



**National Institute of Applied Science And Technology**

**UNIVERSITY OF CARTHAGE**

End of Year Project

Branch: Software Engineering

**Collect Connect Game Development with NLP and  
Image Similarity**

---

**Presented by**

ACHOUR Sirine

KACEM Farah

DERBEL Sahar

**Under the supervision of**

SFAXI Lilia

**Academic Year: 2020/2021**

## **ACKNOWLEDGEMENTS**

We have had the utmost pleasure of working with our INSAT supervisor Lilia Sfaxi, who has shown consistent support throughout the realization of this project. We would also like to thank the Client team for the opportunity they gave us to work with them on this project, and for the fun we had throughout. Our thanks, also, to our families.

# Contents

- List of Figures** v
  
- List of Tables** vii
  
- 1 Introduction** 1
  - 1.1 General introduction . . . . . 1
  - 1.2 Context . . . . . 3
    - 1.2.1 Concept . . . . . 3
    - 1.2.2 Requirements . . . . . 4
  - 1.3 Project Timeline . . . . . 5
  - 1.4 Conclusion . . . . . 5
  
- 2 Game Design and Development** 7
  - 2.1 Introduction . . . . . 7
  - 2.2 Methodology . . . . . 7
  - 2.3 Functional Requirements . . . . . 8
    - 2.3.1 Out-of-game functionalities . . . . . 9
    - 2.3.2 In game functionalities . . . . . 12
    - 2.3.3 Scoring . . . . . 17
    - 2.3.4 Design . . . . . 17
  - 2.4 Notions and Technologies . . . . . 21
  - 2.5 Non Functional Requirements . . . . . 21
    - 2.5.1 Availability and Performance . . . . . 21

---

2.5.2	Portability . . . . .	22
2.6	Conclusion . . . . .	22
<b>3</b>	<b>Database and API</b>	<b>23</b>
3.1	Introduction . . . . .	23
3.2	Database . . . . .	24
3.2.1	Purpose . . . . .	24
3.2.2	Design . . . . .	26
3.3	API . . . . .	29
3.3.1	Purpose . . . . .	29
3.3.2	Design . . . . .	30
3.3.3	Access levels . . . . .	33
3.3.4	API endpoints . . . . .	34
3.4	Non-functional requirements . . . . .	37
3.4.1	Portability : . . . . .	37
3.4.2	Documentation : . . . . .	37
3.4.3	Scalability and Availability : . . . . .	37
3.5	Conclusion . . . . .	38
<b>4</b>	<b>AI Judge</b>	<b>39</b>
4.1	Methodology . . . . .	39
4.2	Dataset . . . . .	41
4.3	Natural language processing . . . . .	41
4.3.1	Text preprocessing . . . . .	41
4.3.2	Model implementaion: . . . . .	42
4.4	Computer vision . . . . .	45
4.4.1	Models: . . . . .	45
4.4.2	Results: . . . . .	46
4.4.3	Model implementation: . . . . .	47
4.4.4	Computer vision output: . . . . .	48

4.4.5	Weighting the data: . . . . .	48
4.5	Flask backend: . . . . .	49
4.6	Non-functional requirements . . . . .	49
4.6.1	Portability: . . . . .	49
4.6.2	Documentation: . . . . .	49
4.6.3	Modifiability: . . . . .	49
4.6.4	Performance: . . . . .	49
4.6.5	Scalability: . . . . .	49
4.7	Conclusion: . . . . .	50
<b>5</b>	<b>Evaluation and Deployment</b>	<b>51</b>
5.1	Introduction . . . . .	51
5.2	Evaluation . . . . .	51
5.2.1	Client feedback and Playtesting . . . . .	51
5.2.2	Performance . . . . .	54
5.3	Deployment . . . . .	55
5.4	Conclusion . . . . .	55

# List of Figures

1.1	An example of a card: “Caravaggio’s Medusa”	3
1.2	Gantt diagram of project timeline	6
2.1	Out-of-game Use Case diagram	9
2.2	Login, Register and Recover screens.	10
2.3	Home, Scoreboard and Collection screens.	11
2.4	Gameplay Use Case diagram	12
2.5	Game board	13
2.6	Tutorial screens and settings menu.	13
2.7	Non playable card, front and back.	14
2.8	Playable card, front and back.	14
2.9	Accepted link screen	15
2.10	Rejected link screen	15
2.11	Pause and settings screen	16
2.12	Final game screens	16
2.13	Screen Flow diagram for unity game	18
2.14	Software class diagram	20
3.1	Database simplified schema	26
3.2	Database detailed schema	28
3.3	API modules diagram	30
3.4	AuthModule services class diagram	31
3.5	CardModule services class diagram	32

3.6	GameModule services class diagram . . . . .	33
4.1	Crisp-DM methodology[2] . . . . .	40
4.2	Jaccard similarity formula . . . . .	42
4.3	ResNet-50 model architecture [7] . . . . .	46
4.4	Sample of cards to test models on . . . . .	47
4.5	Turicreate output . . . . .	48
5.1	Feelings on the scoring system . . . . .	53
5.2	Feedback on the scoring system . . . . .	53
5.3	Feedback on the AI as a character . . . . .	53
5.4	Feedback on the performance aspect . . . . .	54
5.5	Deployment diagram . . . . .	55

# List of Tables

3.1	Register endpoint . . . . .	34
3.2	Recover endpoint . . . . .	34
3.3	Get percentage endpoint . . . . .	35
3.4	Login endpoint . . . . .	35
3.5	New game endpoint . . . . .	35
3.6	Quit game endpoint . . . . .	35
3.7	Play card endpoint . . . . .	36
3.8	Add percentage endpoint . . . . .	36
3.9	Get collection by name . . . . .	37
3.10	Get card by name endpoint . . . . .	37
3.11	Add new card endpoint . . . . .	37
4.1	Card names and their corresponding descriptions . . . . .	43
4.2	Cards name and their corresponding descriptions . . . . .	44
4.3	Comparison between different image similarity models . . . . .	47



# Chapter 1

## Introduction

### 1.1 General introduction

AI and games have been mashed together since the years of Nim and Pong. It was a matter of creating games that were playable on your own, but emulated the distinct feeling of playing with, or against, someone. AI has gone through to revolutionize play, with it beating chess champions in 1997, and Go champions as recently as 2016. Conversely, game development has been on the rise as a career choice, since it requires the perfect mix of creativity and programming knowledge.

The University of Michigan provides courses on game development, the cultural impact of games, as well as machine learning and AI. This project follows this trend of mashing culture and programming to create impactful interactive games to the larger public. More specifically, the Client is interested in different aspects of translation, finding connections between translated works, library artworks and such, as well as cultural collections : Objects that fit to a theme and that can be bagged together into a coherent unit. The University of Michigan Museums and Libraries contain a vast selection of artworks, books, and games. This body of knowledge and fun is mostly left undiscovered by the students that attend there.

From this stems the need to help students discover artwork and find joy in learning. Thus were created the Translation Networks. They encompass various projects that gravitate around the idea of a platform where students can look up connections between catalog items that they have

interacted with, as well as record connections they have found. This network pushes students to expand their knowledge by helping them find which books, artwork, etc to review next.

The Librarians, researchers and professors there have curated some of these items into collections. These collections are sets of artwork gathered around a theme. Providing this user input is a group effort on the part of students as well as professors and librarians. This is where AI can come to the rescue. It is widely used to find text and image similarities across different domains. Basic examples of this are:

- Recognizing handwritten letters, and classifying them.
- Classifying documents around themes.
- Face recognition.

But finding and using these connections would still need to be done by humans for it to be fun and engaging. For this reason, the AI was applied to a game centered around finding and strengthening these connections. This game is called Collect Connect. It strives to provide an enjoyable experience to players as they create connections that an AI can judge and give opinions on, as well as a commentary on the difference that can be found between the thought process of AI and human users. It applies the AI's judgement while also rewarding unexpected connections with a different score.

Collect Connect is a prime example of educational games. It offers the opportunity to read about unusual and historical artifacts, artwork and games, while also pushing the player to make smart connections between them. In a period when visits are limited, games like these help museums and organizations keep in touch with the general public.

The Collect Connect game can also be seen as collaborative work between so many people across campus. Libraries bare their collections through APIs, researchers and students add data, curators choose which items to showcase in their collections, and the Collect Connect game centers around the discovery and linking of the cards.

[6] This game constitutes our contribution to Translation Networks. It combines the fields of game development and AI to expand the network as well as provide a nice, instructional experience to the player.

In this report, we will present the different requirements that we fulfilled in creating this game. It will revolve around three main parts, the game itself as a front end, the supporting backend, and the AI Judge.

## 1.2 Context

### 1.2.1 Concept

Collect connect is a game centering around the discovery of the University of Michigan libraries and museums. The player's task is finding interesting links between different objects from those collections, and creating a chain of related cards. Those cards are stacked with the goal of creating the longest chain between the start and end card.

A card represents an item, its front being an image of this item, and its back presenting a description of this item.



Figure 1.1: An example of a card: "Caravaggio's Medusa"

## 1.2.2 Requirements

We were tasked with making the Collect Connect solo game. This included three main axis:

- Designing and implementing a playable game that was, in its essence, fun.
- Making a robust backend and database, where games could be logged down to the smallest player action.
- Creating an AI, which is the heart of this game. It had to be capable of taking two cards as input and computing different kinds of similarities between them. Most important were the text comparison (description) and the image comparison.

Our starting point was an in-development Unity multiplayer game with a php file as a backend that only took care of authentication. The architecture was not very robust, and the code did not meet our needs.

Next, the game itself needed to be playable. In that sense, there was a need to determine by which standards cards could be stacked on top of each other, and the scoring system to use. We needed to determine which scoring system, if any, could be fair. We also needed to choose which AI technologies to use, their advantages and disadvantages.

Another problem that had to be avoided was prioritising links between cards in the same collection. Making sure that interesting connections could be made was important.

Then, we had to take into account the fact that the behaviour a player has during the game, such as the cards they stack together, the cards they choose to discard, the reasonings they provide, can be very interesting for research purposes. Yet those informations were lost, due to the lack of logging and a proper backend and database supporting it.

There was also the need to account for the player finding the judgement of the judge unfair, and having no hint as to why the judgement was what it was, or how the scoring system worked. Added to that, a simple score felt like not enough. The game is about collections, so the idea would be to let the player collect cards. The final problem we faced was the time necessary for the AI to issue a judgement. If the AI is too slow, the player might lose interest in the game.

### 1.3 Project Timeline

The game development process can be divided into weekly sprints. The backend development took off later in the process, while the AI development ran parallel to the game development during the entirety of the semester.

Please refer to figure [1.2](#) for a detailed timeline of the project.

### 1.4 Conclusion

In this chapter, we have presented the University of Michigan client, and more specifically the network that it will be a part of. We have stated the reason for this game's existence, and the context surrounding this project.

We have also presented the project timeline and objectives.

In the next chapters, we will talk in depth about the three main axis that this project was built on: Game design and development, Logging database and API, and the AI.

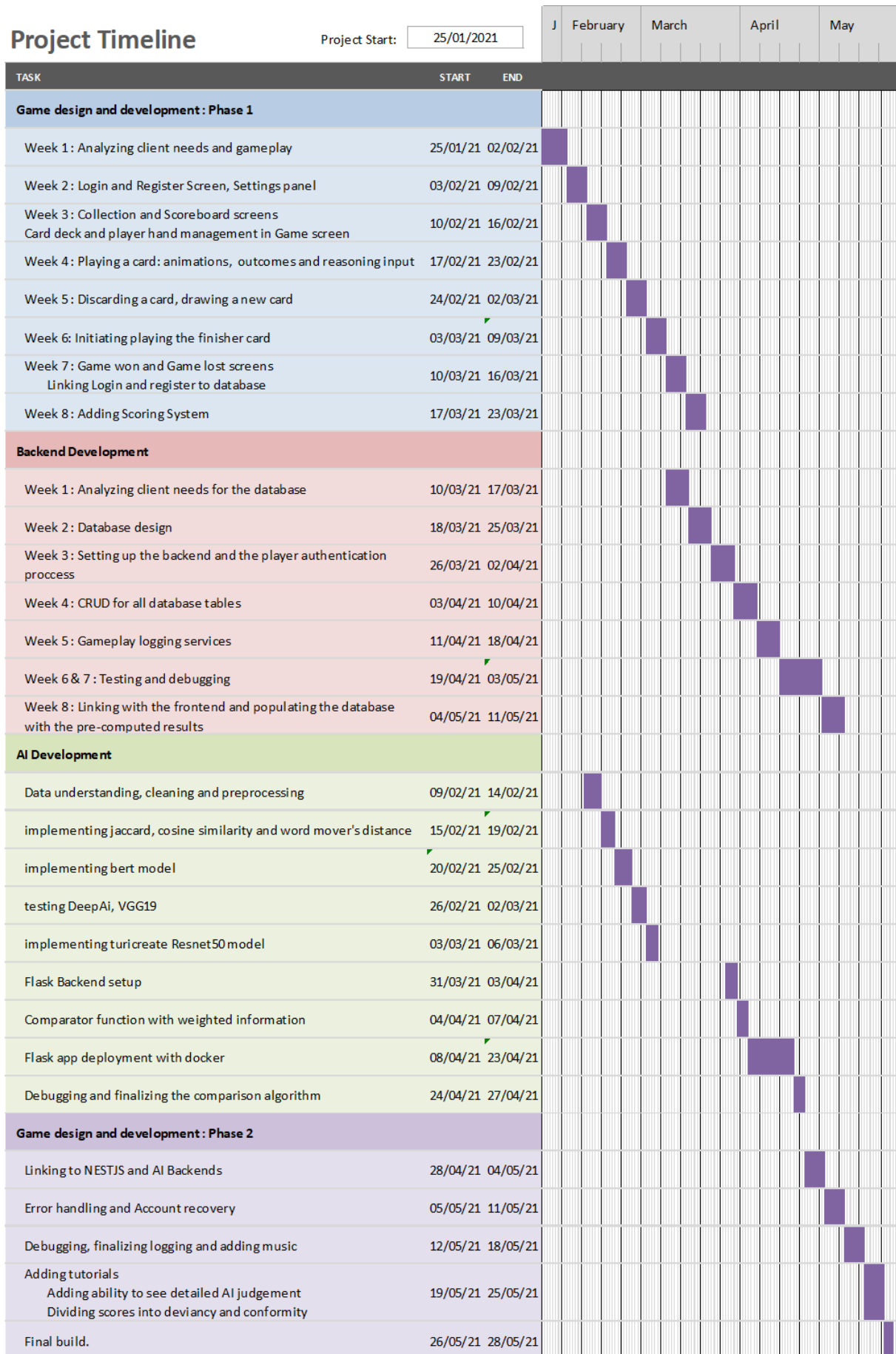


Figure 1.2: Gantt diagram of project timeline

# Chapter 2

## Game Design and Development

### 2.1 Introduction

In this chapter, we present the game development process.

We first describe the methodology we have used, scrum, which is based on agile principles of iteration and takes into account the possibility of changes in the customer requirements. Next, we present the functional requirements that are the starting point of our work. They can be separated into two categories: out of game and in game requirements. We mention scoring as part of our game design. Next, we go more in depth with descriptive diagrams detailing the game structure. Finally, we discuss the technologies used, as well as the non functional requirements.

### 2.2 Methodology

We have followed during the development of this game the scrum methodology by having a weekly meetup with clients and stakeholders. These meetings served the purpose of:

- Presenting the advancement to the clients. This included detailed presentations with recaptulations of the weekly work and demonstrations of the current state of the game.
- Getting the client's feedback on the work and what they felt should be changed or improved upon.

- Clarifying any blurry information. This included explaining our thought processes and asking for the Client's input and approval. The client would also be able to ask questions about the process.
- Going over the backlog and making sure deadlines were either met or rescheduled.

A backlog was agreed upon at the start, where priorities and criticalities were assigned with the Client's help. We followed this backlog in our development process.

We made sure to document our work as we coded. The client was provided with:

- An API documentation for the NestJS backend.
- An API documentation for the Flask AI backend.
- A documentation for each Jupyter notebook.
- Descriptions of each Unity script and what its purpose is.

During the last meetings, rather than demonstrations, the client was able to play the game and give live feedback as they experienced gameplay. This allowed them to point out things they liked and did not like.

The Clients were provided forms that they could fill out to give more in-depth feedback. They were questioned about the appearance of the game, as well as their experience playing.

The most important part was making sure they were having fun and were satisfied with the results. We would then use this feedback to make adjustments to the game, code and AI.

The semester concluded with a last meeting where everyone played the game and gave their last feedback. This feedback was duly noted down for the next team working on this project to use.

### **2.3 Functional Requirements**

The Unity program can be decomposed in two main parts:

On the one hand, the functionalities that go outside the game itself, for example authentication, signing up...

On the other hand, the functionalities that go into a playable game.



### 2.3.1 Out-of-game functionalities

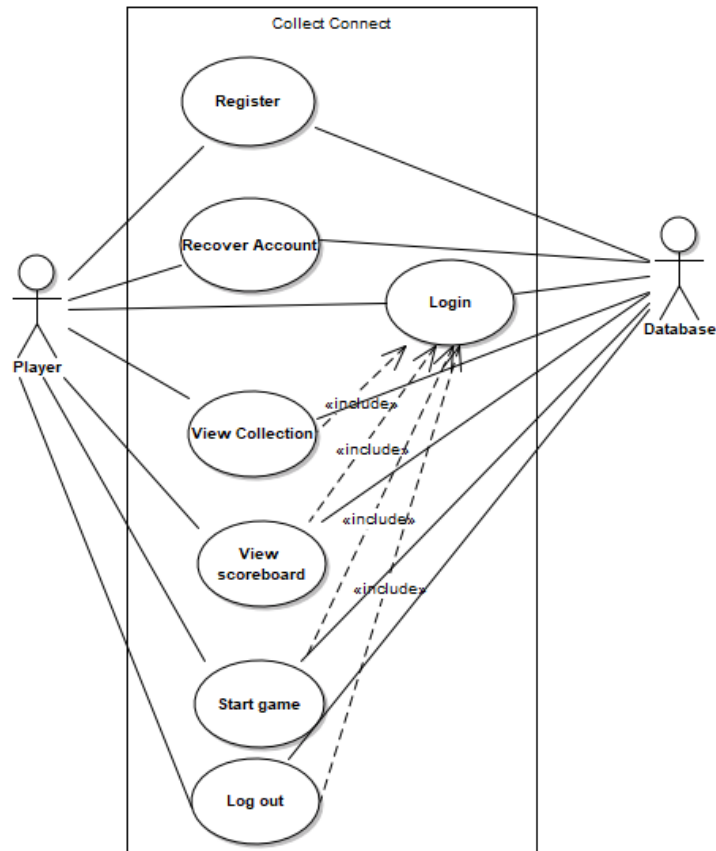


Figure 2.1: Out-of-game Use Case diagram

The player starts in the start screen. They will have to click to be redirected to the next screen. If the player has never been logged in before, they will be redirected to the Login Screen, where they can log in and go to the Home screen.

The player can choose to register instead. The player can choose to recover their account if they have changed devices.

These three screens are presented in the next figure. [2.2](#)

When the registering, login or recovery is complete, they are redirected to the Home screen.

If the player has logged into the game before, they will be automatically redirected to the home screen.

The home screen presents many options to the user. They can:

- View their scoreboard:

The scoreboard is composed of the player's previous scores. [2.4](#)

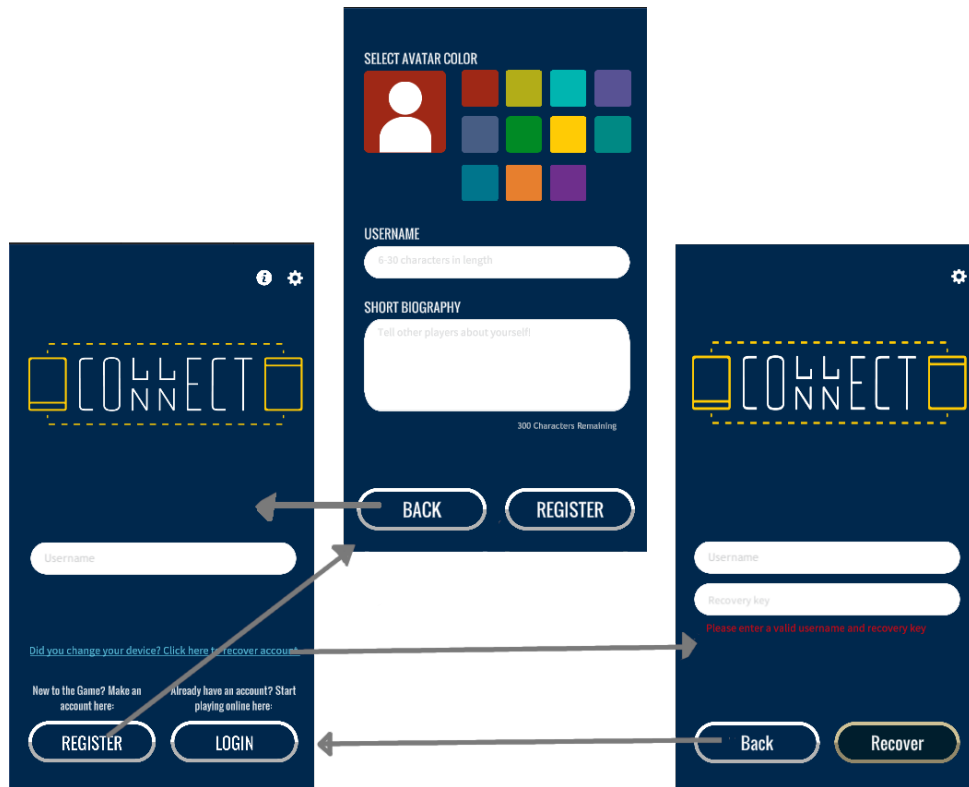


Figure 2.2: Login, Register and Recover screens.

- View their collection:

The collection contains cards that the player has won throughout the game. 2.4

- Log out.
- Start a game.

The player can, at any time, leave the game.

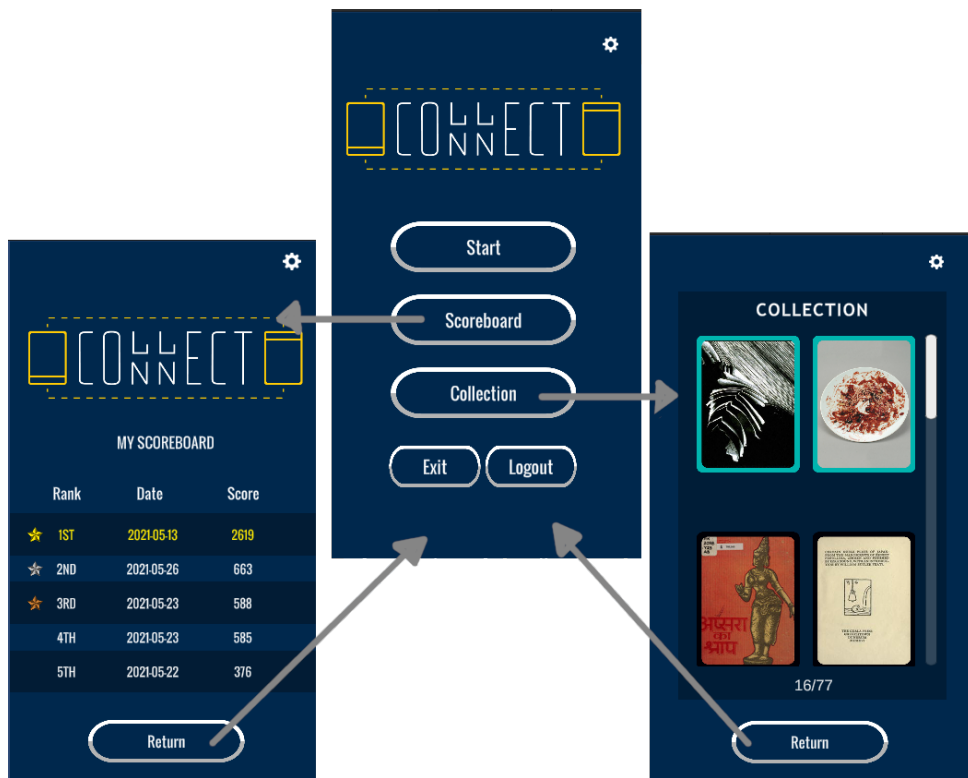


Figure 2.3: Home, Scoreboard and Collection screens.

### 2.3.2 In game functionalities

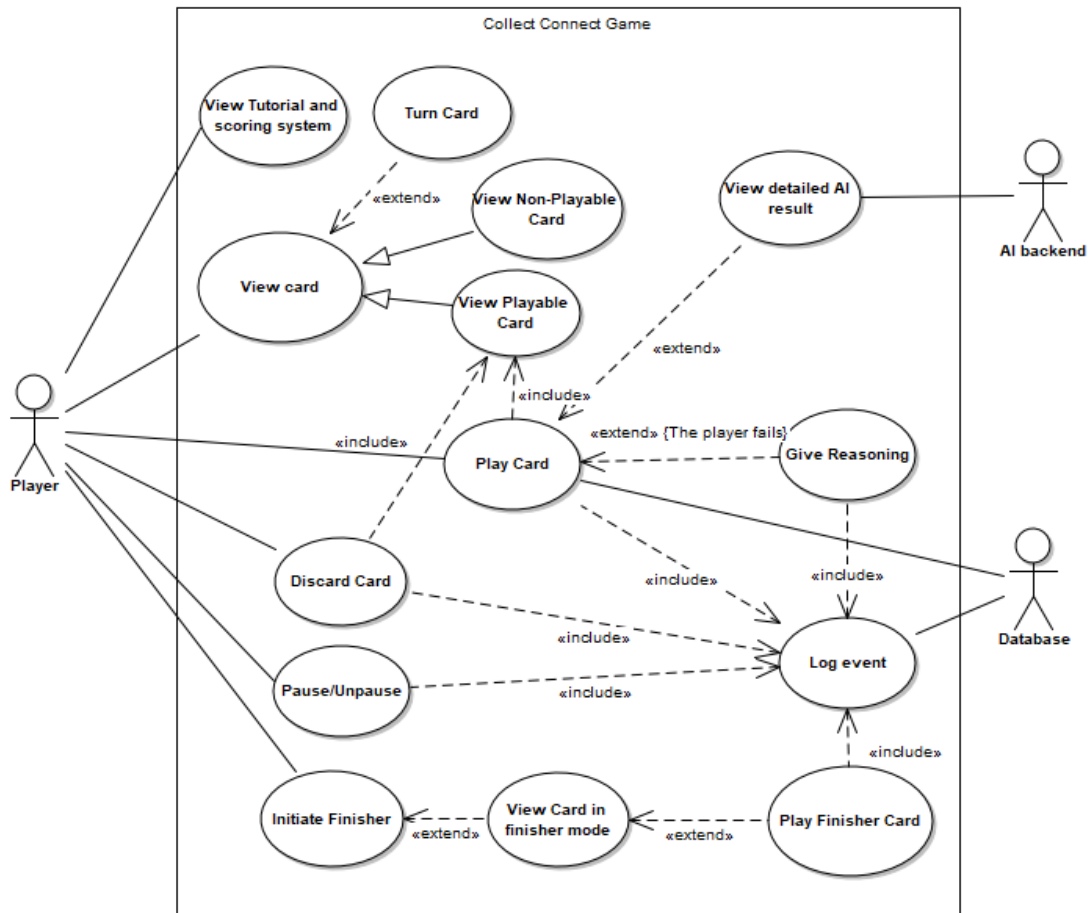


Figure 2.4: Gameplay Use Case diagram

The gameplay is as follows: The player starts a game and is presented a game table, or board. The board contains a hand of four playable cards at the bottom. They also see a start card and end card on the board in front of them. These cards are labeled accordingly as seen in figure 2.5

The top bar contains two scores, the conformity score (or how much the AI agrees with the player's choices), and the deviancy score (or how much the AI disagrees with the player's choices.). The player can click on the question marks to view the tutorial and scoring system explanations, as seen in figure 2.6.

The player can view a non-playable card: either the starting card, or the ending card. They can turn this card and view its back. 2.7

The player can also view a playable card: A card from their hand. 2.8



Figure 2.5: Game board

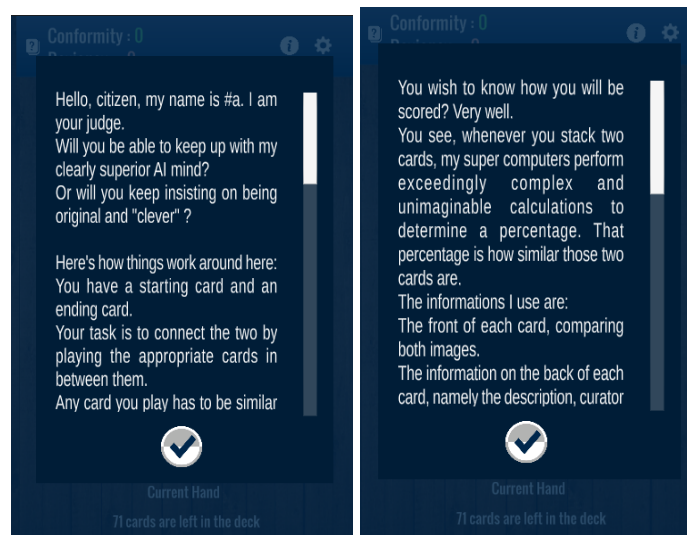


Figure 2.6: Tutorial screens and settings menu.

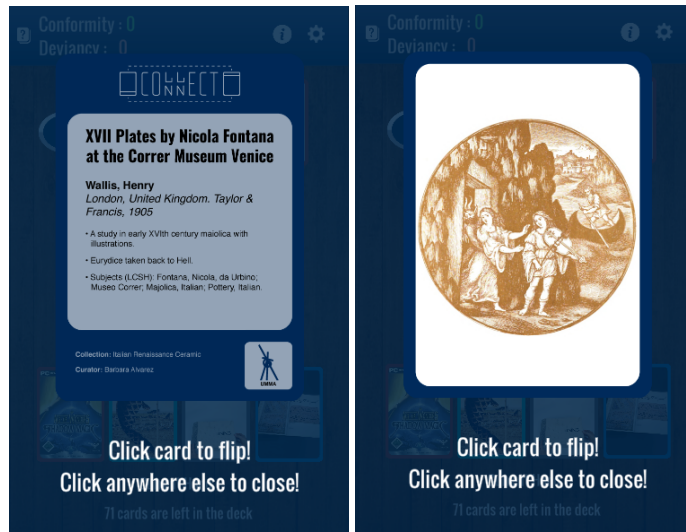


Figure 2.7: Non playable card, front and back.

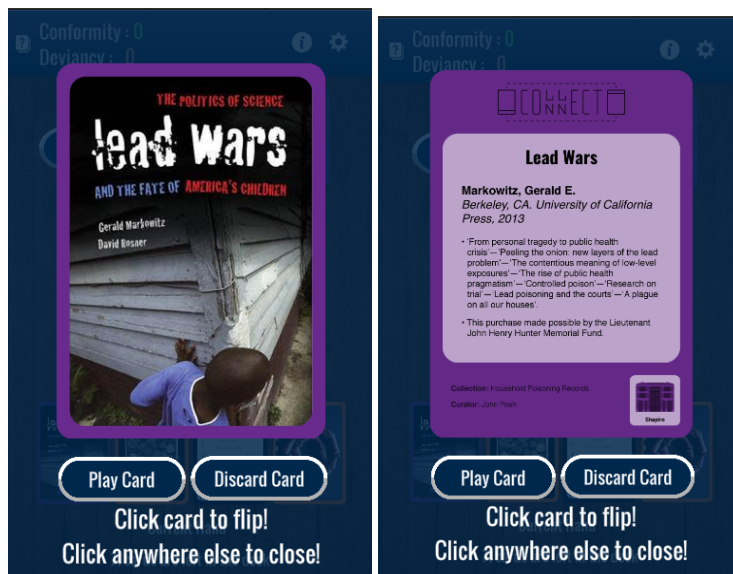


Figure 2.8: Playable card, front and back.

From there, the player can choose to discard a card that they do not like. The card will be replaced by a new one from the card deck.

From there, the player can also choose to play a card.

When they place a card, the AI will judge how relevant the link is. If it is relevant, the starting card is replaced with the played card. The player's conformity score augments. The player can view the detailed AI result. [2.9](#)

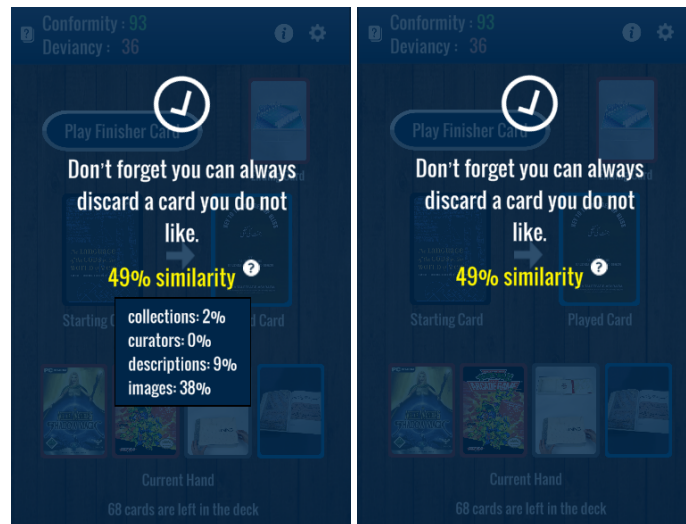


Figure 2.9: Accepted link screen

If it's not, the played card will be discarded. In that case, the deviancy score will augment. The player will be given the chance to give a reasoning, so they can contest the AI's judgement. [2.10](#)

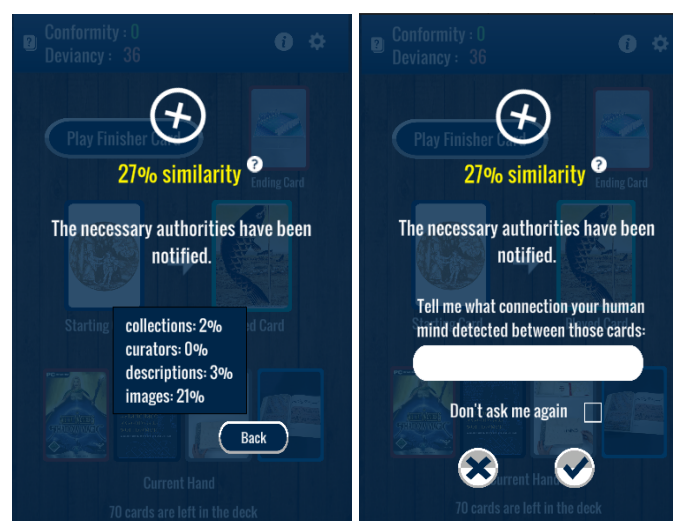


Figure 2.10: Rejected link screen

They can also pause the game, view the settings to control the volume or leave the game, as seen

in figure 2.11.

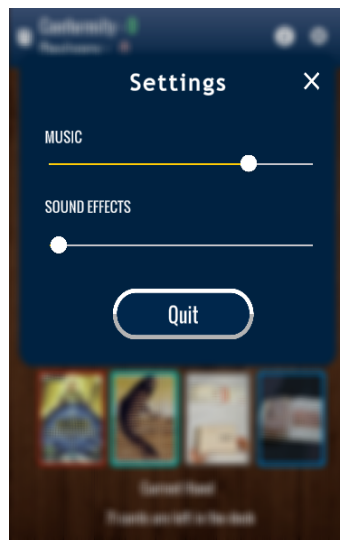


Figure 2.11: Pause and settings screen

The player can initiate their finishing move by clicking on the “Play Finisher Card” button.

After doing so, they are invited to choose the last card they will play. When viewing a card, they will be presented with the option to play the finisher card.

This card will be compared both against the top of the deck, and against the ending card. It is the game ending move. If both connections win, the game is won. In any other case, the game is lost. 2.12

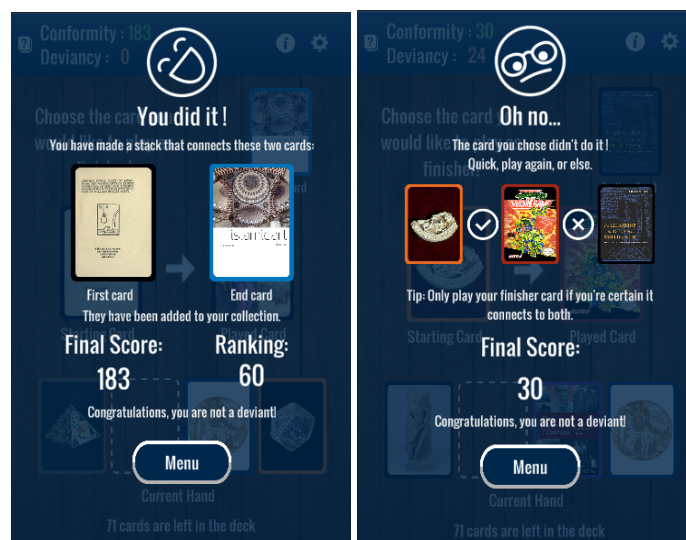


Figure 2.12: Final game screens

When a game is won, the starting card and the ending card get added to the player’s Collection. These cards are then able to be seen through the Collection Screen.



### 2.3.3 Scoring

When a player plays a card, they get awarded points depending on the AI's judgement. The AI returns a similarity score between two cards. The Conformity score is how much the player's decisions conform to the AI's decisions. The Deviancy score is how much the player's decisions deviate from the AI's decisions.

A link is accepted if the score is above a certain threshold, and it is rejected otherwise. The threshold is chosen manually and generally falls into the median value when sorting all possible scores. This ensures that at least half of all possible links are acceptable, so that players will eventually find one that works.

Normal moves:

If the score is  $\geq$  threshold then Conformity score += score

Else if the score is below the threshold then Deviancy score += threshold\*2 - score

This choice was made in order to balance the conformity score and deviancy scores. This way, we are inverting the given similarity score.

Finishing moves: If both connections (top of deck-played card and played card-ending card) win then Conformity score += score1 + score2 + 100

If losing game:

If both connections fail: then Deviancy score += threshold\*2 - score1 + threshold\*2 - score 2

If one wins and one fails then Deviancy score += failing score and Conformity score += winning score

### 2.3.4 Design

#### **Out-of-game functionalities: Screen Flow diagram**

The screen flow diagram [2.13](#) represents the movement of the player from one screen to another. It follows the flow described earlier in section [2.3.1](#).

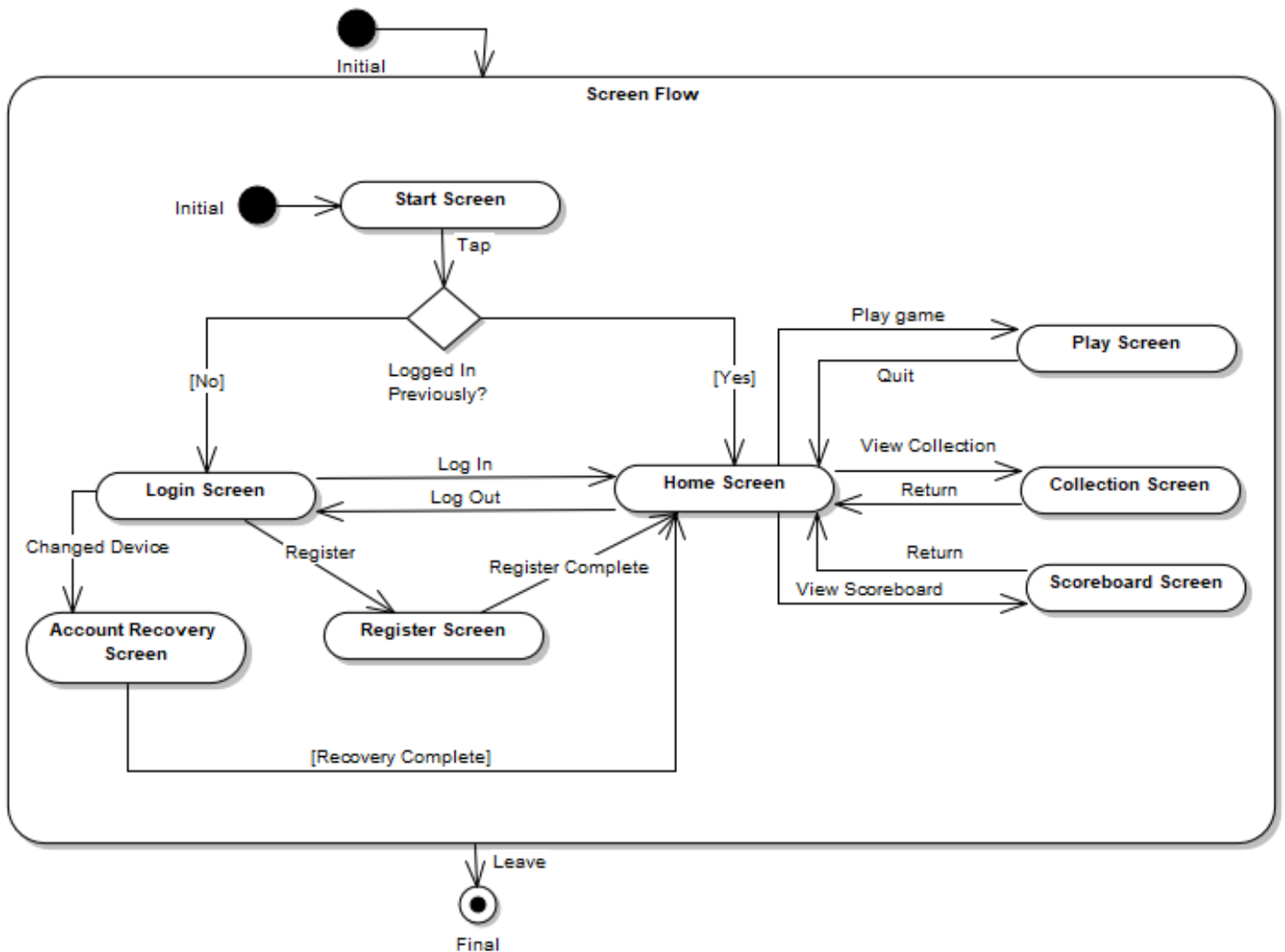


Figure 2.13: Screen Flow diagram for unity game

### In game functionalities: Class Diagram

The class diagram can be seen in figure 2.14

A Card is composed of two images, its back and its front. Images, in Unity, are contained inside a Sprite class, that also has the file name and other attributes. The game Board is composed of two sets of cards: Two non-playable cards and four playable cards.

The GameManager class is central to the functioning of the game. It contains all of the main game mechanics, such as displaying (through the DisplayPanel), discarding or playing a card. It also contains a reference to all objects on the Game scene, such as the board, its cards, the top bar... It is attached to the canvas game object.

Displaying a card happens through the DisplayPanel. The display panel displays one card at a

time, so it only has a reference to one card, through the Status class. The Status class keeps track of the currently selected Card, its position, the current AI algorithm version, and its threshold. It is static, which makes the data that it contains accessible from anywhere in the code. This ensures that the game has a consistent knowledge of its own status, which card is selected and which algorithm version it is running on.

The CardManager is used to keep track of cards. It contains the card history and the card deck. It takes care of creating the deck and dealing cards from it. The card deck is a list of cards, generated dynamically at the start of the game from all the card images that are shipped with the game assets. There is only one CardManager per game, because we only need to use one deck.

The ScoringSystem keeps track of the player's current score and takes care of any changes. There is only one ScoringSystem per game, because the player only has one pair of scores.

The Lines class stores all of the AI's one-liners and takes care of randomizing and looping them. There is only one Lines object per game, containing all possible lines.

The TutorialScreen class lets the player display and hide the tutorial panel. It is completely independent from other classes.

Any request to the backend or AI goes through the Request class. It helps sending GET and POST requests, to query for two cards' similarity, or log player behaviour.

The AIResultDatabase and AIResult classes are the model of the information returned from the database and AI respectively, when queried for card similarity.

The Account class takes care of anything that concerns user authentication. For example, logging in and out, registering, recovering an account, etc...

The Account\_CollectConnectServerResults class is the model of the server response to logging in or registering.

The UI class is used to interpret a user's avatar colour from string to Unity Colour.

The Scoreboard takes care of the scoreboard screen, querying the database for a player's previous scores and displaying them.

The Collection takes care of the Collection screen, querying the database for a player's collected cards and displaying them.

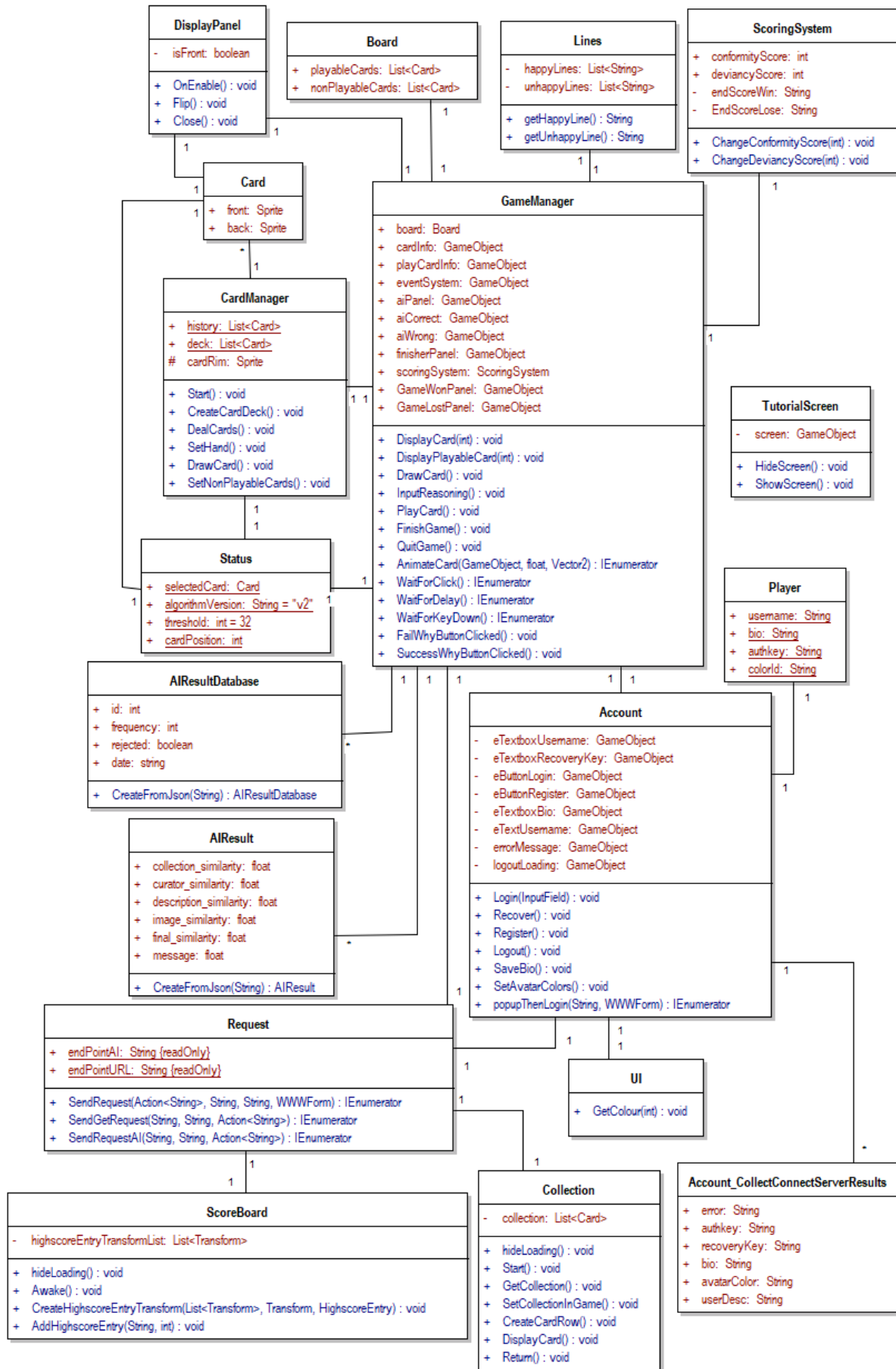


Figure 2.14: Software class diagram

## 2.4 Notions and Technologies

The game engine “Unity” was used in the creation of this game. It is a highly flexible 2D and 3D engine for game-making.

It relies both on its graphical interface, where graphical game objects are placed into a scene view, as well as on written C# scripts. These scripts are attached to game objects as components. They are what makes a user able to interact with a game, as well as game objects interact with each other. They define behaviour for game objects.

Unity also lets us communicate with external API, which are crucial to the functioning of the game currently. Two API are at work here:

- A NestJS API connected to a database in which information will be logged. The game sends requests to this API at each player move, so any game can be recreated. This is useful for researching player behaviour and the connections they make between cards.
- A Flask application that lets us communicate with the game AI.

The separation of these functionalities into two different API ensures that if one fails, the other will be able to back it up. The AI container is also very slow to build (20 minutes), accounting for its size (3G). This contrasts with the NestJS backend, that only takes 5 minutes and is significantly more lightweight.

## 2.5 Non Functional Requirements

### 2.5.1 Availability and Performance

To ensure a smooth playing experience, it was important to take into account the fact that there could be network problems. We have decided that in the case of a logging error, the player would be able to keep playing. All logging would be stopped. This would lower the logging overhead, as well as ensure that the database stays coherent and manageable. (If a turn is lost, logging the next steps becomes tricky). Availability and performance were also ensured with the backup system in place: All of the card similarities exist on the database for each pair of cards. The game always contacts the database first, because information retrieval ( 1s) is much faster than

computing the similarity again ( 4s). If the database fails for any reason, we go through the AI and compute the similarity from scratch.

### **2.5.2 Portability**

The choice of technologies (Unity game with two restful backends) means that the game can be built toward multiple platforms. Indeed, one of the reasons Unity is a go-to engine for game creation is that games that are developed with it are able to be built toward computers as well as phones. Notably, iOS, Android, MacOS and Windows.

## **2.6 Conclusion**

In this chapter, we have discussed in great detail the way the game was designed and implemented. The game would however hardly work without the support of its NestJS backend and AI. We will discuss these next.

# Chapter 3

## Database and API

### 3.1 Introduction

Collect Connect is as much of a game as it is an experiment. Through it, we hope to collect different types of data on the behaviour of the users, as well as crowdsource the connections between the different artwork, in the hopes of finding interesting connections to feed the Translation Networks work, as well as making the AI better in the future.

A solid infrastructure was needed to make sure this data could be saved and easily retrieved.

Another need was for something to act as a backup to the AI. Pre-computing results and saving them to a database would mean faster and more reliable access to the information. This was preferable to live-computation, which would take longer and might fail.

To this end, we created a database schema that would be able to help study user behaviour and thought processes. We then built a NestJS backend on top to be able to access this database.

We then deployed this NestJS backend through Docker. This backend now lives on an apache server image.

## 3.2 Database

### 3.2.1 Purpose

The database has 5 main purposes :

- Saving the players' account data : We needed to save the player's account information along with their collection of saved cards and the avatar color that they chose.
- Saving the AI judge results : Having saved all the cards (their names, their curators and the collection that they are a part of) in the database, we then compute all the possible two-card combinations in order to pre-compute the AI's judgement of the similarity of these pairs.
- Saving the players' reasonings for playing certain cards together : The point of this is to prepare the ground for implementing a reinforcement learning algorithm. The algorithm would take into consideration the frequency of two cards being played together and the reasonings given by the players in order to learn to accept the links that humans found but were previously missed by the AI judge.
- Logging the player's gameplay for research purposes : The client requested that we save the entirety of the gameplay and provided us with a list of questions that they needed to be able to query the database for the answers to, for their research. The following is the list of said questions :
  - Questions concerning the game:
    - \* What is the player's name, score and have they won the game ?
    - \* What is the game's start time, end time and duration ?
    - \* What are the cards in the player's collection ?
    - \* How many pauses did the player take during the game and how long did they last ?



- \* How many turns are taken before a game is concluded ?
- \* How many times did a player choose to look through the deck ?

– Questions concerning a specific turn in a game:

- \* What are the cards in the player's hand at the beginning of the turn ?
- \* What is the turn's starting card ?
- \* What is the turn's ending card ?
- \* What was the turn's played card ?
- \* How many cards did the player draw during a turn ?
- \* What were the drawn cards during a turn ?
- \* When did the turn start/end and how long did it last ?
- \* Was the 'play finisher card' button pushed during a turn ?
- \* How many actions (play/draw card) did the player make before pushing the connect button ?
- \* Was the played card of a turn accepted or rejected ?
- \* How many points did the player gain from a turn ?
- \* How many cards were browsed in a turn ?

– Questions concerning rejected card links :

- \* What are the cards in the player's hand at the beginning of the turn ?
- \* What is the most frequently chosen reasoning for a given card pair ?
- \* What are the most common reasonings for a failed link ?

### 3.2.2 Design

#### Simplified schema

After studying the main functions of our database we came with the following design (3.1) showcasing the different database entities and the “has n” type relationships between them.

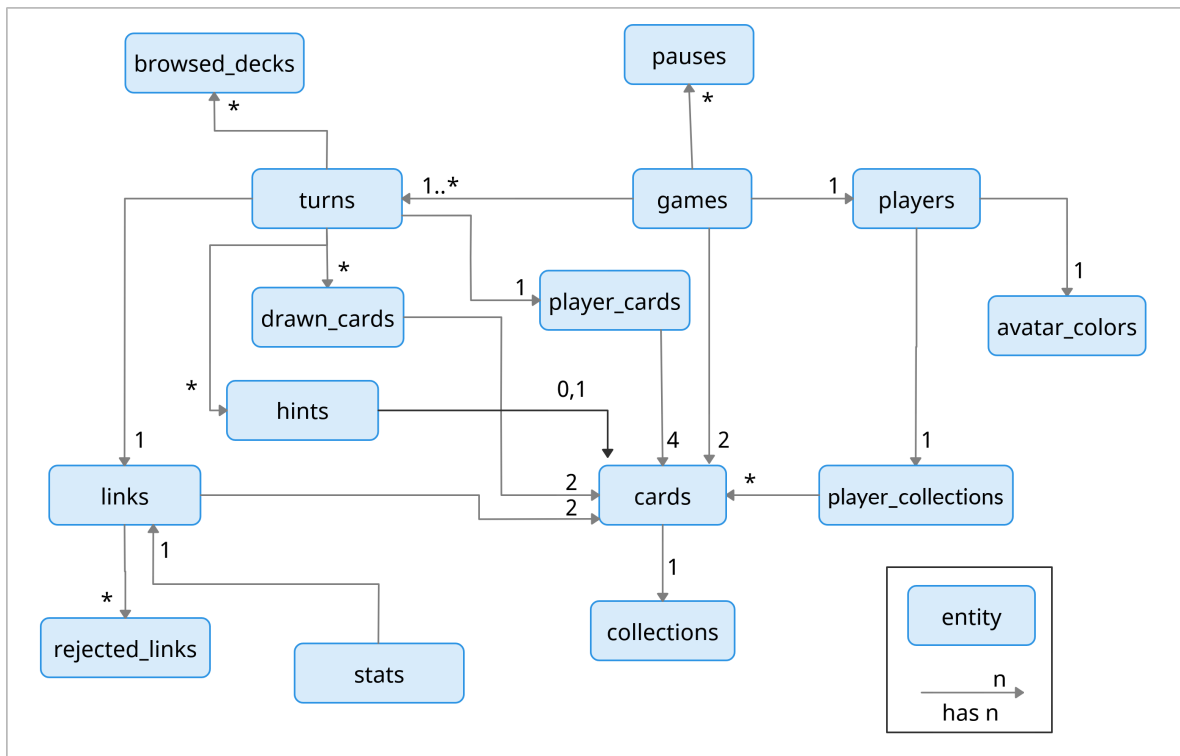


Figure 3.1: Database simplified schema

Starting off with the “players” table containing the players login information, a player has one entry in the “avatar\_color” table and one entry in the “player\_collection” table.

The “player\_collection” table is used to save the cards that the players kept in their collections. A player’s collection is composed of zero or more entries in the “cards” table.

The “cards” table has information regarding the cards (name, curators, collection...). Each “cards” entry has one associated entry in the “collections” table which contains all card collections.

The “pauses” table is for saving when the players pause and resume their games.

The “games” table saves data concerning the games such as the player, the start time, end time, score, whether the game was won or not...

A game is played by one player, which means that each “games” entry would have one associated entry in the “players” table. Every entry would also have one or more entries in the “turns” table since a game is composed of multiple turns and zero or more entries in the “pauses” table. Every game has one entry in the “cards” table that represents the end card and another one that represents the start card.

The “player\_cards” table is for keeping track of the players’ hands during each turn. Every item in this table is composed of four items in the “cards” table.

The “drawn\_cards” table keeps track of the cards that the player drew and discarded during a turn. It has two entries in the “cards” table : a discarded card and a drawn card.

The “hints” table is for saving the hints that were given to the player. A hint can either consist of highlighting a playable card or displaying the text “draw a new card”. It can have zero or one entry in the “cards” table: if there isn’t a card in the hint entry, then the hint was “draw a new card”.

Note: Hints were an additional feature that we set the grounds for in the backend but did not end up implementing in the game.

“rejected\_links” contains the reasoning that the player gives for two cards that were rejected by the AI judge.

The “links” table contains all the possible links that can be made. It has two “cards” entries and zero or more rejected\_links (to save the reasoning inputted by the player for that specific link) Every entry in the “stats” table contains one link and its corresponding AI judgement.

The “browsed\_decks” table contains the number of cards that they player browsed during a turn.

Note: This was a feature that we prepared for in the backend but ended up not implementing in the frontend.

The “turns” table saves all the data concerning a specific turn (start time, end time, points...). It has one link (meaning the turn ends when a card is played) zero or more “drawn\_cards” entries and one “player\_cards” entry. During one turn, a player can ask for hints which means that a turn has zero or more “hints” entries associated with it .

### Detailed schema

We chose to set up a MySQL database since we are obviously dealing with relational data and because the client already had a MySQL database set in place for the multiplayer version of our game and wanted to eventually merge ours with it.

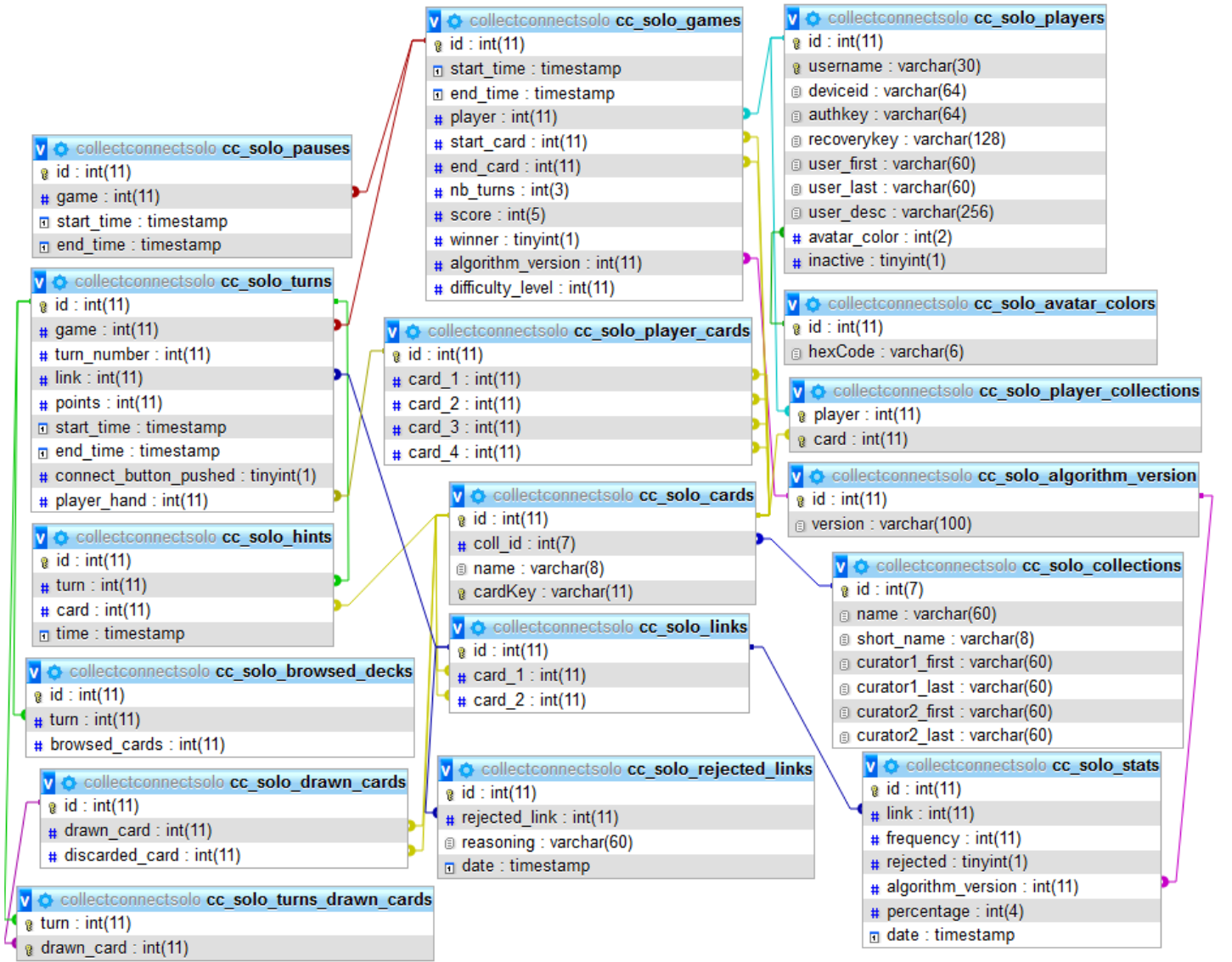


Figure 3.2: Database detailed schema

This schema is the implementation of the simplified design. It has all the previously listed tables along with the necessary junction tables needed for implementing Many-to-Many relationships between entities.

Here the “cc\_solo\_algorithm\_version” table was added to save the different AI algorithm versions in order to keep track of the changing judgements.

## 3.3 API

### 3.3.1 Purpose

The primary need for the NestJS API is to act as a backend and logging tool for the solo player version of Collect/Connect.

It has 3 main tasks :

- Managing players' accounts:
  - Signing up
  - Logging in
  - Logging out
  - Recovering account
  - Generating a new recovery key
  - Setting/Getting the player's bio
  - Setting/Getting the player's avatar color
  - Getting the player's collection
  - Getting the player's highscores
- Logging gameplays:
  - Starting a new game
  - Playing a card
  - Discarding a card
  - Pausing/Resuming the game
  - Saving a reasoning for a rejected link.
  - Quitting the game
- Saving AI judge results:

- This is done with the purpose of time optimization: the game checks if the results are already in the database (which takes about 1 second) before running the AI judge (which takes around 4 seconds).

### 3.3.2 Design

Code wise, we opted for using the microframework nestjsx/crud in order to automatically generate CRUD controller endpoints for most of the database tables. We then had to create a service for each database entity as the nestjsx/crud framework required.

The Many-to-Many relations are not handled by the microframework so we had to implement the CRUD operations for them manually.

The API is divided into 3 main modules.

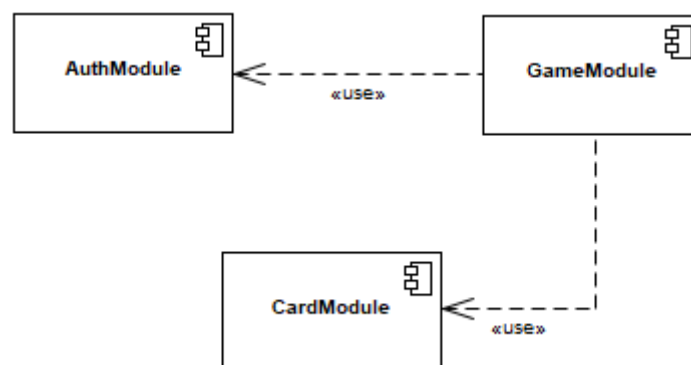


Figure 3.3: API modules diagram

#### Auth Module

This module manages the player authentication (register, login...), accounts (get new recovery key, recover account...) and profile personalization (change player bio, change player avatar-color...).

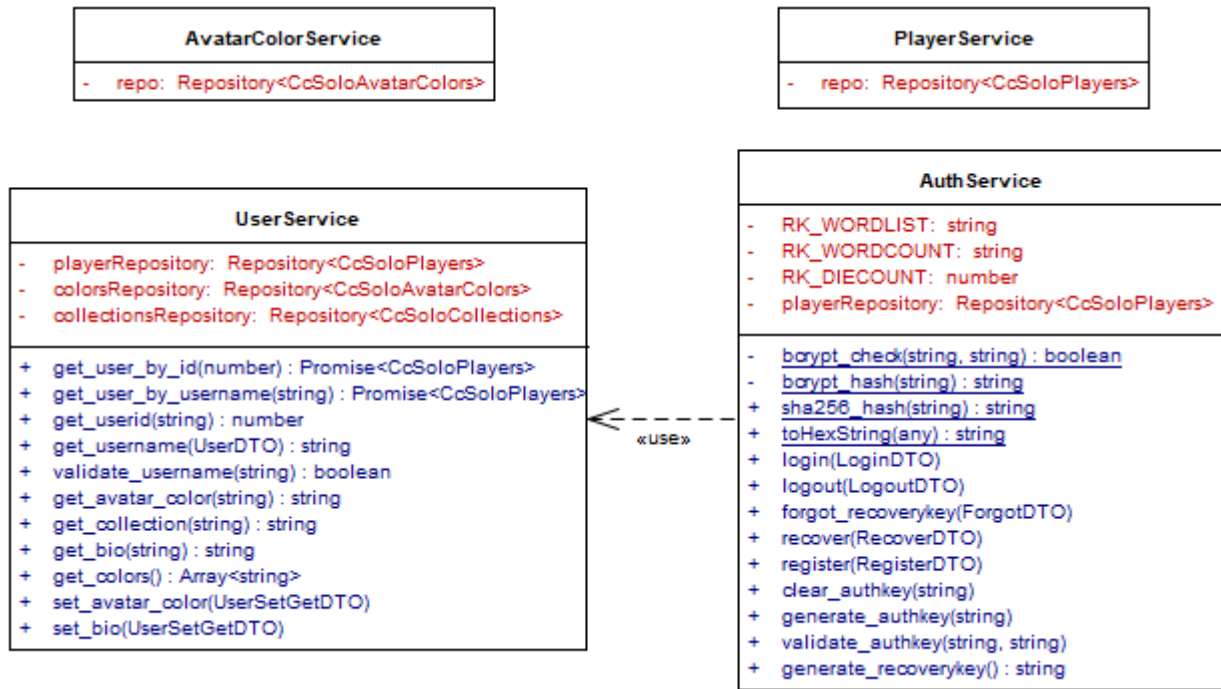


Figure 3.4: AuthModule services class diagram

We used bcrypt hashing to safely save the recovery keys and sha256 hashing for the authentication keys.

Although AvatarColorService and PlayerService don't have any functions, their existence is required by the nestjsx/crud microframework.

UserService handles account management.

AuthService handles all authentication needs and uses some of the functions provided by UserService.

## Card Module

This module handles everything card related, from saving card details (name, collection...) to links and stats (AI judge comparison results).

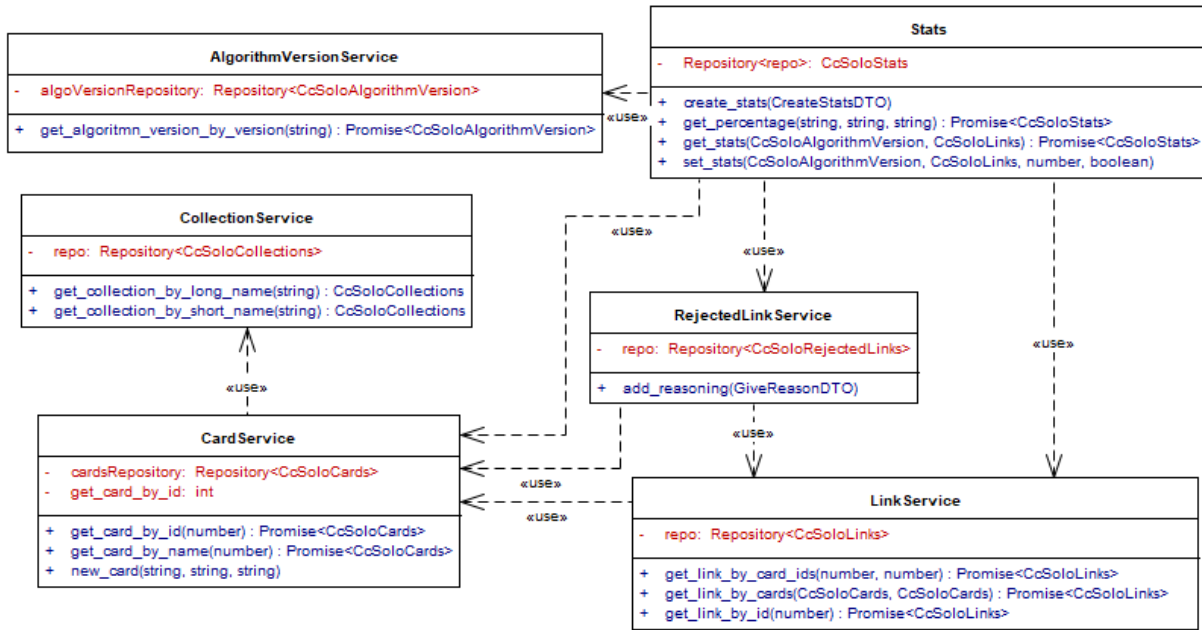


Figure 3.5: CardModule services class diagram

CardService and CollectionService handle getting the cards and collections. StatsService, LinkService and RejectedLinkService handle the links and AI judge comparison results.

### Game Module

This module is responsible for logging the entirety of the gameplay.



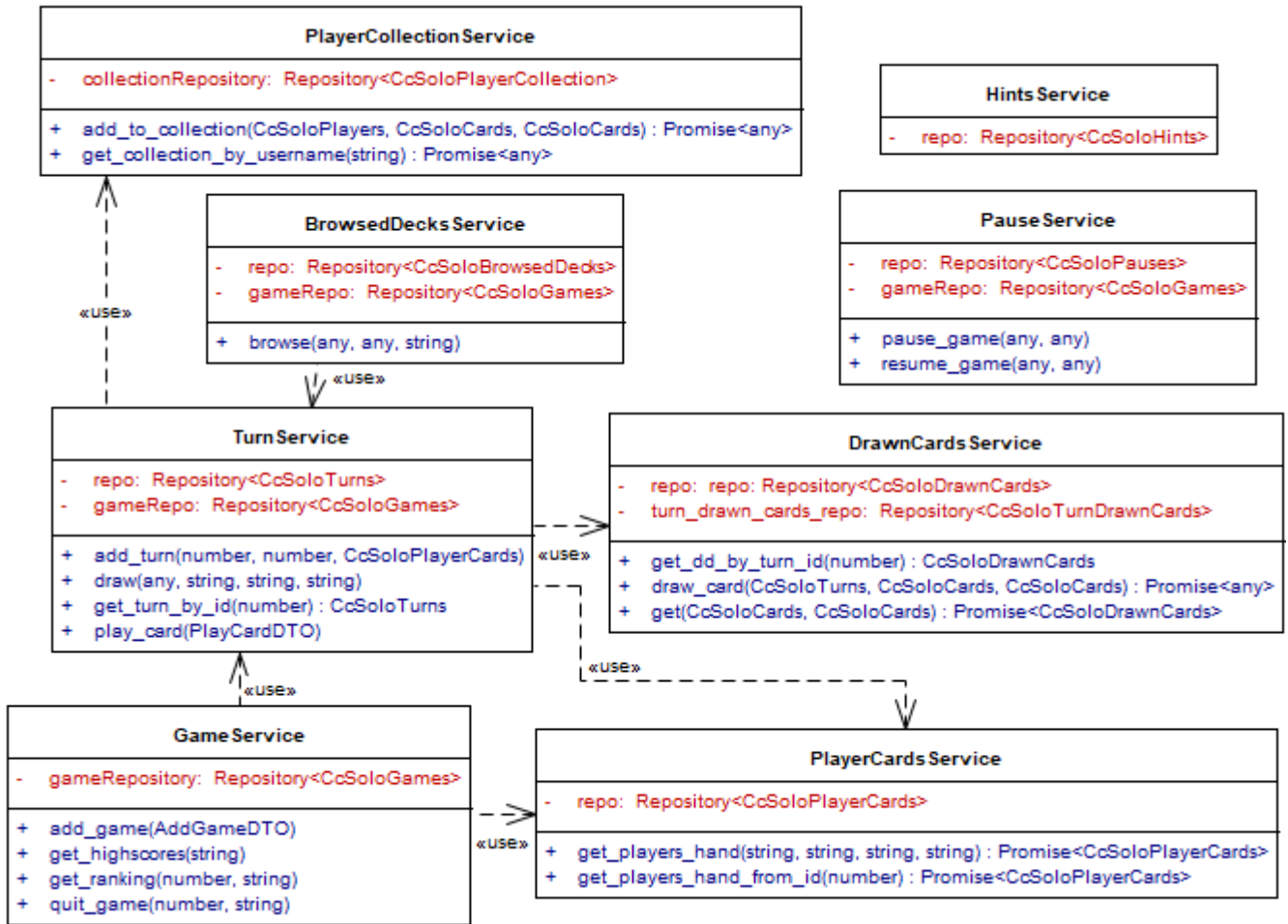


Figure 3.6: GameModule services class diagram

Although HintsService doesn't have any functions, its existence is required by the nestjsx/crud microframework.

### 3.3.3 Access levels

There are 3 different levels of access to this API:

- Public access : Anyone can send requests and get the desired responses with no need for a token or authentication key.
- Player access : Only players can get the desired response from the API since it asks for a username and the corresponding authentication key.
- Admin access : Only someone who has the admin token can get the desired responses.

**What is an authentication key ?** An authentication key is a randomly generated string of

characters that is assigned to a player when they log in and is then used by this player to send requests to the API.

A hashed version of the key is saved in the database using the sha256 hashing algorithm.

**What is an admin token ?** An admin token is a secret server environment variable that allows whoever has it to perform CRUD (Create-Read-Update-Delete) operations on all database tables.

### 3.3.4 API endpoints

#### Public access

These are some of the actions that can be performed by anyone.

#### Register:

When registering, the player has to provide a unique username and their device ID.

The device ID is used as a password for the account, which means that a player can only connect to that account from that specific device.

The recovery key contained in the response is a random set of words that is hashed and saved in the database using the bcrypt algorithm.

This key allows the player to move their account from one device to another.

Endpoint	/auth/register
Method	POST

Table 3.1: Register endpoint

#### Recover Account :

If a player wants to recover their account, they must provide their recovery key along with their username and the device id of the device that they're moving their account to.

Endpoint	/auth/recover
Method	POST

Table 3.2: Recover endpoint

#### Get Percentage :

After the AI judge judges the link between 2 cards, this request gets the saved result from the

database.

Endpoint	/stats/percentage/:algorithm_version/:card_1/:card_2
Method	GET

Table 3.3: Get percentage endpoint

### Login :

When logging in, the player has to provide their username and their device ID. The API then returns an authentication key that the player can use for requests that require player access.

Endpoint	/auth/login
Method	POST

Table 3.4: Login endpoint

### Player Access

This level of access requires that the player sends their authentication key (that they acquired after logging in) in the request headers.

At this level, there is a long list of actions that can be performed including account management and gameplay logging.

Following are some examples of actions that can be performed by players :

#### Start a new game:

Endpoint	/games/new-game
Method	POST

Table 3.5: New game endpoint

#### Quit a game :

Endpoint	/games/:game/quit
Method	POST

Table 3.6: Quit game endpoint

#### Play a card :

If the players played a finisher card, then they must include in the request body 2 extra elements: `end_rejected` and `end_percentage` which represent whether the link with the end card

was rejected or not, and the percentage given by the AI judge.

In the case of a finisher card, the API returns as a JSON object containing a game and a turn object. The turn object represents the final turn in the game.

If the AI judge result sent in this request body (percentage and rejected) is not already saved in the database (stats table) then a new entry is created and saved containing these results

Endpoint	/turns/play-card
Method	POST

Table 3.7: Play card endpoint

### **Admin Access**

The admin access allows complete database access along with a few other actions. The upcoming requests require that the admin token is in the headers as “admin-token”.

- CRUD Operations :

The microframework nestjsx/crud was included in this project in order to automatically generate the CRUD services and controller endpoints.

- Other Actions :

### **Add percentage :**

This adds a new Ai judge result to the database.

This request is used for seeding the database with pre-computed AI judge results.

Endpoint	/stats/add
Method	POST

Table 3.8: Add percentage endpoint

### **Get collection or card by name :**

It is possible to get more information on a card or collection by calling these endpoints:

Endpoint	/collections/by-long-name/:name
Method	GET

Table 3.9: Get collection by name

Endpoint	/cards/card-by-name/:name
Method	GET

Table 3.10: Get card by name endpoint

**Add new card :**

Endpoint	/cards/new-card
Method	POST

Table 3.11: Add new card endpoint

## 3.4 Non-functional requirements

### 3.4.1 Portability :

The choice of technologies that we made (NestJS and MySQL) makes our work portable i.e. easily transferable from one machine/system/container to another. And the fact that everything was later on packaged in a container makes it even more portable.

### 3.4.2 Documentation :

We made sure to write long and detailed technical documentations for the database and the API in order to make it easier for the next team working on our project to pick up where we left off.

### 3.4.3 Scalability and Availability :

The client's choice to use containers to deploy our API made it easily scalable as they would just need to duplicate the container if the need ever rises. It also contributed to its availability since containers can easily be restarted if an issue comes up.

## 3.5 Conclusion

In this chapter, we presented the database design and explained our reasoning for it. We also presented the corresponding API and its main functions. Finally, we talked about pre-computing and saving the AI judge comparison results in the database.

In the upcoming chapter, we will talk in depth about the AI judge and how it judges the links between cards.

# Chapter 4

## AI Judge

The AI is the beating heart of this game. It is as much of a game mechanic as it is a character, responding to the player's input and comparing cards before giving its final judgement. It is therefore a judge. The AI is tasked with taking all of the information concerning two cards, and scoring the similarity between them on a scale of 0 to 100, 0 being completely dissimilar and 100 being identical.

There are two types of data concerning each card: textual data and visual data.

The textual data is the description, collection and curator.

The visual data is the card image, which shows the object the card represents. We have thus chosen to work with NLP to compute the similarity between the textual datapoints, and image similarity with CNNs for the images.

These two approaches are combined into one comparison function that outputs the final similarity.

### 4.1 Methodology

The methodology used during this project is CRISP-DM [10]. (CRoss-Industry Standard Process for Data Mining) which is an open standard process model used for data mining and predictive analytics released by IBM in 1999. this methodology is 6 steps process:

- Business Understanding: during this step we try to focus on understanding the project

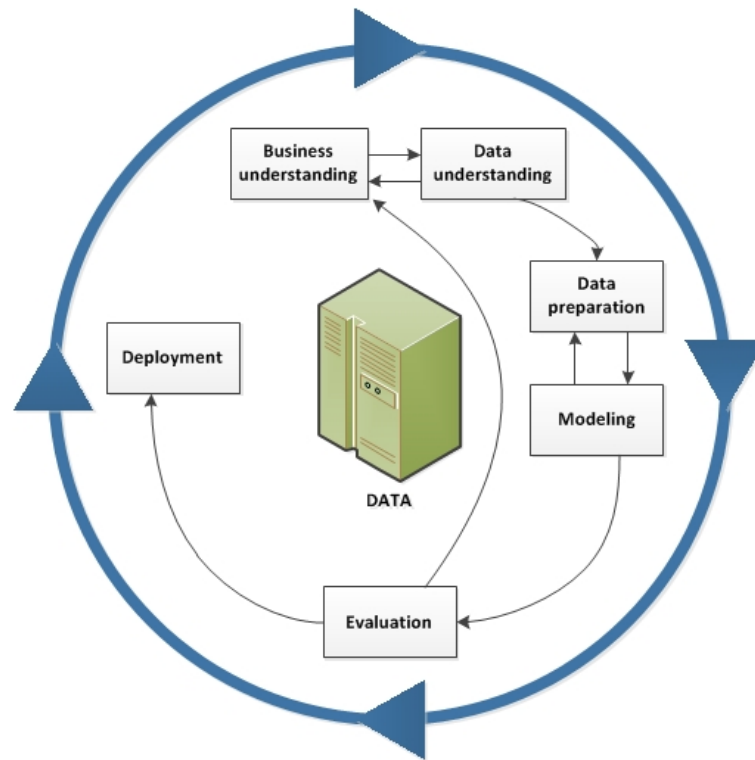


Figure 4.1: Crisp-DM methodology[2]

from a business perspective, determine business objectives and produce a project plan.

- Data Understanding: we then focus on acquiring the data, examining it, digging deeper into it and cleaning it.
- Data Preparation: this phase prepares the data for modeling: we select data, clean it and integrate it.
- Modeling: during this phase we determine which model to use, build it and assess it. We iterate model building and assessment until we find the model that gives us the best results.
- Evaluation: we evaluate results, review the process and determine the next steps.
- Deployment: we focus during this phase on deploying the model and plan its monitoring and maintenance.



## 4.2 Dataset

The dataset we worked with contains 77 instances as of now. However, it can easily be replaced by a larger dataset in the future. For each instance we will be using the description, curator, collection and image to compute the final similarity between two cards. The definition of each data point is as follows:

**Description:** small paragraph that is placed at the back of the card to help the player get more information about the card. **Curator:** the person in charge of collecting art pieces and creating card collections. **Collection:** the collection that the card belongs to **Image:** the image at the front of the card.

In order to compare two cards we need to compare different card features. We have two types of features: text and images. Text typed features are: description, curator and collection with which we will be using the NLP approach. Images typed features are: the card images with which we will be using a computer vision approach.

## 4.3 Natural language processing

Natural Language Processing is an artificial intelligence subfield where computers are able to process, understand and analyze human language. It's a way that helps computers to communicate with humans and be able to read text, hear speech, recognize sentiments and determine the important parts of a text.

### 4.3.1 Text preprocessing

**Removing stopwords:** Stopwords [4] are the most common words used in any language that don't add much to the meaning of the sentence, such as "the", "is", "at", "which", etc... We have decided to use nltk so we can filter them out. nltk.corpus provides a list of english stop words we can use in our AI.

**Stemming words:** Stemming [11] a word is reducing inflected words to their root form, it is an important part of text preprocessing and it can make a significant difference in the model output.

### 4.3.2 Model implementaion:

To be able to figure out which model was the best fitted to our needs, we have gone through different experiments such as:

- Calculating the similarity with and without word stemming
- Calculating the similarity with and without preprocessing
- Calculating the similarity with TextRank
- Calculating the similarity using cosine similarity
- Calculating the similarity using Bert
- Calculating the similarity using Jaccard similarity
- Calculating the similarity using words mover distance

Below is a definition of each model we tried:

**TextRank [8]:** is a keyword extraction and text summarization algorithm

**Cosine similarity:** between two words: each word is represented by a vector and the cosine similarity of the words is the cosine of the angle between the two non null vectors, it's output range is in  $[0,1]$  where 0 is the least similar and 1 is the highest similar. the formula is  $\text{Similarity} = (A.B) / (||A|| \cdot ||B||)$  where A and B are vectors.

**Jaccard similarity:** defined as the intersection between two sentences divided by the intersection of the two sentences which are the common words between the two. The mathematical formula is:

$$\begin{aligned} J(doc_1, doc_2) &= \frac{\{'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'\} \cap \{'data', 'is', 'a', 'new', 'oil'\}}{\{'data', 'is', 'the', 'new', 'oil', 'of', 'digital', 'economy'\} \cup \{'data', 'is', 'a', 'new', 'oil'\}} \\ &= \frac{\{'data', 'is', 'new', 'oil'\}}{\{'data', 'a', 'of', 'is', 'economy', 'the', 'new', 'digital', 'oil'\}} \end{aligned}$$

Figure 4.2: Jaccard similarity formula

**Words Mover distance:** It measures the semantic similarity between two sentences using word2vec embeddings. The euclidean distance is used to calculate the distance between two words and then it is summed up to get the WM distance.

**Bert:** Bidirectional Encoder Representations from Transformers[9] Bert model developed by Google, it learns contextual relations between words (or sub-words) in a text and outputs the similarity between the two words that ranges between 0 and 5. As opposed to directional models, which read the text input sequentially (left-to-right or right-to-left), the Transformer encoder reads the entire sequence of words at once. Therefore it is considered bidirectional. This characteristic allows the model to learn the context of a word based on all of its surroundings.

**Example of description comparisons:**

Card name	Card description
Cleanliness is Next to Godliness	The phrase, "Cleanliness is next to godliness," is first attributed to the 18th-century theologian John Wesley. Here we see two ideas, order and nature, now interpreted and juxtaposed in the recesses of our modern minds with absurd results. Has the awesome grandeur of nature somehow increased, or have our hygienic rituals descended into meaninglessness?
Apsara Warrior	Cambodian artist Ouk Chim Vichet's "Apsara Warrior," on display at the U-M Museum of Art, originated from the Peace Arts Project Cambodia, designed to promote both nonviolence and young Cambodian artists. The piece was donated to the museum by Guy and Nora Barron.
The Kādambarī of Bāna	comic book
Certain noble plays of Japan	Three hundred and fifty copies of this book have been printed

Table 4.1: Card names and their corresponding descriptions

The first pair of cards are not related based on the descriptions. The second pair is similar since both cards are about books. 4.1

Card1	Cleanliness is Next to Godliness	The Kādambarī of Bāna
Card2	Apsara Warrior	Certain noble plays of Japan
Cosine similarity	0.0	0.235
Jaccard similarity	0.0	0.1
World's mover distance	3.06	2.586
Cosine similarity with textRank keyword extraction	0.88	0.46
Jaccard similarity with textRank keyword extraction	0.818	0.21
World's mover distance with textRank keyword extraction	0.79	0.64
Bert model without word stemming	0.533	0.7641
Bert model without preprocessing	0.334	1.449

Table 4.2: Cards name and their corresponding descriptions

- For cosine and jaccard similarity without keyword extraction we don't see big difference in results between two related cards and two non related cards. For world's mover distance with and without we see that the distance between the first and the second pair but results on various other paires weren't convenient.
- For cosine and jaccard similarity with keyword extraction we see that the first paires are more similaire than the second which is false.
- For Bert model both values without stemming and without preprocess makes sense but results on larger sample of cards shown that bert model without preprocessing is more efficient.

After running tests and comparing the results of each experiments we got the best results by using bert model without preprocessing to calculate the description similarity and the cosine similarity to calculate the collection similarity.

The curator similarity is calculated by simply returning 1 if the two cards were curated by the same curator and 0 otherwise.

## 4.4 Computer vision

Since our data is not labeled we will be using a pretrained model. We will be comparing images to measure how similar two card images are.

### 4.4.1 Models:

As usual, we started by trying different models to determine the one that gives us the best results.

The list of the models that were implemented in this process:

**DeepAI[3]**: a deployed Api that compares how visually similar two images are, used through a post request, the lower the output is the more similar the images are. If the output is 0 the images are identical

**VGG19[5]**: a 19 layers convolutional network with a pre-trained version trained on more than one million images from the ImageNet database that returns the structural similarity between two images.

**Turicreate[1]**: The similarity between images is calculated using the resNet-50 model provided by turicreate which is a high-level framework developed by Apple. 4.3 ResNet-50 is a powerful convolutional neural network that is 50 layers deep that is able to detect shapes and forms. The model architecture is explained in the following

- **Convolutional layer:** It is always the first layer in a convolutional neural network, it emphasizes chosen characteristics in the source image by applying a predefined matrix on the input and passes the result to the next layer.
- **Pooling layer:** It's a way of sub-sampling to reduce the size of the image while preserving its relevant characteristics. Average pooling function outputs the average of the input patch values.
- **Batch normalization layer:** As each layer observes inputs produced by the set of layers that precede it, it would be advantageous to obtain centered and reduced inputs for each

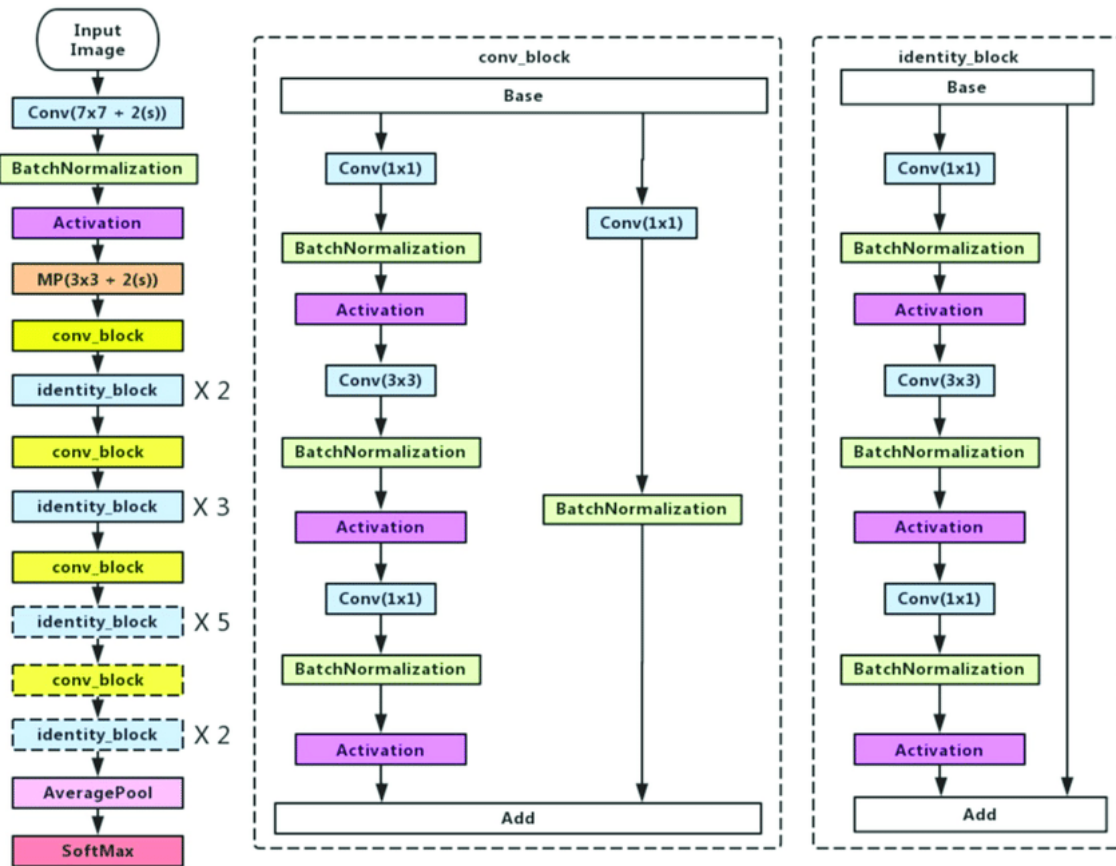


Figure 4.3: ResNet-50 model architecture [7]

layer. The normalization layer is a component introduced between the layers of the neural network and continuously takes the output of a particular layer and normalizes it before sending it to the next layer as input.

- **Activation Layer:** It determines how good the network model will learn from the data, it is a key part of the neural network design sum the weighted input to pass it to the next layer.

#### 4.4.2 Results:

Models results comparison is illustrated in the table below with three different pairs of images as an example

We can clearly see that the second two cards are very similar and the last two ones are less similar but the DeepAi api outputs the same value for both of them.



Figure 4.4: Sample of cards to test models on

		DeepAI	VGG19	Turicreate
card1	card2	24	0.14	14.94
card3	card4	32	0.17	19.17
card5	card6	32	0.15	22.7

Table 4.3: Comparison between different image similarity models

VGG19 returns values that indicates that the first pair is less similaire that the third pair which is false.

With more results on larger sample of cards we can say that turicreate output values are the most efficient.

Based on the results we got when we ran the model on all possible pair of cards, turicreate ResNet50 model seemed to be the best option.

### 4.4.3 Model implementation:

**Step 1:** We load the images

**Step2:** We create the model

**Step3:** We create a similarity graph from which we can retrieve the k nearest neighbors to an input image.

### 4.4.4 Computer vision output:

The turicreate ResNet50 model get as an input an image and returns the rest of the images in the database ordered by similarity from most to the least similar image. 4.5

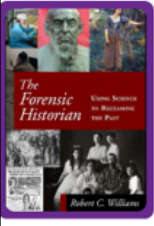





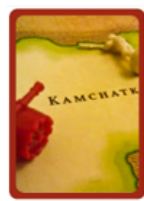
card	similar cards in order from most similar to least					
						
						

Figure 4.5: Turicreate output

### 4.4.5 Weighting the data:

We had to give different weights to the outputs of the AI. These weights went through different iterations in order to feel right for players. Indeed, after two play testing sessions we noticed that the images that were from the same collection were more likely to be accepted. This did not conform to the Client’s vision of the AI encouraging clever and unexpected connections. We tried different weights for each link type and the best values for the weights to keep the game playable, interesting and fun. In the end, we decided on the following weights: description similarity: 47/3 image similarity: 80 collection similarity: 5 curator similarity: 5 collection and curator similarity has the lowest weights since its an easy connection to guess and it doesn’t require too much effort.

Final output = collection similarity \* 5 + curator similarity \* 5 + description similarity \* 47/3 + image similarity \* 80



## 4.5 Flask backend:

We have developed a simple Flask API so that the game would be able to communicate with the AI through restful requests. The API takes a post request with the ids of the two cards and returns the similarities between 2 cards

## 4.6 Non-funtional requirements

### 4.6.1 Portability:

the Ai is deployed through on a docker container service which makes it easy to use on different machines.

### 4.6.2 Documentation:

Every part of the AI is well documented to make future changes easy.

### 4.6.3 Modifiability:

It's easy to add new AI functions to the game since the Ai is an independant api, where the comparator calls the functions that are put in the AIfunctions.py file

### 4.6.4 Performance:

The AI was used to pre-process all of the possible card pairs. That way, the response time was lowered from 4 seconds (live computation) to 1 second (information retrieval from the database).

### 4.6.5 Scalability:

Deploying the AI on a container makes it easy to duplicate the container in case of high demand.

## 4.7 Conclusion:

In this chapter we talked about different models we implemented. We explained each one of them and compared their results. We based our choice of the final model on these experiments. Since our data isn't labeled we had to manually evaluate the chosen model's performance. The outputs we get from each model are summed up with specific weights to make the final result which is the similarity between the two cards.

# Chapter 5

## Evaluation and Deployment

### 5.1 Introduction

With the game development, backend and AI explained, we can now speak about the final solution as a whole. These three pieces are integral to its functioning, and none of them can exist without the others.

The game relies on both the backend and AI, querying them for card similarities and logging. In this concluding chapter, we will talk about the evaluation and deployment of our solution.

### 5.2 Evaluation

#### 5.2.1 Client feedback and Playtesting

To guarantee the clients' satisfaction we organized two playtesting sessions with each major game version, where we had the chance to witness in real time the players' interaction with the game. We took note of any difficulties they may have come across and had detailed discussions about the different parts of the game. Each playtesting session was followed by them filling an anonymized google form so that we could get more specific feedback about their experience with that version of the game. The forms had the following questions:

- Did you encounter bugs while playing? If so please give a brief description of the bug and what you were doing before it came up.
- Was the game slow ?
- Did you find the AI judge to be too strict ?
- Did you find the AI judge to be too lenient ?
- Do you have any suggestions for naming the AI judge?
- What do you think of the scoring system ?
- Do you like the design of the game ?
- If you didn't like the game design, do you have any suggestions for us to make it better?
- Did you like the background music ?
- Do you have any suggestions for future improvements ?
- Please give us a rating.

The most important concerns that they raised were the following:

- The scoring system seemed a bit unclear at first, with the final similarity not being the exact number that was added to the score. This concern can be seen in the mixed responses we had in the first playtesting poll [5.1](#):  
In response to this, we have simplified the scoring system so it would make more sense to the players.
- The scoring system favouring cards that were from the same collection: Which brought us to rethinking the weighting of the final similarity.
- The AI judge only providing a similarity score felt very arbitrary. [5.2](#) We added the functionality that let the player display more information on the score.
- The similarity result screen necessitating a player action to close instead of closing after a certain amount of time, which we changed later.

What do you think of the scoring system ?

6 réponses

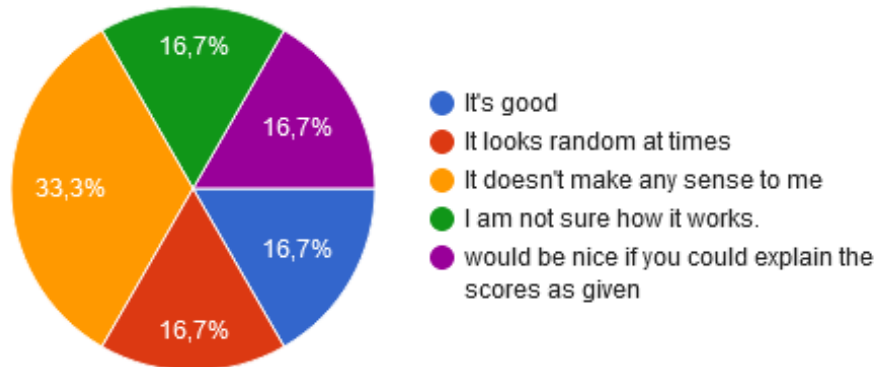


Figure 5.1: Feelings on the scoring system

If you didn't like the game design, do you have any suggestions for us to make it better?

2 réponses

I liked it

explain the AI's decisions where the player fails

Figure 5.2: Feedback on the scoring system

- The game feeling like it revolved around trying to guess what the AI would like to see. We remedied this by changing the scoring system slightly to give personality to the AI, as well as giving a choice to the player on the way they would like to play. This went through well with the client, as portrayed by this review: [5.3](#)

What do you think of the AI judge?

Une réponse

I didn't think about it too much. I know that it works in a "certain" way, like all computers do. The main thing I think about is how is the AI used or how does it fit into the game. I think with the changes to give the AI some personality and add some humor, the AI fits in in a way that makes the game really fun.

Figure 5.3: Feedback on the AI as a character

### 5.2.2 Performance

Running the AI locally on our computers takes 2 seconds to give the final response but when running it on the server it takes about 4 seconds. In order to enhance the server speed we needed to increase our memory allocation in the container service and to store pre-computed similarities between each pair of cards into the database for further use. So whenever we need to calculate the similarity between two cards we check if the wanted result already exists in the database. If that's the case the call to the database will take around 1 second to return the result. If not we call the AI to compute it which takes around 4 seconds.

These measures ensured an overall good experience for the clients, with the following answers given: [5.4](#)

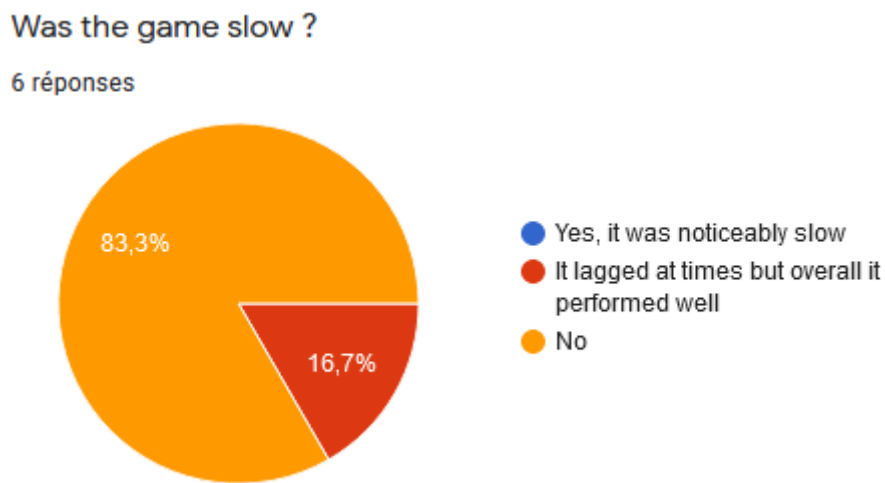


Figure 5.4: Feedback on the performance aspect

In the future, adding reinforcement learning to the game will play a big role in enhancing its performance and making the AI more intelligent. If a card is rejected the players input a reasoning as to why they thought the two cards were similar and why they saw a connection that the AI wasn't able to see. The game will take more into consideration this human perspective and evolve with time.

## 5.3 Deployment

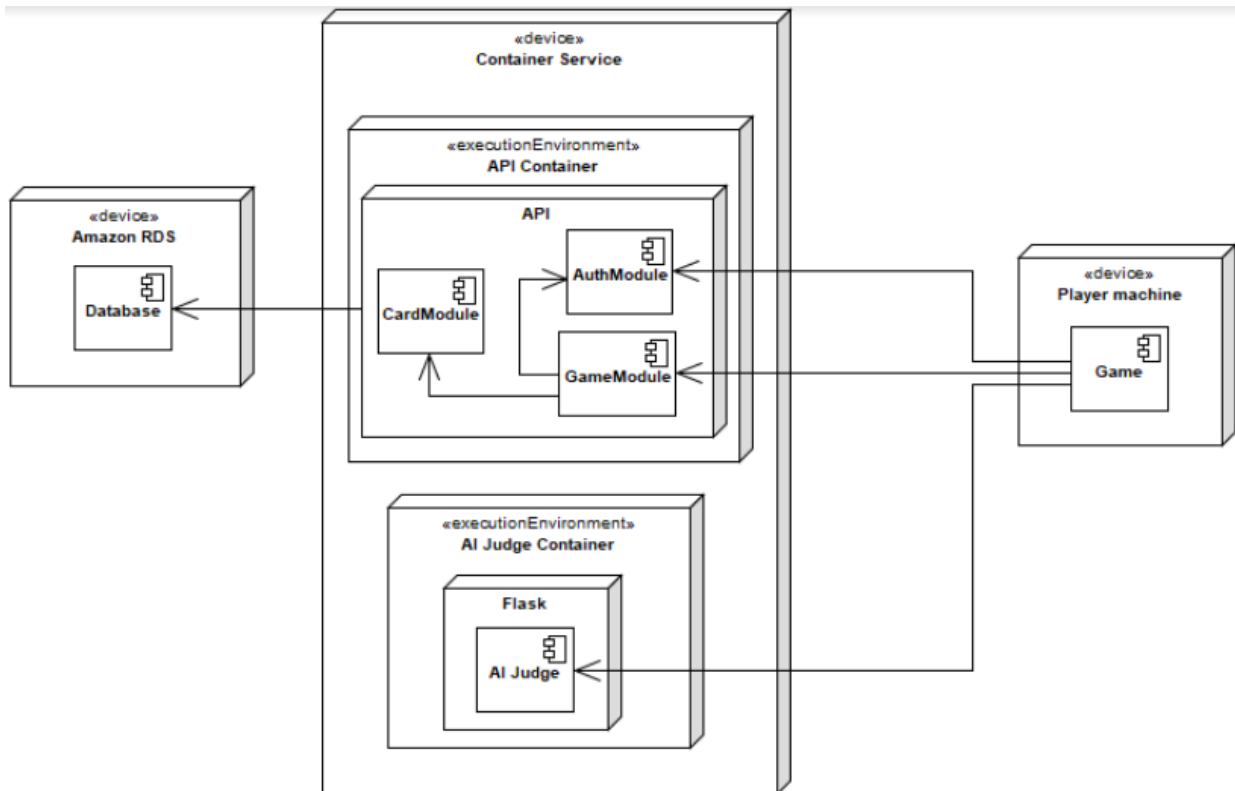


Figure 5.5: Deployment diagram

The player only has to download the game executable on their computer or phone, without the need for installation. The game is immediately playable on the player’s machine. The game then sends requests to the API for logging the gameplay and authentication. The API resides in a container in the container service. The game also sends requests to the AI Judge when the player asks for details on a judgement, or as a backup if the API is unavailable. The API can also be found on this same container service. The API then queries the database. The database resides in the Amazon RDS (Amazon Relational Database Service) which can only be accessed from within the Container Service. The Container Service resides in the cloud.

## 5.4 Conclusion

Ultimately, this project, completed with The University of Michigan, aimed to lead the students to the real-life collections that this game portrays. Its goal is to encourage immersion in art, as well as finding meaning and similarities in unusual places.

In this report, we have documented the work process used to implement our solution. We first presented the project's context, concept and requirements. The functional scope of the project was later built on this foundation. We, then, meticulously designed our solution to meet all of the agreed upon requirements (both functional and nonfunctional).

We found that the best way to tackle this project was to divide it into 3 separate sub-projects corresponding to our 3 main axes : game development (frontend), database and API (backend) and the AI judge. Finally we went into details explaining these 3 axes, their implementations and the links between them.

One of the biggest challenges that we encountered while working on Collect/Connect was assessing different AI algorithms and trying to find the ones giving the best results. The issue was that we did not have set metrics for judging the results so we had to test algorithms with different cards then visually assess them and compare results. Another challenge that we came across while working on the AI part, is choosing the right weights for the different comparisons without making the AI judge too strict or too forgiving. Finally, we struggled a bit with trying to save the huge amount of required data in the database all the while optimizing as much as possible.

As for future improvements, we implemented the backend while keeping in mind a list of features to be added by the next team working on Collect/Connect. The first feature is hints: if a player struggles with finding the right action to take, they could ask the game for a hint which would then get the comparison results for all cards in the player's hand and choose the best one to play. If the game found all links to be unacceptable, it would then suggest that the player discards the worst card and draws a new one. Another feature to add, is browsing the deck of played cards during the game. Lastly, an important feature to add is reinforcement learning which would make the AI judge take into consideration the frequency of links and reasonings given by the players before accepting or rejecting a link.

Thank you for your attention.



# Bibliography

- [1] Apple. *Turi Create API Documentation*. <https://apple.github.io/turicreate/docs/api/>.
- [2] Ana Azevedo and M.F. Santos. *KDD, SEMMA AND CRISP-DM: A PARALLEL OVERVIEW*. <https://recipp.ipp.pt/bitstream/10400.22/136/3/KDD-CRISP-SEMMA.pdf>.
- [3] DeepAI. *Image Similarity API*. <https://deepai.org/machine-learning-model/image-similarity>.
- [4] GeeksforGeeks. *Removing stop words with NLTK in Python*. <https://www.geeksforgeeks.org/removing-stop-words-nltk-python/>.
- [5] Aakash Kaushik. *Understanding the VGG19 Architecture*. <https://iq.opengenus.org/vgg19-architecture/#:~:text=VGG19%20is%20a%20variant%20of,VGG19%20has%2019.6%20billion%20FLOPs..>
- [6] Lynne Raughley. *In this game, everyone wins*. <http://hdl.handle.net/2027/spo.14770791.2020.008>. [Online; accessed 16-June-2021]. 2020.
- [7] *ResNet50 architecture*. [https://www.researchgate.net/figure/Left-ResNet50-architecture-Blocks-with-dotted-line-represents-modules-that-might-be\\_fig3\\_331364877](https://www.researchgate.net/figure/Left-ResNet50-architecture-Blocks-with-dotted-line-represents-modules-that-might-be_fig3_331364877).
- [8] Jan Wijffels. *Textrank for summarizing text*. <https://cran.r-project.org/web/packages/textrank/vignettes/textrank.html>.
- [9] Wikipedia. *BERT (language model)*. [https://en.wikipedia.org/wiki/BERT\\_\(language\\_model\)](https://en.wikipedia.org/wiki/BERT_(language_model)).

- [10] Wikipedia. *Cross-industry standard process for data mining*. [https://en.wikipedia.org/wiki/Cross-industry\\_standard\\_process\\_for\\_data\\_mining](https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_mining). [Online; accessed 22-May-2021].
- [11] Wikipedia. *Stemming*. <https://en.wikipedia.org/wiki/Stemming>.