Bilkent University

Department of Computer Engineering

Senior Design Project

Final Report

DeepGame

Students

Mert Alp Taytak

Betül Reyhan Uyanık

Ömer Faruk Geredeli

Supervisor

Dr. Uğur Güdükbay

Jury Members

Prof. Dr. Özgür Ulusoy

Asst. Prof. Dr. Shervin Rahimzadeh Arashloo

Innovation Expert

Cem Çimenbiçer

December 27, 2020

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Table of Contents

1 Introduction

Video games are a form of entertainment enjoyed by many people on a multitude of platforms. In various video game genres the player plays a character, where the character becomes an extension of themselves. Naturally, this extension may take form as the approximate replica of the player or a person of player's choosing. In order to accommodate this, game developers offer character customization options that have been getting more and more intricate. However, evolving technology allows us to take this beyond what sliders, preset options and limited degrees of freedom can achieve.

The particular technology that enables this is the recent development of algorithms commonly called "deepfake algorithms". Briefly, deepfake algorithms transfer one person's likeness to another person in image or video media.

Deepfake algorithms are a very recent discovery and as a result, their application to different fields has been largely unexplored. One such field is video games. Our intention in this project was to apply this technology to video games as a means of character customization.

However, character customization as a means of transferring one's likeness to a video game is not a recent innovation. There are previous works that either use a simple approach of texture mapping a photograph to a game model or using expensive and relatively rare equipment in 3D depth cameras to construct a detailed model of the person. What DeepGame brings to this concept is the use of deepfake algorithms combined with other helper machine learning methods to take a few photographs of a person and their likeness transfer to a game.

In DeepGame, we use a neural style transfer method for changing the art style of a photograph from photorealistic to one more appropriate for video game art. Then, we put the product through a deepfake image animation method to generate animations of the styled input in various poses matching the game's needs. Finally, we take those animations and splice them into the game where appropriate.

Our original goal was to develop the technology and provide a software integratable with any game. However, throughout the development we realized various challenges and limitations with this goal. A prime example being the requirement of CUDA enabled GPUs in most machine learning techniques. After the realization of such limitations, we diverted our goals to developing a proof of concept with machine learning computations offloaded to the cloud.

Rest of this report will feature in-depth discussion of design and development of DeepGame along with some other related topics such as decision making or group dynamics and personal growth. One thing to keep in mind while reading this report is that the DeepGame project is more about developing a proof of concept to novel technique with broad applications than creating a stand-alone software.

2 Requirements Details

In this section we will discuss the final state of the requirements of DeepGame. This discussion will begin with an informal statement of goals and purposes. Then, we will discuss the requirements under a more formal manner by providing diagrams and specifications.

2.1 Goals and Purposes

The general idea of DeepGame is to put the user's visual likeness into a video game. This idea by itself is relatively simple, methods such as texture wrapping a photograph onto a video game model would be a basic approach. Our goal was to achieve this idea through the usage of deepfake algorithms. We also wanted to create a software that could be readily integrated into any game as a plugin. However, our ambitions were not matched by our abilities as a group. Therefore, we went for a reduction in our goals.

Final state of our goals is to have a proof of concept software that will take the image of a person and an appropriate style image to insert the person into a scene rendered by a game engine. We believe that this proof of concept can be developed into a plugin like we imagine given enough time and resources.

2.2 Use Case Model

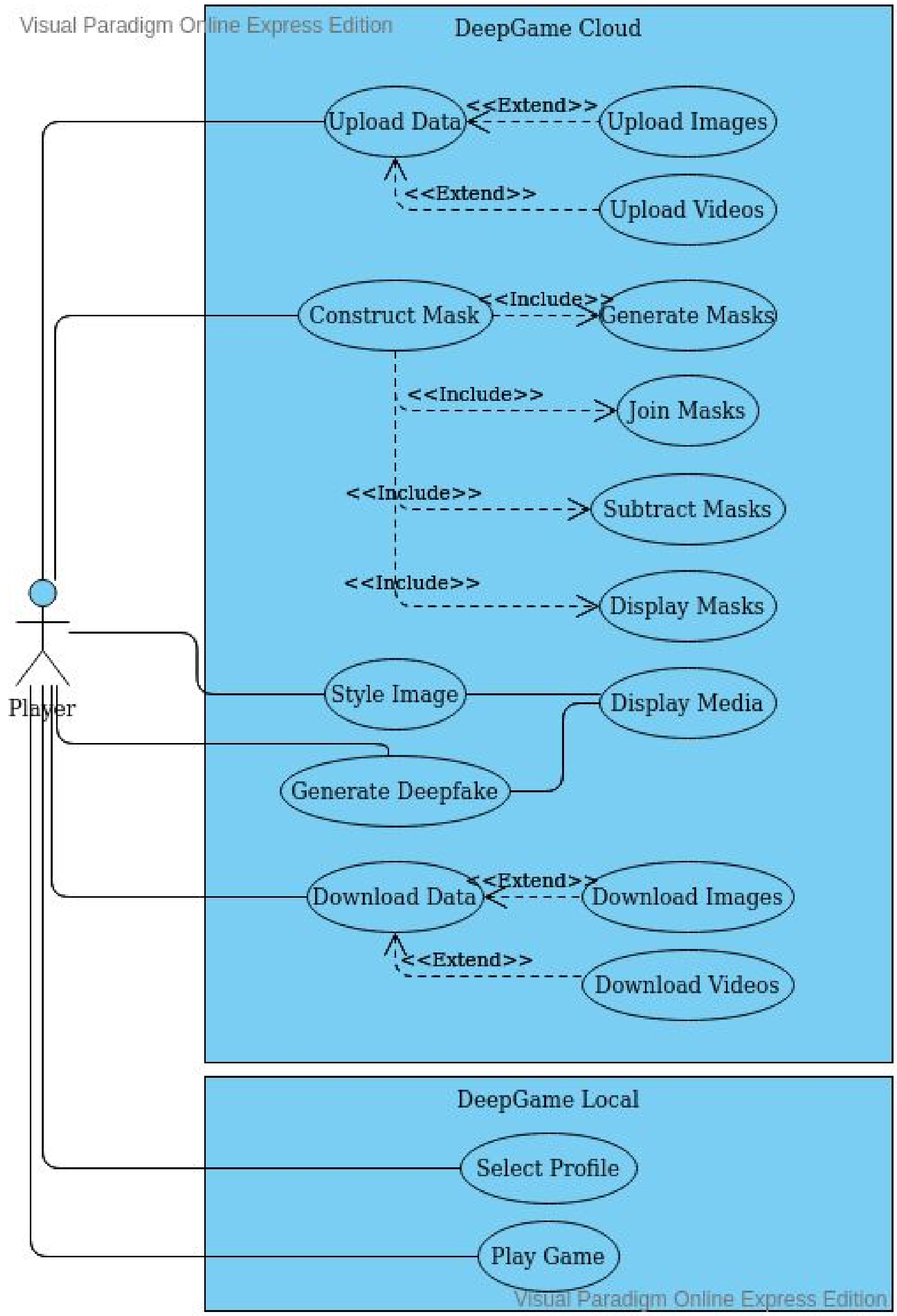


Figure 1. Use Case Diagram for DeepGame

As evident from the diagram, DeepGame is split into two main systems. One is in the cloud on Google Colab, other is a video game that runs on the local machine. Cloud part handles processing of data and producing deepfakes to be used later in the video game. Various use cases in the cloud system are the processing steps. Those steps can be executed in order and get a deepfake to use in the game, or singular parts can be used to produce intermediate products.

2.3 Flow of Events

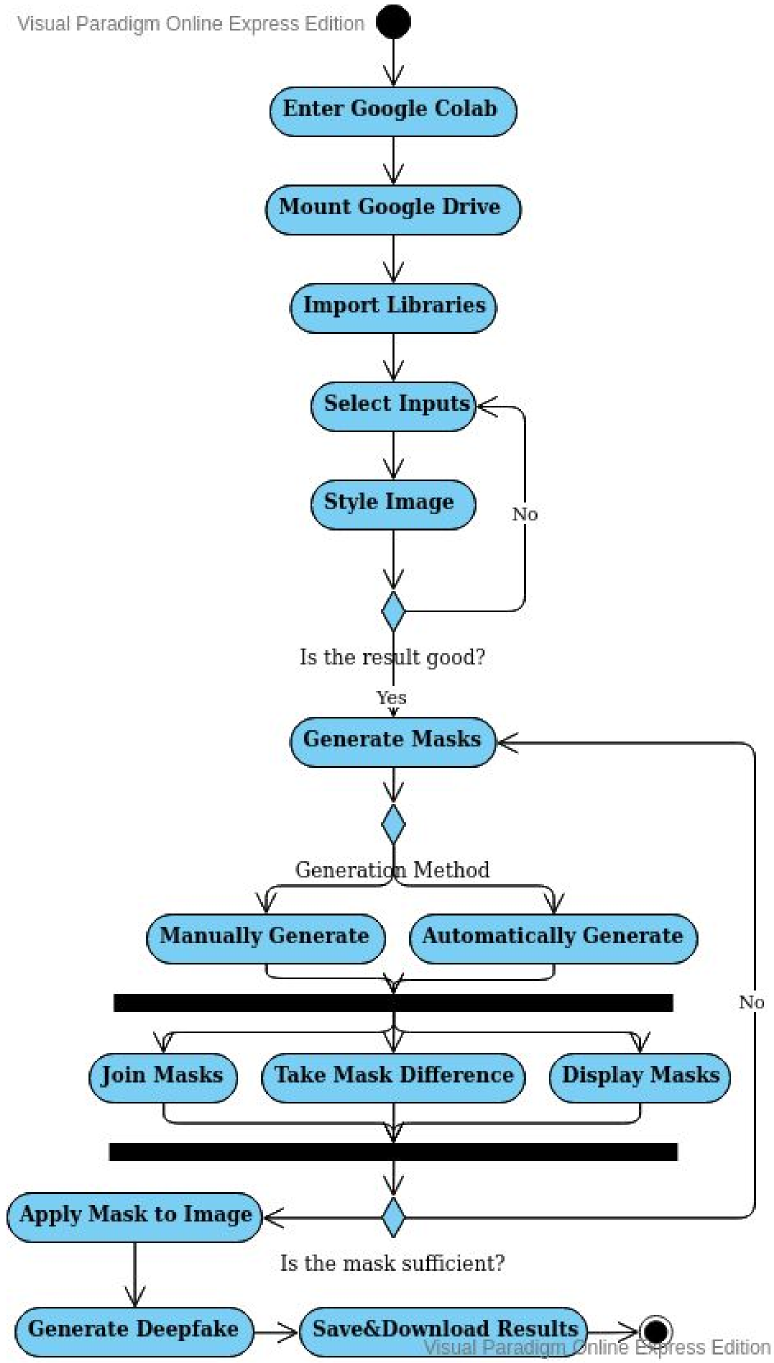


Figure 2. Cloud Side Activity Diagram

The cloud side is hosted on the Google Colab platform. Because Colab uses Jupyter notebooks as its foundation, usage of Colab is fairly linear. Moreover, the process of taking inputs to a deepfake applied video is fairly linear. Only problem is that certain steps require human semi-supervision for judgement of quality and execution of tasks beyond computer comprehension. Those are the steps that introduce loops to the activities.

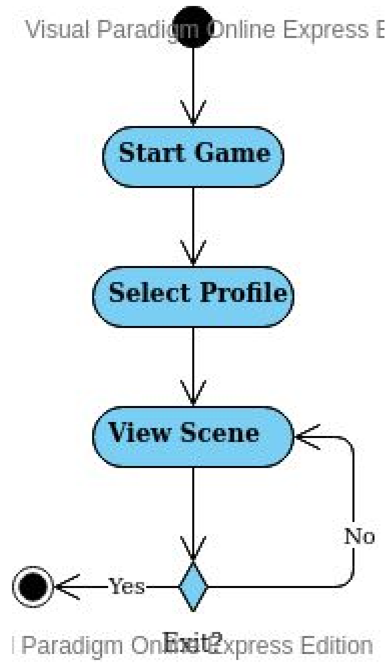


Figure 3. Game Side Activity Diagram

The game is for demonstrating DeepGame only. It does not feature any gameplay elements.

2.4 Limitations

The main architectural limitation is that the machine learning part of the project has to be executed on the cloud. That is because the machine learning ecosystem is based on NVIDIA's CUDA enabled GPUs. Therefore, a part of the project has to be on the cloud. Effects of this could be remedied by implementing a local client that seamlessly connects the cloud server and the local game. However, this was not done.

Limitations of what is done is that the particular style transfer technique chosen also transfers the colors between images. This is due to a tradeoff between usability, performance and quality. Main consequence of this is that a style image must be carefully chosen to not corrupt the result. A similar limitation is that mask generation step requires human supervision and intervention to achieve good results, unless image format is heavily restricted to enable a more hands-off approach.

2.5 Assumptions

Our first assumption is that the user has a computer with internet access and a Google account. Although it is possible to access the cloud side of the project with other devices, running the game requires a computer. Another assumption is that the user has a computer capable of running a barebones game. The final assumption is that the user is knowledgeable enough to execute tasks requiring human intervention.

3 Final Architecture and Design Details

Due to requirements of specific hardware and limitation of our access to such hardware, we split the project into two parts. There is a cloud side responsible for processing the user data and transforming it into a format ready to be used by the video games at the user side. In this section, we will discuss the two sides separately.

3.1 Cloud Side

3.1.1 Subsystem Decomposition

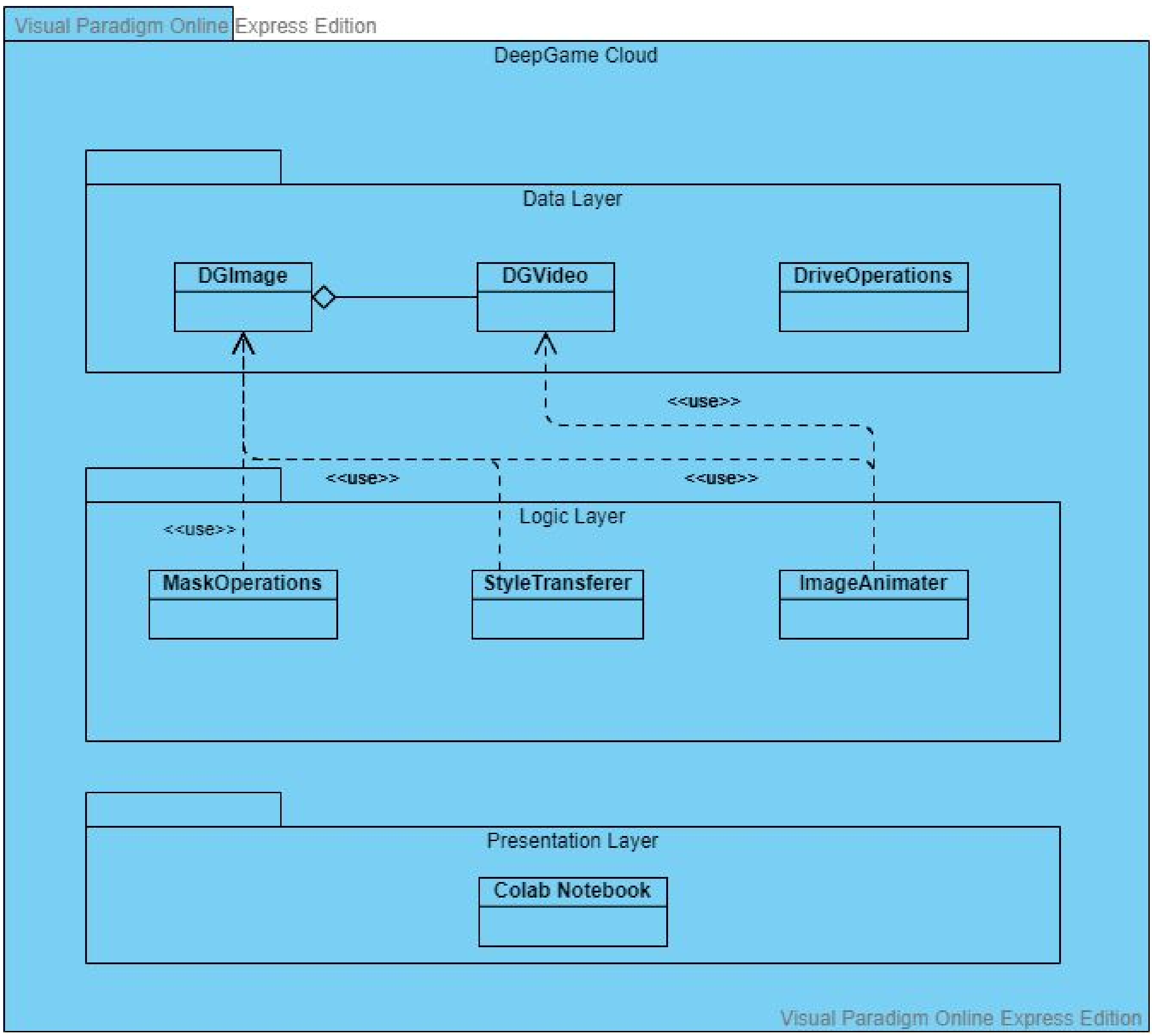


Figure 4. Cloud Side Subsystem Decomposition

The cloud side uses a three tier architecture.

Data layer consists of objects that are passed around the logic and presentation layers, along with a helper class that provides Google Drive utilities. However, data objects themselves contain a lot of utility methods that could be placed in another class in the logic layer. This was not done to keep the code simpler. Data layer also has external dependencies not shown here for input/output and some image processing utilities.

Logic layer consists of objects that provide the necessary functionality to DeepGame. The three classes contain methods to transform and process input media. MaskOperations is mostly self-contained. However, the other classes require importing machine learning models and libraries from external sources such as GitHub and TensorFlow Hub.

Presentation layer is a Jupyter notebook to be hosted on Google Colab. Google Colab provides a virtual machine that hosts a Jupyter notebook. This notebook is used to keep the explanation and scripts that execute various tasks. Because the Colab instance is reset at each new login, a data loader script is used to automatically set up the dependencies inside the instance.

3.1.2 Data Layer

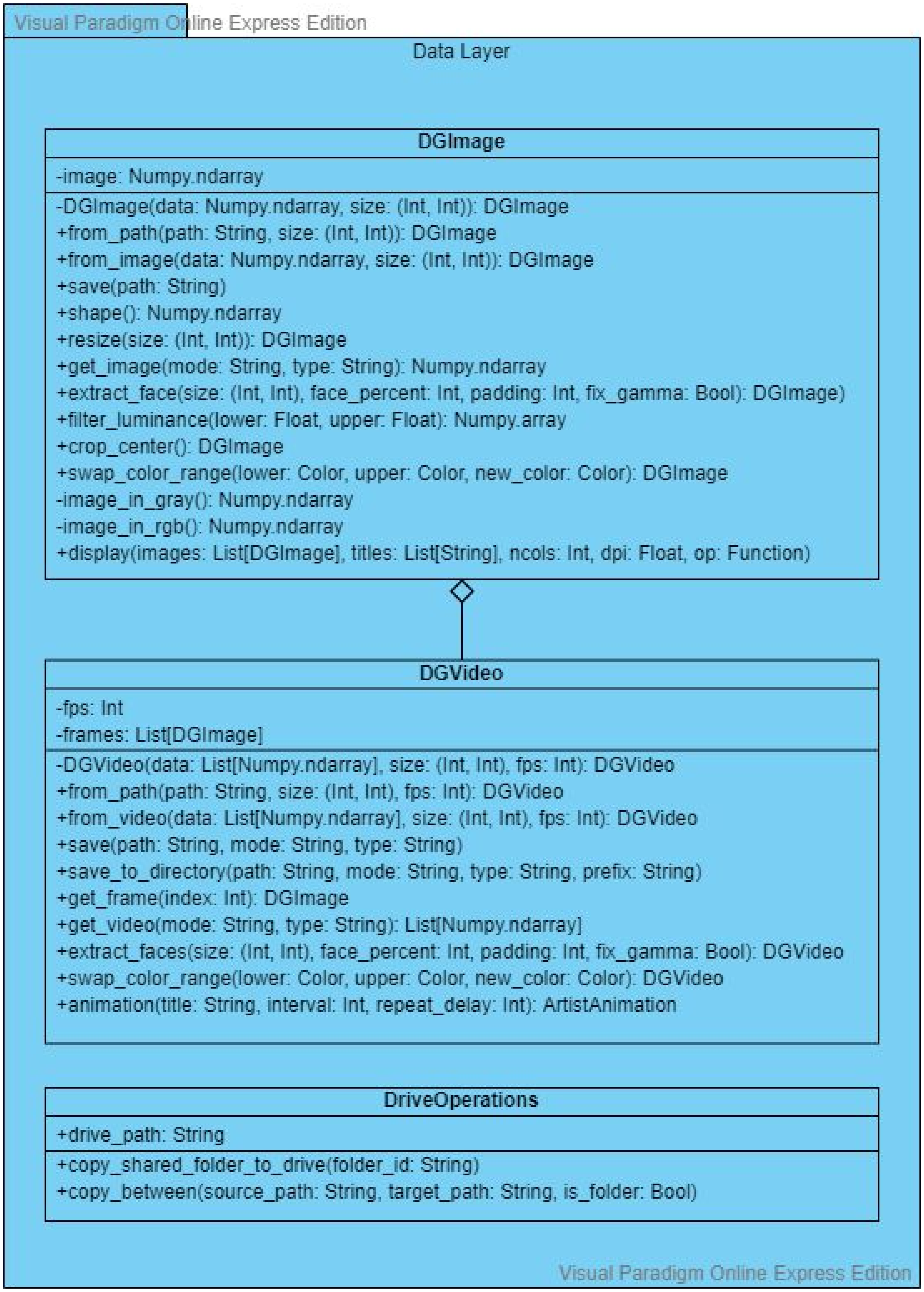


Figure 5. Data Layer Class Diagram

Data layer consists of three classes. Two for actual storage of data and one for cloud storage utility. We will begin with the utility class then move on to the data classes.

DriveOperations is a simple utility wrapper for providing access to Google Drive from Google Colab instances. Machine learning parts of the project requires the usage of models large in file size, thus they do not fit in a GitHub repository. So, we use Google Drive as a cloud storage provider and publicly share the models from a shared folder. DriveOperations enables us to easily import the models from Drive to the Colab instance.

The Python ecosystem for machine learning and image processing mostly revolves around Numpy data structures. DGImage is a wrapper around the Numpy array that represents images. Each image processing library has their own format requirements for the inputs. DGImage, enforces a format on the images it encapsulates but provides access methods to convert the image to any format needed. This results in a unified image object to be used across the DeepGame cloud side. Enforced format comes with immutability. These result in a more robust system at the cost of some performance. Besides holding the image data, DGImage provides many methods for utility or small image processing tasks. Richness of methods in the data layer enables the logic layer to be simpler and more direct.

As videos are made up of individual images in the form of frames, the same philosophy is applied for DGVideo. DGVideo is made up of DGImage objects representing the frames of the video. This architecture enables DGVideo to take advantage of DGImage’s features and apply an operation to the entire video to transform the video one frame at a time.

3.1.3 Logic Layer

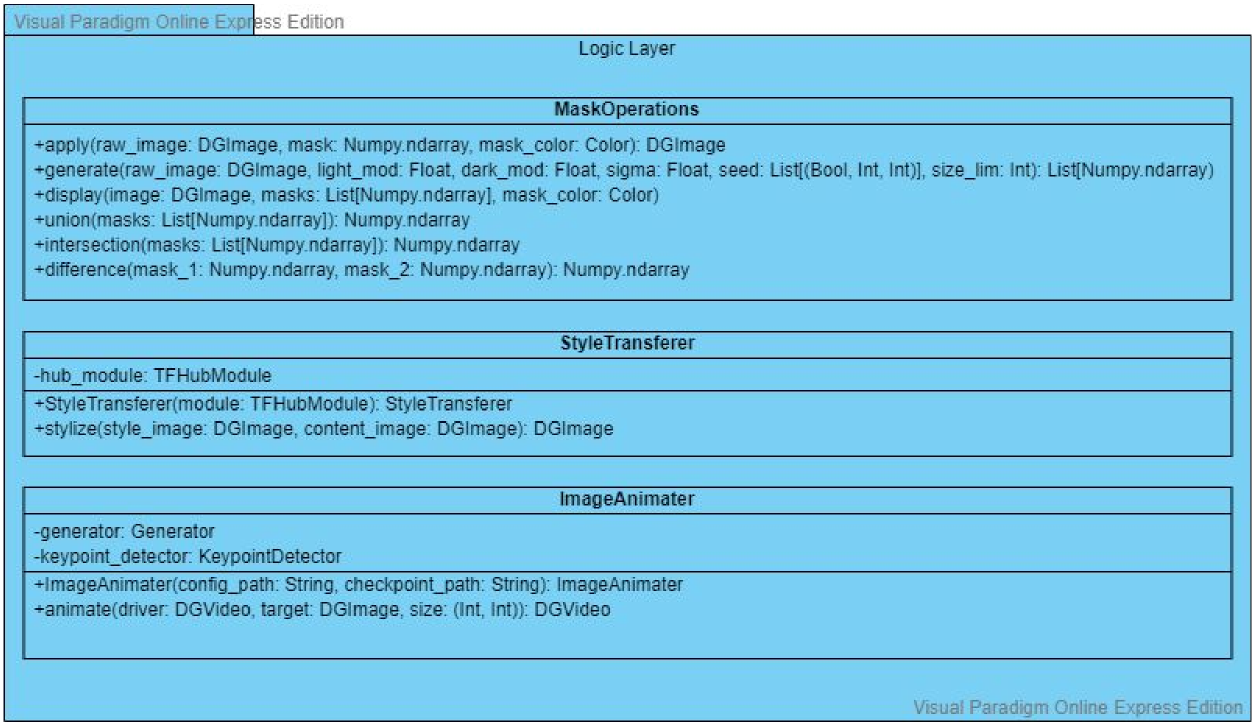


Figure 6. Logic Layer Class Diagram

Logic layer consists of three classes. MaskOperations is for generation and building of masks to cover the background of an image. StyleTransferer and ImageAnimater classes use machine learning libraries to style an image and create a deepfake video respectively.

The MaskOperations class takes a DGImage and manipulates a boolean matrix that denotes whether a particular pixel should be covered by the mask. The generate method produces candidate masks and union, intersection, difference methods manipulate those candidates to build other masks. Finally, the apply method applies a mask with the given color on the given image to create a new image.

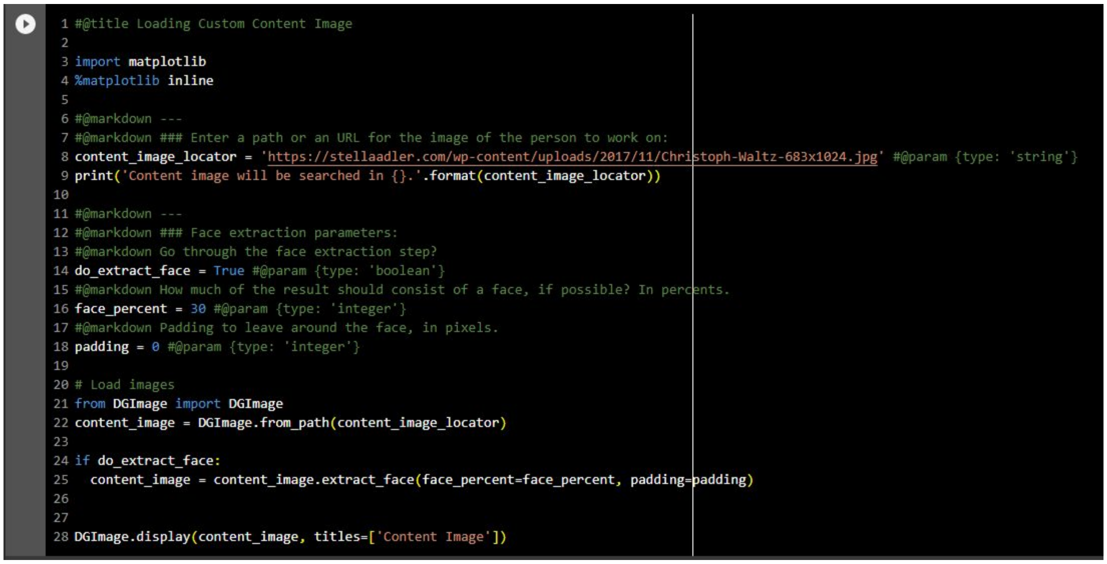
The StyleTransferer class uses a machine learning library to apply the style transfer technique to the given images. Its main purpose is to lose the photorealistic quality of the input images.

The ImageAnimater class uses a machine learning library to create the deepfake of the person in the given image driven by the given video. Its main purpose is to put a single image in any pose needed.

The logic layer is discussed in detail later in the next section.

3.1.3 Presentation Layer

The cloud side presentation layer can not have a traditional class diagram. Because, we use a Jupyter notebook as the interface. Below is a discussion of Jupyter notebooks as an interface.



```
#@title Loading Custom Content Image

import matplotlib
%matplotlib inline

#@markdown ---
#@markdown ### Enter a path or an URL for the image of the person to work on:
content_image_locator = 'https://stellaadler.com/wp-content/uploads/2017/11/Christoph-Waltz-683x1024.jpg' #@param {type: 'string'}
print('Content image will be searched in {}.'.format(content_image_locator))

#@markdown ---
#@markdown ### Face extraction parameters:
#@markdown Go through the face extraction step?
do_extract_face = True #@param {type: 'boolean'}
#@markdown How much of the result should consist of a face, if possible? In percents.
face_percent = 30 #@param {type: 'integer'}
#@markdown Padding to leave around the face, in pixels.
padding = 0 #@param {type: 'integer'}

# Load images
from DGImage import DGImage
content_image = DGImage.from_path(content_image_locator)

if do_extract_face:
  content_image = content_image.extract_face(face_percent=face_percent, padding=padding)


DGImage.display(content_image, titles=['Content Image'])
```

Figure 7. Example Jupyter Notebook Annotated Cell

Jupyter notebooks are made up of cells that contain and run scripts. These scripts can be annotated to turn the cell into an interactive form. Above is one such cell.

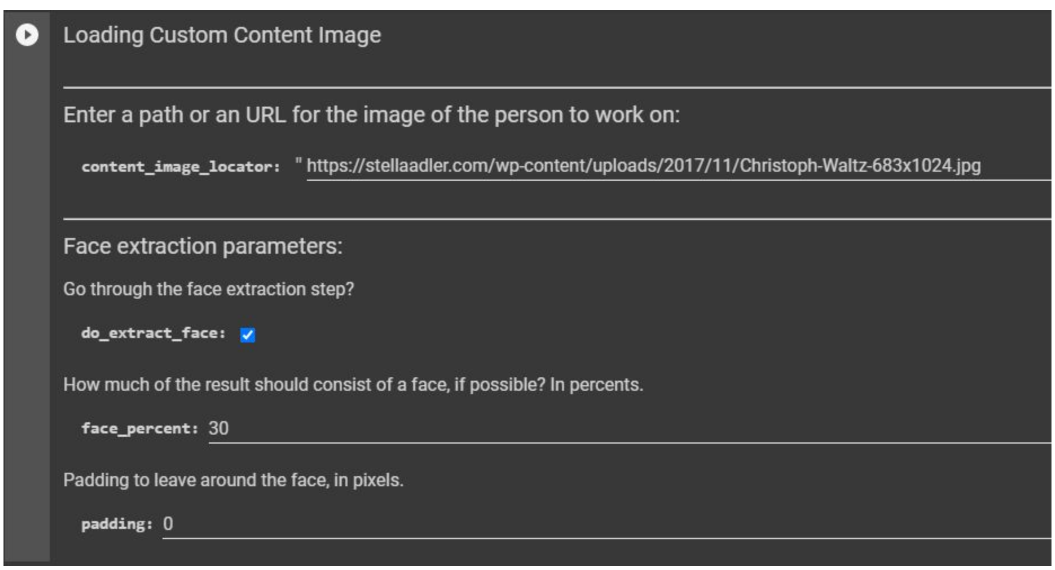


Figure 8. Jupyter Notebook Cell Viewed as a Form

The same cell looks like the figure above, when viewed as a form. The changes to the form are input to the variables inside the script. Hence, Jupyter notebooks can be used as a fully featured user interface for the presentation layer. But do not have a class diagram.

3.2 Game Side

In this section the details of the blueprint classes of the game is provided.

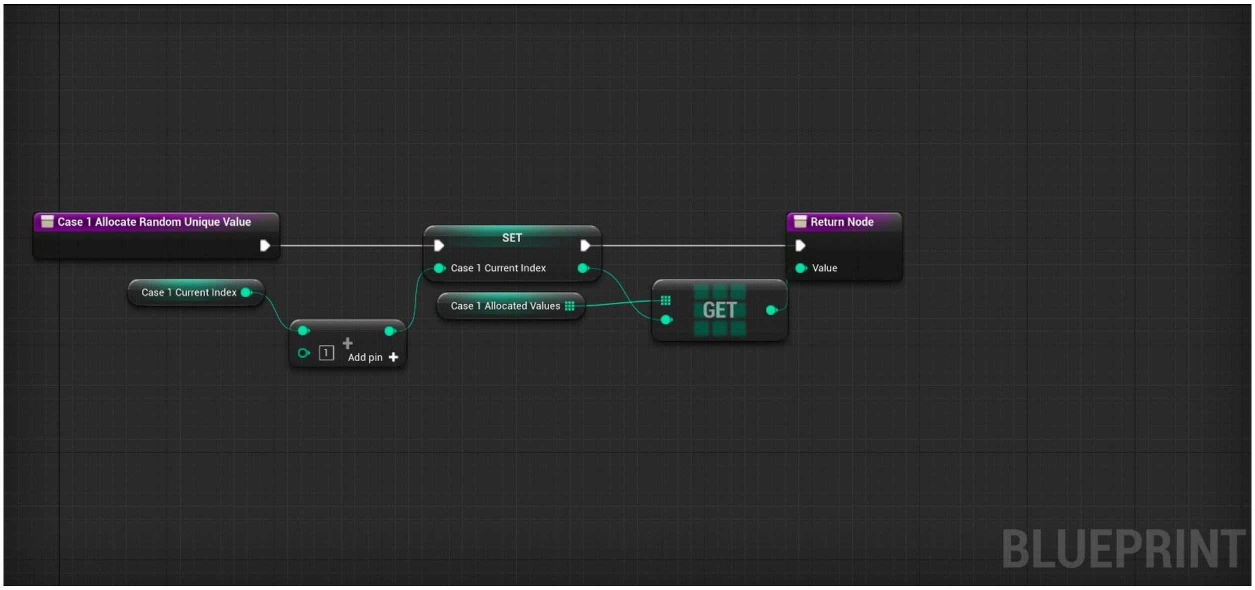


Figure 9. Blueprint class of the randomness manager

Randomness manager controls the animation of the gamblers. In order to make the game scene look as realistic as possible every gambler in the table does different animations and changes their animation in every 10 seconds. The duration of the animation can be changed, and set to different values. New animations can be added easily.

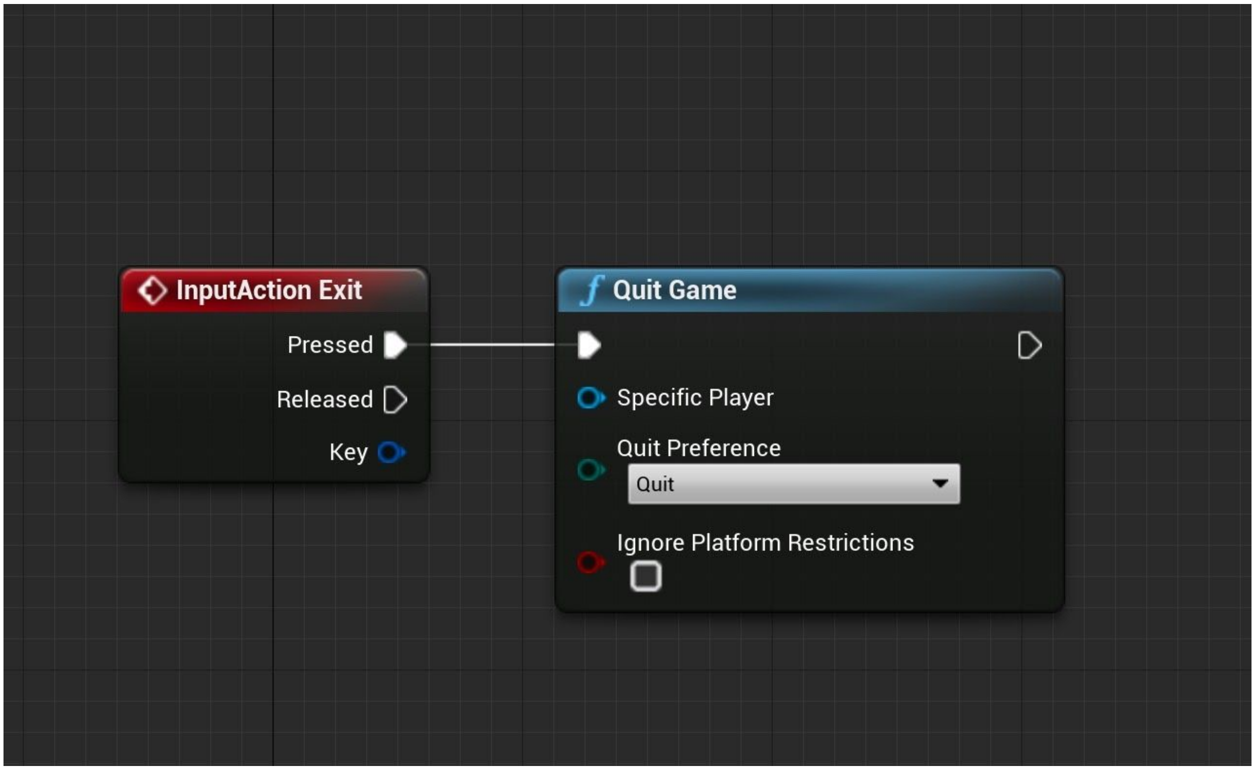


Figure 10. Blueprint class of the main menu

In this blueprint class the exit action is implemented. When the user presses on the exit button the game quits.

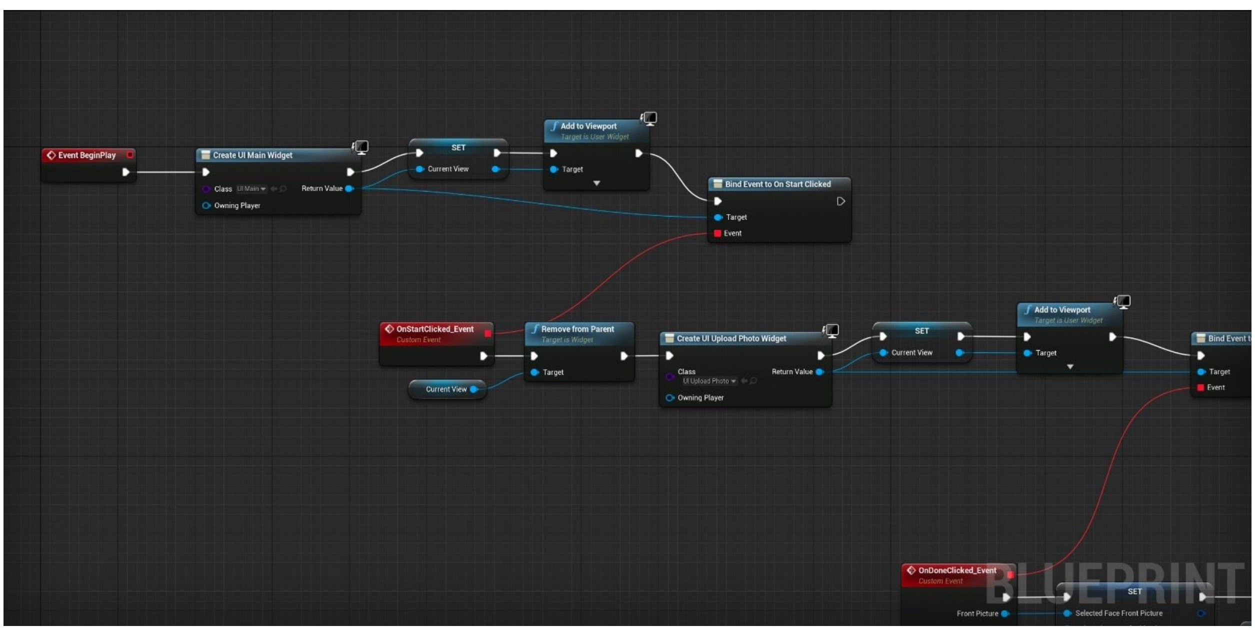


Figure 11. Blueprint class of the main game part I

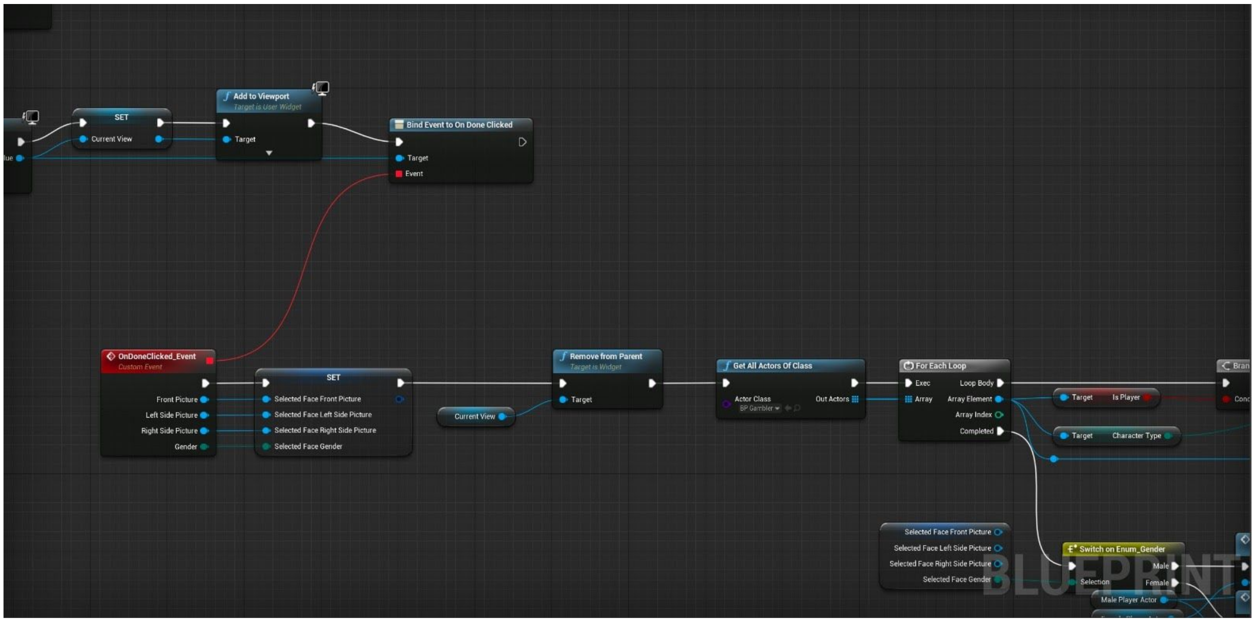


Figure 12. Blueprint class of the main game part II

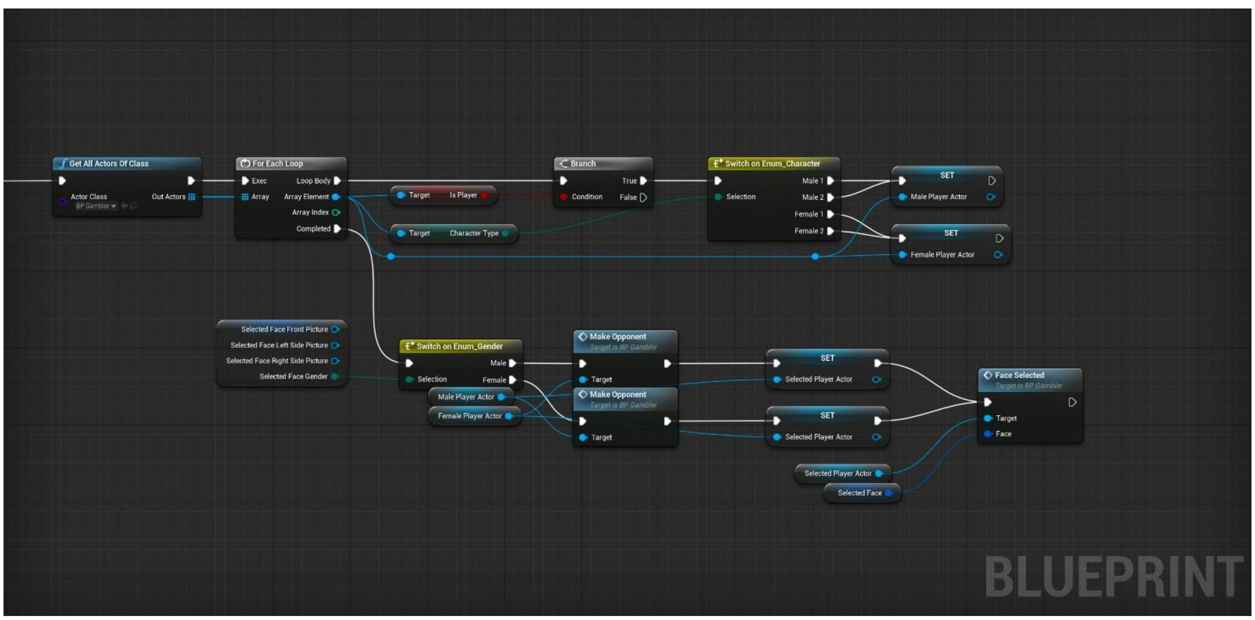


Figure 13. Blueprint class of the main game part III

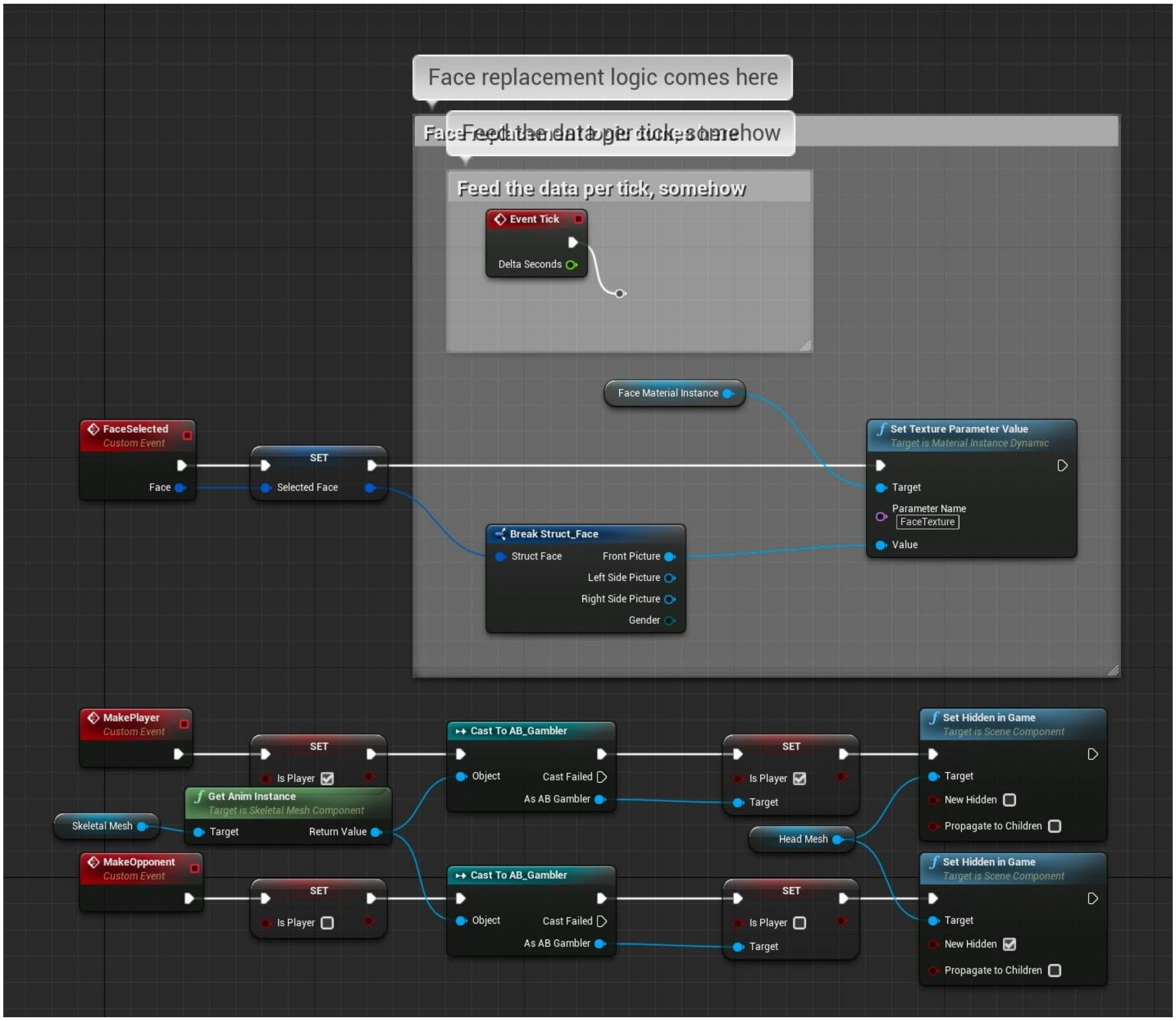


Figure 14. Blueprint class of the gamblers part I

In this class of the game user can upload photos: left side,right side and front. User can choose the gender and the part where face replacement logic takes place stated explicitly.

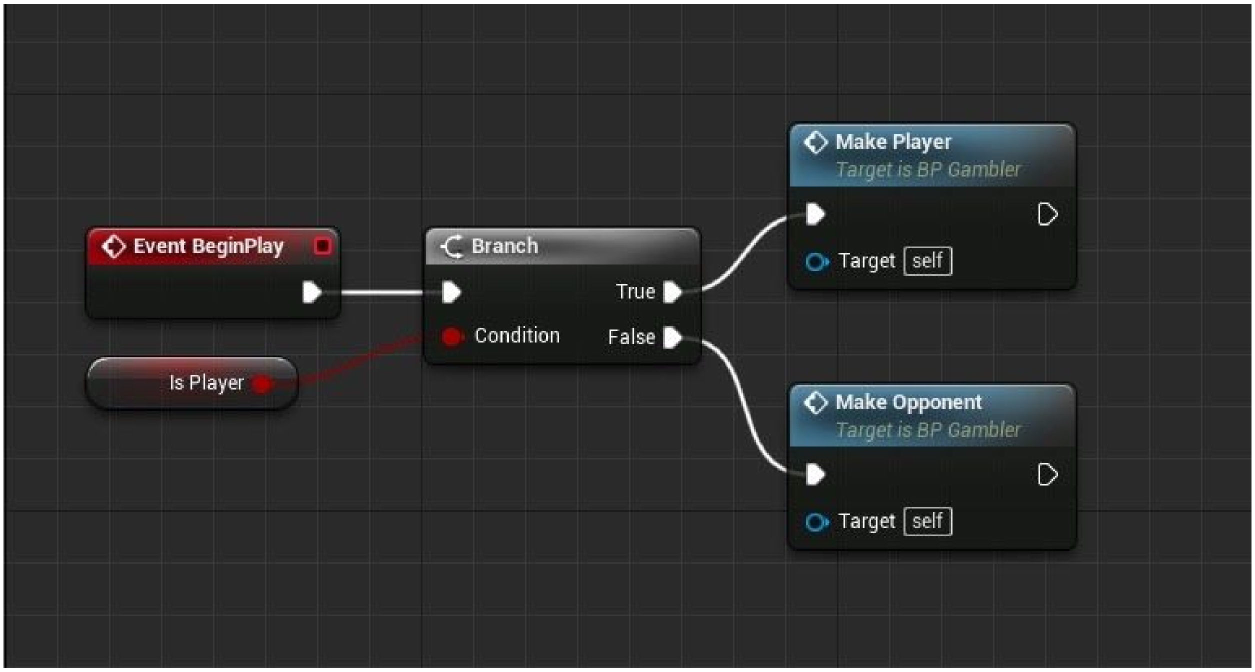


Figure 15. Blueprint class of the gamblers part II

In this part the gamblers which are opponents and players are chosen.

4 Development/Implementation Details

As already mentioned, there is a clear separation in project components. In this section, we will discuss the two sides separately.

4.1 Cloud Side

This section will start with a discussion of an integral part of the project, Google Colab. Then, inner workings of the cloud side will be explained one subsystem at a time.

4.1.1 Google Colab

Google Colab is a service provided by Google for free for anyone to access [1]. Briefly, it is a Jupyter notebook running on a remote instance with freely provided GPUs or TPUs. Since Colab takes Jupyter notebooks as its foundation, it is very well integrated with the Python ecosystem along with git and GitHub. Moreover, being a Google product, it provides easy access to Google Drive instances of users. This access to Drive allows it to be used as cloud storage for users. This helps offload privacy security concerns to Google, as long-term storage of private data is a privacy and security concern. Since any cloud solution requires trusting a third party, this was deemed acceptable. Moreover, usage of Drive as cloud storage is not mandatory. Users can use their personal computers or other cloud storage providers with minor modifications to the source code without affecting the overall product.

On the other side, there is Colab's integration with the Python ecosystem. Since machine learning research basically runs on the Python ecosystem, this integration enables us to run any machine learning task we may need on Colab. Moreover, being built on Jupyter, Colab provides an interactive and intuitive interface to its users.

We selected Google Colab because it delivers powerful features for free. However, after getting invested into Colab, we discovered a downside to it. In order to keep Colab a free service, access to more powerful features of Colab such as GPU usage gets limited based on recent usage on user's end and load on Colab's end. The problem is that there is no logic to why, when and for how long this access gets limited. Colab supposedly offers continuous GPU access 12 hours at a time and users accessing the GPU at short bursts should face usage limits less often. However, that offer did not match the reality and we faced days of unproductivity where we could not use Colab for our needs. This problem was somewhat remedied by using alternate accounts, because the same Drive instance can be accessed from different accounts. But, that did not turn out to be a sufficient solution when both accounts hit the usage limit concurrently.

Another challenge with Colab was that Colab instances are not persistent. Meaning, data within Colab instances get deleted with each restart of the instance. We solved this problem by using GitHub repositories for code storage and Google Drive for data storage as a means of persistent data management and software deployment. One exception to the lack of persistency is the Jupyter notebooks themselves. They, and thus any code written in them, is persistent. Hence, we wrote an initializer script that pulled all the necessary data after authentication with Google for Drive access.

Rest of the Colab notebook features code meant to take the user through steps of data processing along with explanatory text. Only interaction required is selecting input files, running scripts and going through the semi-supervised process of mask creation through image segmentation. The user can see results of the intermediate steps during this process. However, due to lack of a reliable unsupervised mask generator, the process cannot be streamlined down to selecting the inputs only.

4.1.2 Style Transfer

Style transfer is a somewhat recent innovation. Our implementation takes the work of Ghiasi et al. in 2017 as the foundation [2]. The earlier work of Gatys et al. has the advantage of the option of keeping the color palette unchanged during the style transformation [3]. But, Gatys' work runs on the Lua version of Torch, which is not integrated into Colab and effectively impossible to implement. That is because installation takes a lot of time and because of the non persistent nature of Colab, has to be repeated with each new use. There is also a PyTorch implementation of Gatys' work which sould work in theory. However, the code uses a deprecated version of Python and its supporting libraries. So, it was not usable. On the other hand Ghiasi's work is readily available on TensorFlow Hub, supports styles without restriction and works much faster than Gatys' method. The only problem is the changing color palette after the transformation. Which requires selecting your style image carefully. We deemed the sacrifice in fidelity and work put into appropriate style image selection worth the gain in speed and ease of use.

Below is a table showing examples of the style transfer applications with different inputs. The first row features unstyled photographs. The first column features computer generated or drawn portraits of video game characters. The intended effect is to change the style from photorealistic to one matching the game art. Results show that the effect is partially achieved. Rough features of photographs are replaced with smooth surfaces of computer generated imagery. However, the colors also get mixed up during the process. Also, the bottom right example places what looks to be half an eye in the forehead of the result. Because the background will be filtered out, glitches in the background are acceptable. Same does not hold true for glitches on the face. Color invariant transformation approach of Gatys' style transfer would work better. However, we were not able to implement it on Colab. Hence, we will settle for supervising the results and selection of the style image instead.

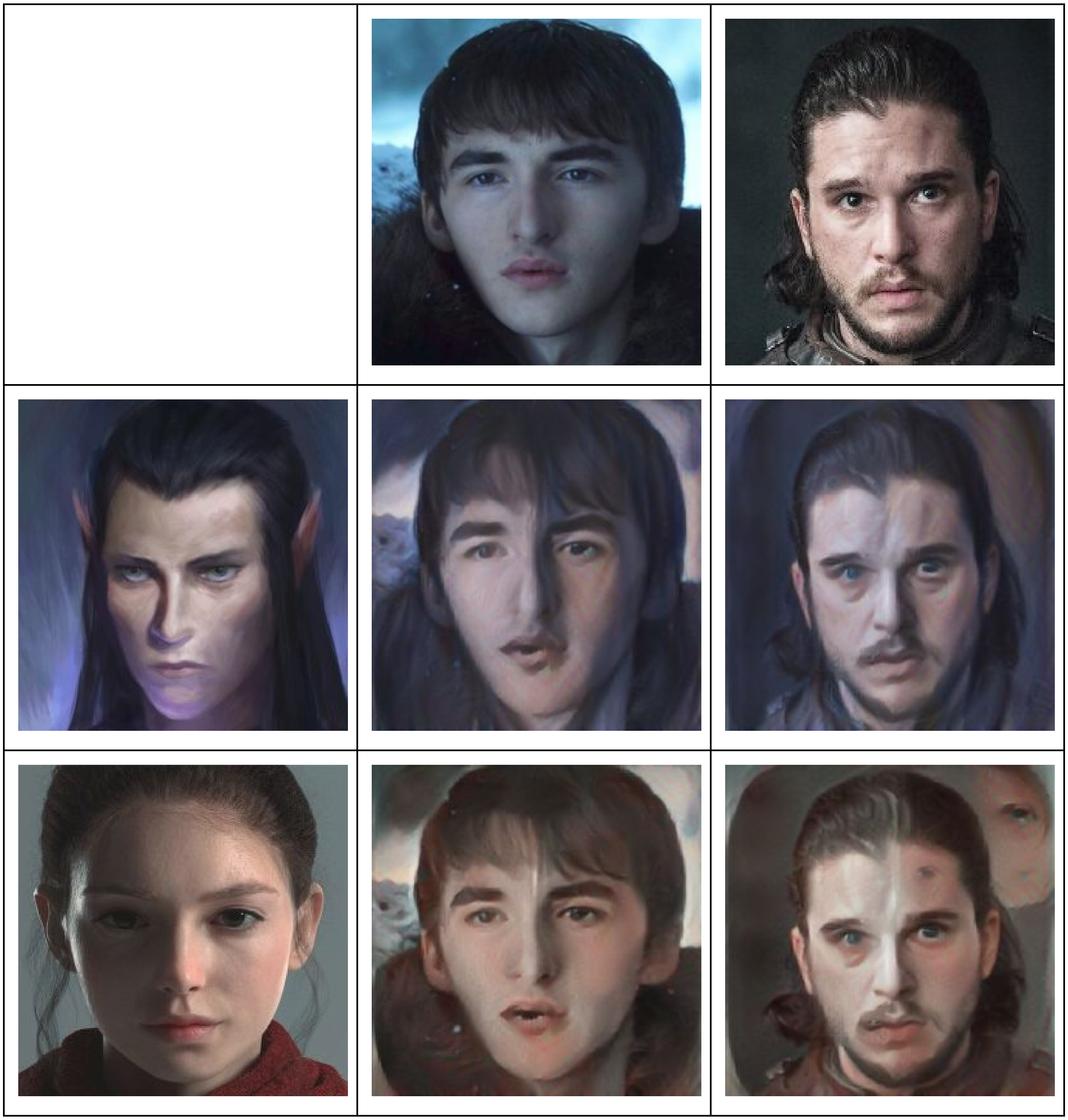


Figure 16. Table of Style Transfer Applications

In order to effectively use Ghiasi's work, we wrote a manager subsystem that deals with the initialization of the TensorFlow Hub model and input/output of the images. This subsystem was implemented in the form of a Python class.

4.1.3 Image Animation

First Order Motion Model for Image Animation is a research on deepfake creation using a single target image and a driver video that was published in late 2019. It is a quite recent development.

This research, Image Animation in short, is the basis of the idea and the technology driving the DeepGame.

Image Animation allows replicating mimics and poses, taken from a driver video, in the input image. In other words, it animates the image of a person to match the motions of another person given with the driver video. This enables DeepGame to require only one or a few images of a person to animate them as needed.

The image animation subsystem is a manager class written to handle the input/output and preprocessing of data that is to be used with the Image Animation research. A powerful feature of Image Animation is that it works without problems when supplied with an image with the background masked. This feature is used to animate images with a background mask applied. The resulting video keeps the background mask and distorts it around the person as the motion requires. Later, this mask is used to filter out the background and apply the deepfake product to the game itself.

Because video cannot be displayed on paper, there is no example for this process.

4.1.4 Mask Generation

Mask Generation is the process of finding a mask for the image of a person that covers the image background and only leaves the person's face if possible and person's self if not. Our implementation does not depend on previous research. But, this topic is not a new idea. Therefore, there are various techniques that can be applied to solve this problem [4,5]. After our research, we settled on image segmentation with a watershed transformation seeded from pixels manually selected or automatically selected via choosing the darkest and brightest spots in the image.

The automatic approach works great when the background is clearly distinct from the person. But, downgrades to producing multiple mask pieces over the entire image. To improve upon this, we implemented a mask manipulating system. Where masks can be joined or removed from each other. This introduces human labor into the system but having the possibility of a better result with extra work is better than not having the option at all.

The manual approach works by selecting pixels that must be covered by the mask and pixels that should not be covered by the mask. Then, a different mask for each case is generated and the mask for the uncovered case is subtracted from the mask for the covered case. The result is a mask that fits the described requirements. The manual mask generation can be used in conjunction with the automatic generation via the mask manipulation system to improve upon the result.

Below is an example featuring automatic mask generation. We can see that certain elements of the image are identified and segmented well. Meanwhile, the approach fails to identify the clothing of the person as one segment. Also, some segments have holes in them.

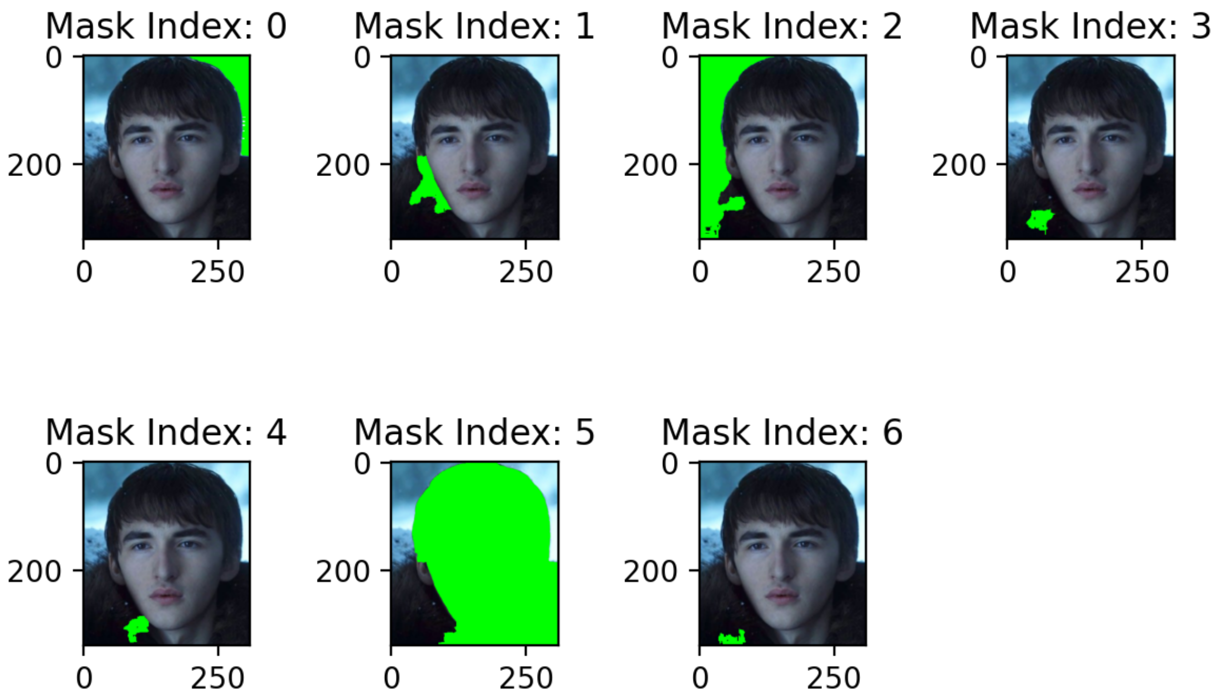


Figure 17. Automatically Generated Masks

From the pieces we see that if we join the masks of indices 1 and 5, we cover the face. Then, we can flip the intermediate result to get a mask that leaves the face open and covers most of the irrelevant parts. The result of that can be seen below:

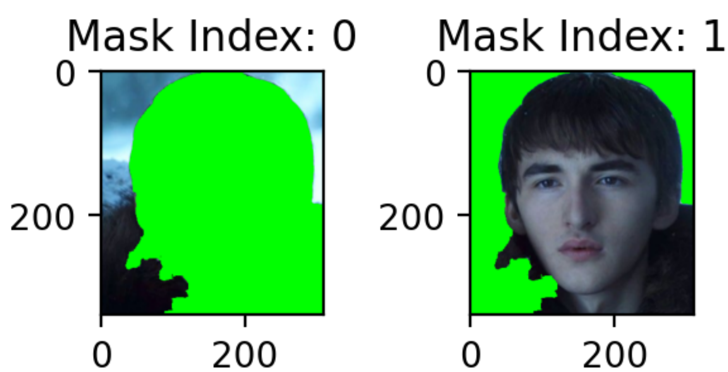


Figure 18. Resulting Mask Constructed as Described

Mask generation tends to work better on unstyled images, because style transfer tends to lose valuable image information. Once a mask is constructed on an unstyled image, that mask can be used on the styled result. One important point is that style transfer enforces an image size on the output. A mask fits the size of the image it is constructed on. Care must be taken to ensure the mask constructed on an unstyled image fits the styled result.

4.2 Game Side

For the game implementation part, all commercial engines available in the internet have been assessed for the depiction of the algorithm and gamification in terms of usability, community support, transparency of the source code, as well as the performance and visual quality. Since the game logic is usually being run every tick, meaning that 30+ times every second, choice of the programming language and the engine architecture is extremely important for having a decent gameplay experience.

In our implementation for the game we have used Unreal Engine. What made Unreal Engine a better implementation candidate among other engines such as Unity are as follows: open source, great community support, support for most up to date rendering techniques, C++ programming language also Blueprint visual programming for basic logic, visual shader programming.

Comparison of the other candidates and their noteworthy properties are below:

1. Unity
    a. C# programming language
    b. VM based JIT process environment
    c. Good community support
    d. Closed source
    e. Easiest learning-curve
    f. Support for most up-to-date rendering techniques
2. CryEngine
    a. C++ programming language
    b. Bad community support
    c. Closed source
    d. Support for most up-to-date rendering techniques
3. Unreal Engine
    a. C++ programming language, also Blueprint visual programming language for basic logic
    b. Visual shader programming (blueprint-like)
    c. Great community support
    d. Open source
    e. Easy-medium learning-curve
    f. Support for most up-to-date rendering techniques
4. Torque 3D
    a. C++ programming language
    b. Terrible community support
    c. Open source
    d. Difficult learning curve
    e. DirectX 9 support, very outdated
5. JavaScript-WebGL based engines (Three.js, Babylon, Play Canvas)
    a. JS programming language

b. Decent community support
c. Medium learning curve
d. Worst runtime performance, since JS is an interpretation-based language

Based on the assessment which has been made, Unreal Engine 4 has been chosen as the development environment. The choice is very important as it affects implementation of the game logic, interaction of the face swapping algorithm via the output as texture stream with the character’s 3D model’s face-texture mapping. Therefore, Unreal Engine’s strong community support, the engine being open source, support for latest rendering techniques and appealing development environment with Blueprints and visual shader editor, C++ based development have been made Unreal Engine to be the best choice for the project development.

Unreal engine starts working when Deepgame.uproject file is opened. Version 4.25 is used in the implementation. Any version greater than 4.25 is compatible and works without a problem. In the application there is a map named Main. The main is an abstraction in which how the scene is created, logics, actors, furniture is made. The characters sitting around the poker table are actors. Actors have gender specification as female and male. The randomization manager has an animationIndex which allows us to animate four characters differently. With the getGameMode function actor's game mod is obtained.

The camera in the game is static and the player can use their mouse cursor to stick around the room. The interface is developed with Slate UI Framework provided by the Unreal Engine.

C++ is used in file picking to map the plane. In order to use face transition faces of the actors should be replaced with users input photos/videos. In the file picker with C++ we are converting the file into a byte array structure. From byte array structure the file is converted to texture. Remaining parts of the game is done in Blueprint.

The game is done with object oriented concepts. The Blueprints Visual Scripting System provided by the Unreal Engine is used to define Object Oriented classes or objects in the engine. The gameplay scripting is done by node-based interface in Blueprint.

For the face replacement implementation main approaches are specified below with their complexities.

1. Hide the head; put a cube and stream the face data to the cube with planar mapping (easier/less time-consuming method, complexity 1)
2. Or as a more advanced and realistic approach; planarize the face in the video; map the planarized face texture to the 3D model texture. (complexity 10)

Figure 19. Opening Screen of the game with exit and start options

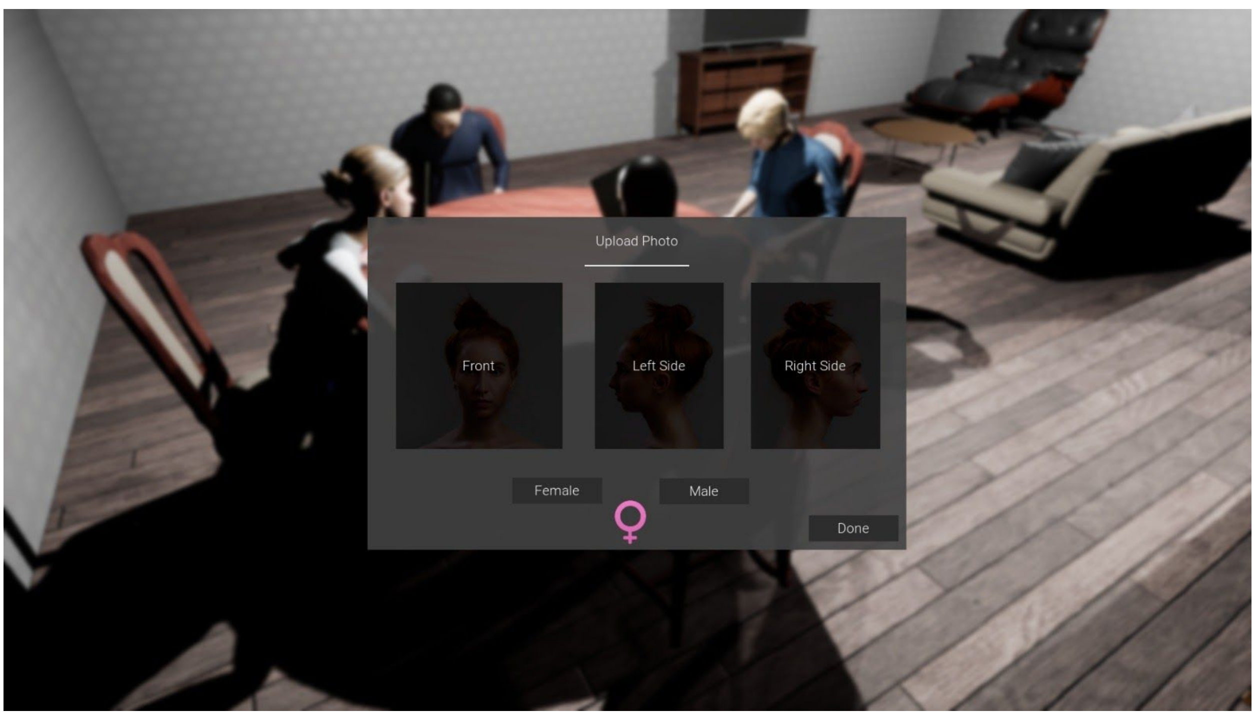


Figure 20. File uploading screen for the character face transition for female player

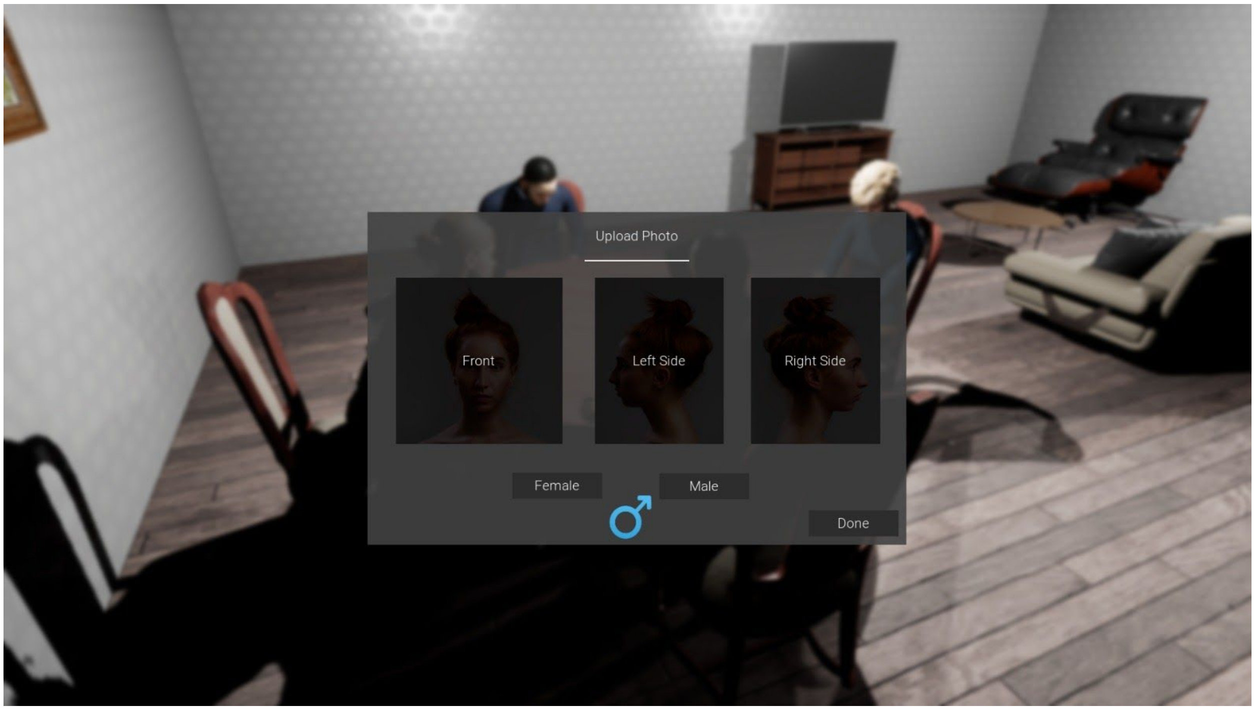


Figure 21. File uploading screen for the character face transition for male player

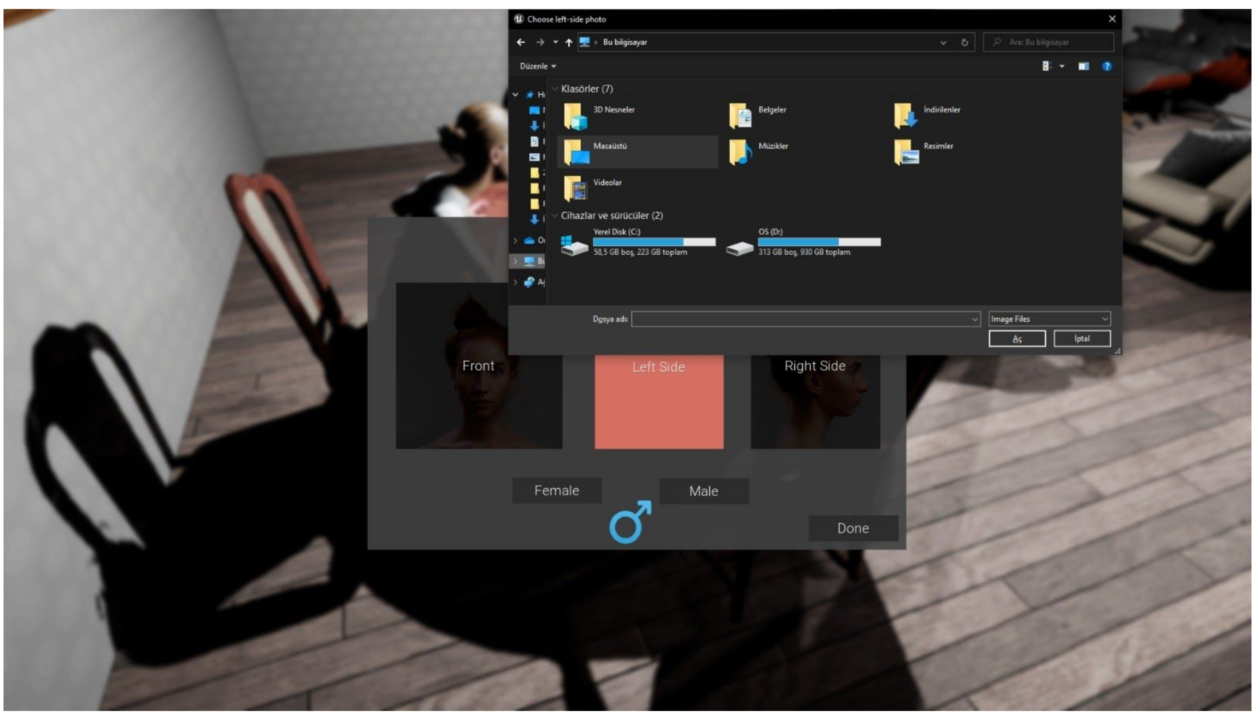


Figure 22. Uploading selected files

Figure 23. Uploaded raw file is attached to the chosen characters face

Figure 24. Showing the table from other perspectives

Figure 25. Details of the game: Sofa

Figure 26. Details of the Game: TV Unit and Chair

Figure 27. Details of the Game: wall decorations, wallpaper and hardwood floor

Figure 28. Details of the Game: chairs, console and painting on the wall

Figure 29. Game from a different perspective, obtained by moving the mouse and cursor

5 Testing Details

Due to the nature of the machine learning field, it is hard to test a project that takes machine learning as its foundation. This is even more pronounced for DeepGame. Because, unlike most other forms of machine learning tasks, measuring the quality of deepfakes is near impossible to achieve automatically. It requires a human eye to judge the results. Therefore, we were not able to implement comprehensive testing for a majority of the DeepGame project. However, for the cloud side we used our friends as sort of beta testers.

Besides the machine learning systems, DeepGame also includes support systems that help, manage and drive the application logic. They were tested in a small number of cases to ensure they work when used correctly. Because the DeepGame project is more about the development of a technique rather than the creation of a software product, that was the extent of our testing.

Testing for the game is done in different operating systems. The game can be played in Mac, Windows, Linux and other environments without a problem.

6 Maintenance Plan and Details

The DeepGame project as the proof of concept of a technique is not fit for long term maintenance. However, there are multiple ways of maintaining and improving the technique itself. The DeepGame technique hinges on the application of style transfer, image animation with deepfakes and image segmentation.

Style transfer is required for matching the art style of a given game. Our current implementation uses an all purpose style transfer model. The model itself can be improved. Moreover, the current implementation transfers the color palette of the style image to the target image. There are not as accessible style transfer methods that keep the color palette of the target image. Such a method would improve the possibilities of DeepGame and improve the visual fidelity of the result. It would also simplify the process. Because, the current implementation requires finding a style image that will not corrupt the colors of the content image. Which diminishes usability.

The current image animation method works well. However, the machine learning model utilized is trained on celebrity interviews. Although this covers most of the required use cases in DeepGame, it still leaves out some edge cases. A new model trained on a larger data set would definitely improve the quality and capabilities of DeepGame.

As a reminder, image segmentation is the process of separating the background of a portrait picture from the person’s face. The current implementation has a helper system but ultimately requires supervision from the user. We do not know of any method that can reliably and correctly segment images without enforcing strict photograph composition guidelines. However, mild guidelines together with machine learning may produce a more powerful image segmentation system that reduces work on the user's end.

In conclusion, foundational methods used in DeepGame are products of a rapidly improving field. As long term maintenance, subsystems of DeepGame can be replaced with improved ones.

7 Other Project Elements

7.1 Consideration of Various Factors in Engineering Design

7.1.1 Entertainment

In the project we have revealed, we have created a result that can take a leading role in the game industry with the technique of transferring face to the game character. Thanks to this technique, people will now be able to create and direct their own original game characters. The gestures, sounds and movements of the characters created with machine learning will also be unique. This technique, which is not common in the game industry, will begin to be used in other games and more realism will be captured in the game world.

7.1.2 Socialization

Our game has both a single player mode and a multi-game mode. Due to the multi-game mode feature, people will compete with their friends and will spend more time on the game. They will compete with each other over the virtual environment by putting the game characters in their place. In addition, they will meet more people and make many friends, and this will create a more social environment, especially among young people.

7.1.3 Privacy

DeepGame deals with data of people's likeness and processing of the said data. Therefore, privacy is an important aspect of the project. The current architecture of the project does not keep any private data. In fact, any external data is supposed to be maintained by the user themselves. This eliminates most of the concern for data privacy. For total privacy, only concern is the use of Google services for computing. Google services are supposed to be safe for this purpose. However the project's architecture is such that if one desires to avoid Google, they can host the project where they see fit with slight effort.

7.1.4 Ownership of One's Likeness

DeepGame processes people's photographs to use them in games. This raises the question of people using other people's images without their consent. Since people's access to publicly available photographs cannot be blocked, this is a concern that cannot be eliminated. On the other hand, people can use other people's images without their consent for their own purposes with or without DeepGame. Therefore, DeepGame does not create a new problem.

7.2 Ethics and Professional Responsibilities

This project uses personal data, user information and pictures. These data could be considered as sensitive, thus as an ethical principle our game does not violate personal privacy. We will not sell personal data to third parties or organizations.

For this purpose, we developed the project with total control of data by the user and the usage of Google services in mind. Since Google can be considered a trustworthy service, this should be sufficient for privacy and security of personal data. However, if the user does not trust Google, the project can be modified to use any storage provider with minor effort or the Colab notebook can be hosted on a private machine with slightly more effort.

7.3 Judgements and Impacts to Various Contexts

	Impact Level (out of 10)	Impact Description
Public Health	N/A	N/A
Public Safety	N/A	N/A
Public Welfare	N/A	N/A
Global Factors	7	The DeepGame is a globally pioneering project in the game industry. Due to the ability to transfer the faces of the players to the game characters, if necessary investments are made, it can make global changes in the gaming industry.
Cultural Factors	2	The DeepGame will not have direct cultural implications. Due to the multiplayer feature of the game, it will allow players from different cultures to play together. Therefore, it has the potential to create a cultural cohesion environment.
Social Factors	8	Because of the deepfake, which is a different technology, people will be more socially active through the game as they will play by seeing themselves in the game characters. Since the game characters will have real human faces, there will be no anonymous or fake characters. Thus, people will meet more quickly through the game.
Economic Factors	6	An important part of game development is character creation and customization. DeepGame can help reduce costs by offering cheaper and faster ways of developing

		character related parts of a game.

7.4 Teamwork Details

7.4.1 Contributing and Functioning Effectively on the Team

We deemed it appropriate to divide this section into group members to fill individually.

Betül Reyhan Uyanık

- Contributions to the reports throughout the project.
- Contributions to the screen shots, and diagrams in reports
- Researching the Game Engines available and deciding on the appropriate one for our project.
- Researching deepfake algorithms
- Sole developer of design of the Game
- Sole developer of the animations
- Sole developer of the Game part of the project
- Sole developer of the image uploading part of the game

Mert Alp Taytak

- Idea and design of the project,
- Research into various machine learning methods,
- Research into infrastructure,
- Sole developer of the cloud side of the project,
- Majority contributions to all reports throughout the project.

Ömer Faruk Geredeli

- Contributions to the reports
- Contributions to the charts and diagrams in previous reports
- Research into deepfake algorithms and contributions to the design of project
- Contributions to the designing the game

7.4.2 Helping Creating a Collaborative and Inclusive Environment

Since we were a small group, we all knew from the beginning of the project that we had to move forward with more help. Therefore, each group member had more responsibility. Since what we need to do and our duties could not be separated sharply, cooperation was at a high level. Each group member progressed with high motivation. We worked on the project in parallel with the joint workspaces we established. We implemented the project in constant communication with each other in online workspaces such as Google Colab and Google Drive.

7.4.3 Taking Lead Role and Sharing Leadership on the Team

We deemed it appropriate to divide this section into group members to fill individually.

Betül Reyhan Uyanık

I believe a leader is the most important part of the project team. With a motivating, sympathetic and a smart leader any team would be able to finish their product on time. In my other projects, during my internship and in my part time job as a software engineer I have been praised by my peers as an active, motivated and hardworking team member. Thus, since the beginning of the project I tried to be an active, motivated team member as possible. I have been able to fully complete the work that our supervisor has assigned to me.

Mert Alp Taytak

Since the very beginning of the project I tried to take an active role. I researched methods of achieving our goals. I thought and tried various methods of architecture and implementation of the software. I tried my best to lead, guide and direct my partners in the project. I shared tutorials, research papers and videos relevant to the project to prepare them.

7.4.4 Meeting Objectives

Due to the pandemic that has occurred since the beginning of our project, we had to hold our meetings online. We held weekly meetings over the Zoom application. In these meetings, we first brainstormed on how the project should be mentally. Since the owner of the project idea was Mert, one of our group members, we contributed as other group members on his ideas. We talked about the tasks shared to everyone at each meeting and how far we have come afterwards. During the summer period, we did researches for the coding parts of the project we will do, and continued our weekly meetings. Simultaneously, we tried to produce concrete results. As we are working in a new field, we faced many difficulties and we discussed how to debug these errors in our project meetings. In general, we paid attention to result-oriented action in our project.

7.5 New Knowledge Acquired and Applied

We deemed it appropriate to divide this section into group members to fill individually.

Betül Reyhan Uyanık

I have learned deepfake algorithms, read articles and learned the technology. I had no previous experience of developing a 3D game engine nor deepfake technologies/concepts. So the concepts were new to me. In the game part of the project I have learned how to create 2d and 3d games in unity from scratch. I have learned techniques, researched example repositories and watched online tutorials. After careful consideration I have decided to move forward with Unreal Engine for our game. I previously had small, dummy project experiences with unreal but the scale and complexity of our game was at another level. I developed the game part of our project including the animations, characters, room decoration, character creation page, photo upload page, and replacing the character's face with raw photos.

The only thing left here is just replacing this logic with the existing deepfake library that we have been working on.

Creating a game engine from scratch was very challenging but also educating. In addition to learning a new technology, I have developed better engineering and debugging skills. This project helped me decide on my career path. Although my specific focus was developing the game engine, I learned valuable information about deepfake algorithms, image processing, techniques such as texture wrapping. On the other hand since this project was done during the Covid-19 pandemic, I have learned that soft skills, team members in a project group, distribution of the work, communication, and understanding is equally important as engineering skills.

Mert Alp Taytak

The biggest effect of this project on my personal skills and knowledge was being introduced to another aspect of the Python ecosystem in machine learning applications, libraries and tools. I had previous experience with machine learning and image processing in a more theoretical level, only utilizing MATLAB if there ever was a practical part. That experience diminished my interest in the field. But, reexamining the field under the Python ecosystem was a nice experience. I also learned about Google Colab in this project. Which makes machine learning and image processing projects very accessible to the average person.

Other than tooling and software, I also learned new techniques. Mainly style transfer and image animation with deepfakes. My needs did not require me to learn the theory behind style transfer, but I studied image animation to get a better understanding of its capabilities. I learned that the idea is to train a model that detects the keypoints in an image and apply affine transformations to those polygons defined by the keypoints to match them to the arrangement of keypoints of frames in a driver video.

I applied all of this cumulative knowledge to come up with the idea of a technique that would achieve DeepGame, and developed the cloud side of the project.

Ömer Faruk Geredeli

When I started the project I had some ideas about what the deepfake is and how it looks. The fact that the subject was interesting and untested was also one of the reasons that affected my participation in the project. I did detailed research on deepfake algorithms and realized that this new technology area has no limits. We developed the algorithms we use with machine learning with artificial intelligence. Since we will build our project on a game, I researched the popular game engines Unity and Unreal Engine, and learned how to use them. I learned to create games in 2 dimensional and 3 dimensional universe. In addition to these, I created joint workspaces in Google Colab and Unity Colab to see Deepfake algorithms that require high performance without using the hardware features of the computer.

8 Conclusion and Future Work

From a growth perspective, this has been a very enlightening project. We learned about new tools and methods. We learned about what goes into collaborative work. We learned about do's and don'ts of project planning, software engineering and team work.

From an achievements perspective, this project has been a partial success. We were not able to meet our goals of creating a complete game engine plugin that can be integrated into any game to offer the DeepGame experience. Due to our circumstances, we had to settle for a proof of concept instead.

For future work, there are many possibilities. The most obvious one is to take the project to its initial goal and complete the DeepGame as an engine plugin. The other one is to take the cloud side off Google Colab and turn it into something deployable to any instance with machine learning infrastructure.

For the future, there is also the idea of improving the machine learning models as discussed earlier in the maintenance section. Machine learning is a fast moving field and new research is coming out every year that overperforms earlier research. In our project, we used very recent research. But, the future will most likely bring better projects. Therefore, machine learning parts of the DeepGame can be replaced by better models and research as they come out.

9 Glossary

Deepfake: Common name for the process of constructing videos with image of a person transferred onto a person in the video, without the transferred person being a part of the original video.

Image Animation: Image animation is the process of animating a single image using motion information from a driver video. Motion of the keypoints of the video frames are applied to the keypoints of the input image. Resulting in a video from the input image, changing like the driver video.

Mask: A boolean matrix that decides whether a particular pixel of an image should be selected or not.

Mask Construction: Process of applying set operations to masks to create a new mask from mask pieces.

Mask Generation: Process of generating masks on a given image.

Style Transfer: Style transfer is the process of taking two images, one for style one for content. Then, extracting style and texture from the style image; extracting object and shape information from the content image, extracted information is merged to produce a new image. This new image looks like the content image painted with the art style of the style image.

Content Image: Image that receives the style transfer. Also used for the image of the target person, because that image is used as the content image.

Style Image: Image that supplies the style for the style transfer.

APPENDIX A - User Manual

Cloud Side Manual

The prerequisites to running the cloud side is to have a computer with internet access and a Google account.

The first step is to ensure familiarity with Google Colab usage. Google provides an introductory material for this purpose. Which can be found in the link below:

https://colab.research.google.com/notebooks/intro.ipynb

The second step is to open the DeepGame notebook in Colab. Go to the GitHub repository of DeepGame, located in the link below:

https://github.com/mertalpt/Deepgame

The third step is to locate the notebook file with the '.ipynb' extension located in the top level files of the repository. The file name should be 'DeepGame.ipynb' . Direct link is given below:

https://github.com/mertalpt/Deepgame/blob/master/DeepGame.ipynb

The fourth step is to open this notebook in Colab. There is a button at the beginning of the notebook file that will automatically do that for you. Left click the button or right click and 'Open in a New Tab'. It should open the notebook in Colab.

Afterwards, we are ready to run the notebook. Connect to the Colab instance. Because the notebook requires GPU usage, Colab may block you due to usage limits. This happens if you have used Colab with GPU a lot recently. The solution is to use another Google account or wait a couple days. If you have connected successfully, follow the steps below. Note that there are instructive and explanatory texts in the notebook to help you along the way.

1. Run the cell 'Install and Setup Dependencies'. It will ask you to authenticate your Google account by providing a link and asking for an input. Go to the link and select a Google account. It will ask for access permissions, click 'Allow'. It will produce a code, click the button to the right to copy the code. Return to Colab, click the input field and use 'Ctrl+V' or paste shortcut of your system to paste the code. Hit the 'Enter' key. If authentication is successful, it will download files from a publicly shared Drive folder to the Colab instance. The notebook does not access your files.
2. Run the cell 'Initialize Workspace'. It will download some files and initialize certain systems. This process may take a few minutes.
3. Follow the instructions under the 'Loading Images' section to load a content image and a style image. These images can be selected from the example data that comes with DeepGame or from files manually uploaded to Colab or from the

internet by providing a direct URL to the image file. Any custom image comes with the option to crop the image around the face. You can play with the parameters to produce a satisfying result. Cells need to be rerun after each change to see the result. Face detection may not always work and throw an error. Try with different images or parameters if that happens. Also, the last loaded image is saved for each content image and style image.

4. Follow the instructions under the ‘Loading the Video’ section to load an example video or a custom video from manually uploaded files. Instruction from the ‘Loading Images’ section also applies here.
5. Run the cells under the ‘Display Your Inputs’ section to verify that your selections are what you desired. There should be two images and a video. If there is any error or what is displayed is not what you desired, go back to steps 3 and 4.
6. Run the cell under the ‘Style Transfer’ section to apply style transfer to your images. Style transfer may produce undesirable results due to color palette changes. Glitches in the image background will later be masked out. If there is a glitch on the face itself or the result is undesired in general, go back to step 3 and try again with different images.
7. Run the first cell under the ‘Construct a Mask for the Image’ section. It will produce mask pieces that cover segments of the content image. You can play with the parameters to alter what pieces are generated. Threshold parameters affect how bright a pixel constitutes as the start of a different segment. Sigma affects edge sharpness. The size filter filters out mask pieces that fall below a certain pixel count. It is recommended to set it to at least 100, because hundreds of small mask pieces can be generated.
8. This step requires programming intervention. Look at the mask pieces generated at the previous step and using instructions written above the cell, construct a mask.
9. Run the cell named ‘Apply the Mask’ to apply the mask to the style image and display the result. The mask color is pure green by default, but it can be changed by changing the script underlying the cell to input the desired color to the relevant function.
10. Next, run the cells under the ‘Image Animation’ section consecutively. The first two cells create a deep fake video, the last two cells filter out the mask to make it transparent. Mask filter is not perfect, but can be improved by changing the script to alter the color range.
11. Finally, run the cell named ‘Save the Result’. Here, there are two options. The first is to save the video as a single ‘mp4’ file to the given path. The second is to save the video as a series of images. This option will take a path to a directory and fill it with images for each frame.

At the end you have a video inside the Colab instance. You can download the result manually using Colab’s own interface. If you choose to save the result as a series of images, there is an example cell that can be altered to zip the directory. Then, the zip file can be manually downloaded.

Game Side Manual

Unreal supports Mac, Windows, IOS, Android, even playstation. When packaged on a Mac, the game can be executed on a Mac with the executable file. When packaged in Windows the game can be played in Windows machines with the exe file. Cross compiling among Windows and Mac does not work properly. Cross compiling among Windows and Linux, or Windows and playstation works without a problem. Opening the executable is enough to play the game.

Game is implemented mainly for the purpose of showing the character creation using the deepfake algorithms.

10 References

[1] “Colaboratory FAQ,” *Colaboratory - Google*. [Online]. Available: https://research.google.com/colaboratory/faq.html. [Accessed: 15-Dec-2020].

[2] G. Ghiasi, H. Lee, M. Kudlur, V. Dumoulin, and J. Shlens, “Exploring the structure of a real-time, arbitrary neural artistic stylization network,” *arXiv.org*, 24-Aug-2017. [Online]. Available: https://arxiv.org/abs/1705.06830. [Accessed: 15-Dec-2020].

[3] L. A. Gatys, A. S. Ecker, and M. Bethge, “Image Style Transfer Using Convolutional Neural Networks,” *Page Redirection*, 2016. [Online]. Available: https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Gatys_Image_Style_Transfer_CVPR_2016_paper.html. [Accessed: 15-Dec-2020].

[4] “How do I remove the background from this kind of image?,” *Stack Overflow*, 28-Mar-2015. [Online]. Available: https://stackoverflow.com/questions/29313667/how-do-i-remove-the-background-from-this-kind-of-image. [Accessed: 15-Dec-2020].

[5] F. L. Bourdais, “Removing the Background from an Image using scikit-image,” *Removing the Background from an Image using scikit-image | Frolian's blog*, 01-Sep-2016. [Online]. Available: https://flothesof.github.io/removing-background-scikit-image.html. [Accessed: 15-Dec-2020].