Bilkent University

Department of Computer Engineering

Senior Design Project

Low-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

October 5, 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.

Another aspect to characterization in gameplay is dialog, or rather the voice of a character. While story heavy video games include extensive dialog, other types of games may include one liners to be thrown out according to a script to enhance the immersion. Common point between all games with dialog is that they all offer pre-recorded, limited number of voice packages. Once again, evolving technology allows us to take dialog customization to another level.

The aforementioned evolving technology is the rise of deepfake algorithms. A new class of algorithms dubbed "deepfake" makes use of deep learning to transfer likeness of visuals and audio between persons within photos, audio recordings and videos [1].

Novelty, and the challenge, of our project is to apply deepfake algorithms to video games [2]. The application of deepfake algorithms from real environments to virtual environments is largely unexplored and there has not been a known video game application so far. Our final goal is to create the framework software that can be integrated into video games themselves that will enable the premise of our project.

Rest of this section discusses decisions concerning the overall goals of the project and guidelines we used in the documentation and development of the project. The following sections define the general structure of the program architecture and blocks that make up the architecture.

1.1 Object Design Trade-offs

1.1.1 Functionality vs Usability

The project will be software that is intended to be integrated into other software. As a result, our target user base is developers. This allows us to be more demanding of our users. Due to the complex nature of our project, proper use of the product requires insight into the theory behind the software. Because game developers are expected to be knowledgeable in their domain, we can focus on functionality more than we focus on usability. Therefore, we will prefer providing more functionality over making less functionality more usable.

1.1.2 Performance vs Privacy and Security

The project requires processing large amounts of biometric data. Fastest way of doing the processing would be sending the data to external servers and processing the data there.

However, this creates privacy concerns and security risk. Seeing how gaming computers would have powerful hardware, we can do the processing locally for a massive reduction in privacy and security risks at the cost of a slight hit to the performance.

1.1.3 Performance vs Immersion

Immersion is the main goal of this project. However, achieving the best possible immersion comes with a high cost in performance. We will focus on immersion over performance while still maintaining a comfortably playable performance. For this purpose we can develop our project with the target of better quality while providing options to reduce quality in favor of more performance. This will enable the game developers and video game players to fine tune the game to their taste.

1.2 Interface Documentation Guidelines

In this document, classes are documented with the following components:

1. Package Name: Name of the package the class belongs to.
2. Class Name: Name of the class.
3. Description: Summary of the purpose of the class.
4. Attributes: Public (“+”) and private (“-”) attributes of the class with their types.
5. Methods: Public (“+”), protected (“#”) and private (“-”) methods of the class with their signatures.

As evident from above, the guidelines follow the UML principles [4].

1.3 Engineering Standards

In this document there are two standards followed. The first is the use of UML notation in our diagrams. The second is the use of IEEE citation guidelines for our referencing and citations [3].

1.4 Definitions, 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.

2 Packages

Because feature extraction capabilities of our project can be separated into a standalone software to produce feature data that can be used across multiple games, we separate our packages into two groups.

The purpose of this separation is to provide modularity where it is possible. Moreover, the ability of separating feature extraction from game specific to standalone further reduces the workload of game developers.

Note that due to the feature extractor and transferer being two parts of a whole they share a common structure, sometimes to the point of seeming redundant.

2.1 Feature Extractor Packages

Following is the package diagram of feature extractor part of the project:

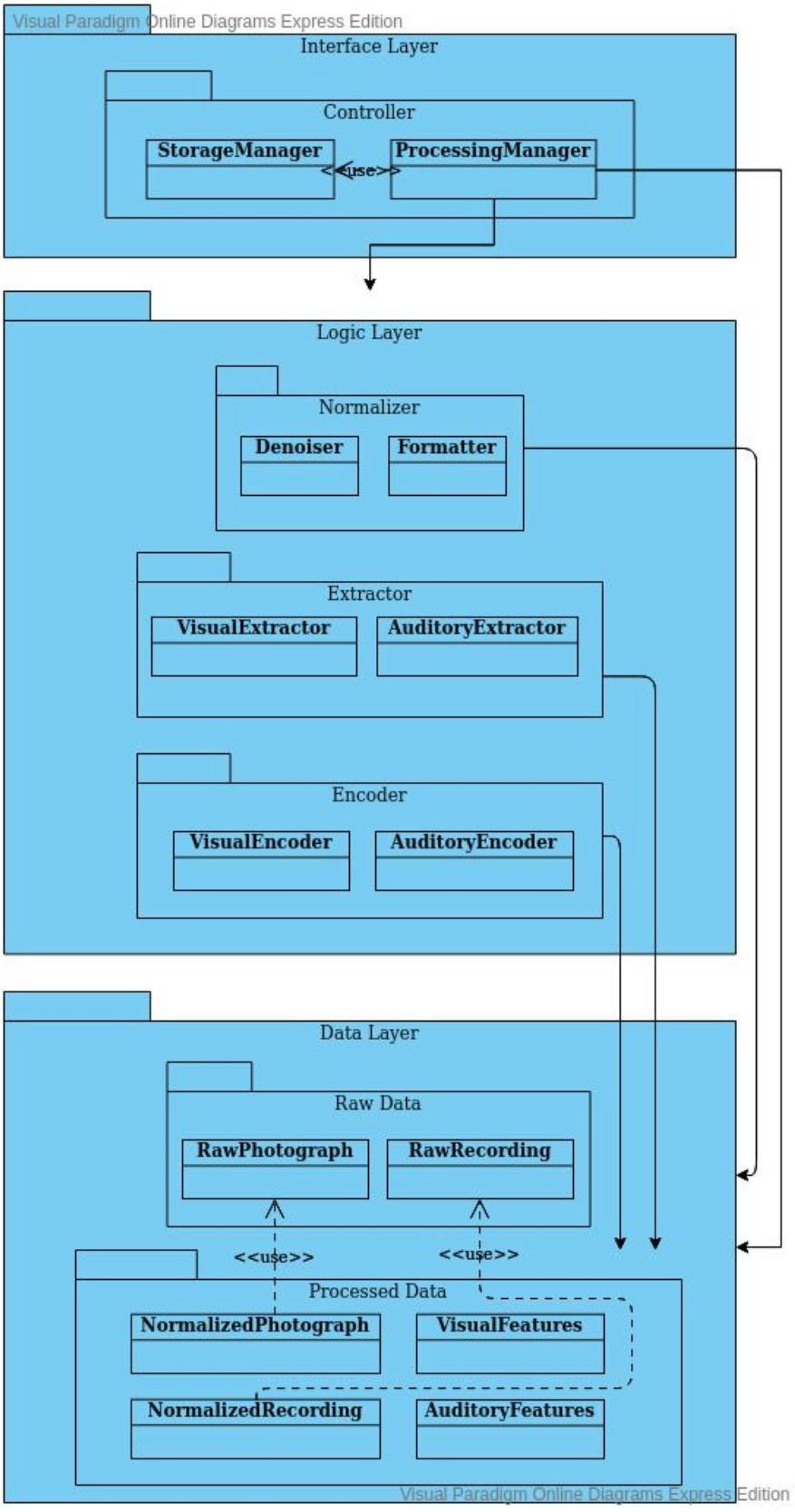

Figure 1. Extractor Packages Diagram

Note that due to limitations of our drawing software and our desire to lessen clutter in the diagram we used improper notation. In the diagram above, straight arrows are used to denote a class or classes in a package use the classes from the package or packages at the end of the arrow.

The architecture featured here follows the guidelines of “Presentation - Logic - Data” architecture. In absence of direct interaction with the software, we opted to call the “Presentation Layer”, “Interface Layer” instead.

2.1.1 Interface Layer Packages

The interface layer consists of packages handling requests from the greater software to the Deepgame API. Existence of the interface layer allows software developers to use Deepgame Feature Extractor as a blackbox.

2.1.1.1 Controller

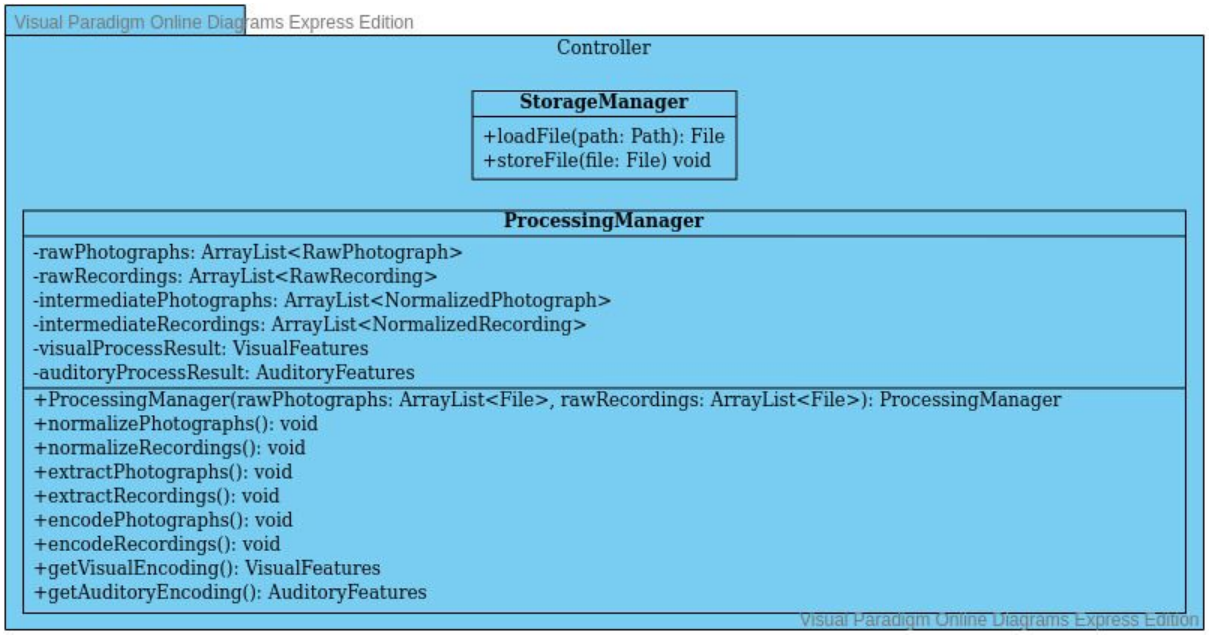

Figure 2. Extractor-Controller Package Diagram

The controller package contains classes to handle the requests of taking the raw data, processing the data and producing feature data as a result. This is done through instantiating new processors and moving the data through them.

2.1.2 Logic Layer Packages

The logic layer consists of packages that handle the stages of data processing.

2.1.2.1 Normalizer

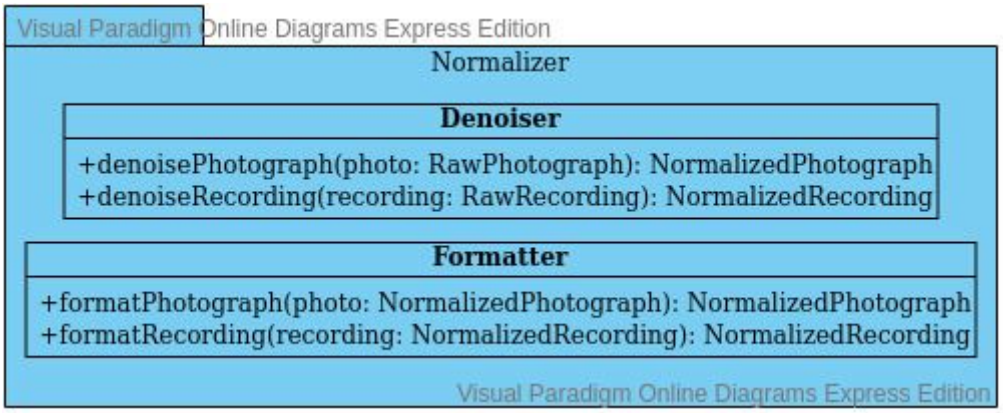

Figure 3. Extractor-Normalizer Package Diagram

The normalizer package contains classes that are responsible for taking the raw data, cleaning up the data and formatting the clean data into a format usable by the extractor.

2.1.2.2 Extractor

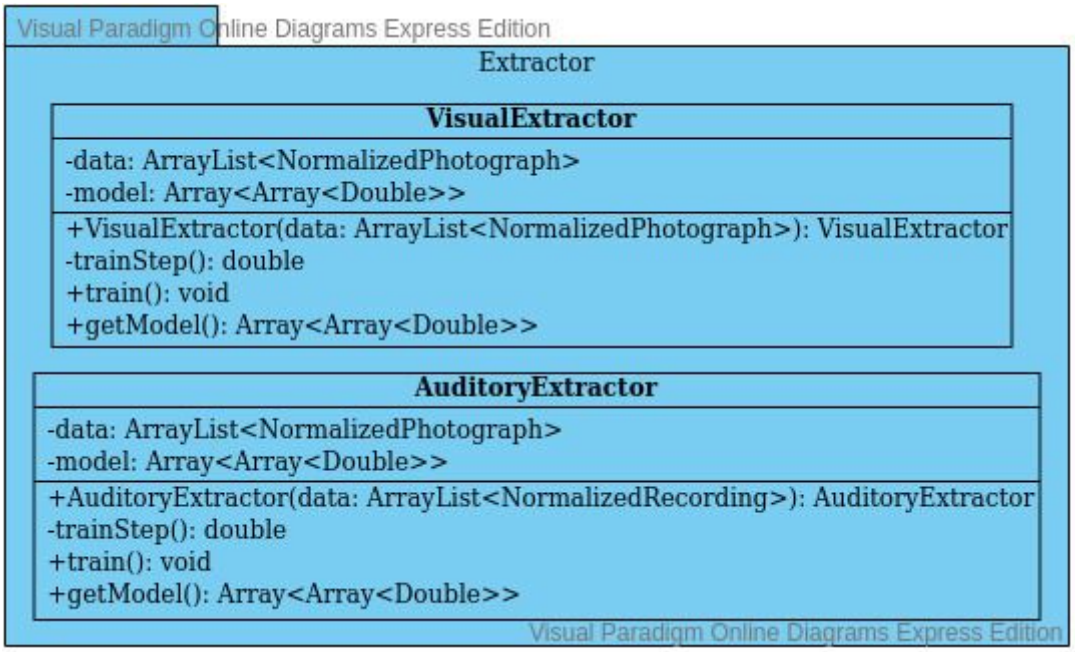

Figure 4. Extractor-Extractor Package Diagram

The extractor package contains classes that are responsible for taking the normalized data and turning it into features usable by the feature transferer. For this purpose deep learning is leveraged through the use of neural networks.

2.1.2.3 Encoder

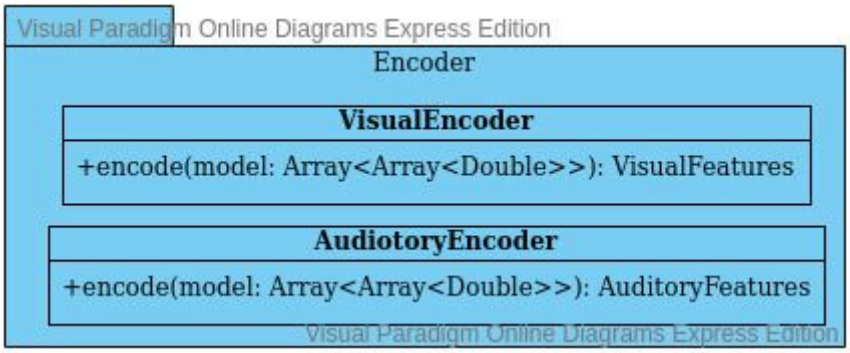

Figure 5. Extractor-Encoder Package Diagram

The encoder package contains classes that are responsible for the finished feature data and encoding it into storable, encrypted data to be later used by feature transferers.

2.1.3 Data Layer Packages

The data layer consists of packages that represent data in various stages from raw to processed.

2.1.3.1 Raw Data

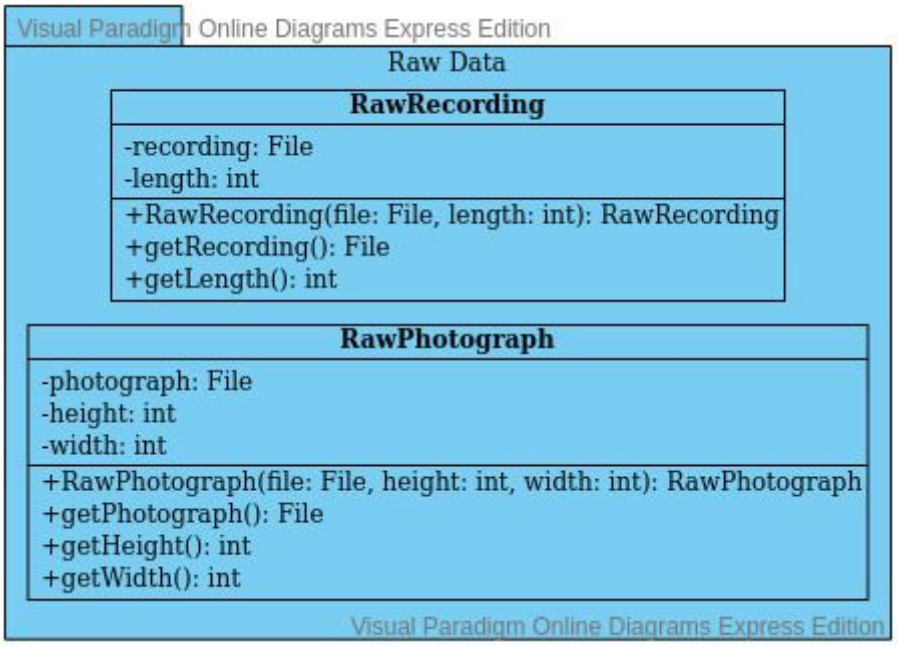

Figure 6. Extractor-Raw Data Package Diagram

The raw data package contains classes that represents raw audio, video and photograph files.

2.1.3.2 Processed Data

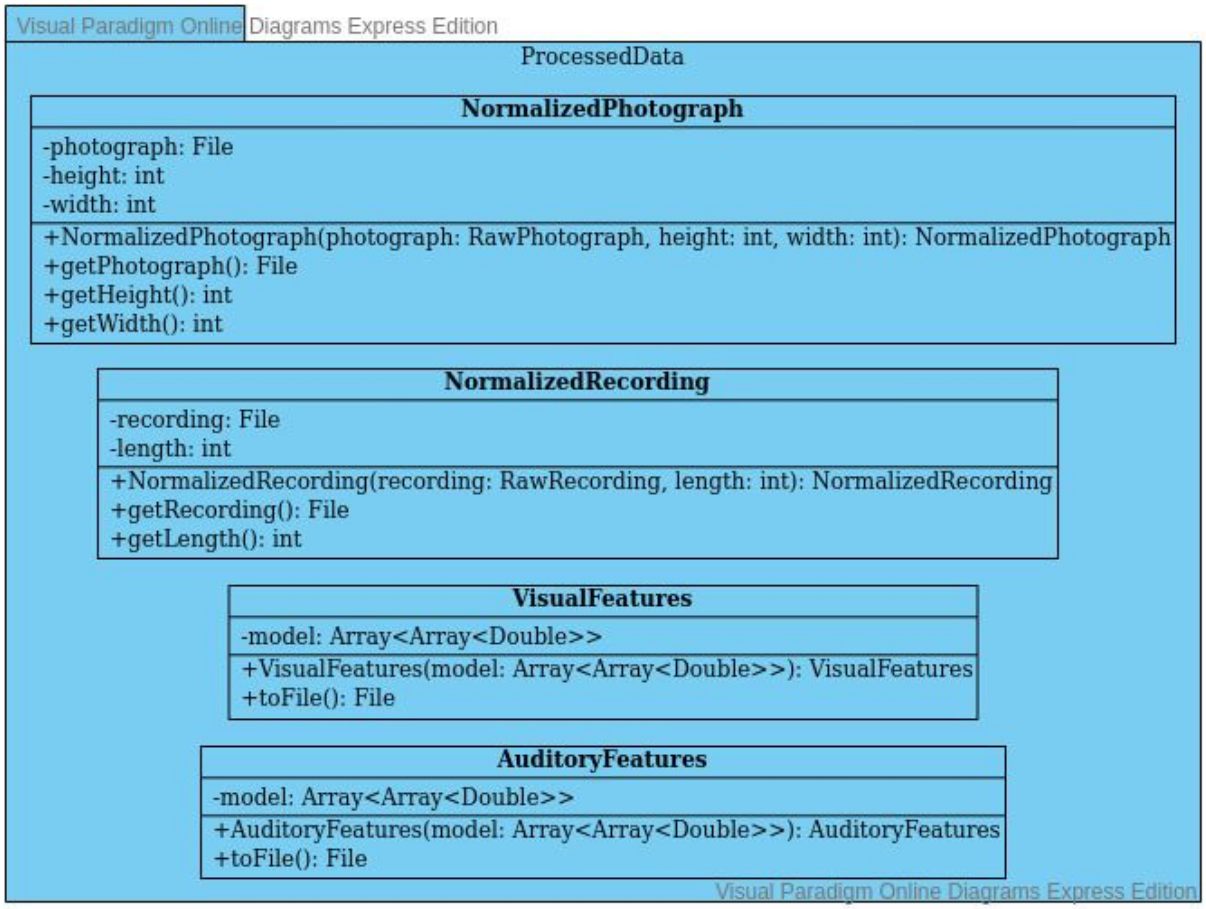

Figure 7. Extractor-Processed Data Package Diagram

The processed data package contains classes that represent data in various stages of processing such as denoised or normalized. Also, final products such as features in the form of encoded neural networks are represented in this package as well.

2.2 Feature Transferer Packages

Following is the package diagram of feature transferer part of the project:

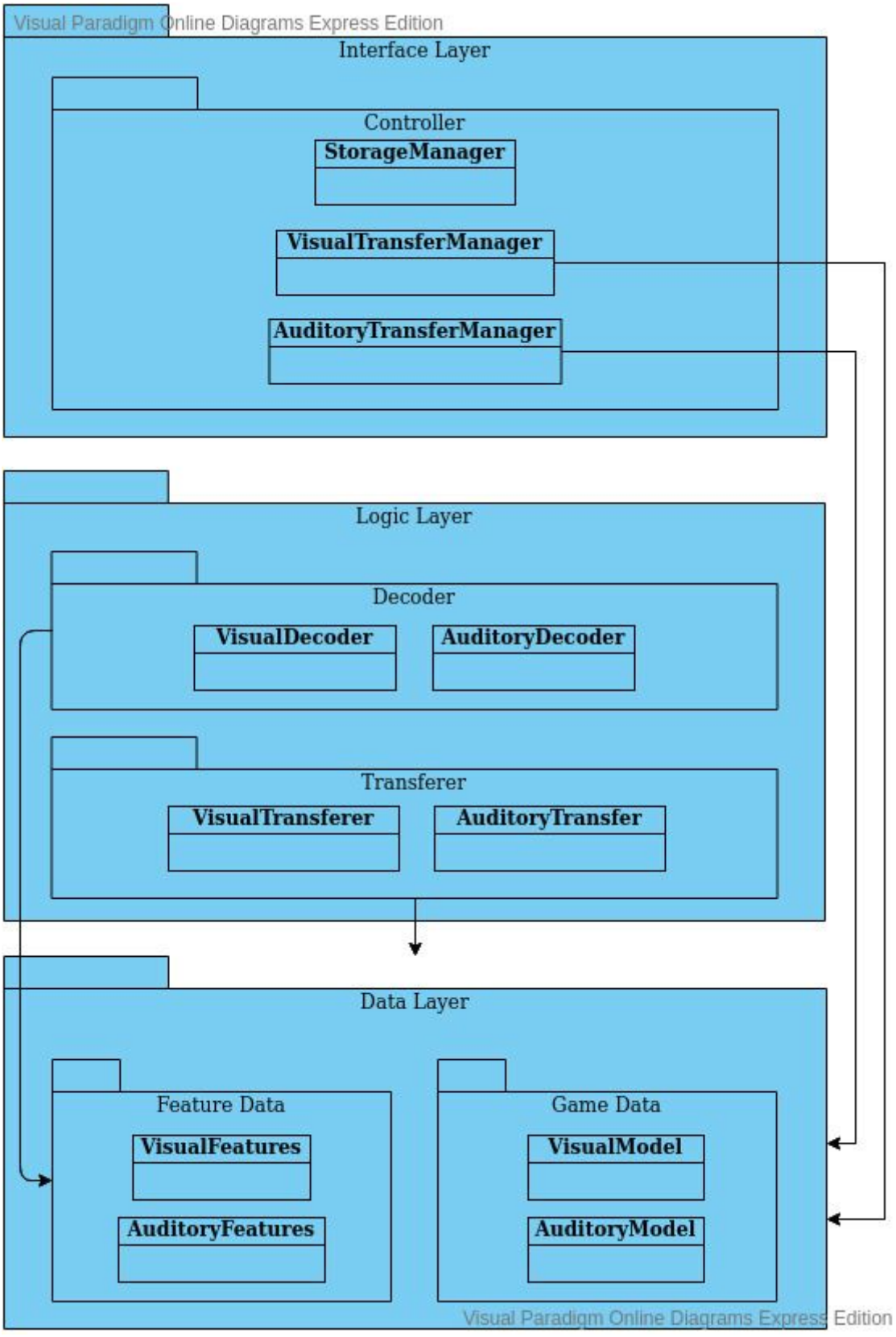

Figure 8. Transferer Packages Diagram

Once again, note that due to limitations of our drawing software and our desire to lessen clutter in the diagram we used improper notation. In the diagram above, straight arrows are used to denote a class or classes in a package use the classes from the package or packages at the end of the arrow.

Similar to the feature extractor part, the architecture featured here follows the guidelines of “Presentation - Logic - Data” architecture. In absence of direct interaction with the software, once again we opted to call the “Presentation Layer”, “Interface Layer” instead.

2.2.1 Interface Layer Packages

The interface layer consists of packages handling requests from the greater software to the Deepgame API. Existence of the interface layer allows software developers to use Deepgame Feature Extractor as a blackbox.

2.2.1.1 Controller

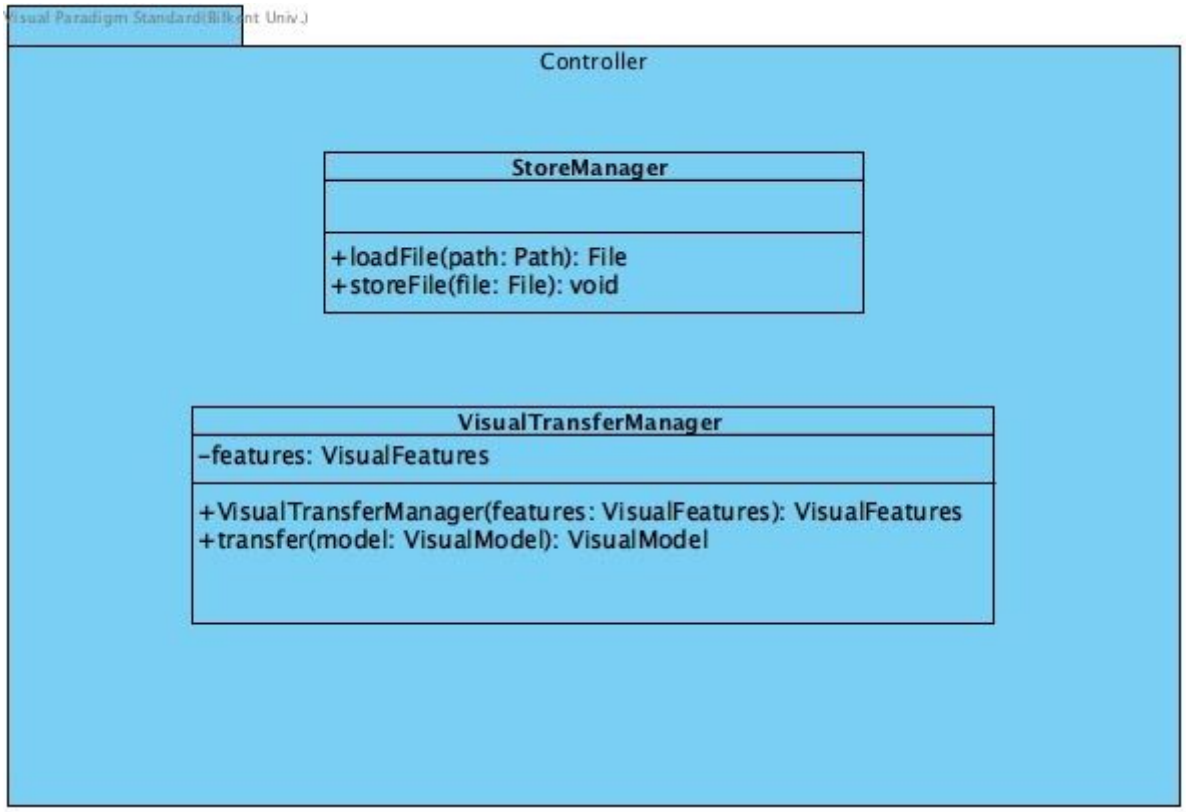

Figure 9. Transferer-Controller Package Diagram

The controller package contains classes to handle the requests of taking processed feature data along with game data and producing a model to be rendered by the game.

2.2.2 Logic Layer Packages

The logic layer consists of packages handling the use of feature data and game data to produce a game model with transferred features.

2.2.2.1 Decoder

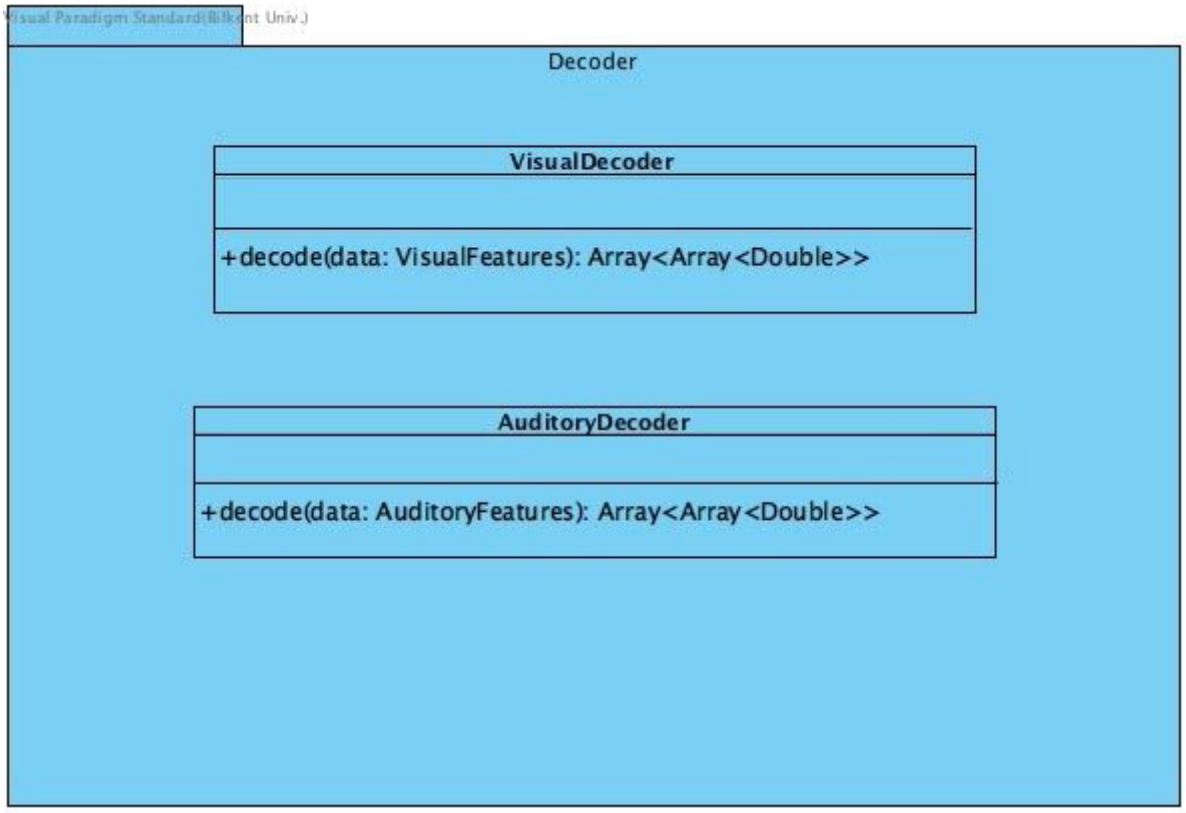

Figure 10. Transferer-Decoder Package Diagram

The decoder package contains classes that take feature data encoded by a feature extractor and decodes the data into data usable by the feature transferer.

2.2.2.2 Transferer

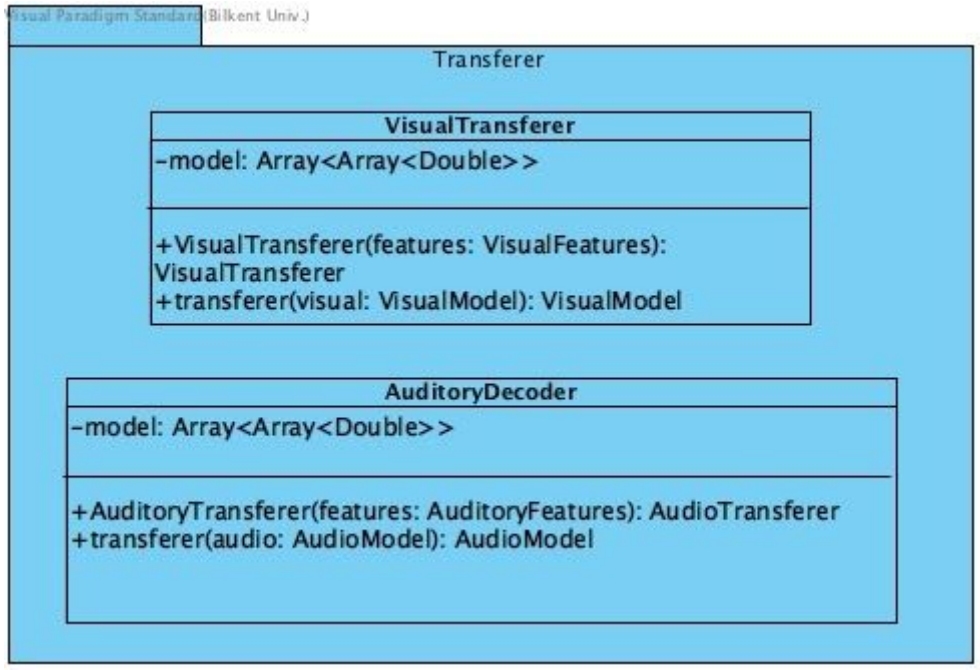

Figure 11. Transferer-Transferer Package Diagram

The transferer package contains classes that take feature data together with game data and produces a game model through the application of deepfake techniques [5].

2.2.3 Data Layer Packages

The data layer consists of packages that represent data from the feature extractor and the game integrating the Deepgame software. Then, these packages can be used by the rest of the feature transferer in a recognizable format.

2.2.3.1 Feature Data

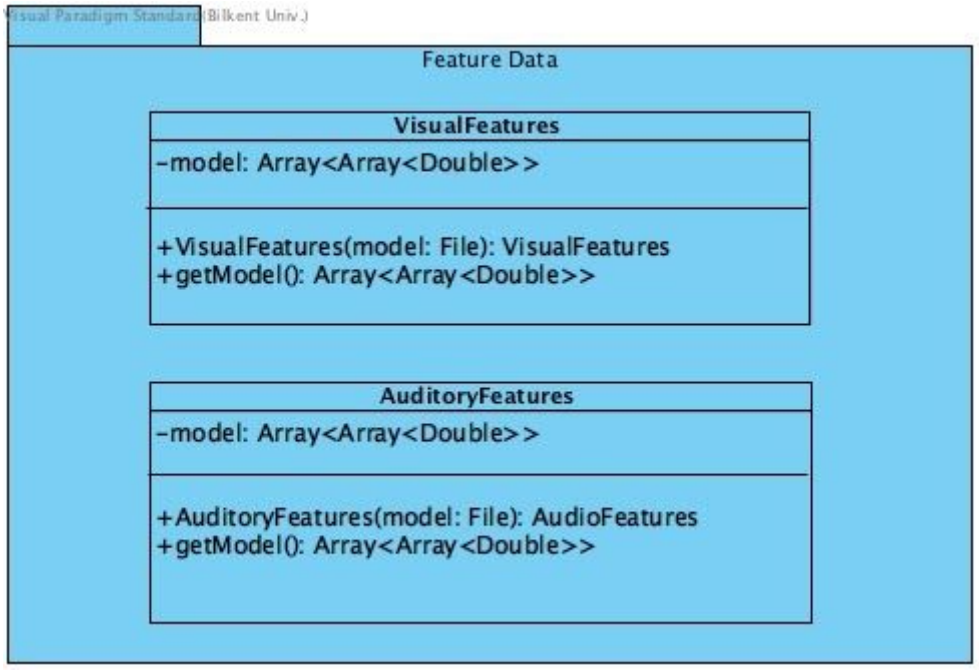

Figure 12. Transferer-Feature Data Package Diagram

The feature data package contains relevant classes from the processed data package of the feature extractor. That is because the extractor and the transferer are separate software but the data format is shared between these two parts.

2.2.3.2 Game Data

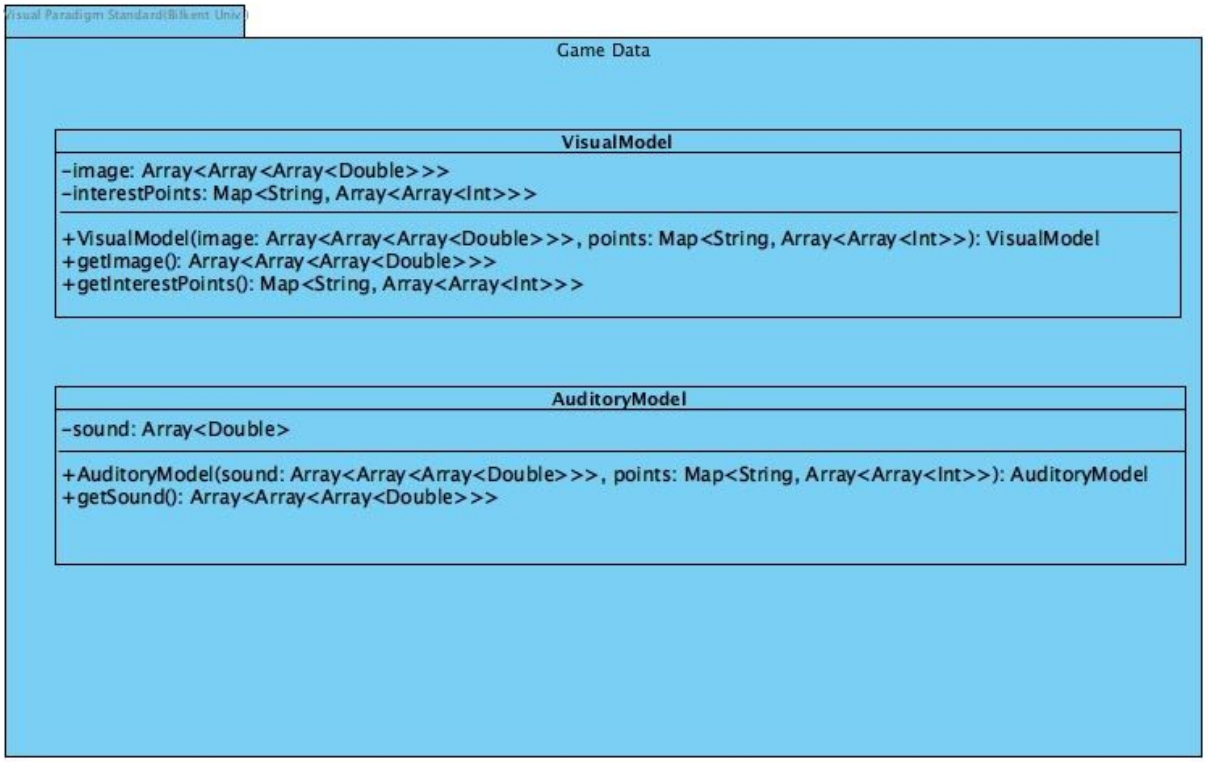

Figure 13. Transferer-Game Data Package Diagram

The game data package contains classes that define the data formats for the games that integrate the Deepgame software. Since Deepgame requires input from the game and produces an output to the game, a common data format is necessary.

3 Class Interfaces

3.1 Feature Extractor Classes

3.1.1 Interface Layer Classes

3.1.1.1 Controller

Package Name	Controller
Class Name	StorageManager
Description	
Manages the storage and retrieval of files.	
Attributes	
None.	
Methods	
+loadFile(path: Path): File Opens specified file. **+storeFile(file: File): void** Closes and saves specified file.	

Package Name	Controller
Class Name	ProcessingManager
Description	
Provides methods for the processing of data.	
Attributes	
-rawPhotographs: ArrayList<RawPhotograph> -rawRecordings: ArrayList<RawRecording> -intermediatePhotographs: ArrayList<NormalizedPhotograph> -intermediateRecordings: ArrayList<NormalizedRecording> -visualProcessResult: VisualFeatures -auditoryProcessResult: AuditoryFeatures	
Methods	
+ProcessingManager(rawPhotographs: ArrayList<File>, rawRecordings: ArrayList<File>): ProcessingManager Constructor. **+normalizePhotographs(): void** Takes photographs through normalization. **+normalizeRecordings(): void** Takes recordings through normalization. **+extractPhotographs(): void** Takes normalized photographs through extraction. **+extractRecordings(): void** Takes normalized recordings through extraction.	

+encodePhotographs(): void Encodes extracts from photographs. **+encodeRecordings(): void** Encodes extracts from recordings. **+getVisualEncoding(): VisualFeatures** Gets visual extraction result. **+getAuditoryEncoding(): AuditoryFeatures** Gets auditory extraction result.

3.1.2 Logic Layer Classes

3.1.2.1 Normalizer

Package Name	Normalizer
Class Name	Denoiser
Description	
Provides methods for denoising data through signal processing.	
Attributes	
None.	
Methods	
+denoisePhotograph(photo: RawPhotograph): NormalizedPhotograph **+denoiseRecording(recording: RawRecording): NormalizedRecording**	

Package Name	Normalizer
Class Name	Formatter
Description	
Provides methods for formatting denoised data.	
Attributes	
None.	
Methods	
+formatPhotograph(photo: NormalizedPhotograph): NormalizedPhotograph **+formatRecording(recording: NormalizedRecording): NormalizedRecording**	

3.1.2.2 Extractor

Package Name	Extractor
Class Name	VisualExtractor
Description	
Takes an array of normalized visual data and extracts visual features from them.	
Attributes	
-data: ArrayList<NormalizedPhotograph> -model: Array<Array<Double>>	
Methods	
+VisualExtractor(data: ArrayList<NormalizedPhotograph>): VisualExtractor Constructor. **-trainStep(): double** Goes through one iteration of training, returns error amount. **+train(): void** Goes through the entire neural network training. **+getModel(): Array<Array<Double>>** Returns the inner model.	

Package Name	Extractor
Class Name	AuditoryExtractor
Description	
Takes an array of normalized auditory data and extracts visual features from them.	
Attributes	
-data: ArrayList<NormalizedRecording> -model: Array<Array<Double>>	
Methods	
+AuditoryExtractor(data: ArrayList<NormalizedRecording>): AuditoryExtractor Constructor. **-trainStep(): double** Goes through one iteration of training, returns error amount. **+train(): void** Goes through the entire neural network training. **+getModel(): Array<Array<Double>>** Returns the inner model.	

3.1.2.3 Encoder

Package Name	Encoder
Class Name	VisualEncoder
Description	
Takes the model provided by the VisualExtractor and encodes it into a format usable by Deepgame feature transferers.	
Attributes	
None.	
Methods	
+encode(model: Array<Array<Double>>): VisualFeatures	

Package Name	Encoder
Class Name	AuditoryEncoder
Description	
Takes the model provided by the AuditoryExtractor and encodes it into a format usable by Deepgame feature transferers.	
Attributes	
None.	
Methods	
+encode(model: Array<Array<Double>>): AuditoryFeatures	

3.1.3 Data Layer Classes

3.1.3.1 Raw Data

Package Name	Raw Data
Class Name	RawPhotograph
Description	
File containing an unprocessed photograph.	
Attributes	
-photograph: File -height: int	

-width: int
Methods
+RawPhotograph(file: File, height: int, width: int): RawPhotograph Constructor. **+getPhotograph(): File** Photograph getter. **+getHeight(): int** Pixel height of the photograph. **+getWidth(): int** Pixel width of the photograph.

Package Name	Raw Data
Class Name	RawRecording
Description	
File containing an unprocessed recording.	
Attributes	
-recording: File -length: int	
Methods	
+RawRecording(file: File, length: int): RawRecording Constructor. **+getRecording(): File** Recording getter. **+getLength(): int** Duration of the recording in seconds.	

3.1.3.2 Processed Data

Package Name	Processed Data
Class Name	NormalizedPhotograph
Description	
File containing a photograph after normalization.	
Attributes	
-photograph: File -height: int -width: int	
Methods	
+NormalizedPhotograph(photograph: RawPhotograph, height: int, width: int): NormalizedPhotograph Constructor. **+getPhotograph(): File** Photograph getter. **+getHeight(): int** Pixel height of the photograph. **+getWidth(): int** Pixel width of the photograph.	

Package Name	Processed Data
Class Name	NormalizedRecording
Description	
File containing a recording after normalization.	
Attributes	
-recording: File -length: int	
Methods	
+NormalizedRecording(recording: RawRecording, length: int): NormalizedRecording Constructor. **+getRecording(): File** Recording getter. **+getLength(): int** Duration of the recording in seconds.	

Package Name	Processed Data
Class Name	VisualFeatures
Description	
Neural network model trained from visual data encoded in a data package.	
Attributes	
-model: Array<Array<Double>>	
Methods	
+VisualFeatures(model: Array<Array<Double>>): VisualFeatures Constructor. **+toFile(): File** Method to create a file encoding of the model.	

Package Name	Processed Data
Class Name	AuditoryFeatures
Description	
Neural network model trained from auditory data encoded in a data package.	
Attributes	
-model: Array<Array<Double>>	

Methods
+AuditoryFeatures(model: Array<Array<Double>>): AuditoryFeatures Constructor. **+toFile(): File** Method to create a file encoding of the model.

3.2 Feature Transferer Classes

3.2.1 Interface Layer Classes

3.2.1.1 Controller

Package Name	Controller
Class Name	StorageManager
Description	
Manages the storage and retrieval of files.	
Attributes	
None.	
Methods	
+loadFile(path: Path): File Opens specified file. **+storeFile(file: File): void** Closes and saves specified file.	

Package Name	Controller
Class Name	VisualTransferManager
Description	
Makes up the interface between Deepgame Visual Engine and the game integrating the software. Constructed by feeding a premade feature data, this class is fed game visual data and returns visuals to be rendered that have the likeness of the target from the feature data.	
Attributes	
-features: VisualFeatures	
Methods	
+VisualTransferManager(features: VisualFeatures): VisualTransferManager Constructor. **+transfer(model: VisualModel): VisualModel** Takes visuals from game and returns new game visuals with target's features transferred.	

Package Name	Controller
Class Name	AuditoryTransferManager
Description	
Makes up the interface between Deepgame Auditory Engine and the game integrating the software. Constructed by feeding a premade feature data, this class is fed game voice lines and returns lines that sound like they were spoken by the target person from the feature data.	
Attributes	
-features: VisualFeatures	
Methods	
+AuditoryTransferManager(features: AuditoryFeatures): AuditoryTransferManager Constructor. **+transfer(model: AuditoryModel): AuditoryModel** Takes voice lines from the game and returns new voice lines with the target's features transferred.	

3.2.2 Logic Layer Classes

3.2.2.1 Decoder

Package Name	Decoder
Class Name	VisualDecoder
Description	
Takes the encoded data produced by the feature extractor and decodes it into something usable by the feature transferer.	
Attributes	
None.	
Methods	
+decode(data: VisualFeatures): Array<Array<Double>>	

Package Name	Decoder
Class Name	AuditoryDecoder
Description	
Takes the encoded data produced by the feature extractor and decodes it into something	

usable by the feature transferer.
Attributes
None.
Methods
+decode(data: AuditoryFeatures): Array<Array<Double>>

3.2.2.2 Transferer

Package Name	Transferer
Class Name	VisualTransferer
Description	
Keeps an internal neural network model of the target person and transfers the target's features to the input game visual data.	
Attributes	
-model: Array<Array<Double>>	
Methods	
+VisualTransferer(features: VisualFeatures): VisualTransferer Constructor. **+transfer(visual: VisualModel): VisualModel** Takes unprocessed game visual and returns it after transferring features onto it.	

Package Name	Transferer
Class Name	AuditoryTransferer
Description	
Keeps an internal neural network model of the target person and transfers the target's features to the input game auditory data.	
Attributes	
-model: Array<Array<Double>>	
Methods	
+AuditoryTransferer(features: AuditoryFeatures): AuditoryTransferer Constructor. **+transfer(audio: AuditoryModel): AuditoryModel** Takes an unprocessed voice line and returns it after transferring features onto it.	

3.2.3 Data Layer Classes

3.2.3.1 Feature Data

Package Name	Feature Data
Class Name	VisualFeatures
Description	
Neural network model trained from visual data encoded in a data package.	
Attributes	
-model: Array<Array<Double>>	
Methods	
+VisualFeatures(model: File): VisualFeatures Constructor. **+getModel(): Array<Array<Double>>** Gets the model from within the object.	

Package Name	Feature Data
Class Name	AuditoryFeatures
Description	
Neural network model trained from auditory data encoded in a data package.	
Attributes	
-model: Array<Array<Double>>	
Methods	
+AuditoryFeatures(model: File): AuditoryFeatures Constructor. **+getModel(): Array<Array<Double>>** Gets the model from within the object.	

3.2.3.2 Game Data

Package Name	Game Data
Class Name	VisualModel
Description	
Standard format for the games feeding Deepgame’s feature transferer visual data. Data is both input and output as this class. Conversion between this format and the game's internal format is left to the game developers. There are no setters because this class is meant to be immutable.	

Attributes
-image: Array<Array<Array<Double>>> -interestPoints: Map<String, Array<Array<Int>>>
Methods
+VisualModel(image: Array<Array<Array<Double>>>, points: Map<String, Array<Array<Int>>): VisualModel Constructor. **+getImage(): Array<Array<Array<Double>>>** Returns the image array. **+getInterestPoints(): Map<String, Array<Array<Int>>>** Returns interest points.

Package Name	Game Data
Class Name	AuditoryModel
Description	
Standard format for the games feeding Deepgame's feature transferer auditory data. Data is both input and output as this class. Conversion between this format and the game's internal format is left to the game developers. There are no setters because this class is meant to be immutable.	
Attributes	
-sound: Array<Double>	
Methods	
+AuditoryModel(sound: Array<Double>): AuditoryModel Constructor. **+getSound(): Array<Double>** Returns the sound data.	

4 Glossary

Following is the information about the pages where these terms are used.

5 References

[1] Li, Y. and Lyu, S., 2020. [online] Openaccess.thecvf.com. Available at: <https://openaccess.thecvf.com/content_CVPRW_2019/papers/Media%20Forensics/Li_Exposing_DeepFake_Videos_By_Detecting_Face_Warping_Artifacts_CVPRW_2019_paper.pdf> [Accessed 5 October 2020].

[2] D. Güera and E. J. Delp, "Deepfake Video Detection Using Recurrent Neural Networks," 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), Auckland, New Zealand, 2018, pp. 1-6, doi: 10.1109/AVSS.2018.8639163.

[3] Libraryguides.vu.edu.au. 2020. *Library Guides: IEEE Referencing: Getting Started With IEEE Referencing*. [online] Available at: <https://libraryguides.vu.edu.au/ieeereferencing/gettingstarted#:~:text=%E2%80%9CIEEE%E2%80%9D%20stands%20for%20The%20Institute,paper%2C%20provided%20in%20square%20brackets.&text=This%20is%20known%20as%20an,of%20the%20work%20are%20provided.> [Accessed 5 October 2020].

[4] Agilemodeling.com. 2020. *Introduction To The Diagrams Of UML 2.X*. [online] Available at: <http://agilemodeling.com/essays/umlDiagrams.htm#:~:text=Although%20there%20is%20far%20more,a%20system%20or%20business%20process.> [Accessed 5 October 2020].

[5] Nguyen, Thanh Thi, et al. “Deep Learning for Deepfakes Creation and Detection: A Survey.” *Https://Arxiv.org*, 28 July 2020, arxiv.org/pdf/1909.11573.pdf.