Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: Planetarium

High Level Design Report

Boran Yıldırım, Kıvanç Gümüş, Ümitcan Hasbioğlu, Hüseyin Eren Çalık, Yiğit Bedişkan

Supervisor: Uğur Doğrusöz

Jury Members: Fazlı Can and Uğur Güdükbay

1. Introduction

In this report, the high-level design of Planetarium is proposed. Purpose of the system and design goals are explained. Afterwards, software and hardware architecture are proposed while making use of system decomposition and hardware-software mapping of the system. Finally, the access control and security of the system along with global software control and the boundary conditions are investigated.

1.1 Purpose of the system

Mankind has always dreamt about spreading far over the space and beyond. This dream is being fueled with the current events, such as the colonization of Mars. Therefore, the purpose of the system is to provide a game, in which people can experience an environment that revolves around space civilizations. The aim of the game Planetarium is to advance your civilization, make friends, battle enemies, share resources to be ruler of space within a great competition amongst many players.

In the game, each player will be controlling a distinct space civilization that will be assigned to them at the beginning. Then, players will improve their planet from a default base. To improve their planets, users have to mine their planet’s resources and use these resources to develop various structures. Through these various structures players can build weapons, spaceships and other types of technological equipment that will overtop them against other players. Players can also interact through a universal market by selling their resources and trading with each other. the uniqueness of this game is mainly based on the battle dynamics. Players will battle using the gyroscope of their mobile-phones when user attack and defend their own planet.

1.2 Design goals

1.2.1 Reliabilty

Planetarium is a freemium competetive game so there will be in-app purchases and ranking among players. Therefore, the system should be as reliable as possible. To achieve this aim;

- Data will be stored precisely to prevent any unfairness among players.
- System parts that provide realtime service will be ensured to be as reliable as possible.
- The game should be updated frequently and be bug free.

1.2.2 Extensibility

- The game should be able to include new features with ease, so it would be developed in a way that makes it easy to update.
- The system should be open to future additions.

1.2.3 User-Friendliness

- The game should be easy to use for all types of users and provide a user-friendly and a easy-to-use interface.
- The game should give recommendations and hints according to the user's needs and specifications.
- There should be a tutorial to teach players how to play the game.

1.2.4 Security

- Client side is not supposed to change server data directly by manipulating persistent local data.
- Firebase database provides encrypted data transfer with SSL.
- Users shouldn't be able to data and money manipulation.
- Client shouldn't take the information from Firebase database and give it to PUN server. It may cause hacking problems.

- Usage of two different serves cause security problems so there have to control mechanizms to check whether data transformed correctly.

1.2.5 Efficiency

- The game should respond to user while using the least possible amount of system resources and internet bandwidth as possible.
- In multiplayer session, player should play without any lag.

1.3 Trade-offs

In this section, we will explain which of the design goals should be prefered to sacrifice to make the overall system better. Trade-offs are necessary to find the most optimized version of the final product.

1.3.1 Reliability vs Efficiency

Since efficiency is important in online games which need rapid updates and changes in game screen, we prefer to give importance to efficiency instead of reliability. Because, if we not provide user proper game view, it will not also satisfied reliability constraint. We prefer to use UDP server instead of TCP to make the client and server connection as fast as possible. Although, TCP is more reliable protocol, we prefer to use reliable UDP to make our game efficient.

2. Current software architecture

OGame by Gameforge is very similar to our game since OGame is based on starting on an undeveloped planet and later aiming to create a galactic empire. It also includes gathering resources and using those resources to build better structures and weaponry. The difference is that OGame doesn’t actually include much of a graphical user interface. It is more based on numbers and you control your empire with your properties listed variously. In Planetarium, users will be able to realistically control their empire by actually interacting with each separate buildings, structures, weapons etc.

Some aspects of Planetarium is similar to the game Clash of Clans by Supercell. The main screen where the user will be able to display and interact with his main base is quite similar to the main screen of Clash of Clans. In Planetarium players will be building structures like R&D building, Weaponry building, defensive turrets, etc. In Clash of Clans building types are similar which are defensive, resource, army and others. This similarity is because that separate controllable buildings give easiness to the gameplay and also it creates an agile vibe. The best part of Planetarium is two enemies battling with each other. Currently, no other game has a gyroscope including space battle. Players will be attacking each other with their army while controlling the troops they want manually. This type of battling and the all other actions like gathering resources, using telescope etc. together makes Planetarium a fun and a unique experience.

3. Proposed software architecture

Planetarium’s main system is formed of many subsystems. These subsystems are vital to the main system’s success and must work in harmony among each other. In this section, Planetarium’s subsystems will be displayed. The interactions between the subsystems, the classes they consist of and their functions will be discussed.

3.1 Overview

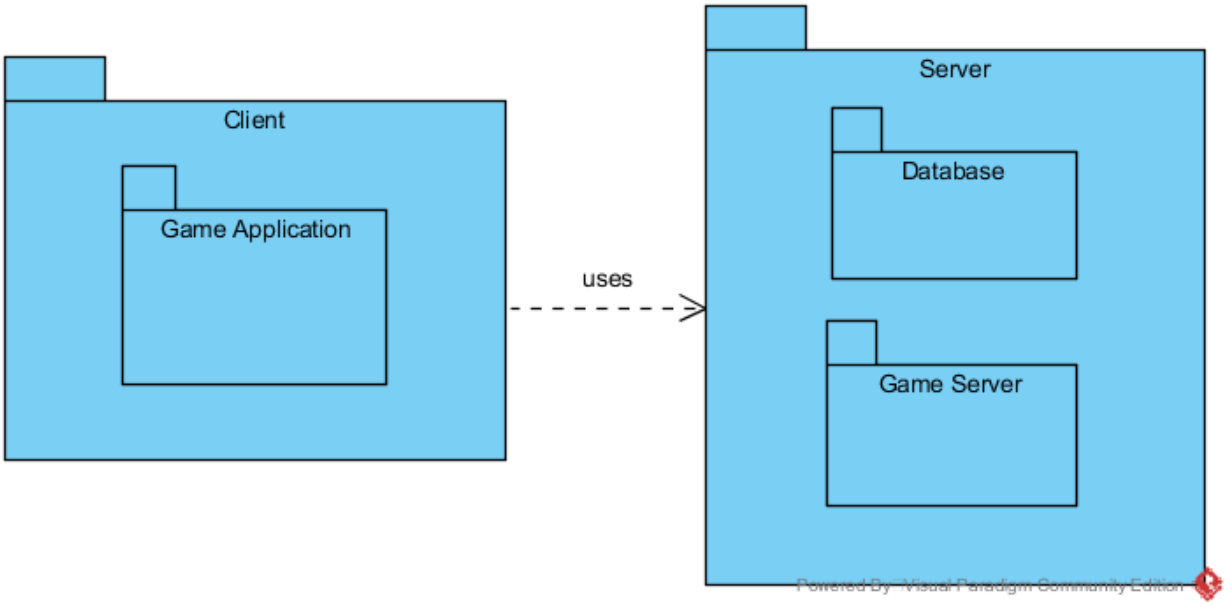
Planetarium’s system decomposition aims to present the structures of its systems and subsystems in great detail. Before getting into the diagrams, it is important to realize the importance of a well defined subsystem structure for a project. The subsystem decomposition showcases how each of our chosen subsystems (explained in greater detail in the following sections) interacts with each other. From the diagrams, it is possible to see the responsibilities of each subsystem and how the transition between different subsystems takes place. To get these traits of the subsystems correct before starting the implementation translates to fewer errors during coding, and less revision required.

For these reasons, we spent a lot of time and effort identifying the different subsystems that make up Planetarium when combined. We focused our resources into this aspect for we know this is the one with greatest return in the long run.

After realizing the importance of a proper subsystem decomposition, we are ready to dive deeper into the subject.

3.2 Subsystem decomposition

In this section, we will the describe the major components, i.e. subsystems, of our system. Our two main subsystems are client and server. This type of architectural design is the best way to go for a mobile game which includes many players interacting with each other and also, for huge datas to be stored and transferred to the players even though they are playing the game or not.

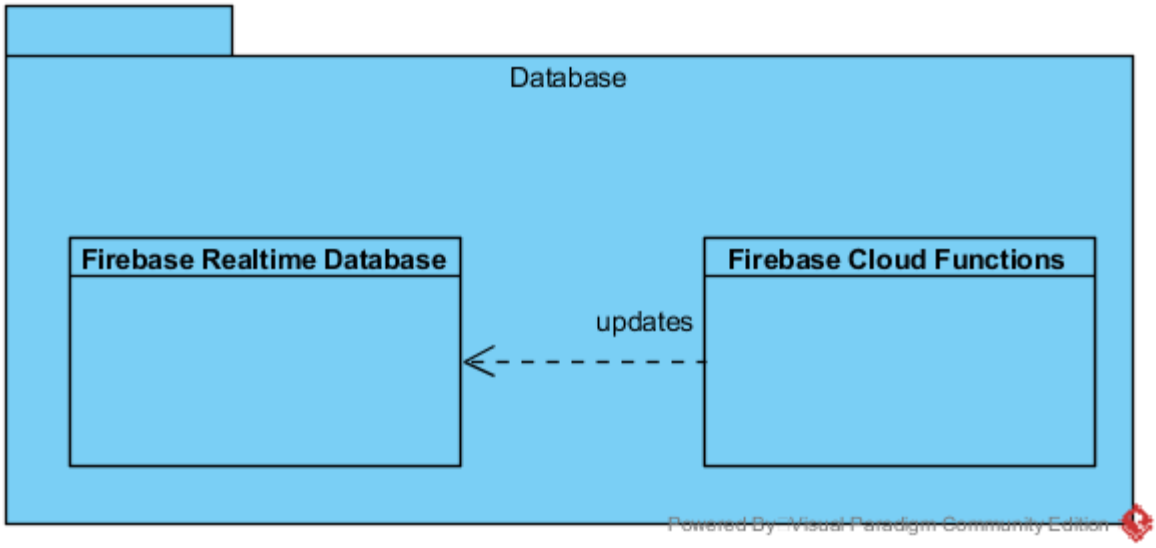


3.2 Server Subsystem

Planetarium is a massively multiplayer online mobile game. Therefore, the server subsystem plays a huge role in our game.

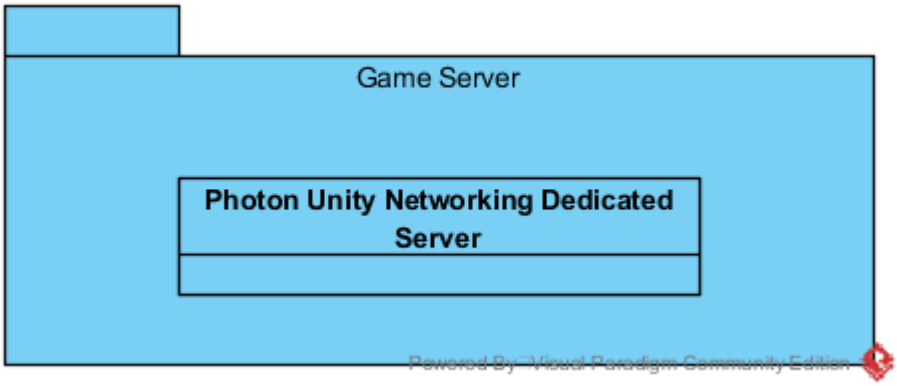
3.2.1 Database Subsystem

Database subsystem will receive data from the client side, store all the information for each client (e.g. log in information, planet information etc.) and continuously update the firebase database with its server-logic.



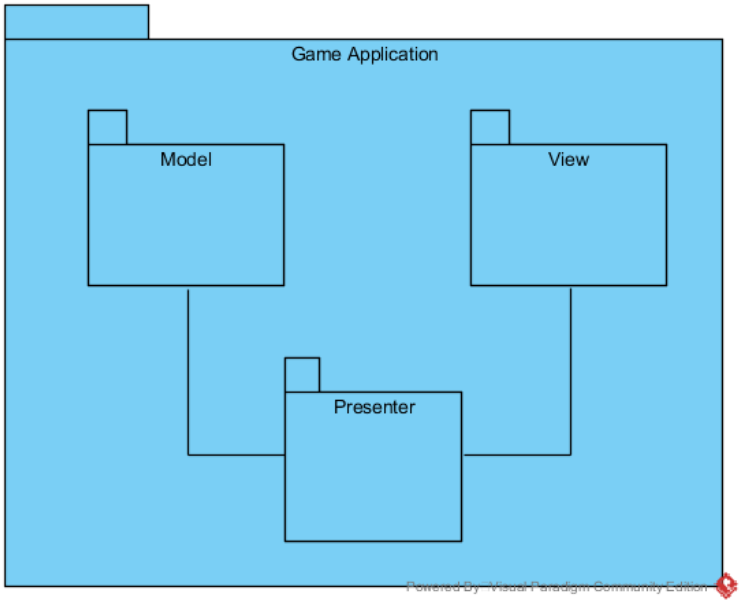
3.2.2 Game Server Subsystem

Game Server subsystem is responsible for player versus player battles that are an essential part of Planetarium. It will connect two clients(players) with each other and transfer data to one another in real time.



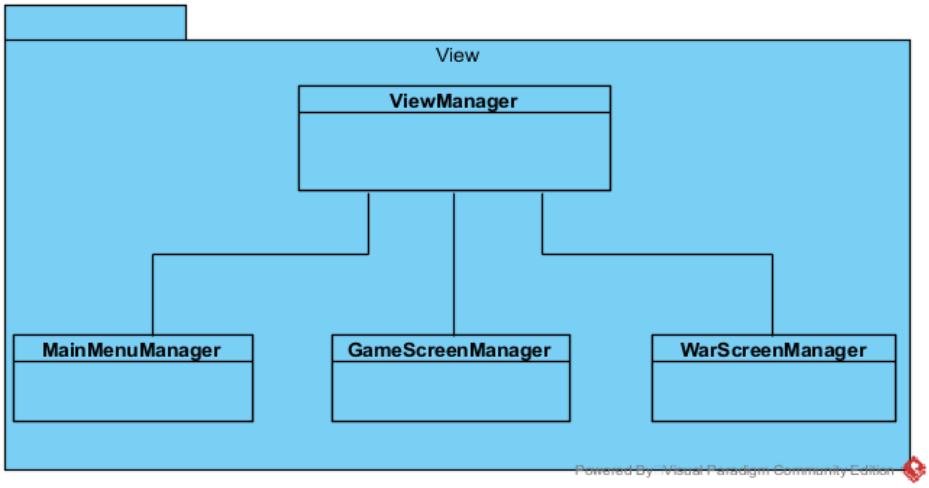
3.3 Client (Game Application) Subsytem

Client subsystem is the user end of Planetarium. This subsystem is that is executed in the mobile phones of the players. Client is responsible for displaying the game graphics and managing user's interactions with the game.



3.3.1 View Subsytem

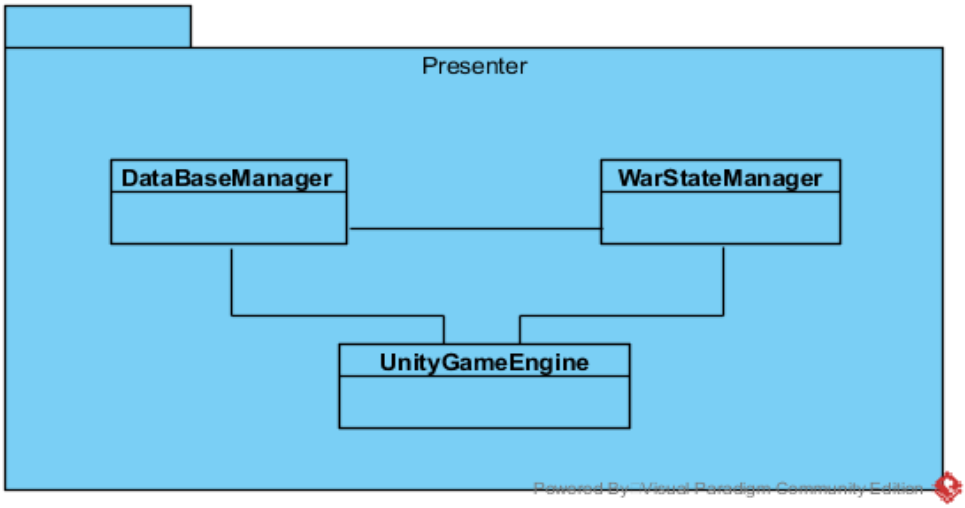
View subsystem is the user interface part of the client subsystem. It is responsible for handling the user interface components.



3.3.2 Presenter Subsytem

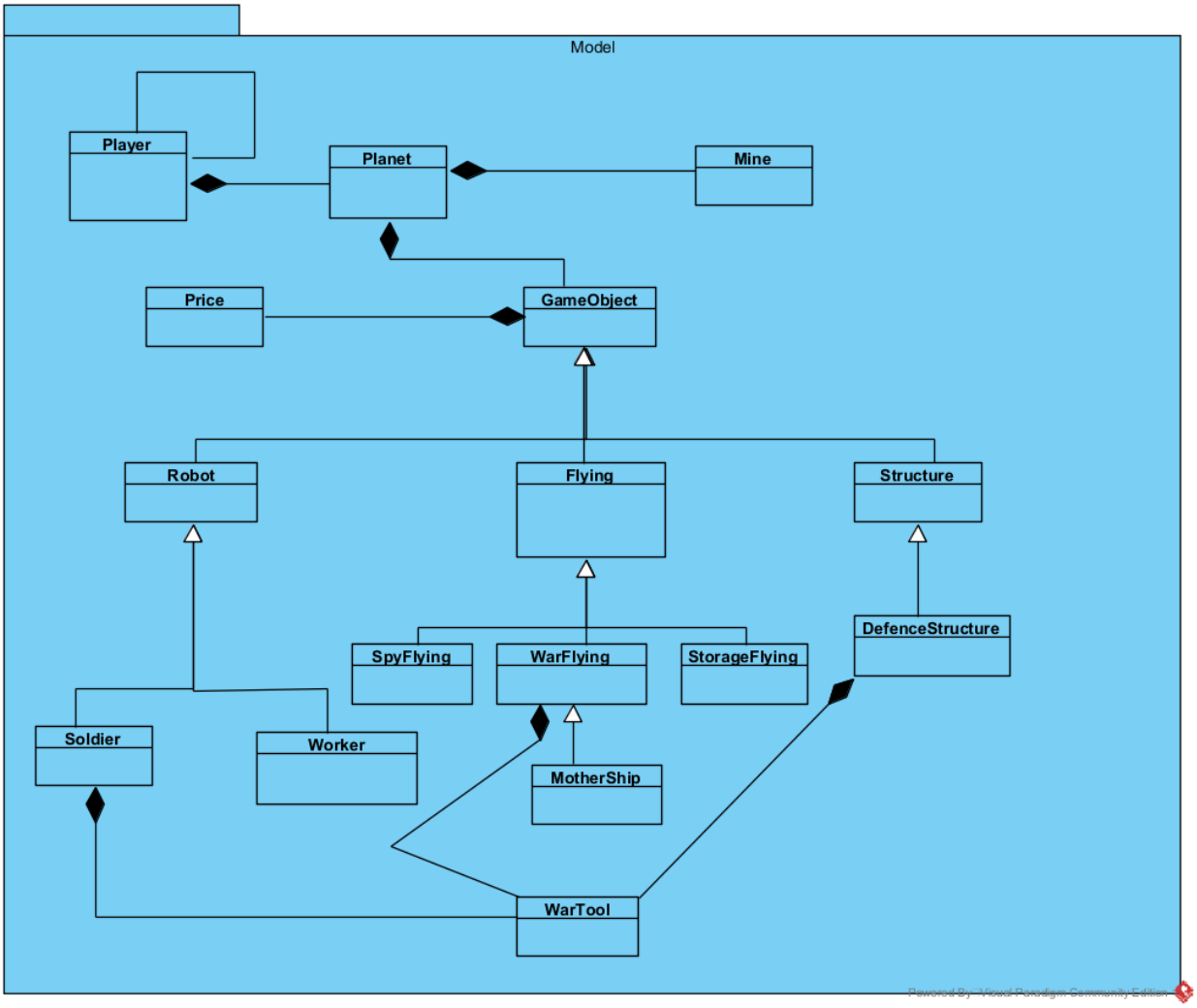
Presenter subsystem is the part where the main game engine and logic is. This subsystem is responsible for several things;

- Updating the view subsystem,
- Receving data from and updating the model subsystem,
- Handling the interaction between client subsystem and game server subsystem,
- Handling the data transfer between client and server subsystems.



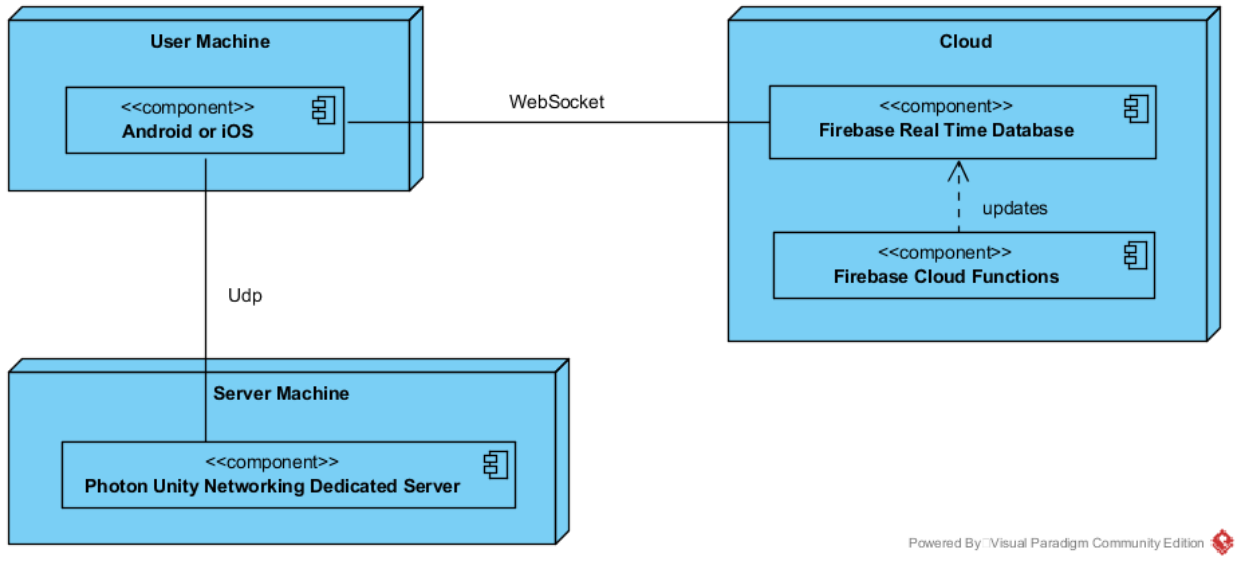
3.3.2 Model Subsytem

Model subsystem is where the necessary data is stored in the client. It gets updated by the presenter subsystem when the database is updated due to user actions.



3.3 Hardware/software mapping

Hardware of the clients are mobile phones. In the server side Google Cloud Platform, Firebase is used. Client/server database communication will be handled by Firebase Realtime Database. Photon Cloud is used by Photon Unity Network (PUN) which can establish multiplayer feature of the game. The game will be implemented in Unity platform with C#.



3.4 Persistent data management

- Firebase take care of user id, name, email, phone number etc.
- Planet properties such as number of coins, soldiers, planes, vehicles etc.
- Market store data for purchasing will be persistent on database.
- The game data will be stored on Google Cloud so data protection and backup is done by Google.

3.5 Access control and security

- All users will be able to edit their email, password, username. The password is hashed with a password-based encryption algorithm [1].
- The game requires an account so a user can sign up with email/password or Facebook or Google account.
- For security reasons, users can never create multiple accounts with the same email address.
- User's permission is requested for Facebook friend list which provides us to access to the list of friends that also use our game.

3.6 Global software control

All data will be managed from server to prohibit cheating. Server will not trust the data come from client and data change,verification will be done from server.

When real time multiplayer game session starts, Photon Unity Network server will run the game and clients, which will make predictions based on user inputs while waiting the server, will be updated and synced according to server result data.

3.7 Boundary conditions

3.7.1 Start

Both Android & iOS users need to download the app to their mobile phones. The initial phase of the user should be to sign up. The user will be requested his/her credentials. The user will be given the opportunity to authenticate with Google and Facebook accounts. In the case that the user fails to sign up, the sign up page will be displayed again.

3.7.2 Termination

The game state will be saved and paused and, if the user wishes to start the game again, respective data will be loaded.

3.7.3 Failure

The multiplayer game session will fail to work if there is no internet connection. If the game crashes, Firebase Crashlytics will log and send us the fail/crash information.

6. References

[1] https://github.com/firebase/scrypt#password-hashing