TarUNO

Tarot Card Reading using UNO cards and Augmented Reality

Javier AGOSTINI
Brandon FOBUGWE
Mario AROCA PÁEZ
Kai ZHU

27/January/2023
IG.3504 - 3D, Mixed, and Augmented Reality
ISEP

TABLE OF CONTENTS

INTRODUCTION

This application is the final project from group 8 for the IG.3504 - 3D, Mixed, and Augmented Reality module in Institut Supérieur d'Electronique de Paris. It consists of doing a tarot reading with your phone using UNO cards. It is as simple as scanning 6 random UNO cards and the app will tell you to which Tarot card it is equivalent and will show the meaning of this card so you can see what the future awaits for you.

TarUNO is designed using Unity with its respective technology for AR and exclusive for IOS devices.

TEAM

The team that has developed this application is composed by:

- Javier Agostini - Software Engineering - Mexico - (62516)
- Kai Zhu - Medical Engineering - Germany - (62546)
- Brandon Fobugwe - Electrical Engineering - Germany - (62526)
- Mario Aroca Páez - Software Engineering - Spain - (62518)

CHAPTER 1: GAME DESIGN AND SETUP

Goals and Objectives

After the first session the team did not have any clear objectives. However they were interested in using AR technology for the project. After watching some examples of AR applications done with Unity it was decided that they wanted to do something related to cards, the first idea was to create a translator from spanish deck cards to poker deck cards and vice versa but the idea was discarded as the team thought it was not that interesting.

The idea of using cards was still there so they focused on UNO cards as they are the most popular among young people and it seems likely that everyone has a deck of UNO cards around them. Thus after some different ideas it was decided to do a game in

which the user can read his fortune using UNO cards like if they were Tarot cards. In addition to all the different technologies, the team had to learn about Tarot cards, how to use them, the different types that are available, the differences and especially how to do a simple Tarot read in order to implement it for the application.

App Concept

The idea of the app is the following. The user will start the app and start to read the different cards. When the reading begins the app will display the camera of the mobile device, the user will have to point his camera to a UNO card until the app recognises it. As soon as this happens, the card will show the equivalent Tarot card in AR and a button will appear giving the user the option to read the meaning.

For understanding how the reading works it is fundamental to understand that there are two types of Tarot cards, the Major Arcanas and the Minor Arcanas. Major Arcanas are special ones like The Magician, The Fool, The Emperor and The World. Minor Arcanas consist of Wads, Cups, Swords and Pentacles. Each one of these 4 types has twelve cards (two of wands, five of cups, seven of pentacles…) However, only 10 of each type were used in order to level the amount of Tarot cards to the number of UNO cards, so the app Tarot deck for Minor Arcanas only consists of numbers between two and ten and one ace.

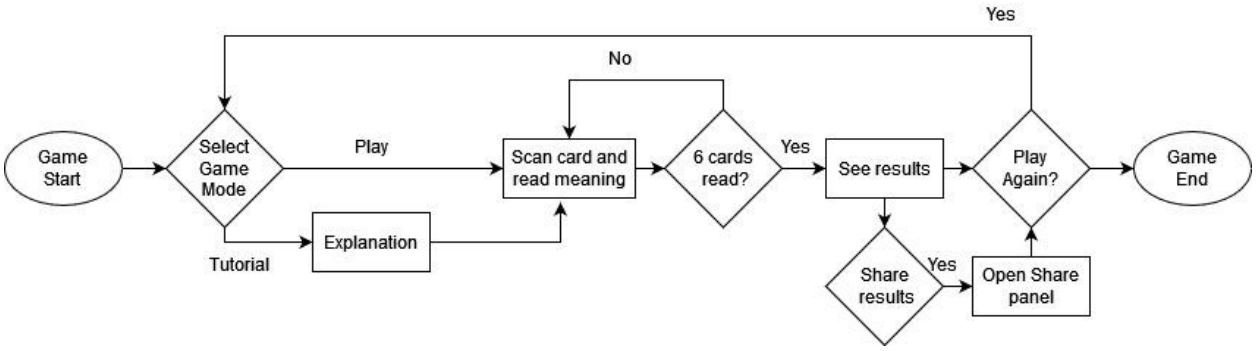
Each Tarot card has two different meanings: a good one that is the one the user gets in real Tarot when he displays the card upright and a bad one that is the result of displaying the card in reverse. To simulate this by software the two meanings were added to each card and the app will display the good or the bad meaning randomly.

The equivalences of Tarot card to UNO card is done in a way where special UNO cards are equivalent to Major Arcana cards. For example Red Skip is equivalent to The Fool, Green +2 to The hanged man, +4 to The Sun and so on. Subsequently normal UNO cards are equivalent to Minor Arcana cards. Red cards to Wands, blue to Cups, greens to Swords and yellow to Pentacles. Normal UNO cards go from 0 to 9 so the equivalence for the four different types is: 10 Green - Ace of Swords, 1 Green - Ace of Swords, 2 Green - 2 of Swords, 3 Green - 3 of Swords and so on.

So the Tarot reading for TarUNO is performed in the following way: The user will take six random UNO cards and will scan them one by one. After scanning each card the

user will see an animation with the equivalent Tarot card and will be able to read the meaning of that card (Can be good or bad). He will do this six times for six different cards (The app will detect if the user is trying to scan the same UNO card again) and after that he will have a summary of his reading with the different cards he got and their meanings having the option to share it with his friends.

Game Loop Diagram



System Requirements

In order to use TarUNO the user needs a mobile device running IOS 11 or higher

Logo

The original idea for the logo was mixing the words Tarot and UNO thus TarUNO, was the result. For designing it Canva was used, and the word “TAR” was added to the original UNO logo, so it has its own identity but still feels loyal to the UNO brand as it was an original product from the brand.

CHAPTER 2: FIRST DEMO

For the third week the team had already built a basic demo to test how AR works for unity and mobile development. The first version did not have any type of UI or game loop, when the user entered the app it was only the camera. Some really different from one another cards were used for this version so when the application detected them it would display some basic figures in AR like a square or a circle.

This demo worked mainly in order to see if the team had understood the concept

of AR and how it worked, from that point on it started developing. For the fourth week new AR figures were introduced in testing like a rubik's cube with a floating animation (from the asset store) and it was also introduced that these AR figures would follow the cards. So if the user would move the card and the app would still detect it, the AR figure would follow the card with its movement.

Subsequently in the following weeks the concept of reading the meaning of the cards would be introduced. Whenever the user saw one of these AR figures (which were still no Tarot cards) a button would pop up with the option to read the meaning and when clicked the meaning of that card would be displayed.

So after this addition a basic UI was added to the project and this was concluded as the first working demo of the application.

CHAPTER 3: PREFAB AND ANIMATIONS

Prefab Design

The AR aspect of our App manifests in the instantiation of virtual Tarot cards above the detected UNO cards. The AR instance is supposed to symbolize the Tarot reading. The Tarot card is displayed as a cuboid with negligible thickness. That way the prefab appears as a plane with the same aspect ratio as the UNO card it refers to. The prefab displays an image of a specific Tarot Arcana illustration on both sides. We included the Tarot card illustrations from “The Pictorial Key to the Tarot” which was written in 1911 byArthur Edward Wait. The illustrations were created by Pamela Colman Smith.

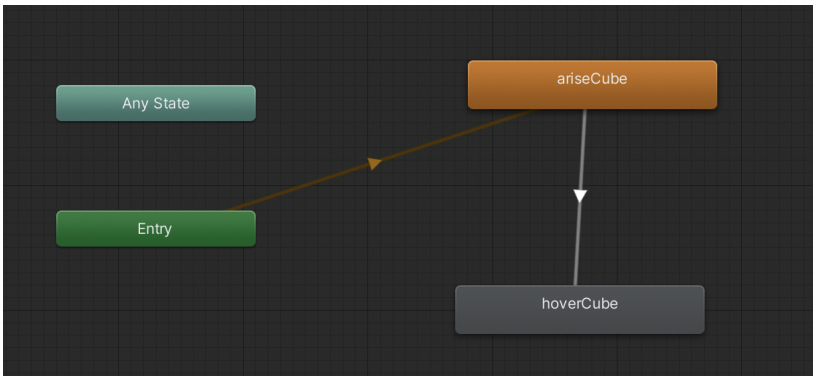
Her versions of the numerously illustrated Tarot Arcanas have been one of the most popular depictions since their publication, which is why we decided on implementing them. After US copyright law creations become part of the public domain 60 years after the death of the creator. Both the author and the illustrator died more than 60 years ago allowing us to use these pictures without running into any legal claims whatsoever.

In upcoming updates we are considering improving the appearance of the prefab. For the prefab it remains to discuss the perfect size. Currently the prefab spawns at a size

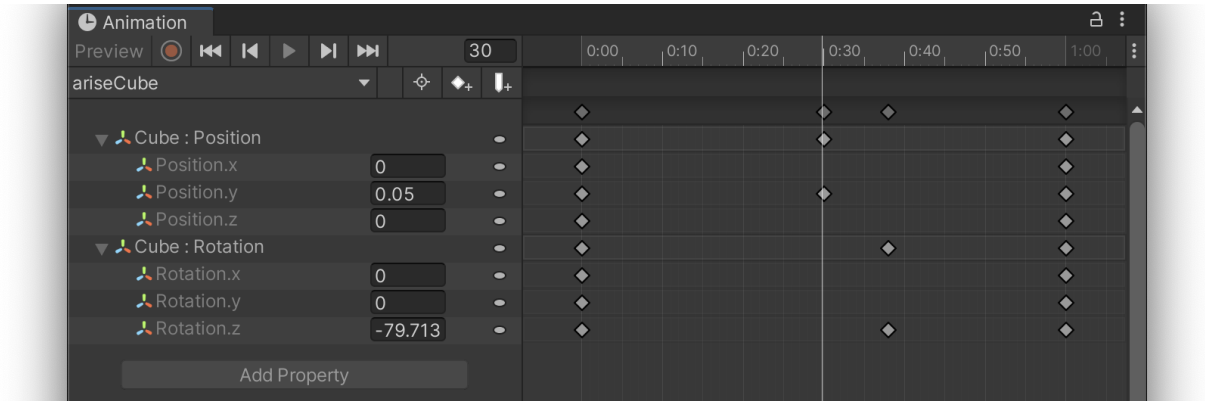
which makes it easily identifiable. However it covers the Uno card when viewed from the front. For some users this might appear too big. A more subtle implementation of the visual representation could prove to be more pleasant to look at.

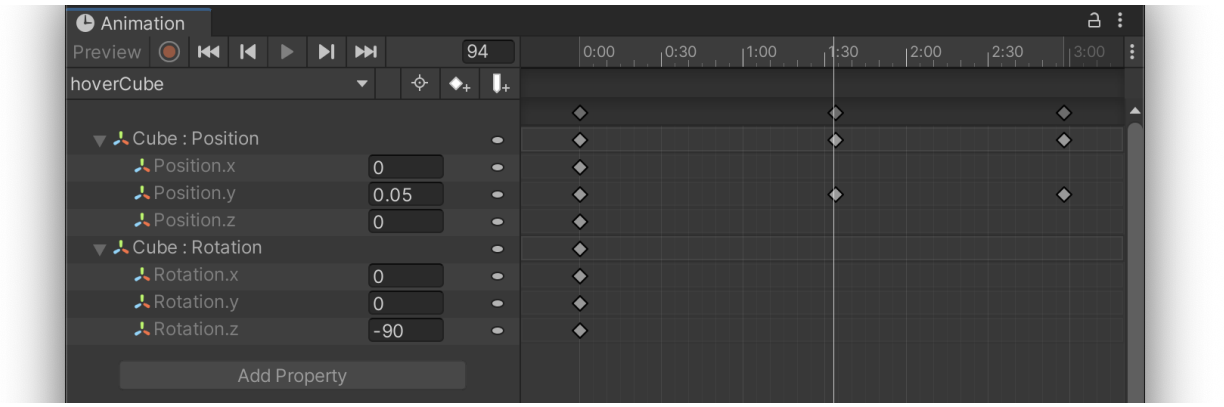
Animations

Animating the card is an important part of the user Experience. The card immediately starts animating after being instantiated. The animations are managed by an Animator controller which initiates an arise animation ("ariseCube") that lifts the card up from a horizontal position into a vertical one. Here the parameters were chosen in such a way that the following hovering animation is transitioned into seamlessly. So as soon as the 'arise' animation has concluded the hovering animation ("hoverCube") is entered. This is then looped for as long as the prefab exists.



After instantiation the card arises once and then keeps hovering





A particle system is used in order to enhance the visual appeal of the card. It spawns floating orbs that emit from the center of the card in a circular manner. They individually illuminate the card while moving outside. For that we use a yellow point light source which is attached to every particle that is spawned. The size of the particles is varied over time creating an impression of seamless birth and death of the particles. They also demonstrate a slight buoyancy which is implemented by using a negative gravity coefficient resulting in an upwards floating illusion.

Problems

Implementing the animated prefabs did not come without difficulties. Notably it is important to animate an object nested inside the actual prefab so that the Animator uses local space when moving the cuboid. Besides that Unity did not provide us with a streamlined way of ascertaining the correct size the prefab is supposed to be spawned within the AR environment. That problem is going to be touched on further in chapter 5.

Next steps

The appearance of the animation, especially the seamless instantiation of the Tarot card from the UNO card, has room for improvement. Furthermore the particle system bears a lot of potential. Lastly, sound effects are greatly beneficial in enhancing the user experience. They could assist the animation in creating a satisfying and beautiful illusion of the virtual card arising from the real one.

CHAPTER 4: COLOR DETECTION IN AR

Color Recognition in AR Foundation

AR Foundation allows the detection of images or objects from the camera view to then spawn a prefab or perform an action in the virtual world. When we first started working on the project, we were using random objects such as ID cards, and then eventually two random UNO cards to test out the different capabilities that AR Foundation had and how they could be used for the project.

As we progressed and added all the UNO card deck, it became apparent that the application wasn't able to tell the images apart. Instead, it would try spawning all the different prefabs all at once, making the application significantly slower and confusing the user.

The professor noted that AR Foundation image detection doesn't include color. Instead, it converts the camera feed to grayscale before looking for the feature points. In our particular case, since there were four cards with the same shape, number, and form, but just a different color, it was impossible for the image detector to know which card the user was truly showing.

Solution 1: Color Detection Asset

The first solution consisted of getting a color detection asset from the Unity Asset Store and avoiding any additional coding complexity. This asset was able to detect and track colored objects in the camera view, so it would allow us to identify a card's color to then show the user their fortune.

The main problem with this option was that the asset was not free. The cost for said asset is 24.95 EUR, so as a team, it was decided that it would be best to first try achieving the color detection through code, rather than an imported asset.

Solution 2: Take Screenshot and Read Pixels

Our second solution consisted in taking a screenshot when an image is detected on the camera view and then reading the pixels of the screenshot to analyze what the main color of the image is and assign the card a color.

This was a challenging task, since it involved having some knowledge of how images and the camera work on Unity and how to read and write pixels. Also, since the card can be recognized from various positions and angles, it doesn't require to cover the

full screen, meaning the full screenshot would have additional pixels that would be part of the background and would interfere with the color detection of the card.

For this reason, it was decided to crop the original screenshot to just the section where the card is and then read said pixels. In order to do so, we followed these steps:

1. Take screenshot of size Screen.width by Screen.height and save it to a Texture2D game object
2. Create Rect variable of the same dimensions as the screenshot
3. Call method ReadPixels for screenshot texture
4. Call method Apply
5. Crop the screenshot by calling method GetPixels with the tracked image position as parameters, as well as values 200 and 300 for width and height
6. Instantiate a new Texture2D and call method SetPixels with the cropped pixels

After the screenshot texture is obtained, a loop checks every single pixel on the screenshot and saves the average value in RGB. Then, the color calculations are:

- If the difference between the red and green average is less than 50 points, and the red and green values are larger than the blue one, then the assigned color is yellow
- Else if red is the largest average value, then the assigned color is red
- And so on for the green and blue values

Spawning of Prefab on Screenshot

After implementing this solution, an image game object was added to the canvas for testing purposes. Here, the pixels of the cropped screenshot were set to see in real time and make sure the program was working properly.

After a few tries, we noticed that the screenshot would sometimes include the spawned prefab corresponding to the card. This would happen because the screenshot would be taken right at the moment the card got recognized and the prefab was shown, so the code was modified to make sure there was no prefab active at the time of the screenshot.

Since the screenshot function is a coroutine, this approach did not work, as it was difficult to track when the coroutine started and ended. This meant that even though all

the prefabs were deactivated, the prefab would sometimes show up on the screenshot.

After trying different ways to surpass this issue, we came to the conclusion that this was a very unreliable solution and required more complexity that ended up slowing down the game and creating some lag when playing.

Solution 3: Accessing the Camera Image on the CPU

Finally, after talking to the professor about the issue, he shared some resources and documentation on accessing the camera image on the CPU. This approach solved the prefab screenshot issue since the camera image contains the original camera feed, without any of the prefabs or other AR resources.

After looking at the documentation, we learned that CameraImage, a struct which represents a native resource, provides both synchronous and asynchronous conversion methods.

For the game, it was necessary to convert to color synchronously, as the color of the card needs to be detected before continuing the game. This method converts the CameraImage into the TextureFormat specified by the conversion parameters and writes the data to the destination buffer. Some of the conversion parameters are:

- inputRect: The portion of the CameraImage to convert
- outputDimensions: Dimensions of the output image
- outputFormat: It could either be RGB24, RGBA24, ARGB32, BGRA32, Alpha8, R8
- transformation: For example, mirroring the image across the X or Y (or both) axis

This approach was the best one, since it was possible to retrieve the camera pixels without having to worry about the AR resources spawning on the virtual space. After testing it out, the game did slow down a bit, especially when trying to share screen or playing music, but it was a better, more reliable way to detect a card's color.

CHAPTER 5: PREFAB SCALING AND POSITIONING

When placing 3D objects in a scene, the scales between the unity plane and the real world are not the same. For instance, 1 unity scale corresponds to 1 meter in the real world. For this reason, when the first prefabs were created, the dimensions were too big

for the UNO card dimensions. This would cause the prefab to barely be seen and make the experience less genuine and immersive.

Resizing Prefabs

When dealing with prefab dimensions, there's 2 ways to adjust the scale: either scaling down the scene objects or scaling up the AR Session Origin Object. The first instinct was to scale down the original prefabs so that they would be around the same size as the real world UNO card. After testing and adjusting the values, the prefabs looked more realistic and blended better with the scene.

Regardless of the decision to scale down the prefabs, it is important to notice that scaling up the AR Session Origin Object is often a better solution, especially when working with physics collisions, since they work more precisely with bigger 3D model scales. Since our project does not deal with collisions, this decision does not affect the overall performance of the application, but it would be something important to consider for a future development of the project.

Animating in AR

Since the point of the application is to create an immersive and entertaining experience reading tarot with UNO cards, the design of the prefabs and the animations were very important aspects to consider.

The original idea was to have a tarot card floating on top of the UNO card (perpendicularly) with some fog or other effects. There is an animation called 'arise', which rotates the card 90 degrees so that it is perpendicular to the UNO card, while at the same time lifting the card up. Then the animation called 'hover' keeps the card going slightly up and down, to give an effect that it is floating.

Although the first attempt worked on its own, when integrating it to the main project, we noticed that the prefab was not spawning on top of the UNO card, but rather at the origin. After debugging and going through tutorial videos and other documentation, we realized that this issue occurred because the animations that were set on the prefab were set on the parent game object, rather than a child game object.

Since the animations are all based on the origin (0, 0, 0) as the starting values, if the animation is set to the parent game object, the prefab will be placed at the origin and

the animation will start there. On the other hand, if the animation is set to the child game object, then the animation will be relative to the parent game object, which is placed where the UNO card is detected.

CHAPTER 6: VERSION CONTROL ON UNITY

As it is commonly known, working collaboratively in Unity can be difficult. There are many different aspects that have to be considered, such as the version in which the project is built, the operating system of the computer, the size of the project, etc.

MacOS vs. Windows

Half of the teammates worked on MacBooks, while the other half worked on Windows PCs. This made the work on Unity a little tricky. Regardless, it wasn't very complicated to work, as there are only some files that are specific for MacOS Unity projects that can cause some issues on Windows.

GitHub and Unity

During the first weeks, it was difficult to push the project to GitHub. It would take a very long time and at a certain point, the push would get stuck and the project would not upload successfully. It was originally thought that the size of the project was the issue, but it was later discovered that the problem was network related. After switching to a different network, the project was pushed successfully.

Even though the project is fully posted on GitHub, after reviewing the work and the use of the tool for collaboration, it can be concluded that we could have used it more efficiently throughout the semester.

At the beginning, issues were posted with the different tasks that needed to be done, as well as the category they belonged to, and the person assigned to them. Ideally, this would be updated every week, but with time, it was harder to keep track of what was being done and as new bugs arised, the issues stopped being updated.

For future development, it would be important to better understand and correctly use GitHub, in terms of issue usage, branch naming, as well as pushing and merging changes. This would allow for more smooth collaboration and would let every team

member to better understand what is being done and track the overall progress.

CHAPTER 7: DESIGNING THE UI/UX

Starting screen

This part of the project began by designing the starting screen. The first mockup was a plain dark blue background with a big UNO logo and a headline that said “TAROT READER” in a font that was hard to read and did not match with the rest of the UI. Also three different buttons were implemented, Play, Options and Exit. All in all it was a very rudimentary design which lacked a colorpalette, shapes and aesthetic, so the first mockup was abandoned quickly. For the second draft a background picture was added instead of having a plain colored background.

The big UNO logo that was the eyecatcher in the first try was now being scaled down a lot and put on the bottom of the starting screen, so the big colorful logo will not be the center of attention anymore. In addition a button to start the main game and a button for the tutorial were implemented to the starting screen. The project was given the name “TarUNO” so now the headline of the starting screen reads “Welcome to TarUNO”. The second draft was already rather acceptable but it still needed some more adjustments. For the final prototype the background was changed one more time and the color of the buttons were adjusted to match the color scheme of the new background.

End screen

For the end screen the same background picture as the starting screen was used. In the middle of the screen we added a transparent white panel which would show a summary of the six cards that were drawn in the latest tarot session. A share button beneath the panel was also included which allows the player to share his results with his contacts via the classic IOS sharing UI. Similarly like in the starting screen there are also two buttons available, but this time it is the Start new game and back to main menu button.

Tutorial

While doing the tutorial, the camera is activated at all times. When the app is explaining how the game is supposed to be played a text will appear on the screen and

the recording from the camera in the background is displayed in a more desaturated way. By tapping the screen after reading the shown text, the next part of the tutorial appears with new instructions.

Playing the tutorial will also allow the player at one point to scan a random UNO card for demonstration purposes. If the player does not scan the UNO card and wants to continue with the tutorial by tapping on the screen, a notification pops up, warning the player that there is no card on view and advises him to point the camera on a UNO card to continue. After finishing with the turarial the player will reach the end screen and then he will be ready to play the main game.

Making the UI Responsive

Because of the fact that different iPhones have different screen sizes the app has to be responsive and scaled according to the device it is running on. Working on the responsiveness of the application, the main problem was that the whole screen got distorted when the phone was held horizontally. In the end this issue was solved, but not in Unity but instead through Xcode. The solution was to restrict any turning movements of the app, which means that even if the phone is held horizontally, the game is not adapting to that rotation and will still work and display everything as if the phone is held vertically.

Adding sound effects

Sound effects were also added to the game to improve the user experience even more and give the player a more realistic feeling of an actual tarot reading session. Every segment of the game received its individual sound effect that fit the best. For example the music added to the starting screen is sort of mystic compared to the sound effects that are playing in the end screen which are more victorious. In case the player prefers playing the game without sound effects, a mute button was implemented on the starting screen, allowing him to change the sound settings as he wishes. The next step would be adding sound effects every time a button is pressed to give the player audible feedback and increase the quality of the user experience.

CHAPTER 8: DEPLOYING ON MOBILE

iOS

In order to deploy for iOS, it was necessary to do some research on how unity programs worked on iOS. After building the project on said platform, the newly created folder was opened on XCode.

An iPhone was connected to the computer to download the app but we soon realized that we needed to fix some 'Signing & Capabilities' settings, as well as other configurations in Unity, such as the camera usage description. After fixing these issues, the app was able to be downloaded on an iPhone device for testing and demo-ing purposes.

Android

A whole session was dedicated to trying to deploy the application on an android device, having to change most of the configuration for the build and having no success. Hence it was decided to focus on IOS due to the fact of the ease of deploying the app in this platform for different devices with no problem and it was also a crucial fact that most of the members in the group owned IOS devices rather than Android ones.

CHAPTER 9: SHARING FEATURE

It was originally envisioned that being able to share the results of one's reading would be an interesting feature to have on the application. This would not only give exposure to the app, but it would also encourage users to keep using it.

In order to be able to share one's results, we created a new script; this script contains a coroutine that sets the subject, text, link, and callback of the sharing. This coroutine is triggered when the user clicks on the share button that appears below the summary of their reading.

This feature makes use of the native share panel of the user's device and allows the user to share their results on social media or through text messaging apps, such as WhatsApp or iMessage.

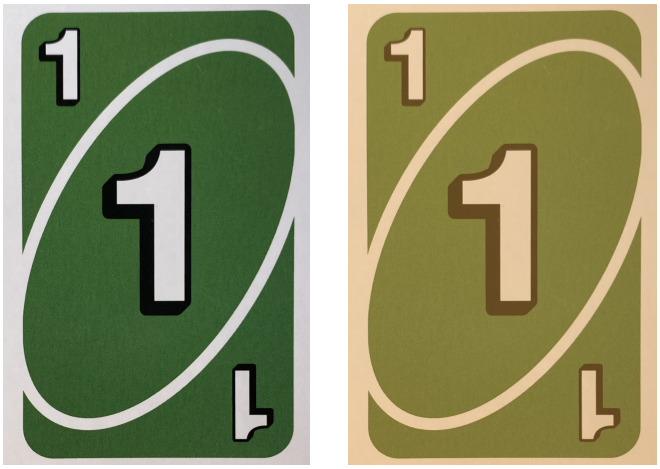
CHAPTER 10: NEXT STEPS

Environment Lighting

What is the problem with very warm white light?

The color detection currently is based on reading out the raw sensor data at a specific position. This data is accessible in RGB values. These sensor readings may vastly differ under different lighting conditions. While the stability of the color detection has yet to be tested it can be suspected that a very low temperature light (yellow white light) may significantly shift the Hue value of the card's color.

This is why the green card for instance might appear yellow in strongly yellow light. This is an issue we have to deal with considering the use case of our App. Take a typical situation the Game is going to be played in by our target group: A get together at night in a room that is dimly lit by colorful party LED lights. The App might struggle to decide which color the presented UNO card has if we stick to our rudimentary RGB color perception model.



Green UNO card under neutral white light (left)
and under warm white light (right)

The inner color of the right card actually corresponds to a desaturated yellow. Nevertheless, our brain includes the assumed lighting of the environment which is why we still correctly perceive the card to be a shade of green.

How does our brain deal with yellow light?

One has to note that for human perception up until a certain point illumination by warm white light does not significantly impact the perceived color of an illuminated object.This is due to a phenomenon referred to as chromatic adaptation. The brain recognizes the color of the light sources and adjusts the perception so that the color is perceived as under neutral white light.

To some extent camera software in a smartphone can apply an automatic white balance algorithm which imitates the human chromatic adaptation. Nevertheless, this only works for illumination that broadly resembles white light. Meaning that in a room that is illuminated by a yellow light source for example the white balance correction fails and misrepresents the real perception.

How do we deal with the problem?

One possible solution for this problem might be to use color appearance models. The goal of these is to imitate the human eye's/brain's ability to incorporate the color of the light source in the color perception process. A color appearance model takes in the color of the object as well as the color of the light source. One of them is CIELAB which represents color with a lightness "L" and color coordinates "ab". There exist third party libraries which enable the conversion of RGB to Lab.

Usually when digitally ascertaining the rendition of an object under specific light one has to measure the spectrum reflected by the object in question but also the reflection of a so-called white standard under the same illumination. These to measurements allow us to simulate the perception

Our proposition is to use the white frame of the UNO card as the white standard. Both the RGB values of the frame and the inner colored area would then be fed into an algorithm that transforms the card RGB to Lab values.

What other extreme lighting scenarios are problematic?

Low light scenarios bring issues as well. The phone's camera software increases the sensor's sensitivity when presented with a scene lacking brightness. This enables the camera to keep the same shutter speed thus the same framerate while keeping the video brightness up. On the other side this comes with a decrease in image Quality. The SNR decreases resulting in a visible noise. One component of this noise is color noise. This

might implicate difficulties when trying to determine the color as the saturation of the hue of the color is likely to be lower under big signal noise.

Deploying on Android

The workflow regarding development, testing and improving has been optimized for iOS which is why the App has for now only been implemented for iOS. Nevertheless the plan is to extend the possible user base by implementing an Android version of the App. In order for that to work we need to consider the numerous different Android builds. All the aforementioned other problems have to be addressed as well.

Calendar and Reading History

Tarot readings can change over time. This is obvious considering the randomness of the process. In popular belief this reading history is an important part of the reading that the participant receives. At the moment it is only possible to have a copy of the tarot session if the results are shared with a second person and there is no option on saving these dates on the application itself. Adding a calendar and a reading history would allow the user to access his past readings and he would even be able to check if the fortune telling was correct or not.

Dynamic Fortune

The last feature that could be added to the application is dynamic fortune telling. Maybe through the help of an AI the meaning of each card would change every day or week and even have different meanings depending if it is day time or night time. These additions would give the user a reason to use the app on a regular basis and to switch things up a little bit so it will not get boring after a period of time.

CONCLUSION

After many weeks of hard work, we are very proud of the final result. We believe that the application is unique and, if perfected, has the potential to be launched. There were many challenges that we had to overcome, such as the fact that the majority of the team members did not have any previous experience with Unity or AR projects, as well as time constraints and other responsibilities.

This project allowed us to understand the importance of a good first stage of design and planification before starting to code and deploy. By giving enough time to think and develop the idea of the game, the structure, the sequence, and the rest of the setup, as well as defining what our vision was, we were able to better understand what we needed to do and divide the tasks to complete the project in a timely manner.

In terms of modifications or improvements, we would say that having a better use of GitHub would definitely be important, as it is here where we register all the changes and are able to retrieve the most updated version of the project, in case something wrong happens.

For a future version of the application, we would like to improve the color recognition algorithm, so that the card's can be accurately detected in different illumination settings, such as low light, warm light, etc. Also, having the app available on both Android and iOS is important, as we could increase the amount of users enjoying the game. Finally, features such as changing the meaning of the cards every certain time or according to specific events, such as sunrise or sunset, as well as having a history or calendar with previous readings would further increase the usability of the application and would encourage users to download the app and use it.

REFERENCES

Biddy Tarot. (n.d.). *Learn the Tarot Card Meanings*. Biddy Tarot. Retrieved January 27, 2023, from https://www.biddytarot.com/tarot-card-meanings/

Brackeys. (2017, November 29). *START MENU in Unity*. YouTube. Retrieved January 27, 2023, from https://youtu.be/zc8ac_qUXQY

Playful Technology. (n.d.). *Augmented Reality (AR) tutorial for beginners using Unity 2022*. Wikipedia. Retrieved January 27, 2023, from https://www.youtube.com/watch?v=gpaq5bAjya8&t=316s

Stenudd, S. (n.d.). *Tarot Card Copyright — What is copyrighted and what is public domain*. Tarot Card Meanings. Retrieved January 27, 2023, from https://www.tarotcardmeanings.net/tarot-copyright.htm

Unity. (n.d.). *Accessing the Camera Image on the CPU | AR Foundation | 3.0.1*. Unity - Manual. Retrieved January 27, 2023, from https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@3.0/manual/cpu-camera-image.html

Unity. (2013, January 4). *AR tracked image manager | AR Foundation | 4.1.13*. Unity - Manual. Retrieved January 27, 2023, from https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.1/manual/tracked-image-manager.html

Waite, A. E., & Smith, P. C. (n.d.). *The Pictorial Key to the Tarot Index*. Sacred Texts. Retrieved January 27, 2023, from https://www.sacred-texts.com/tarot/pkt/index.htm