales well with small and big screen resolutions.
User profile and the option to edit said profile. Provide the student with a customizable avatar, the avatar will also grow with the student as they age and go trough the different school levels. The avatar needs to be inclusive and politically correct.
App must be modular and dynamic to allow for content updates if and when the digital security curriculum changes.
The game must be engaging for the user and provide a useful learning experience.
The game must work with multiplayer so that a group of students can play together e.g. co-op, it must also work in a school environment i.e. classroom and at home.

2 Scope of development

2.1 Subject area

Our main subject area will be developing a mobile application for android devices. We will be developing using kotlin in Android Studio. We aim to use an OpenGL library for Android Studio to program our 3D games and animations. We are using Firebase for our database solution, and we also use Firebase for our user creation, login and authentication functionality.

2.2 Delimitation

We will mainly focus on just android development over iOS and web, because of the size of the scope will become to large if we are to implement support for all platforms. Developing the application with focus on android support will make it more manageable, and we might also have to cut down even more on the scope to have time to make a working product.

2.3 Task description

The task at hand is to create an application that could be used in norwegian schools to teach about digital security. Utdanningsdirektoratet(UDIR) has come up with a new curriculum for digital security for all ages(primary school, junior high school and high school), where digital security will be a part of every other subject, and therefore not a subject by it self. The goal of the application is therefore to help the teachers, who more often than not will not be greatly familiar with the subject of digital security. The application will have the functionality so that the teachers can make games, quizzes etc to help teach about the subject. The traditional learning applications are often perceived as quite boring and dull, and do not engage the students enough, and that is why NTNU IIK wants to make an app that will help make it easier for the teachers and more relevant, engaging and fun for the students.

3 Project organisation

3.1 Responsibilities and roles

To structure ourselves we have set up a hierarchy where we have one group leader and one secretary. The group leader responsibilities is to organizes the group members e.g. when to meet, what to prioritize in the development process, contact with the stakeholders and scheduling of guidance sessions with said stakeholders. The secretary's responsibilities is to make minutes of meetings with the stakeholders and contact with the stakeholders.
Andreas Blakli is the group leader and Vegard O. Årnes is the group secretary. As for the general work structure we will mainly do weekly scrums. To organize the scrum sessions and decide who's working on the different tasks in the project we are using Trello.com's card system. We do not have a super strict setup where one person only works with one aspect of the system/application, we have a more "free" approach where people can take which task they want from the available tasks set up in Trello. When we are scrumming we are all sitting together in a voice chat application so we can communicate directly and clear up issues quickly and efficiently.

3.2 Routines and group rules

1. Meet up at the scheduled group sessions.

2. If a group member is unable to join said session they must notify the other group members.

3. Follow deadlines and deliver work on time.

4. If stuck on a problem don't be afraid to ask for help.

5. If a group member fails to follow the rules that are laid out above, the group leader will make contact with the troubled member and find out why things is not working out and try to find a solution for all of the involved parties. If no solution can be found, the study supervisor will be contacted and notified of the problem the group is experiencing.

4 Planning, Follow-up and reporting

4.1 Division of project

The software development method used for the project is as mentioned before, Scrum, though adjusted for a smaller sized group. It was chosen due to its free nature in that optimising each part of the code, layout element, or functionality will be a constant task based on feedback from other developers, users, or product owners. In addition, the learning curve is relatively steep in the start, and an agile method takes education of team members more in to account through development, instead of for other more streamlined methods (i.e. waterfall) where everyone needs to be fully aware of each step of the development process early in development.
The development process will follow all main principles of scrum, however as the Corona pandemic is ongoing during this, some additional steps are taken for better communication between team members, organisation and discipline. More weight is put on constant voice communication in work hours, and an online scrum board is regularly updated to keep track of each others progress. To hinder unexpected issues during development, merging of branches will also either happen in pairs, or during group meetings with all present.

4.2 Plan for status meetings and decision points in the period

Status meetings are held daily since we are all scrumming together in a voice chat application 5 days a week between 09:00 - 16:00.
Important executive decisions are made on the 25.01.2021 after our meeting with the stakeholders to get the final specifications for the application. The next major decision point is when we have the application ready for user testing. And after that the next major decision point will be with the stakeholders after we have analyzed the results of the user testing process. Minor decisions regarding UI elements/layout and program architecture will be made throughout the system development process.

5 Organisation of quality assurance

5.1 Documentation, standard use and source code

Due to how the application is planned to work, trough using customizable modules made for unique classes by the teacher, the documentation should be a walkthrough of how modules are set up for the teachers and how to navigate and get trough the education content in the modules for the students. The walkthrough should cover every variation of modules that are possible to make. Alongside documentation for the end-user, a documentation on implementation should also be made to increase understanding amongst the developers and any future developers. The documentations should be updated regularly to ensure the information end users/development acquire from it is not outdated. Things to consider when making the documentation should be to make it easy to access the information needed, so an FAQ is needed for the end user. The source code will be open source in GitHub throughout development until further notice from product owners. Documentation of source code and the iterative changes made on it in development will be part of the documentation mentioned above.

5.2 Configuration control

Throughout development, maintaining the application's integrity is important to uphold and demonstrate the safety and security of the app, and it is crucial for our application to be resilient against attempts at tampering with information inside the applications or rooms made for students as well as protecting private information of students and teachers alike. Applying configuration management does just that. More specifically, configuration management handles changes systematically so that we may maintain the application's integrity. During development of the application, the architecture of the application's system should be defined so that any changed made to it can be easily tracked and documented. With the system's functionality documented, changes to the system can then be proposed and approved by the development team. Once the proposed changes are approved and under development, the relation between functionalities are to be reported on often so changes can be notified as soon as possible. Finally once the changes has been implemented, the changes should undergo testing to see if it has achieved its intended functionality. The changes made are then to be documented with details on functionality before and after implementation.

5.3 Risk analysis (identify, analyze, measures, follow-up) (Technological, Business-wise, Project group-wise)

To make sure the application is secure and resilient to attacks, using the STRIDE model is a good start to identify threats based on the method of attack and affected part of the system. STRIDE categorizes such attacks based on:

- Spoofing – Authenticity, gaining access to other user's accounts.

- Tampering – Integrity, illegal modification of data.
- Repudiation – Non-reputability, hiding the authenticity of user actions.
- Information disclosure – Confidentiality, exposure of private and sensitive data.
- Denial of Service – Availability, denying access to a service.
- Elevation of Privilege – Authorization, gaining a higher level of privilege illegally

Once threats are identified, an evaluation of the threats are needed to properly assess those threats. A scoring model is often used for ranking threats in terms of severity and risk for a project. The DREAD is a scoring model that categorizes the severity of an attack in how easy it is to execute for example. The threat is then given a final score based on the five aspects listed below.

- Damage – how damaging would the attack be to a system or project
- Reproducibility – how easy is it to replicate an attack
- Exploitability – how easy is it to launch the attack
- Affected users - how many are affected by the attack
- Discoverability – how easy it is to notice the attack

An example of how scoring threats would work.

Threat	D	R	E	A	D	Total	Rating
Brute force password	2	1	3	1	3	10	Medium

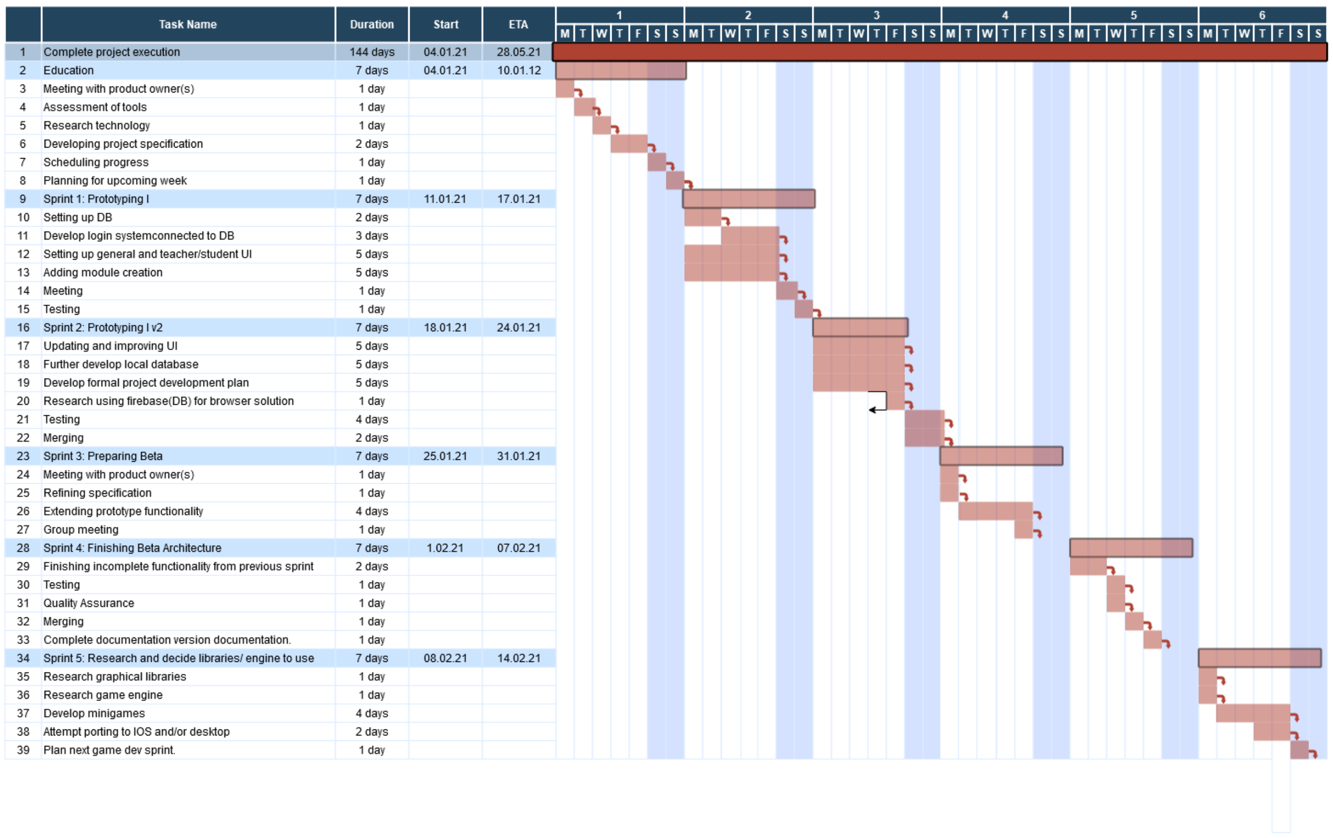
After analysing the threats, the focus should be set to restrict the damage done to the system by the attacks. The various methods of mitigating the threats can also be generally classified in the same way threats are (STRIDE) but may also vary from project to project. Implementing mitigations and security functions should be done alongside the development of the application. In our case to mitigate spoofing attacks such as brute force password, the use of authenticators and password encryption would be sufficient.

6 Plan for implementation

6.1 Activities (Work Breakdown Structure), milestones and decision points

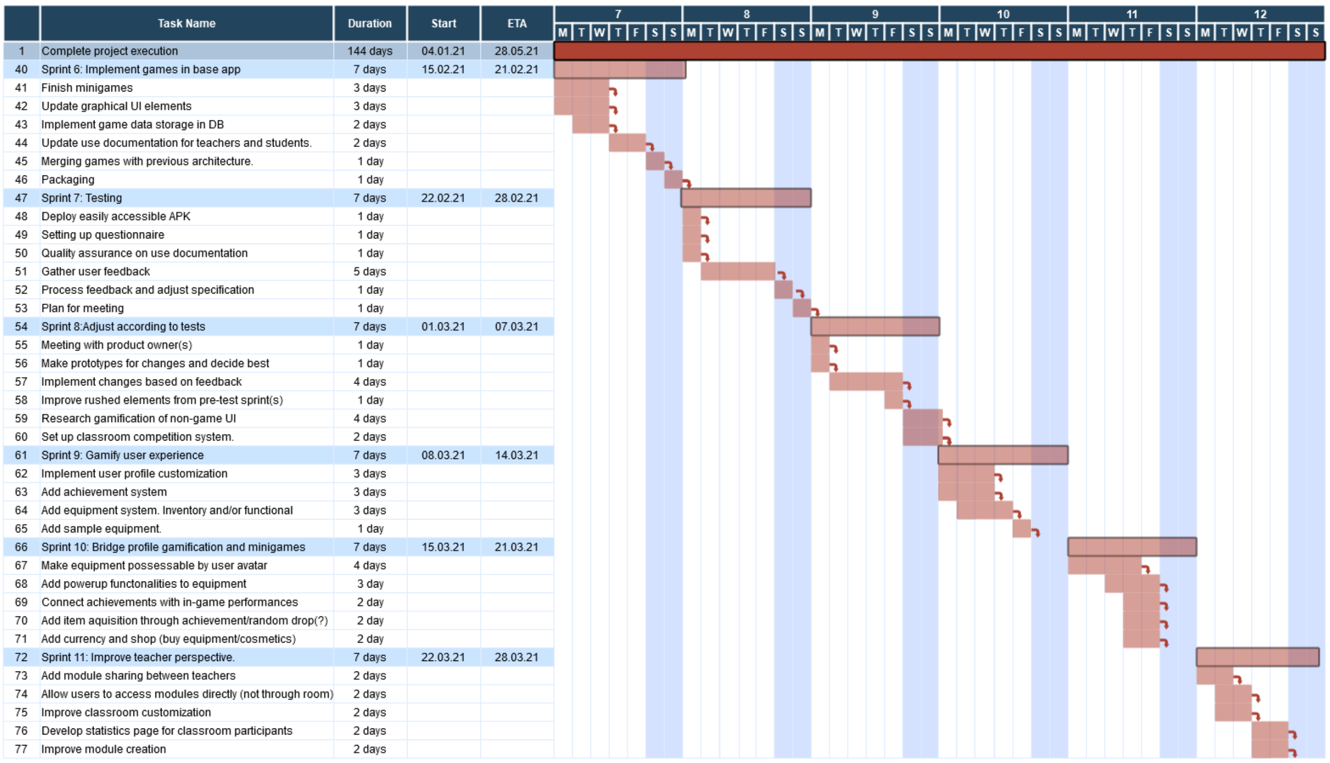
As the development process will be following an agile process, the service will be subject to change throughout the development lifecycle. As such activities for implementation involve the previously mentioned scrum-board, using “trello” for keeping track of smaller functionalities that can be developed separately. Each functionality will be developed in different dedicated sprints but will also be going through one or more refactoring cycles as product owner feedback, user feedback and developer feedback will illuminate changes or extensions to build towards the final product. Milestones will be measured according to this feedback as well as specific checkpoints related to meeting with product owner as well as the different iterations of testing done with actual users. Progress is therefore best seen when both product owner and userbase are satisfied with the different implementations, which means that we will not need to refactor or extend previously worked on functionalities more than adjustments related to other elements (I.E. User interface), they can therefore be considered the decision points of the project as they influence the next weekly scrums the most.

6.2 Gantt-schema



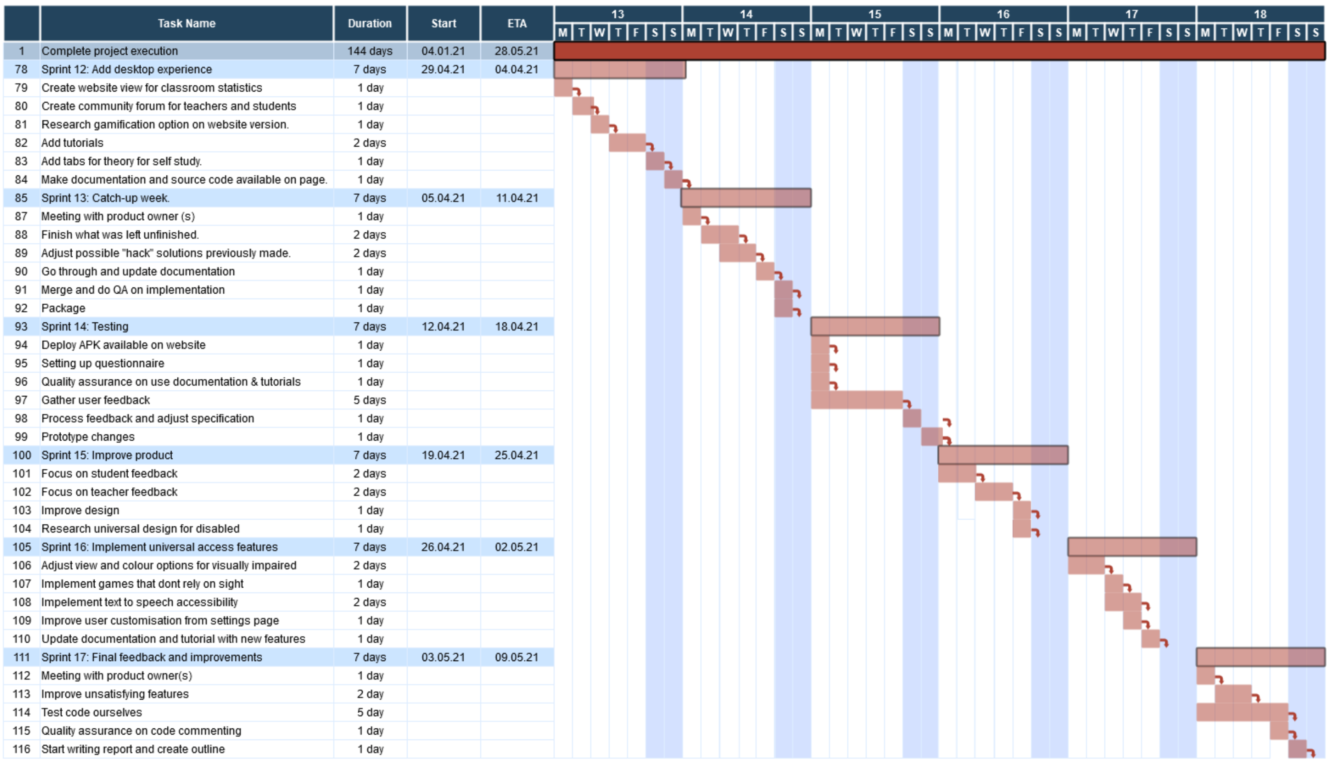
	Task Name	Duration	Start	ETA
1	Complete project execution	144 days	04.01.21	28.05.21
2	Education	7 days	04.01.21	10.01.12
3	Meeting with product owner(s)	1 day		
4	Assessment of tools	1 day		
5	Research technology	1 day		
6	Developing project specification	2 days		
7	Scheduling progress	1 day		
8	Planning for upcoming week	1 day		
9	Sprint 1: Prototyping I	7 days	11.01.21	17.01.21
10	Setting up DB	2 days		
11	Develop login systemconnected to DB	3 days		
12	Setting up general and teacher/student UI	5 days		
13	Adding module creation	5 days		
14	Meeting	1 day		
15	Testing	1 day		
16	Sprint 2: Prototyping I v2	7 days	18.01.21	24.01.21
17	Updating and improving UI	5 days		
18	Further develop local database	5 days		
19	Develop formal project development plan	5 days		
20	Research using firebase(DB) for browser solution	1 day		
21	Testing	4 days		
22	Merging	2 days		
23	Sprint 3: Preparing Beta	7 days	25.01.21	31.01.21
24	Meeting with product owner(s)	1 day		
25	Refining specification	1 day		
26	Extending prototype functionality	4 days		
27	Group meeting	1 day		
28	Sprint 4: Finishing Beta Architecture	7 days	1.02.21	07.02.21
29	Finishing incomplete functionality from previous sprint	2 days		
30	Testing	1 day		
31	Quality Assurance	1 day		
32	Merging	1 day		
33	Complete documentation version documentation.	1 day		
34	Sprint 5: Research and decide libraries/ engine to use	7 days	08.02.21	14.02.21
35	Research graphical libraries	1 day		
36	Research game engine	1 day		
37	Develop minigames	4 days		
38	Attempt porting to IOS and/or desktop	2 days		
39	Plan next game dev sprint.	1 day		

Figure 1: Gantt-schema for first 6 weeks



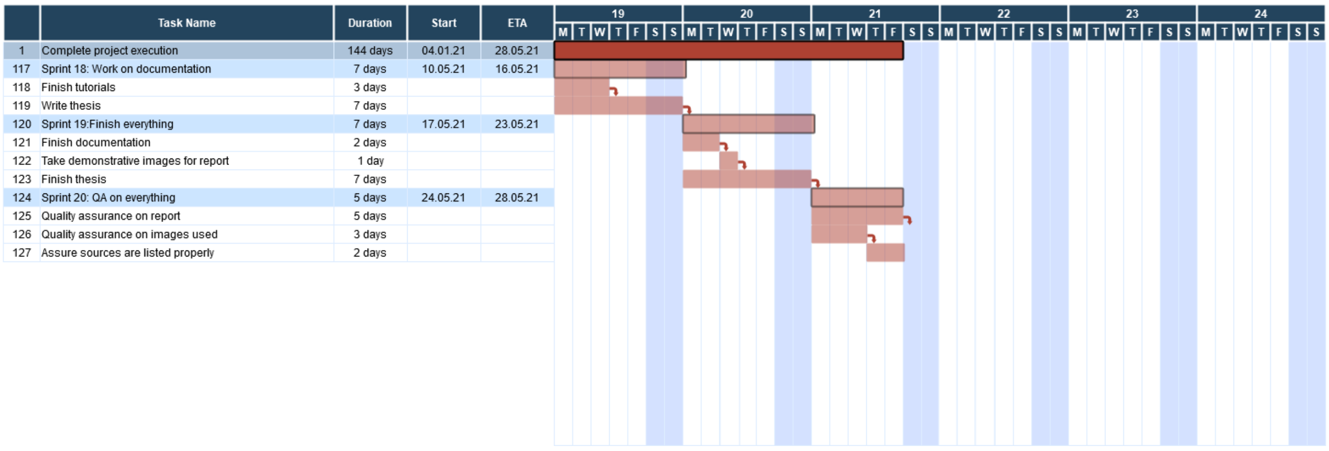
	Task Name	Duration	Start	ETA
1	Complete project execution	144 days	04.01.21	28.05.21
40	Sprint 6: Implement games in base app	7 days	15.02.21	21.02.21
41	Finish minigames	3 days		
42	Update graphical UI elements	3 days		
43	Implement game data storage in DB	2 days		
44	Update use documentation for teachers and students.	2 days		
45	Merging games with previous architecture.	1 day		
46	Packaging	1 day		
47	Sprint 7: Testing	7 days	22.02.21	28.02.21
48	Deploy easily accessible APK	1 day		
49	Setting up questionnaire	1 day		
50	Quality assurance on use documentation	1 day		
51	Gather user feedback	5 days		
52	Process feedback and adjust specification	1 day		
53	Plan for meeting	1 day		
54	Sprint 8:Adjust according to tests	7 days	01.03.21	07.03.21
55	Meeting with product owner(s)	1 day		
56	Make prototypes for changes and decide best	1 day		
57	Implement changes based on feedback	4 days		
58	Improve rushed elements from pre-test sprint(s)	1 day		
59	Research gamification of non-game UI	4 days		
60	Set up classroom competition system.	2 days		
61	Sprint 9: Gamify user experience	7 days	08.03.21	14.03.21
62	Implement user profile customization	3 days		
63	Add achievement system	3 days		
64	Add equipment system. Inventory and/or functional	3 days		
65	Add sample equipment.	1 day		
66	Sprint 10: Bridge profile gamification and minigames	7 days	15.03.21	21.03.21
67	Make equipment possessable by user avatar	4 days		
68	Add powerup functonalities to equipment	3 day		
69	Connect achievements with in-game performances	2 day		
70	Add item aquisition through achievement/random drop(?)	2 day		
71	Add currency and shop (buy equipment/cosmetics)	2 day		
72	Sprint 11: Improve teacher perspective.	7 days	22.03.21	28.03.21
73	Add module sharing between teachers	2 days		
74	Allow users to access modules directly (not through room)	2 days		
75	Improve classroom customization	2 days		
76	Develop statistics page for classroom participants	2 days		
77	Improve module creation	2 days		

Figure 2: Gantt-schema for weeks 7-12



	Task Name	Duration	Start	ETA
1	Complete project execution	144 days	04.01.21	28.05.21
78	Sprint 12: Add desktop experience	7 days	29.04.21	04.04.21
79	Create website view for classroom statistics	1 day		
80	Create community forum for teachers and students	1 day		
81	Research gamification option on website version.	1 day		
82	Add tutorials	2 days		
83	Add tabs for theory for self study.	1 day		
84	Make documentation and source code available on page.	1 day		
85	Sprint 13: Catch-up week.	7 days	05.04.21	11.04.21
87	Meeting with product owner (s)	1 day		
88	Finish what was left unfinished.	2 days		
89	Adjust possible "hack" solutions previously made.	2 days		
90	Go through and update documentation	1 day		
91	Merge and do QA on implementation	1 day		
92	Package	1 day		
93	Sprint 14: Testing	7 days	12.04.21	18.04.21
94	Deploy APK available on website	1 day		
95	Setting up questionnaire	1 day		
96	Quality assurance on use documentation & tutorials	1 day		
97	Gather user feedback	5 days		
98	Process feedback and adjust specification	1 day		
99	Prototype changes	1 day		
100	Sprint 15: Improve product	7 days	19.04.21	25.04.21
101	Focus on student feedback	2 days		
102	Focus on teacher feedback	2 days		
103	Improve design	1 day		
104	Research universal design for disabled	1 day		
105	Sprint 16: Implement universal access features	7 days	26.04.21	02.05.21
106	Adjust view and colour options for visually impaired	2 days		
107	Implement games that dont rely on sight	1 day		
108	Impelement text to speech accessibility	2 days		
109	Improve user customisation from settings page	1 day		
110	Update documentation and tutorial with new features	1 day		
111	Sprint 17: Final feedback and improvements	7 days	03.05.21	09.05.21
112	Meeting with product owner(s)	1 day		
113	Improve unsatisfying features	2 day		
114	Test code ourselves	5 day		
115	Quality assurance on code commenting	1 day		
116	Start writing report and create outline	1 day		

Figure 3: Gantt-schema for weeks 13-18



	Task Name	Duration	Start	ETA
1	Complete project execution	144 days	04.01.21	28.05.21
117	Sprint 18: Work on documentation	7 days	10.05.21	16.05.21
118	Finish tutorials	3 days		
119	Write thesis	7 days		
120	Sprint 19:Finish everything	7 days	17.05.21	23.05.21
121	Finish documentation	2 days		
122	Take demonstrative images for report	1 day		
123	Finish thesis	7 days		
124	Sprint 20: QA on everything	5 days	24.05.21	28.05.21
125	Quality assurance on report	5 days		
126	Quality assurance on images used	3 days		
127	Assure sources are listed properly	2 days		

Figure 4: Gantt-schema for weeks 19-21

A.0.3 Bachelor Thesis Task

Oppdragsgiver

Oppdragsgiver:	NTNU IIK
Kontaktperson	Espen Torseth
Adresse:	NTNU Gjøvik, A217
Telefon:	+47 916 90 629
Epost:	espen.torseth@ntnu.no

Gameification av nytt pensum i digital sikkerhet for barne- og ungdomsskole og videregående

Utdanningsdirektoratet (UDIR) har utarbeidet nytt pensum i digital sikkerhet for barneskole (1.-7. trinn), ungdomsskole (8.-10. trinn) og videregående skole. Digital sikkerhet skal innlemmes i alle emner, og vil derfor ikke være et selvstendig emne. Dette vil gjøre undervisningen mer utfordrende for lærere, spesielt med tanke på en helhetlig tilnærming til fagområdet. Digital sikkerhet er også sjelden kjernekompetansen til lærerne.

NTNU IIK ønsker å støtte opp under lærernes jobb ved å utvikle en app som støtter undervisningen i digital sikkerhet. Erfaring fra andre tilsvarende apper er at de ofte oppleves som kjedelige, lite relevante og ikke engasjerer elevene. Dette ønsker vi å endre til en engasjerende app som føles relevant for både elvene og lærere.

Oppgavene

Vi har ikke utarbeidet en fullstendig kravspesifikasjon ennå, men vi vet allerede at vi vil trenge komponentene under. Hver komponent er en bacheloroppgave i seg selv:

1) Profileditor
    a. Brukere skal ha sin egen profil og avtar og ved behov en midlertidig avatar
    b. Avatarer skal vokse med dem gjennom trinnene
    c. Avatarer må være inkluderende i sin utforming r

2) Digital assets
    a. Primær brukerflate for elever vil være mobil
    b. Sekundær brukerflate er tablet eller laptop
    c. Alle assets må være designet slik at de fungerer godt på små og store flater
    d. Full liste over assets er ikke lagd ennå

3) Innholdseditor
    a. Digital sikkerhet er i konstant utvikling så vi må kunne lage nytt og endre innhold
    b. Vi ønsker ikke den normale statiske flyten i tilsvarende løsninger
    c. Innholdet bør kunne utforskes mer som en quest enn som en presentasjon

4) Spilldesign/scoring
    a. Hvordan kan kunnskap og evner i digital sikkerhet gjøres engasjerende?
    b. Hvilke spillformater kan fungere for klasserom?
    c. Hvilke spillformater kan fungere for gruppearbeid?
    d. Hvilke spillformater kan fungere for hjemmearbeid?

A.0.4 Version History

Version History

α - Version

29 Dec. v0 : – Empty project start

27 Jan. v0.3: - Login authentication, core fragments, navigation system.

Bugfixes:

Clean-up, Register user compatible with DB, added DB test functions.

Add viewbinding to DB related fragments and updated navigation   functionality used.

Code clean-up

1 Feb. v0.7: - Settings, Classroom Recyclerview, User Info, DB data retrieval, Teacher/student role.

Bugfixes:

Fix Navigation Error & Login Error.

9 Feb. v1.0: - Menu hide/show functionality added, User profile, User Sessions

Bugfixes:

Improved settings, can change theme without resetting app.

Fixed logout button.

β - Version

19 Feb. v1.1: - Main functionality of KTX engine added, Moveable sprite in game, Imported alpha into game repository, game engine launchable from application UI.

Quality Assurance:

Quality assurance of all database functionality in beta repository.

Code Clean-up

26 Mar. v1.2: - DBobject with all user data, Classroom creation updated and improved, Classroom now compatible with database, improved user profile, added different student and teacher profiles.

Bugfix:

Branch merge errors

Branch merge errors, fix consistency error making layout change depending on screensize.

14 Apr. v1.4: - Game map creation, Screen switching improved, improved entity disposal, Quiz creation fragment added, Read/Write Quiz to file functionality, Improved game movement, Implemented intractability & collision detection with objects.

Quality Assurance/Bugfix:

Pair programming of game engine

20 Apr. v1.6: - Text rendering functional, Player score HUD displayed, Quiz entities now present on map, can be answered by colliding with, Other questions loaded upon answering, Quiz Question and answer positioned dynamically, Communication between game and app established through shared prefs, Movement improved to be joystick-Like.

Quality Assurance/Bugfix:

Pair debugging of game engine

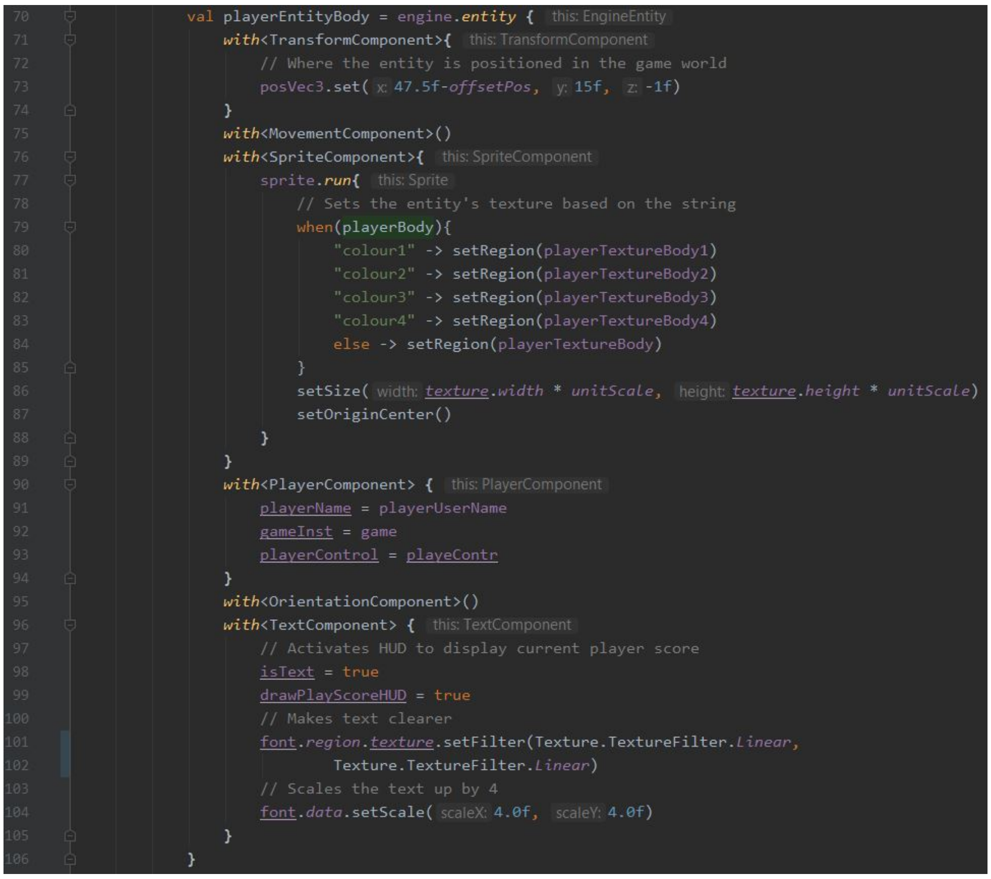
5 May. v1.8: - Quiz creation compatible with DB, Local quiz deletion, Achievement system, Avatar creation, Open world game created with playable quiz and interactable teachers, Created Avatar now displayed in-game, Open world compatible with DB, Result screen after completed quiz added, Classroom/Module/Quiz creation/importing and user interface in application completed, Interact button added,

Bugfix:
Scaling issues
Open World game debugging and clean up.
Classroom/Module/Quiz creation/import compatible with DB

19 May. v2.0: - Made teacher-only functionality only visible to teachers, home fragment updated to work as open world launcher, open world teacher and quest entities now dynamically added and placed, Teacher now decides what specific quizzes are available in open world, Imported quizzes made by other teachers now functional in open world, Audio feedback added when a quiz is completed, Teacher can now delete modules and/or their quizzes + delete quizzes from the open world quiz list, and graphical assets in the game are updated.

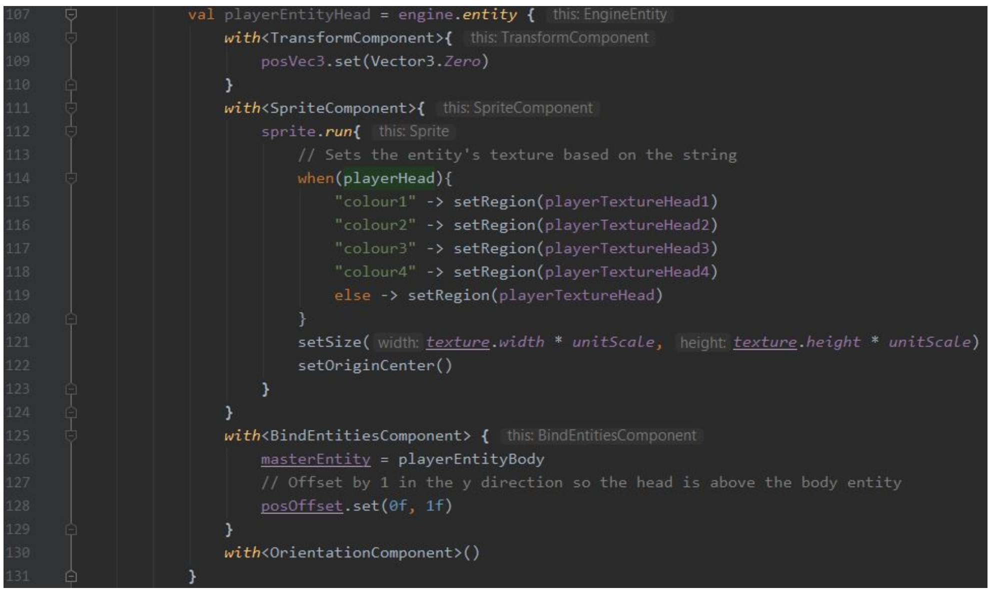
Bugfix:
Database error handling, not allowing to add non-existent modules/quizzes to be imported.
Profile and settings fragment no longer randomly breaks.
In-game collision and activation fixes

A.0.5 Miscellaneous



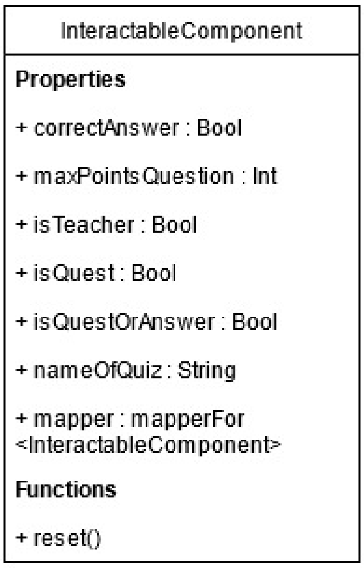
```
val playerEntityBody = engine.entity {  this: EngineEntity
    with<TransformComponent>{  this: TransformComponent
        // Where the entity is positioned in the game world
        posVec3.set( x: 47.5f-offsetPos,  y: 15f,  z: -1f)
    }
    with<MovementComponent>()
    with<SpriteComponent>{  this: SpriteComponent
        sprite.run{  this: Sprite
            // Sets the entity's texture based on the string
            when(playerBody){
                "colour1" -> setRegion(playerTextureBody1)
                "colour2" -> setRegion(playerTextureBody2)
                "colour3" -> setRegion(playerTextureBody3)
                "colour4" -> setRegion(playerTextureBody4)
                else -> setRegion(playerTextureBody)
            }
            setSize( width: texture.width * unitScale,  height: texture.height * unitScale)
            setOriginCenter()
        }
    }
    with<PlayerComponent> {  this: PlayerComponent
        playerName = playerUserName
        gameInst = game
        playerControl = playeContr
    }
    with<OrientationComponent>()
    with<TextComponent> {  this: TextComponent
        // Activates HUD to display current player score
        isText = true
        drawPlayScoreHUD = true
        // Makes text clearer
        font.region.texture.setFilter(Texture.TextureFilter.Linear,
                Texture.TextureFilter.Linear)
        // Scales the text up by 4
        font.data.setScale( scaleX: 4.0f,  scaleY: 4.0f)
    }
}
```

Figure A.1: playerEntityBody program code



```
val playerEntityHead = engine.entity {  this: EngineEntity
    with<TransformComponent>{  this: TransformComponent
        posVec3.set(Vector3.Zero)
    }
    with<SpriteComponent>{  this: SpriteComponent
        sprite.run{  this: Sprite
            // Sets the entity's texture based on the string
            when(playerHead){
                "colour1" -> setRegion(playerTextureHead1)
                "colour2" -> setRegion(playerTextureHead2)
                "colour3" -> setRegion(playerTextureHead3)
                "colour4" -> setRegion(playerTextureHead4)
                else -> setRegion(playerTextureHead)
            }
            setSize( width: texture.width * unitScale,  height: texture.height * unitScale)
            setOriginCenter()
        }
    }
    with<BindEntitiesComponent> {  this: BindEntitiesComponent
        masterEntity = playerEntityBody
        // Offset by 1 in the y direction so the head is above the body entity
        posOffset.set(0f, 1f)
    }
    with<OrientationComponent>()
}
```

Figure A.2: playerEntityHead program code



InteractableComponent

Properties

+ correctAnswer : Bool
+ maxPointsQuestion : Int
+ isTeacher : Bool
+ isQuest : Bool
+ isQuestOrAnswer : Bool
+ nameOfQuiz : String
+ mapper : mapperFor <InteractableComponent>

Functions

+ reset()

Figure A.3: UML class InteractableComponent

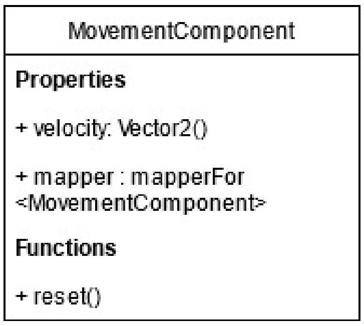


Figure A.4: UML class MovementComponentFig

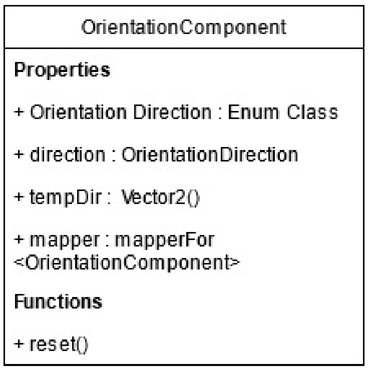

OrientationComponent
Properties
+ Orientation Direction : Enum Class
+ direction : OrientationDirection
+ tempDir : Vector2()
+ mapper : mapperFor <OrientationComponent>
Functions
+ reset()

Figure A.5: UML class OrientationComponent

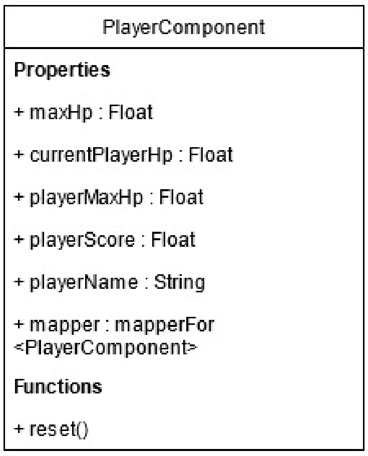


Figure A.6: UML class PlayerComponent

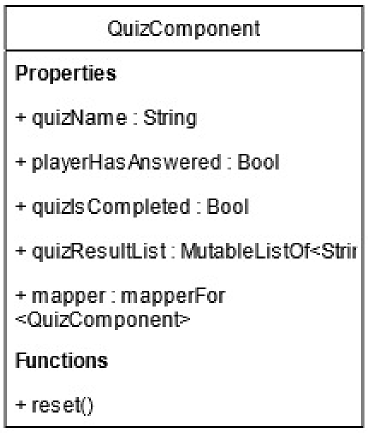

QuizComponent
Properties
+ quizName : String
+ playerHasAnswered : Bool
+ quizIsCompleted : Bool
+ quizResultList : MutableListOf<Strir
+ mapper : mapperFor <QuizComponent>
Functions
+ reset()

Figure A.7: UML class QuizComponent

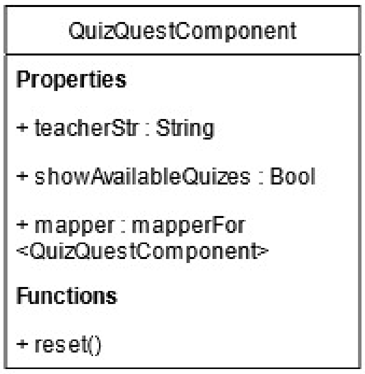

QuizQuestComponent
Properties
+ teacherStr : String
+ showAvailableQuizes : Bool
+ mapper : mapperFor <QuizQuestComponent>
Functions
+ reset()

Figure A.8: UML class QuizQuestComponent

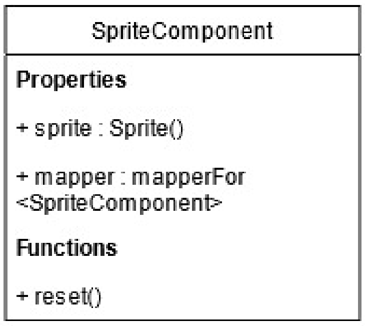


Figure A.9: UML class SpriteComponent

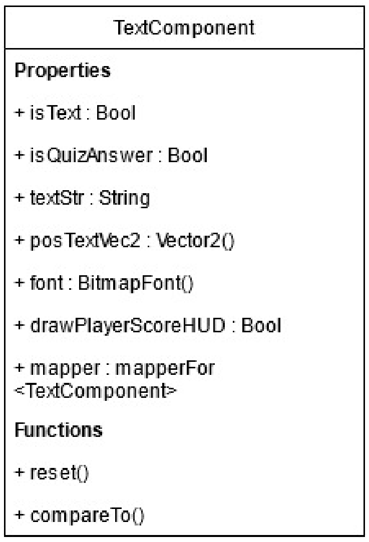


Figure A.10: UML class TextComponent

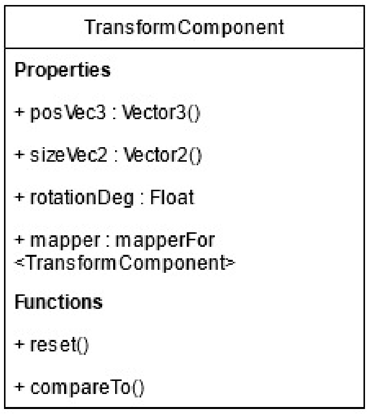


Figure A.11: UML class TransformComponent

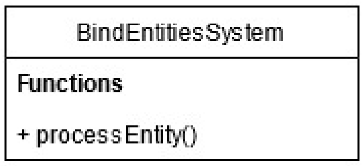


Figure A.12: UML class BindEntitiesSystem

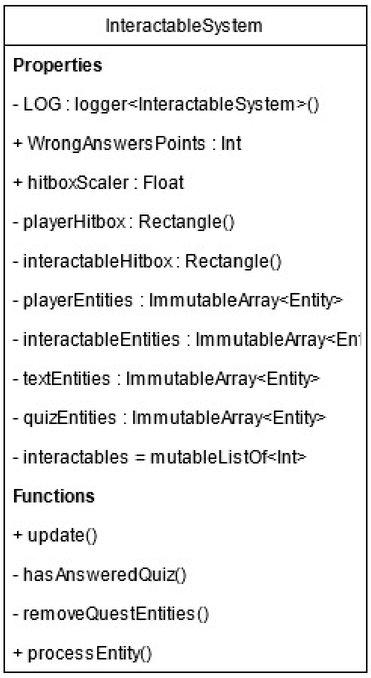

InteractableSystem
Properties
- LOG : logger<InteractableSystem>()
+ WrongAnswersPoints : Int
+ hitboxScaler : Float
- playerHitbox : Rectangle()
- interactableHitbox : Rectangle()
- playerEntities : ImmutableArray<Entity>
- interactableEntities : ImmutableArray<Ent
- textEntities : ImmutableArray<Entity>
- quizEntities : ImmutableArray<Entity>
- interactables = mutableListOf<Int>
Functions
+ update()
- hasAnsweredQuiz()
- removeQuestEntities()
+ processEntity()

Figure A.13: UML class InteractableSystem

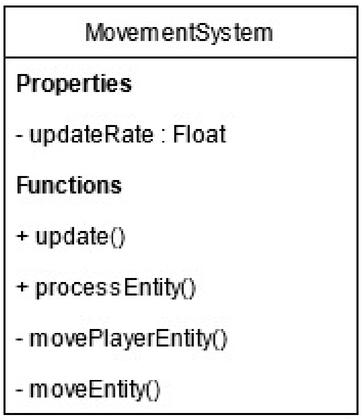


Figure A.14: UML class MovementSystem

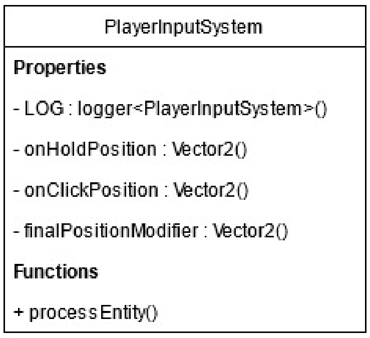

PlayerInputSystem
Properties
- LOG : logger<PlayerInputSystem>()
- onHoldPosition : Vector2()
- onClickPosition : Vector2()
- finalPositionModifier : Vector2()
Functions
+ processEntity()

Figure A.15: UML class PlayerInputSystem

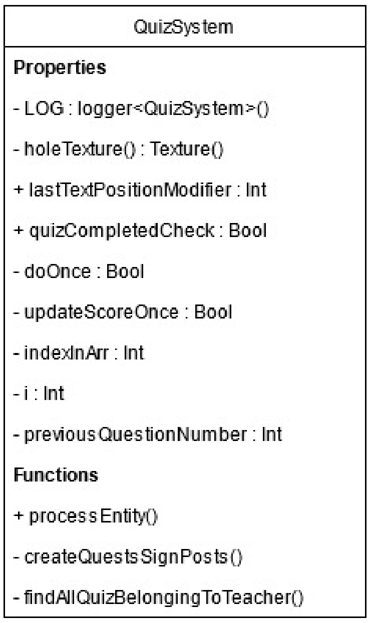

QuizSystem
Properties
- LOG : logger<QuizSystem>()
- holeTexture() : Texture()
+ lastTextPositionModifier : Int
+ quizCompletedCheck : Bool
- doOnce : Bool
- updateScoreOnce : Bool
- indexInArr : Int
- i : Int
- previousQuestionNumber : Int
Functions
+ processEntity()
- createQuestsSignPosts()
- findAllQuizBelongingToTeacher()

Figure A.16: UML class QuizSystem

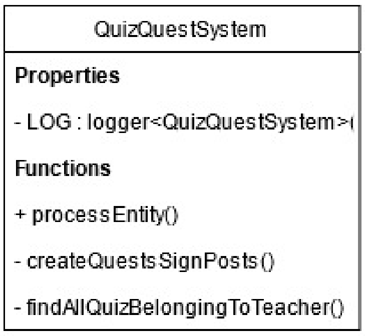

QuizQuestSystem
Properties
- LOG : logger<QuizQuestSystem>(
Functions
+ processEntity()
- createQuestsSignPosts()
- findAllQuizBelongingToTeacher()

Figure A.17: UML class QuizQuestSystem

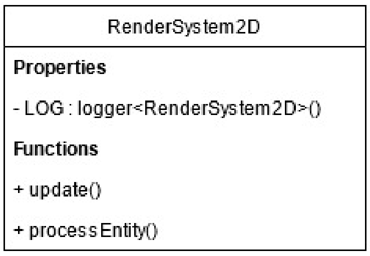


Figure A.18: UML class RenderSystem2D

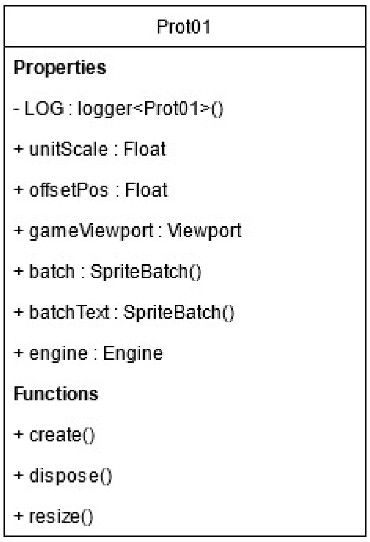


Figure A.19: UML class prot01

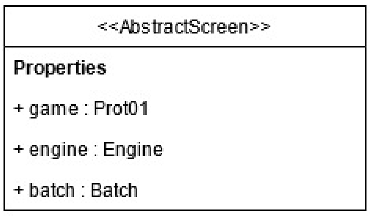


Figure A.20: UML class abstractScreen

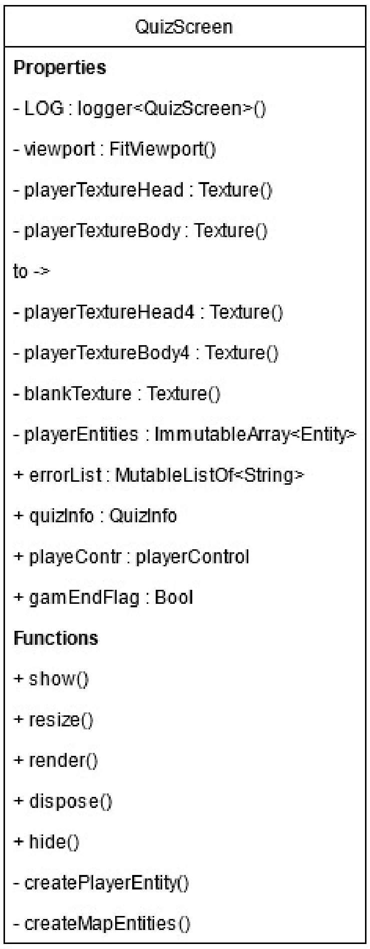

QuizScreen
Properties
- LOG : logger<QuizScreen>()
- viewport : FitViewport()
- playerTextureHead : Texture()
- playerTextureBody : Texture()
to ->
- playerTextureHead4 : Texture()
- playerTextureBody4 : Texture()
- blankTexture : Texture()
- playerEntities : ImmutableArray<Entity>
+ errorList : MutableListOf<String>
+ quizInfo : QuizInfo
+ playeContr : playerControl
+ gamEndFlag : Bool
Functions
+ show()
+ resize()
+ render()
+ dispose()
+ hide()
- createPlayerEntity()
- createMapEntities()

Figure A.21: UML class QuizScreen

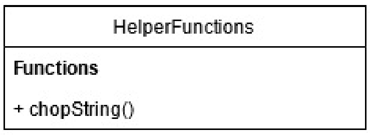


Figure A.22: UML class HelperFunctions

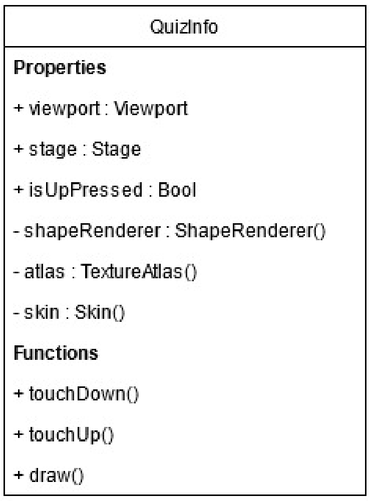


Figure A.23: UML class QuizInfo

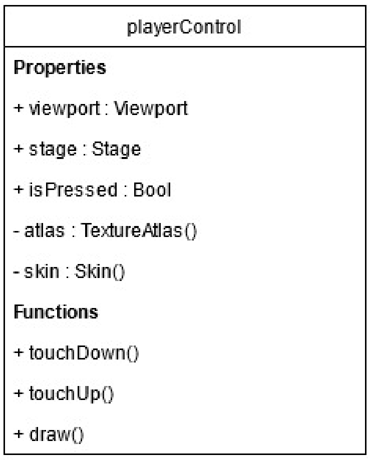


Figure A.24: UML class playerControl

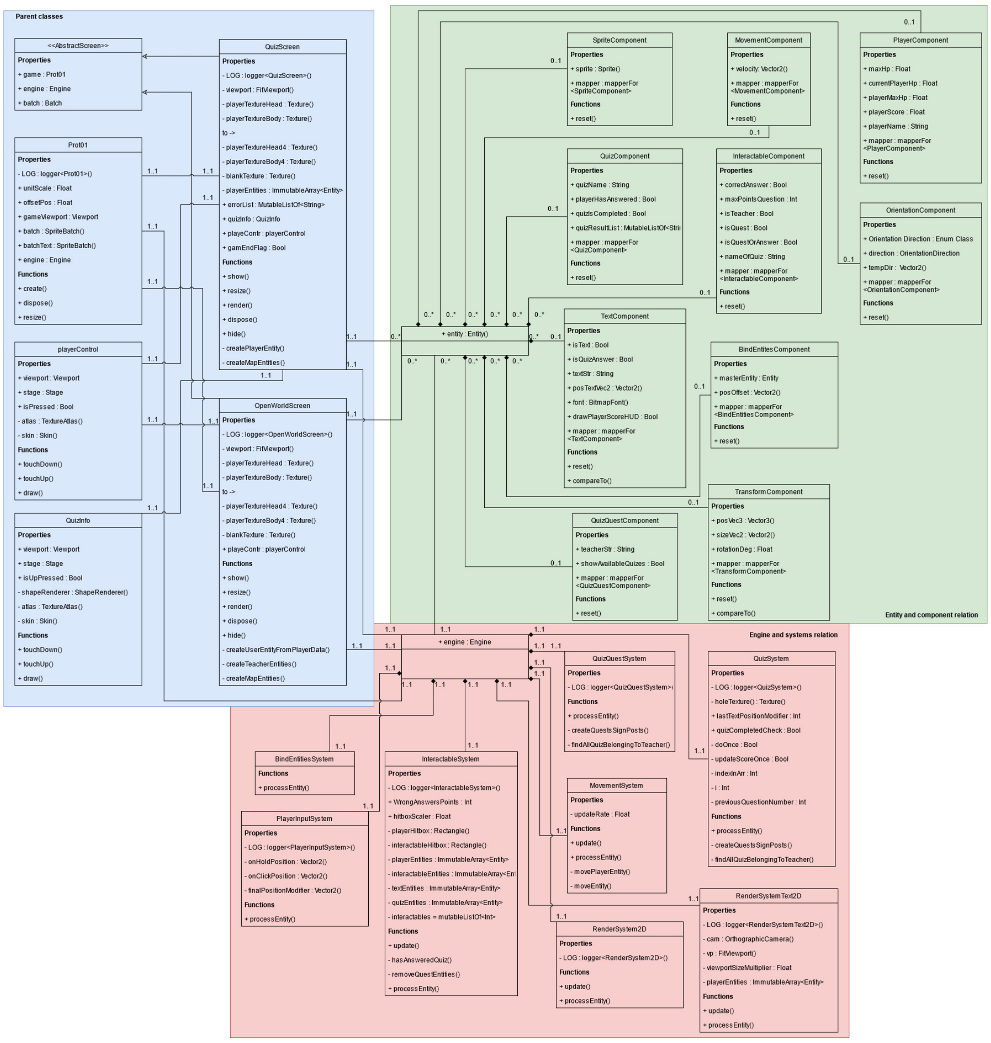
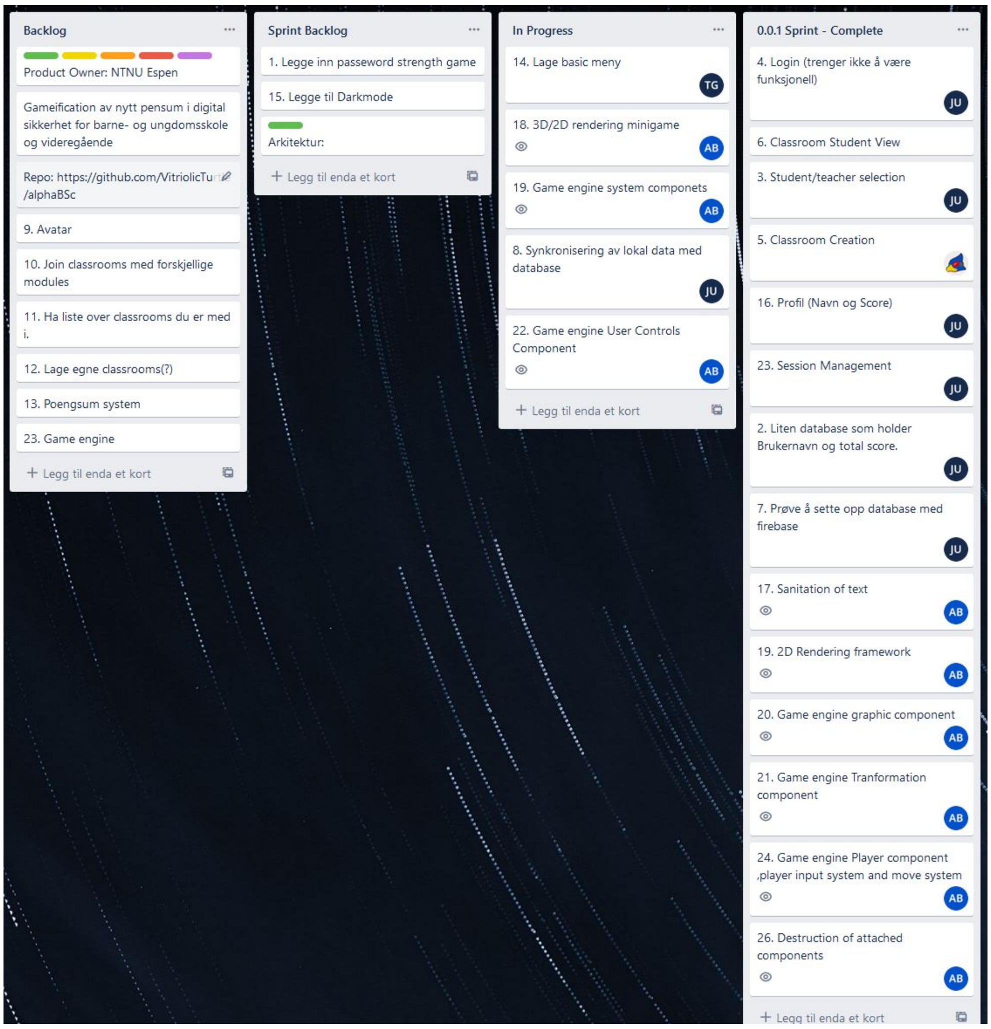


Figure A.25: Exploded UML class diagram ECS and parent classes



Backlog

- Product Owner: NTNU Espen
- Gameification av nytt pensum i digital sikkerhet for barne- og ungdomsskole og videregående
- Repo: https://github.com/VitriolicTu /alphaBSc
- 9. Avatar
- 10. Join classrooms med forskjellige modules
- 11. Ha liste over classrooms du er med i.
- 12. Lage egne classrooms(?)
- 13. Poengsum system
- 23. Game engine

+ Legg til enda et kort

Sprint Backlog

- 1. Legge inn passeword strength game
- 15. Legge til Darkmode
- Arkitektur:

+ Legg til enda et kort

In Progress

- 14. Lage basic meny — TG
- 18. 3D/2D rendering minigame — AB
- 19. Game engine system componets — AB
- 8. Synkronisering av lokal data med database — JU
- 22. Game engine User Controls Component — AB

+ Legg til enda et kort

0.0.1 Sprint - Complete

- 4. Login (trenger ikke å være funksjonell) — JU
- 6. Classroom Student View
- 3. Student/teacher selection — JU
- 5. Classroom Creation
- 16. Profil (Navn og Score) — JU
- 23. Session Management — JU
- 2. Liten database som holder Brukernavn og total score. — JU
- 7. Prøve å sette opp database med firebase — JU
- 17. Sanitation of text — AB
- 19. 2D Rendering framework — AB
- 20. Game engine graphic component — AB
- 21. Game engine Tranformation component — AB
- 24. Game engine Player component ,player input system and move system — AB
- 26. Destruction of attached components — AB

+ Legg til enda et kort

Figure A.26: Early sprint in Trello

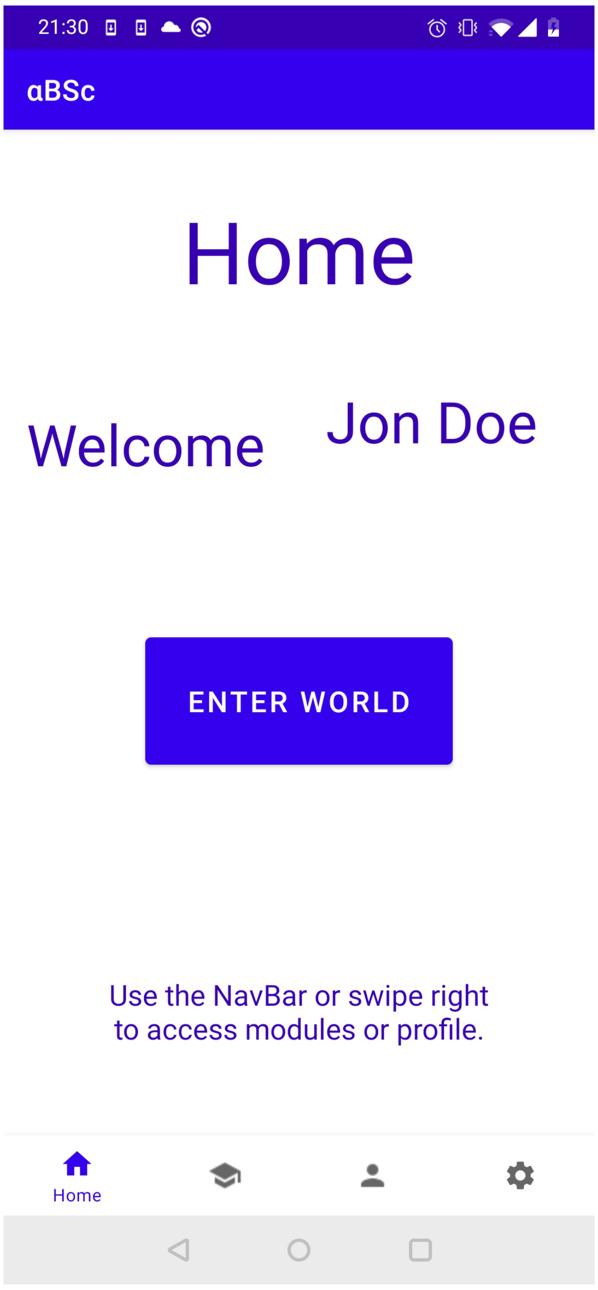


Figure A.27: Screenshot from app - The home screen that welcomes a user upon successful login

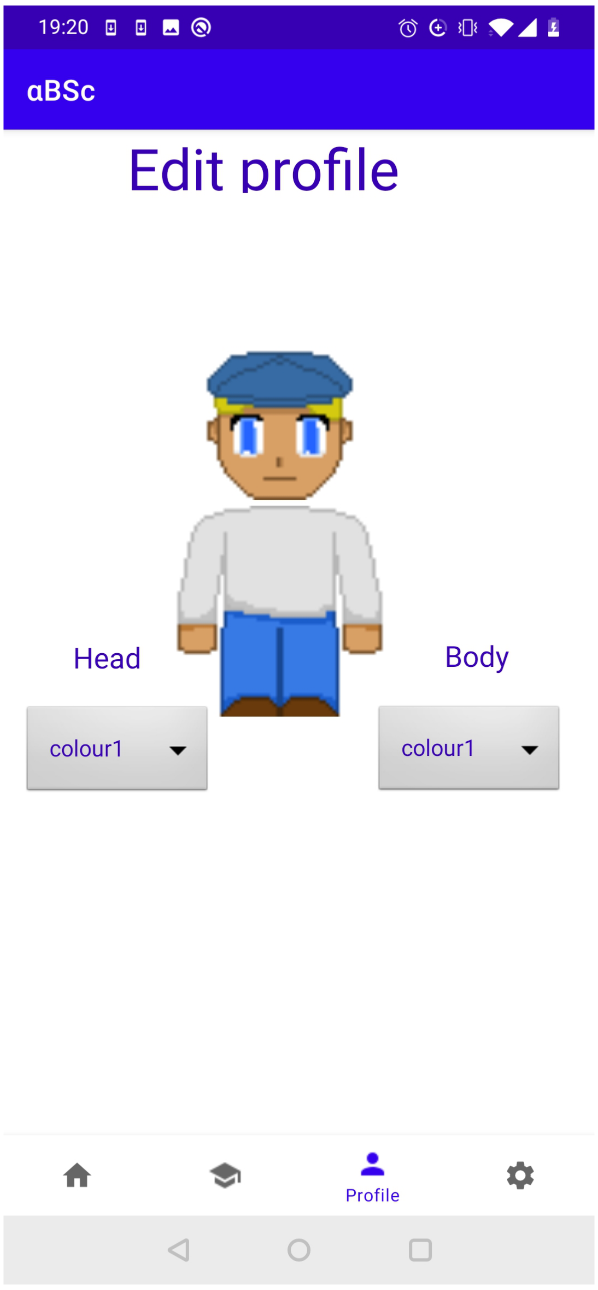


Figure A.28: Screenshot from app - The edit profile fragment, lets the user edit their avatar. This fragment can be accessed from a profile fragment

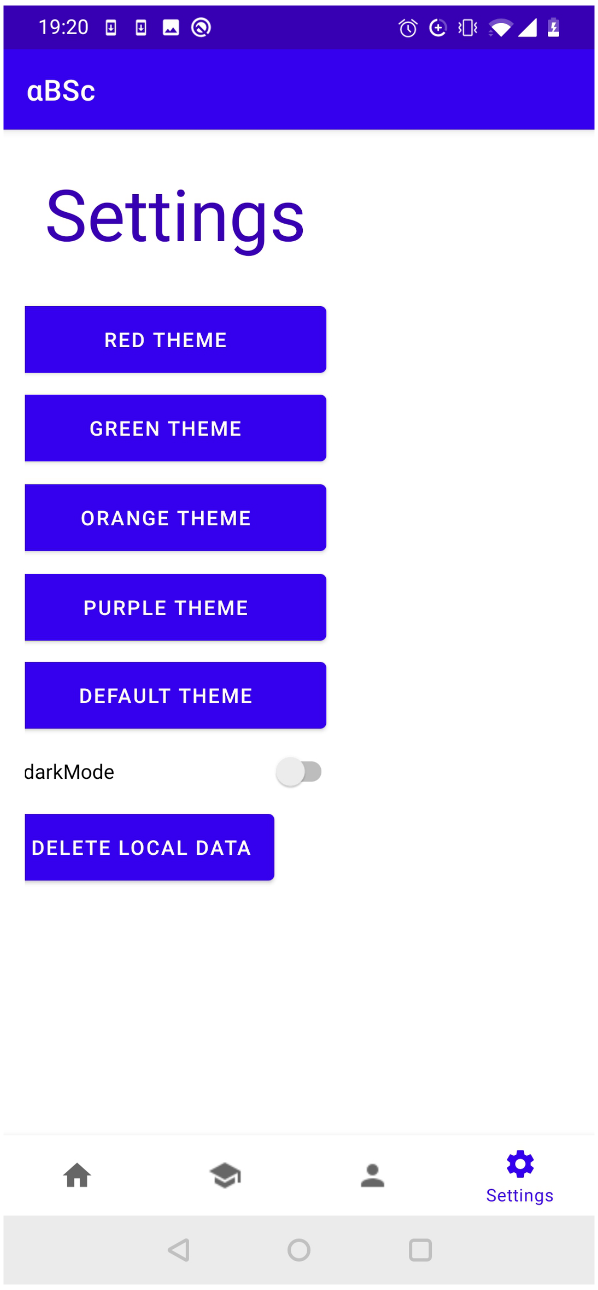
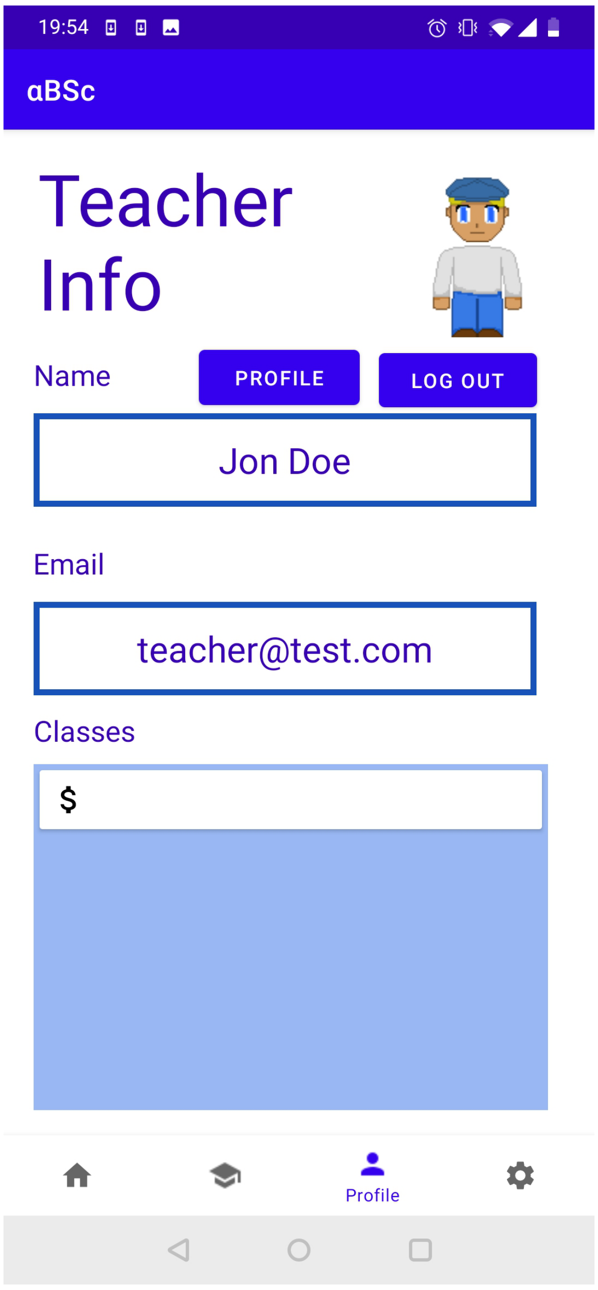


Figure A.29: Screenshot from app - This fragment can change themes and light/dark mode. Accessed through navigation bar



Teacher Info

Name

Jon Doe

Email

teacher@test.com

Classes

$

Figure A.30: Screenshot from app - The teacher's information fragment, accessed from their profile fragment

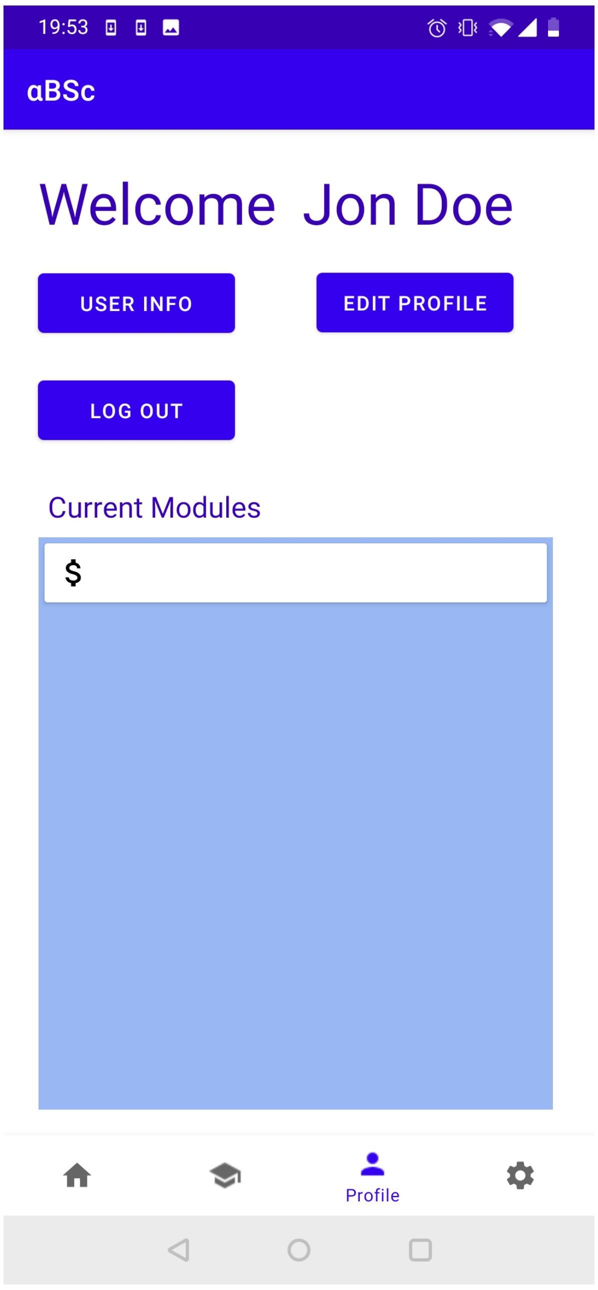
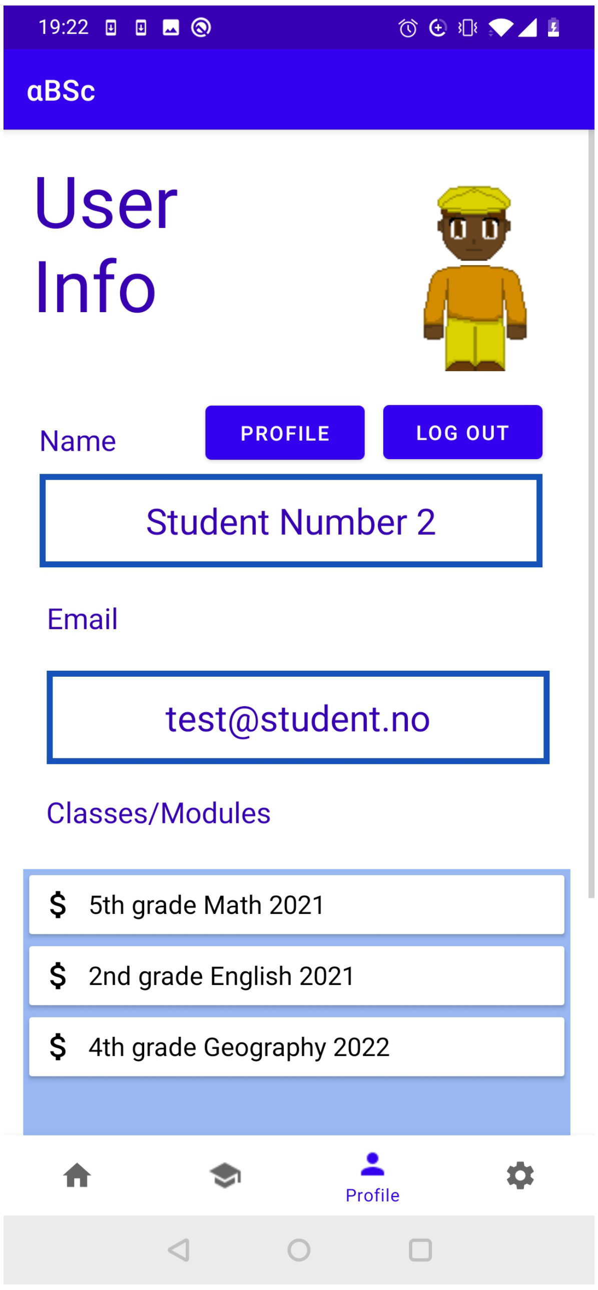


Figure A.31: Screenshot from app - The teacher's profile fragment, accessed from the navigation bar and menu. Other fragments may be accessed from here


19:22

αBSc

User Info

Name

PROFILE

LOG OUT

Student Number 2

Email

test@student.no

Classes/Modules

$ 5th grade Math 2021

$ 2nd grade English 2021

$ 4th grade Geography 2022

Profile

Figure A.32: Screenshot from app - The user information fragment, accessed from the user profile fragment

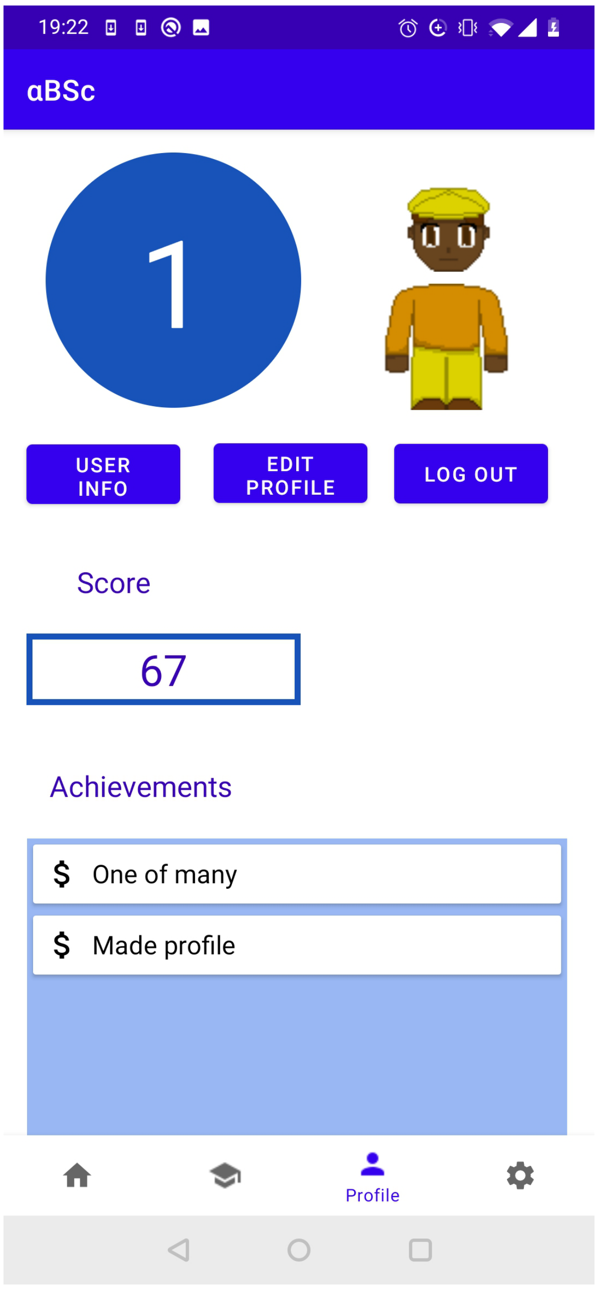


Figure A.33: Screenshot from app - The user profile accessible from the menu and navigation bar. The user may access other fragments from here

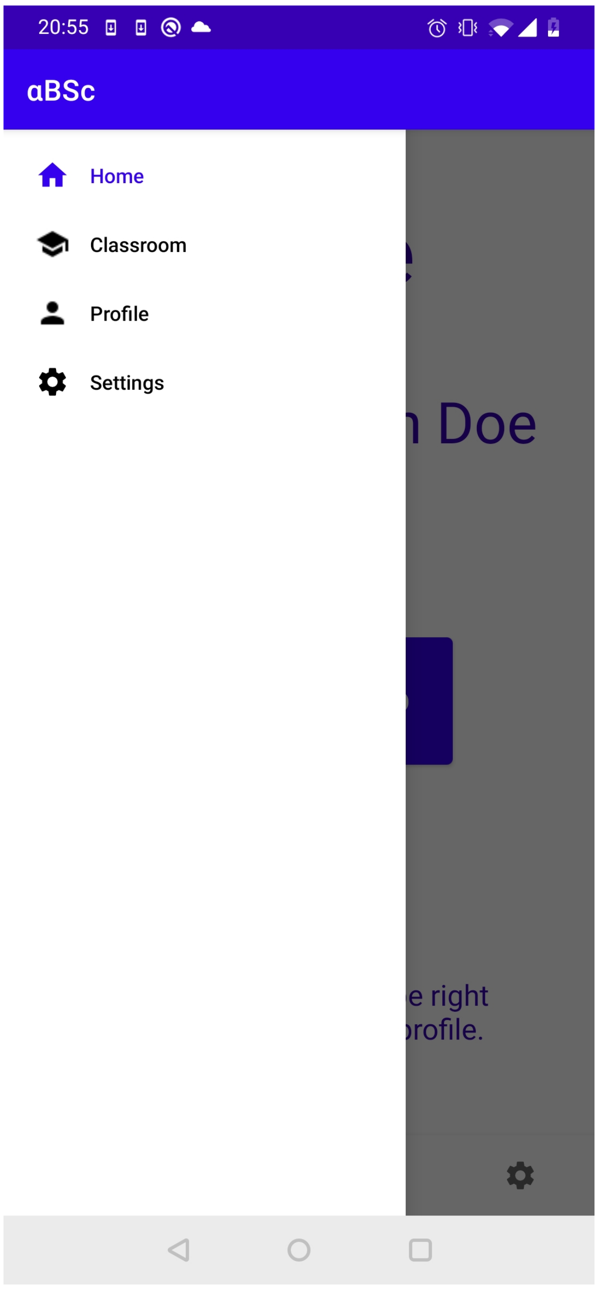


Figure A.34: Screenshot from app - The side menu, accessible by swiping to the right on any fragment

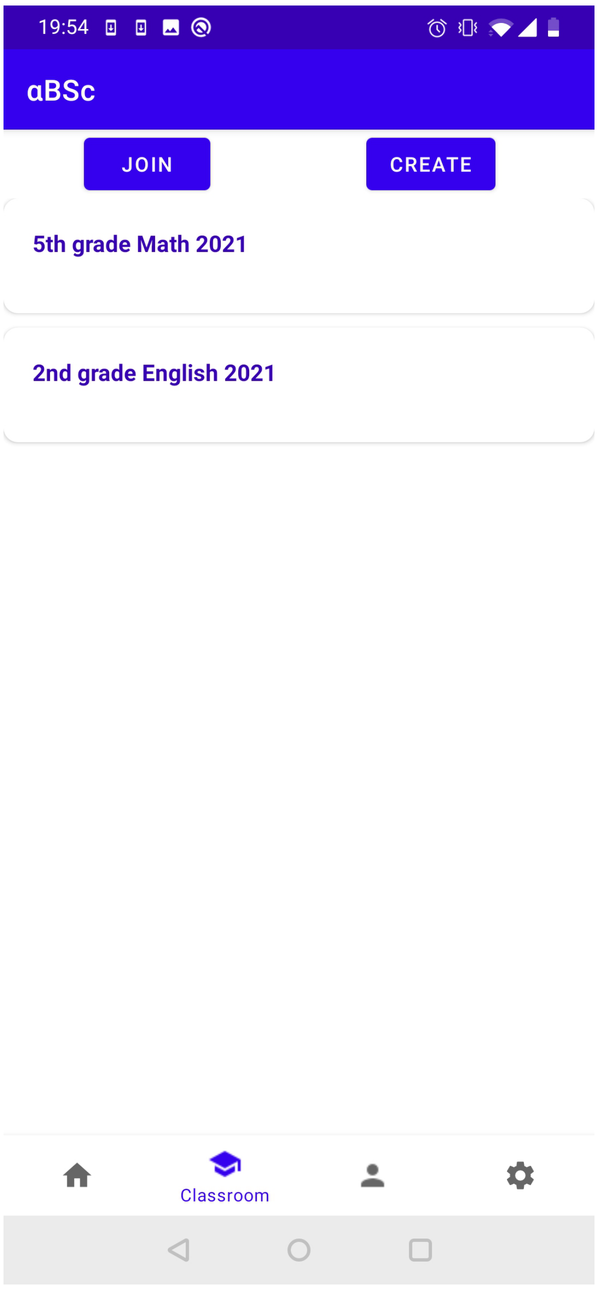


Figure A.35: Screenshot from app - The classroom list, accessible from the navigation bar. A user may click one to view it's content