Table 1: Brief description of the BiometricLogic class and all its methods

BiometricLogic	
The BiometricLogic class is responsible for analyzing biometric values and determining if certain conditions are met.	
calculateSTD_DEV(): void	This method is responsible for calculating the standard deviation of the biometric values in order to establish a baseline. The standard deviation that is calculated is then stored in the attribute std_dev in the class.
isSuspicious(float, float, float): void	This method is responsible for comparing the new biometric values that were read from the Emotibit with the already established standard deviation. If the new values are within the standard deviation then the suspicious attribute inside the class is set to false, if they are not within the standard deviation then the suspicious attribute is set to true.
getSuspicious(): bool	This method is responsible for returning the value that is stored inside the suspicious attribute. This is necessary to access this value outside of the class for other uses.

Table 2: Brief description of the DatabaseAPI class and all its methods

DatabaseAPI	
The DatabaseAPI class is responsible for interaction between the classes and the database.	
insert(float,float, float, float, float): void	This method is sending a query to the database to insert the time and biometric data to the Biometric Feedback table. The connection to the database should already have been established.
insert(float, float, bool, string, string): void	This method is sending a query to the database to insert the start and finish time of the user input, the discrepancy flag, the user input, and the generated response. The connection to the database should already have been established.
fetch(float, float, float): tuple	This method is sending a query to the database to fetch the biometric data in the database.The connection to the database should already have been established.

fetch(string): tuple	This method is sending a query to the database to fetch the interaction between the user and the NPC in the database. The connection to the database should have already been established.

Table 3: Brief description of the EmotibitReader class and all of its methods

EmotibitReader	
The EmotibitReader class is responsible for handling data collection and data transfer.	
readData(): void	This method is responsible for collecting the biometric data from the Emotibit that is connected to the user. The data collected is written to a csv file.
setBiometricValues(): void	This method is responsible for opening the csv file where the biometric data is stored and searching through the file biometric values we need. When it finds the temperature, heart rate, and skin conduction values it sets those values in the appropriate attributes in the object for future use.
sendToDB(): void	This method calls the insert(float, float, float, float, float) method from the DatabaseAPI object that is stored as a reference inside the EmotibitReader class. The arguments for the insert method are the biometric and time values stored in the attributes, startTime, finishTime, temperature, heart rate, and skin conduction.
compareValue(): void	This method calls the isSuspicious(float, float, float) method from the BiometricLogic object that is stored as a reference inside the EmotibitReader class. The arguments for the isSuspicious method are the values stored in the attributes temperature, heart rate, and skin conduction.
setFilename(): void	This method is responsible for setting the file name of where the biometric data that is collected from the Emotibit is being stored. This method allows us to set different file names outside of the class if necessary.
setStartTime(float): void	This method is responsible for setting the start time of when the biometric data was collected. This method allows us to set the startTime attribute outside of the class
setFinishTime(float): void	This method is responsible for setting the finish time of when the biometric data was collected. This method allows us to set the finishTime attribute outside of the class

Table 4: Brief description of the GameController class and of all its methods

GameController	
The GameController class is responsible for generating the NPC responses and sending those responses to the database.	
analyzeInput(): void	This method is responsible for analyzing the input from the user to detect any discrepancies, this is done by utilizing OpenAI as it has memory capabilities. If there are discrepancies in what the user says then the discrepancy attribute will be set to true, if there is not then it will be set to false.
generateResponse(): void	This method is responsible for generating the response to the user input. This method calls the prompt(OpenAI) method from the NLP interface object; it holds a reference to generate the response, the response is then returned and stored in the response attribute within the class.
getResponse(): string	This method is responsible for returning the value that is stored within the response attribute. This is necessary for the NPC class to have access to the new response.
sendToDB(): void	This method calls the insert(float, float, string, string, bool) function in the DatabaseAPI object which is stored as a reference in the class. The parameters that are passed into the insert method are startTime, endTime, userInput, response, discrepancy.
setNLPInterface(NLPInterface): void	This method allows us to set the NLP interface that chooses what kind of response is generated. This method allows us to determine the NLPInterface the GameController should use outside of the class.

Table 5: Brief description of the NLPInterface class and of all its methods.

NLPInterface	
The NLPInterface class is just responsible for defining the interface for the different implementations of these methods. The implementations are within the children of this class.	
prompt(OpenAI): string	This method is responsible for defining the abstract method, the parameter is an OpenAI object, and it returns a string. The implementation is within the children that inherit from this class.

Table 6: Brief description of the IntenseStyle class and all of its methods.

IntenseStyle	
The IntenseStyle class inherits from the NLPInterface class so it's responsible for implementing its own version of the abstract methods.	
prompt(OpenAI): string	This method is responsible for defining the abstract method, the parameter is an OpenAI object, and it returns a string. In this method we prompt chatGPT to generate a response with an intense style.

Table 7: Brief description of the FriendlyStyle class and all of its methods.

FriendlyStyle	
The FriendlyStyle class inherits from the NLPInterface class so it's responsible for implementing its own version of the abstract methods.	
prompt(OpenAI): string	This method is responsible for defining the abstract method, the parameter is an OpenAI object, and it returns a string. In this method we prompt chatGPT to generate a response with a friendly style.

Table 8: Brief description of the ProfessionalStyle class and all of its methods.

ProfessionalStyle	
The ProfessionalStyle class inherits from the NLPInterface class so it's responsible for implementing its own version of the abstract methods.	
prompt(OpenAI): string	This method is responsible for defining the abstract method, the parameter is an OpenAI object, and it returns a string. In this method we prompt chatGPT to generate a response with a professional style.

Table 9: Brief description of the NPC class and all of its methods.

NPC	
The NPC class is responsible for storing all the NPC components and triggering their speech functionality.	
setModel(ModelComponent): void	This method is responsible for controlling the visuals for the NPC. This is done by setting the model component in this class.

setTransform(TransformComponent): void	This method is responsible for controlling the position, rotation, and scale of the NPC. This is done by setting the transform component in this class.
setSpeak(SpeakComponent): void	This method is responsible for controlling how the NPC talks to the user. This is done by setting the speak component in this class.
getModel(): ModelComponent	This method is responsible for returning the model of the NPC. This is necessary to access the model outside of the class.
getTransfrom(): TransformComponent	This method is responsible for returning the transform of the NPC. This is necessary to access the position, rotation, and scale of the NPC outside of the class.
getSpeak(): SpeakComponent	This method is responsible for returning the speak component of the NPC. This is necessary to access the speak component outside of the class.
speak(): void	This method is responsible for calling the playAudio(string) method from the speak component object that is stored as a reference in this class. The arguments for the playAudio method is the response from the GameController object that is stored as a reference in this class.

Table 10: Brief description of the SpeakComponent class and all of its methods.

SpeakComponent	
The SpeakCompnent class is responsible for storing the necessary information for the NPC to talk and to implement the talking functionality.	
setModel(string): void	This method is responsible for defining the model that is being used for OpenAI's text-to-speech functionality. This method allows us to define the model outside of the class.
setVoice(string): void	This method is responsible for defining the voice of the model being used for OpenAI’s text-to-speech functionality. This method allows use to define the voice outside of the class.

getModel(): string	This method is responsible for returning the model that is being used for OpenAI's text-to-speech functionality. This method allows us to access the model outside of the class.
getVoice(): string	This method is responsible for returning the voice of the model being used for OpenAI's text-to-speech functionality. This method allows us to access the voice outside of the class.
playAudio(string): void	This method is responsible for using OpenAI's text-to-speech functionality to play the audio of the generated response to the user. The response is passed in as an argument from the NPCs class, where this method is invoked.

Table 11: Brief description of the SpeechRecognition class and all of its methods.

SpeechRecognition	
The SpeechRecognition class is responsible for collected speech input from the user and .	
getUserInput(): string	This method is responsible for returning the speech user input after it has been transcribed to a string. This allows us to access the user input from outside the class.
setStartTime(float): void	This method is responsible for defining the time when the user started talking. This is necessary to keep track of for the database.
setEndTime(float): void	This method is responsible for defining the time when the user finished talking. This is necessary to keep track of for the database.
getStartTime(): float	This method is responsible for returning the time when the user started talking. This is necessary to access the time the user started talking outside the class.
getEndTime(): float	This method is responsible for returning the time when the user finished talking. This is necessary to access the time the user finished talking outside the class.
listen(): void	This method is responsible for collecting the speech input from the user and transcribing it into an understandable form. It is then stored inside the userInput attribute in the class.