Bilkent University
Department Of Computer Engineering

Senior Design Project

Project short-name: AugCards

Final Report

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

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

Final Report
April 30, 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.

Contents

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.

AugCards is a multiplayer AR mobile game creation engine. 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. AugCards gives users the freedom to create their own cards with their own assets, and introduce their own animations. Similar applications like Dulst [2] and Runeterra [3] do not support specifying their own game rules, which is AugCards' key functionality.

Imagine you and your friends sitting around a table and want to have a good time. AugCards 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 models, rules. Even the complex rules can simply be introduced with the help of user-friendly design. 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 3d models of the cards are seen on the table by everyone looking through the cam of AugCards.

In this report, we will provide the final architecture and design of AugCards. First, development and implementation details will be discussed. Then, testing details will be described, given attention to the maintenance plan. Requirements being functional and non-functional will be provided. Also, other project elements will be provided, such as consideration of various factors in engineering, ethics, and teamwork details. After, the new knowledge acquired during this project will be discussed. Then a user manual of AugCards will be provided. Last but not least, the future work regarding the project will be stated as a conclusion.

2. Requirements Details

2.1. Functional Requirements

Functional requirements aid in capturing the system behavior in terms of functions, systems and services. User functionality, Desktop system, Mobile system and Cloud system requirements will be provided.

2.1.1 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.

2.1.1.1 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.

2.1.1.2 Game-Developer Functionality

The system should:

- 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.

2.1.2 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.

2.1.3 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.

2.1.3.1 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.

2.1.4 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.

2.2. Nonfunctional Requirements

Nonfunctional requirements, which define the quality of the system will be discussed. AugCards' requirements include usability, reliability, maintainability, accessibility,extendability and portability.

2.2.1 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.

2.2.2 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.

2.2.3 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.

2.2.4 Accessibility

- Should be free to download.
- Should have integration with Google Services.
- Should require less than 1GB of RAM.
- Project should be licensed with the MIT License since it provides the most flexibility and will be less likely to lead to legal problems we may come across in the future [4].
- Should utilize GPU usage since AR will highly consume memory and load highly on the GPU. This will make it accessible devices with low cpu power [5].

2.2.5 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.

2.2.6 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. Final Architecture and Design Details

We had separate development progress for Desktop System and Mobile System. We implemented the desktop system in Java language to ensure multi-platform features. The UI design of desktop system developed via FXML and CSS supported JavaFX. In data transmission from Desktop to Mobile, we have decided to use JSON files to store game models along with other resources. It eliminated any language dependencies in the development progress of Mobile System. On the Mobile side, we implemented AR features by using Vuforia Engine supported via Unity Engine. Other mobile features developed via Unity Engine which uses C# programming language. In cloud system, we used Firebase cloud services for storing 3D models and JSON files used in the game, they are downloaded by the mobile system in the runtime.

3.1 Overview

We will describe the architecture and design details under three categories as Desktop, Mobile and Cloud systems in Figure 1.

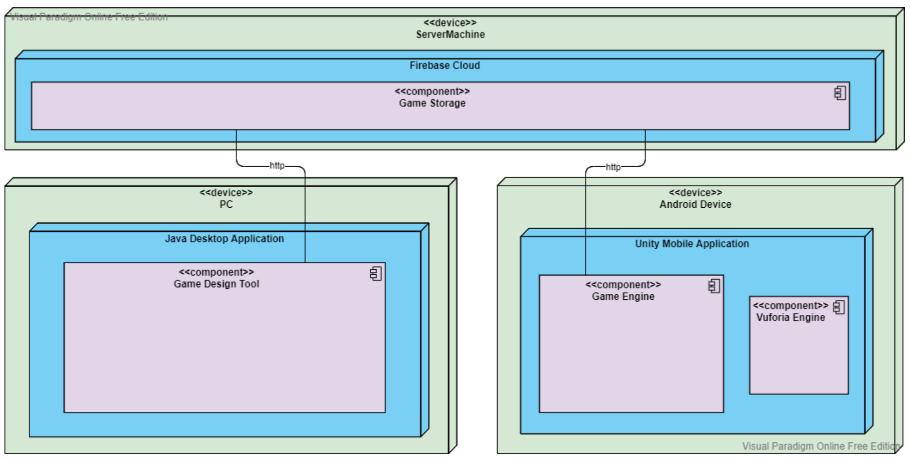


Figure 1. Overview of Systems

3.2 Desktop System

In desktop, we used MVC software architecture to implement the system. Using Java, we decided to make the development in the latest version as Java 15.0.2. Using MVC, it is easy to distribute system functionality in separate modules. While model modules defining the relations among the data, view modules take the role of handling display and user inputs and controller modules perform required manipulations on data regarding user requests. In view modules, we defined view class and their properties by using FXML and defined their styles via CSS. Then, we easily used these defined view classes to design the view of the desktop system. In the data output phase, we used JSON file format to store game models and a custom library named as JSON-simple to handle parsing/formatting purposes.

3.3 Mobile System

In the mobile system, we used Unity Game engine for game rendering tasks. Unity has an Android SDK for building games for building high engaging 3D games in Android platform. We decided to use the Unity engine for easier integration of Vuforia in Android platforms. Unity is very popular and easy to use for deploying generated games in many platforms.
We also used the Vuforia Engine for the Augmented Reality implementation on the mobile system. Vuforia Engine has Android SDK where mobile apps can be built using Unity. Vuforia Engine eases the functions such as recognizing images and rendering 3D models in real spaces.

3.4 Cloud System

In the cloud system, we used Firebase which is a Cloud Storage built by Google. Firebase has a Unity SDK which made us integrate it with our Unity project directly, easily and securely. Through Firebase, we stored the 3D models of type DAE which were inputted by the user. We also stored JSON files which specify game instance types, model-card association and event scripts which were generated by the Desktop system. Unity project which runs on Android device downloads and makes use of these files from Firebase Storage service.

4. Development/Implementation Details

4.1. Desktop Application

Through the development of desktop system, our main focus was to provide maximum usability. That’s why we tried to make game design as easy as possible. We came up with a simple card and game modelling which contains all essentials but any complex options. The essentials is only to determine attributes, attribute values and specialization relationships. In game event design, it is enough to determine modifications on game objects and eventflow among the game. For this purpose, we developed a script based event design for practicality and usability. To ensure usability, we should provide an object oriented event design in which each event is defined over a card, player, game, etc. For this manner, in event scope, users can refer to attributes of an object to define modification and check rules over them, also, users can refer to events to raise them. The details for the required script types given in Table 1.

Scripts / Accessibles	Modify Script	Condition Script	Call Script	Wait Script
Numeric	SET, ADD, SUBTRACT, DIVIDE, MULTIPLY	CHECK, ITERATE	-	-
Text	SET	CHECK	-	-
Boolean	SET, INVERT	CHECK	-	-
Custom	SET, CLONE, EXCHANGE	CHECK, TYPEOF	-	-
List	INSERT, REMOVE, SHUFFLE, TRUNCATE,	CHECK, ITERATE	-	-
GameEvent	-	-	RAISE	WAIT

Table 1: Applicable Types on Accessibles under specific Script Type

For clarity, there are four types of scripts as Modify, Condition, Call and Wait. As their name suggest, each has different effect; modify script to define modification on attributes, condition script to determine a check rule and its script branches, call script to raise an event by specifying event args and wait script to define an wait for user input to hang up script execution. Scripts are composed of two main components; the first is focus attribute or event, the second is further specification of script type like SET, ADD, CHECK, RAISE, etc. Some scripts require additional expressions to provide all their functionality. Such expressions are given in Table 2.

In the game design, another important thing is to provide customizability for visual models of cards. We provided a simple model binding for defined cards. The overall implementation of the desktop system is within such order.

Expressions	Expressions to support scripting functionalities
Reference	Expression to refer an accessible attributes or events
List Query	Expression to define a query over list attributes
Init	Expression to initialize a new value
Arithmetic	Expression to perform arithmetic operations
Logic	Expression to perform logic operations
Arithmetic Condition	Expression to define arithmetical conditions
Text Condition	Expression to define textual conditions
Logic Condition	Expression to define logical conditions
Custom Type Condition	Expression to define custom-type related conditions
List Condition	Expression to define list related conditions
Event Call	Expression to define an event raising by specifying required event args

Table 2: Expression Types used along with Scripts

4.2. Mobile Application

Mobile application is implemented as a Unity project where Unity is an engine that has Android SDK which is widely used for creating games that require high user integration. Because the 3D models are added and used in Unity, we thought using Vuforia SDK was a precise choice. Unity uses C# scripts for modeling Game Object behaviors and other implementations.

In the implementation stage, 2 members were selected to be responsible for learning Vuforia and AR technologies. These members started the implementation from an empty Unity Vuforia integrated app. They started by using trial and error to implement a wanted UI integrated AR behavior in a dynamic approach. First, we tried to implement AR without depending on any image target, then we changed our mind regarding the ease and reliability using markers/image targets would give us. We started by implementing by associating each 3D model with a separate image target. Later in the implementation, we thought having separate image targets would decrease the usability of the project as players would have to get a hardcopy print for every game they would want to play. Hence, we changed our implementation such that we had a specific generic marker/image target which indicated the middle area of the game board and the 3D models of each player were rendered with respect to the marker location.

We introduced a default cube model for cards with unspecified 3D models. Additionally, each 3D model has a canvas as their child, which displays the name of the card the 3D model represents and the card attributes to be visible by all players when the card is played.

Each 3D model is downloaded from Firebase Storage and each 3D model is associated with specified cards which are read from a models.JSON file from Firebase Storage. Then, in runtime when a game instance is played, the model associated with the card is instantiated as a child of the marker. For implementing clicking on 3D models, cube colliders are used. The name of each model is encoded with the associated card ID for easing the referring.

During implementation the team used Unity Collab which is a cloud hosted environment that enables small teams to sync on a Unity project. This way, we could collaborate on the Unity project without having to depend on an external version control system.

4.3. Mobile Application Network

The mobile application uses the Unity UNet framework to handle the server-client logic of the game. The UNet framework is a framework that allows developers to integrate multiplayer capabilities into their game with ease. The network logic of the game is as follows: When a player hosts a game on their mobile device, the framework creates both a server and a local client on their machine as in Figure 2. This client is created so that all players, whether they be local or remote, can be treated the same way, reducing overhead function calls. Whenever a new client joins, they are provided with a unique connection to the network, a unique player object over which they have authority and networked objects that need to be synchronized across the network.

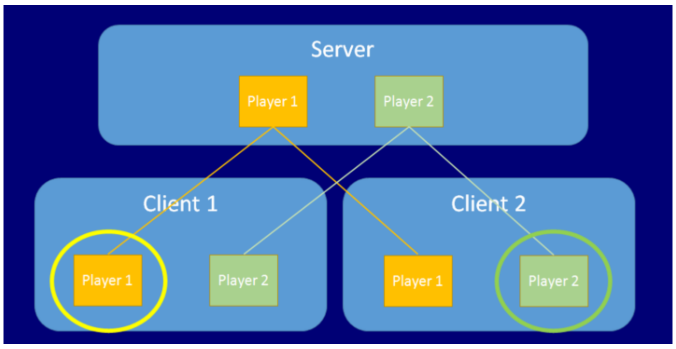


Figure 2: The network authority states in UNet.

The server and clients can communicate over either ‘Command’ and ‘RPC’ calls or network messages. Commands are called on the server by the clients to run certain events, and RPCs are similarly called on the clients by a server. The network message approach allows for fine-tuning of what is sent over the network, thus we chose to implement the network communication using custom network messages and a custom network manager, which handles different types of messages and makes relevant function calls to relevant classes.

5. Testing Details

Testing is of great importance to enable a smooth gameplay for our users. The testing details executed during the implementation of AugCards are provided in this section.

5.1. Continuous Integration

As a team we picked up the Software Engineering principles into consideration by using the Continuous Integration principle. In development, every developer integrated their code which has no errors, into a main branch frequently. We wrote test code with high line coverage in order to ensure that each pushed code to the main branch is free of errors.

5.2. User Groups

As AugCards is an engine, testing using a user group was vital in order to ensure that the engine is as user-friendly as we intended it to be. Also, to fine-tune any variability points, user feedback was of great importance. Hence, we formed a user group consisting of gamers and game development enthusiasts. Each user created a card game using the AugCards engine and gave the development team feedback and reported bugs.

6. Maintenance Plan and Details

Maintenance is an important aspect of AugCards regarding It is server dependent. Additionally, the libraries need to be kept up to date due to the fact that they might get deprecated.

6.1. Server Maintenance

AugCards uses firebase cloud for models and JSON files to be transported which form the game the user has created. Free version of Firebase Storage was used in this project, however in the maintenance process rental services need to be payed in each term. Also, depending on the user traffic, the team might have to switch to a faster responsive service.

6.2. Google Play Store

As a team one of our goals is to upload AugCards to Google Play Store after we get necessary feedback on our product and implement the changes. Users agreements will be written in order to send It to the reviewing authorities in the future.

6.3. New Versions

New versions which involve UI and variability point changes can be frequently released. The new versions of AugCards will get closer each time to become a fully developed engine.

6.4. Issue Tracking

AugCards has an open source GitHub repository which is public. Issues posted by the users will be frequently addressed by our team. This way, AugCards will become a better product with less bugs as It is used.

7. Other Project Elements

7.1. Consideration of Various Factors in Engineering Design

Various factors were considered while designing our project. The factors are scored on a scale of 0-10, taking how much they might be affected by our project into consideration. The scores can be found in Table 3 below. The reasonings for the scores and more details can be found in the respective subsections.

Factor	Score (0-10; 0 not affected, 10 very crucial)
Public Health	1
Public Safety	6
Public Welfare	5
Global	4
Social	7
Cultural	6
Economic	6

Table 3: How our project considers some factors

7.1.1 Public Health

We do not believe that public health will be very much affected by our application, since our application is just a way for users to create and play games with their friends. The users' mental health may improve by socialising, and that's why 1 is given as a score.

7.1.2 Public Safety

Since the app offers a variety of options to users, users may use this variety to create games, images or characters that may include offensive figures to others, which might possibly threaten public safety. In order to prevent this, AugCards will have a way of filtering the media, game style and other components that may be a potential threat to the users. Public Safety and ways to protect it must be one of our priorities, that's why 6 is given as a score.

7.1.3 Public Welfare

We believe the welfare of society will be affected by our game since we will provide a way for people to enjoy themselves and spend leisure time with their friends. This will lead to people becoming happier and releasing stress. However, welfare is not only about happiness, so 5 is given as a score.

7.1.4 Global Factors

Since our game will allow users from all around the world to create and share games, we believe our application will have a global impact. However, the card game community is not a very large niche, so we do not expect a ground-breaking impact and that's why 4 is given as a score.

7.1.5 Social Factors

We believe people will be able to socialise using our application and they will be generally happier and more social. Each game represents a community through a platform page in which players interact with the game and others. Also, custom games on AugCards application are AR featured card games and are supposed to be played with a group of people around a table, which is another socializing factor. That's why 7 is given as a score.

7.1.6 Cultural Factors

Our application will allow users from different countries to create and share games with each other. We believe that each game will have cultural influences from the developer and when a foreigner plays their game, they will be exposed to a new culture. This will lead to a cultural exchange between different countries which is why 6 was given as a score.

7.1.7 Economic Factors

Card game creating companies that sell printed cards come to mind when talking about factors, and we need to consider their revenues as well when we are creating our application. The card game developers may also need to pay for the work that they do, which also needs to be considered, so economic factors are given a score of 6.

7.2. Ethics and Professional Responsibilities

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

7.2.1 Ethics

Regarding the **global impact**, the goal of AugCards is to be an engine such that everyone can create their card games. AugCards will have a global impact on the card game making process. Regarding the impact in **societal** context, 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 is also prioritised, since we need to keep data generated by users. The data will only be kept for the purpose of enabling the player to play, and not be shared with third parties.

Regarding the **economic impact**, AugCards is free but we may introduce advertisements in the app for monetizing in the future to generate profit. The economic constraints we faced included the cost of Firebase and 3D models to be used. The softwares we used was in free mode.
The **environmental impact** of AugCards was related to the electricity consumption which was insignificantly small.

7.2.2 Professional Responsibilities

We communicated through online meetings and messaging, minding the pandemic. We conducted weekly meetings through Discord, an online communication application [7]. For important decisions and changes, we used WhatsApp to communicate.We kept our GitHub repository private for the time being since it would benefit us in terms of security.

7.3. Judgements and Impacts to Various Contexts

The impact levels in Table 4 are out of 10.

	Impact Level	**Impact**
Impact in Societal Context	8	Variety of card games in various contexts can be generated with this engine which is risky in socitaly.
Impact in Economic Context	1	Costs of using cloud and costs of using 3D models restrict usability.
Impact in Environmental Context	2	Consumption of electricity when the engine is used is small.
Impact in Global Context	6	We generated a tool that eases the card game creating process.

Table 4: Impacts of judgement

7.4. Teamwork Details

In this section, each member's contributions to the project will be explained

.

7.4.1. Contributing and functioning effectively on the team

Member	**Contributions**
Çerağ	Worked in the implementation of the AR, system and cloud system.
Yusuf	Worked on designing the event generation and parsing logic, and related GUI. Wrote the event running logic for the mobile component.
Yiğit	Worked in the implementation of the AR system, mobile system and network system.
Burak	Worked on backend development of Desktop application, specifically, custom game concepts and instance design. Worked on designing the event generation and parsing logic.Drew UML diagrams [8].
Bora	Worked in the UI and front-end of desktop and mobile. Implemented the initial mobile application as a starting point.

Table 5: Contributions of each member

7.4.2. Helping creating a collaborative and inclusive environment

Member	Contributions
Çerağ	Helped set up meetings and resolved most of the conflicts among group members.
Yusuf	Helped setting up a project management software Trello [6]. Suggested enforcing code reviews.
Yiğit	Set up the GitHub repositories and Google Drive folder.
Burak	Participated in group meetings and other scheduled works.
Bora	Created a written to do list & summary after every meeting.

Table 6: How each member helped create a collaborative and inclusive environment

7.4.3. Taking lead role and sharing leadership on the team

Member	Contributions
Çerağ	Participated in the group's thought process.
Yusuf	Took an important part in the decisions regarding the implementation. Took lead in coordinating desktop and mobile systems.
Yiğit	Took part in the decision making process and dividing tasks.
Burak	Contributed into the decision making process.
Bora	Made sure that everyone took equal responsibility.

Table 7: How each member took leadership roles

7.5. Meeting objectives

This part of the report discusses how the product met the objectives stated in the Requirements part of the report.

7.5.1. Functional Requirements

Functional requirements aid in capturing the system behavior in terms of functions, systems and services. User functionality, Desktop system, Mobile system and Cloud system requirements will be provided.

7.5.1.1. User Functionality

- Users are either game-developers or players.

7.5.1.2. 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 with respect to the marker.
- 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.

7.5.1.3. Game-Developer Functionality

The system can:

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

By using UI option of AugCards engine, developers are 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).
- hierarchical 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.
- custom graphical models/assets for corresponding game instances.
- Developers can upload/update the models for their own created games on the cloud.

7.5.1.4. Desktop System Functionality

The system can:

- create local/cloud repositories for custom game projects.
- delete all repository files if the developer decides to remove the game.

- restore a custom game from their cloud/local repository.
- provide a sophisticated and interactive UI for each model design feature.
- provide relationships such as defining 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 scripting tool for events-triggers; definition of triggers involves check rules, updates for accessible properties, and raising other events.
- provide import and bind graphical models to game instances and events.

7.5.1.5. Mobile System Functionality

The system can:

- 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.

7.5.1.6. Common Graphics/Network Subsystem

The system can:

- 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 which highly impacts ar quality [7].
- support advanced graphics effects like light-estimation, shadows, anti-aliasing, etc.

7.5.1.7. Cloud System Functionality

The system can:

- store the 3D game models.
- store json files regarding the events created by the developer.
- store json files regarding the game instances created by the developer.

7.5.2. Nonfunctional Requirements

Nonfunctional requirements, which define the quality of the system will be discussed. AugCards’ requirements include usability, reliability, maintainability, accessibility,extendability and potability.

7.5.2.1. Usability

- Game creation does not require programming knowledge.
- Tools are self-explanatory, and do not require extensive tutorials or guides to be understood.
- The expression of complex card game rules are simplified using event-based scripts.
- AR-based graphics have a refresh rate of at least 25 Hz to not affect game experience adversely.
- Popular graphical model/animation formats like DAE and COLLADA are supported.

7.5.2.2. Reliability

- Ensures that changes in game models are not lost on connection errors.
- Has a back-up mechanism for ongoing games in a network failure situation.
- Contradictory game rules aren’t allowed to cause errors.

7.5.2.3. Maintainability

- Is modular to reduce the complexity of the codebase.
- Network maintenance costs are lower than 50TL per month while profits are low.

7.5.2.4. Accessibility

- Is free to download.
- Requires less than 1GB of RAM.

7.5.2.5. Extendability

- The addition of new possible game mechanics does not require changing existing code.

- The code itself is properly structured,using design patterns and clever modularity.Adding new features and mechanics requires minimal or zero amount of change in the code.

7.5.2.6. Portability

- Does not cause any compatibility error with changing Android device sizes and camera resolutions.
- The game creation tool runs on Windows.

7.6. New Knowledge Acquired and Applied

It was not possible for the team to develop AugCards with the knowledge we already had, none of us were familiar with AR technologies, Unity and game development beforehand. Hence, we did learning on the topics of:

- Vuforia Library
- Unity Engine
- Firebase Cloud
- Android Development
- Networking Systems

Developing an engine required the use of gaming technologies which were not covered in Bilkent's lectures. However, in scripting the event processes from the game, Bilkent's lectures were helpful. For self-learning we utilized gray literature review, online material and trial-error. Gray literature and online materials were very helpful as they provided tutorials on gaming technologies. However, the best way to learn was trial and error, as we could learn from our mistakes.

8. Conclusion and Future Work

8.1. Conclusion

All in all, we succeeded in building the multiplayer AR mobile game creation engine that we aimed for at the start of senior year. AugCards is an engine where various card games can be generated freely by the users with AR visual support. We are content with what we have managed to achieve. The project consisted of a number of systems and we managed to merge them together as a working engine in a relatively small time. We learned a lot during the design and implementation of the project, we would be happy to get advice on AugCards, anytime.

8.2. Future Work

Our goal is to create a platform for AugCards, where created games can be shared in the platform to be played by many users. We also want to make AugCards downloadable from App Stores after revising AugCards after receiving feedback. We will also extend the engine capabilities and provide more variability points in the game creating phase for users.

9. 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].
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 [7].
WhatsApp	WhatsApp is a messenger cross-platform instant messaging application.
Dulst	Dulst is an online card game playing software [2].
UML	Unified Modeling Language, is a standardized modeling language consisting of an integrated set of diagrams [8].

10. References

[1] J. Clement, “Topic: Mobile gaming market in the U.S.,” *Statista*. [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] Contributors to Wikimedia projects, “Discord (software),” 30-Jan-2016. [Online]. Available: https://en.wikipedia.org/wiki/Discord_(software). [Accessed: 21-Nov-2020]

[8] “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]

User Manual

Introduction

With AugCards, card game designers can implement their card game. The project has a desktop and mobile component. Game creators can create their games in the desktop component. The games run on Android mobile devices. Created games are in JSON format. Games have 3 main components; card manager, event manager and visual manager. In the card component, you can define what exists in the game, event manager is used to write the game logic and visual manager is used to select textures and models for game objects.

Desktop Component

When AugCards is launched, users are welcomed by the screen as in Figure 3.

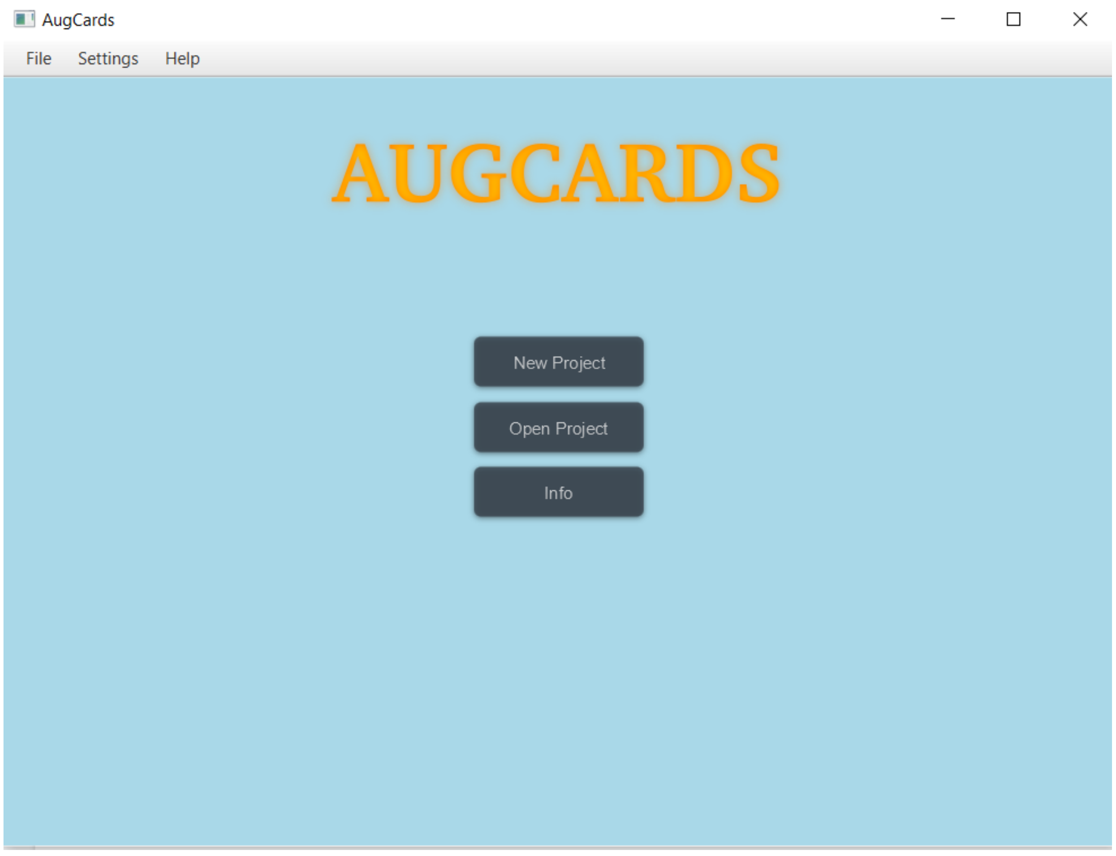


Figure 3. Login Screen

Opening A Project

The options are creating a new project or opening an already existing project.

Opening A New Project

To open a new project, users should click on the ***New Project*** button as in Figure 4.

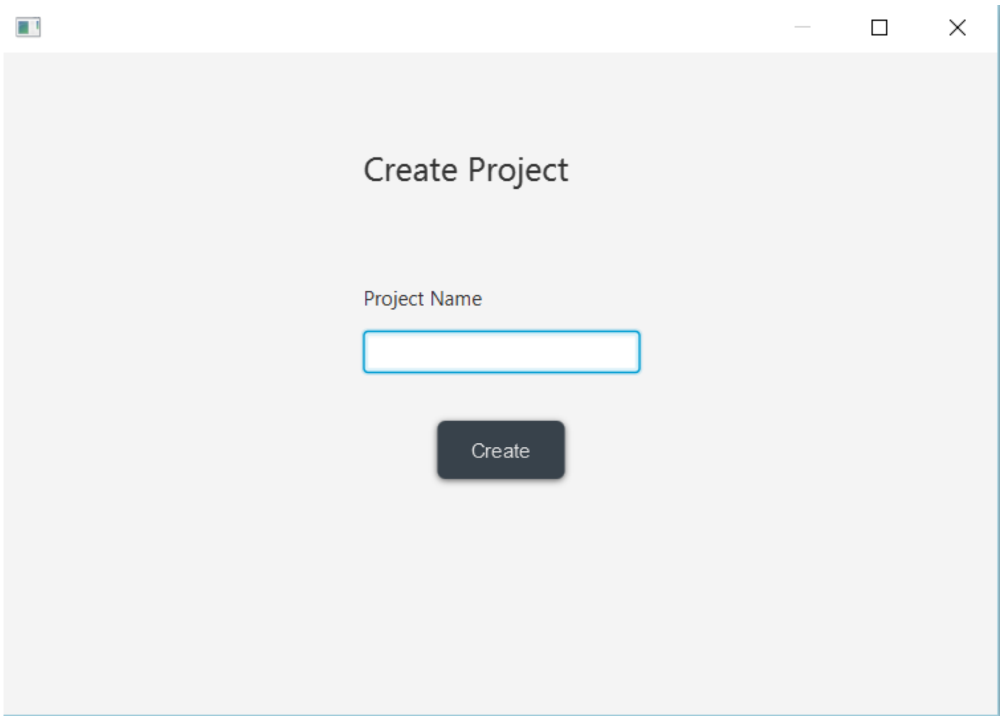


Figure 4. Project Creation

This opens a project creation wizard window in which users choose a project name and click create. Created projects will be saved in ***.../User/Documents/AugCards*** directory.

Loading An Existing Project

To load a project, users should click on the ***Open Project*** button as in Figure 5.

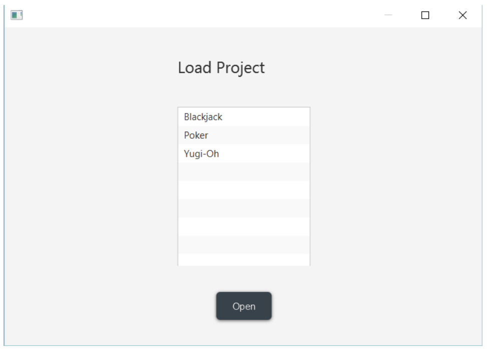
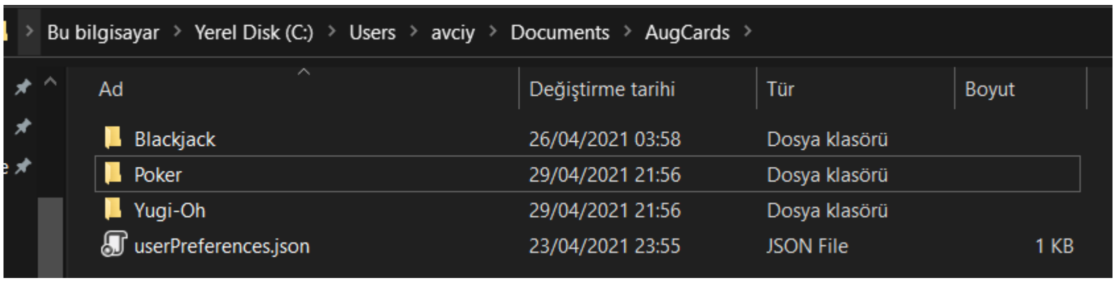


Figure 5. Project Load

This opens a project loading window in which users can choose a project from the list to open. The projects are read from ***.../User/Documents/AugCards*** directory as in Figure 6.


Bu bilgisayar > Yerel Disk (C:) > Users > avciy > Documents > AugCards >

Ad	Değiştirme tarihi	Tür	Boyut
Blackjack	26/04/2021 03:58	Dosya klasörü	
Poker	29/04/2021 21:56	Dosya klasörü	
Yugi-Oh	29/04/2021 21:56	Dosya klasörü	
userPreferences.json	23/04/2021 23:55	JSON File	1 KB

Figure 6. Games

Users can also manually add a valid project to this directory if they backed-up a project.

Basic Navigation

When a new project is created, this screen will be shown as in Figure 7.

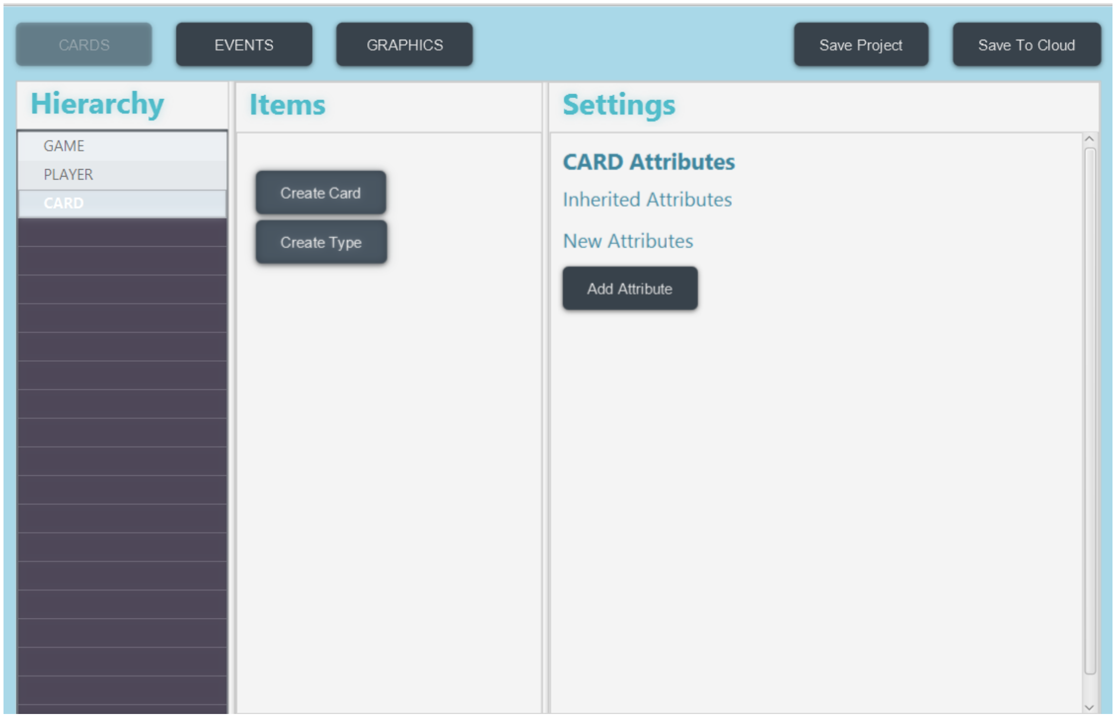
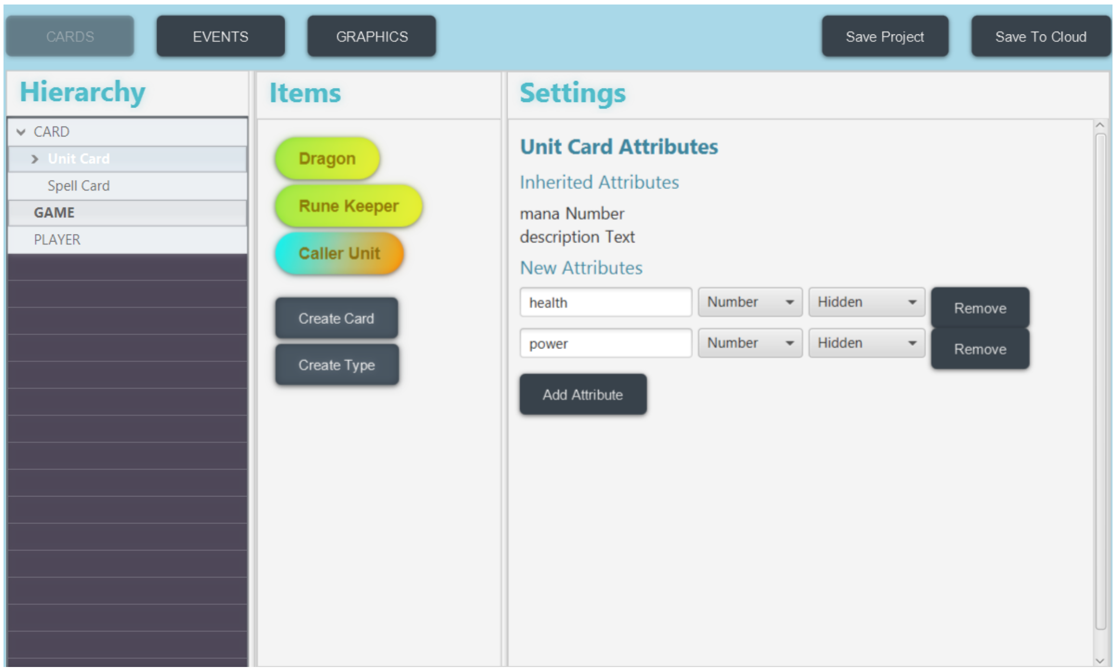


Figure 7. Menu

Games have 3 main components; card manager, event manager and visual manager as in Figure 8.

Figure 8. Bar

To open a different manager, use the top bar.



Unit Card Attributes

Inherited Attributes

mana Number
description Text

New Attributes

health	Number	Hidden	Remove
power	Number	Hidden	Remove

Figure 9. Card manager

A Card manager can look like as in Figure 9 after adding some instances. On the Hierarchy section; Card types are shown. **Click a type** on the hierarchy to **display the items under it**. This will also **show the type attributes on the right side**. Cards are not shown in the hierarchy to keep the hierarchy lighter and simpler.On the Items section, types and cards under the selected type are shown. To indicate types and cards visually; cards are colored ***green*** and subtypes are colored ***blue/orange.*** Click a ***green*** card button to open card attribute values.

Card Manager

In AugCards, a new project comes with 3 predefined constructs. These constructs are GAME, PLAYER and CARD. CARD can contain game cards and/or card types/subtypes. Types can have their subtypes as well. Types define required attributes for cards so that the cards under that type must have a value for these attributes. More detail will be given in the subsequent sections.

Adding An Attribute

When an attribute is added to a type, cards under that subtype should have a value for that attribute as in Figure 10.

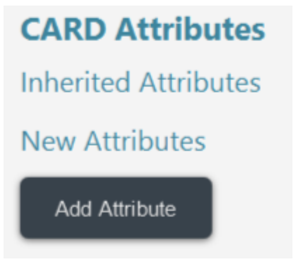


Figure 10. Attributes

To add a new attribute, click on Add Attribute as in Figure 11.

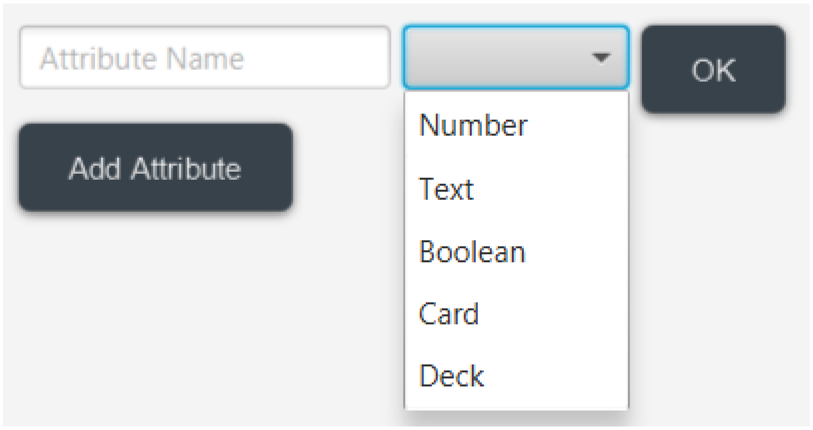
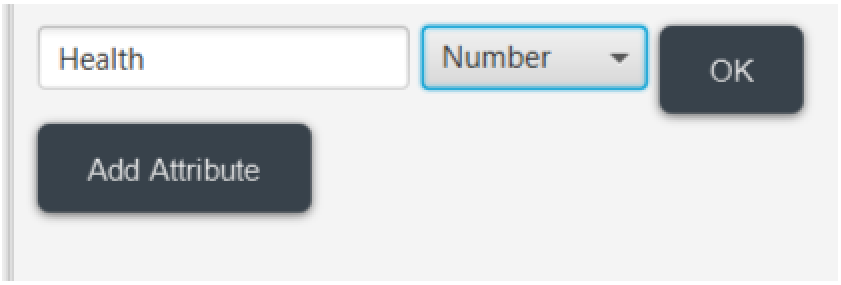


Figure 11. Add Attributes

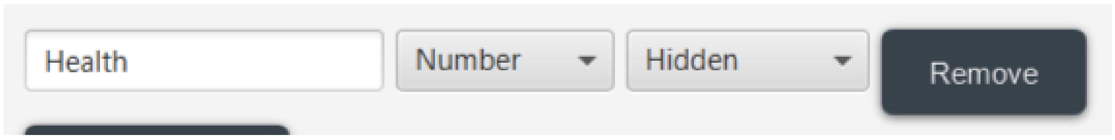
This will bring an attribute field. Here, Attribute Name field should be filled with the attribute's name and a type for the attribute should be selected as in Figure 12.



Health
Number

Figure 12. Attribute Name field

Then click OK. This will create the attribute. Note that filling these fields is not optional as in Figure 13.



Health
Number
Hidden

Figure 13. Attribute Create

When the attribute is created, a **third** field appears. This field specifies whether the attribute should be visible to the players in game. Some attributes may be left hidden so that they are used only for in game events.

Removing Attributes

Click remove to remove the attribute.

Attribute Types

Numeric Attribute	Value is a number. Example use: value attribute for a card.
Text Attribute	Value is a text. Example use: Description for a card.
Boolean Attribute	Value is True or False. Example use: Is card chosen.
Card Attribute	Value is another Card. Example use: Specifying the card type that the card is weak against.
Deck Attribute	Value is a Deck. Example use: When that card is played, the cards in deck are also played.

Table 8: Attribute Types

Adding Card/Deck Attributes

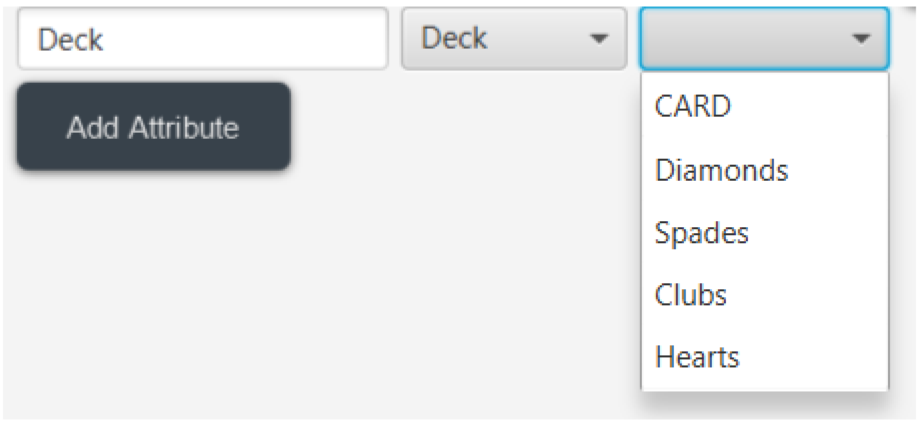
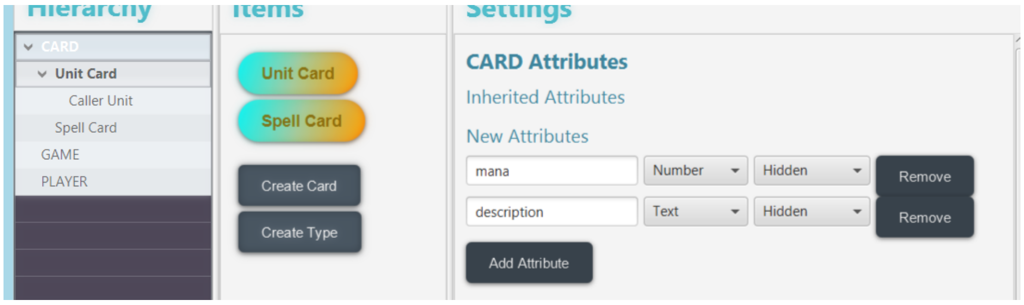


Figure 14. Attribute selection

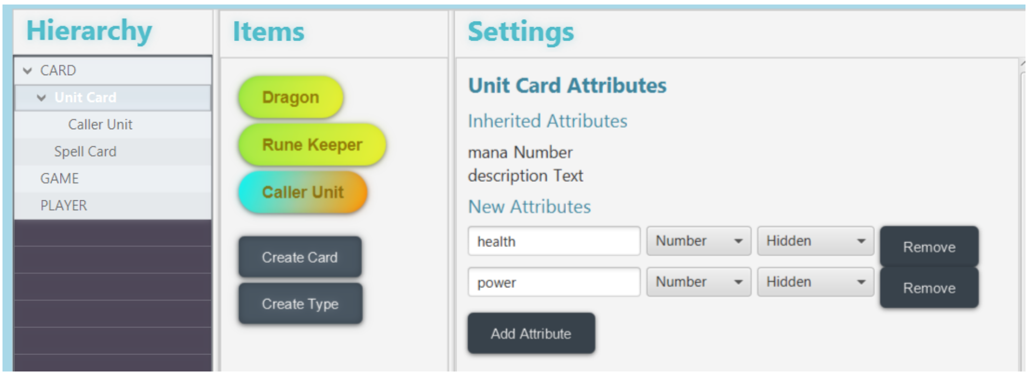
When an attribute of type Deck or Card is created, a new selection appears as in Figure 14. Here, you should specify which types of cards this Deck can contain (or Card attribute can be). The other details are the same as other attributes.

Attribute Inheritance



mana Number Hidden
description Text Hidden

Figure 15. Attribute inheritance example



mana Number
description Text

health Number Hidden
power Number Hidden

Figure 16. Attribute inheritance example 2

In the above examples as in Figure 15 and 16, Unit Card type is a subtype of CARD type. Therefore, mana and description attributes are inherited to the Unit Card.

Attribute Visibility

There are options for attributes' visibility. You can choose to show an attribute on screen canvas, AR marker. You can make them hidden. For example, a Player's card object can be shown on AR, the deck can be shown on the screen and player luck can be hidden.

Type Creation

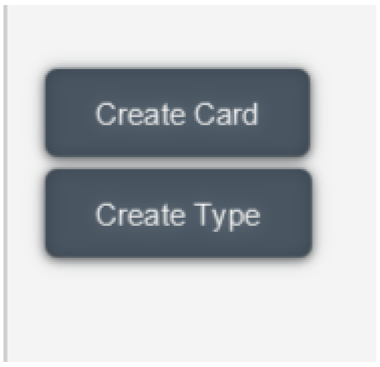


Figure 17. Create type

To create a type, click create type as in Figure 17.

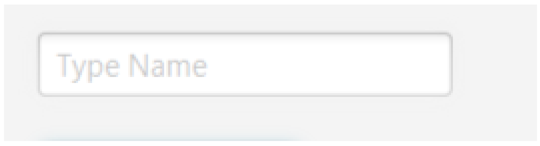


Figure 18. Create type click

In the opened field, type the type name and press ***ENTER*** as in Figure 18***.***

Removing A Type

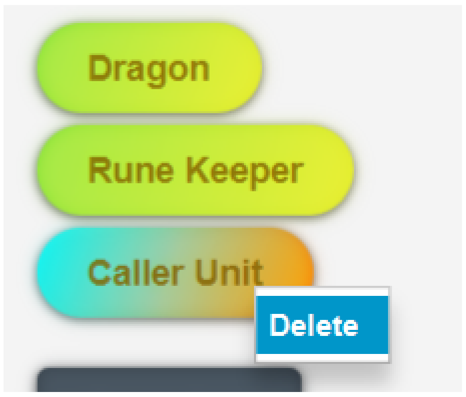


Figure 19. Remove Type

To remove a type, right click and choose delete as in Figure 19.

Card Creation

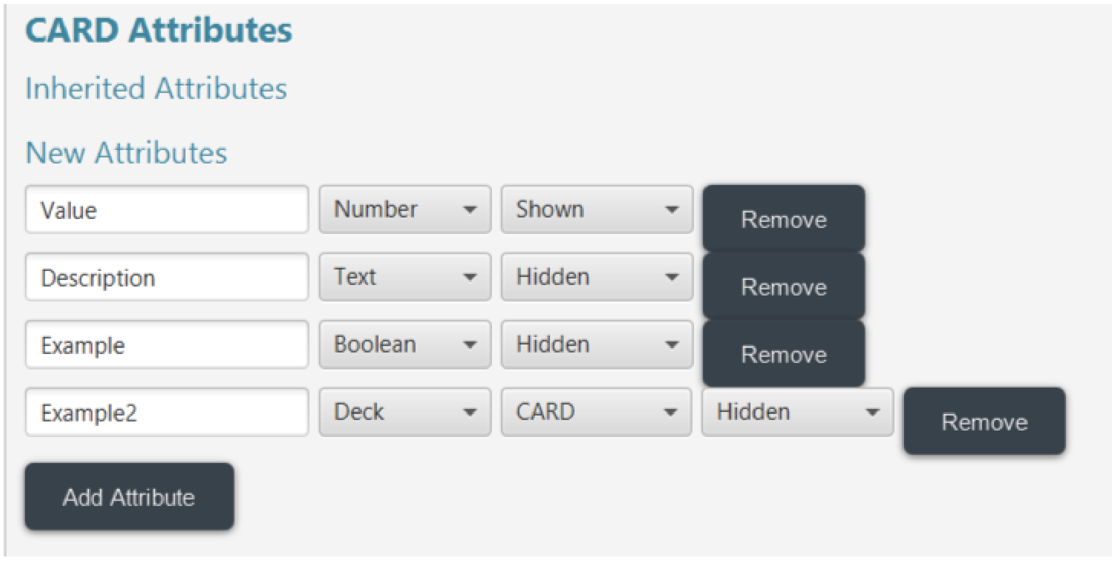
Creating a card is similar to creating a type. Just click create card instead. In the opened field, type the card name and press ***ENTER.***

Removing A Card

Removing cards is the same as removing types.

Setting Attribute Value

After setting attributes of a type, specify the values for cards as in Figure 20.



CARD Attributes

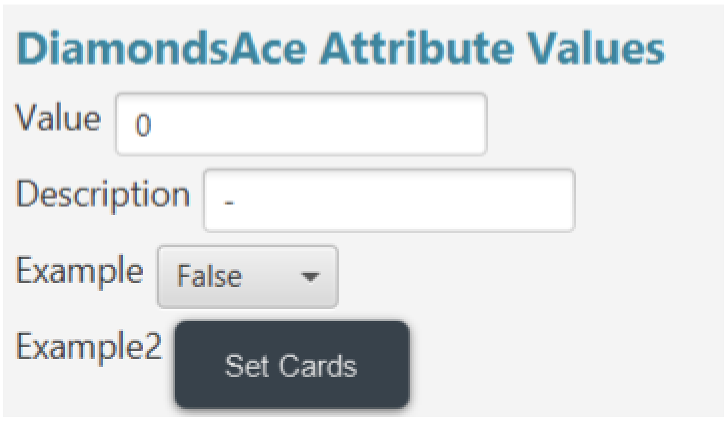
Inherited Attributes

New Attributes

Value	Number	Shown	Remove	
Description	Text	Hidden	Remove	
Example	Boolean	Hidden	Remove	
Example2	Deck	CARD	Hidden	Remove

Figure 20. Setting Attribute Value

After adding attributes as shown above to a type, open a card under that type as in Figure 21.



DiamondsAce Attribute Values

Value: 0

Description: -

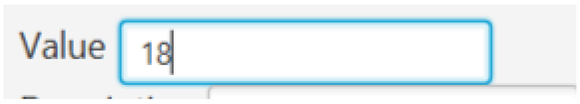
Example: False

Example2

Figure 21. Open a card

You can see that there are value fields for each attribute of the type.

Setting Numeric Attribute Value

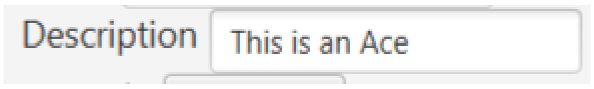


Value: 18

Figure 22. Open a card value field

Type a number to the value field. Initial value is 0. Note that you can only type numbers here as in Figure 22.

Setting Text Attribute Value



Description: This is an Ace

Figure 23. Text to the value field

Type a text to the value field. Initial value is “-”. Note that some characters and names are not allowed as in Figure 23.

Forbidden Inputs.

- ARGS
- ITERATE
- THIS
- GAME
- SELECT_NEXT
- SELECT_FIRST
- POP

Those inputs are not allowed because they are reserved keywords for out event system.

Setting Boolean Attribute Value

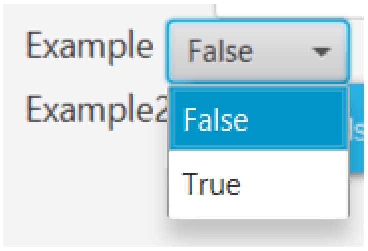


Figure 24. Dropdown menu

Select ***True*** or ***False*** from the dropdown menu as in Figure 24.

Setting Deck Attribute Value

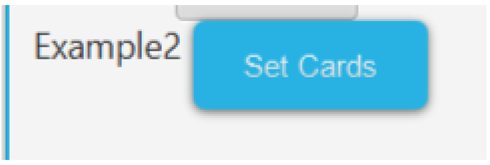


Figure 25. Set cards

Click set cards as in Figure 25. This will open a popup in which you can set the cards in the deck as in Figure 26. Note that you change the deck content via events as well.

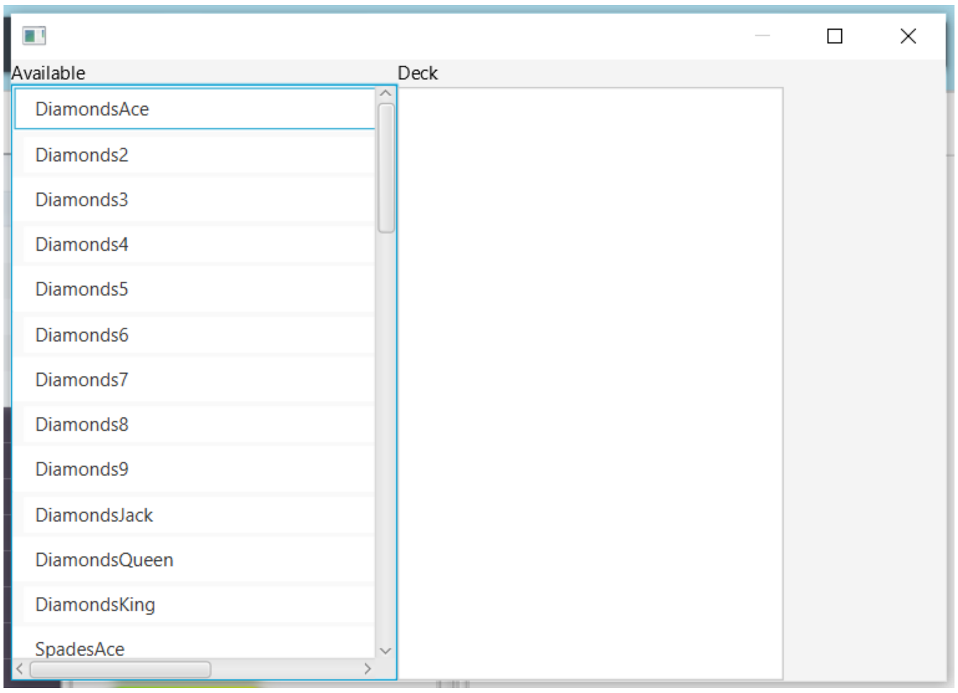


Figure 26. Set the cards

Left side shows available cards and the right side shows the currently added cards. Click a card on the left side to add to deck and click right side to remove from the deck as in Figure 27.

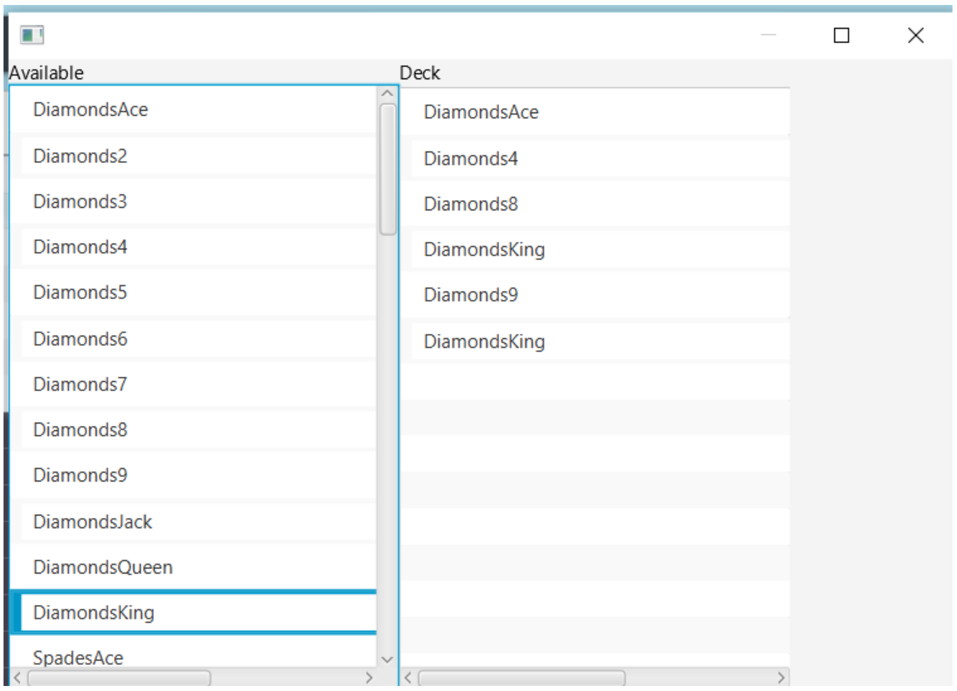


Figure 27. Deck filled

In the above example, a deck is filled with 5 cards from the available cards. It is possible to add the same card multiple times.

Player & Game Objects

Player and Game are special constructs in AugCards as in Figure 28.

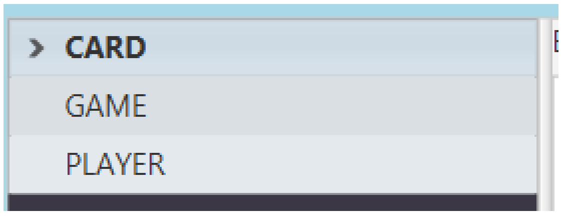
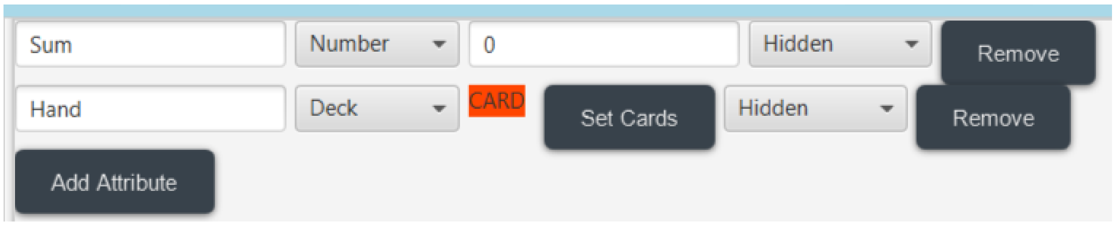


Figure 28. Special constructs

They have attributes and attribute values the same way Cards have. However, they define their attributes and values at the same time. I.e they don’t have subtypes etc.
Game has two fixed attributes named Players and ActivePlayer which users cannot modify as in Figure 29. Note that Players is a Deck attribute and ActivePlayer is a Card attribute. Player is not a Card in actual sense. In AugCards terminology, or game objects are “Card” type and all lists are Deck typed. These two predefined attributes are used in the game logic.



Sum	Number	0		Hidden	Remove
Hand	Deck	CARD	Set Cards	Hidden	Remove

Figure 29. Fixed attributes

As explained above, Player object has attribute and attribute value together. The deck attributes in Player are shown in game each list being a column of cards. Other attributes of the player are shown on the upper screen.

Event Manager

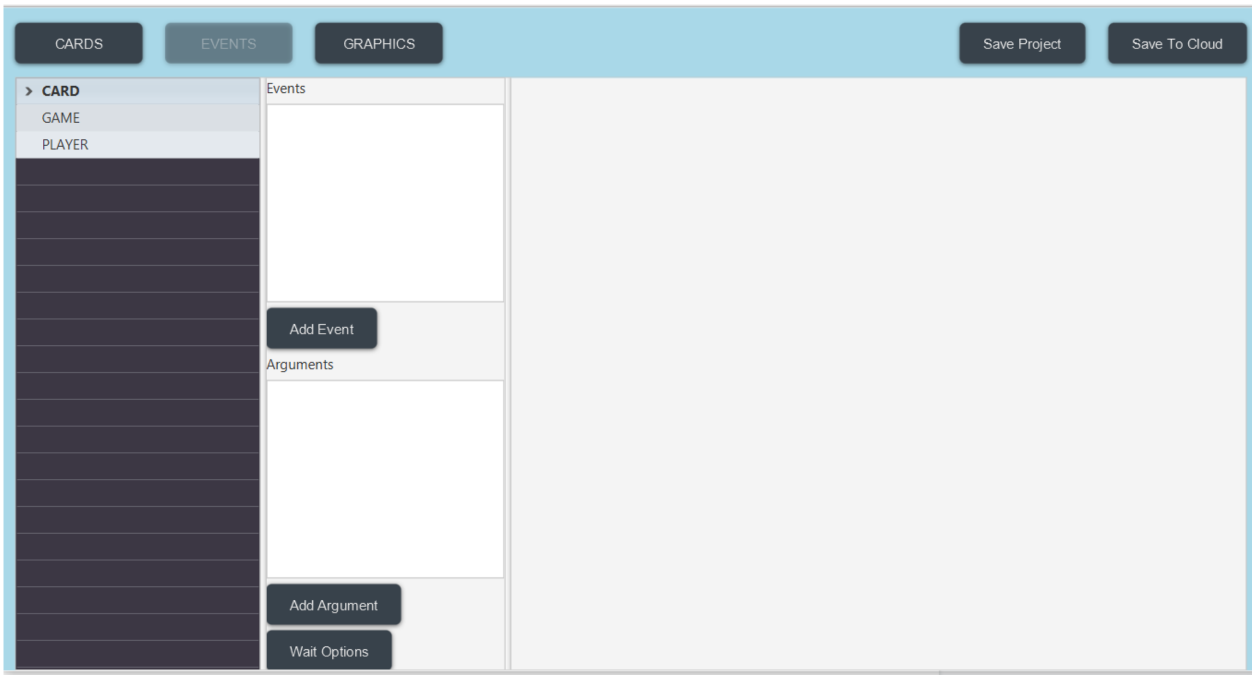
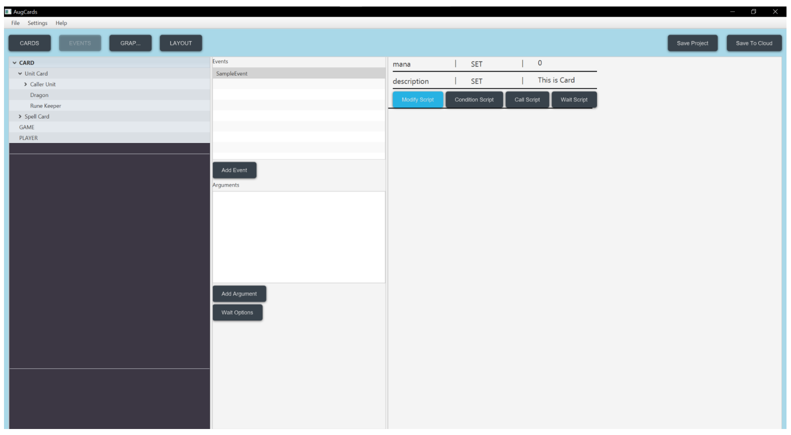


Figure 30. Event design scene

The general view of the event design scene as in Figure 30. Any event modification can be performed over game instance selection from the mostleft area. Some options:

- Add event - define new event
- Add argument - define argument types for selected event
- Wait options - set text visual when a wait triggered over such event

The view of inserting a new argument. Users need to select an event to add a new event. Then, a pop-up window will appear to specify details of the argument like name and type as in Figure 31.



mana | SET | 0
description | SET | This is Card

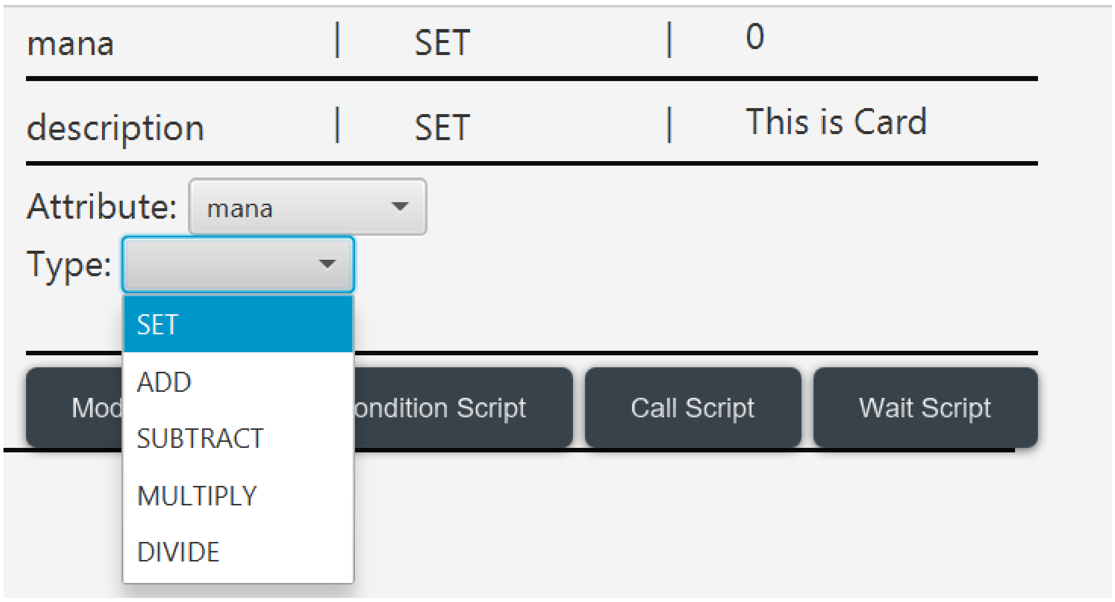
Figure 31. Argument scene

In the most right broad area, users can edit the scripts of the selected event. In script area, there are button for each script type:

- Modify Script - attribute modification scripts
- Condition Script - check rules and their script block for branching after checks
- Call Script - raising an event through specifying its event arguments
- Wait Script - hanging up the game while waiting an event to occur

Inserted scripts transformed into layout views while they are not modified. In script modification, users can select accessible attributes within the event scope (game instance).

Modify Script



mana | SET | 0

description | SET | This is Card

Attribute: mana

Type:

Figure 32. Modify script

The different types of modifications for a numeric attribute as in Figure 32. "Set" allows the developer to set the value of a numeric attribute, "Add", "Subtract", "Multiply" and "Divide" perform the corresponding arithmetic operations on the attribute when another value is provided as in Figure 33.

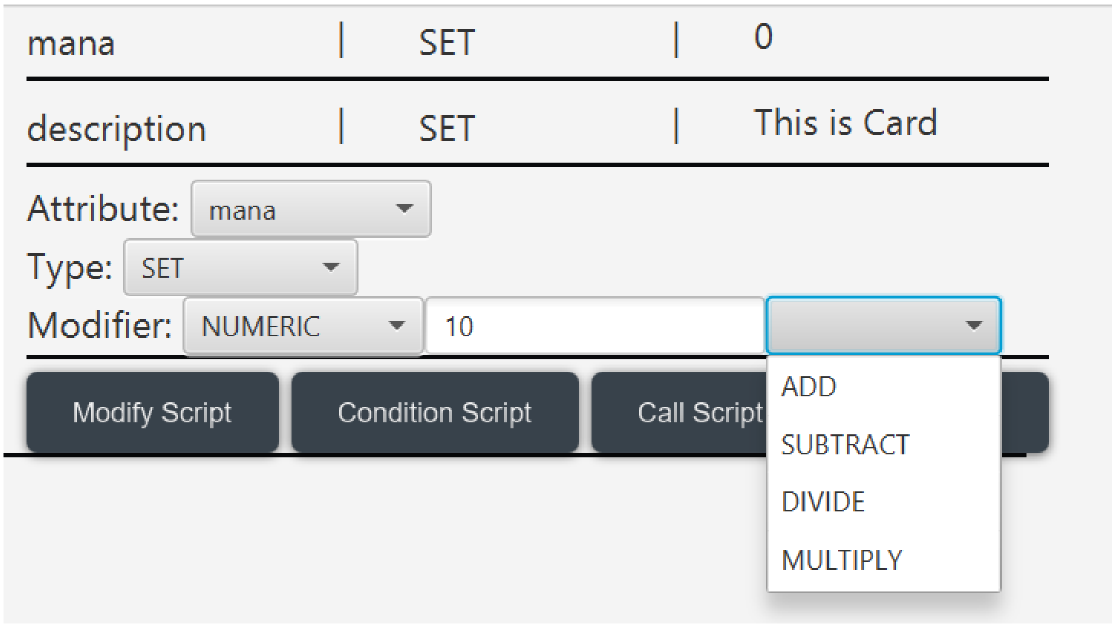


Figure 33. Arithmetic operations on the attribute

Setting the value for a numeric attribute as in Figure 34. In the above example, a numeric value used to define required modification.

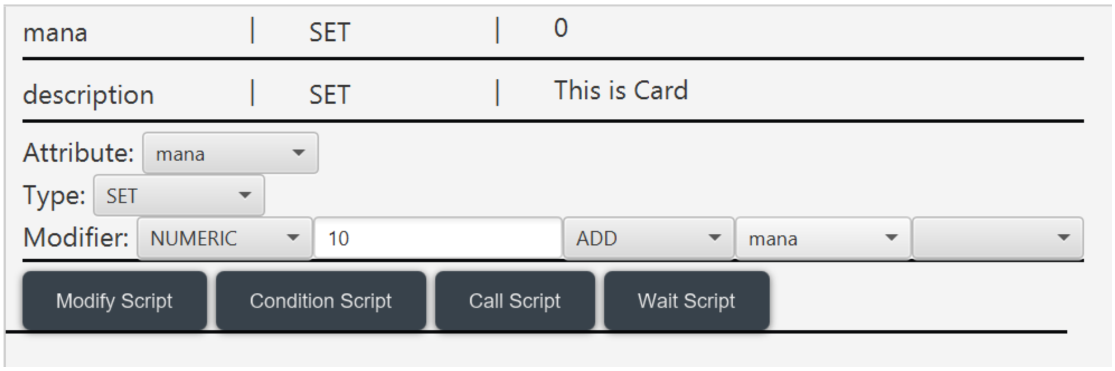
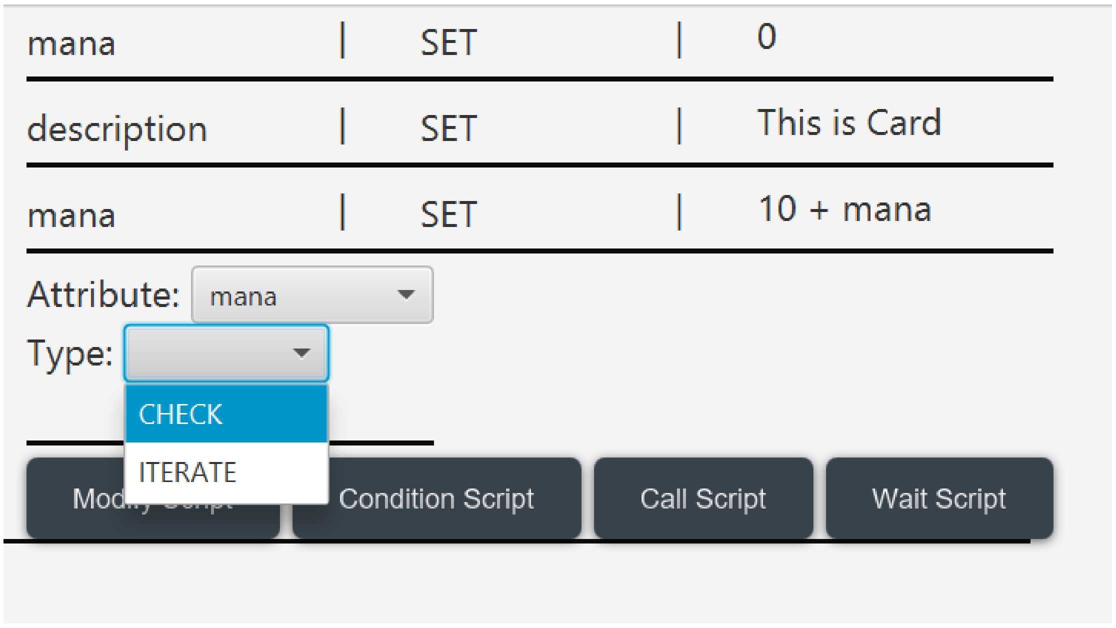


Figure 34. Set the value for a numeric attribute

Arithmetic operations can be chained to perform complicated calculations. Also, it is possible to refer to other numeric attributes to use them in operation.

Case Script



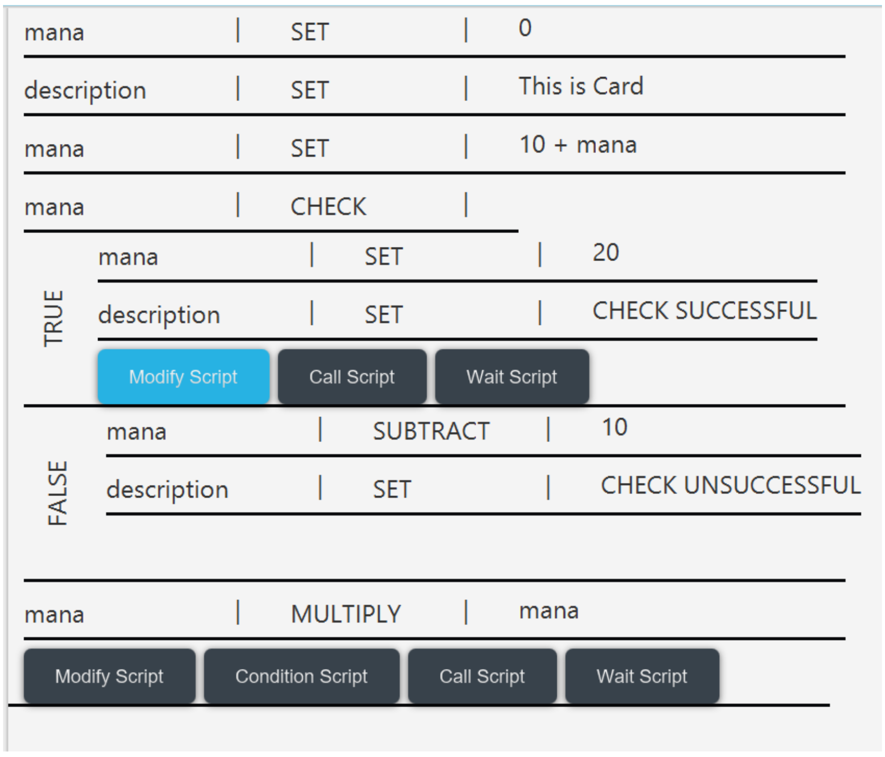
mana	SET	0
description	SET	This is Card
mana	SET	10 + mana

Attribute: mana

Type:

Figure 35. Case script

A case script can be of type Check and Iterate as in Figure 35. Check is similar to the If, Else and Iterate is similar to for. You can iterate a deck or over a number.

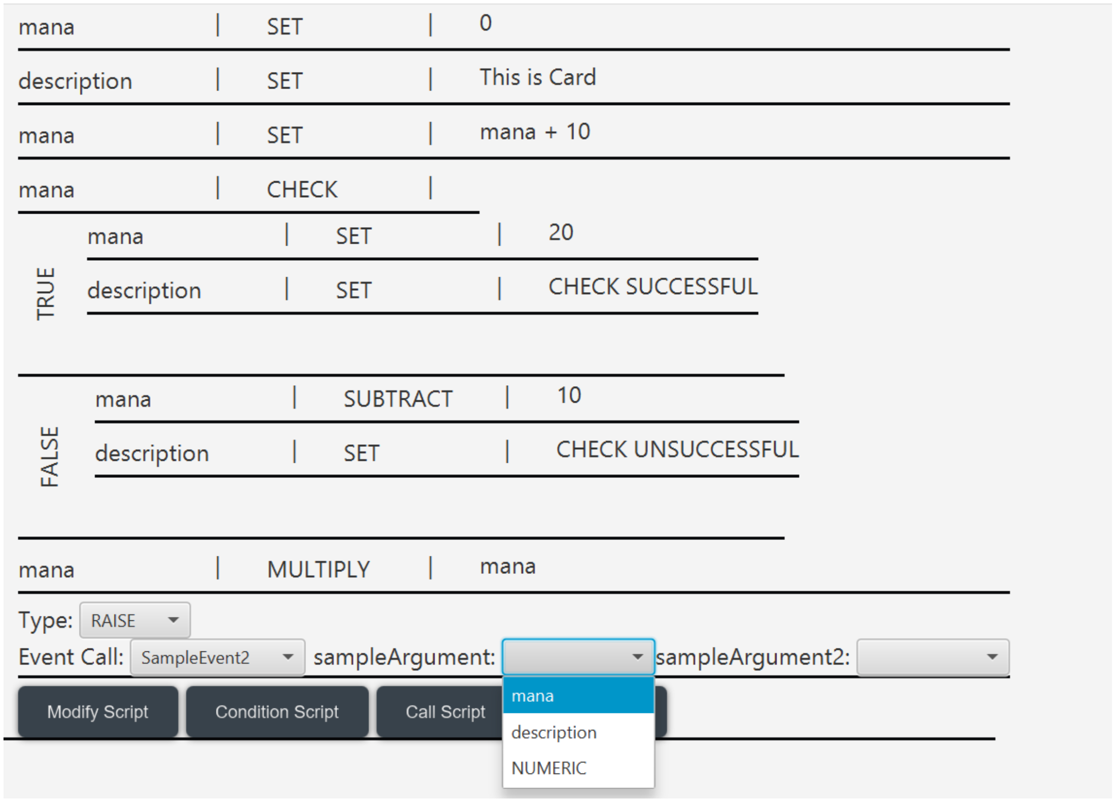


```
mana          | SET        | 0
description   | SET        | This is Card
mana          | SET        | 10 + mana
mana          | CHECK      |
TRUE
    mana          | SET        | 20
    description   | SET        | CHECK SUCCESSFUL
FALSE
    mana          | SUBTRACT   | 10
    description   | SET        | CHECK UNSUCCESSFUL
mana          | MULTIPLY   | mana
```

Figure 36. Case script True and False blocks

Case script has True and False blocks under it as in Figure 36. When the checked case is true, true block scripts are executed and the reverse is true for vice versa.

Raise Event



mana | SET | 0

description | SET | This is Card

mana | SET | mana + 10

mana | CHECK |

TRUE

mana | SET | 20

description | SET | CHECK SUCCESSFUL

FALSE

mana | SUBTRACT | 10

description | SET | CHECK UNSUCCESSFUL

mana | MULTIPLY | mana

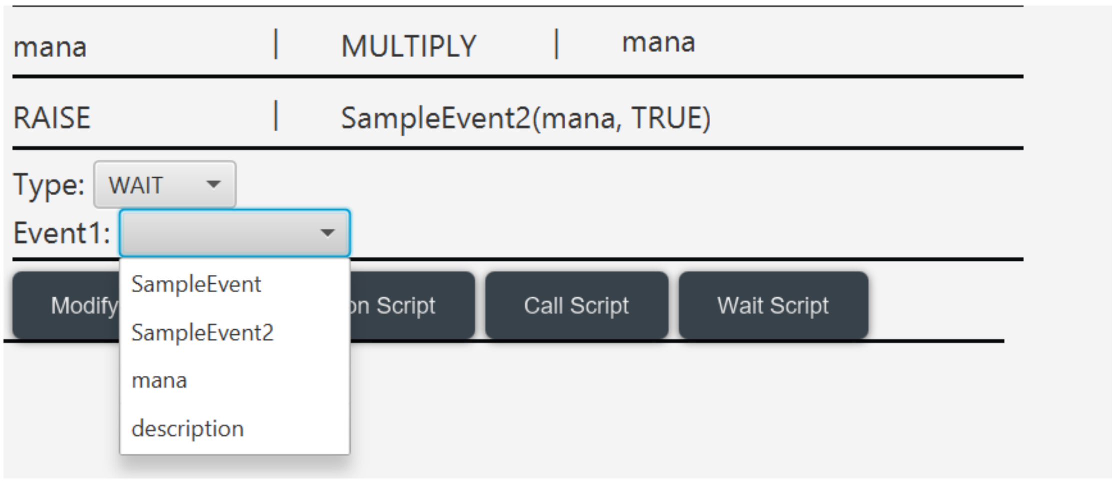
Type: RAISE

Event Call: SampleEvent2 sampleArgument: sampleArgument2:

Figure 37. Raise event

Users can define event call scripts by selecting the event to raise and specifying event arguments as in Figure 37. Similar to modification, in argument specification, an attribute reference or new value can be used.

Wait Script


mana | MULTIPLY | mana

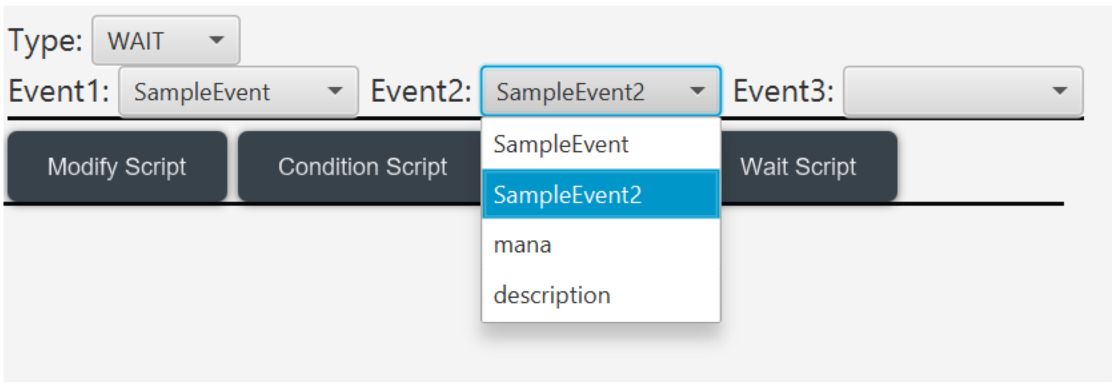
RAISE | SampleEvent2(mana, TRUE)

Type: WAIT

Event1:

Figure 38. Wait script

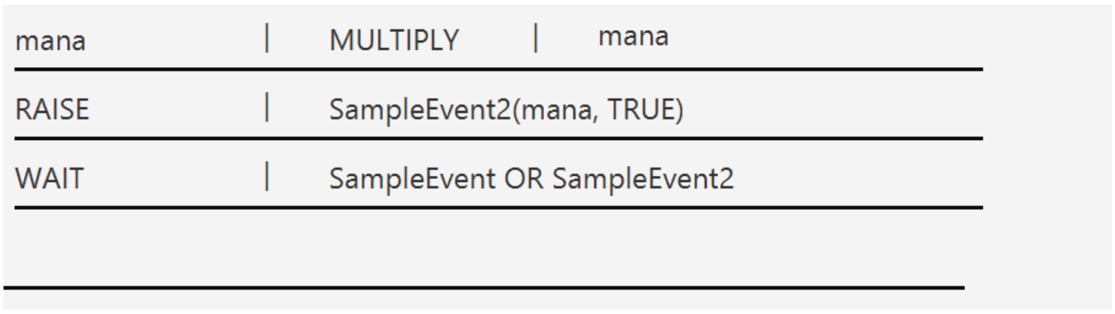
Users can define wait scripts by selecting only event references as in Figure 38.


Type: WAIT

Event1: SampleEvent Event2: SampleEvent2 Event3:

Figure 39. Event references

Users can define multiple events to wait at once as in Figure 39.


mana | MULTIPLY | mana

RAISE | SampleEvent2(mana, TRUE)

WAIT | SampleEvent OR SampleEvent2

Figure 40. Layout view of the wait script

Here the layout view of the wait script as in Figure 40.

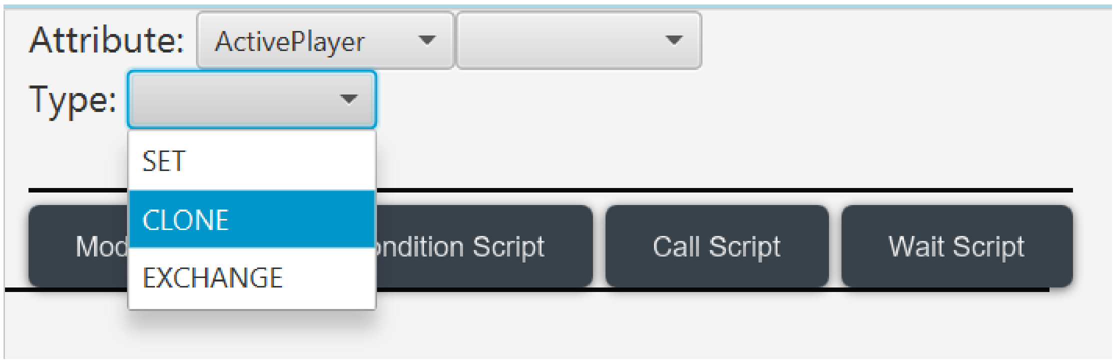


Figure 41.Modification

On custom attribute, there are three modification type can be applied as in Figure 41:

- SET
- CLONE
- EXCHANGE

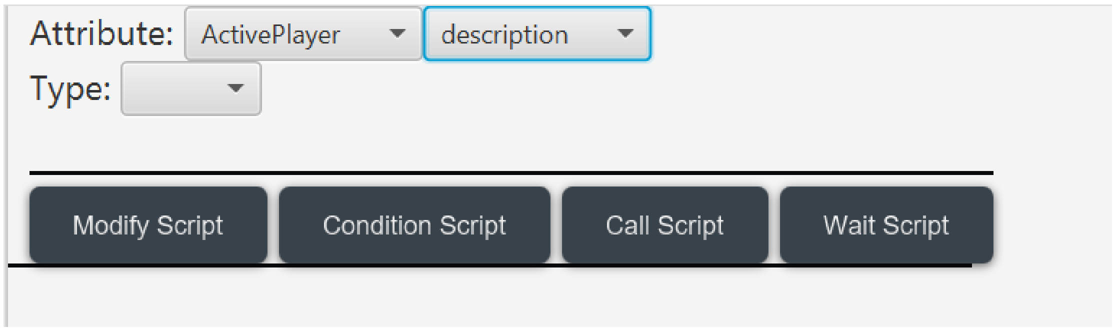


Figure 42. Attributes of a custom instance

Users can refer into further attributes of a custom instance as in Figure 42 and as in Figure 43.

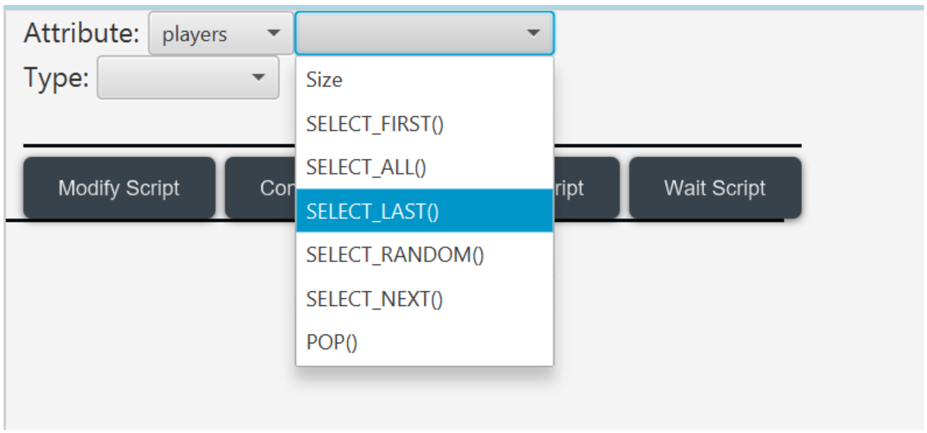


Figure 43. Attributes of a custom instance

There are some options to query a deck.

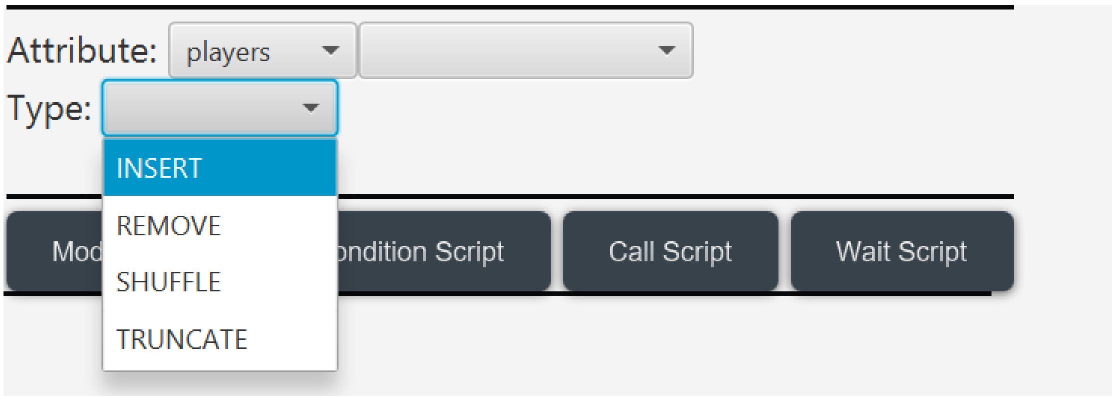


Figure 44. Options to query a deck.

On list attributes, there are four modification types as in Figure 44:

- Insert
- Remove
- Shuffle
- Truncate

Visual Manager

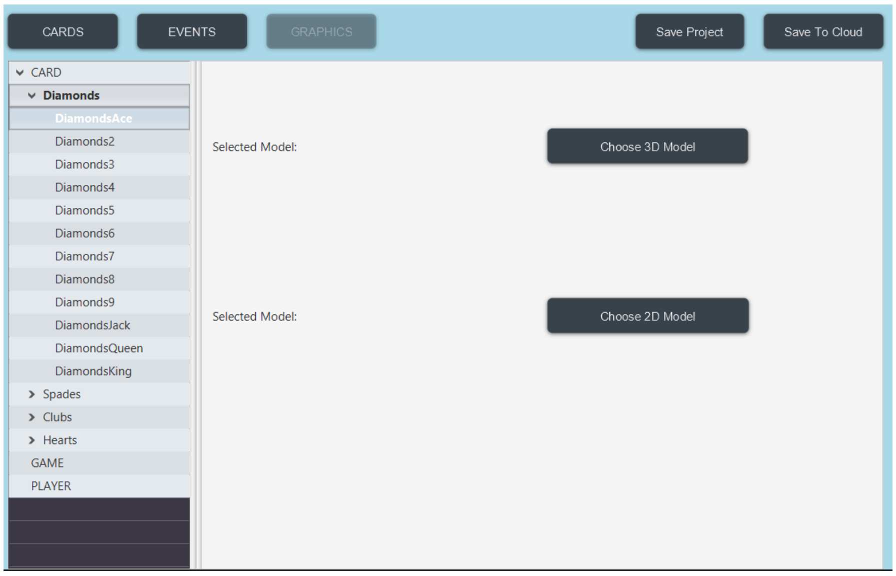


Figure 45. Visual Manager

Select 3D models and 2D textures here as in Figure 45 as in Figure 46.

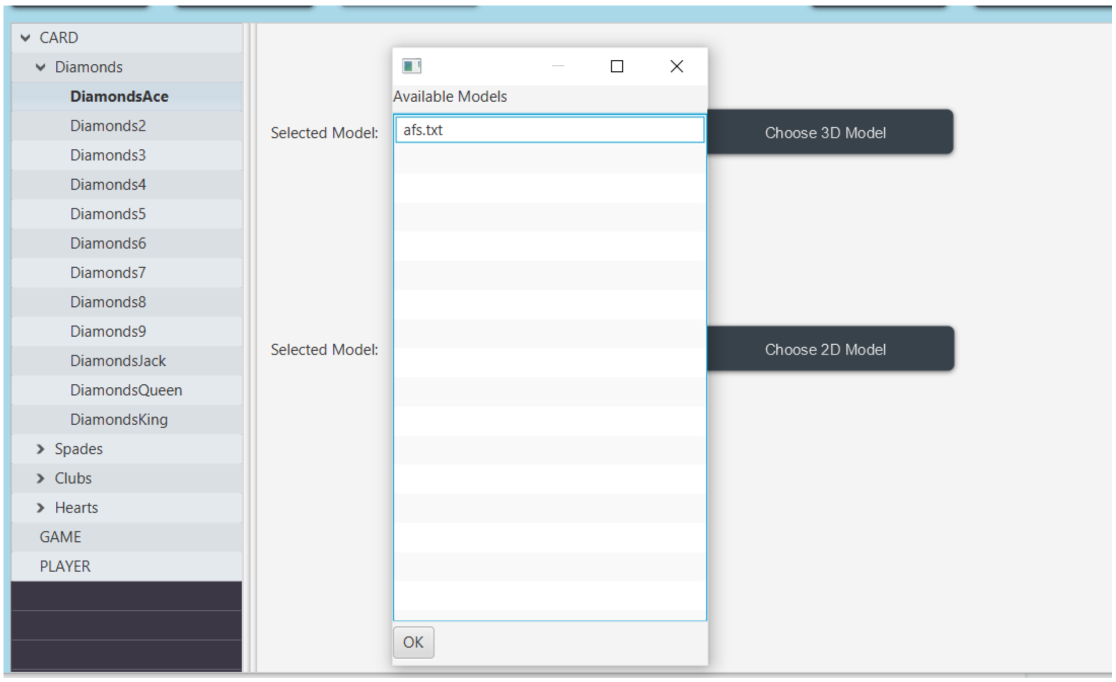


Figure 46. Select 3D models and 2D textures

Models in the ***.../User/Documents/AugCards/models*** directory are shown as options. The game will use these 2D and 3D models as game assets for the objects.

Mobile Component

To use the mobile component, the user should either print or draw the marker that is provided to the right. This marker is used as a way to track where objects should be placed using Augmented Reality as in Figure 47.

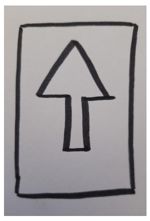


Figure 47. Marker

Launch Scene

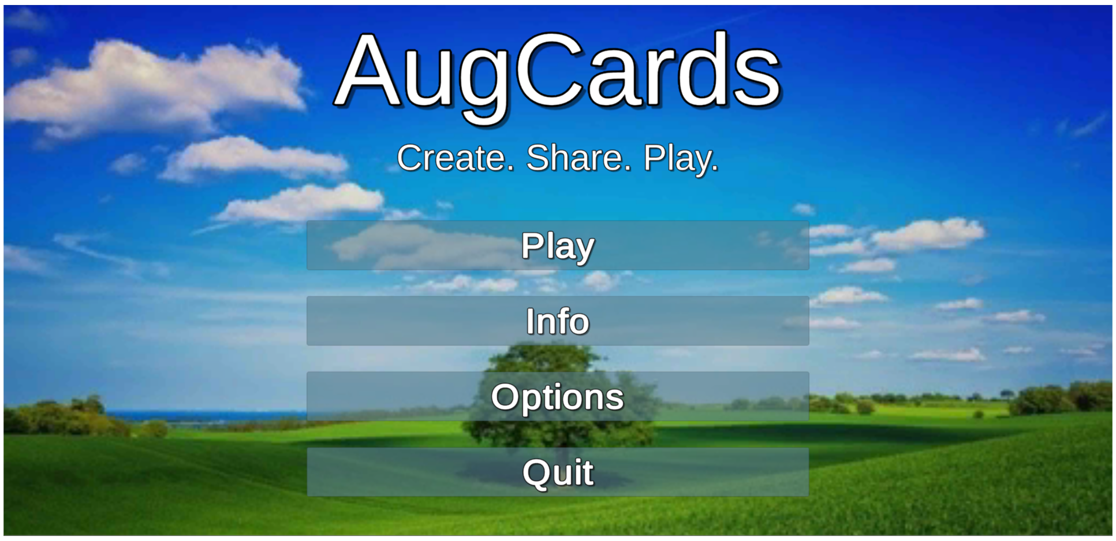


Figure 48. Launch

In the launch scene, the player can choose to play the game by pressing the "Play" button, Get information about the game's development team by pressing the "Info" button, configure various game options by pressing the "Options" button, and quit the game by pressing the "Quit" button as in Figure 48.

Game Selection Scene

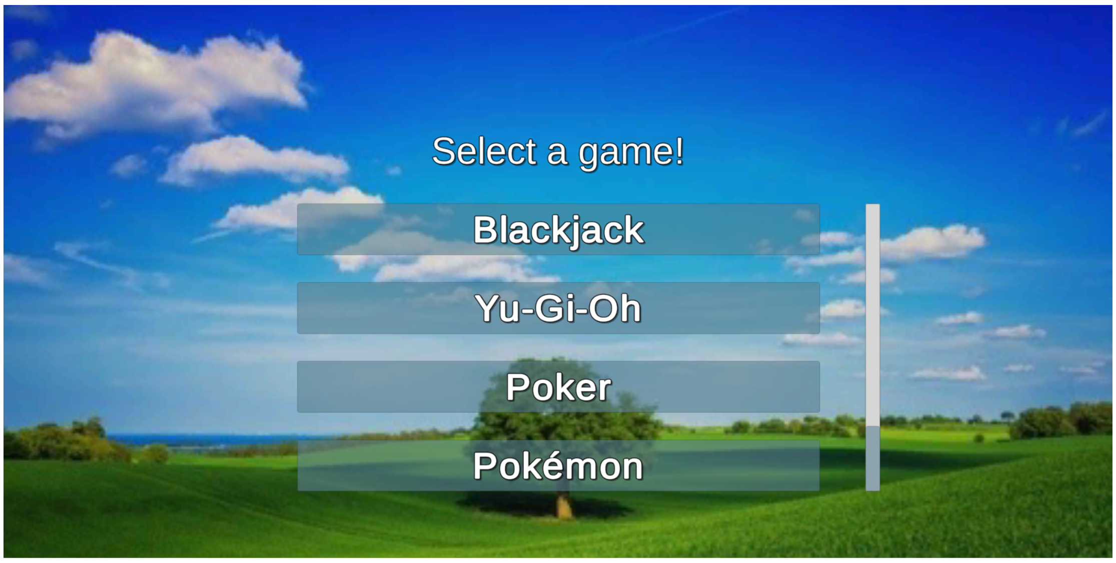


Figure 49. Game Selection

When the player chooses to play a game, they are presented with the different games that are available for them to play as in Figure 49. These games are shown in a list, and the user can choose a game to play a round.

Game Hosting Scene

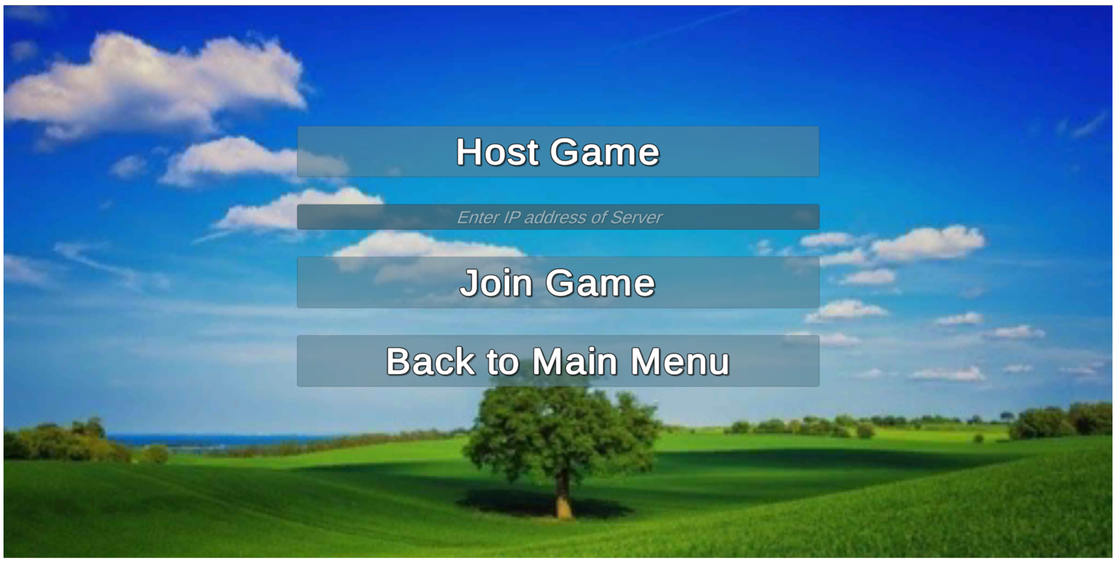


Figure 50. Game hosting

After the player chooses which game they would like to play, they are presented with the options to host a lobby or join a pre-existing lobby by providing the IP address of that lobby as in Figure 50. The “Host Game” button allows a user to host a game on their local machine for other players to join. When the player wants to join a game, they can fill out the input field that says “Enter IP address of server” with the IP address of the server that they would like to join and click the “Join Game” button.

Lobby Scene

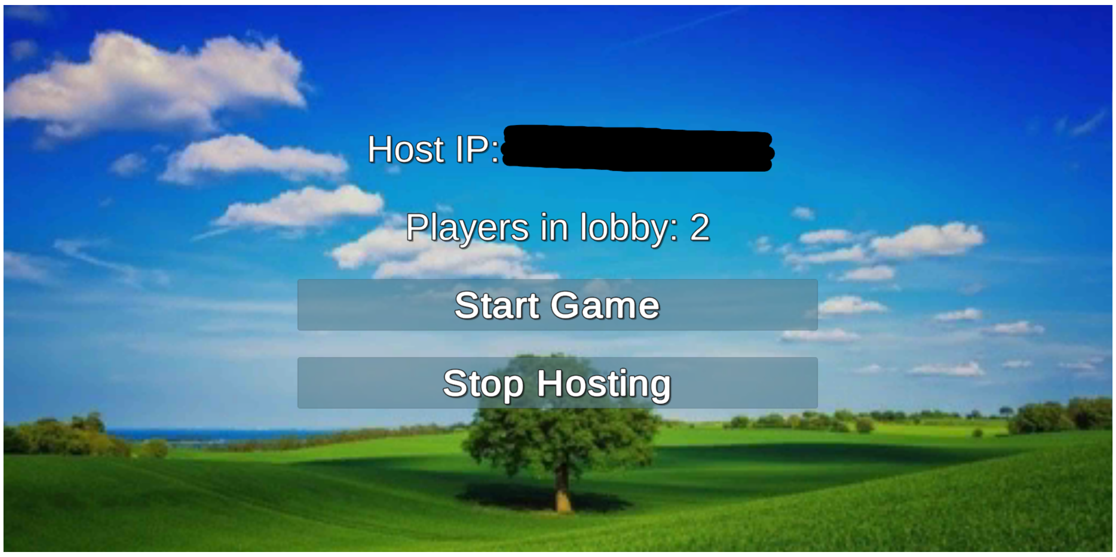


Figure 51. Game Selection

In the lobby scene, the host is presented with the options to start playing the game or to stop hosting the game as in Figure 51. If the host presses the "Start Game" button, the game will start on all connected clients. If the host decides to press the "Stop Hosting" button instead, all clients and the host themselves are disconnected from the game lobby. The status of the server, the IP address of the host and the number of connected clients are also visible on this scene, and this is what the clients of a hosted game can see. They do not see the options that the host is provided with.

Main Game Scene

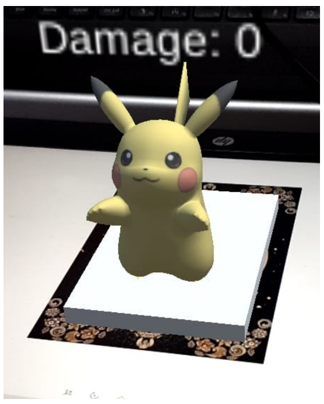


Figure 52. Ingame model

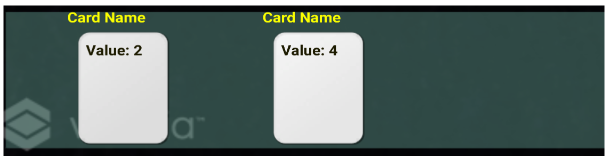


Figure 53. Ingame Cards

When the game starts, the users can see the hands that are handed to them Figure 52. Ingame. On the screen, the players can see the cards that they have, which player's turn it is, any relevant values for the game and when it's their turn, the possible actions that they can take during their turn Figure 53. Ingame. The players can also see the game field through their camera in Augmented Reality. The players can toggle whether their hand is visible and whether the values of cards are visible. As an example, when the user is playing Blackjack or 21, they can see the sum of their hand and they are provided with the options to a) "Hit" and b) "Stand", and they can choose to either a) draw a card or b) stand with their hand and sum. Some actions may require additional steps, such as choosing a card or entering a value to apply that action. The result of clicking an event can be seen in Figures 54 (before) and 55 (after), where a blackjack game is taking place.

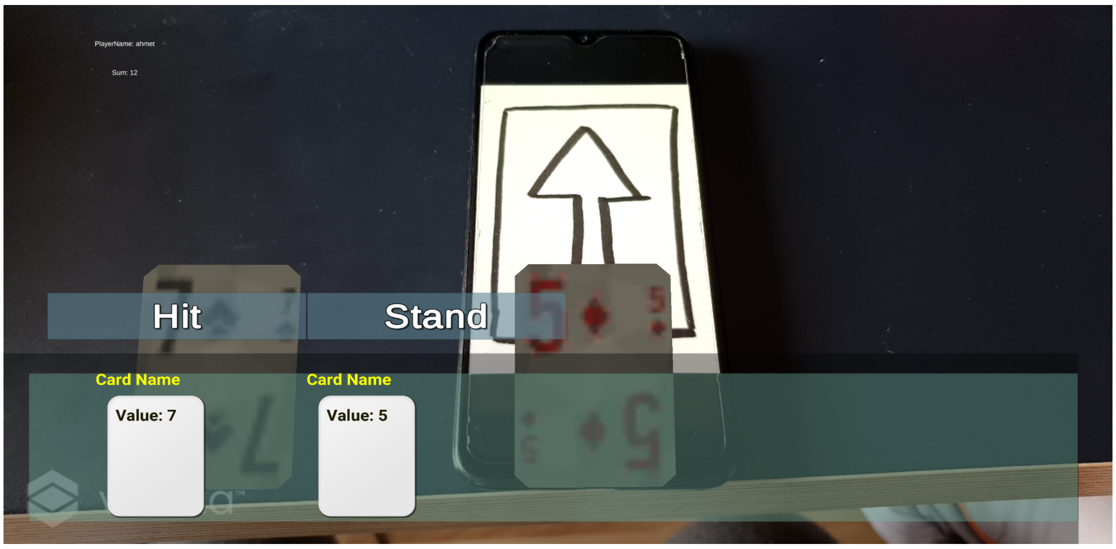


Figure 54: Ingame Rendering of Cards Before Hit

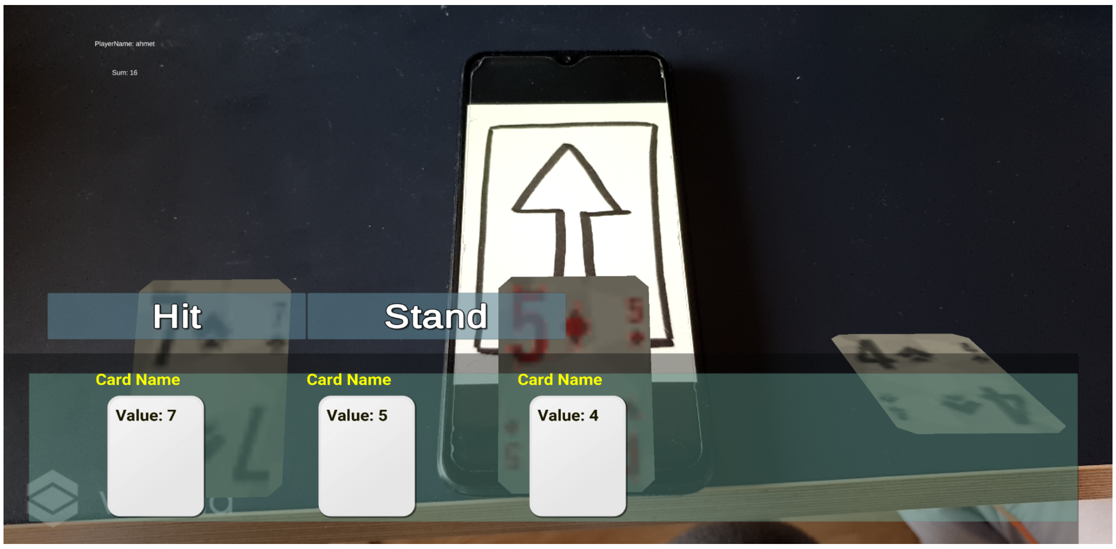


Figure 55: Ingame Rendering of Cards After Hit