Bilkent University

Department Of Computer Engineering

`

Senior Design Project

Project short-name: AugCards

Analysis Report

Yusuf Avcı, Burak Mutlu, Çerağ Oğuztüzün, Yiğit Görgülü, Bora Kurucu

Supervisor: Prof. Dr. Uğur Güdükbay
Jury Members: Asst. Prof. Dr. Can Alkan, Assoc. Prof. Dr. Çiğdem Gündüz Demir
Innovation Expert: Prof. Dr. Veysi İşler

Analysis Report
October 12, 2020

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 CS491/2.

Contents

Analysis Report

Project short-name: AugCards

1 Introduction

Mobile games are becoming more popular day by day [1]. With mobile gaming, a new era in the gaming sector has emerged. People started to lose their habit and passion of playing games with physical equipment, such as board games and card games. The emergence of mobile gaming, due to its appealing graphics, ability to play online, and the sky-high imagination of the mobile game developers, lead people to quit playing card games physically. Additionally, card games have limited assets, rigid visuals, and static rules.

The main philosophy of AugCards is reuniting the tradition and old-school fun of playing card games with your friends sitting around the table, the dynamism of mobile games. AugCard gives users the freedom to create their own cards with their own assets, introduce their own animations, and specify their own game rules.

Imagine you and your friends sitting around a table and want to have a good time. AugCard helps to limit social isolation caused by individual gaming, and bring the people together to create a game. You and your friends would first open the AugCards Desktop application and create the game cards, the event triggers, game animations, rules. Even the complex rules can simply be introduced with the help of user-friendly design which makes use of flowcharts and such structures. After the game is complete, you and your friends can open the complementary AugCards mobile application and everyone can tune in to play the game you just created, a network is established among the table and multiplayer mode is enabled. The cards and the AR versions of the assets on the cards accompanied by animations are seen on the table by everyone looking through the cam of AugCards. If you are proud of the game you have created, you can share it on the AugCards platform for other users to play, and play a game made by another user.

In this report, we will provide an analysis of the system. First, differences and weak points of the existing systems will be discussed. Then, the proposed system will be described, given attention to its featuristic details. Requirements being functional, non-functional and pseudo requirements will be provided. Also, models of the systems will be provided, such as class diagrams, and dynamic models. After, project screen mock-ups and app navigations will be included. Last but not least, a discussion on the social aspect of the project will be stated as a conclusion.

2 Current System

Although there are many web-based card game making platforms, there isn't any mobile integrated one that provides AR supported visuals and animations. Dulst is a free platform that provides users to create their own cards and play with them on desktop [2]. Dulst also allows the users to share the games they have created and allows users to play with other users and review their games. We aim our project's desktop side to be an extended version of what Dulst has implemented. We aim to extend what Dulst, a platform with prominent success in this domain, achieved by adding mobile game and AR features to conform to the card game tradition of playing it with a group of friends sitting around a table.

3 Proposed System

3.1 Overview

AugCards is a card game development tool where users can create their own cards and build custom games to be played on Android devices with multiplayer capabilities and AR supported visuals. The tool is innovative in the sense that it transforms the user-designed game logic and cards into AR multiplayer mobile games that can be played instantaneously by a group of friends around a table. Currently, there are engines that are aimed at making card games, however, there is no engine that supports mobile device integration and makes use of AR supported visual

experience. A platform that brings together developers and players on this new era AR gaming is another innovative feature.

AugCards will also contain a cloud service where users can share the games they created for other people to see and play. AugCards eliminates the need to buy physical equipment for any card game and gives the user the opportunity to create and customize their own cards. Thanks to the AR system, the experience of playing a game will feel very similar to playing a game with a group of friends around a table, as the cards and effects will all be animated and displayed to the user. Furthermore, by using systems similar to flowcharts, we intend to make expressing complex rules of the game less of a challenge for creators. An example of complex rules is the intertwined mechanics of Legends of Runeterra [3]. Our tool will overcome designing these complex rules. We will provide a custom diagram model which will help users design their game rules. An example for our diagram model can be seen in Figure 3.

The project will consist of two applications: one desktop application and one Android application. The game making part, where the user designs their own assets, cards, events, triggers, set of game rules, and card properties, will be provided via the desktop application. After the game is made and ready to be played, each user will use their mobile phones and use the complementary AugCards mobile app to see the AR visuals and cards on the table and play the game.

3.2 Functional Requirements

3.2.1 User Functionality

- Users can be either game-developers or players.
- Users can search and look out for shared games on the platform.
- Users can add the games to their library by downloading binaries.
- Users can update the games on their library if creators release a new patch.
- Users can rate and comment on the games.
- Users can see the statistics of a game about downloads and ratings or read comments.
- Users should have an account to interact with the platform features.

3.2.1.1 Player Functionality

- Players can play downloaded games.
- Players can create and connect to private hosts for game sessions in the local network.
- Players should have the required number of human-players/mobile-devices connected to the host regarding the player constraints of the game.
- Players should set up the camera of the mobile device regarding that game content will be rendered on a planar surface.
- Players can send inputs to the game by touching the screen.
- Players can interact with game instances/cards and perform certain game events.
- Players can see other players' interaction and ongoing game events simultaneously.

3.2.1.2 Game-Developer Functionality

- Developers do not need programming knowledge.
- Developers can create custom card games.

By using UI options, developers should be able to define:

- required/custom game instances like the player, card, card set, card types, etc.,
- properties of game instances like player's health, card's properties in-depth (type, initial value, constant).
- class relationships between defined game instances.
- game events and their effects on game instances.
- trigger mechanism of game events.
- trigger mechanism of user-input events.
- rules for the events and the properties of game instances.
- heuristic function with accessible game data to enable AI scription.
- custom graphical models/assets for corresponding game instances.
- custom animations for each graphical model.
- bindings of each animation to certain events.

- Developers can upload/update the models for their own created games on the cloud.
- Developers can see all commits and pull an old version of the game.
- Developers can generate source code and compile the game on the cloud for Debugging or Release.
- Developers can generate binaries directly on the local machine for Debugging.
- Developers cannot access the generated source code for the game.
- Developers can download compiled binaries from the cloud.
- Developers can sync binaries of the game on desktop applications into a mobile application.
- Developers can debug the game with the desktop application over a virtual plane for AR-enabled graphics.
- Developers can debug the game with the mobile application over AR applied camera frames.
- Developers can deactivate certain game rules for easier debugging.
- Developers can share their custom game to make it accessible for users in the cloud platform.
- Developers can update/unshare their shared games.
- Developers can remove the repository of an unshared game from the cloud.

3.2.2 Desktop System Functionality

The system should:

- ask the developer for login credentials.
- create local/cloud repositories for custom game projects.
- delete all repository files if the developer decides to remove the game.
- restore the last committed version of a custom game from their cloud/local repository.
- restore a game over design models like game instances, instance relationships, events-triggers, etc.
- provide a sophisticated and interactive UI for each model design feature.

- provide a table design tool for game instances and their relationships such as inserting a new table to define new instance type, inserting a new line into instance tables to define properties, columns for the detail of properties like type, initial value, etc.
- provide a sequence diagram tool for events-triggers; definition of triggers contains determining check rules, updates for accessible properties, and invoking other events.
- provide a model viewer to import and bind graphical models/animations to game instances and events.
- give warnings about possible development issues like incomplete event-triggers, infinite-loop danger, etc.
- provide a commit tool to investigate commit details, push/pull commits, update game models over commits.
- provide a debugging tool to generate binaries on local/cloud, run the binaries on Android Emulator, or sync the binaries with the mobile application.
- provide an additional debugging feature to simulate the game with a desktop version to ensure quick debugging sessions.
- provide a platform interface to edit the community page for the game, share/unshare the game and interact with other users' comments.

3.2.3 Mobile System Functionality

The system should:

- asks the user for login credentials.
- provide a platform interface in which users can look out for custom games shared by developers.
- provide a community page view for each shared game, in which users can see and interact with stats/ratings of the game and comments by others.
- provide a download/remove button integrated within the community page to add the game to the library.
- provide a library to view the games installed on the user devices from both source Cloud and Desktop-sync.
- remove the games unshared by developers from the library.

- provide a game-sessions view to create/join hosts in a local network.
- provide a game-lobby view to see other players/devices connected to the host for a game session and launch the game session.
- provide an interactive in-game view to play the game; see the game content rendered on camera frames and send input to the game by touching the screen.
- include a common subsystem for all custom games, which performs the graphical/network operations on games.

3.2.3.1 Common Graphics/Network Subsystem

The system should:

- be a dynamic load library to ensure that downloadable/storable game binaries do not include it.
- provide a network functionality that any content update on a device should simultaneously appear on other devices connected to the same network.
- provide a graphics functionality that detects the planar surface from camera frames and render a custom content anchored to the surface on camera frames.
- provide motion detection to render the content with an accurate camera angle on each frame.
- support advanced custom contents like 3D/2D model/animations and UI elements.
- support advanced graphics effects like light-estimation, shadows, anti-aliasing, etc.

3.2.4 Cloud System Functionality

The system should:

- create private repositories for the game projects.
- store the game models along with commit logs in a private repository.
- generate source code and compile binaries for Debugging/Release.
- not store the debugging/release binaries.
- create public repositories to share games.

- store the last shared version (binaries) of the game in a public repository.
- store contents for the community page of the game in a public repository.
- provide access for developers to their private repositories.
- provide access for users to public repositories.
- provide file transfer feature to upload/download project commits, generated binaries, and shared games.

3.3 Nonfunctional Requirements

3.3.1 Usability

- Game creation should not require programming knowledge.
- Tools should be self-explanatory, shouldn't require extensive tutorials or guides to be understood.
- The expression of complex card game rules will be simplified using flowcharts.
- AR-based graphics should have a refresh rate of at least 25 Hz to not affect game experience adversely.
- Popular image formats such as PNG and JPEG should be supported as assets.
- Popular graphical model/animation formats like OBJ and COLLADA should be supported.

3.3.2 Reliability

- Should ensure that changes in game models are not lost on connection errors.
- Should have a back-up mechanism for ongoing games in a network failure situation.
- Contradictory game rules shouldn't be allowed to cause errors.
- Cheats should be detected via checksums.

3.3.3 Maintainability

- Should be modular to reduce the complexity of the codebase.
- Network maintenance costs should be lower than 50TL per month while profits are low.
- Should use design patterns that will allow changing used libraries.

3.3.4 Accessibility

- Should be free to download.
- Should have integration with Google Services.
- Should require less than 1GB of RAM.

3.3.5 Extendability

- The addition of new possible game mechanics should not require changing existing code.
- The code itself should be properly structured,using design patterns and clever modularity.Adding new features and mechanics should require minimal or zero amount of change in the code.

3.3.6 Portability

- Should not cause any compatibility error with changing Android device sizes and camera resolutions.
- The game creation tool should run on Windows and Linux.

3.4 Pseudo Requirements

- Since the mobile application will be developed for the Android platform, the implementation language should be Java or Kotlin. For our project, we choose to use Kotlin because it is a modern language and is widely being used for Android development. We will use Android Studio for development.
- We will use Vuforia, which is a library to develop Augmented Reality applications for mobile devices. Vuforia supports Java, and Kotlin runs on the JVM which allows it to interact with Java libraries.
- To make integration with the mobile application easier and less problematic, the desktop application should also be written in Kotlin. For development, we will use IntelliJ IDEA.
- Since the projects will be hosted on a cloud service, we will need to interact with the chosen service's interface to use their cloud service. We will use Firebase because it is free and developed by Google, which will help with integration with the Android application.
- We will license our project with the MIT License [4] since it provides the most flexibility and will be less likely to lead to legal problems we may come across in the future.

3.5 System Models

This section includes the models that are going to represent our system.

3.5.1 Scenarios

In this section, some usage scenarios of our desktop and mobile applications will be described in detail.

Scenario 1

Use case name	Register
Participating actors	Player
Flow of events	1. The player enters their credentials (email and password). 2. The player clicks the register button. 3. The player registers into our system.
Entry condition	The player is not logged in.
Exit condition	Registration is successful. This is possible only if: ● The player does not have an account. ● The provided email address is valid.
Quality requirements	-

Scenario 2

Use case name	Register
Participating actors	Developer
Flow of events	1. The developer enters their credentials (email and password). 2. The developer clicks the register button. 3. The developer registers into our system.
Entry condition	The developer is not logged in.
Exit condition	Registration is successful. This is possible only if: ● The developer does not have an account. ● The provided email address is valid.
Quality requirements	-

Scenario 3

Use case name	Login
Participating actors	Player
Flow of events	1. The player enters their credentials (email and password). 2. The player clicks the login button. 3. The player is logged into our system.
Entry condition	● The player is not logged in. ● The player has an account.
Exit condition	Login is successful. This is possible only if: ● The player has an account. ● The provided credentials are valid.
Quality requirements	The player should stay logged into the system in

	the mobile application.

Scenario 4

Use case name	Login
Participating actors	Developer
Flow of events	1. The developer enters their credentials (email and password). 2. The developer clicks the login button. 3. The developer is logged into our system.
Entry condition	• The developer is not logged in. • The developer has an account.
Exit condition	Login is successful. This is possible only if: • The developer has an account. • The provided credentials are valid.
Quality requirements	The developer should stay logged into the system in the desktop application.

Scenario 5

Use case name	CreateGame
Participating actors	Developer
Flow of events	1. The developer chooses to create a game. 2. The developer names their game. 3. The developer clicks the create button.
Entry condition	• The developer should be in the game creation screen. • The developer should not be managing any games currently.
Exit condition	The developer creates the game.

Quality requirements	-

Scenario 6

Use case name	ManageGameModels
Participating actors	Developer
Flow of events	1. The developer adds, edits or deletes game models. 2. The developer saves their modifications.
Entry condition	The developer chooses to manage their game models.
Exit condition	The developer saves their progress and exits the management screen.
Quality requirements	-

Scenario 7

Use case name	ManageCards
Participating actors	Developer
Flow of events	1. The developer adds, edits or deletes cards. 2. The developer saves their modifications.
Entry condition	The developer chooses to manage their cards.
Exit condition	The developer saves their progress and exits the management screen.
Quality requirements	-

Scenario 8

Use case name	ManageGameRules
Participating actors	Developer
Flow of events	1. The developer adds, edits or deletes game rules. 2. The developer saves their modifications.
Entry condition	The developer chooses to manage their game rules.
Exit condition	The developer saves their progress and exits the management screen.
Quality requirements	-

Scenario 9

Use case name	ManageAnimations
Participating actors	Developer
Flow of events	1. The developer adds, edits or deletes animations. 2. The developer saves their modifications.
Entry condition	The developer chooses to manage their animations.
Exit condition	The developer saves their progress and exits the management screen.
Quality requirements	-

Scenario 10

Use case name	PublishGame
Participating actors	Developer
Flow of events	1. The developer sends their game to our game cloud. 2. The server places the game into the games list.
Entry condition	The developer chooses to publish their game.
Exit condition	The game is published on our cloud.
Quality requirements	-

Scenario 11

Use case name	UpdateGame
Participating actors	Developer
Flow of events	1. The developer sends their updated game to our game cloud. 2. The server replaces the old version of the game with the new version.
Entry condition	The developer chooses to update their game.
Exit condition	The game is published on the cloud.
Quality requirements	-

Scenario 12

Use case name	DeleteGame
Participating actors	Developer
Flow of events	1. Developer chooses which game to delete. 2. The cloud performs an operation.
Entry condition	• The developer should be logged in. • The developer should have at least one game.
Exit condition	The developer confirms their operation and their game is deleted.
Quality requirements	-

Scenario 13

Use case name	ManageGameLibrary
Participating actors	Player
Flow of events	1. The player chooses to manage their game library. 2. They are directed to the management screen.
Entry condition	The player should be logged in.
Exit condition	The player is in the game library management screen.
Quality requirements	-

Scenario 14

Use case name	SearchGame
Participating actors	Player
Flow of events	1. The player chooses to search the game cloud. 2. They are directed to the search screen. 3. They search using the name of the game.
Entry condition	• The player should already be in the library management screen. • The player should choose the search option.
Exit condition	The player finds the game that they are looking for, or no matching game for the provided string.
Quality requirements	-

Scenario 15

Use case name	AddGame
Participating actors	Player
Flow of events	1. The player chooses to add a game to their game library. 2. The game they choose is downloaded and added to their library.
Entry condition	The player should choose a game to add.
Exit condition	The game is downloaded and added to the library of the player.
Quality requirements	-

Scenario 16

Use case name	UpdateGame
Participating actors	Player
Flow of events	1. The player chooses to update a game in their game library. 2. The updated version of the game is downloaded from the cloud and the old version is removed.
Entry condition	• The player should choose a game to update. • The chosen game should have an update.
Exit condition	The game is updated.
Quality requirements	-

Scenario 17

Use case name	RateGame
Participating actors	Player
Flow of events	1. The player chooses to rate a game in their game library. 2. Their rating is recorded and the total rating of the game is updated.
Entry condition	• The player chooses a game to rate. • The chosen game should be downloaded and played by the player.
Exit condition	The game's rating is updated.
Quality requirements	-

Scenario 18

Use case name	CommentGame
Participating actors	Player
Flow of events	1. The player chooses to comment on a game in their game library. 2. Their comment is recorded and displayed on the game information screen.
Entry condition	• The player chooses a game to comment on. • The chosen game should be downloaded and played by the player.
Exit condition	The game's comments are updated.
Quality requirements	-

Scenario 19

Use case name	ManageSession
Participating actors	Player
Flow of events	1. The player chooses a game. 2. The player manages their session in the game.
Entry condition	The player should choose a game.
Exit condition	They are directed to the session management screen.
Quality requirements	-

Scenario 20

Use case name	CreateSession
Participating actors	Player
Flow of events	1. The player creates a game session.
Entry condition	The player should choose to create a session.
Exit condition	The session is created and visible to nearby players.
Quality requirements	-

Scenario 21

Use case name	JoinSession
Participating actors	Player
Flow of events	1. The player chooses a game session to join. 2. The player joins that session.
Entry condition	• The player should choose to join a session. • There should be an available session.
Exit condition	The player joins into the session.
Quality requirements	-

Scenario 22

Use case name	EditSessionSettings
Participating actors	Player
Flow of events	1. The player chooses to edit their session's settings. 2. They save their settings.
Entry condition	• The player should have created a session. • They should be in the session.
Exit condition	The player saves their settings.
Quality requirements	-

Scenario 23

Use case name	PlayGame
Participating actors	Player
Flow of events	The player chooses an action to perform in the game.
Entry condition	• The player should be in a game.
Exit condition	The player confirms their action.
Quality requirements	-

Scenario 24

Use case name	CheckTable
Participating actors	Player
Flow of events	The player chooses to inspect the table.
Entry condition	• The player should be in a game.
Exit condition	The player exits the table view.
Quality requirements	-

Scenario 25

Use case name	PlayCard
Participating actors	Player
Flow of events	The player chooses a card to play.
Entry condition	• The player should be in a game. • It should be the player's turn.
Exit condition	The player plays a card.
Quality requirements	-

3.5.2 Use Case Model

We created two use case models; one for the desktop system and one for the mobile system.

3.5.2.1 Desktop System Use Case Model

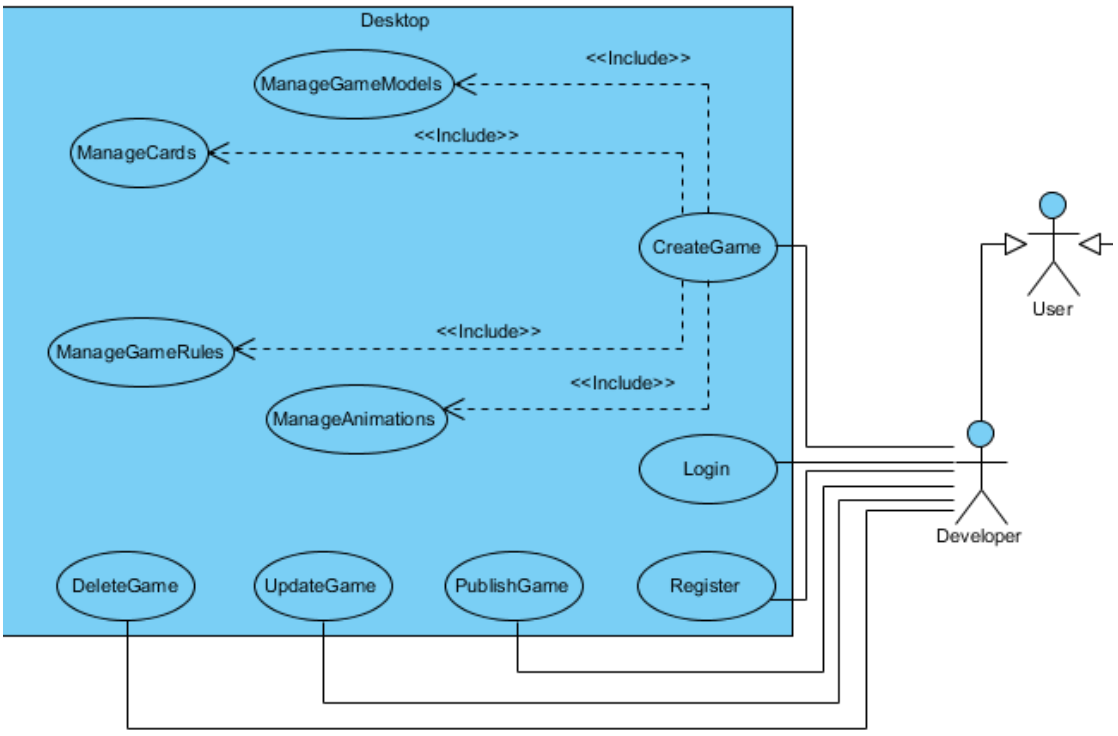

Figure 1: Desktop Application Use Case Diagram

3.5.2.2 Mobile System Use Case Model

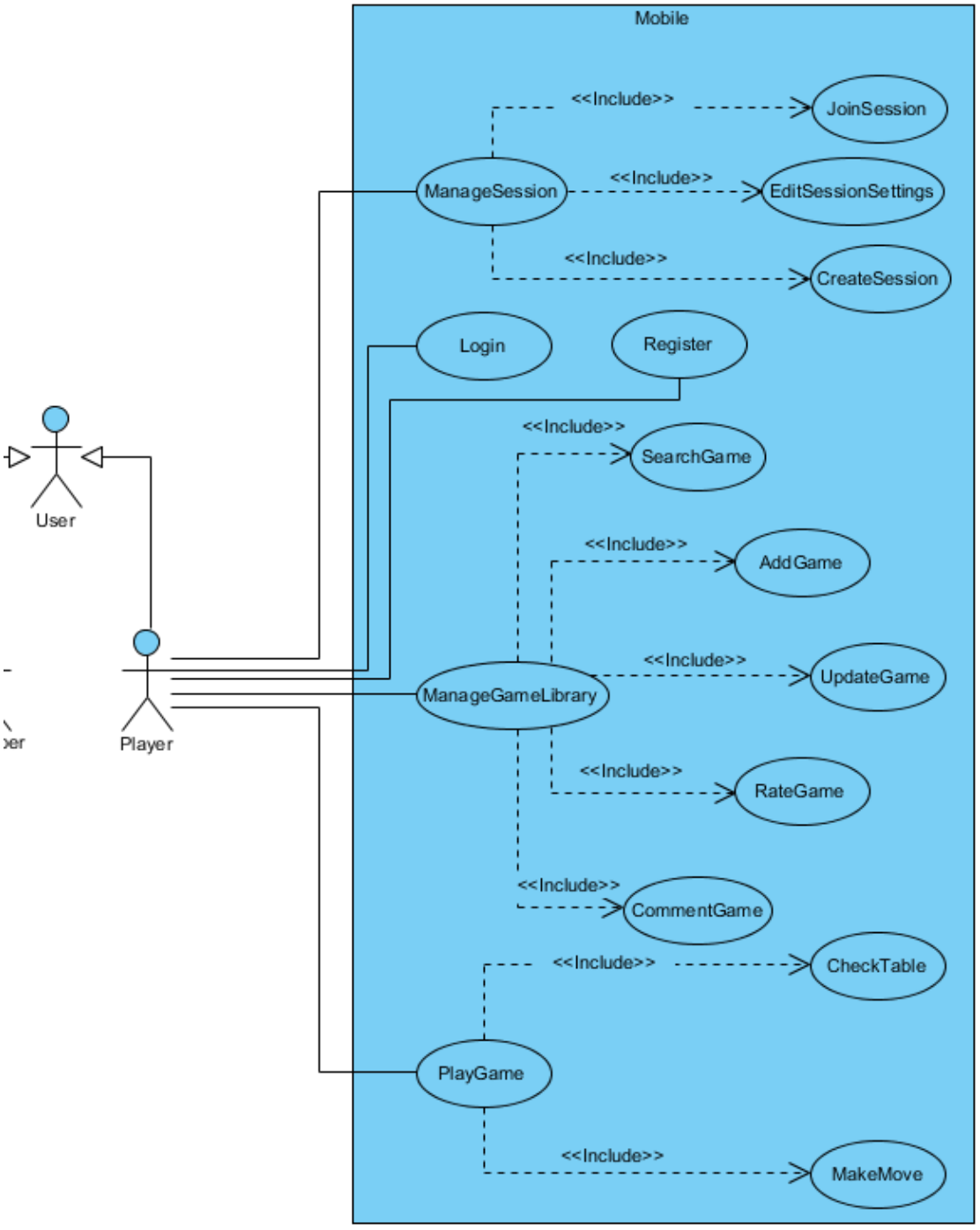

Figure 2: Mobile Application Use Case Model

3.5.3 Object and Class Model

We will include object-class diagrams to represent the static state of our systems. As mentioned, we have four systems as Desktop, Mobile, Cloud, Common Graphics/Network Subsystem. We included class diagrams for Desktop and Mobile System but not for Cloud and Graphic/Network System since they are implementation specific.

3.5.3.1 Desktop System Class Model

Desktop System stands for game-design tool, that's why we should have classes to store elements of the designed game like game-instances, events, rules, scripts, graphical models etc.

- **Project:** A class to store basics and elements of a custom game project created by the developer.
- **Component:** An abstract class to generalize all types of elements inserted into the game models.
- **Instance:** An abstract class to generalize objects in the game.
- **GameInstance:** A specific **Instance** to represent the custom game objects like card, player, avatar defined by the developer.
- **ListInstance:** A specific **Instance** to represent the list objects used in the game.
- **Attribute:** A class to represent the properties of defined game objects.
- **DataType:** An enumeration to specify the type of attributes' value.
- **InitType:** An enumeration to specify the type of attributes' initialization.
- **Event:** An abstract class to generalize events in the game.
- **GameEvent:** A specific **Event** to represent the custom game events like attack, nextTurn, playCard defined by the developer.
- **InputEvent:** A specific **Event** to represent the possible input events like cardDrag, cardClick, deckClick etc.
- **EventArgs:** A class to represent the arguments of custom events.
- **EventTrigger:** A class to represent the trigger mechanisms for defined events.

- **Conditioner:** A class to generalize in-script rule types.
- **Rule:** A specific **Conditioner** to represent one-condition two branch scripting.
- **Iteration:** A specific **Conditioner** to represent iterative scripting over one-condition.
- **Script:** A class to represent custom one-line instruction.
- **Effect:** A class to represent the modifications applied on scripts.
- **EffectType:** An enumeration to specify effects' modification type.
- **Scriptable:** An interface to generalize the elements over which a script can be applied.
- **Accessible:** An interface to generalize the elements which are obtainable in the sequential flow of scripts.
- **Access:** A class to represent accessible reference hierarchies
- **InitValue:** A class to represent basic initialization value.
- **Game:** A specific **GameInstance** to represent the state and actions during a game session.
- **Board:** A class to represent the hierarchy of the elements with respect to their state.
- **BoardField:** A class to represent hierarchical areas in which many elements can be involved.
- **InstanceView:** A class to represent the view for defined game objects.
- **ViewManager:** A class to store the view objects created in the game project.
- **Asset:** An abstract class to generalize the graphical elements.
- **GraphicalModel:** A specific **Asset** to represent the models within graphical data.
- **Animation:** A specific **Asset** to represent the pre-designed transform sequence for graphical models within animation data.
- **AssetStorage:** A class to store the graphical assets loaded within the game project.

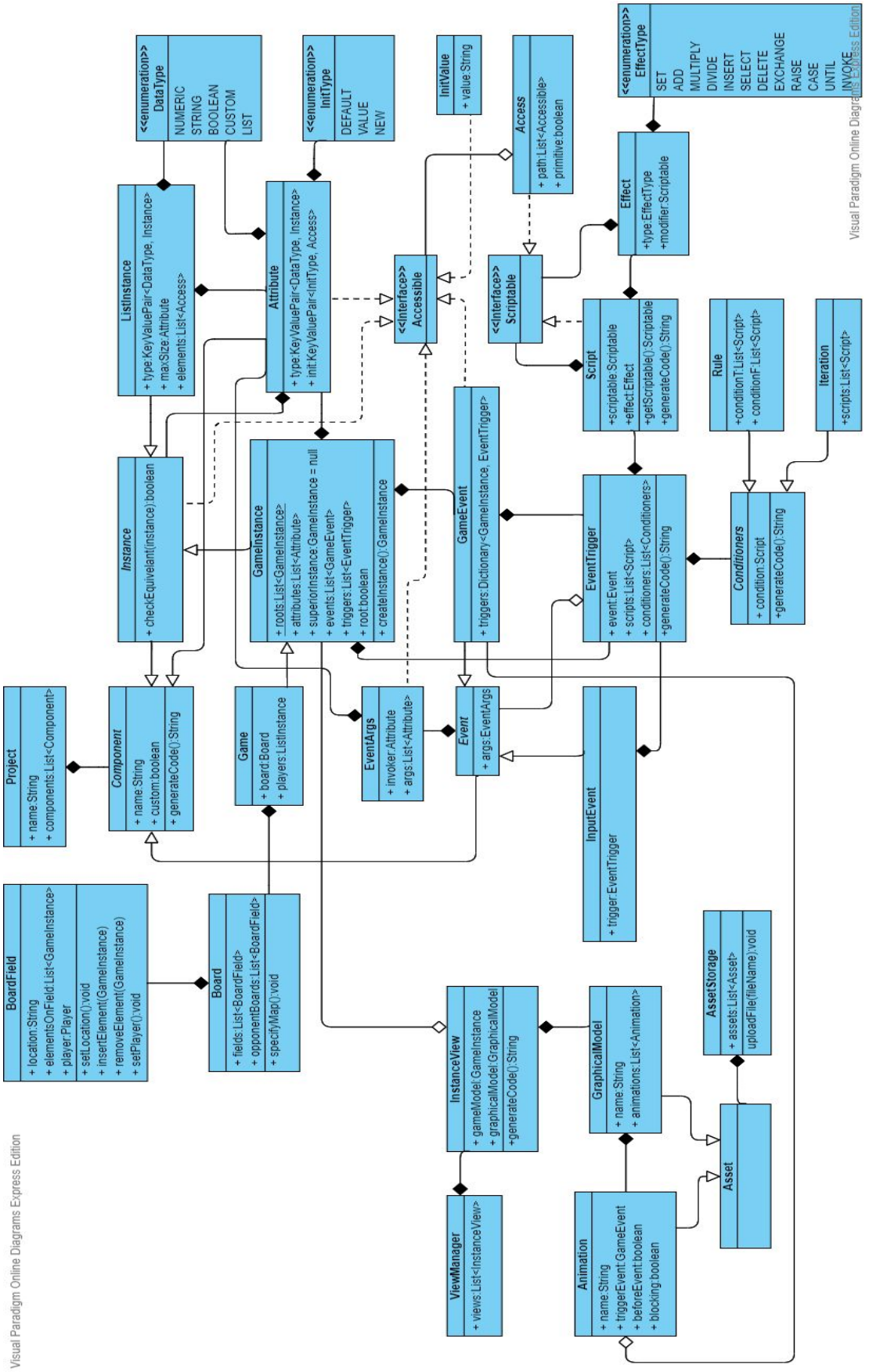

Figure 3: Desktop System Class Model

3.5.3.2 Mobile System Class Model

Mobile System stands for a platform-like application where users can look for games, like or comment on the game, inserting the game to their own library etc. That's why we should have model classes to store information about users, games, game-library, platform and game lobby etc. Also, there should be view classes because of its being a highly front-end application.

- **AugCards:** Entry-point class into the model state of the mobile system; it stores the most fundamental objects in the system.
- **User:** A class to represent only type of actor in the system by storing its unique and private attributes like username, email, etc.
- **Platform:** A class to represent the common point where users and developers meet through shared games.
- **GameLibrary:** A class to represent the customizable game storage where users can insert new ones and pick favorites.
- **Game:** A class to represent the state of a shared custom game in the platform.
- **GameLobby:** A class to represent created and in-preparation game sessions in which players can join.
- **MainFrame:** A class to store the view objects and their hierarchies among the system.
- **View:** A class to generalize all sophisticated view objects like library-view, platform-view, game-view, etc.
- **AugCardsView:** A specific **View** to represent the main view of AugCards application; it can contain inner view.
- **HomeView:** A specific **View** to represent the home-screen of the application.
- **AppStartView:** A specific **View** to represent the initial screen displayed when application launched.
- **LoginView**: A specific **View** to represent the screen where users can enter their credentials to access the system.
- **LibraryView:** A specific **View** to represent the illustration of a game-library customized by the user.

- **GameMainView:** A specific **View** to represent the alone illustration of a game along with all details like likes, comments, etc.
- **GameLayoutView:** A specific **View** to represent the highlights of a game for listing purpose in library or platform.
- **GameLobbyView:** A specific **View** to represent the room-like view in which displays the information about the game session in preparation phase.
- **PlatformView:** A specific **View** to represent the store-like display in which displays the popular or searched games.

There is an interface of Common Graphics/Network Subsystem because the mobile system is interchangeably processing with this subsystem to handle game-lobby creation/join through the local network and to run downloaded game executables with sophisticated graphics pipeline.

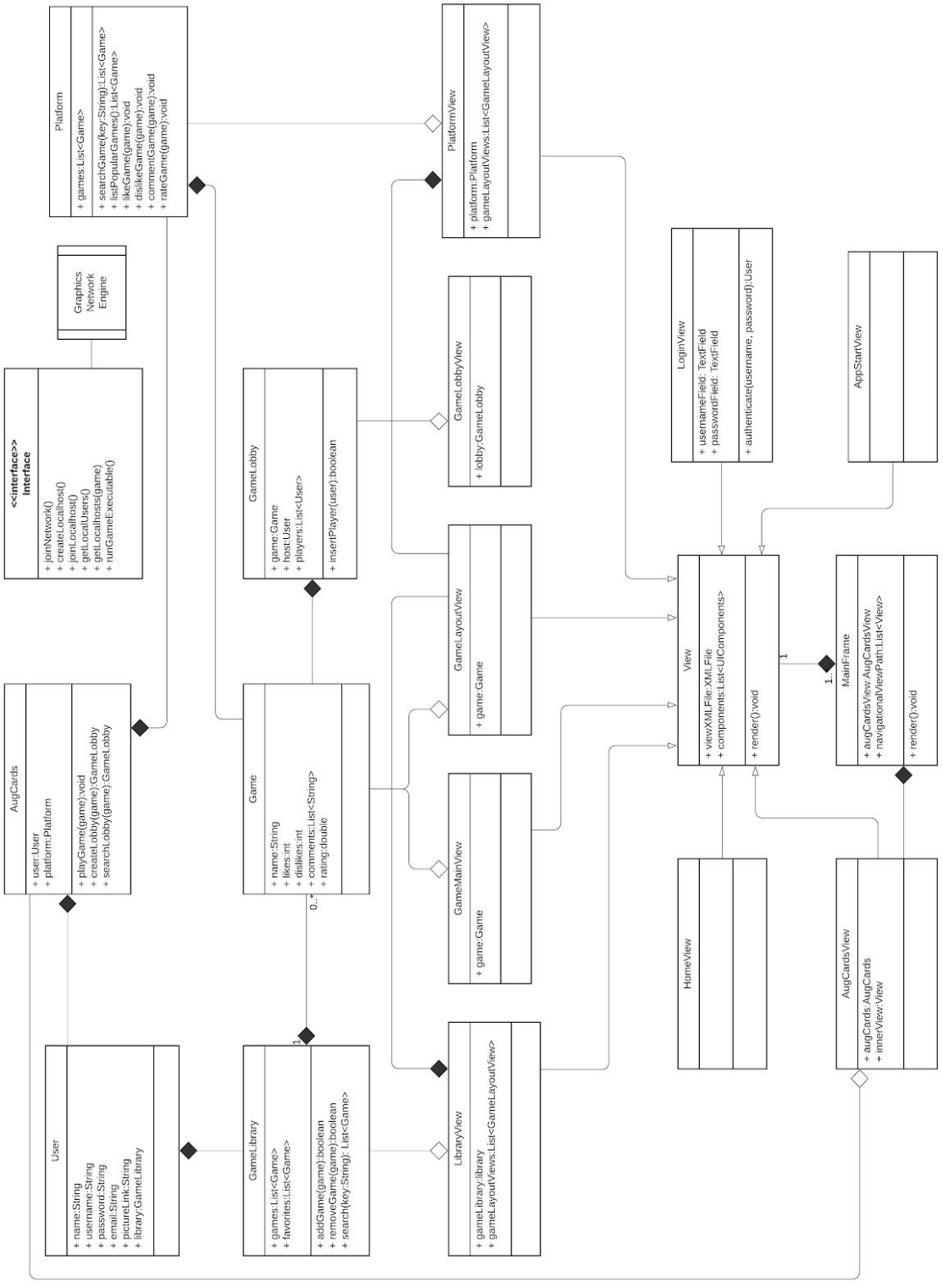

Figure 4: Mobile System Class Model

3.5.4 Dynamic Models

Dynamic modeling of a system shows the interactions between classes and users through function calls, which demonstrate the scenarios regarding the processing of the system.

3.5.4.1 Sequence Diagrams

Sequence diagrams display the interactions between the actor interacting with the program, and display the program functionalities. The following five diagrams are the sequence diagrams of some of our functionalities.

3.5.4.1.1 Login Sequence Diagram

The Login Sequence Diagram shows the login and register activity of the user to our mobile system, which utilizes a Database.

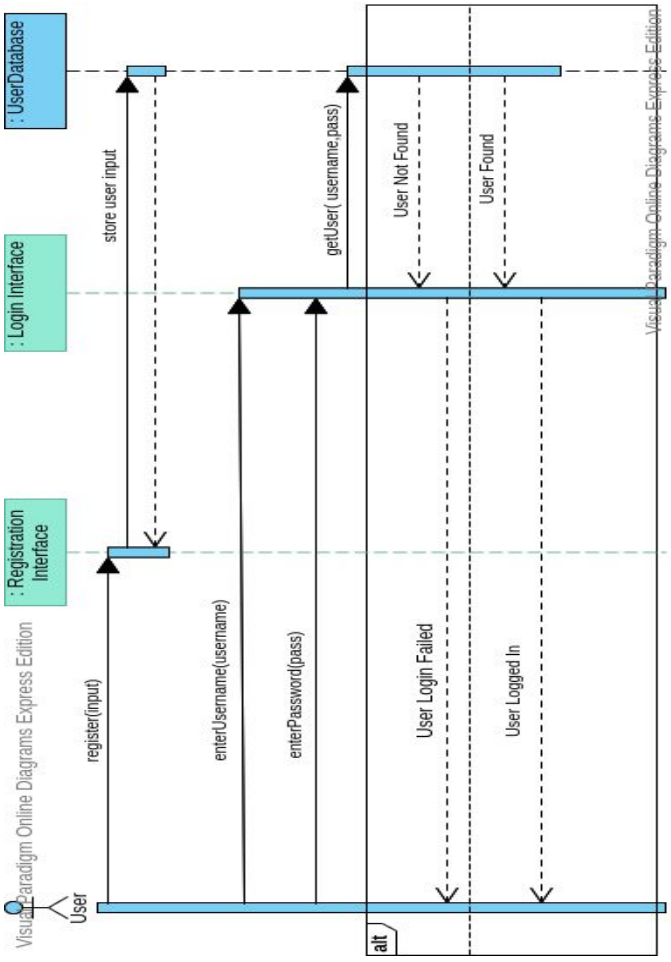

Figure 5: Login Sequence Diagram

3.5.4.1.2 Create Game Instance Sequence Diagram

The diagram shows how a Developer creates a game instance and assigns attributes to it. Game instances are going to consist of cards mainly in our app.

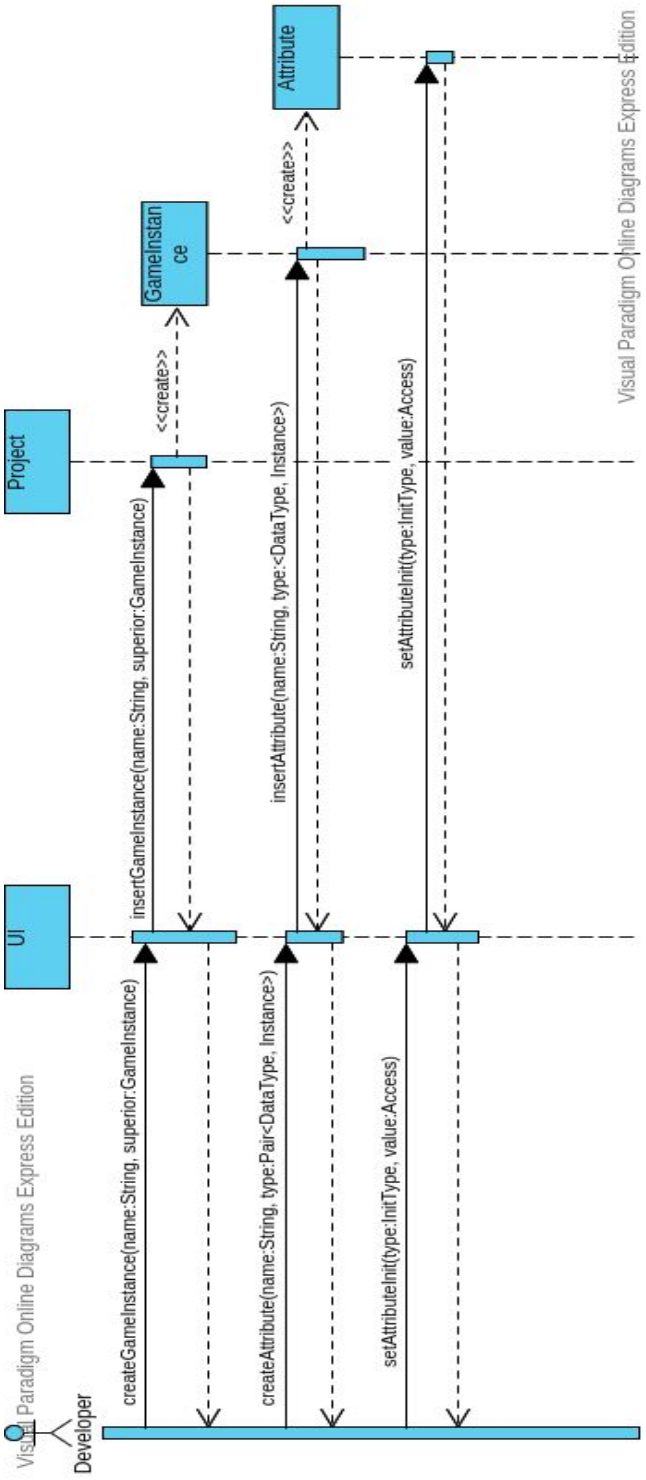

Figure 6: Create Game Instance Sequence Diagram

3.5.4.1.3 Create Game Event Sequence Diagram

The diagram displays how the Developer creates a game event by introducing event triggers. The end result of a game event is in script form.

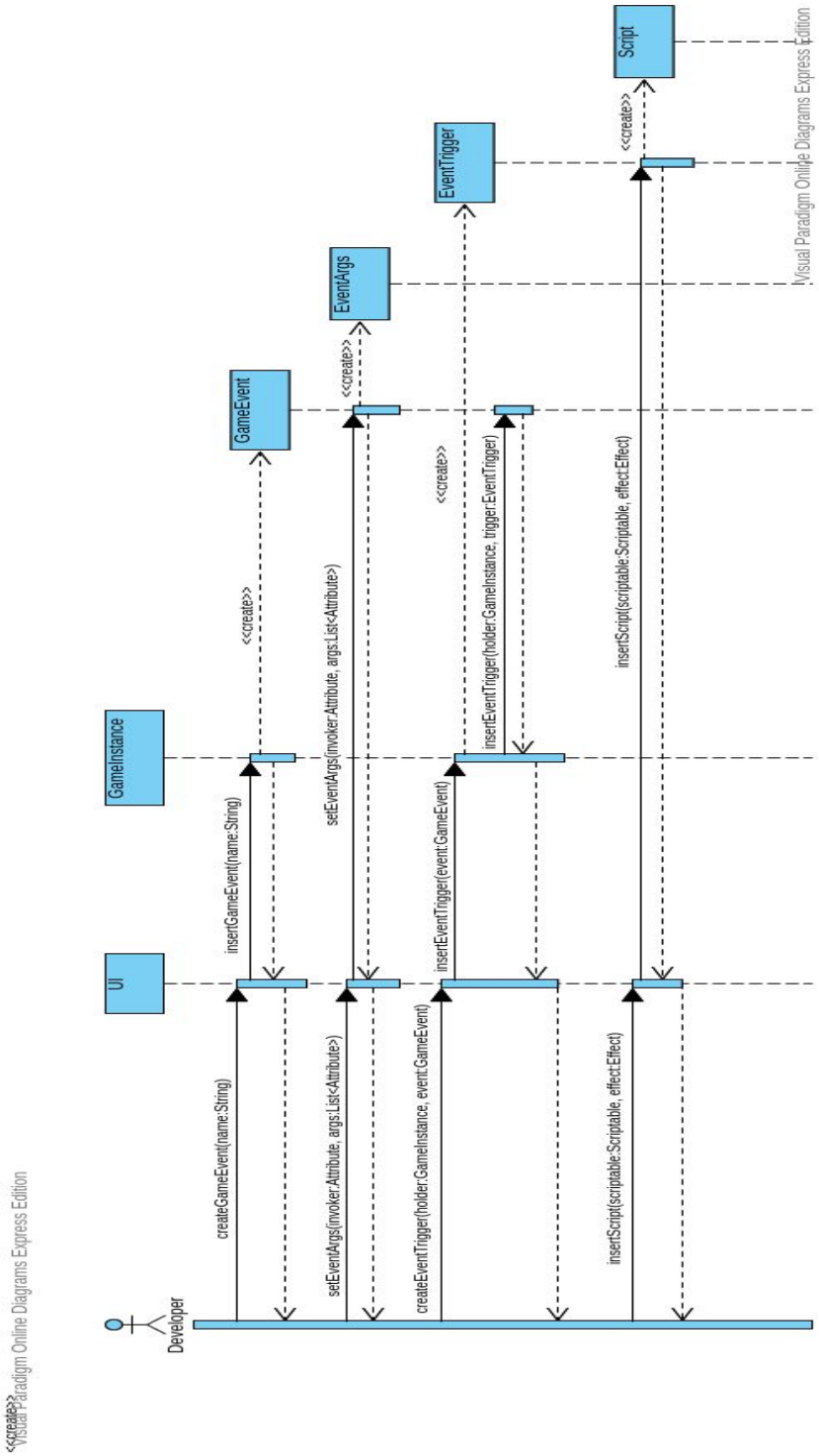

Figure 7: Create Game Event Sequence Diagram

3.5.4.1.4 Create Instance View Sequence Diagram

The following diagram shows how developers can bind their own graphical assets for defined game instances.

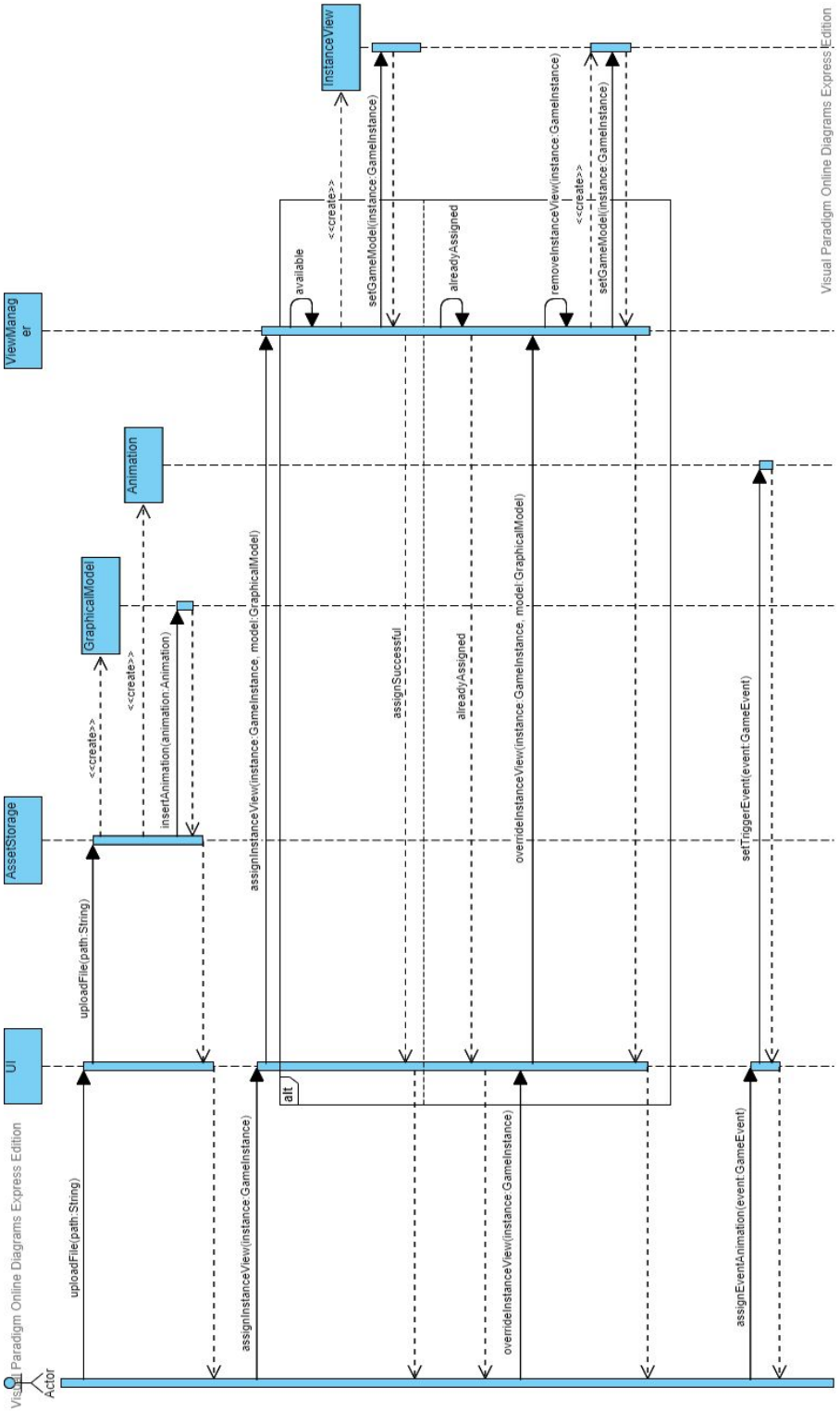

Figure 8: Create Instance View Sequence Diagram

3.5.4.1.5 Play Game Sequence Diagram

The diagram shows how the User joins a lobby to play the game. The game is accessed through the Game Library. If the game runs without an error, the User is navigated to the Game View.

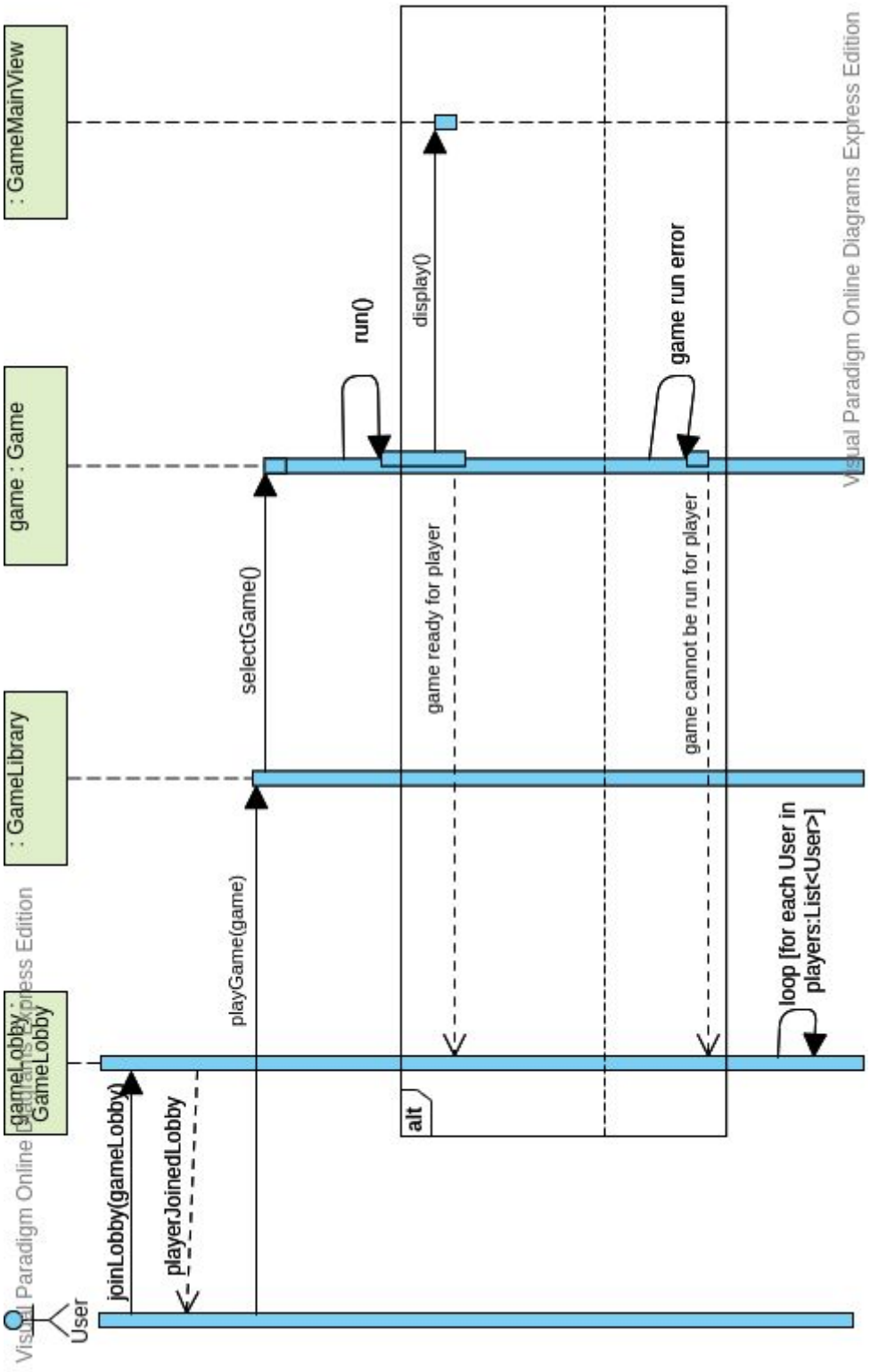

Figure 9: Play Game Sequence Diagram

3.5.4.1.6 Manage Game Library Sequence Diagram

The diagram shows how the User manages their game library by utilizing the features of playing the game, liking the game, disliking the game, commenting on the game, sharing a game, and listing the popular games through the Platform. The Platform is where games are accessed in general view, and the game libraries are lists of games, within the Platform.

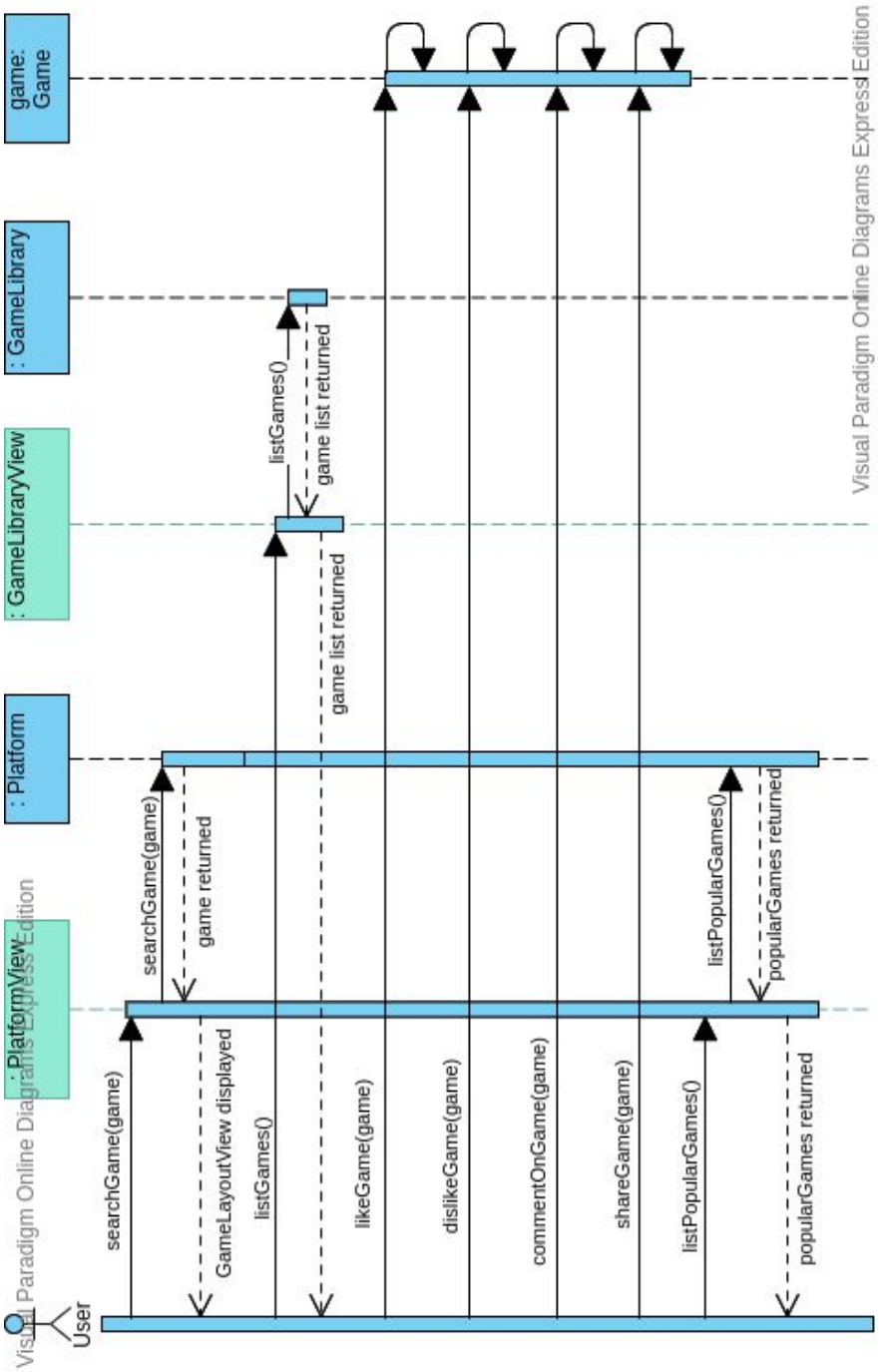

Figure 10: Manage Game Library Sequence Diagram

3.5.4.2 Activity Diagrams

Activity diagrams describe the system by displaying the flow between activities. We produced a diagram that shows the activity flow in our Mobile System.

3.5.4.2.1 Mobile System Activity Diagram

This diagram provides information about how the Mobile System of AugCards navigates through activities, by using the UI elements provided to the user. A more graphical display is provided in the Mock-up section of the report.

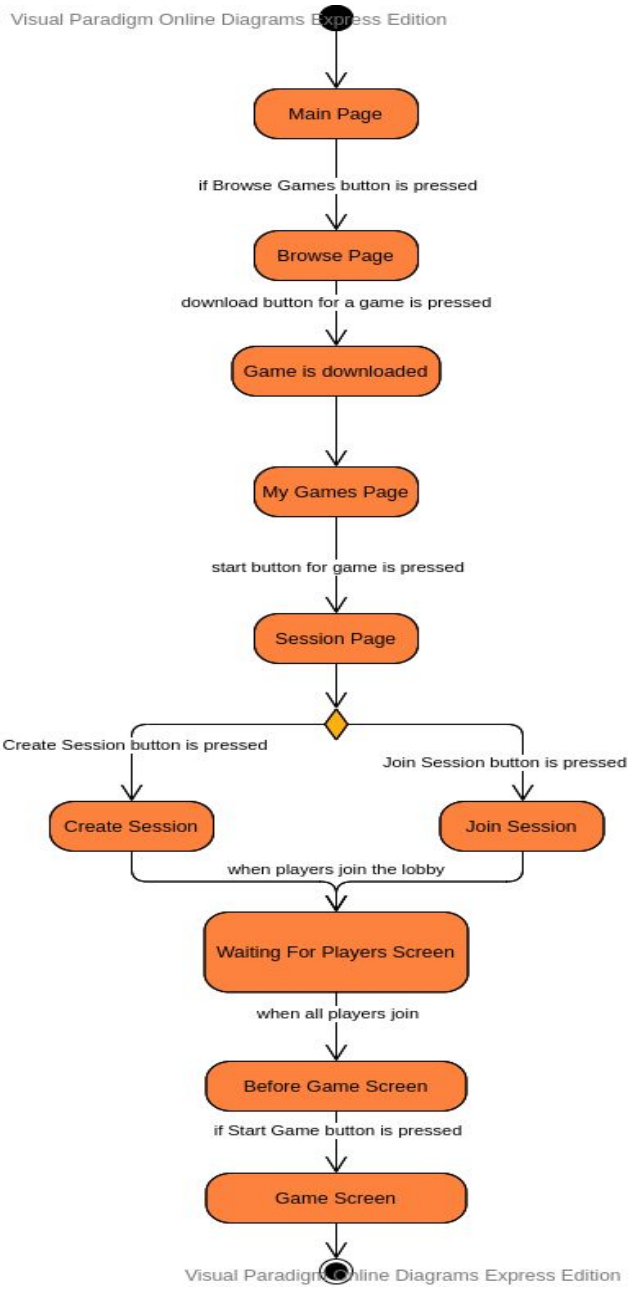

Figure 11: Mobile System Activity Diagram

3.5.4.2.2 Desktop System Activity Diagram

The following diagram demonstrates basic action flow among the application use. Here, the functionalities described on the use-case model furtherly specified regarding with their flow relations. In mock-up section, there will be illustrations of such activities.

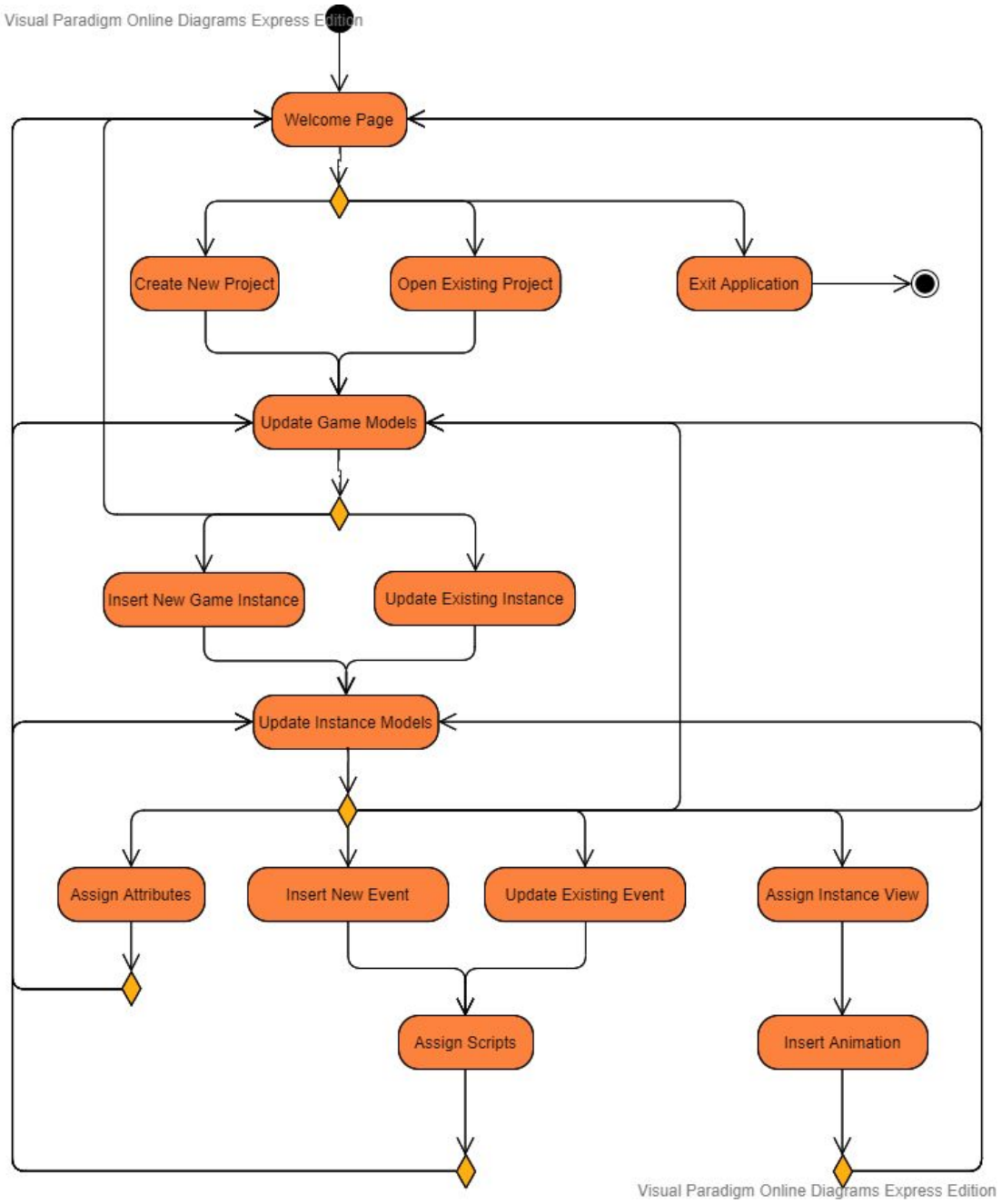

Figure 12: Desktop Activity Diagram

3.5.5 User Interface-Navigational Paths and Screen Mockups

Our project consists of a desktop game creation app and a mobile game launcher app. Desktop app has object creation, event creation, game layout and settings windows.

3.5.5.1 Desktop Application Mockups

3.5.5.2 Object Creation Mockups

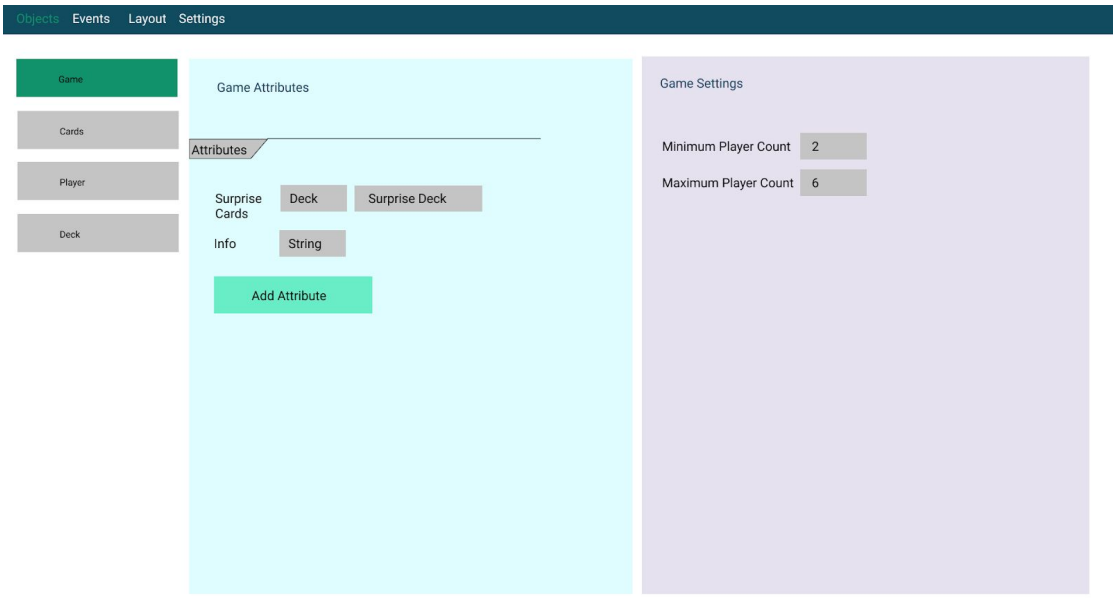

Figure 13: General Game Options

Game settings such as player count will be set here. Also, attributes (variables) of the game will be created here.

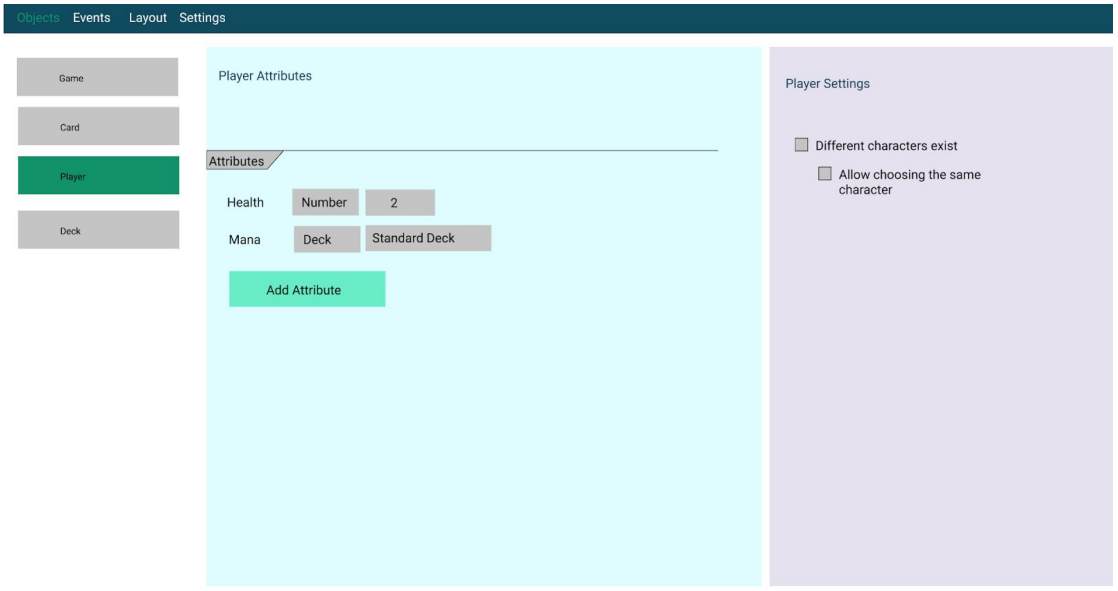

Figure 14: Player Settings

Attributes for each player are defined here. If all the players have the same properties (there aren't different characters like mage, etc.), types and initial values of players' attributes will be set here.

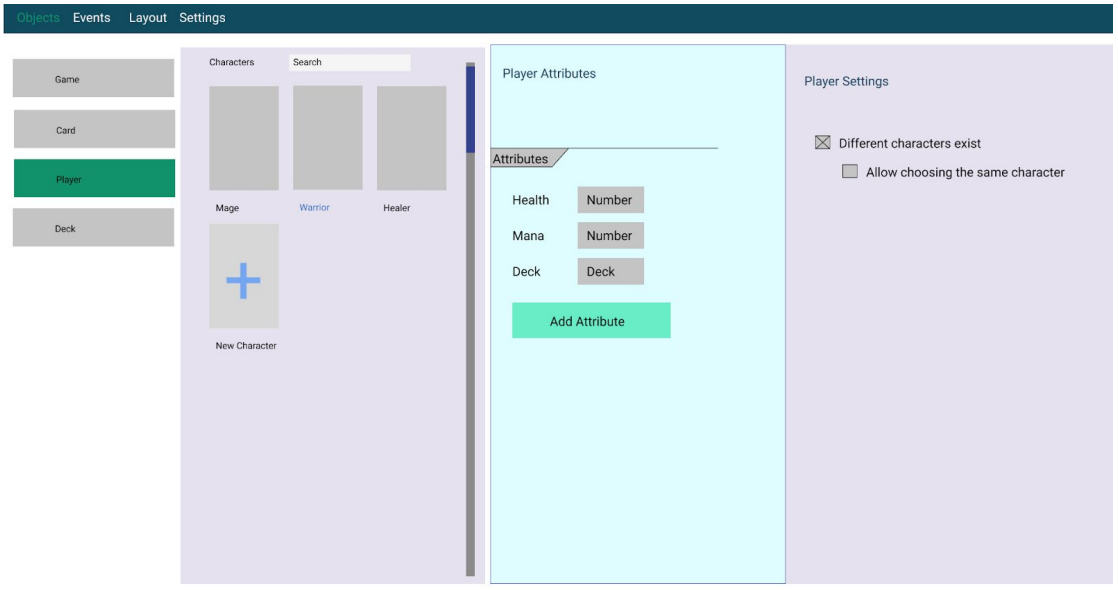

Figure 15: Alternative Player Settings

If a game has different characters (enabled from a checkbom), player types will be created from the left and character attributes will be set for all the characters. Allowing choosing the same character will also be an option.

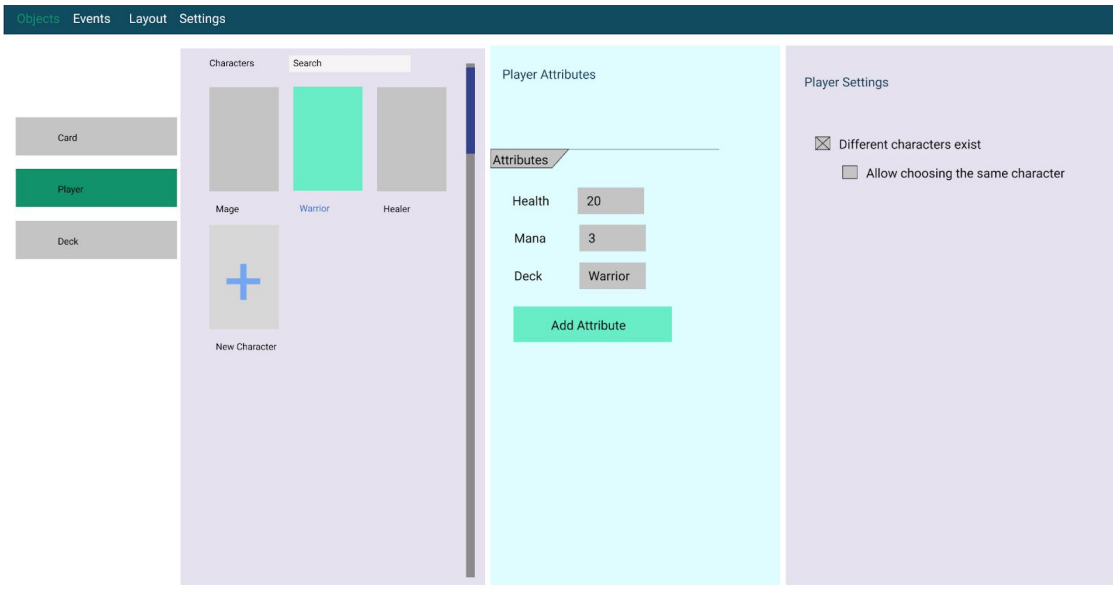

Figure 16: Character Creation

Values of attributes for each character can be set from this menu.

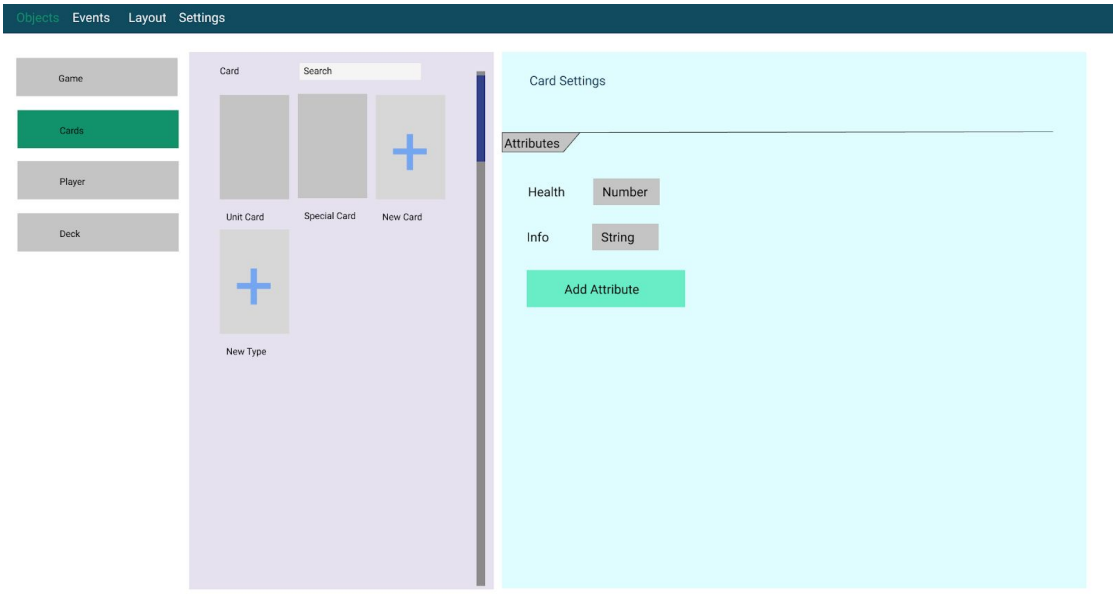

Figure 17: Card Settings

Cards of the game are defined in the cards section. In AugCards, card types and subtypes can be created. Card types and subtypes define card attributes and subtypes inherit parent card types' attributes.

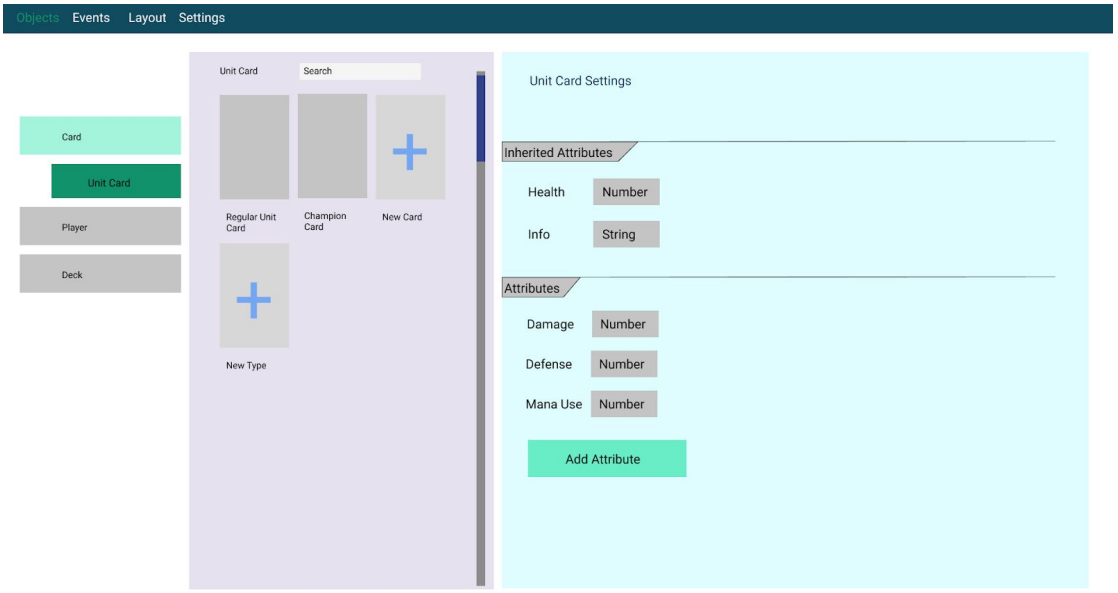

Figure 18: Card Types

Unit cards (Unit Cards is an example, any other type name can be created) inherit card attributes.

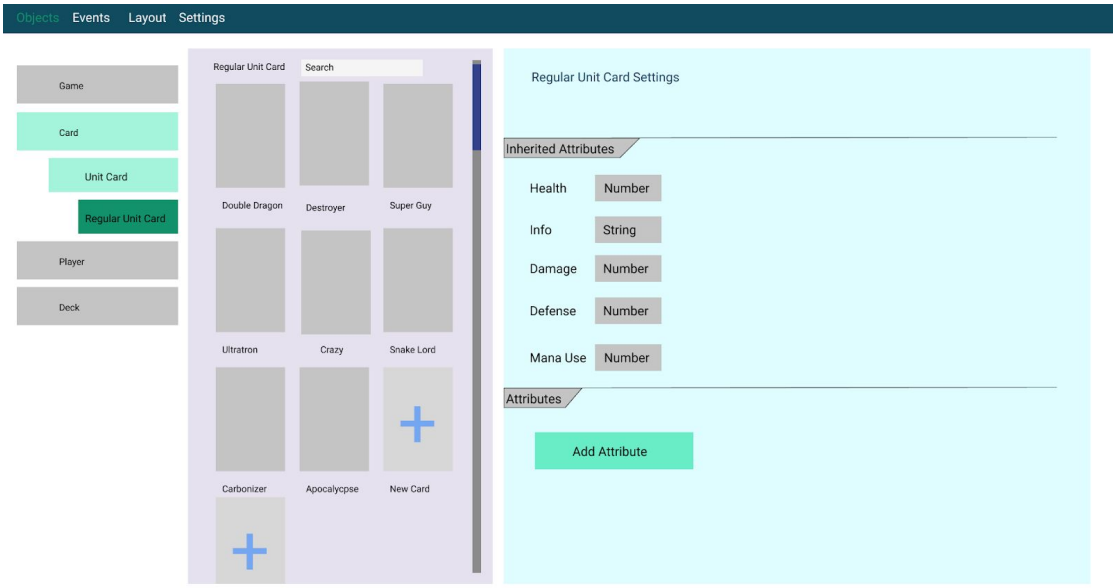

Figure 19: Card Types-II

Regular Unit Cards (also an example) inherit Unit cards.

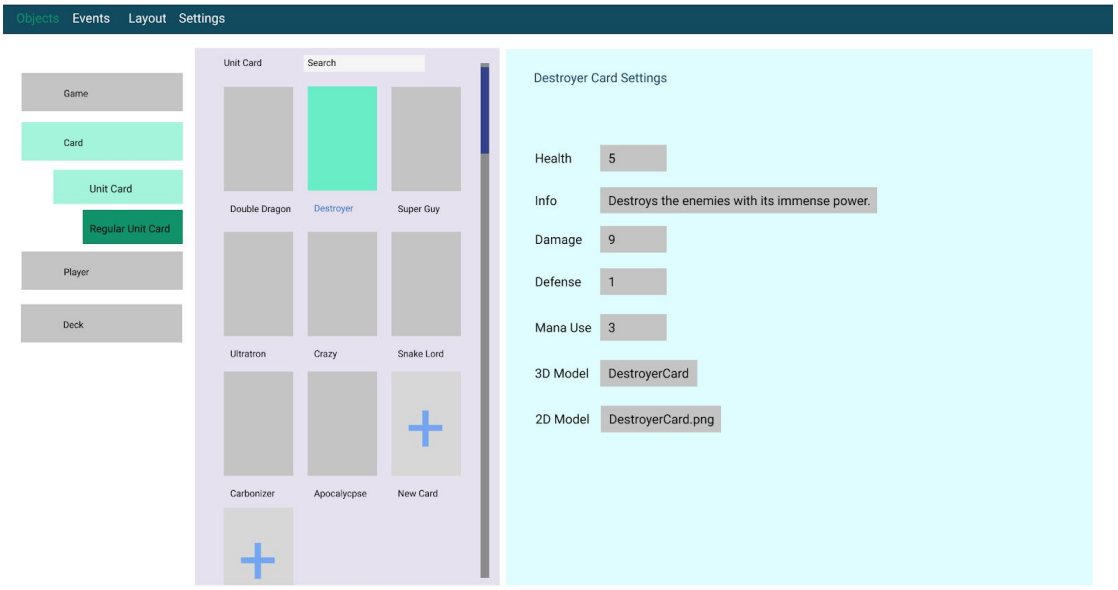

Figure 20: Card Creation

Destroyer is a regular unit card and it's attributes should be set. These attributes are defined in parent types. It's visual model is also set here.

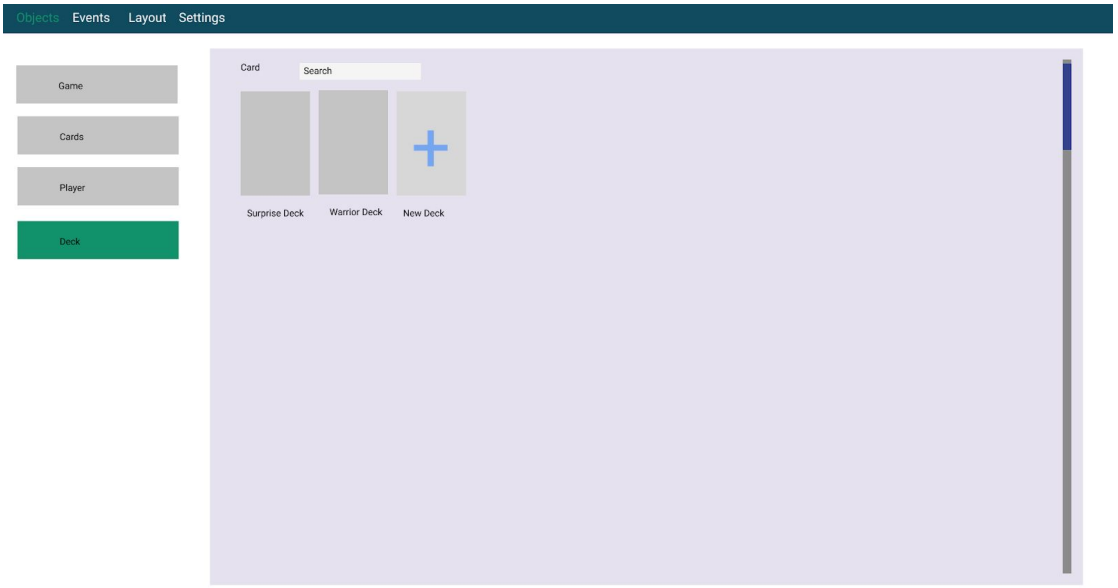

Figure 21: Deck Creation

The games will have decks consisting of cards. New decks can be created here. In-game deck creation is not considered in the first phase. Decks can be player and game attributes.

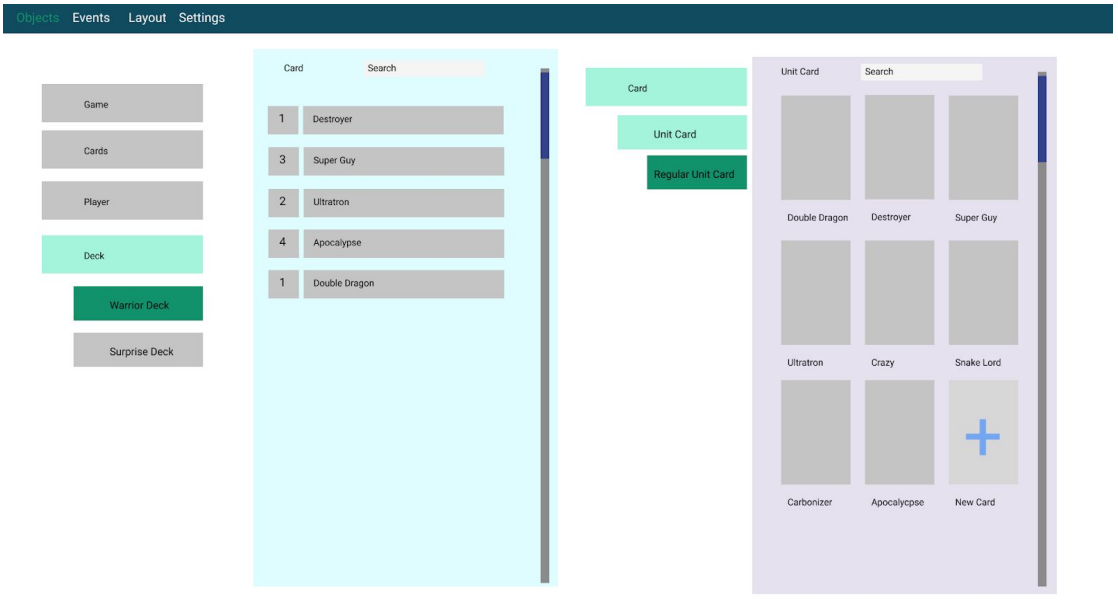

Figure 22: Deck Setup

When clicked on a deck, card navigation will open on the right side of the screen and cards will be moved to decks.

3.5.5.3 Event Creation Mockup

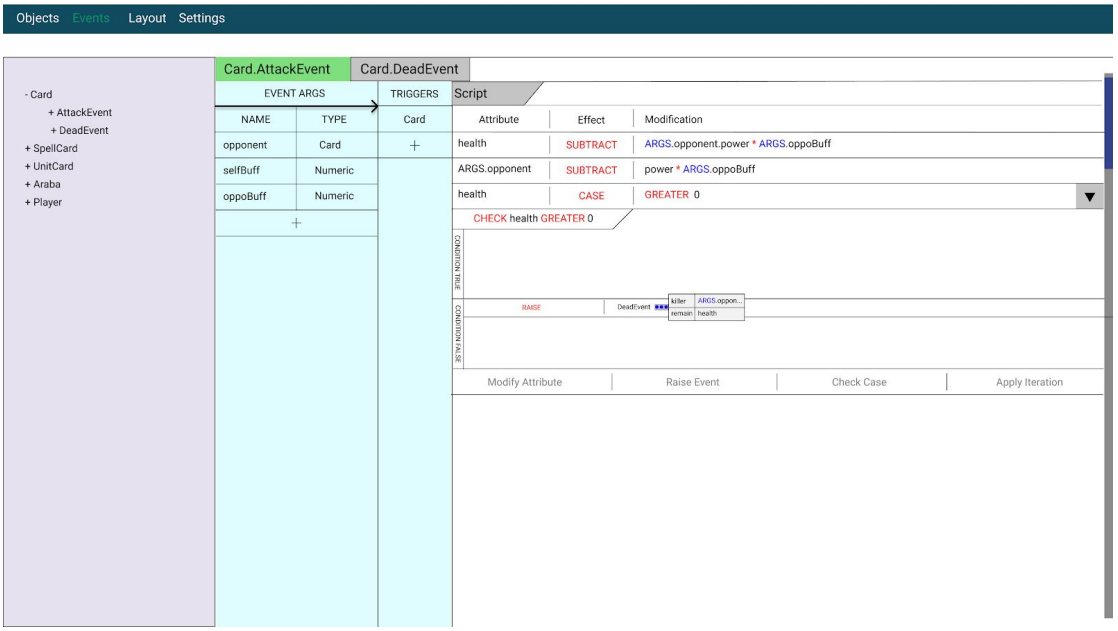

Figure 23: Events

Event design illustration includes two example events defined for Card instance as Attack and Dead events. Attack event furtherly described with event args and some script lines. There are four types of script that can be inserted as Attribute Modification, Event Raising, Case Checking and Iteration Applying. Each script involves three basic elements as Attribute, Effect, Modification.

3.5.5.4 Layout Settings Mockups

Figure 24: Layout Settings

In the layout window, the visual parameters are set.

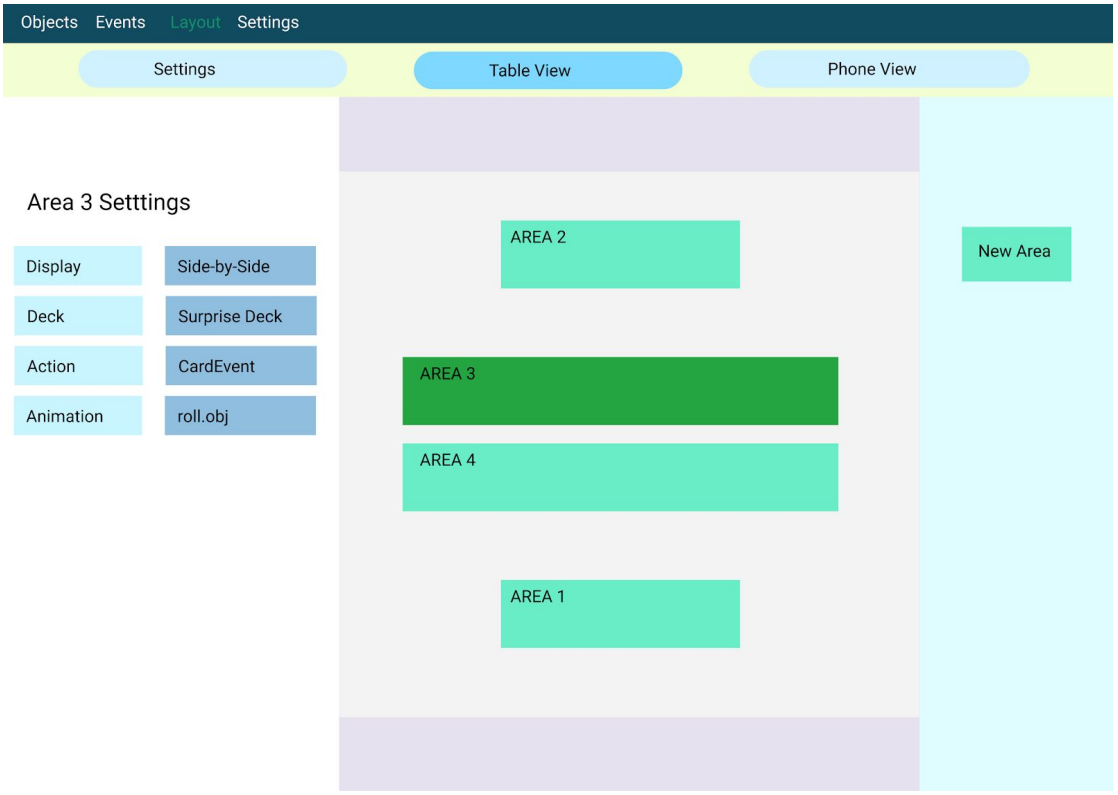

Figure 25: Table View

The games will be played with AR on a table and in this window, areas on the table are created. Areas can have a deck, display setting (show only top, bottom, side-by-side etc.). The event to be fired when clicked on an area can be set. Also click, hover animations can be set.

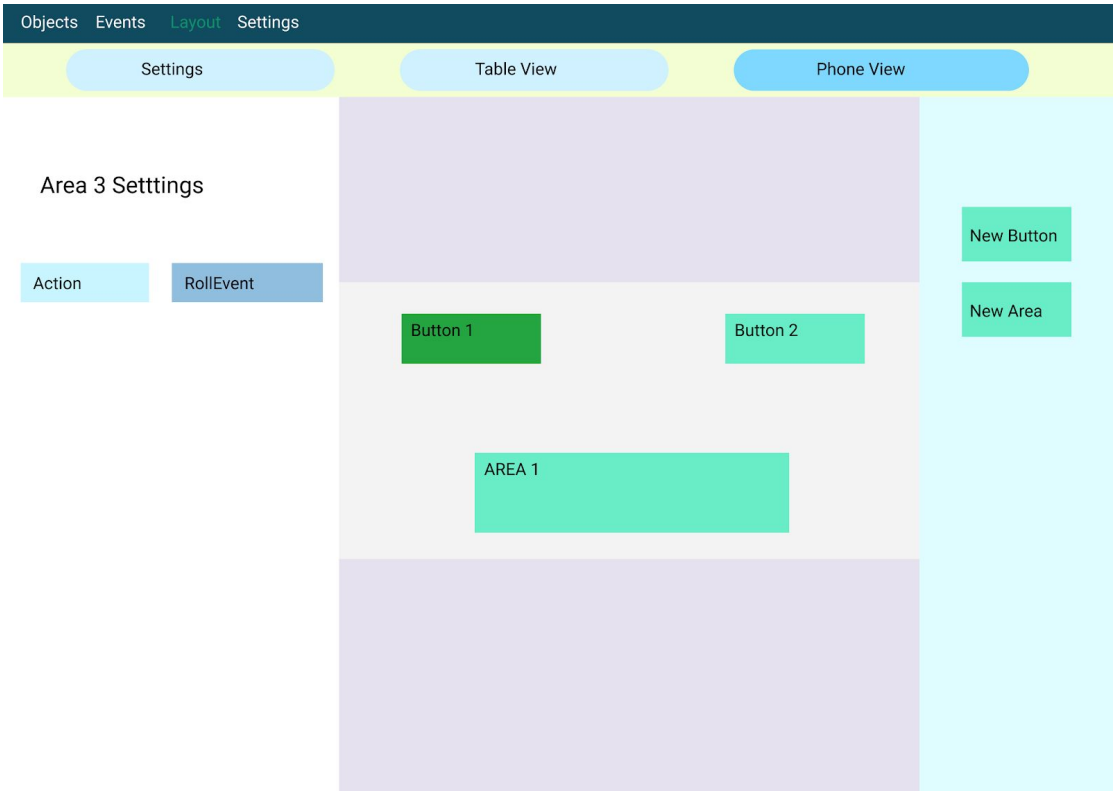

Figure 26: Phone View

Phone screen will show the specified buttons and areas on top of the Camera input. Areas shown on the phone screen can be specified here.

3.5.5.5 Mobile Application Mockups

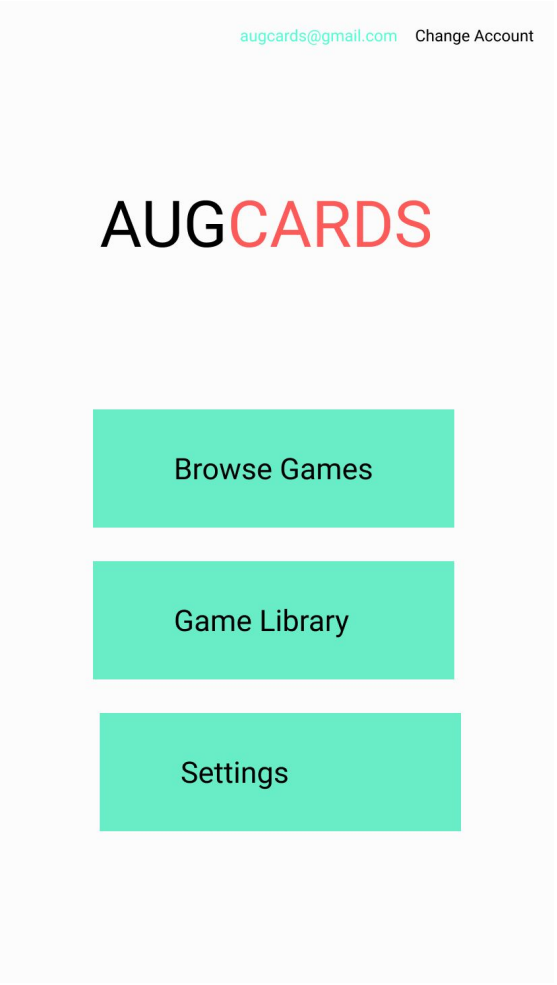

Figure 27: Mobile Home Screen

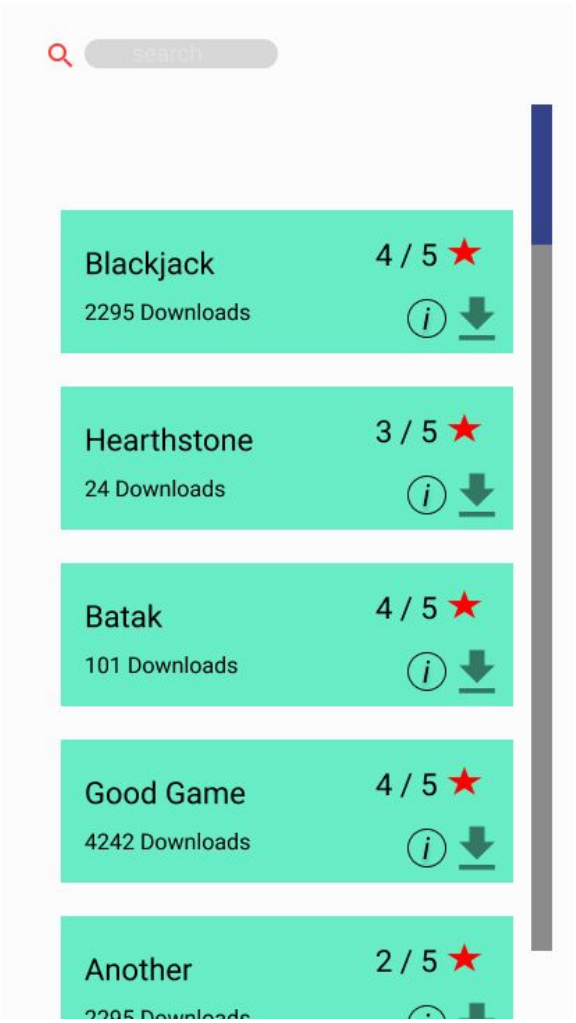

Figure 28: Browse Games

Games exported from the desktop app will be shown here. They can be inspected and downloaded.

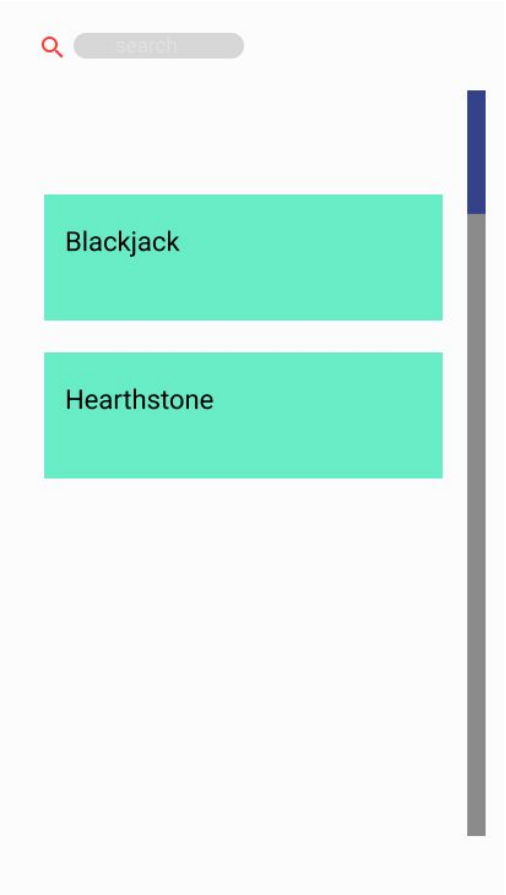

Figure 29: Game Library

Games players downloaded can be seen here.

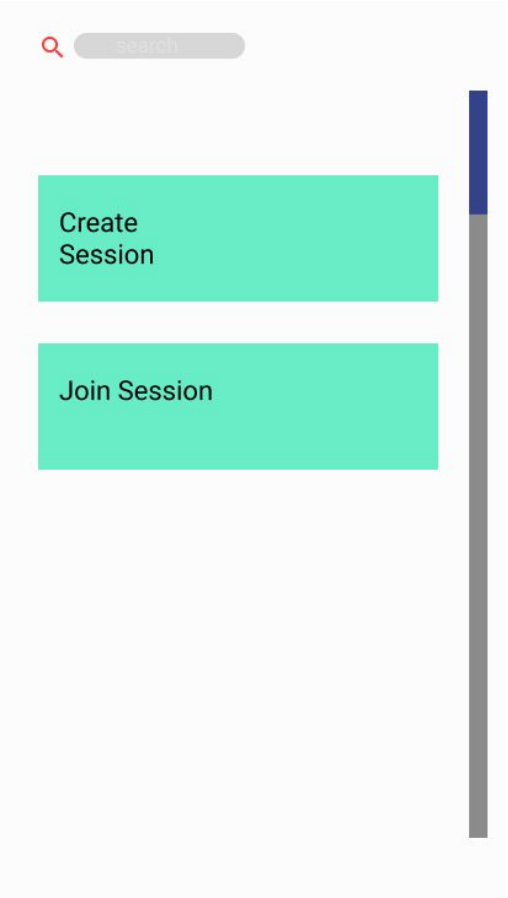

Figure 30: Game Settings

When the player chooses a game, they can either host or join the game.

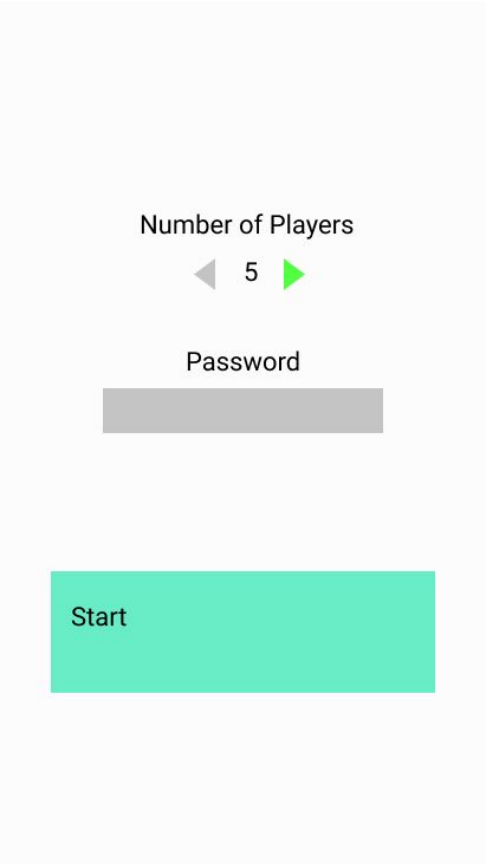

Figure 31: Session Creation

A player can create a game session by setting up parameters.

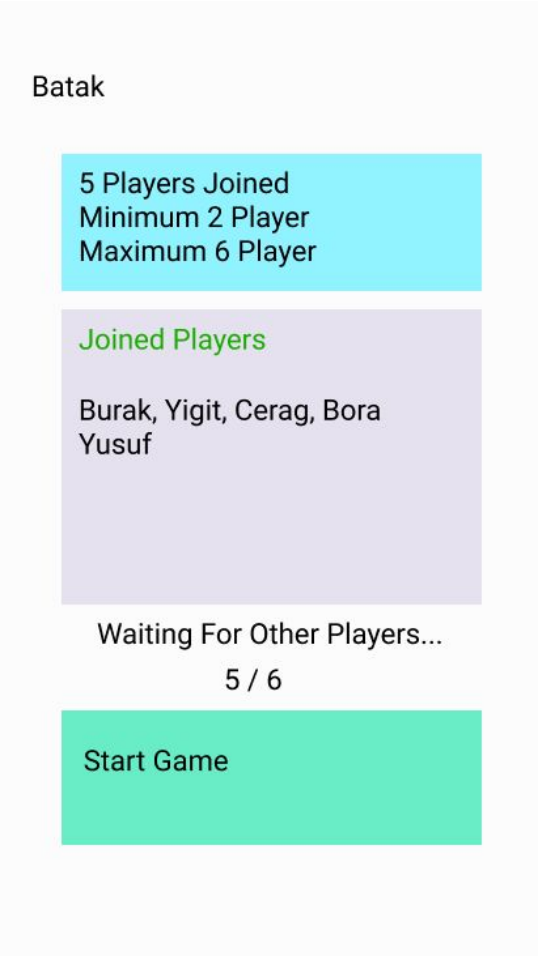

Figure 32: Wait Screen

The host can see the joined players and can start the game.

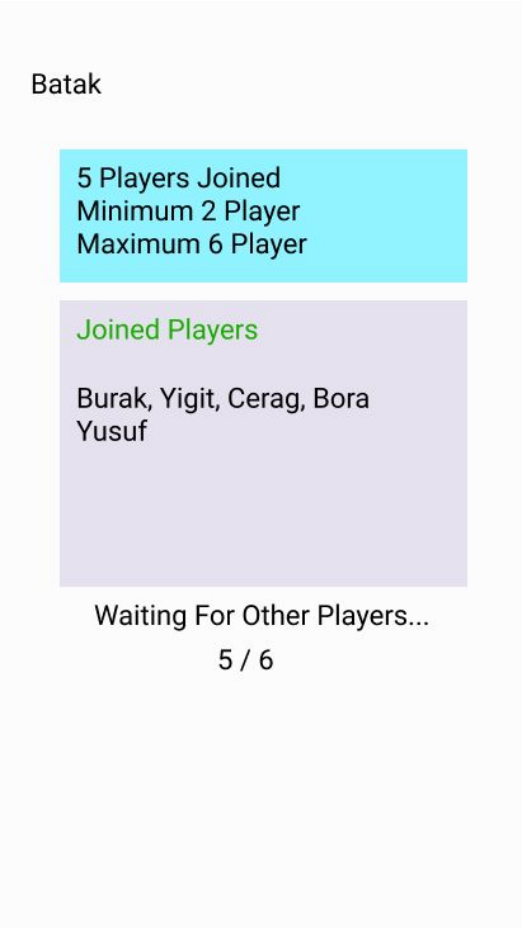

Figure 33: Wait Screen-II

Other players can wait for the game to start.

4 Other Analysis Elements

4.1 Consideration of Various Factors

There are some factors that limit our project that require extra attention.

4.1.1 Public Safety

Safety of the public may be threatened. Since the app offers a variety of options to users, users may use this variety to create games,images,characters that may include offensive figures to others. In order to prevent that, AugCards should have a way of filtering the media,game style, or other components that may be a potential threat to the users.

4.2 Risks and Alternatives

AugCards has several risks regarding implementations which will be faced during the development phase. Those risks will be provided in this section of the report, with their B plans.

1. Unsatisfactory Server Performance:

   AugCards will require us to use Cloud service, which requires servers. It is important to find one that satisfies users in terms of gameplay. AugCards will be GPU Heavy and server communication time is essential for establishing the game within multiple players. If the server performance is unsatisfactory, a better server will be equipped.
2. Infeasible Server Costs:

   We have economic constraints and it poses a risk to be having to provide the server costs. To handle this risk, we will optimize the models and make optimizing changes in our features of AugCards.
3. Low Performance :

   AugCards will use AR supported visuals. Since using AR will highly consume memory and load highly on the GPU, a decrease in performance is a risk. We will do research to utilize more optimum working libraries in terms of graphics.

4. Discrepancy Among Group Members:
   It is a risk that discrepancy occurs within the developers team. In this case, we will obey to be democratic, and trust in the vision of the majority.

Below is a summary table of mentioned risks:

	Likelihood	Effect on the Project	B Plan
Unsatisfactory Server Performance	Medium	Decreases the user experience	Better server will be equipped
Infeasible Server Costs	Medium	Project needs to be optimized	Optimization
Low Performance	High	Decreases the user experience. Project needs to be optimized.	Optimization
Discrepancy Among Group Members	Medium	Decreases working efficiency.	Democratic decision taking

Table 1: Risks and B Plans

4.3 Project Plan

Project planning is crucial to meet the critical deadlines and perform productive progress. There are many project planning methods but the most reasonable one to use in our project is decomposition of the project into the smaller tasks. At this point, it appears that there are four main tasks as the development of Desktop, Mobile, Cloud and Graphics/Network Engine. We should divide these main tasks into further subtasks to ease the distribution of workload among the members and perform the required progress. Firstly, we need to identify project goals:

slm

- Deliver Project Specifications Report
- Build a website for the project
- Deliver Analysis Report
- Develop the initial version of Desktop System
- Develop the initial version of Common Graphics/Network Subsystem
    - Implement the graphics pipeline
    - Implement the network structure (P2P)
- Develop the initial version of Mobile System
- Develop the initial version of Cloud System
- Deliver High-Level Report
- Generate a demo which shows currently developed systems
- Revise and Develop systems furtherly
- Deliver Low-Level Report
- Revise the user interface elements of the systems
- Optimize the systems to provide much usable elements
- Deliver Final Report
- Prepare a project demo and presentation

Deliverables	**Reviewer**	**Members**	**Deadline**
Spec. Report	All	All	Oct. 12, 2020
Website	All	Yiğit, Yusuf	Oct. 9, 2020
Analysis Report	All	All	Nov. 21, 2020
High Level Report	All	All	Dec. 21, 2020
Imp. Desktop Sys.	Burak, Çerağ	Bora, Yusuf	Jan. 10. 2021
Imp. Graphics Pipeline	Burak	Çerağ, Yusuf	Jan. 10. 2021

Imp. Network Structure	Yusuf	Yiğit, Burak	Jan. 10. 2021
Imp. Mobile Sys.	Yiğit, Bora	Çerağ	Jan. 10. 2021
Imp. Cloud Sys.	Yusuf	Yiğit	Jan. 10. 2021
First Demo	All	All	Jan. 15, 2021
Revise and Further Dev.	All	All	?
Low Level Report	All	All	Feb. 8, 2021
Revise User Exp.	All	All	?
Optimize Sys.	All	All	?
Final Report	All	All	Apr. 30, 2021
Final Demo	All	All	May. 15, 2021

Table 2: Tasks-Work Dist. and Deadlines

The mentioned tasks are tentative to be branched into further subtasks and their deadlines are subject to change.

4.4 Ensuring Proper Teamwork

We were confident in each other before the senior project because most of us did projects together before and we discussed and agreed on that everyone will contribute equally to the project. We are also utilising several methods to ensure that the teamwork is done and distributed properly.

4.4.1 Weekly Meetings

We are doing weekly meetings every monday before CS491 seminars. In those meetings, each of us discuss and present what we have done and we decide what each of us will do next week. We make important decisions about the project in these meetings. These meetings help

ensure that everyone is doing their job properly and the project is being developed properly. Meetings take place on Discord or Zoom.

4.4.2 Using Project Management Tools

We are using Trello to keep track of each group member's tasks. We have chosen to use Trello because it is free and we can divide the tasks to ToDo, In Progress, In Review, Done and possibly other categories. We assign group members and deadlines to tasks.

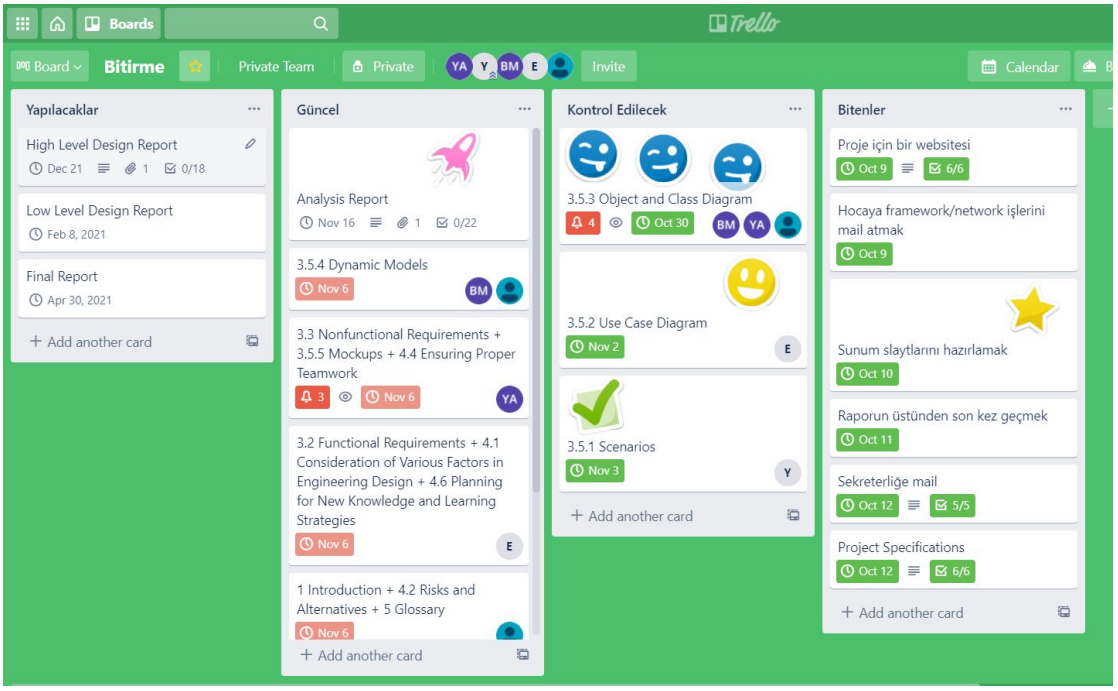

Figure 34: Trello View

4.4.3 Code Review

We want to ensure a high quality codebase in our project and we plan to enforce code reviews by another group member for each pull request on GitHub. Code reviews will also ensure that everyone has written high quality code.

4.4.4 Pair Tasks

We are giving some of the big and important tasks to 2 group members. We are inspired by pair programming. It's advantageous because 2 people have higher chances than one person in catching errors and maintaining high quality because one person may overlook errors. Tasks that are done in pairs are simultaneously reviewed.

4.5 Ethics and Professional Responsibilities

In this section of the Analysis Report, the ethical and professional responsibilities that arise with the development of this project are going to be discussed.

4.5.1 Ethics

First of all, the content of the games should be monitored so that they do not contain offensive content, hate symbols or speech. This is important since we intend to create a platform for people that want to have fun playing and designing games.

Another issue is that we need to keep the creative rights of game developers in mind, since these rights allow developers to protect their work from being stolen or copied. The games should also be monitored to make sure that this occurs as little as possible.

Privacy of the users' data should also be prioritised, since we need to create profiles for users and keep their emails. The emails will only be kept for profile purposes, and not be shared with third parties.

4.5.2 Professional Responsibilities

We will communicate through online meetings and messaging, since there is a pandemic going on. We conduct weekly meetings through Discord, an online communication application. For important decisions and changes, we use WhatsApp to communicate.

We will keep our GitHub repository private for the time being since it would benefit us in terms of security and stopping people from stealing or copying our idea.

4.6 Planning for New Knowledge and Learning Strategies

Our current knowledge is not enough to develop this project.There are various new technologies and skills we need to develop in order to complete this project successfully.

- Augmented Reality
- Cloud Computing
- Networking
- Android Development

Learning AR is necessary since the Android hand of the application will be supported by AR.Cloud computing is necessary in order for users to store their games to Android. Note that users will store their desktop games in the cloud, where users will be able to download it to their mobiles.
We will use similar techniques that we used to communicate and learn in the CS319 course. We will do weekly meetings, report every meeting in written format. We will learn new technologies by constantly practising,learning step by step.

5 Glossary

Game Instance A specific instance in AugCards to represent the custom game objects like card, player, avatar defined by the developer.

Game Event A specific event in AugCards to represent the custom game events like attack, play card or navigate to the next turn, defined by the developer.

Event Trigger An event trigger represents the trigger mechanisms for defined events.

Game Rules A set of conditioners for a game specified by Developers in AugCards to represent the rules of the created game.

Session A session refers to a game session in which the game is played by the players.

Asset An abstract class to generalize the graphical elements.

Animation A specific Asset to represent the pre-designed transform sequence for graphical models within animation data.

Developer A developer of AugCards represents the type of actor in the system which creates card games.

Player A player of AugCards represents the type of actor in the system which attends games.

Platform A platform in AugCards represents the common point where users and developers meet through shared games.

Game Library A game library in AugCards represents the customizable game storage where users can insert new ones and pick favorites.

Game Lobby A game lobby in AugCards represents created and in-preparation game sessions in which players can join.

GPU Graphics Processing Unit. GPU is designed for handling graphics operations, including 2D and 3D calculations to render 3D graphics [5].

Git A version control system used for project teams for reviewing and tracing code changes.

GitHub An online platform which hosts software development versions for software development teams by using Git.

Trello Trello is a collaboration tool that organizes your projects into boards [6].

Augmented Reality Augmented Reality is a technology for producing an enhanced environment [7].

Android System The Android operating system is a mobile operating system developed for mobile platforms.

Discord an American VoIP, instant messaging and digital distribution platform designed for creating communities [8].

WhatsApp WhatsApp is a messenger cross-platform instant messaging application.

Dulst Dulst is an online card game playing software [9].

UML Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams [10].

Vuforia Vuforia is an engine that supports the use of AR and computer vision functionalities [11].

6 References

[1] "Topic: Mobile gaming." [Online]. Available: https://www.statista.com/topics/1906/mobile-gaming/. [Accessed: 09-Oct-2020]

[2] "Dulst." [Online]. Available: https://dulst.com/. [Accessed: 08-Oct-2020]

[3] "Legends of Runeterra." [Online]. Available: https://playruneterra.com/tr-tr/. [Accessed: 09-Oct-2020]

[4] "MIT License." [Online]. Available: https://mit-license.org/. [Accessed: 08-Oct-2020]

[5] "GPU (Graphics Processing Unit) Definition." [Online]. Available: https://techterms.com/definition/gpu. [Accessed: 21-Nov-2020]

[6] Trello, "What is Trello?" [Online]. Available: https://help.trello.com. [Accessed: 21-Nov-2020]

[7] "augmented reality." [Online]. Available: https://www.dictionary.com. [Accessed: 21-Nov-2020]

[8] Contributors to Wikimedia projects, "Discord (software)," 30-Jan-2016. [Online]. Available: https://en.wikipedia.org/wiki/Discord_(software). [Accessed: 21-Nov-2020]

[9] "Dulst." [Online]. Available: https://dulst.com/. [Accessed: 21-Nov-2020]

[10] "What is Unified Modeling Language (UML)?" [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/. [Accessed: 21-Nov-2020]

[11] "Vuforia Developer Portal." [Online]. Available: https://developer.vuforia.com/. [Accessed: 21-Nov-2020]