Bilkent University

Department of Computer Engineering

Senior Design Project

Project short-name: Planetarium

Analysis Report

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

Supervisor: Uğur Doğrusöz

Jury Members: Fazlı Can and Uğur Güdükbay

Progress Report

March 19, 2017

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/1.

Contents

1. Introduction

Mankind has always dreamt about spreading far over the space and beyond. In today’s world, technological developments have conquered the globe, making anyone able to reach any information easily. Our purpose is to create 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.

Each player will be controlling a distinct space civilization that will be assigned to them at the beginning of the game. 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. Player 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.

This report will contain detailed information about this space game such as: the system and design which distinguishes Planetarium from the other similar examples. The functional requirements and, the possible scenarios expected to be encountered by the user will be examined in this report. Also, there will be some mock-up images which help reader to better understanding of game in terms of appearance and graphical visuals. Thus, it is essential to show the detailed information and user interface of the game to make the report more understandable.

2. Current System

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 System

In this section, the necessary requirements (functional & non-functional) for the game will be specified.

3.1.Functional Requirements

3.1.1. Signup and Login

- User should be able sign up via Google and Facebook accounts. In the case absence of these accounts, the user can sign up via Planetarium.
- The user should login using these accounts to play the game.
- In case of losing or forgetting password, the password should be reset via e-mail.
- After signing up, the user should choose a nickname for his/her planet.

3.1.2. Short Game Mode

- The user can create a game room for a short game.
- Other users can see available game rooms and participate.
- At the beginning of each game, system should assign a planet.

3.1.3. Long Game Mode

- Initially, the system should assign a particular planet to player.
- The user’s planet is integrated into a big galaxy where other users have planets.

3.1.4. Gameplay

- The system should give a default planet setup initially.
- The user should be able to construct and develop buildings, structures, weapons by using mines, coins or diamonds.
- The user can produce troops and upgrade them by using mines, coins and diamonds.
- The user should be able attack to other planets and exploit their resources.
- The user can gather mines from his/her planet resources.
- The player should command the attack by using gyroscope or 2D attack mode.
- The player should defend his/her planet by using gyroscope or 2D defence mode. If the user do not want to defend by himself/herself, the system should handle defense.
- The user can trade his/her mines via market and sell mines in quick sell market.

3.2.Non-Functional Requirements

3.2.1.Reliability

- The game should be updated frequently and be a bug free game.
- The game should collect real time data for analyzing users for better experience.
- The game should give recommendations specific to the user.

3.2.2.Usability

- 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 according to the user’s needs and specifica-tions.
- The game will provide necessary information on how to use it.

3.2.3.Accessibility

- Accessibility is not a problem for the game because the target audience is smartphone users where number of smartphone users is 2.53 billion.[1]
- The game size should be as small as possible.

3.2.4.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.

3.2.5.Portability

- The game should be able to run on different hardware and software platforms, in our case on both Android and iOS devices.

3.2.6.Efficiency

- The game should respond to user while using the least possible amount of system resources and internet bandwidth as possible.
- The game should provide navigation through the game with as few steps as possible.

3.3.Pseudo Requirements

- Git and GitHub for version controlling and tracking will be used.
- GitHub will also be used as an issue tracking system.
- The game will have 2 servers, Firebase and Unity and will be available on App Store & Google Play.

3.4.System Models

To start off the system model, the report will be explaining the scenarios that a user in the game may go through. To explain each scenario in detail, they are divided into subsections as: Use Case Name, Actors, Entry Conditions, Exit Conditions and Main Flow of Events. Secondly, the system model will include a use-case diagram. The use-case diagram will be used to describe the events and actions between the user and the system.

3.4.1.Scenarios

In scenarios section, we will discuss how our app will function during different user program interactions. Importance of this section is demonstrating the storyline of the game, as well as detecting possible alternative solutions to problems we are trying to solve.

Scenario 1

Use Case Name: Login

Actors: John

Entry Conditions:

• User John is on Login screen

Exit Conditions:

• User John is on his planet scene

Main Flow of Events:

1. User John enters his email as "john@mail.com."
2. User John enters his password.
3. User John taps the “Login” button.

4. Firebase checks if the username and the password is correct.
5. Firebase confirms that the credentials are correct.
6. Firebase returns a callback for confirmation.
7. User is navigated to his game scene.

Scenario 2

Use Case Name: Create New Account

Actors: John

Entry Conditions:

• User John is on Login screen

Exit Conditions:

• User John is on his planet scene

Main Flow of Events:

1. User John taps on the “Sign up with Email” button on the screen.
2. User is navigated to Create New Account page.
3. User John enters his username as “John”
4. User John enters his email address as "John@mail.com"
5. User John enters his choice of password twice.
    a. User John taps the “Register” button.
6. It was checked whether the two passwords entered are same.
7. Firebase checks if the username “John” exists in the database.
8. Firebase checks if the email address "john@mail.com" exists in the database.
9. Firebase creates a new account.
10. User is navigated to his game scene.

Scenario 3

Use Case Name: Sign in with Google

Actors: Jack

Entry Conditions:

• User John is on Login screen

Exit Conditions:

• User John is on his planet scene

Main Flow of Events:

1. User John taps on the “Sign in with Google” button on the screen.
2. Firebase navigates to Google Login website.
3. User John enters his email address as "john@gmail.com."
4. User John enters his Google password.
5. User John taps the “Sign in with Google” button.
6. Firebase checks if the email address "john@gmail.com" exists in the database.
7. Firebase creates a new account.
8. User is navigated to his game scene.

Scenario 4

Use Case Name: Sign in with Facebook

Actors: John

Entry Conditions:

• User John is on Login screen

Exit Conditions:

• User John is on his planet scene

Main Flow of Events:

1. User John taps on the “Sign in with Facebook” button on the screen.
2. Facebook SDK navigates to Facebook Login website.
3. User John enters his email address as "john@mail.com."
4. User John enters his Facebook password.
5. User John taps the “Sign in with Facebook” button.
6. Firebase checks if the email address "john@mail.com" exists in the database.
7. Firebase creates a new account.

8. User is navigated to his game scene.

Scenario 5

Use Case Name: Change Preferences

Actors: Jack

Entry Conditions:

• User Jack is on Me screen

Exit Conditions:

• User Jack is on Me screen

Main Flow of Events:

1. User John taps the “Menu” icon on the right side.
2. User John taps the “Preferences” button on menu.
3. Game is navigated to the Preferences page.
4. User John views preferences for his account.
5. User John navigates to the “Change Username” page of Preferences.
6. User John changes his username.
7. User John taps the “Save Preferences” button.
    a. Planetarium checks if all the given information are accurate and valid.
    b. Planetarium saves the changes of User Jonh’s account.
    c. Planetarium navigates to the “Menu”.

Scenario 6

Use Case Name: Assigning Workers for Mining

Actors: John

Entry Conditions:

• User John is on Game screen

Exit Conditions:

• User John is on Game screen

Main Flow of Events:

1. User John taps the “Worker” icon on the screen.

2. User John selects a worker robot among different levels of workers sorted vertically depending on their current level.
3. User John assigns worker/s to mine for a desired duty time.
4. When the assigned duty time finishes for the worker, worker will return to the base and the user will obtain the mines that the worker had brought in.
5. User is navigated to his game scene.

Scenario 7

Use Case Name: Building a structure

Actors: John

Entry Conditions:

• User John is on Game screen

Exit Conditions:

• User John is on Game screen

Main Flow of Events:

1. User John selects a structure which is affordable for John by his level, mine and coins.
2. User John select an empty space for construction.
3. A worker robot will start to construct the building.
4. User John will have building.

Scenario 8

Use Case Name: Manufacturing a war vehicle

Actors: John

Entry Conditions:

• User John is on Game screen

Exit Conditions:

• User John is on Game screen

Main Flow of Events:

1. User John taps on the “Vehicle” icon on the screen.

2. User John selects spaceship from a list of 2 types of war vehicle, spaceship and ground vehicle.
3. User John selects a spaceship which is affordable for John by his level, mine and coins.
4. User John will have a spaceship.

Scenario 9

Use Case Name: Selling mines

Actors: John

Entry Conditions:

• User John is on Game screen

Exit Conditions:

• User John is on Market screen

Main Flow of Events:

1. User John taps on the “Market” icon on the screen.
2. User John choose Sell option.
3. User John selects amount of mines which he wants to get coin.
4. User John waits for another user to buy his mines.
5. User John will get coin by selling his mines.

Scenario 10

Use Case Name: Buying a spaceship

Actors: John

Entry Conditions:

• User John is on Game screen

Exit Conditions:

• User John is on Market screen

Main Flow of Events:

1. User John taps on the “Market” icon on the screen.
2. User John choose Buy option.
3. User John selects spaceship menu.
4. User John search a spaceship according to his amount of coin.
5. User John buys a spaceship.

Scenario 11

Use Case Name: In app purchasing coin

Actors: John

Entry Conditions:

• User John is on In-app Purchase screen

Exit Conditions:

• User John is on In-app Purchase screen

Main Flow of Events:

1. User John selects 1000 coins for his needs.
2. User John pays $2.
3. User John have 1000 coins.

Scenario 12

Use Case Name: Sending a spy

Actors: John

Entry Conditions:

• User John is on Game screen

Exit Conditions:

• User John is on Game screen

Main Flow of Events:

1. User John taps on the “Telescope” icon on the screen.
2. User John selects a planet for sending by looking around using the gyroscope of the phone.
3. User John sends spy for a desired duty time.
4. User John gets information about the planet which he wants to learn about.

Scenario 13

Use Case Name: Attacking an Enemy

Actors: John, Jack

Entry Conditions:

• User John is on Game screen

Exit Conditions:

• User John is on Game screen

Main Flow of Events:

1. User John taps the “Telescope” icon on the screen.
2. User John selects a planet to attack by looking around using the gyroscope of the phone.
3. User John presses the “Battle” button.
4. User John’s current troops are displayed on the screen.
5. User John picks the number of different types of troops he wants to attack to the enemy and then presses the “Engage” button.
6. User Jack is notified, user John is transferred to “Wait” screen and is obliged to wait for a countdown of 1 minute.

7. User Jack can enter the defense to control his troops or let his planet defend itself automatically.
8. User John enters the “Outer Battle” screen where he battles the outer troops of User Jack and the defender’s planet shield.
9. After the planet shield is destroyed user John enters the “inner planet” scene.
10. User John attacks the structures and troops of user Jack while user Jack defends.
11. When user John destroys the main base of user Jack, user John wins the battle.
12. User John collects the mines that have dropped from the user Jack’s planet due to the destruction he makes and goes leaves the planet.
13. User John gets back to his planet.

Scenario 14

Use Case Name: Defending to an Enemy

Actors: John, Jack

Entry Conditions:

• User John is on Game screen

Exit Conditions:

• User John is on Game screen

Main Flow of Events:

1. User John is attacked by user Jack.
2. User John is notified that he is being attacked and gains a wait time of 1 minute. User John can decide not to enter the battle personally and wait for his troops to battle automatically.
3. Regardless of user John entering the battle or not, after the countdown of 1 minute, user Jack attacks user John “outer planet” troops and the planet shield.
4. User John can enter any troop in outer planet and control that troop to attack user Jack’s army.
5. If user Jack cannot destroy the planet shield of user John, John wins the battle.
6. If user Jack destroys the planet shield of user John, user Jack enters the “inner planet” scene.

7. User John can control any troop manually to defend his planet.
8. If user John destroys the mothership of user Jack, he wins the battle.
9. If user Jack destroys the main base of user John, user John loses the battle.
10. When the battle is lost, user John loses the equivalent of mines to the structures and the troops that user Jack destroyed.
11. User Jack leaves the planet, user John gets back to the “Game Screen” where he remains with the destroyed structures due to the aftermath of the battle.

Scenario 15

Use Case Name: Using the telescope

Actors: John

Entry Conditions:

• User John is on Game screen

Exit Conditions:

• User John is on Game screen

Main Flow of Events:

1. User John presses the “Telescope” icon.
2. User John enters the “Telescope Screen” in which he can the planets around his planet using the gyroscope of the mobile phone.
3. User John can look around himself in all directions, pick a planet and decide to whether spy ot attack the picked planet.
4. User John presses the “x” icon and goes back to the game screen.

3.4.2.Use Case Model

In the following section, we will introduce use-case models of our system. A use case is a methodology used in system analysis to identify, clarify, and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal [12]. This is important for showing our user-system interactions.

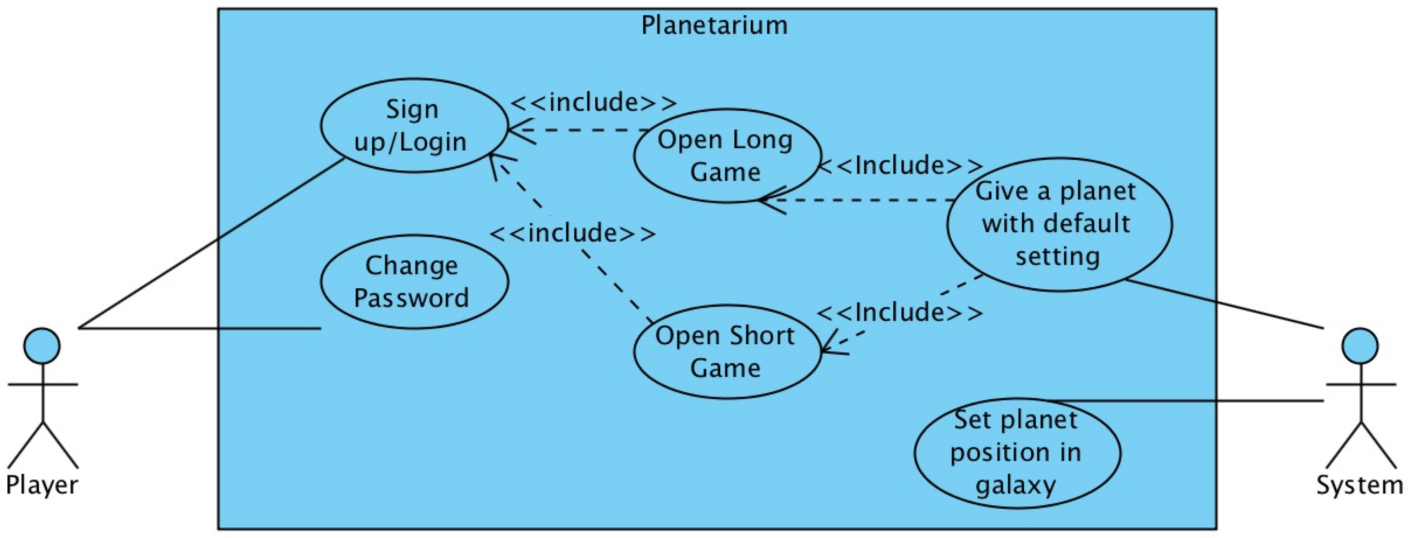


Figure 1 - Use Case Diagram

Figure 1 shows the functionalities of the game that involves signing up, logging in and initialization of the game.

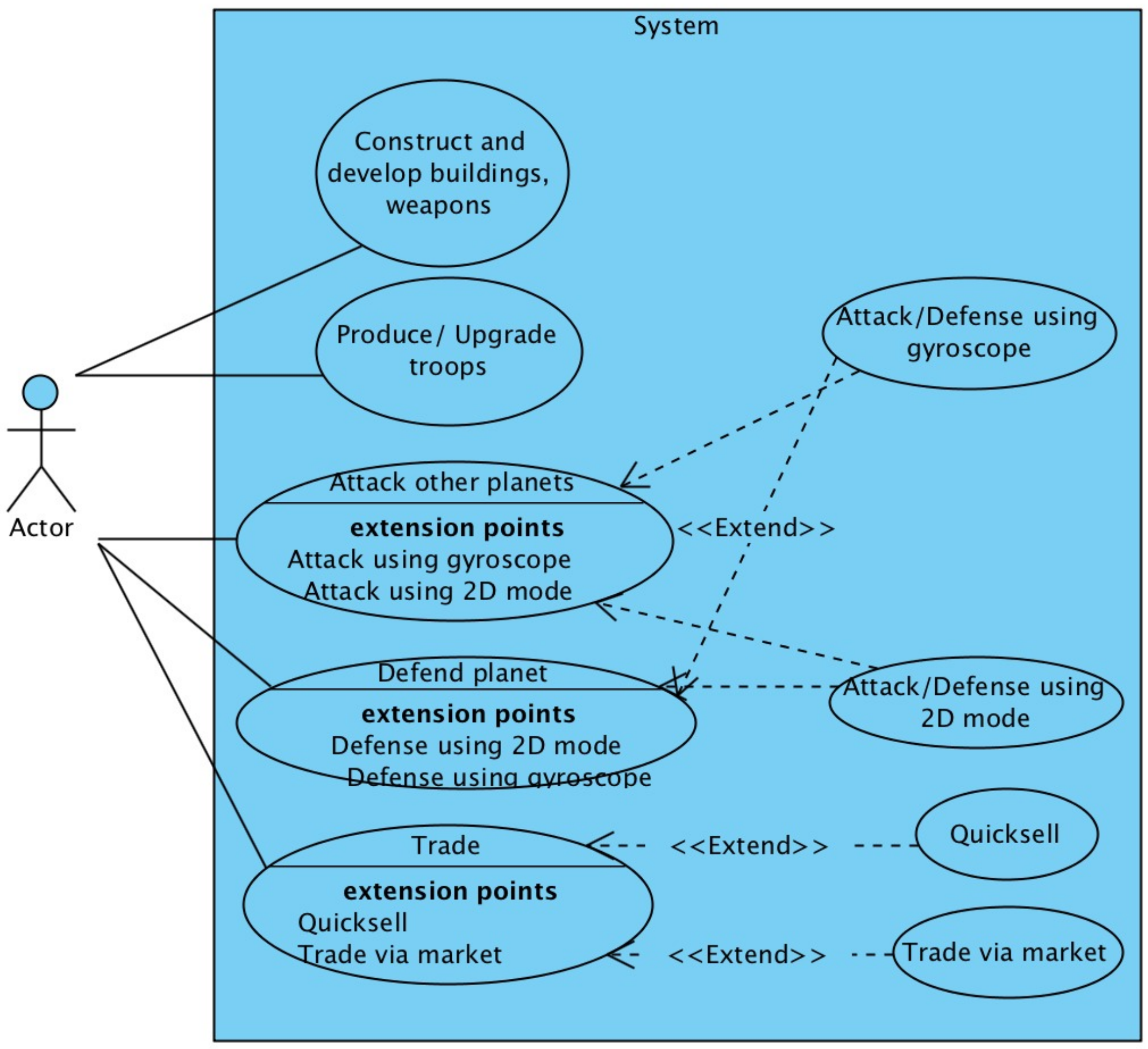


Figure 2 - Use Case Diagram

Figure 2 shows the gameplay functionalities.

3.4.3. Object and Class Model

For demonstrating our initial design for program architecture, we will introduce object class diagram in this section. The purpose of class diagram is to model the static view of an game. Class diagrams can be directly mapped with object-oriented languages and thus widely used at the time of construction. Object diagram(Figure 2) is in next page.

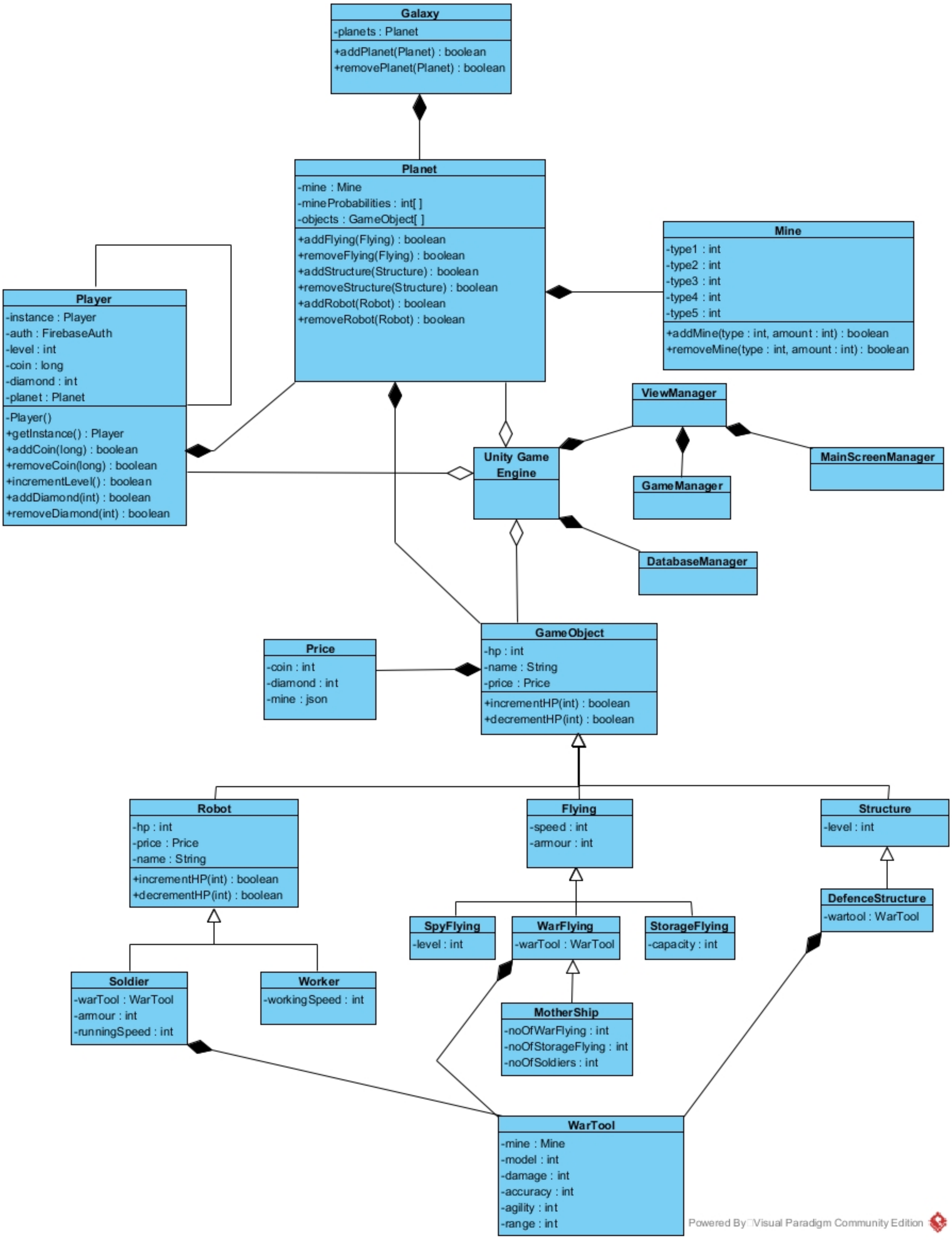

Galaxy
-planets : Planet
+addPlanet(Planet) : boolean
+removePlanet(Planet) : boolean
Planet
-mine : Mine
-mineProbabilities : int[ ]
-objects : GameObject[ ]
+addFlying(Flying) : boolean
+removeFlying(Flying) : boolean
+addStructure(Structure) : boolean
+removeStructure(Structure) : boolean
+addRobot(Robot) : boolean
+removeRobot(Robot) : boolean
Mine
-type1 : int
-type2 : int
-type3 : int
-type4 : int
-type5 : int
+addMine(type : int, amount : int) : boolean
+removeMine(type : int, amount : int) : boolean
Player
-instance : Player
-auth : FirebaseAuth
-level : int
-coin : long
-diamond : int
-planet : Planet
-Player()
+getInstance() : Player
+addCoin(long) : boolean
+removeCoin(long) : boolean
+incrementLevel() : boolean
+addDiamond(int) : boolean
+removeDiamond(int) : boolean
ViewManager
Unity Game Engine
GameManager
MainScreenManager
DatabaseManager
Price
-coin : int
-diamond : int
-mine : json
GameObject
-hp : int
-name : String
-price : Price
+incrementHP(int) : boolean
+decrementHP(int) : boolean
Robot
-hp : int
-price : Price
-name : String
+incrementHP(int) : boolean
+decrementHP(int) : boolean
Flying
-speed : int
-armour : int
Structure
-level : int
DefenceStructure
-wartool : WarTool
SpyFlying
-level : int
WarFlying
-warTool : WarTool
StorageFlying
-capacity : int
MotherShip
-noOfWarFlying : int
-noOfStorageFlying : int
-noOfSoldiers : int
Soldier
-warTool : WarTool
-armour : int
-runningSpeed : int
Worker
-workingSpeed : int
WarTool
-mine : Mine
-model : int
-damage : int
-accuracy : int
-agility : int
-range : int

Figure 3 - Object and Class Diagram

Player: Player class is a singleton class and has the information about the player with the given account. Player information is taken from firebase realtime database after authentication. This class holds the instance of the planet class which is associated with the player. It also holds coin and diamond amount player has and which level player is.

- To increase the level of the player, incrementLevel() method is invoked. It increments the player level by one and it returns a boolean value.
- To increase the coin of the player, addCoin(long) method is called. It takes a long value as a parameter, which is the amount of coin that will be given to the player, and returns a boolean value.
- To decrease the coin amount of the player, removeCoin(long) method is invoked. It takes a long value as a parameter, which is the amount of coin that will be taken from the player, and returns a boolean value to notify if the coin could be taken or not. If the player has less coin than the given long value, player's coin amount is not decreased and false value is returned.
- To increase and decrease the diamond amount of the player, addDiamond(int) and decreaseDiamond(int) are used. These methods have the same structure with addCoin(long) and removeCoin(long) methods except they have int values as parameters.

Planet: Planet class holds and administrates the quantity and the types of mines and game objects player has. It also has an integer array, which defines the probabilities that the player can mine a particular mine. This class has

- addFlying(Flying)
- addStructure(Structure)
- addRobot(Robot) methods to add new game objects to the particular planet

- removeFlying(Flying)
- removeStructure(Structure)
- removeRobot(Robot) methods to remove the existing game objects from the planet.

Also, planet class has an instance of Mine class, which keeps and manipulates the amount of mine player has.

Game Object: GameObject class is the parent of all classes which are the classes of objects player can create, upgrade, buy or sell. It holds health point, name and price values. The children classes has some other attributes exclusively for them according to their type and use.

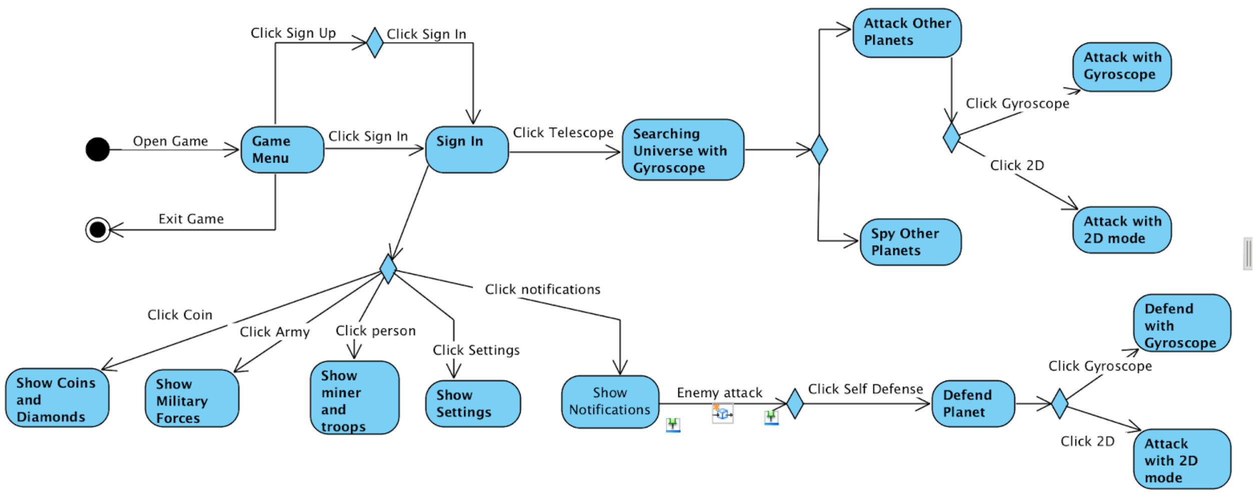
Wartool: Wartool class holds the necessary information such as damage,accuracy and agility of the fighting objects whose classes are Soldier, WarFlying and DefenceStructure.

UnityGameEngine: Unity game engine provides event handling and 3D rendering.

3.4.4.Dynamic Models

3.4.4.1.Activity Diagrams

Activity diagram is another important diagram in UML to describe the dynamic aspects of the system that represents the flow from one activity to another activity. We included following two examples of activity sequences for our system.

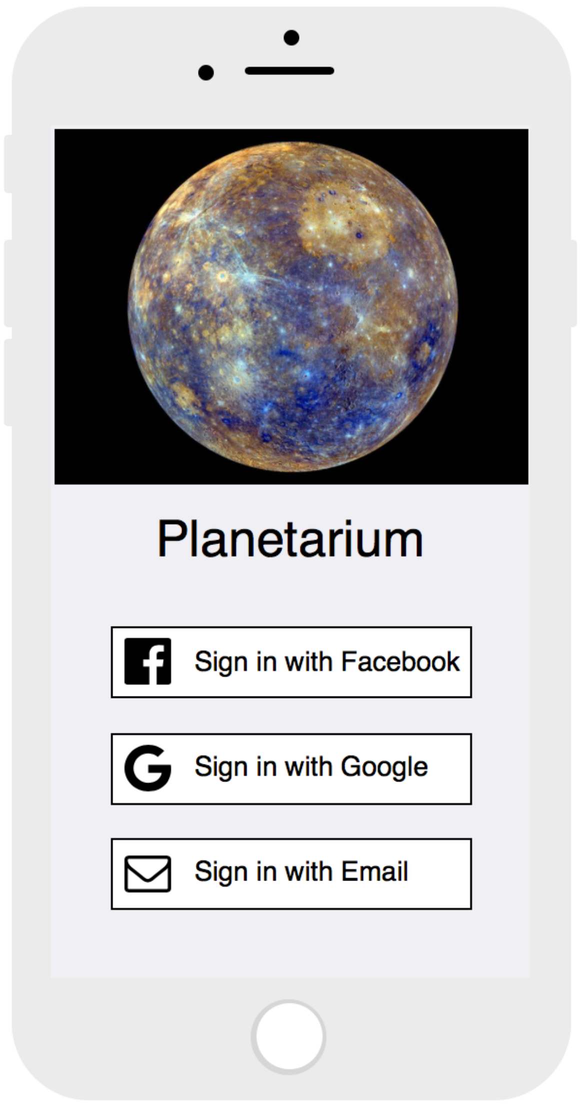


3.4.5. User Interface

In this part of the report, the user-interface of the game will be presented. Each sketch is provided with an explanation to better specify how it address the requirements mentioned above.

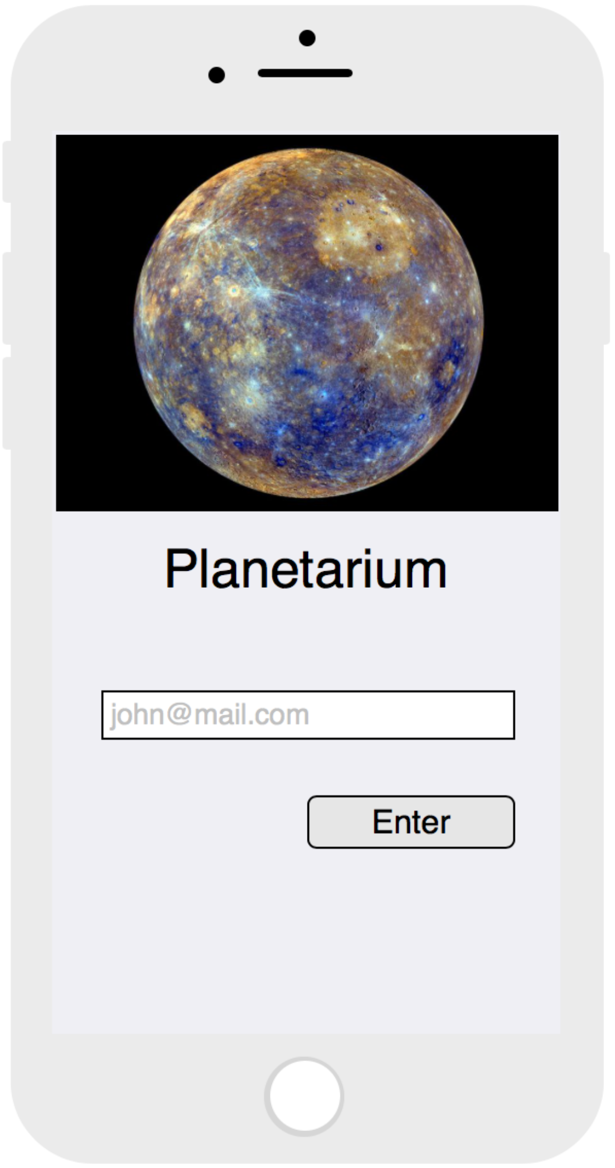
3.4.5.1. Authentication screen

The user will log in to LodeStar from the Login page as showcased below. The user will have three different options. The user will also have an option to sign in using their social media accounts.



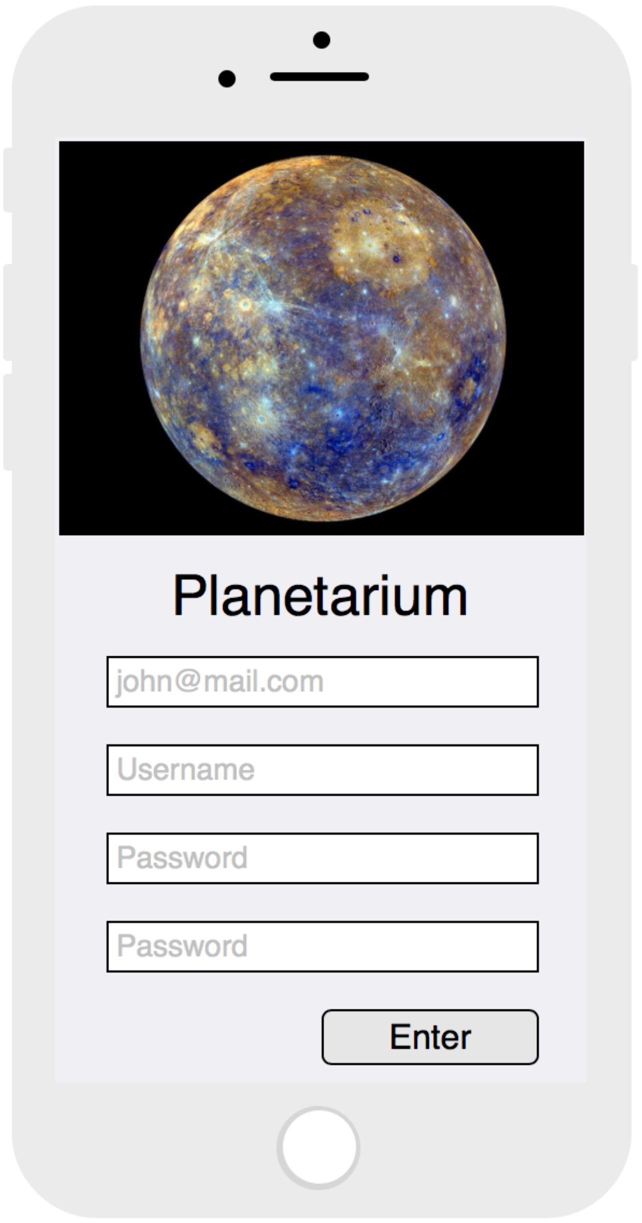
3.4.5.2. Email entry screen

After pressing “Sign in with Email” button, the user enters his email adress. If the email address already has an account associated with, user will be directed to Login screen, if not user will be directed to Signup screen.



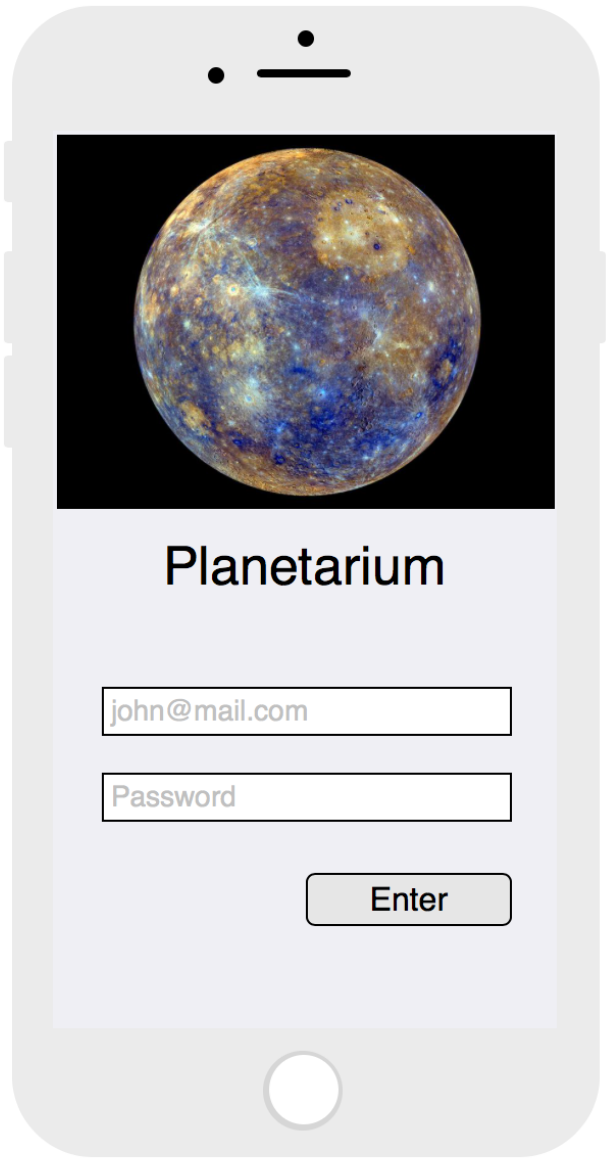
3.4.5.3.Sign up screen

If the user does not wish to sign up using their social media accounts and wants to sign up using their personal email address, then the he will be navigated to the following page.



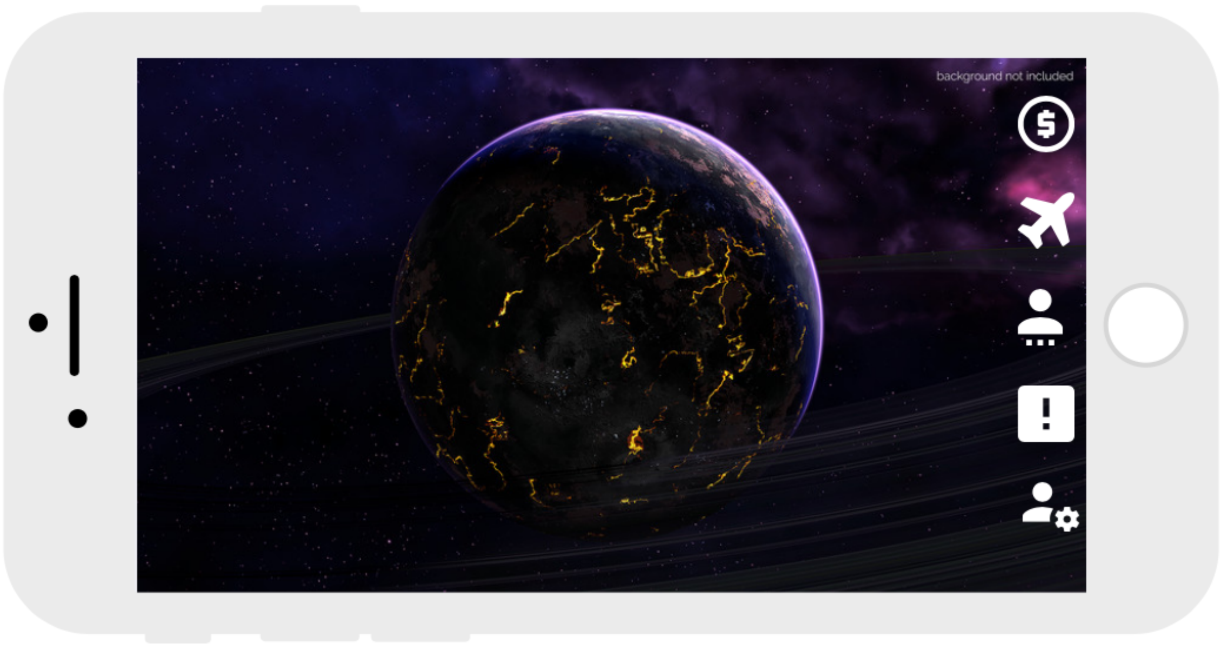
3.4.5.4. Login screen

The email address of the user is taken from email entry screen. User enters his password to enter the app.



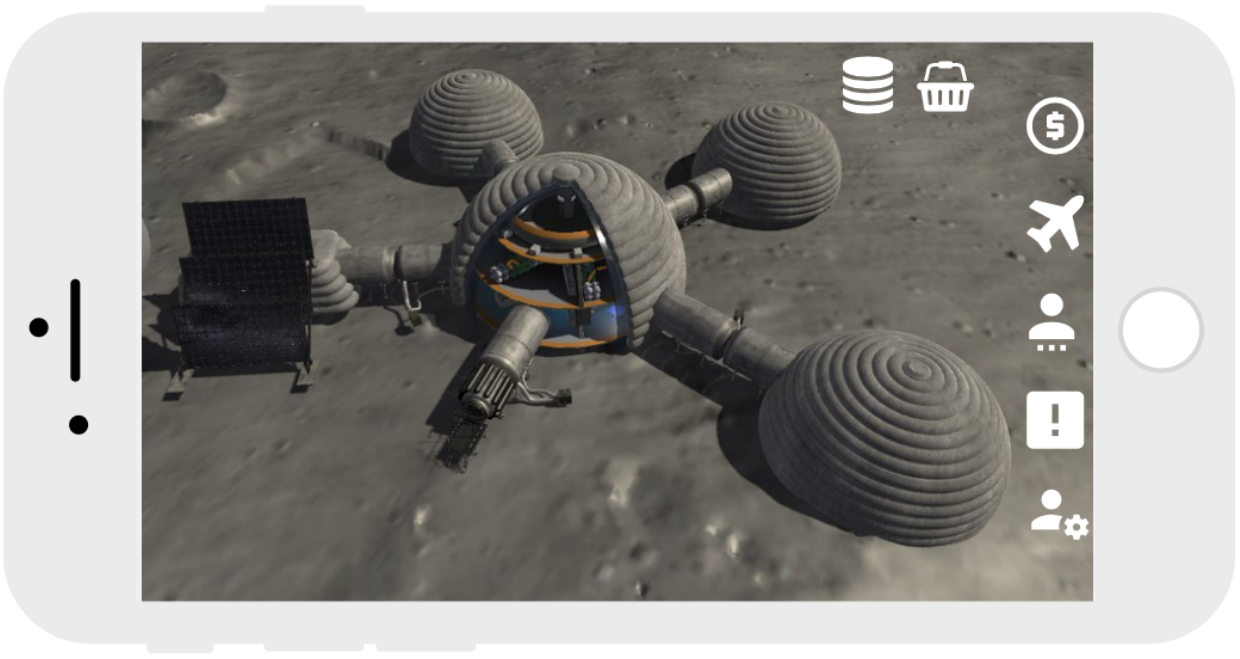
3.4.5.5. Galaxy View screen

When the app is opened, user sees his planet from the space. By pressing on planet the user moves to Base view scene.



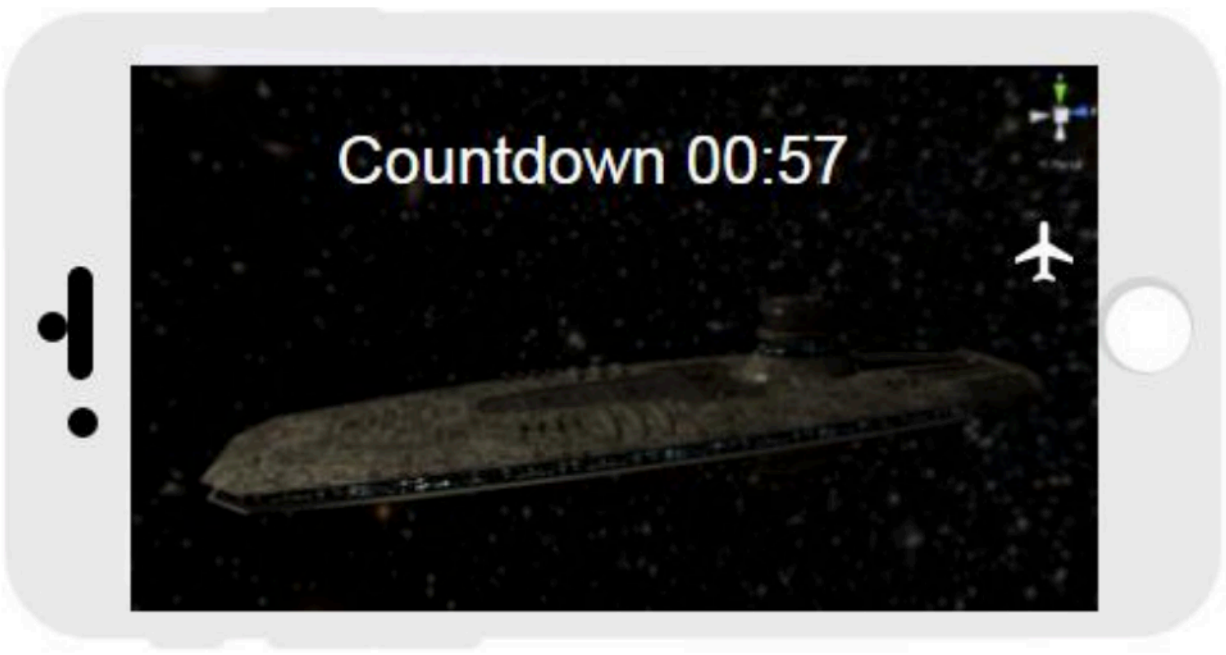
3.4.5.6. Base View screen

The user sees his base area which has structures, equipments, robots, soldiers and flying objects. He can manage his planet in that view and uses the features of app in this view.



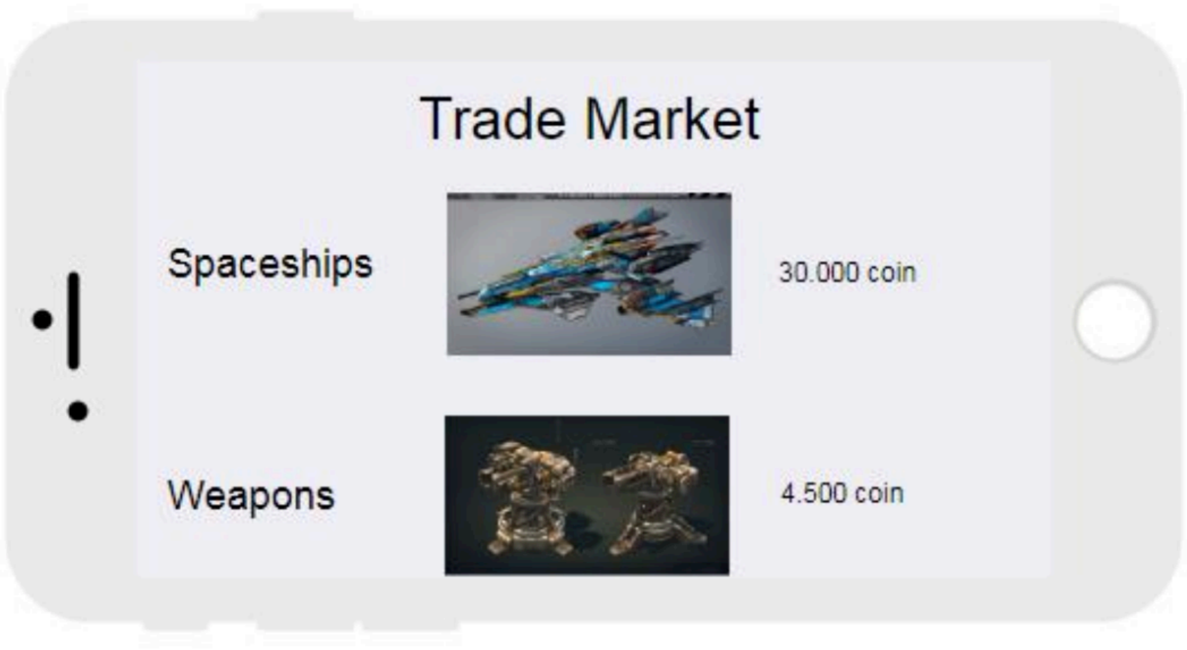
3.4.5.7.Attack Page

When the user decides to attack to a planet, 1 minute countdown starts. Defender will be notified and game server will be prepared for a battle. view the pictures of the aircraft.



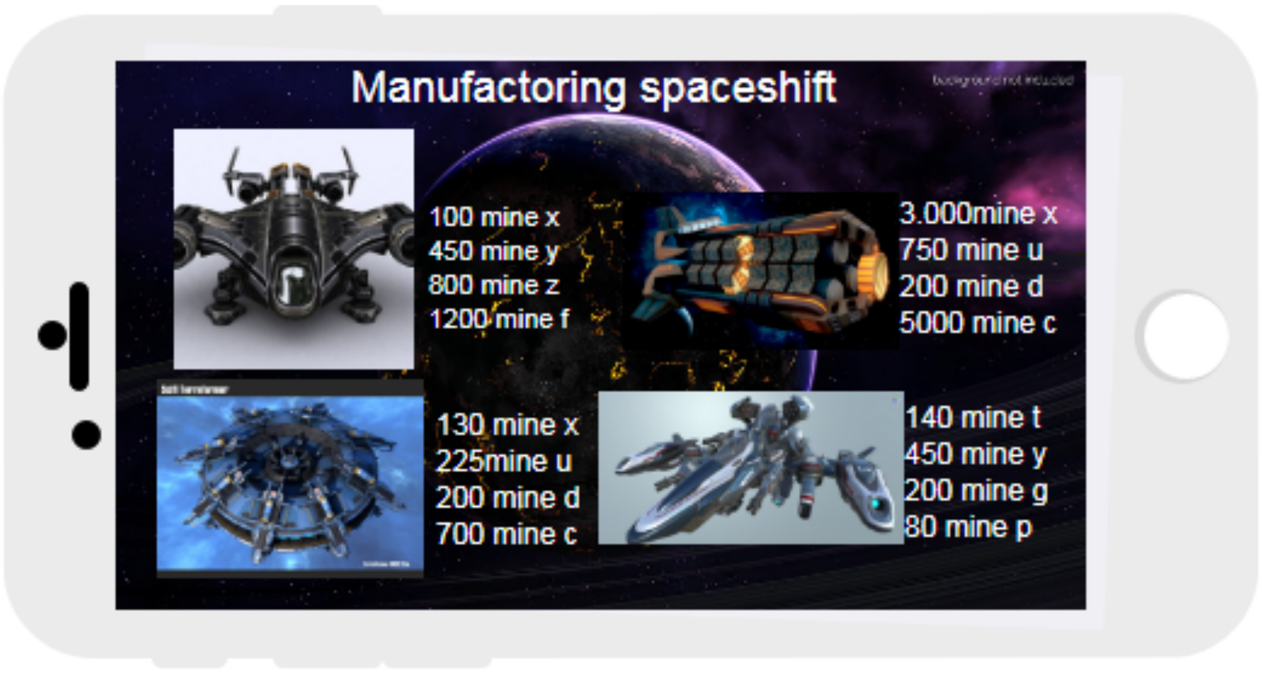
3.4.5.8. Trade Market page

User can buy new items for their planet from this screen.



3.4.5.9. Manufacturing Spaceshift page

User can produce spaceshifts by spending specified amount of different mines. User can choose any of the spaceshift by clicking the picture.


Manufactoring spaceshift

background not included

100 mine x
450 mine y
800 mine z
1200 mine f

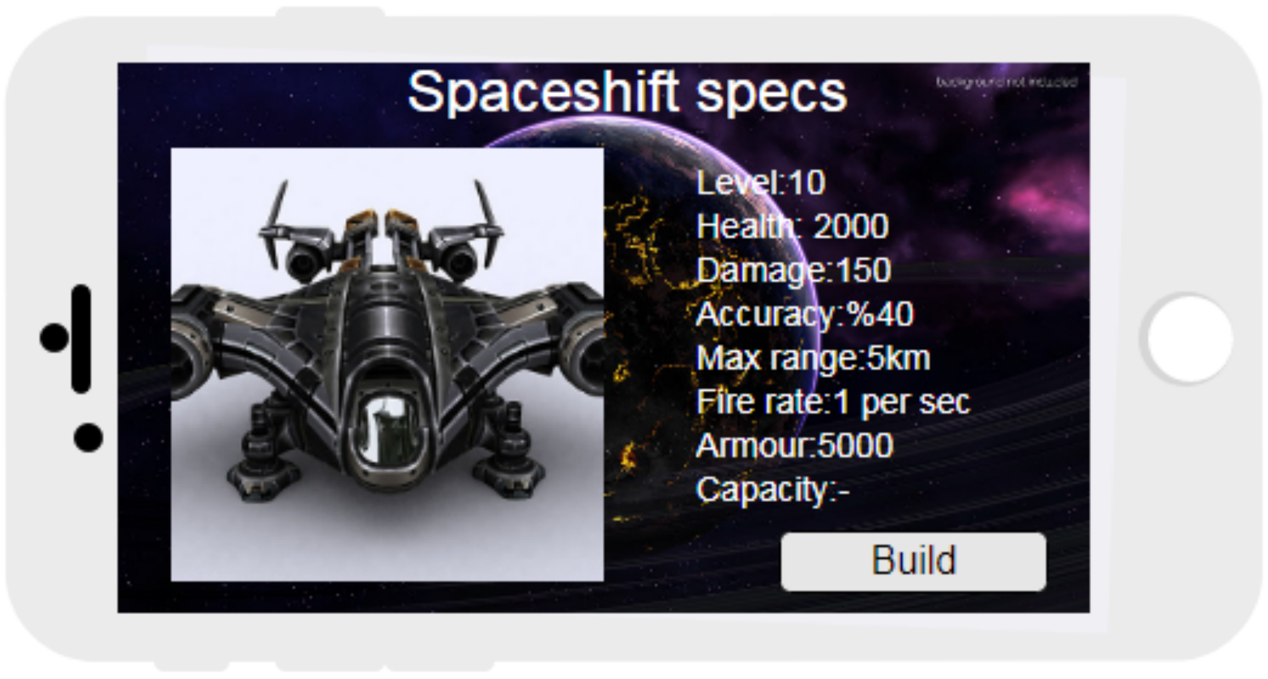
3.000mine x
750 mine u
200 mine d
5000 mine c

130 mine x
225mine u
200 mine d
700 mine c

140 mine t
450 mine y
200 mine g
80 mine p

3.4.5.10. Spaceshift Specs Page

User sees information about spaceshifts such as its level, health, damage etc. User can build spaceshift by clicking on “Build” button.



Level:10
Health: 2000
Damage:150
Accuracy:%40
Max range:5km
Fire rate:1 per sec
Armour:5000
Capacity:-

4. References

[1] https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/