Bilkent University

Department Of Computer Engineering

Senior Design Project

Project short-name: AugCards

Low Level Design Report

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

Supervisor: Prof. Dr. Uğur Güdükbay
Jury Members: TBA
Innovation Expert: Prof. Dr. Veysi İşler

Low Level Design Report
February 08, 2021

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.

Table of Contents

Low Level Design 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 a low level design of the system. First, the trade-offs of the system and the engineering standards will be discussed. Then, the packages and interfaces of AugCards software of the software architecture

will be described under headers: Mobile and Desktop subsystems. Additionally, class diagrams will be presented in this report.

1.1 Object Design Trade-offs

1.1.1 User Friendliness vs Functionality

One of our focuses is a good user experience. We make the game simple, clear and understandable. Thus, the game will not include complex functionality or too many options. For instance, we don't include complex card development options so that the user will not make an effort to learn how to play the game with those, rather than that, the user can enjoy the simple game, and everything will be clear.

1.1.2 Compatibility vs Extensibility

In the regard of compatibility, the mobile system has a crucial constraint on the library used for AR support. Many AR libraries support Android or iOS or both, but many of them support a range of Android or iOS devices. That is, device compatibility is an important consideration in choosing the right AR library. We will use Vuforia to provide AR support. Vuforia has an Android SDK to develop AR applications on a wide range of Android devices which are our target in the project. On the desktop side, we will use Java to develop the application, that is, it will be a cross-platform game design application.

Extensibility is one of the fundamental requirements for both game applications and design tools. Initially, we focused on providing the maximum customizability in both visual aspects and game concept. However, there should be user-friendly interactions in the games from the player perspective. That is, we should follow an extensible development progress to insert such features in the proceeding progress. We should provide an integration of such interactions into the games on the desktop side and reflect them to the games on the mobile side.

1.1.3 Quality vs File Size

Higher quality graphics data require high storage. Our cloud system needs to store game data which includes graphics data such as textures and animations.

This storage requirement is a constraint for us because of cloud storage's cost. Also, it will cause longer download times for game players and longer upload times for game creators. Thus, we have to limit asset size to be able to maintain the service. We want to allow a reasonable game size (around 50MB) that will allow good graphic and animation quality. We may also provide paid options to be able to use big sized assets.

1.1.4 Robustness vs Cost

AugCards' aim is to provide a service such that Its' outputs are reliable and provides the user the AugCards functionalities to the fullest extent. In order to achieve this, better services will be used regarding cloud maintenance. We will use Firebase for cloud maintenance as It is more preferred for services which are not large [4]. Using Firebase services will cost money, so AugCards' robustness is achieved with more cost of money, which is a design goal to be automated to the largest extent.

1.2 Interface Documentation Guidelines

All of our classes are named according to PascalCase. Attribute and method names follow the camelCase convention. In the interface documentation, we use a table to describe our class interfaces, which includes the name, description, class attributes, class methods and the descriptions of these methods. A sample table can be found below.

class SampleClass	
This is a sample class...	
Attributes	
private String sampleString private int sampleInt	
Methods	
public String getString() public void setNo(int no)	returns sample string sets the no

1.3 Engineering Standards

For the descriptions of the class interfaces, as well as the diagrams, scenarios and the use cases, followed by the subsystem compositions and hardware depictions, our reports utilizes the guidelines of UML as this is widely used for the purpose of generating diagrams [10]. For the citations, we used IEEE standards as they are widely used in reports within the engineering domain.

1.4 Definitions, Acronyms, and Abbreviations

AR: AR (Augmented Reality) is an interactive experience of a real-world environment where the objects that reside in the real world are enhanced by computer-generated perceptual information. Accordance between real-world environment and computer generated information is ensured via sensors (or camera) and algorithms.
Action: Actions are building blocks of events. 15 different actions with different parameters are available to the game creators.
Event: Game rules are implemented by using events. Events consist of actions. Custom events are triggered by the main event.
Card: Cards are customizable and interactive game objects.
Deck: Card Collections.
Attribute: Properties of game objects.
Trigger: Event handling mechanisms which consists of some script lines. In this way, it determines the event flow of game logic.
Script: Instructive lines to specify updates on the game objects during event flow.
Expression: The struct to refer game object reference or their operational relations.

2 Packages

AugCards' subsystem decomposition consists of a Mobile subsystem and a Desktop subsystem regarding the platforms which will be used in the software by the users. A subsystem diagram and composition carries great importance for our project, as it displays the interactions and information exchange between the systems, and how It composes a system of harmonically working subsystems. The information exchange that is happening between desktop and mobile

systems includes the compilation of the game created on the desktop, in the mobile system. For example, the clarification of this process is of great importance regarding the implementation of both of the systems such that less error is faced by the developers. Regarding these issues, in our report, we gave great consideration into subsystem composition and software architecture designs.

2.1 Desktop Subsystem

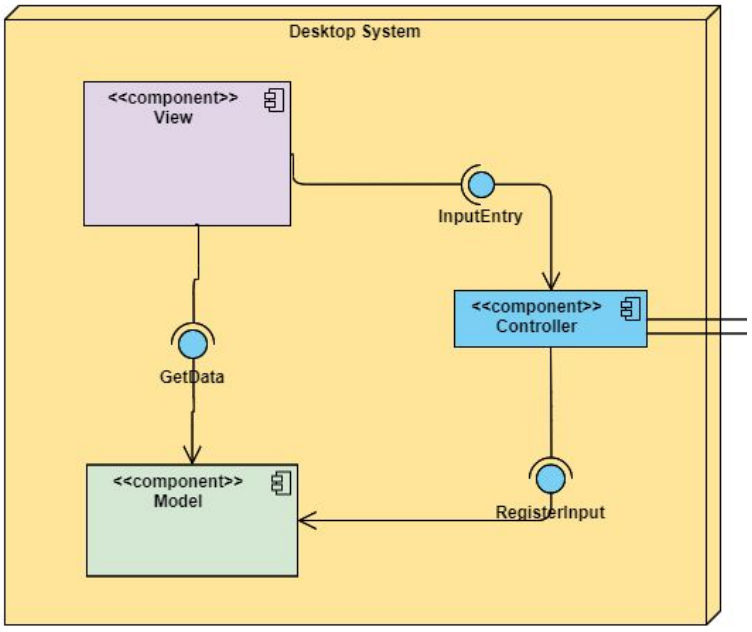

Figure 3: Desktop subsystem's components

The desktop subsystem is responsible for handling the Model and View components of the desktop application of AugCards which uses a Controller component for receiving and providing information from the Mobile subsystem of AugCards.

View: The View component is responsible for the UI operations of the Desktop application. It is related to the Controller via input entries.
Model: The Model component implements the logic of the Desktop application which shares data between the View component and is related to Controller with its register input.
Controller: The Controller subsystem is responsible for handling the information exchange between the Mobile and Desktop applications.

2.1.1 View Component

The View component manages the UI of our desktop application, communicating with the Model component through the Controller component.

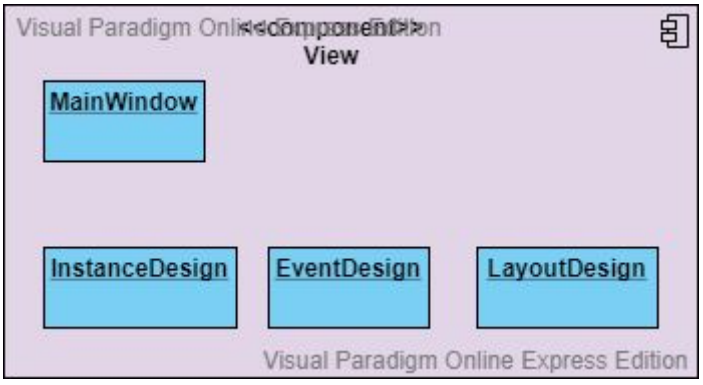

Figure 4: View component and its classes

MainWindow: The main window of the application. It contains other components and lets the user see what they are doing.
InstanceDesign: Where the user designs instances.
EventDesign: Where the user designs events.
LayoutDesign: Where the user designs layouts of their game.

2.1.2 Model Component

Model component is supposed to store the information for designed games, that is, defined objects and game rules.

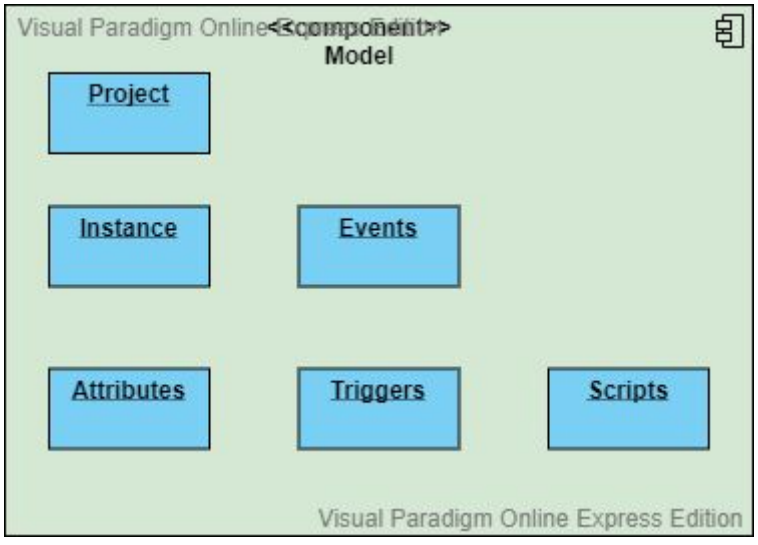

Figure 5: Model component and its classes

Project: A class to represent a particular game project. It stores all custom elements like **Instances** and **Events.**

Instance: A class to represent a specific game object like card, player, etc. It can store any custom **Attributes** defined.

Events: A class to represent a specific game event like game-init, card-attack, etc. It should store all its own **Triggers** defined.

Attributes: A class to represent a property of defined **Instance.** It should store its reference name, type and initial value.

Triggers: A class to represent a handling mechanism of an **Event.** It should store a sequence of **Scripts** to define the handling mechanism.

Scripts: A class to represent an effective change on an **Attribute.** It should store its effect type, focus attribute and modifiers.

2.1.3 Controller Component

Controller component basically manages the application flow according with incoming inputs from the View component. Application flow contains authentication and exporting game data.

Figure 6: Controller component and its classes

AuthenticationManager: A manager class to handle authentication requests for the desktop system. It should communicate with **Cloud Subsystem** to check users' access permission.

ModelManager: A manager class to handle manipulations on game models according to the inputs from View Component. It should apply manipulations to Data Component and generate output files for game models.

2.2 Mobile Subsystem

The details of the mobile subsystems is provided below.

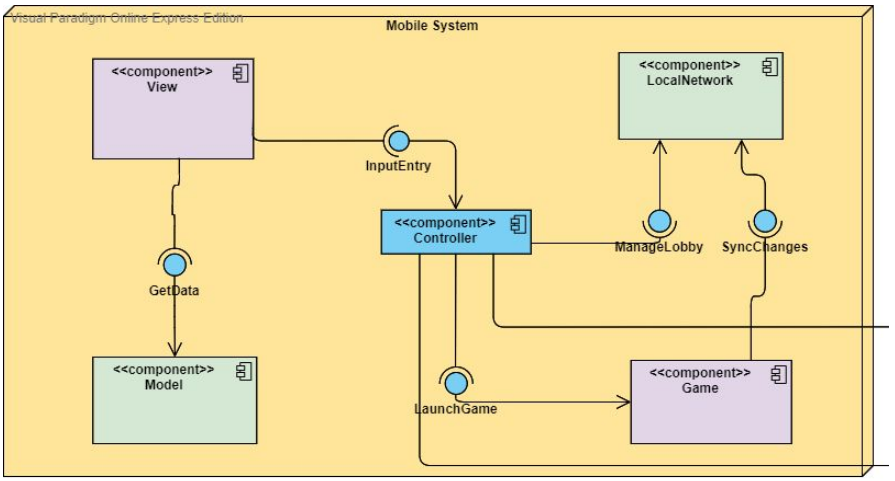

Figure 7: Mobile subsystem's components

2.2.1 View Component

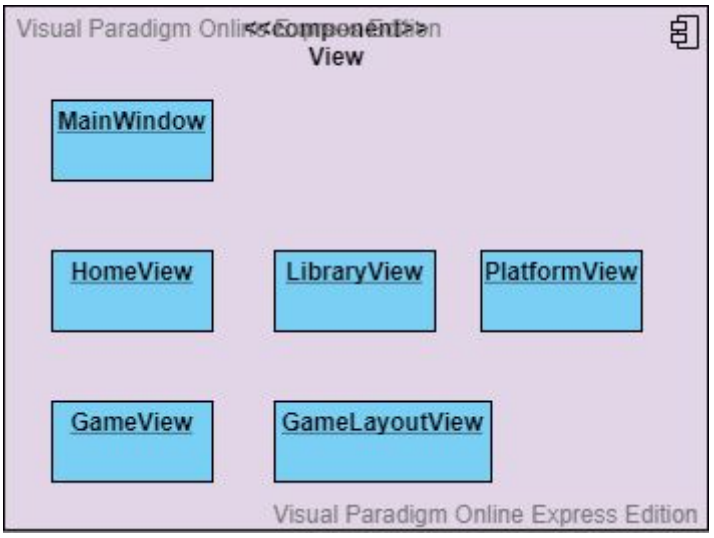

Figure 8: View component and its classes

The View component is composed of the following parts:

MainWindow: A class which provides the view of the main window of AugCards.

HomeView: A class that is responsible for the display of the home page.

GameView: A class that represents the view of the game to be played in AugCards.

LibraryView: A class which provides the library view of AugCards which displays a number of games.

GameLayoutView: A class to represent the layout of where each game instance will be placed for a game.

PlatformView: A class that is to display the platform where users can search for games.

2.2.2 Model Component

The Model component keeps information about the app itself and the game is managed by another component. This component keeps user information such as authentication information and the user's game library, as well as the game lobby.

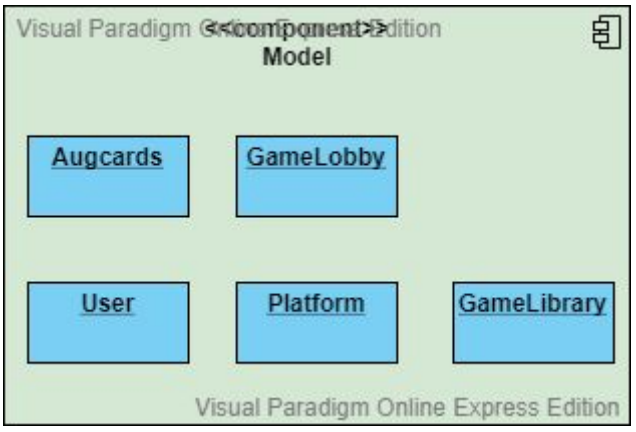

Figure 9: Model component and its classes

Augcards: The mainframe of the mobile game.
GameLobby: Information about the running game lobby is kept here.
User: Information about the user themselves will be managed by this class.
Platform: The shared games and their community pages will be kept here.
GameLibrary: The user's game library information will be kept here.

2.2.3 Controller Component

Controller component basically manages the application flow according with incoming inputs from the View component. Such application flow may contain authentication or creating/joining a lobby or launching a game session.

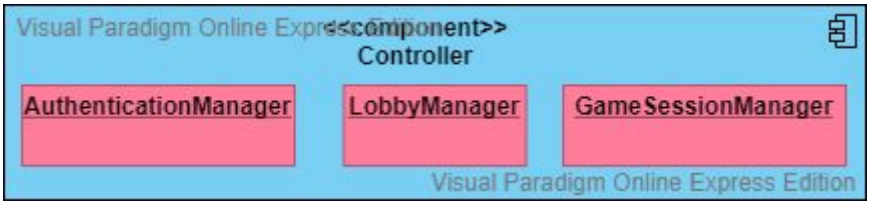

Figure 10: Controller component and its classes

AuthenticationManager: A manager class to handle authentication requests. It should communicate with **Cloud Subsystem** to check users' access permission.
LobbyManager: A manager class to handle lobby joining/creation requests. It should communicate with **Local Network Component** to establish connection with a lobby.
GameSessionManager: A manager class to handle game launch requests. It should communicate with **Game Component** to run a game execution process.

2.2.4 Game Component

Game component is responsible for executing the actual game. It loads a downloaded game model and it's assets and parses the data to create a game session.

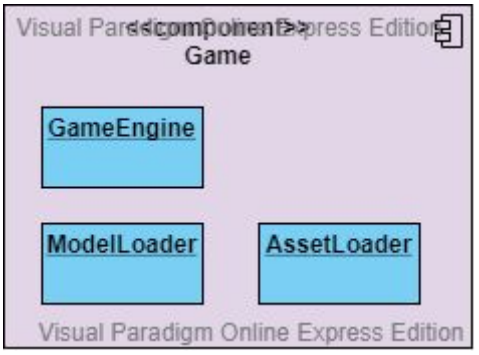

Figure 11: Game component and its classes

GameEngine: Executes the game using the game model and game assets.
ModelLoader: Loads the game model data for the game.

AssetLoader: Loads the game assets.

2.2.5 Local Network Component

Local Network component manages the hosting for game lobbies/sessions on a local network. Since it will use P2P structure, there will be both sending and receiving tasks to/from a connected host.

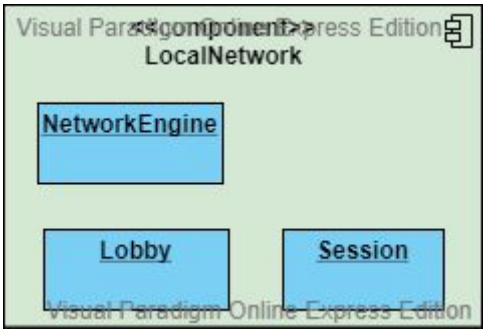

Figure 12: Local network component and its classes

Lobby: A class to represent lobby information of a host. It should contain users connected to the host.
Session: A class to represent session information of a host. It should update the state of content according with changes on the host.
Network Engine: A manager class to handle connection to the host. It executes sending and receiving tasks through the connection.

3 Class Interfaces

3.1 Desktop Subsystem

3.1.1 Model

class Project	
A class to represent a custom game project.	
Attributes	
public String name	
Methods	

3.1.1.1 Instancing

class InstanceType	
A class to represent a type to define game objects.	
Attributes	
public String ID public InstanceType genericType public InstanceType superiorType	
Methods	
public boolean equals(InstanceType other) public String toString()	returns whether two types are equivalent returns string representation of the type

class Instance	
A class to represent a specific game object like card, player, etc.	
Attributes	
public InstanceType type private Instance superiorInstance private List<Attribute> attributes private List<Event> events private List<EventTrigger> triggers	
Methods	
public Instance getSuperiorInstance()	returns superior instance
public boolean insertAttribute (Attribute attribute)	inserts given attribute
public boolean removeAttribute (Attribute attribute)	removes specified attribute
public boolean setAttributeInit (Attribute attribute, Instance init)	sets the init value of specified attribute with given parameters
public List<Attribute> getSelfAttributes()	returns a list for the attributes self defined
public List<Attribute> getInheritedAttributes()	returns a list for the attributes inherited
public Instance copy()	copies and returns the instance
public Instance instantiate()	instantiates an Instance derived from the instance
public boolean checkEquivelant(Instance other)	checks whether two instances are equivalent

class Attribute	
A class to represent a property of defined **Instance.** It stores its reference name, type and initial value.	
Attributes	
public String ID public Pair<DataType, InstanceType> type private Pair<InitType, Instance> init	
Methods	
public boolean setInit(InitType, Instance init) public Pair<InitType, Instance>getInit() public Attribute copy()	sets the init value with given parameters returns the init value copies and returns the attribute

enum DataType
An enum to define data types applicable on **Attributes**
Constants
NUMERIC STRING BOOLEAN CUSTOM LIST

enum InitType
An enum to define initialization types applicable on **Attributes**
Constants
UNINITIALIZED DEFAULT VALUE

class InitValue	
A child class of Instance to represent a primitive type of initialization.	
Attributes	
public String value	
Methods	
public InitValue copy()	returns a copy of the init value

enum FixComp
An enum to represent constant components in games
Constants
GAME PLAYER CARD

3.1.1.2 Event

class Event	
A class to represent specific game events like game-init, card-attack, etc. It stores all its own **Triggers** defined.	
Attributes	
public String ID public EventArgs args public Instance invoker private Map<Instance, Trigger> triggers	
Methods	
public boolean insertTrigger(Instance instance, Trigger trigger) public boolean removeTrigger(Instance instance)	inserts and matches the given trigger with specified Instance removes the trigger of specified Instance

class EventArgs	
A class to represent arguments properties for Events.	
Attributes	
private List<Attribute> args	
Methods	
public boolean insertArgument(Attribute attribute) public boolean removeAttribute(Attribute attribute)	inserts the given argument attribute removes the specified argument attribute

class EventTrigger	
A class to represent trigger mechanisms for Events.	
Attributes	
public Event event private List<Script> scripts	
Methods	
public void insertScript(Script script) public boolean removeScript(Script script)	inserts the given script removes the specified script

class Script	
A class to represent the general structure of **Event** scripts.	
Attributes	
public Expression attributeExpression public EffectType effect public Expression modifierExpression	
Methods	

class Expression	
An abstract class to represent in-script variables.	
Attributes	
Methods	

enum EffectType
An enum to represent in-script effect types.
Constants
SET ADD SUBTRACT MULTIPLY DIVIDE INSERT REMOVE EXCHANGE TRUNCATE RAISE CHECK ITERATE INVOKE

3.1.2 View

class EventCreationScene	
A class which provides the view for event creation for created games. UI elements to create/edit/delete are defined here.	
Attributes	
private TreeView<String> hierarchyView private VBox eventList private StackPane attributes private Pane UpperPane private Pane bottomPane	
Methods	
public void initialize()	Sets the listeners and variables.
ItemLabelView setTree()	Creates the card navigation hierarchy.
Button createEventView(GameInstance instance)	Creates a event view.
private void setTypeAttributeView(List<Attribute> attributes, VBox attribView, boolean inherited)	Used to create the view of an attribute of a type.
private void setEventAttributeView(List<Attribute> attributes, VBox attribView)	Used to create the view of an attribute of a event.

!!!1!!!!!!!!!BULLSHITTING MOMENTUM!!!!!!!!!!!!!!11!

class CardCreationScene	
A class which provides the view for card creation for created games. UI elements to create/edit/delete are defined here.	
Attributes	
private TreeView<String> hierarchyView private VBox cardList private StackPane attributes private Pane UpperPane private Pane bottomPane	
Methods	
public void initialize()	Sets the listeners and variables.
ItemLabelView setTree()	Creates the card navigation hierarchy.
Button createCardView(GameInstance instance)	Creates a card view.
private void setTypeAttributeView(List<Attribute> attributes, VBox attribView, boolean inherited)	Used to create the view of an attribute of a type.
private void setCardAttributeView(List<Attribute> attributes, VBox attribView)	Used to create the view of an attribute of a card.

class LayoutSettingScene	
A class which provides the view for setting the game layout. Game creators set how the table areas will be used/divided in the game. They also set the game button/control layout.	
Attributes	
private TreeView<String> hierarchyView private VBox toolBox private StackPane gameTableView private StackPane phoneLayoutView private Pane upperPane private Pane bottomPane	
Methods	
public void initialize() void setToolBox() void setGameTableView() void setPhoneLayoutView()	Sets the listeners and variables. Creates a toolbox view with different tools to manipulate the game layout. Creates game table view. Creates phone layout view.

class MenuScene	
A class which provides the view for the main menu. UI elements to go to card creation,settings,etc. are here.	
Attributes	
private VBox menuBox private UpperPane upper private StackPane	
Methods	
private void initialize() private void createContent() private void addBackGround() private void addTitle() private void addMenu() private void openInfo()	 adds all the components to the screen adds the background image adds the title adds the menu buttons opens the info section

class UpperPane	
A class which provides a quick menubar to quickly switch between important scenes like homescene,settings,etc.	
Attributes	
private Rectangle backToMenu private Rectangle musicButton private VBox rectangles	
Methods	
public void initialize()	creates the screen

3.1.3 Controller

class InstanceManager	
A Singleton manager class to handle instance modelling regarding incoming requests from the **View** component.	
Attributes	
private List<Instance> roots	
Methods	
public void initialize()	initializes fix components
public Instance insertInstance(String id, Instance superior)	creates and return new instance with given parameters
public boolean removeInstance(Instance instance)	removes the specified instance
public List<Instance> retrieveSuperInstances()	returns a list for super instances
public List<Instance> retrieveSubInstances(Instance instance)	returns a list for sub-instances of specified instance
public boolean insertAttribute (Instance instance, String attributeID)	creates and insert new attribute with given parameters
public boolean removeAttribute (Instance instance, Attribute attribute)	removes the specified attribute
public boolean setInit(Instance instance, Attribute attribute, Instance init)	set the init value of specified attribute with given parameters

class EventManager	
A Singleton manager class to handle event modelling regarding incoming requests from the **View** component.	
Attributes	
private List<Instance> roots	
Methods	
public void initialize()	initializes fix events
public boolean insertEvent(Instance instance, String eventID)	inserts the specified event into the given instance
public boolean removeEvent(Instance instance, Event event)	removes the specified event
public boolean insertEventTrigger(Event event, Instance instance, EventTrigger trigger)	inserts the specified event trigger into the given event
public boolean removeEventTrigger(Event event, EventTrigger trigger)	removes the specified event trigger
public void insertScript(EventTrigger trigger, Script script)	inserts the specified script into the given trigger
public void removeScript(EventTrigger trigger, Script script)	removes the specified script

3.2 Mobile Subsystem

3.2.1 View

class MainWindow	
A class which provides the view of the main window of AugCards.	
Attributes	
private XML mainWindowXML	
Methods	
public void start()	displays the main view of AugCards

class HomeView	
A class that is responsible for the display of the home page.	
Attributes	
private XML homeViewXML	
Methods	
public void startHomeView()	displays the home view of AugCards

class LibraryView	
A class which provides the library view of AugCards which displays a number of games.	
Attributes	
private XML libraryViewXML	
Methods	
public void startLibraryView()	displays the library view of AugCards

class PlatformView	
A class that is to display the platform where users can search for games.	
Attributes	
private XML platformViewXML	
Methods	
public void startPlatformView()	displays the platform view of AugCards

class GameView	
A class that represents the view of the game to be played in AugCards.	
Attributes	
private XML gameViewXML	
Methods	
public void startGameView()	displays the game view of AugCards

class GameLayoutView	
A class to represent the layout of where each game instance will be placed for a game.	
Attributes	
private XML gameLayoutViewXML	
Methods	
public void startGameLayoutView()	displays the game layout view of AugCards

3.2.2 Model

class Augcards	
The mainframe of the mobile game.	
Attributes	
private GameLobby gameLobby private User user private Platform platform	
Methods	
public GameLobby getGameLobby() public User getUser() public Platform getPlatform()	returns game lobby returns user returns platform

class GameLobby	
Information about the running game lobby is kept here.	
Attributes	
private List<String> usersInLobby private List<String> usersInvited private String host private boolean isGameReady	
Methods	
public List<String> getUsersInLobby() public List<String> getUsersInvited() public String getHost() public Boolean isGameReady()	returns users in Lobby returns users who are invited returns host of Lobby returns if game is ready to be played

class User	
Information about the user themselves will be managed by this class.	
Attributes	
private String username private GameLibrary gameLibrary	
Methods	
public String getUsername()	returns user's username

class Platform	
The shared games and their community pages will be kept here.	
Attributes	
private List<String> allGames private List<String> featuredGames	
Methods	
public String getGame(String id) public String getFeaturedGame(String id)	returns a game with the given ID sets the no returns a featured game with the given ID

class GameLibrary	
The user's game library information will be kept here.	
Attributes	
private List<String> gameIDs private List<String> favGameIDs	
Methods	
public String getGameID(String name)	returns the ID of the game by given name
public List<String> getFavGames()	returns the user's favourite games

3.2.3 Controller

class AuthenticationManager	
A manager class to handle authentication requests. It should communicate with **Cloud Subsystem** to check users' access permission.	
Attributes	
private String authenticationToken	
Methods	
public boolean authenticate(String token)	returns if user is authenticated or not

class LobbyManager	
A manager class to handle lobby joining/creation requests. It should communicate with **Local Network Component** to establish connection with a lobby.	
Attributes	
private GameLobby lobby	
Methods	
private GameLobby createLobby() private String inviteUser(String id) private String kickUser(String id)	creates lobby invites users to lobby kicks users from lobby

class GameSessionManager	
A manager class to handle game launch requests. It should communicate with **Game Component** to run a game execution process.	
Attributes	
private Game game	
Methods	
private boolean startGame()	starts the game

3.2.4 Game

class GameEngine	
Executes the game using the game model and game assets.	
Attributes	
private AssetLoader assetLoader private ModelLoader modelLoader	
Methods	
public boolean executeGame()	executes game

class ModelLoader	
Loads the game model data for the game.	
Attributes	
private List<Files> jsonFiles	
Methods	
public List<Files> getJSONFiles	returns the JSON files of the game

class AssetLoader	
Loads the game assets.	
Attributes	
private List<Asset> assets	
Methods	
public List<Asset> getAssets()	returns the assets

3.2.5 Local Network

class NetworkEngine	
A manager class to handle connection to the host. It executes sending and receiving tasks through the connection.	
Attributes	
private Session session	
Methods	
public connectToSession(session)	establishes connection to session

class Lobby	
A class to represent lobby information of a host. It should contain users connected to the host.	
Attributes	
private GameLobby gamelobby	
Methods	
public GameLobby getGameLobby()	returns game lobby

class Session	
A class to represent session information of a host. It should update the state of content according with changes on the host.	
Attributes	
public String hostIP public String hostPort	
Methods	
public String getHostIP() public String getHostPort()	returns the host's IP address returns the host's port number

4 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].

Firebase Firebase is Google's mobile platform that helps you quickly develop high-quality apps and grow your business [4].

JSON JSON (JavaScript Object Notation) is a lightweight data-interchange format [12].

5 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] "ARCore overview | Google Developers," *Google*. [Online]. Available: https://developers.google.com/ar/discover. [Accessed: 27-Dec-2020].

[4] *Google*. [Online]. Available: https://firebase.google.com/. [Accessed: 27-Dec-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]

[12] "Introducing JSON," *JSON*. [Online]. Available: http://www.json.org/json-en.html. [Accessed: 27-Dec-2020].