Bilkent University

Department of Computer Engineering

Senior Design Project

High-Level Design Report

Deepgame

Students

Betül Reyhan Uyanık

Mert Alp Taytak

Ömer Faruk Geredeli

Supervisor

Dr. Uğur Güdükbay

Jury Members

Prof. Dr. Özgür Ulusoy

Asst. Prof. Dr. Shervin Rahimzadeh Arashloo

Innovation Expert

Cem Çimenbiçer

May 22, 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

1.1 Purpose of the System

Modern times offer people various kinds of video games through a multitude of platforms. In some genres of games the user plays as a character. Since the appearance of the player character is important be it for immersion or role playing purposes, games have been offering character customization options. Some of these customization options take the form of accessories or clothing that changes the player character superficially. Another option is customization of character models. Some games, with the help of improving technology, offer extensive and elaborate character creation menus. Which makes it possible to create a visually diverse range of characters. However, customization is not limited to visuals. Character voices are another part of the customization process. Due to its nature, human speech is not fit for modeling and customization in the way the human body is. Many games we have seen in the market use a limited set of recorded, generic voice lines for characters. Some do not voice act player character's lines at all. Perhaps, due to how expensive it gets to offer various voices for extensive dialogue options.

We would like to develop a software tool that can be integrated into games themselves to allow the user to put themselves, or maybe others, into the game visually and auditorily. While there exist games with similar features, they require expensive hardware. Using techniques developed in recent years, Deepgame will work with photographs and audio recordings. Easily acquirable through hardware common to almost every computer user.

1.2 Design Goals

Deepgame is a tool designed to customize gaming experience. Therefore, it has design goals that are defined around giving players a better experience along with other universal software concerns. Following are our design goals in no particular order and a brief explanation for each one.

Note that Deepgame is a tool to be integrated into video games. So, our goals will involve both Deepgame and the games that will use Deepgame in general.

1.2.1 Immersion

Immersion is defined by the quality of imitation from reality to in game and how well the result matches the game's art style. There is no proper metric to measure these properties. Therefore, we will be using subjective judgement instead. One objective measure is to be consistent within the model and have no glitches occur in the models.

1.2.2 Performance

Most important part of a game experience is to have sufficient performance to be able to enjoy the game. There are two parts to performance. First is model creation and second is running the game itself. For the first, we would like to keep model creation below ten minutes. However, model creation is a one time event for each person to model; so, it is not vital as long as the creation times are not absurdly long. For the second, we will aim for frame rates of over 60 frames per second and input lag of less than 100 milliseconds to be achievable on a mid-range computer with a dedicated GPU.

1.2.3 Usability

Usability comes in two categories: usability for players and usability for developers.

Player usability is about user experience in navigating the interface for creation and use of models. Our target is to make it as easy and familiar as copying a file between directories in a computer.

Developer usability is about how easy it is for game developers to integrate Deepgame into games. We will define model formats and an API so that developers can integrate Deepgame into their games with minimal work and alteration required on their end.

1.2.4 Cost

Cost is about the level of hardware required to run Deepgame integrated games smoothly. It is something we want to minimize. However, we target mid-range computers with dedicated GPUs for the computer side and common smartphones for the photographing and recording.

1.2.5 Security

Use of Deepgame requires handling biometric data of people. Therefore, it is important to ensure data safety. We will use RSA encryption to keep sensitive data secure.

1.2.6 Privacy

As above, Deepgame involves sensitive biometric data. Our goal is to ensure personal data does not get distributed without consent of the person. RSA encryption allows us to manage distribution of data and additional protocols can be used to make sure data can be safely shared and deleted as necessary in multiplayer games.

1.3 Definition, Acronyms and Abbreviations

Deepgame	Integrable software tool that helps model real persons in games.
Model	Digital data representing the likeness of appearance or likeness of voice of a person.

Feature	Appearance or voice data that belongs to a specific person.
Feature Transfer	Process of creating a game model from a feature model of a person.
Feature Extraction	Process of extracting features from photographs and recording of a person.
Target	Person to be modeled.
Unity3D	A game engine [2].
Plugin	Code that is written to be integrable to Unity3D projects like a library or a framework in programming languages.

1.4 Overview

Character customization in video games is an important part of the player experience. Deepgame is a software tool that uses photographs and voice recordings of a person to imitate their likeness in games.

We will develop Deepgame to be a two-part software. First part will be the standalone feature transfer engine that will take photographs and recordings to create game-agnostic models of people. Second part will be a Unity3D plugin that takes those generalized models from the standalone software and uses them to create in-game models in video games that use the plugin with some tweaks by the game developers. Separation of the standalone software and the plugin will allow us to get the most performance in each part and highest portability of models between games.

In order to demonstrate the capabilities of Deepgame, we will also develop a video game in Unity3D with the Deepgame plugin.

2 Current Software Architecture

Aside from relatively archaic methods of imitation through photographs as avatars or textures to map onto in-game models, only instance of imitation of real persons is found in Kinect Sports Rivals [1].

Kinect Sports Rivals is a multiplayer sports game on XBOX One that utilizes the console’s Kinect motion-sensing camera to take depth images of the target person and create a three dimensional avatar in the game.

On the superficial level requirement of a Kinect camera is a relatively expensive prerequisite. On a more technical level, feature transfer in Kinect Sports Rivals does not create unique avatars but rather composes numerous subfeatures such as face shape and hair color that best matches the target’s subfeatures to come up with a functionally pregenerated avatar.

3 Proposed System Architecture

3.1 Overview

As a concept Deepgame is simple. It takes data about real life people and imitates their likeness in video games. As a result Deepgame's system architecture is also simple. The real complexity in this project comes from developing the theory required to achieve good imitation.

With Deepgame, there are two main systems in play. First is the Feature Extractor and second is Feature Transferer. The Feature Extractor system is built to be a standalone software whose sole job is processing biometric data into game-agnostic feature models. The Feature Transferer system is the product of Unity3D plugin part of Deepgame that is built into the games and configured by the game developers to take feature models from the Feature Extractor and create in-game models. These systems are then further broken down into subsystems. Note that there is no server involved and everything takes place in the player's computer; and Feature Transferer is built into the games.

3.2 Subsystem Decomposition

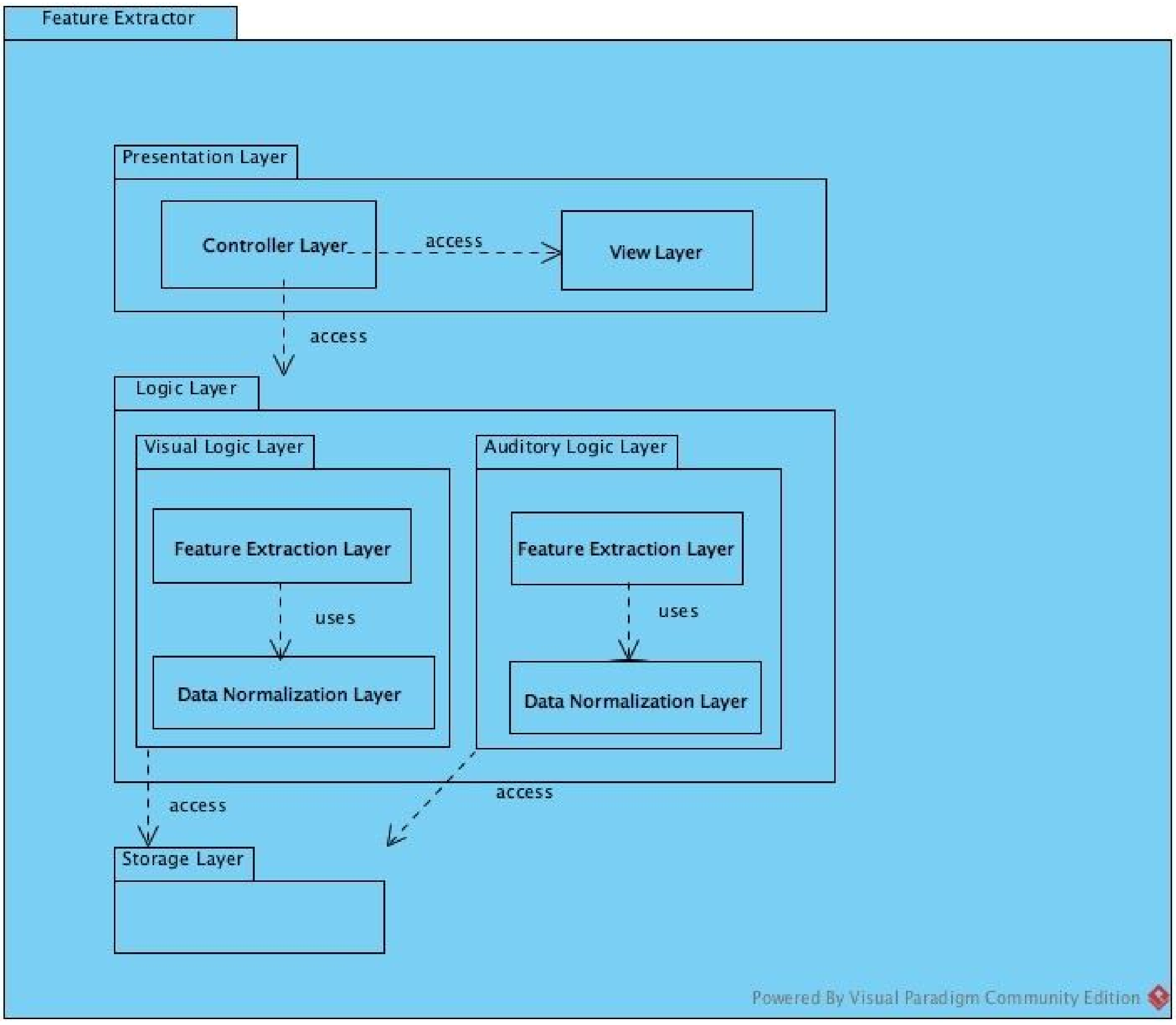

Figure 1: *Feature Extractor* Subsystem Decomposition

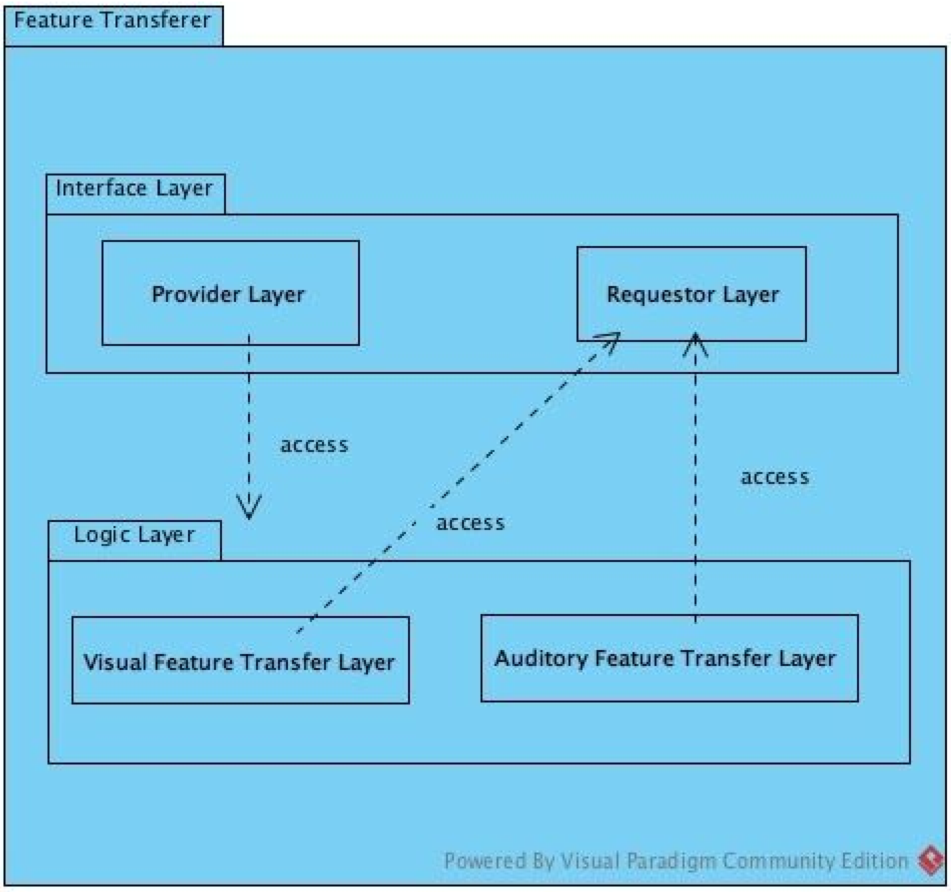
Feature Transferer
Interface Layer
Provider Layer
Requestor Layer
access
access
access
Logic Layer
Visual Feature Transfer Layer
Auditory Feature Transfer Layer
Powered By Visual Paradigm Community Edition

Figure 2: *Feature Transferer* Subsystem Decomposition

Deepgame runs on PC with no access to a server. Deepgame's systems are split into two parts: Feature Extractor and Feature Transferer. These systems are then broken down into further subsystems. Following is a brief explanation of each subsystem component.

3.2.1 Feature Extractor

Feature Extractor is a standalone software designed to go from raw biometric data to universally applicable feature models.

3.2.1.1 Presentation Layer

Presentation Layer is responsible for providing the user with an interface and managing the internal logic as requested by the user. It has two further subsystems:

- *Controller Layer:* Responsible for managing the internal logic.
- *View Layer:* Responsible for providing the user interface.

3.2.1.2 Logic Layer

Logic Layer is responsible for turning raw biometric data provided by the user into universally applicable feature models. It has two further architecturally identical, functionally different subsystems: Visual Logic Layer and Auditory Logic Layer. These layers do the processing of data. Such a logic layer is then further broken down into:

- *Feature Extraction Layer:* The layer where actual processing happens. Uses machine learning techniques to do the processing.
- *Data Normalization Layer:* Processes raw data into manageable chunks and does denoising and formatting in the process.

3.2.1.3 Storage Layer

Manages access to computer disk for file retrieval and file storage.

3.2.2 Feature Transferer

Feature Transferer is built into the games developed on Unity3D that uses the Deepgame plugin for Unity3D. It is designed to fit into the game developers' vision for the game it is used in. Therefore, all user interface is left to the game developers and Feature Transferer only provides an interface layer to manage access.

3.2.2.1 Interface Layer

Manages access between the game and the inner logic. It has two further subsystems:

- *Provider Layer:* Component of the interface that is used by the game to request the Feature Transferer to provide a service.
- *Requester Layer:* Component of the interface that is used by the Feature Transferer logic to request the game to do tasks it does not have access to such as storage access or game data to transfer features onto.

3.2.2.2 Logic Layer

Contains feature transfer layers that take an universal feature model produced by a Feature Extractor and transfer it onto game models provided by the game. For example, given an auditory feature model and a sound file containing a voice line, the Auditory Feature Transferer will use the model to produce a sound file with the voice line made to sound like it was spoken by the person that made the feature model.

3.3 Hardware/Software Mapping

Deepgame is a two-part software tool that is designed to work with Unity3D engine. Although Unity3D targets multiple platforms and standalone part of Deepgame can be written to run on any platform, Deepgame requires computational power that makes it infeasible on mobile. Therefore, we will target PC for our project. Although not part of the software itself, Deepgame indirectly requires a camera and a microphone to provide data that can be worked on with machine learning.

Software components of the Feature Extractor part is as follows:

- Python and TensorFlow will be used to develop machine learning related parts.
- Java will be used to develop interface and controller parts.
- Python part will have binding that will be used by the Java part to join two parts together.

Software components of the Feature Transferer part is as follows:

- C# will be used to develop interface components of the plugin in Unity3D.
- Python and TensorFlow will be used to develop machine learning related parts.
- Two parts will be joined together through language bindings.

Besides the CPU, only two relevant hardware for the Deepgame itself are storage and GPU. Both are directly accessible through language standard libraries. Following diagrams gives a representation of the Hardware/Software Mapping.

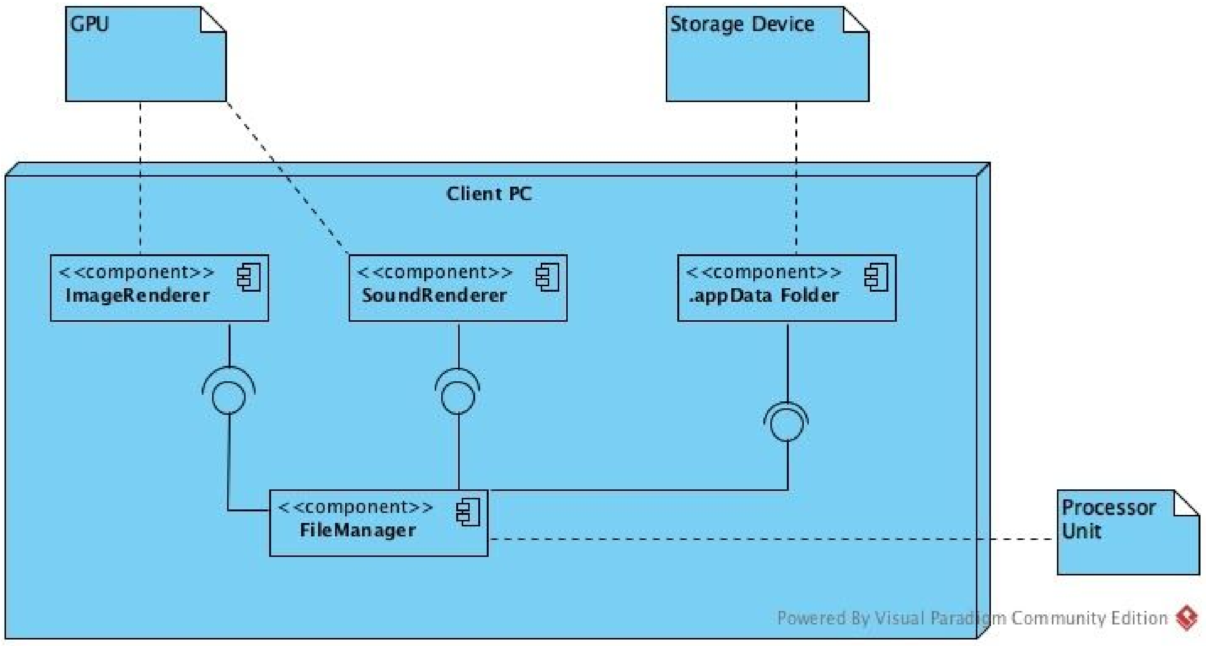
GPU
Storage Device
Client PC
<<component>> ImageRenderer
<<component>> SoundRenderer
<<component>> .appData Folder
<<component>> FileManager
Processor Unit
Powered By Visual Paradigm Community Edition

Figure 3: Deployment Diagram of Feature Extractor/Transferer

3.4 Persistent Data Management

Persistent data is very important for the functioning of Deepgame. Since Deepgame deals with imitating real people it requires data about those people's features. However, for each game those feature data is processed into game specific models. Considering the use case, a database is not required and file systems provided by operating systems are enough to store and process the data as files.

3.5 Access Control and Security

As previously mentioned, Deepgame requires no database or network access. Therefore, access control is not required. However, data safety is important since we are dealing with biometric data. For this purpose we will implement an optional encryption feature that uses RSA encryption. In case a game developer wants to use Deepgame with a multiplayer game, we will provide basic tools to enable the safe use of RSA encryption with multiple players. However, we will leave it to the game developers to ensure total safety. Therefore, it will be developers' responsibility to ensure data safety in multiplayer.

3.6 Global Software Control

Parts of Deepgame provide asynchronous, one time services to users without network usage. Therefore, we cannot talk about *global* software control. Everything happens in the local.

Locally, Deepgame works on an event-driven model. When a user wants to extract or transfer features from a person, they use the corresponding software to perform a one time task. Here, one time task means having a process with a definitive start and an end. It is possible to perform multiple feature work, but each requested work is a new event. Hence the event driven model.

3.7 Boundary Conditions

Using an event-driven model of performing one time tasks, there are three boundary cases for Deepgame: Initialization, termination and failure. Following is the discussion of those cases. Since they are functionally similar, the term *application* will be used to cover both Feature Extractor and Feature Transferer.

3.7.1 Initialization

Initialization is the process of starting up the application and performing a work order. Naturally, the first prerequisite is to have the application installed. The second is to have a storage device and a GPU. Finally, data must be provided to the application.

3.7.2 Termination

There are two cases for termination. First is for the work order to be finished. Second is for the user to cancel the process. If the former is the case, then the final product will be shown to the user to be evaluated. If found sufficient, then it will be saved onto the location provided by the user. If not, then additional data and configuration will be requested and the work will be repeated. If the latter is the case, then garbage collection will be performed and the application will quit.

3.7.3 Failure

There are two cases for failure. First is for an error to occur. Second is to have a deadlock in the machine learning process. We will have no uncaught exceptions in our code. Hence, if the former is the case the user will be informed and the system will be restored to a state where the cause of error can be fixed or the work can be cancelled. If the latter is true, it means that improvement into given parameters is not achievable. To handle this type of error, the user will be given tools to monitor the process so that they can cancel the work if such a failure is happening.

4 Subsystem Services

This section will take brief explanations from *3.2 Subsystem Decomposition* and build up on it with additional information such as main classes in each layer and their responsibilities.

4.1 Feature Extractor

Feature Extractor is a standalone software designed to go from raw biometric data to universally applicable feature models.

4.1.1 Presentation Layer

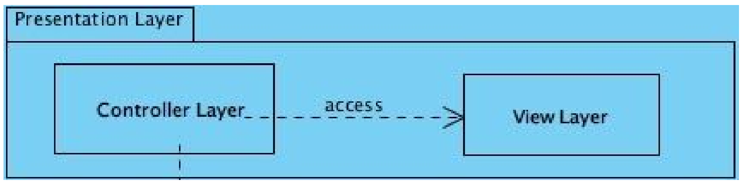

Figure 4: Presentation Layer Diagram

Presentation Layer is responsible for providing the user with an interface and managing the internal logic as requested by the user. It has two further subsystems:

- *Controller Layer:* Responsible for managing the internal logic.
- *View Layer:* Responsible for providing the user interface.

4.1.2 Logic Layer

Logic Layer is responsible for turning raw biometric data provided by the user into universally applicable feature models. It has two further architecturally identical, functionally different subsystems: Visual Logic Layer and Auditory Logic Layer. These layers do the processing of data. We will generalize those logic layers and discuss their subsystems.

4.1.2.1 Feature Extraction Layer

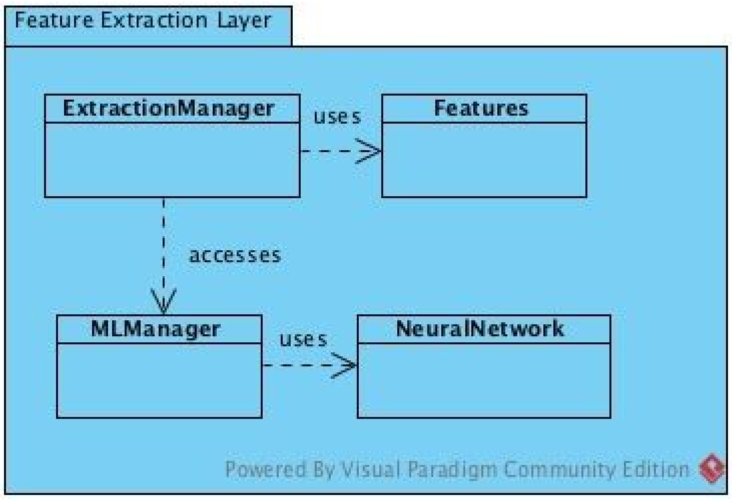

Figure 5: Feature Extraction Layer Diagram

Feature Extraction Layer is responsible for taking normalized data and processing it into a model.

- *Features* is a class that holds key characteristics such as face ratio, eye distance, eyebrow shape.
- *NeuralNetwork* is a class that learns from the data to provide a model.
- *MLManager* is a class that governs the learning process.
- *ExtractionManager* is a class that governs the extraction process and manages access to and from the Presentation Layer and the Data Normalization Layer.

4.1.2.2 Data Normalization Layer

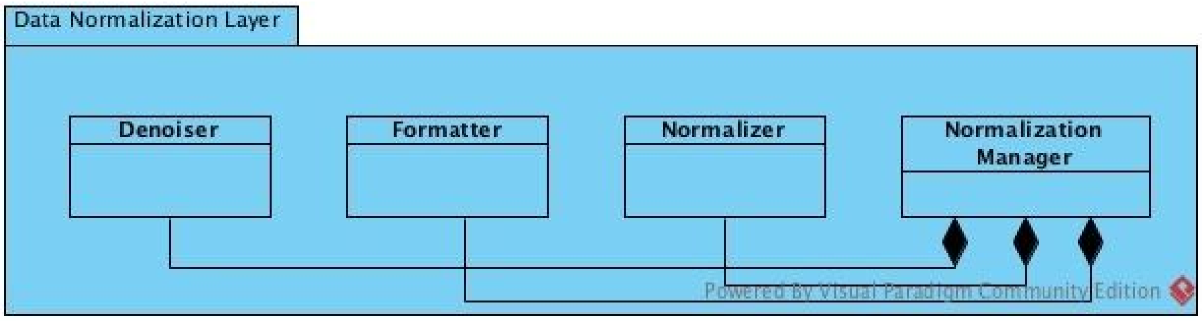

Figure 6: Data Normalization Layer Diagram

Data Normalization Layer is responsible for taking raw data and normalizing it into a format recognizable and workable by the Feature Extraction Layer.

- *Denoiser* processes data to remove noise.
- *Formatter* processes data to format it to fit into parameters given such as resizing the image.
- *Normalizer* processes data into workable chunks such as dividing the recording of a sentence into sound files of words.
- *NormalizationManager* uses the classes given above to normalize the data it is given.

4.1.3 Storage Layer

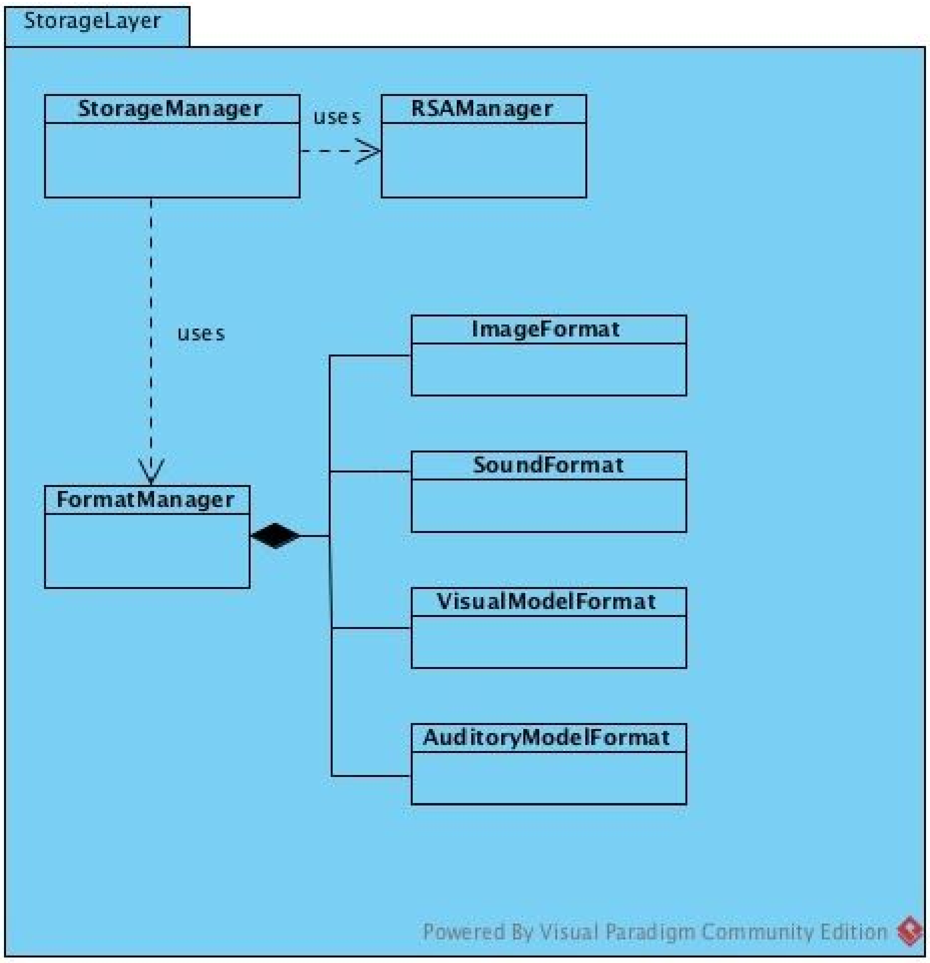

Figure 7: Storage Layer Diagram

Manages access to computer disk for file retrieval and file storage, handles encryption and decryption. Also contains file formats used to store raw and processed data.

4.2 Feature Transferer

Feature Transferer is built into the games developed on Unity3D that uses the Deepgame plugin for Unity3D. It is designed to fit into the game developers' vision for the game it is used in. Therefore, all user interface is left to the game developers and Feature Transferer only provides an interface layer to manage access.

4.2.1 Interface Layer

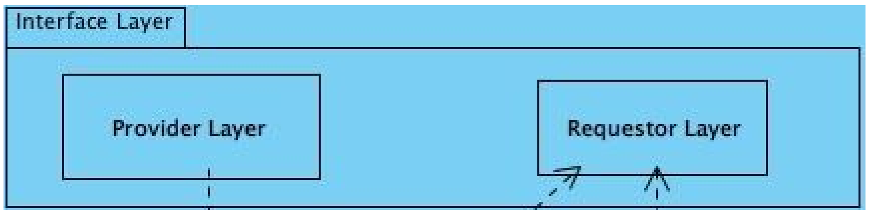

Figure 8: Interface Layer Diagram

Manages access between the game and the inner logic. It has two further subsystems:

- *Provider Layer:* Component of the interface that is used by the game to request the Feature Transferer to provide a service.
- *Requester Layer:* Component of the interface that is used by the Feature Transferer logic to request the game to do tasks it does not have access to such as storage access or game data to transfer features onto.

4.2.2 Logic Layer

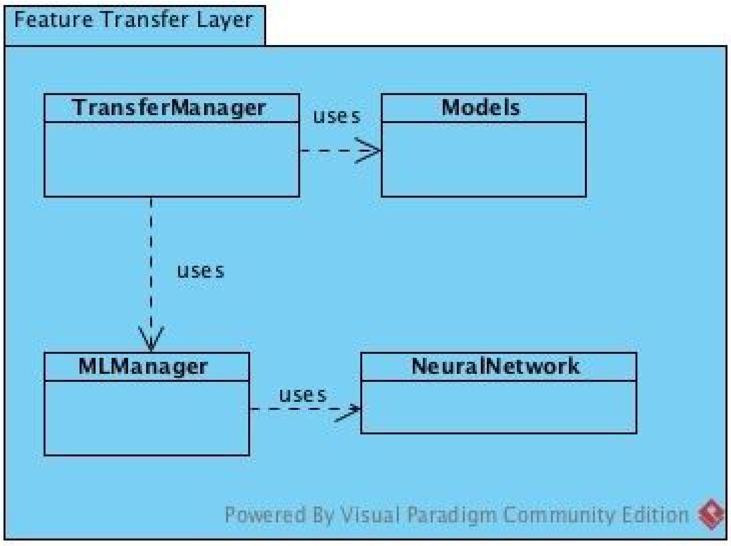

Figure 9: Generalized Feature Transfer Layer

Contains feature transfer layers that take an universal feature model produced by a Feature Extractor and transfer it onto game models provided by the game. For example, given an auditory feature model and a sound file containing a voice line, the Auditory Feature Transferer will use the model to produce a sound file with the voice line made to sound like it was spoken by the person that made the feature model.

- *Models* is a class that holds the format of the model product by the Feature Extractor.
- *NeuralNetwork* is a class that learns from the feature model to provide a game model.
- *MLManager* is a class that governs the learning process.
- *TransferManager* is a class that governs the feature transfer process and manages access to and from the interface.

5 New Knowledge Acquired and Learning Strategies Used

As we have mentioned in the Project Analysis Report, we are planning to learn about new fields, algorithms and technologies in order to have a working project. For the visual and auditory feature transfer technology part of the Deepgame we studied deepfake algorithms for both visuals and sounds. We researched and learned from academic articles and other written sources and understood the basic logic. We have researched what is neural network and its sub branches, and how it is used in applications. For the second part, a video game integrated with Deepgame, we researched the features required for a good game and learned the logic of creating a game. We researched the game engines and decided on the game engine we will use. We learned how to use the game engine we decided on. We are planning to go over different algorithms and various applications of these so that we can come up with our own. Main challenge of the project is coming up with an algorithm which is designed entirely by us, and is capable of matching the game's art style. Our main source for these algorithms were GitHub and various articles. Since the concept of deepfake is relatively new, applications and algorithms of the technology are scarce. In the summer, we are planning to study the implementation of our source code and apply the techniques that we learned on our project.

6 References

[1] X. W. Staff, “Creating a Champion in Kinect Sports Rivals,” Xbox Wire, 20-Oct-2015. [Online]. Available: https://news.xbox.com/en-us/2014/04/01/games-ksr-developer-qa/. [Accessed: 23-Feb-2020].

[2] “Unity Asset Store - The Best Assets for Game Making,” Unity Asset Store - The Best Assets for Game Making. [Online]. Available: https://assetstore.unity.com/. [Accessed: 23-Mar-2020].