BINUS UNIVERSITY INTERNATIONAL

Assignment Cover Letter

(Individual Work)

Student Information:		Surname	Given Names	Student ID Number
	1.	Akbar	Gardyan	2301902296

Course Code	: COMP6502	Course Name	: Introduction to Programming
Class	: L1AC	Name of Lecturer(s)	: Ida Bagus Kerthyayana
Major	: CS		
Title of Assignment (if any)	: AI Roulette		
Type of Assignment	: Final Project		
Submission Pattern			
Due Date	: 17-01-2020	Submission Date	: 15-01-2020

Project Specification

The purpose of the program is to be a party game that can be played between friends. The game involves chance and artificial intelligence (AI). One of the players would choose an AI to play with. They will be asked to draw a character (a number from 0-9 or an alphabet character). Their chosen AI will try to predict what character did the player drew. Should the AI guess it correctly, every other player gets a “punishment”, otherwise the player who drew and chose the AI gets the “punishment”. This “punishment” can be anything that the players have decided upon: a game of truth or dare, do push-ups, sing, and so on. Some information regarding AIs and neural networks are presented to the players during loading screens as to allow them to glean some little information regarding them as they play.

More specifications regarding the project are as follows:

- Input of the program:
    - Chosen AI
    - A drawing/sketch of a character for the AI to predict
- Output of the program:
    - Result of the AI’s guess being displayed unto the game screen
- Third-party libraries used in the making of the program:
    - Matplotlib
        - Used for evaluating the result of the neural network during its training period
    - Numpy
        - Used to manipulate the input and output of the neural network
    - OpenCV
        - Used to load images and format them for neural network input
    - Pandas
        - Used to load and process csv files
    - Pygame
        - Used to make the game and program’s GUI
    - Tensorflow
        - Used for creating neural network

Solution Design

The game will have a series of panels that identifies each section of the game. These panels can be navigated by clicking on the buttons displayed on the screen. The list of panels in the game are as follows:

1. Main menu
2. Instructions panel
3. Game mode selection
4. AI selection
5. AI selection (Random)
6. Draw panel
7. Draw panel (Random)
8. Loading screen
9. Results panel

<u>Main Menu</u>

The main menu will be the landing page of the game. As the player launches or runs the game, this panel will be the first thing they see. There are 3 buttons available on this panel: Start, Instructions, and Exit. The Start button will guide the player to start the game itself, as it brings them to the game mode selection panel. The Instructions button, on the other hand, will open the Instructions panel. Finally the Exit button, as the name suggests, will exit the game.

<u>Instructions Panel</u>

In this panel, players will be able to view how to play the game alongside its accompanying rules. This is a simple panel with only one button that says “Ok” and will bring the players back to the main menu when clicked.

<u>Game Mode Selection</u>

In this panel players will be able to select what game mode they wish to play from the currently available game modes: Fun House and Random Chaos. As they move their mouse to the buttons to hover, information regarding each game mode will be displayed on the side of the buttons to inform the players. The list of available buttons on this panel alongside their actions are listed below:

- Fun House
    - Brings the player to the AI selection panel.
- Random Chaos

    - Brings the player to the AI selection (Random) panel.
- Back
    - Brings the player back to the main menu.

<u>AI Selection</u>

The purpose of this panel is to provide a platform for the player to choose their AI. In this panel, which is accessible through the Fun House button in the game mode selection panel, the player will be able to choose their AI by clicking on their buttons. Hovering their mouse over each of the AI buttons will display the image of the AI and its details. Players can select the AI they want to play with by clicking on the respective button. A green highlight will be added to the button to signify that that AI is selected. Clicking on the same AI button again will deselect it and remove the green highlight. The Choose button, however, will only be enabled once the player has chosen the AI they wish to play with. Should the player deselect an AI, the Choose button will become disabled again. This ensures the game to know which neural network model to load and make predictions from. A Back button is also provided here that will bring the user back to the game mode selection panel.

<u>AI Selection (Random)</u>

This panel is very much similar to the AI selection panel described above, except that the AI buttons will not be displaying the names of the AIs, but a mysterious "???????" as to not reveal the AI itself. Details regarding the AI and its corresponding image will not be displayed. The Choose and Exit buttons works very much the same. This panel is only accessible by clicking on the Random Chaos button on the game mode selection panel.

<u>Draw Panel</u>

This panel is a special panel as it gives the player more functionality with their mouse rather than just for clicking on buttons. Using their mouse, and holding the left mouse button, the user can draw into the screen. They will be able to draw a letter, a word, a house, or pretty much anything they can imagine sketching. However, this panel is intended for the player to draw a single character from the alphabet or a number from 0 to 9 that their chosen AI will try their best to guess. A label is displayed to guide the player as to what to draw. The buttons available here and their functions are listed below:

- Back
    - Brings the player to the previous panel (the AI selection panel).
- Clear
    - This button clears the screen from the player's drawn image. This button will be useful when they wish to reset their drawing or when they made a mistake.
- Guess

- Brings the game to the loading screen where the neural network model of their chosen AI will be loaded and their drawn image predicted.

Draw Panel (Random)

This panel is similar to the above mentioned Draw panel, except that the Back button will bring the player back to the AI Selection (Random) panel. Everything else remains the same.

Loading Screen

This section of the game is unique in the sense that there will be no user interaction. There will be nothing to draw nor any buttons to click. This panel serves as a buffer for players to wait as their AI's neural network model is loaded, their drawing loaded and formatted before being fed to the neural network, and where it will be predicted. A label is displayed alongside an hourglass animation that gets updated as the loading process progresses. Another label is also displayed with variable text but with a much larger font size that displays random characters in the alphabet or a number from 0 to 9 as to make the player wait with anticipation as the neural model is loaded up and hoping it will predict their drawing correctly. Additionally, a final label will be displaying random trivia texts regarding AI and neural networks. The aim for this is so that as the players wait, they would get little insights regarding the subject instead of just boringly awaiting the results.

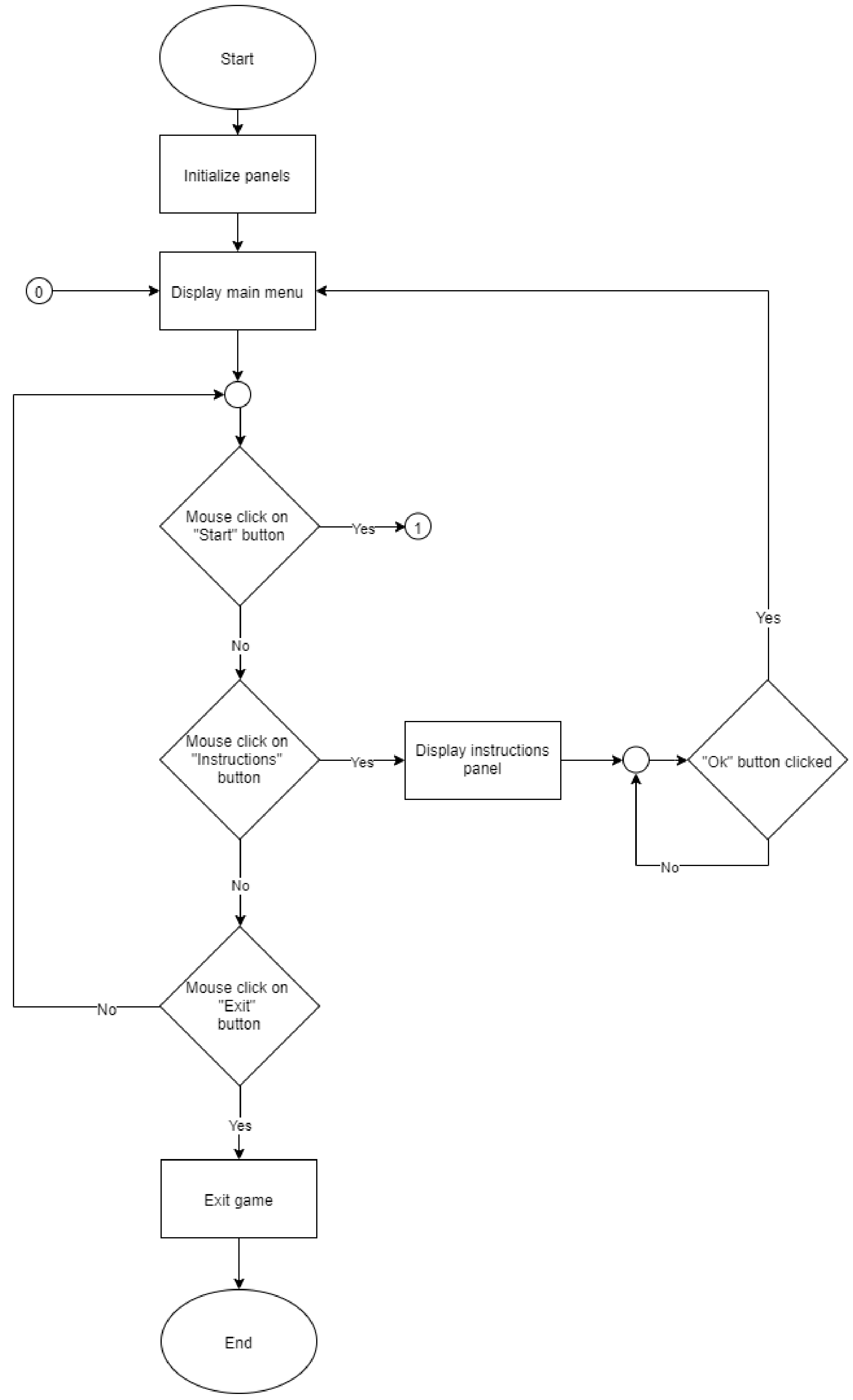
Results Panel

Speaking of results, once the loading screen has completed its tasks, the game will land on this panel. Here, a copy of the player's drawing is displayed alongside their AI's prediction towards it. A label is displayed to give out instructions on what to do next. To complete the round (or the game), a Finish button is given that will bring the player to the main menu when clicked.

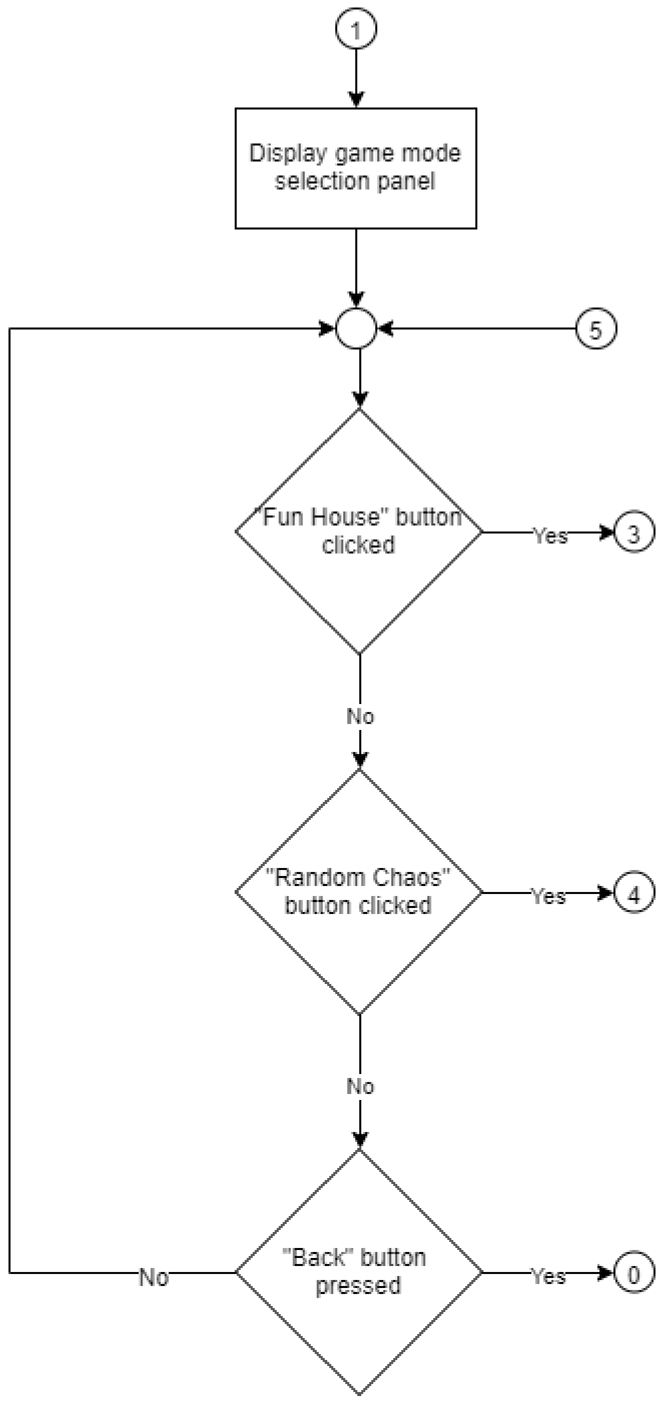
The flowchart for the whole game details how the game itself flows and interacts with each of the available panels. It can be found on the next page.

Flowchart

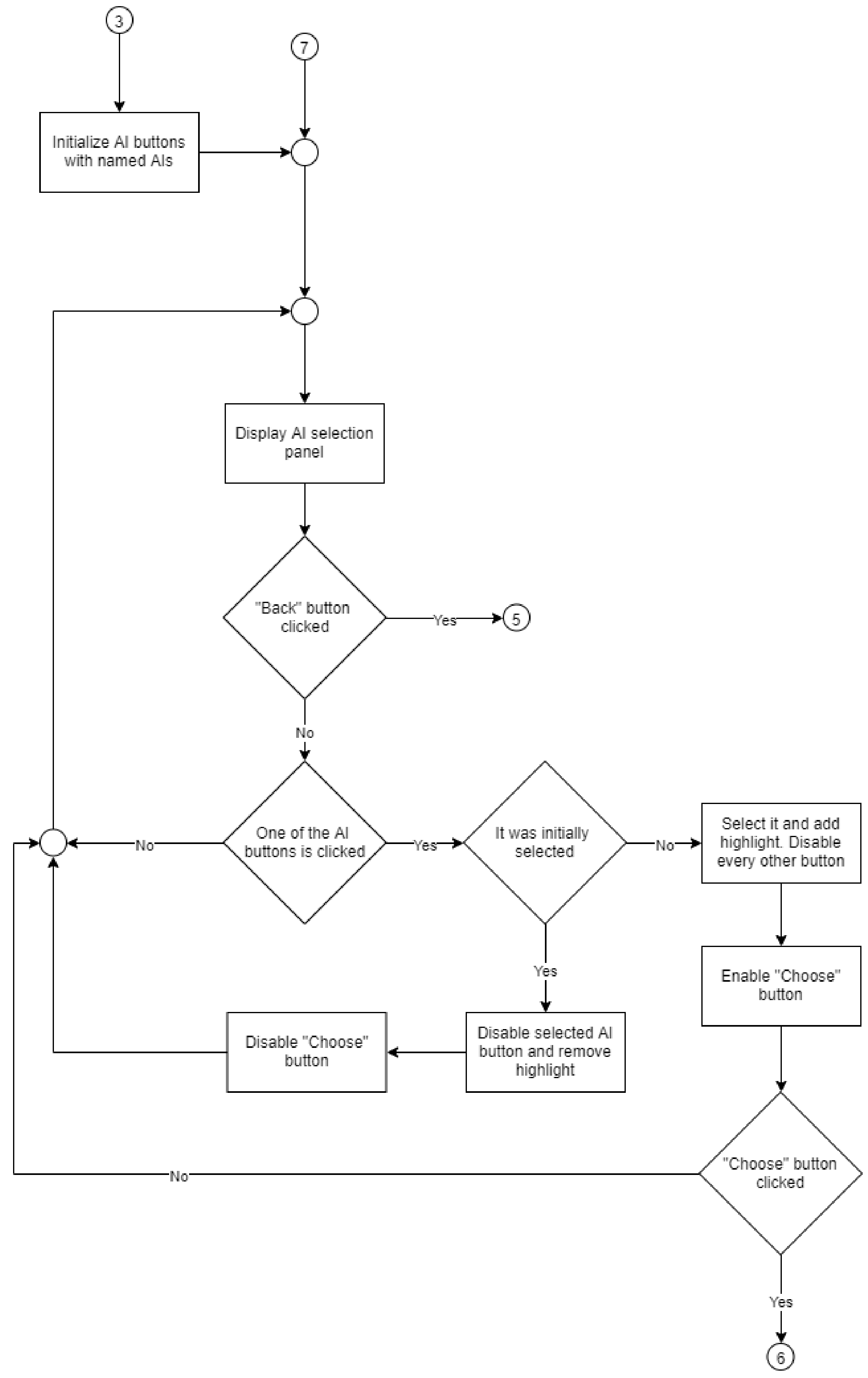
Main Menu:


Start
Initialize panels
0
Display main menu
Mouse click on "Start" button
Yes
1
No
Mouse click on "Instructions" button
Yes
Display instructions panel
"Ok" button clicked
Yes
No
No
Mouse click on "Exit" button
No
Yes
Exit game
End

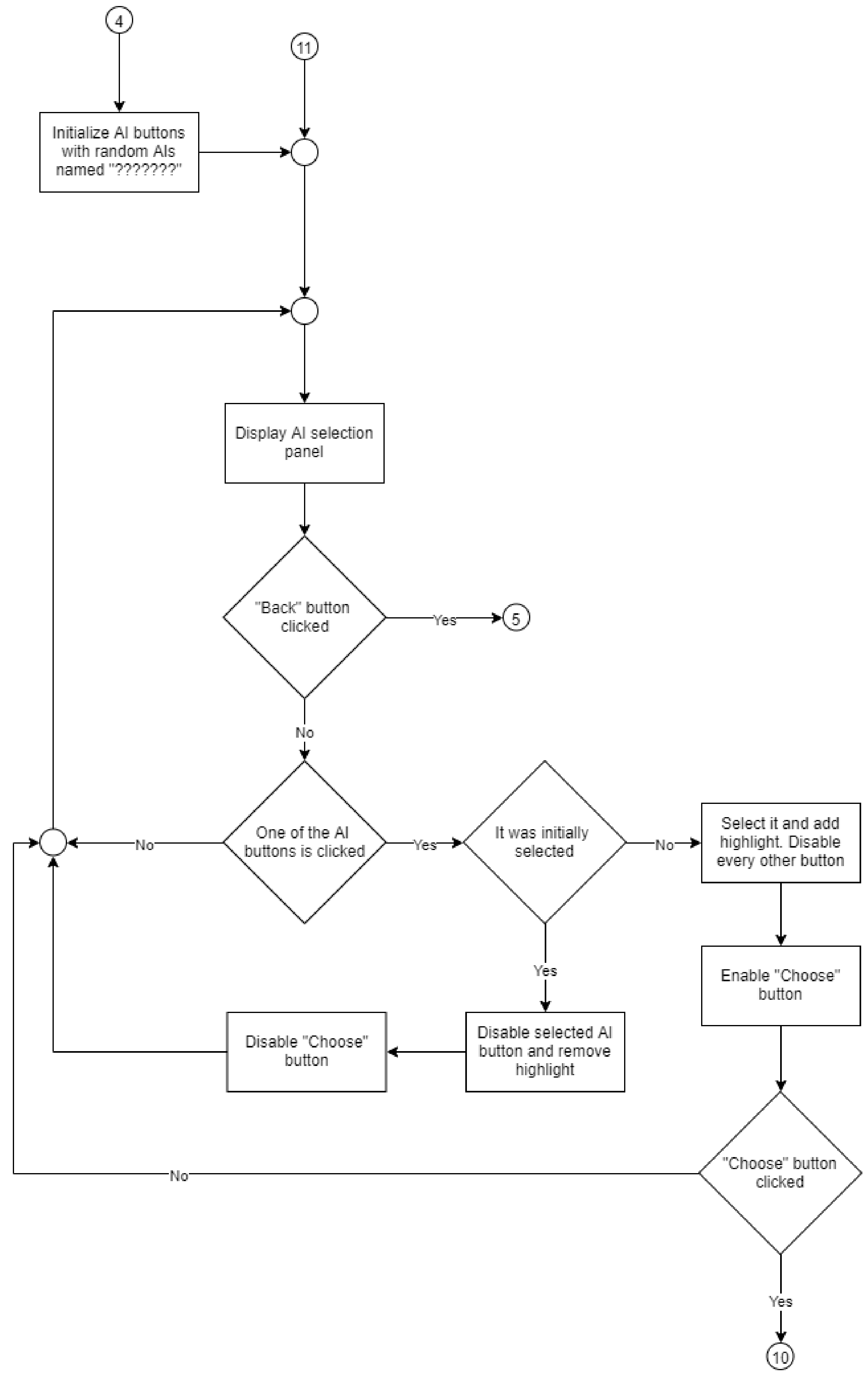
Game Mode Selection:


1
Display game mode selection panel
5
"Fun House" button clicked
Yes
3
No
"Random Chaos" button clicked
Yes
4
No
"Back" button pressed
Yes
0
No

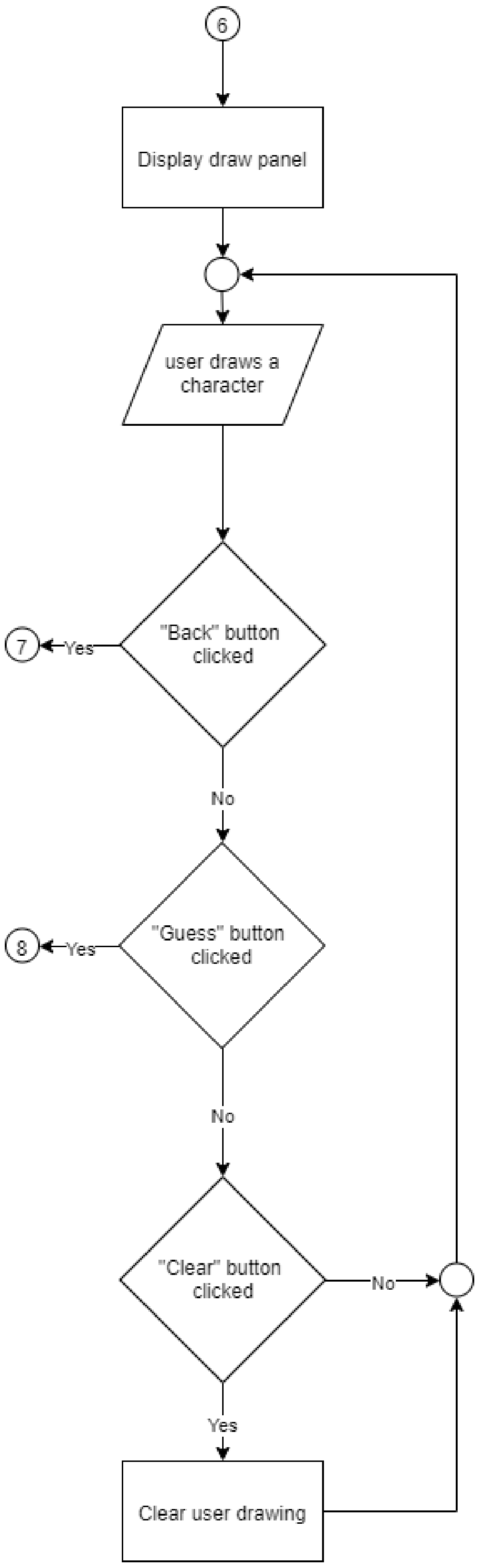
AI selection:

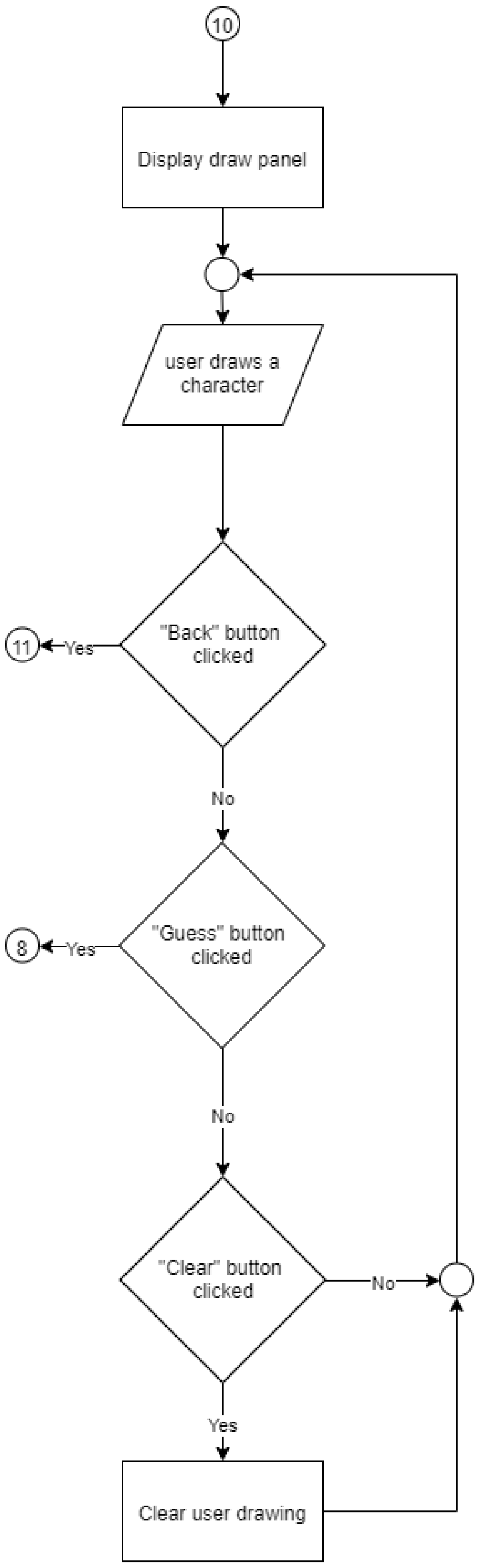

3
7
Initialize AI buttons with named AIs
Display AI selection panel
"Back" button clicked
Yes
5
No
One of the AI buttons is clicked
No
Yes
It was initially selected
No
Select it and add highlight. Disable every other button
Yes
Disable selected AI button and remove highlight
Disable "Choose" button
Enable "Choose" button
"Choose" button clicked
No
Yes
6

AI Selection (Random):

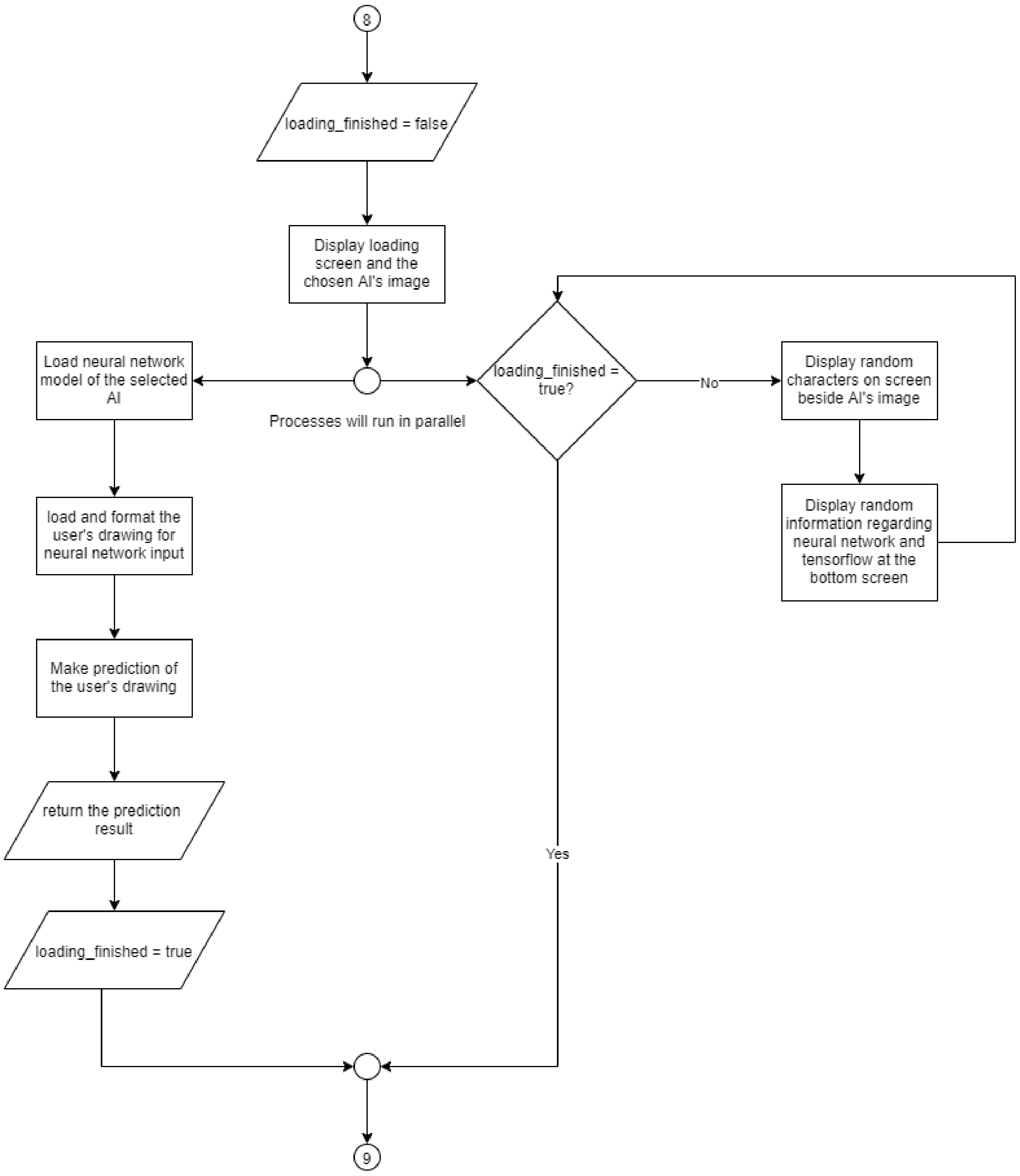

4
11
Initialize AI buttons with random AIs named "???????"
Display AI selection panel
"Back" button clicked
Yes
5
No
One of the AI buttons is clicked
No
Yes
It was initially selected
No
Select it and add highlight. Disable every other button
Yes
Disable selected AI button and remove highlight
Disable "Choose" button
Enable "Choose" button
"Choose" button clicked
No
Yes
10

Draw Panel and Draw Panel (Random) (Left and right respectively):



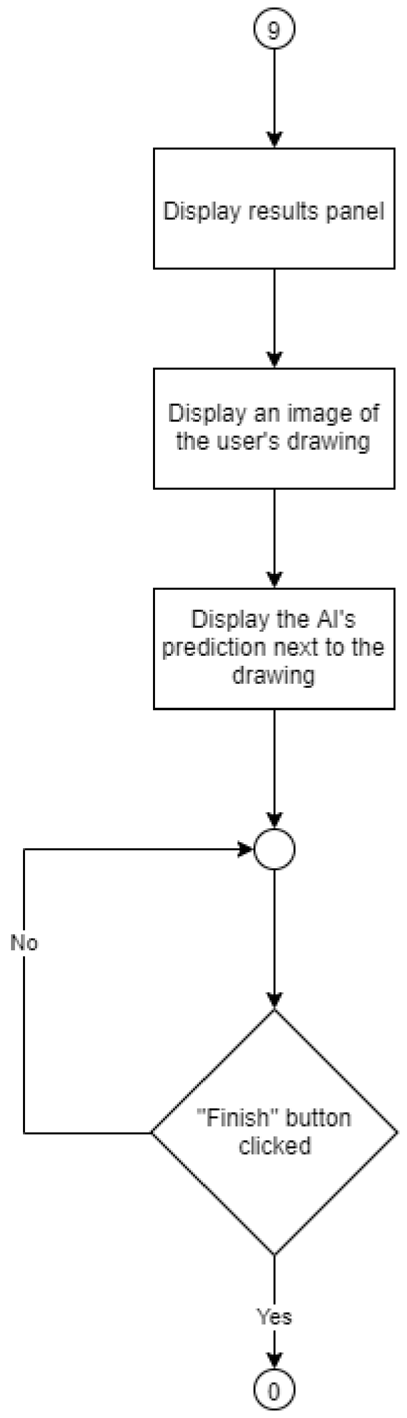


Loading Screen:



8
loading_finished = false
Display loading screen and the chosen AI's image
Load neural network model of the selected AI
Processes will run in parallel
loading_finished = true?
No
Display random characters on screen beside AI's image
Display random information regarding neural network and tensorflow at the bottom screen
load and format the user's drawing for neural network input
Make prediction of the user's drawing
return the prediction result
loading_finished = true
Yes
9

Results Panel:


9
Display results panel
Display an image of the user's drawing
Display the AI's prediction next to the drawing
No
"Finish" button clicked
Yes
0

Game Rules and Other Information

- Rules to play the game:
  1. Play with 2 or more players
  2. Pick an AI to play with. Choose wisely, as some AIs are better than others. Some information regarding the AI is available to help you decide.
  3. Make a drawing of any alphabet character, or any number from 0 to 9. The AI will try to guess what it is.
  4. If the AI guessed it incorrectly, the player who chose the AI and drew gets a punishment. Otherwise, every other player gets the punishment. Punishment can be in any form, such as a round of truth or dare, push-ups, sing, etc.
  5. Then take turns with another player to try their luck with the AI. You cannot draw the same character with the same AI.
  6. Have fun :)
- Available AIs
  - Cintra - 01000010
  - Echo - 01100001
  - Glados - 01100111
  - House - 01110101
  - Talos - 01110011

Implementation

<u>Neural Network</u>

The Basic Idea:

The character recognition AI used in the game is created using the Keras API, which comes with the Tensorflow library. In order to detect the pattern of the drawing anywhere on the image, a deep convolutional neural network is used. The basic idea is that the image is dissected into a number of pixels that gets smaller and smaller. For example, if an image had the dimensions 128x128 pixels, the neural network will divide into sections of, let's say, 4x4 pixels. It will then narrow it down by checking an area of 3x3 pixels and then 2x2 pixels from that 4x4 pixels area. In this way, the AI can look for the necessary patterns at every stage. To that effect, convolutional neural networks usually takes a much longer time to train than a simple neural network.

For the case of this project, all of the AIs have the following general structure:

- Input layer
- Convolutional layer
- Hidden layer
- Output layer

The input layer receives a series of images of size 128x128 pixels. It is directly linked to the convolutional layer as it is a Conv2D layer already. As input images go through the next series of Conv2D layers in the convolutional layer, it will then reach the MaxPooling2D and Flatten layers. These 2 layers provides the bridge between the convolutional and hidden layer of the neural network. The MaxPooling2D layer job, generally speaking, is to take the max (or most apparent feature) of each 2x2 pixel block of the image. The Flatten layer, on the other hand, converts the 2 dimensional image into a 1 dimensional input. These layers were decidedly used as it is the suggested structure provided by Keras.

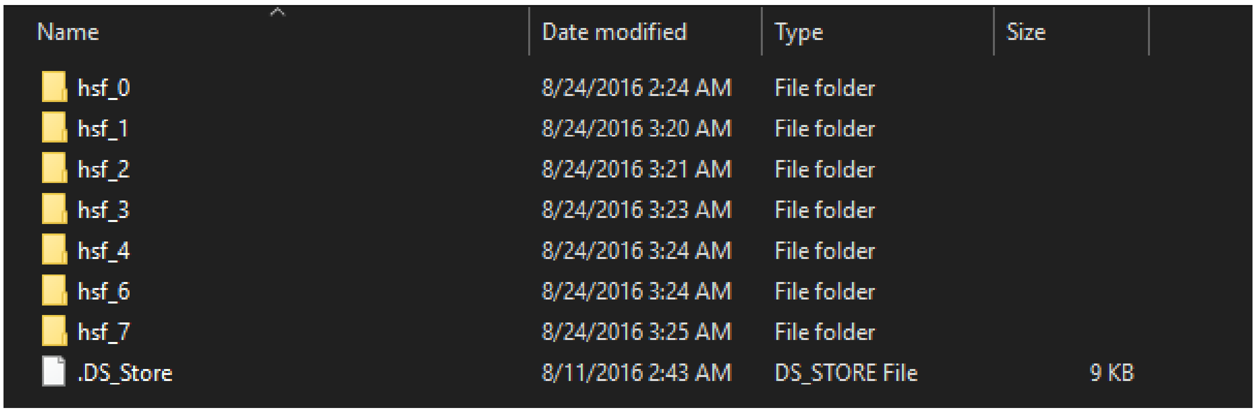
Moving on, the hidden layer is where the flattened images are processed before being fed to the output layer. They are made of Dense layers, though the amount of these layers and the amount of neurons in each may vary. The biasing of weights in each neuron happens here.

After being processed, the flattened image are sent to the output layer (which is also a Dense layer). The output layer has a fixed size of 62 neurons which corresponds to each 62 characters in the alphabet (uppercase and lowercase) and the numbers 0 to 9. Each of the neurons returns a floating number between 0.0 and 1.0. These numbers represents how much the neural network is certain the input image corresponds to what character. The function argmax() can be called after that to get the result with the highest probability.

A small note to pay attention to is that a Dropout layer may be placed in the convolutional and/or hidden layers. The function of Dropout layers is to try to distribute the learning process across all of the available neurons so that the weights distribution will be balanced and there won't be one neuron that does most of the heavy lifting. This is achieved by "dropping out" the outputs of several neurons (basically ignoring them). The idea might sound crazy, but it has been proven to work quite effectively.

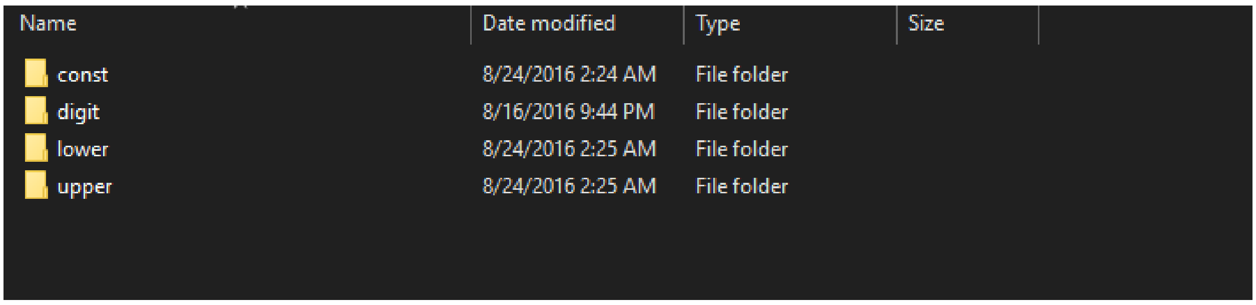
Training the Neural Network:

Each and every AI used in the game was trained from the same source: NIST Special Database - 19 (https://www.nist.gov/srd/nist-special-database-19). This database contains over 810,000 images that isolates each character. For the purpose of this project, the "by_field" file (available for download from the previous link) is used as it separates those 810,000 images into groups, with each group containing varieties of each character. In this way, the computer's RAM will not be overloaded when loading all of the images. For added context, here is the content of the download file:



Name	Date modified	Type	Size
hsf_0	8/24/2016 2:24 AM	File folder	
hsf_1	8/24/2016 3:20 AM	File folder	
hsf_2	8/24/2016 3:21 AM	File folder	
hsf_3	8/24/2016 3:23 AM	File folder	
hsf_4	8/24/2016 3:24 AM	File folder	
hsf_6	8/24/2016 3:24 AM	File folder	
hsf_7	8/24/2016 3:25 AM	File folder	
.DS_Store	8/11/2016 2:43 AM	DS_STORE File	9 KB

And each of these folders contain the following subfolders:



Name	Date modified	Type	Size
const	8/24/2016 2:24 AM	File folder	
digit	8/16/2016 9:44 PM	File folder	
lower	8/24/2016 2:25 AM	File folder	
upper	8/24/2016 2:25 AM	File folder	

For this project, all of the images in these subfolders were used for training, except for the images in the "const" folder.

The following function was created to load the images:

```
def load_nist_database(dataset_num = 1, data_size:float = 1.):
    """Get dictionary that maps the images for training"""
    #dataset_num = hsf folder number
    #data_size is the amount of data to be loaded, between 0 and 1. 1 
means all of the data.
    images:{} = {}

    #Checking for invalid arguments being parsed
    if data_size > 1.:
        data_size = 1.
    elif data_size < 0.:
        data_size = 0.

    #Loading the images
    groups:() = ("digit", "upper", "lower") #The subfolders to scan
    for group in groups:
        for hexa in constants.ascii_hex:
            files = glob.glob(constants.nist_database_location + 'hsf_' 
+ str(dataset_num) + "/" + group + "/" + hexa + "/*.png")
            for i in range(int(len(files) * data_size)):
                filename = files[i]
                image = image = load_image(filename)
                key = bytearray.fromhex(hexa).decode()  #Convert from 
hexa to ascii
                if images.get(key) != None:    #If key exists in 
dictionary
                    images[key].append(image)
                else:
                    images[key] = [image]
#                 print('images/nist_database/hsf_' + str(dataset_num) + 
"/" + group + "/" + hexa + "/" + filename, "loaded.")

    return images
```
This function returns a dictionary of the images that will be used for training. The images have been separated to their corresponding characters. So for example, the images of the character “A” will be stored in a list under the “A” key in the dictionary.

Next, the image dictionary is separated into two arrays: A list containing the images and a numpy array containing the dictionary keys that corresponds to each individual image (So there will be multiple “A”s, “B”s, etc. in the numpy array). The following function does the operation:

```
def convert_data_map_to_lists(data_map:{}):
    """Converts the data dictionary to a numpy array of the labels (keys 
as integers) and a numpy array of the images (used to insert into neural 
net model)"""
    raw_images:[] = []
    image_labels:[] = []
    for character in data_map:
        for image in data_map[character]:
            raw_images.append(image)
            image_labels.append(character)

    return raw_images, numpy.array(image_labels)
```

In order to adapt to the input shape of the neural network (which has been decided to be 128x128), the images needs to be shaped accordingly. This is achieved by using the OpenCV library as seen in the code snippet below:

```
def resize_image(image, width:int = constants.image_width, height:int =
constants.image_height):
    """Method to resize image, aspect ratio not preserved"""
    image_resized = cv2.resize(image, (width, height), interpolation =
cv2.INTER_AREA)
    return image_resized

def resize_images(images, width:int = constants.image_width, height:int
= constants.image_height):
    """Method to convert a list of images"""
    for i in range(len(images)):
        image = resize_image(images[i], width, height)
        images[i] = image
    return images
```

Due to the nature of the Keras API, the input images needs to be reshaped into a numpy array in a particular format. The following functions does the necessary reshaping:

```
def convert_to_numpy_arr(images, width, height, color_channels):
    """Method for converting the list of images as a numpy array fit for
neural network input"""
    array = numpy.zeros((len(images), width, height, color_channels))
    for i in range(len(images)):
        array[i,:,:,:] = images[i]

    return array
```

```
def shape_image_for_2d_mlp_input(images, width:int =
constants.image_width, height:int = constants.image_height,
color_channels:int=3):
    """A method to reshape the images to be ready for neural net
input"""
    #Color channels is the amount of possible colors -> grayscale = 1,
colored = 3

    #Sometimes Keras puts color channels first before width and height,
so we need to check for that
    if tensorflow.keras.backend.image_data_format() == 'channels_first':
        reshaped_images = images.reshape(images.shape[0],
color_channels, width, height)
    else:
        reshaped_images = images.reshape(images.shape[0], width, height,
color_channels)

    reshaped_images = reshaped_images.astype('float32')
    reshaped_images /= 255      #Divide by 255 because 255 color scheme,
so that the input will be decimals between 0-1

    return reshaped_images
```

For convenience the previously mentioned functions has been collected into one function:

```
def prepare_images_for_mlp_input(images,
                                 width:int = constants.image_width,
                                 height:int = constants.image_height,
                                 color_channels:() =
constants.color_channels):
    """Prepares the image for neural network input compatible"""
    images = resize_images(images, width, height)   #Resize the images
to a uniform size
    images = convert_to_numpy_arr(images, width, height, color_channels)
#Convert the list to a numpy array
    images = shape_image_for_2d_mlp_input(images, width, height,
color_channels)    #Convert image_array according to keras specification
(color channels first or last), and normalize the images to be between 0
and 1
    return images
```

The last operation that needs to be done on the input data is to format the data labels (the numpy array that contains the dictionary keys earlier). Keras does not natively support alphabet characters; it can only read numbers. Hence, we need to convert our array of characters into an array of numbers. However, it is not the simple case of simply converting the alphabet characters into numbers (like assigning the letter “A” to the number 11). Earlier on it was explained that neural networks outputs a list of probabilities (between 0.0 and 1.0) from its neurons in the output layer. As a result, we would need to shape the input to follow suit. The technique is called “one-hot format”. What it does is converting, let’s say the letter “A”, into the number 1 in a position relative to all the possible character outputs. Here is an example:

Alphabet:

A

List of possible character outputs:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F, G, …, z]

One-hot format:

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, …., 0]

With this format, the neural network will be able to map which of its neuron in the output layer corresponds to what character. The two functions below does the heavy lifting for the required task:

```
def convert_labels_to_index_correspondence(labels):
    """Method to convert the characters of the sorted labels to 
correspond to the index of the char_list in constants"""
    new_labels:[] = []
    i = -1
    current_char = ""
    prev_char = ""
    for a in range(len(labels)):
        current_char = labels[a]
        if current_char != prev_char:
            i+=1
        prev_char = labels[a]
        new_labels.append(str(i))
    return new_labels
```

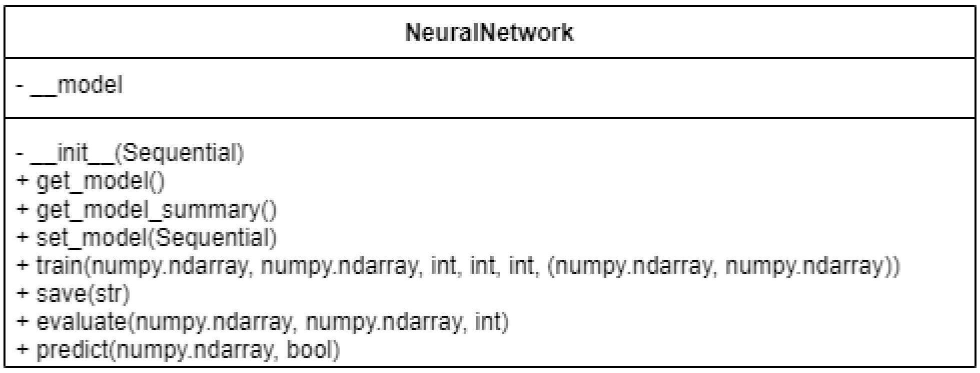
```
def convert_labels_to_one_hot(labels, categories:int):
    """Convert the labels to one-hot format (so that we know what 
character it is)"""
    new_labels:[] = convert_labels_to_index_correspondence(labels)
    return tensorflow.keras.utils.to_categorical(new_labels, categories)
```

It might have looked like there was a lot of functions and operations that was called and executed to format the input data. In order to make life a lot easier, a single function was created to do the converting and formatting of the input image and its corresponding labels:

```
def get_image_and_label_for_mlp_input(image_map:{}, width:int = 
constants.image_width, height:int = constants.image_height, 
color_channels = constants.color_channels):
    """Method to automatically format input image dictionary to be 
usable for neural network input. Method returns the formatted image as 
4D numpy list and the associated labels"""
    images = image_map

    image_list, image_labels = convert_data_map_to_lists(images) 
#Convert the dictionary to a list, and convert the keys to a list as 
well that matches the size of the image list

```

```
    image_list = prepare_images_for_mlp_input(image_list, width, height,
color_channels)

    image_labels = convert_labels_to_one_hot(image_labels,
len(constants.char_list))    #Convert the labels to one-hot format for
neural network classification

    return image_list, image_labels
```

With the hassle regarding input data over, it is now time to deal with the neural network itself. Although Keras has its own object for the neural network model, another class was created to store the neural network object with functions that are adjusted for this project. Here is the UML class diagram for said class:



```
NeuralNetwork
- __model
- __init__(Sequential)
+ get_model()
+ get_model_summary()
+ set_model(Sequential)
+ train(numpy.ndarray, numpy.ndarray, int, int, int, (numpy.ndarray, numpy.ndarray))
+ save(str)
+ evaluate(numpy.ndarray, numpy.ndarray, int)
+ predict(numpy.ndarray, bool)
```

To train the neural network, one can simply call the train() function. It contains the following code:

```
 def train(self,
              train_images,
              train_labels,
              batch_size:int = 100,
              epochs:int = 10,
              verbose:int = 2,
              validation_data = None):
        """
        Train the neural network model
        train_images: the images to train with
        train_labels: the image labels to train with
        batch_size: amount of images to train with at one given time
        epochs: training iterations to do
        verbose: verbose mode. (0=silent, 1=minimal, 2=every batch)
        validation_data: the data used to validate the neural network
model
        """
        return self.get_model().fit(train_images, train_labels,
                                      batch_size = batch_size,
                                     epochs = epochs,
                                     verbose = verbose,
                                     validation_data = validation_data)
```

This function simply calls the function to train the neural network model that is built-in from Keras. The training process might a long while, especially when it is running purely on CPU and not GPU.

The next important step is to actually save the trained neural network model. That way, we won't need to retrain the model again. The save() function in the NeuralNetwork class does this by calling the save() function from Keras.

```
 def save(self, filename):
        """save the current state of the model as a file so that it can
be loaded in the future"""
        self.get_model().save("models/" + filename + ".h5")
```

For this project, and due to the fact that training a single model takes a long time, a function is created to automate the training and saving procedure. This function can be accessed in the "methods" module.

```
def train_neural_network(neural_network:NeuralNetwork,
                         loaded_model:int,
                         iterations:int,
                         width:int = constants.image_width,
                         height:int = constants.image_height,
                         color_channels:() = constants.color_channels):
    """
    Trains the selected neural network with default settings

    neural_network: the NeuralNetwork class object
    loaded_model: the currently loaded model number
    iterations: amount of neural network model files to create
    width: width of the input image
    height: height of the input image
    color_channels: amount of color channels in the input image
    """
    #Load training and test images
    train_images, train_labels =
get_image_and_label_for_mlp_input(file_operation.load_nist_database(1),
width, height, color_channels)
    test_images, test_labels =
get_image_and_label_for_mlp_input(file_operation.load_nist_database(2,
0.3), width, height, color_channels)

    #Train neural network
    for i in range(loaded_model+1, loaded_model + 1 + iterations):
        neural_network.train(train_images,
                             train_labels,
                             batch_size=64,
                             epochs=3,
                             verbose=1,
                             validation_data=(test_images, test_labels))
        neural_network.save("model_" + str(i))  #Save the neural network
        neural_network =
NeuralNetwork(file_operation.load_training_model("model_" + str(i)))
#Load it for next training iteration
```

With this function, a neural network model file will be generated after every 3 training iterations. This would prevent the computer's hard drive space to be used up too quickly (as each model file might go over 1 GB), all the while minimizing the chances of overfitting. After being saved, the model is loaded again and used for further training. The code to load the model file is simply:

```
def load_training_model(filename:str):
    """A method to load the training CNN model"""
    return load_model("models/" + filename + ".h5")

def load_model(path:str):
    """
    Method to laod the desired neural network model
    path: the path of the model file, complete with .h5 extension
    """
    return tensorflow.keras.models.load_model(path)
```

Using Neural Network in the Game:

There are five different neural network models that corresponds to the five different AIs present in the game. Below shows the list of AIs and their respective function that forms their neural network structure:

AI Name	Neural Network Structure Function

Cintra - 01000010	`def create_model(input_shape:int, label_count:int):`
	`    """Creates the neural network model"""`
	`    model = Sequential()`
	`    model.add(Conv2D(16, kernel_size=(4, 4),`
	`activation='relu', input_shape=input_shape))`
	`    model.add(Conv2D(32, kernel_size=(3, 3),`
	`                     activation='relu'))`
	`    # 64 3x3 kernels`

```
    model.add(Conv2D(64, (3, 3), activation='relu'))
    # Reduce by taking the max of each 2x2 block
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # Dropout to avoid overfitting
    model.add(Dropout(0.25))
    # Flatten the results to one dimension for passing into
our final layer
    model.add(Flatten())
    # A hidden layer to learn with
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    # Another dropout
    model.add(Dropout(0.5))
    # Final categorization 0-9, A-z with softmax
    model.add(Dense(label_count, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

    return model
```

Echo - 01100001

```
def create_model(input_shape:int, label_count:int):
    """Creates the neural network model"""
    model = Sequential()
    model.add(Conv2D(8, kernel_size=(5, 5),
activation='relu', input_shape=input_shape))
    model.add(Conv2D(16, kernel_size=(4, 4)))
    model.add(Conv2D(32, kernel_size=(3, 3),
                     activation='relu'))
    # 64 3x3 kernels
    model.add(Conv2D(64, (3, 3), activation='relu'))
    # Reduce by taking the max of each 2x2 block
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # Dropout to avoid overfitting
    model.add(Dropout(0.25))
    # Flatten the results to one dimension for passing into
our final layer
    model.add(Flatten())
    # A hidden layer to learn with
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dropout(0.5))
```

	model.add(Dense(32, activation='relu'))
	    # Another dropout
	    model.add(Dropout(0.5))
	    # Final categorization 0-9, A-z with softmax
	    model.add(Dense(label_count, activation='softmax'))
	
	    model.compile(loss='categorical_crossentropy',
	optimizer='adam', metrics=['accuracy'])
	
	    return model

Glados -
01100111
	def create_model(input_shape:int, label_count:int):
	    """Creates the neural network model"""
	    model = Sequential()
	    model.add(Conv2D(32, kernel_size=(3, 3),
	                     activation='relu',
	                     activation='relu',
	input_shape=input_shape))
	    # 64 3x3 kernels
	    model.add(Conv2D(64, (3, 3), activation='relu'))
	    # Reduce by taking the max of each 2x2 block
	    model.add(MaxPooling2D(pool_size=(2, 2)))
	    # Dropout to avoid overfitting
	    model.add(Dropout(0.25))
	    # Flatten the results to one dimension for passing into
	our final layer
	    model.add(Flatten())
	    # A hidden layer to learn with
	    model.add(Dense(1024, activation='relu'))
	    model.add(Dense(1024, activation='relu'))
	    model.add(Dense(1024, activation='relu'))
	    model.add(Dense(1024, activation='relu'))
	    model.add(Dense(512, activation='relu'))
	    model.add(Dense(128, activation='relu'))
	    model.add(Dense(64, activation='relu'))
	    # Another dropout
	    model.add(Dropout(0.5))
	    # Final categorization 0-9, A-z with softmax
	    model.add(Dense(label_count, activation='softmax'))
	
	    model.compile(loss='categorical_crossentropy',
	optimizer='adam', metrics=['accuracy'])
	
	    return model

House - 01110101	def create_model(input_shape:int, label_count:int):

```
    """Creates the neural network model"""
    model = Sequential()
    model.add(Conv2D(16, kernel_size=(4, 4),
activation='relu', input_shape=input_shape))
    model.add(Conv2D(16, kernel_size=(4, 4),
activation='relu'))
    model.add(Conv2D(32, kernel_size=(3, 3),
activation='relu'))
    # 64 3x3 kernels
    model.add(Conv2D(64, (3, 3), activation='relu'))
    # Reduce by taking the max of each 2x2 block
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # Dropout to avoid overfitting
    model.add(Dropout(0.25))
    # Flatten the results to one dimension for passing into
our final layer
    model.add(Flatten())
    # A hidden layer to learn with
    model.add(Dense(512, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(64, activation='relu'))
    model.add(Dense(64, activation='relu'))
    # Another dropout
    model.add(Dropout(0.5))
    # Final categorization 0-9, A-z with softmax
    model.add(Dense(label_count, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

    return model
```

Talos - 01110011

```
def create_model(input_shape:int, label_count:int):
    """Creates the neural network model"""
    model = Sequential()
    model.add(Conv2D(16, kernel_size=(4, 4),
activation='relu', input_shape=input_shape))
    model.add(Conv2D(16, kernel_size=(4,
4),activation='relu'))
    model.add(Conv2D(32, kernel_size=(3,
3),activation='relu'))
    model.add(Conv2D(64, kernel_size=(3,
3),activation='relu'))
```

```
    # 64 3x3 kernels
    model.add(Conv2D(128, (3, 3), activation='relu'))
    # Reduce by taking the max of each 2x2 block
    model.add(MaxPooling2D(pool_size=(2, 2)))
    # Dropout to avoid overfitting
    model.add(Dropout(0.25))
    # Flatten the results to one dimension for passing into
our final layer
    model.add(Flatten())
    # A hidden layer to learn with
    model.add(Dense(1024, activation='relu'))
    model.add(Dense(512, activation='relu'))
    # Another dropout
    model.add(Dropout(0.5))
    # Final categorization 0-9, A-z with softmax
    model.add(Dense(label_count, activation='softmax'))

    model.compile(loss='categorical_crossentropy',
optimizer='adam', metrics=['accuracy'])

    return model
```

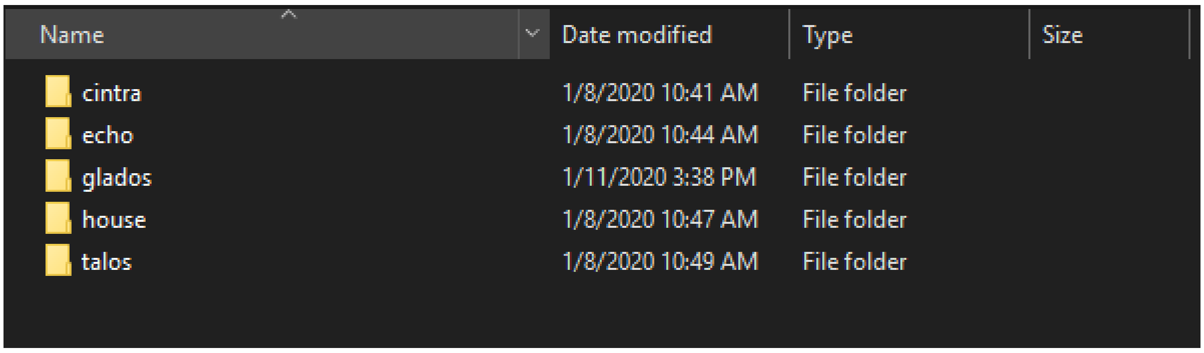
Some additional information regarding the five AIs are also available:

AI Names	Data Samples	Training time (Hours)	Model Size (GB)
Cintra - 01000010	74,758	4.0	2.8
Echo - 01100001	74,758	4.0	2.6
Glados - 01100111	37,379	13.5	3.0
House - 01110101	74,758	5.0	1.3
Talos - 01110011	74,758	4.0	4.9

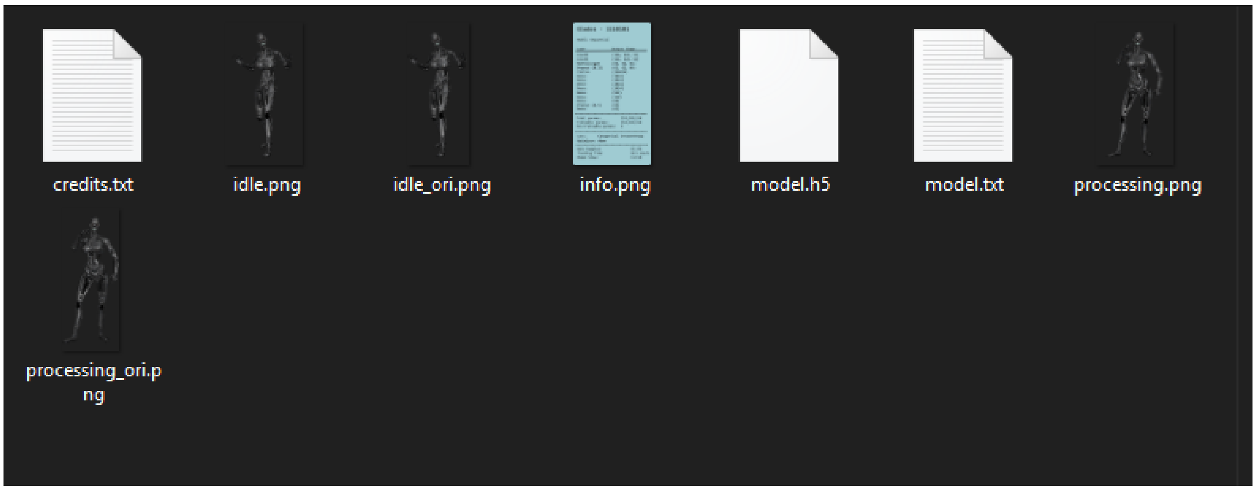
With these variations in model structure, coupled with different training data samples and training time, produces neural network model with variable accuracy. Some neural network models are better than others in guessing certain characters. To boot, more training time or data samples does not equal a better neural network (as some players might soon find out when they play). For example (spoiler alert!), Glados was trained for 13.5 hours with half the amount of training data samples compared to Echo who trained for 4 hours. Despite these differences, Glados is objectively a better AI than Echo. On the other hand, Cintra also trained for 4 hours with the same amount of samples as Echo, however scored better accuracy rating than Glados.

As previously mentioned, there are two game modes currently available in the game: Fun House and Random Chaos. In the former, all of these information (except for those that are too much of a spoiler, i.e. which AI is better) are available for the players to view. When the player finishes drawing their character, the game locates their selected AI's folder and load its associated files. This includes its neural network model file.

All files involving each AI is stored in its own folder.

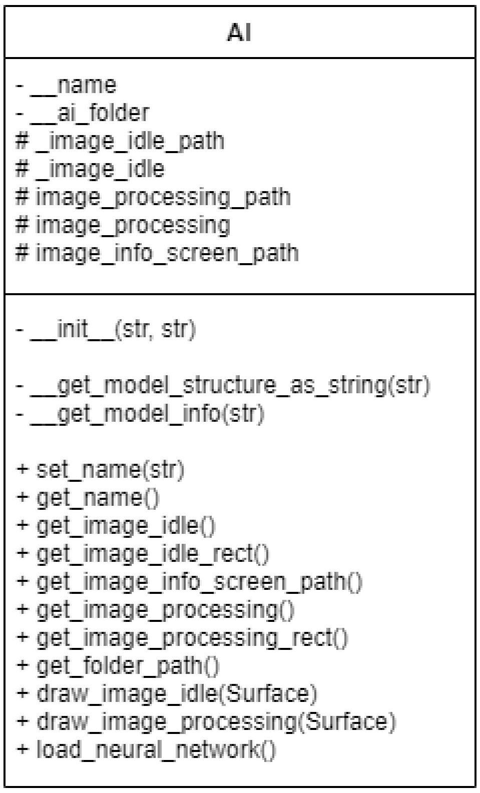


Name	Date modified	Type	Size
cintra	1/8/2020 10:41 AM	File folder	
echo	1/8/2020 10:44 AM	File folder	
glados	1/11/2020 3:38 PM	File folder	
house	1/8/2020 10:47 AM	File folder	
talos	1/8/2020 10:49 AM	File folder	



credits.txt idle.png idle_ori.png info.png model.h5 model.txt processing.png processing_ori.png

To actually store all of this information in code, the AI class is created:



In the game, the AI class is used in the AI selection panel, and the selected AI (alongside the relevant AI class instance object) is carried over all the way until the results panel. For a better user experience, the neural network model of the AI itself does not get loaded until the game reaches the loading screen. Further details regarding this will be explained in the next section.

The Game

The product of this project is a game, hence the Pygame library is used as it has all the necessary classes and functions in order to create a functioning game.

For creating the game for this project, there are four main file groups:

1. Main modules
2. GUI modules
3. Data drivers
4. Subsidiary modules

Main modules:

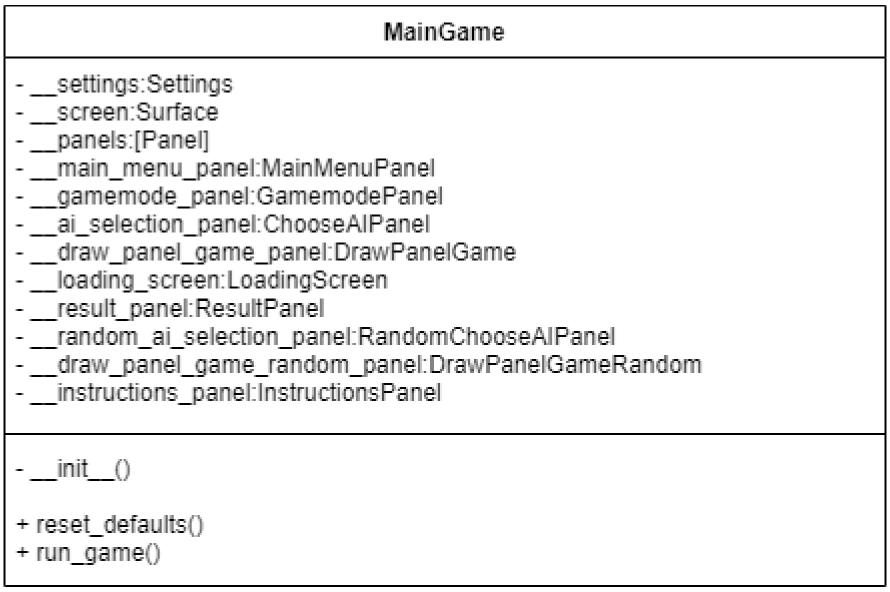
The purpose of these modules is to be the starting point of the game. The necessary game components is to be initialized here, alongside managing how to actually render the game. There are two main modules in this project:

- main.py
- main_game.py

main.py is the starting point when running the game; should one try to run the game, they should run it from this file. To that effect, main.py only has the following lines of code:

```python
from final_project.handwritting_recognition.pygame.main_game import
MainGame

game = MainGame()
game.run_game()
```

main_game.py on the other hand, is where the game is initialized and rendered. Here is an overview of the MainGame class:



```
MainGame
- __settings:Settings
- __screen:Surface
- __panels:[Panel]
- __main_menu_panel:MainMenuPanel
- __gamemode_panel:GamemodePanel
- __ai_selection_panel:ChooseAIPanel
- __draw_panel_game_panel:DrawPanelGame
- __loading_screen:LoadingScreen
- __result_panel:ResultPanel
- __random_ai_selection_panel:RandomChooseAIPanel
- __draw_panel_game_random_panel:DrawPanelGameRandom
- __instructions_panel:InstructionsPanel

- __init__()

+ reset_defaults()
+ run_game()
```

The __panels field is a list that stores the fields __main_menu_panel all the way down to __instructions_panel. The game operates by utilizing the __panels field in order to know which panel to draw and render.

Speaking of drawing and rendering, the function that handles such operation is the run_game() function. Below is the code snippet for the function:

```python
from final_project.handwritting_recognition.pygame import globals
import final_project.handwritting_recognition.pygame.game_functions as
gf

    def run_game(self):
        """Method to run the game"""
        #While game runs
        while True:
            gf.check_events(self.__panels[globals.panel_index])
            gf.update_screen(self.__screen, self.__settings,
self.__panels[globals.panel_index])
```

```
            #Check for reset ping
            if globals.reset_defaults:
                self.reset_defaults()
```

When this function is called, it will continuously run the game in a while loop. The game_functions module (abbreviated to gf) is one of the subsidiary modules (alongside globals). The gf.check_events() function call checks for any events such as mouse and keyboard presses. At its core, this function checks if the user presses the "X" button on the top window which would close the game and terminate the python process. It also checks for additional events that are specified by the individual panels. Pay attention to the check_events() function that is detailed below:

```
def check_events(panel:Panel):
    """Method to check any keyboard or mouse events"""
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()  #Stops game
        panel.check_events(event)   #Check for events in the current
panel
```

Notice how it can only check for events in one panel, as opposed to every panel in the game. This is by choice, as it would be terribly unnecessary to check for every single event in every single section of the game. The game only needs to check for events in the currently active panel (the one currently being rendered and shown). As an effect, there needs to be a variable to keep track of which panel is currently active. That's where the panel_index variable from the globals module comes in. This variable is nothing more than a simple integer value that is used to access the __panel list. Only the panel in the current index is being checked for events.

```
gf.check_events(self.__panels[globals.panel_index])
```

The same thing is used in the gf.update_screen() function call, which is a function used to render and display the game screen: only render and display the currently active panel. This function is simple:

```
def update_screen(screen, settings:Settings, panel:Panel):
    """Method to update the game\'s screen"""
    screen.fill(settings.background_color)
    panel.draw_components()
```

It will just fill the screen with the background color (which is controlled by the Settings class in the settings.py module), and then draw the currently active panel alongside its components.

Still talking about the run_game() function, it does one last important operation: resetting components to defaults.

```
#Check for reset ping
if globals.reset_defaults:
    self.reset_defaults()
```

```
    def reset_defaults(self):
        """Method to reset defaults of the game components"""
        #Reset panels
        for panel in self.__panels:
            panel.reset_defaults()

        #Reset global variables
        globals.mouse_left_pressed = False
        globals.active_ai = None
        globals.loading_active = False
        globals.show_results = False
        globals.loading_progress = ""

        globals.reset_defaults = False  #turn off identifier
```

The purpose of this function is to simply return all values, fields, and components to their original state, to how they were initially when starting the game for the first time. As it can be observed, this operation is not executed every time; it waits for the globals.reset_defaults flag to be True. Once the operation is complete, that flag is returned back to False. This global variable is only turned True at particular instances in the game, which will be discussed in the next section.

GUI Modules:

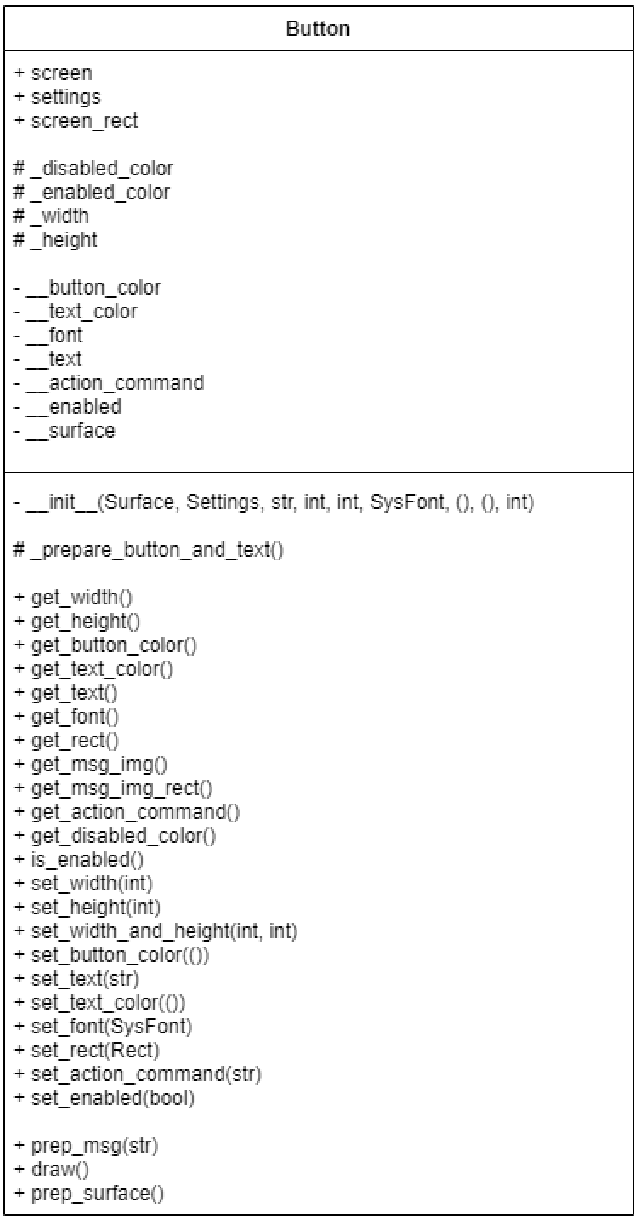
Graphical User Interface (GUI) is a key component of the game. It controls what the user sees and aims to give a better user interface compared to using a command-line interface (CLI). While Pygame has provided with some basic (but crucial) GUI elements, such as a game window, displaying text and images, and rendering shapes, some additional GUI components are required to be manually created. There are four general GUI objects that has been created for this project:

- Button
- Image Panel

- Animation
- Panel

These custom made objects utilize some of the core functionalities provided by Pygame, while also adding some of its own.

Starting with buttons, this essential GUI component is used to navigate the program and execute instructions, but seems to be missing in Pygame. Hence, one will need to create them. Here is an overview of the Button class:



This class, in essence, draws a Rect object available from Pygame on to the screen. It will then blit the text on to the Rect object, simulating a button. An instance of Pygame's Surface is also required, as to allow the background color of the button to be drawn with alpha values (allowing transparency). The surface is drawn on top of the Rect object. Below are the code snippets for this operation.

```
def prep_msg(self, msg):
        self.__msgImg = self.__font.render(msg, True, self.__text_color, None)
        self.__msgImgRect = self.__msgImg.get_rect()
        self.__msgImgRect.center = self.__rect.center
```
A Rect object of the drawn text is created and placed in the center of the button's Rect object.

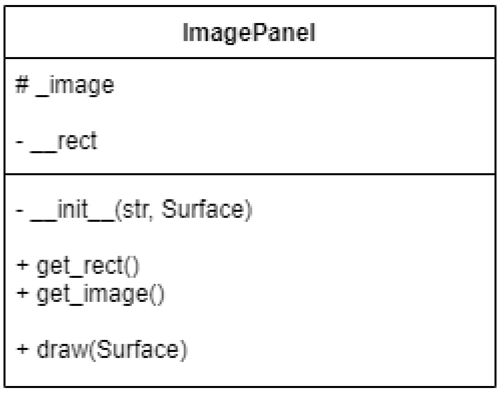
```
def prep_surface(self):
        self.__surface = Surface(self.__rect.size, pygame.SRCALPHA)
        self.__surface.fill(self.__button_color)
```
The Surface object is created with the same size of the button's Rect object. It will then color itself with the same color of the button.

```
def draw(self):
        #Draw blank button and then draw message
#         self.screen.fill(self.__button_color, self.__rect)
        self.screen.blit(self.__surface, self.__rect)
        self.screen.blit(self.__msgImg, self.__msgImgRect)
```
The objects are drawn using the above function. Notice that the button's Rect object doesn't actually get drawn. It is only used for positioning purposes of the Surface and text label.

A key feature of this Button class is that it has an enabled/disabled functionality. It uses the _enabled_color and _disabled_color fields to show user whether the button is enabled or disabled. This is controlled in the set_enabled() function detailed below:

```
def set_enabled(self, b:bool):
        """Enable or disable the button"""
        self.__enabled = b

        if (b): #enabled
            self.__button_color = self._enabled_color
        else:   #disabled
            self.__button_color = self.get_disabled_color()
        #recreate surface color and text
        self.prep_surface()
        self.prep_msg(self.get_text())
```
Notice how the class does not have any instructions or function to handle mouse clicks or what to do when the button is disabled/enabled other than changing its color. That is because this class only serves as a blueprint to create a button alongside being able to draw it on the screen, but nothing more. That is handled separately, in the Panel class.

While buttons draws a rectangle with a text label, Image Panels are used to display an image on the screen. Although Pygame already has this functionality, the ImagePanel class makes the process simpler and easier. A UML class diagram for the ImagePanel class is provided below:



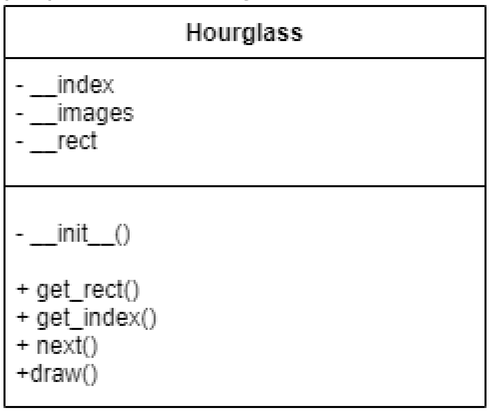
This is just a simple class, with its main feature being able to retain the original position of the image's Rect after the current image is replaced with another image. This is achieved in the following function in the class:

```
def set_image(self, path:str = None, image = None):
        """
        Fill one of the following parameters:
        path: path of the image
        image: the image itself in pygame.image.load() format
        """
        try:
            self.image = pygame.image.load(path) if path != None else
image

            #Retain some of the original rect positions
            rect:Rect = self.__rect
            self.__rect = self.image.get_rect()
            if rect != None:
                self.__rect.center = rect.center
        except:
            pass
```

So after a new image is specified, the settings from the old Rect is used. Note that the Rect object in this class is different from the Rect object of the image itself. In this class the Rect object instance is only used to position the image on the screen. It does not get drawn, as evident in the draw() function:

```
def draw(self, screen):
        """
        Method to draw the image to the screen

        screen: the pygame screen
        """
        screen.blit(self.image, self.get_rect())
```

Animation is similar to images that it is a set of images being displayed at different intervals in time. Pygame does not natively support animation, so the algorithm for displaying animation needs to be developed. For this project, animation is used in the Loading Screen panel, specifically in displaying an animation involving an hourglass being turned. For that purpose, the Hourglass class is created.



The __images is a list that holds all of the images for the animation. The hourglass images are loaded as such:

```python
from final_project.handwritting_recognition import file_operation

self.__images:[] = file_operation.load_hourglass_images()
```

```python
def load_hourglass_images() -> []:
    """Method to load the hourglass animation images"""
    hourglass:[] = []

    for i in range(10):
        hourglass.append(pygame.image.load("images/hourglass/" + str(i)
+ ".png"))

    return hourglass
```

Meanwhile, the __index field is an int that is used to access the __images list and control what image of the hourglass animation is to be shown. Showing the next image (or frame) of the animation is done by simply calling the next() function, which has the following operation:

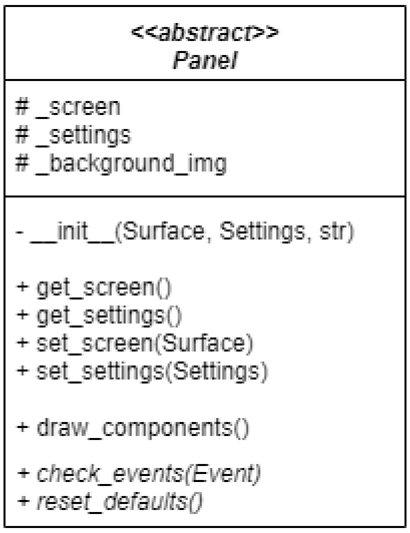
```python
def next(self):
        """Increment the image index of the hourglass image by one"""
        if self.__index + 1 >= len(self.__images):
            self.__index = 0
        else:
            self.__index += 1
```

From the above snippet it can be seen that the __index field will just increment by 1 until the end of the __images list, at which after it would reset itself to 0. Drawing the animation is also as easy as telling which image (by using the __index field) is to be blitted to the screen, with the __rect object serving as position anchor.

```python
def draw(self, screen):
        """Draw the hourglass to the screen"""
        screen.blit(self.__images[self.__index], self.__rect)
```

Similar with the Button object, the Hourglass class is only a shell that provides the much needed function to display the object. It does not manage the interactions with the user or other components of the game. Things such as how long does one frame (one image) of the hourglass takes before moving on to the next image is managed by the Panel objects, which will be discussed next.

Panels are what players see in every section of the game, and yet have no physical form; it is not something that gets drawn to the screen. The Panel class is simply a container that stores all the other GUI objects. However, since each section of the game would have its own different objects and components with their own placements, and in order to flood the main_game.py module with these configurations, the Panel class was designed to be an abstract class.



By abstracting the check_events() and reset_defaults() functions, any child of the Panel class can implement its own way of checking for events and what components needs to be put back to their default state respectively. The draw_components() function on the other hand, is not an abstract function as it needs to draw the _background_img object, though only if a background image is specified.

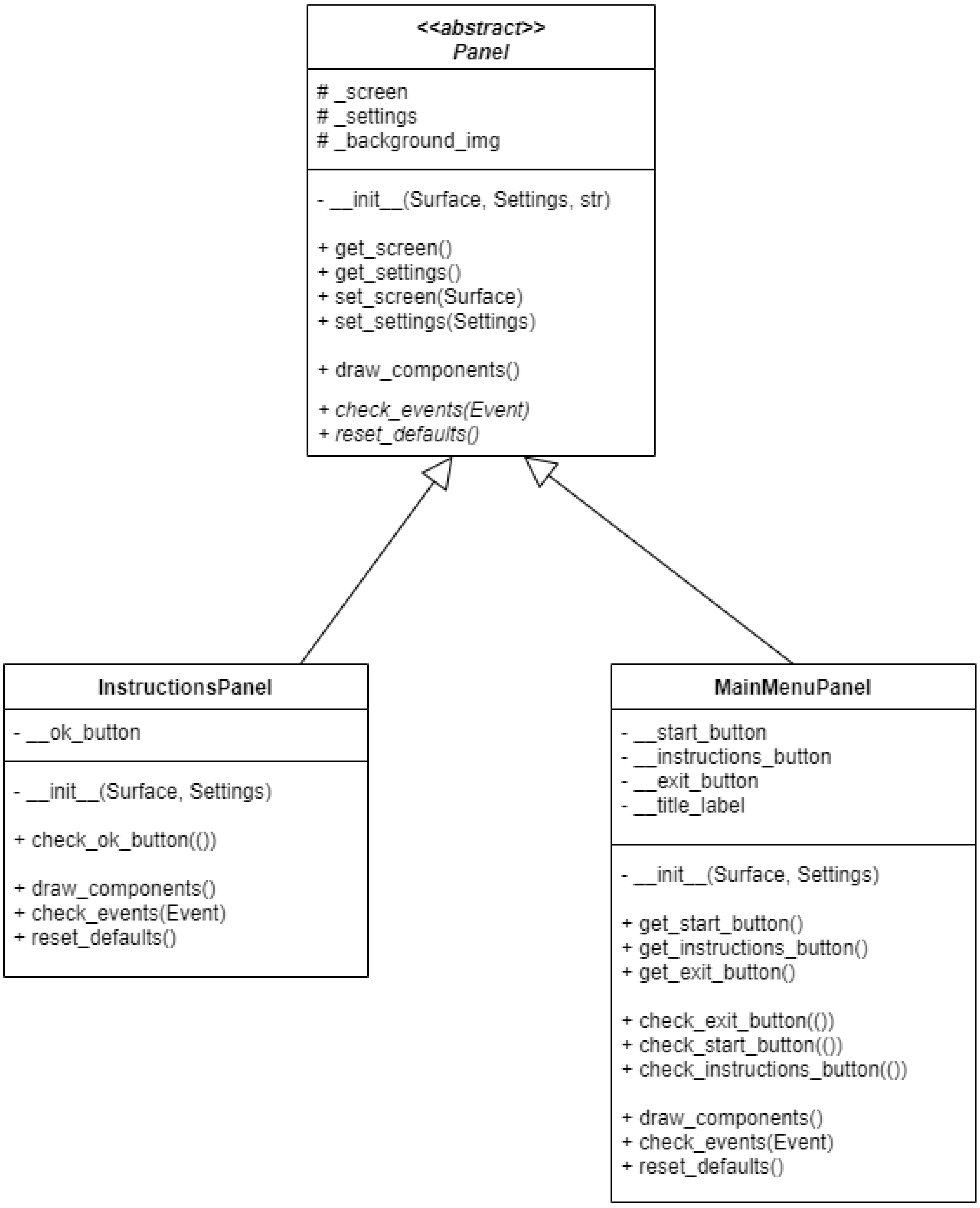
```
def draw_components(self):
        """Method to draw the components of the screen"""
        if (self._background_img != None):
            self._background_img.draw(self._screen)
```

Now since this class is an abstract class, it cannot be instantiated. It depends on inheritance to do so. For this project, the child classes of the Panel class corresponds to the various sections of the game, as such:

- Panel
    - MainMenuPanel
    - GamemodePanel
    - ChooseAIPanel
        - RandomChooseAIPanel
    - DrawPanel
        - DrawPanelGame
            - DrawPanelGameRandom
    - LoadingScreen
    - ResultPanel
    - InstructionsPanel

Above shows the flow of inheritance of the game panels.

The and MainMenuPanel and InstructionsPanel class is a great example to show the benefits of abstraction. Below outlines the two classes in a UML diagram:



Now even though both classes inherits from the Panel class and thus have the check_events() and reset_defaults() functions, they implement them differently. For example in the InstructionsPanel class:

```
def check_events(self, event:Event):
        x, y = game_functions.get_mouse_position()

        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1:       #1 is left mouse button
```

```
                self.check_ok_button((x, y))
```
It only checks for button press event for its “Ok” button. Meanwhile on the MainMenuPanel class:
```
def check_events(self, event:Event):
        super().check_events(event)

        #Check mouse button press events
        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1:       #1 is left mouse click
                x, y = gf.get_mouse_position()
                self.check_exit_button((x, y))
                self.check_instructions_button((x, y))
                self.check_start_button((x, y))
```
It checks for events in its “Start”, “Instructions”, and “Exit” buttons which are not present in the other’s class.

Speaking of buttons, the check_events() function is how the game is able to check for button presses. In the InstructionsPanel’s check_events() function snippet, the following two lines of code is how the game checks for when the left mouse button is clicked:
```
if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1:       #1 is left mouse button
```
Checking if the button is pressed, however, uses another set of function (still using the InstructionsPanel button as an example):
```
def check_ok_button(self, mouse_pos:()):
        """Check if the ok button was clicked"""
        if game_functions.mouse_on_button(self.__ok_button, mouse_pos):
            globals.panel_index = 0 #Go to main menu
```

```
def mouse_in_area(rect:Rect, mouse_pos:()):
    """Returns True if the rect collides with the mouse pointer location
(if coordinates are the same)"""
    return rect.collidepoint(mouse_pos[0], mouse_pos[1])

def mouse_on_button(button:Button, mouse_pos:()):
    """Check if the mouse pointer is on a button"""
    return mouse_in_area(button.get_rect(), mouse_pos)
```
So if the left mouse button is pressed and its pointer is located on the button, it signifies that the button is pressed, and the associated operation can be conducted (globals.panel_index = 0 in this case). The panel_index variable, to recall, is used to tell the game which panel to check for events and render.

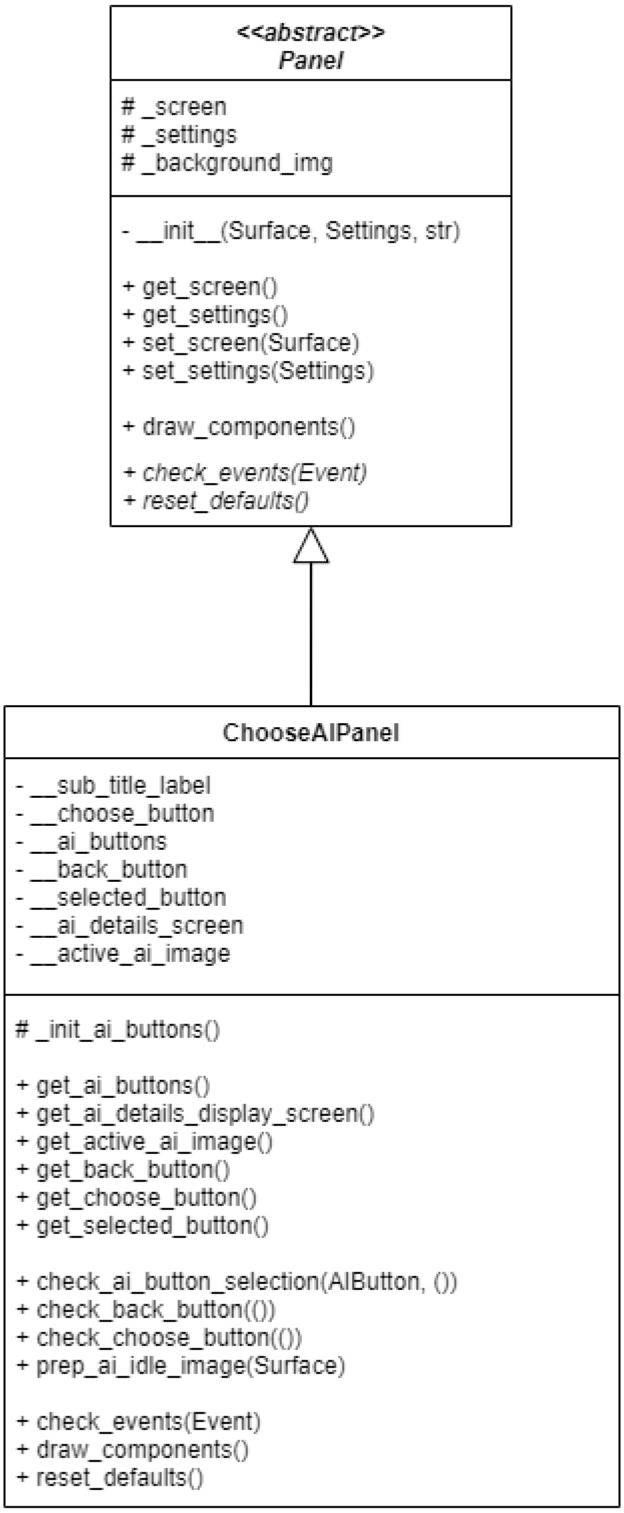
While the MainMenuPanel and InstructionsPanel class are similar that they both handle mouse click events, the GamemodePanel has something extra: it can detect mouse hover operations as well.
```
def check_events(self, event:Event):
        super().check_events(event)
        x, y = gf.get_mouse_position()

        #Check for mouse hovers
        if gf.mouse_on_button(self.get_funhouse_button(), (x, y)):

self.get_console().set_image(constants.path_img_gamemode_funhouse_consol
e)
        elif gf.mouse_on_button(self.get_random_chaos_button(), (x, y)):

self.get_console().set_image(constants.path_img_gamemode_random_chaos_co
nsole)
        else:

self.get_console().set_image(constants.path_img_gamemode_welc_console)

        #Check for mouse presses
        if event.type == pygame.MOUSEBUTTONDOWN:
            if event.button == 1: #1 is left click
                self.check_funhouse_button((x, y))
                self.check_random_chaos_button((x, y))
                self.check_back_button((x, y))
```

In addition to checking for mouse presses, mouse hovers can be checked as well by using the same function used to check if the mouse pointer is on a button.

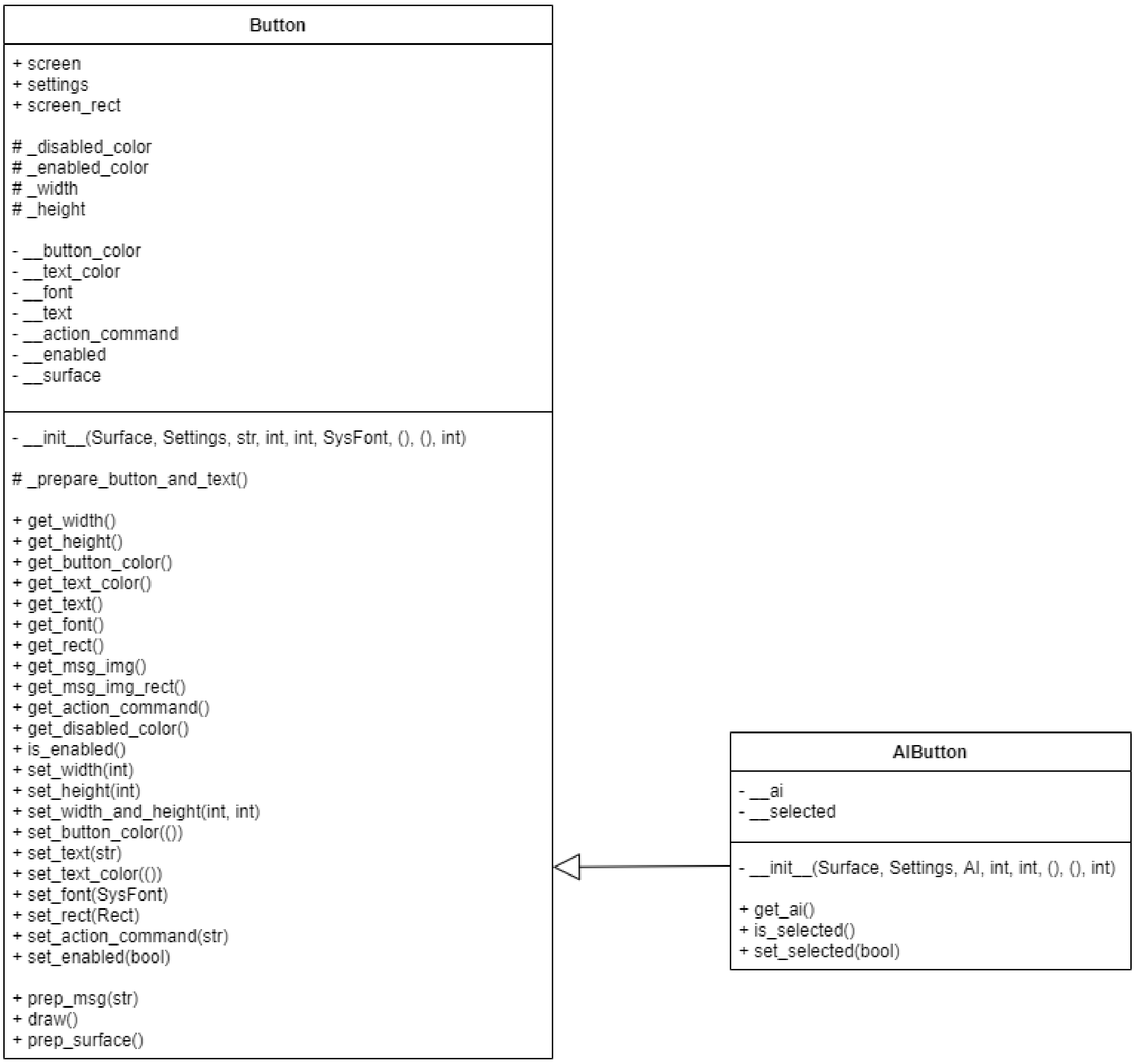
```
#Check for mouse hovers
        if gf.mouse_on_button(self.get_funhouse_button(), (x, y)):

self.get_console().set_image(constants.path_img_gamemode_funhouse_consol
e)
        elif gf.mouse_on_button(self.get_random_chaos_button(), (x, y)):

self.get_console().set_image(constants.path_img_gamemode_random_chaos_co
nsole)
        else:

self.get_console().set_image(constants.path_img_gamemode_welc_console)
```

With gf being game_functions, it calls the same function as shown in the InstructionsPanel example earlier. For this case, the get_console() function returns the ImagePanel object of in the panel. If the mouse hovers over the Fun House button, it will display an image of the funhouse_console image, or if it's over the Random Chaos button, it will display the random_chaos_console image instead. Otherwise, it would display the welc_console image.

The same mouse hover functionality is applied to the next panel, which is the ChooseAIPanel class, but with even more features. To start, here is an overview of the class:



The extra feature with this panel is that in addition to displaying images during mouse hovers, when a button is clicked, the image will remain constant and will not change. However, if the button is then clicked again, the mouse hover action resumes. This feature is attached to the buttons that are used to identify the player's AI selection. The __ai_buttons field is a list that holds a number of AIButton objects. The AIButton class is a child class of Button that has been added with the necessary components for this particular action.



The __ai field stores an instance of AI class. The __selected field is a boolean used to identify whether the button is selected.

Back to the ChooseAIPanel class, the AI selection process is controlled by the following function:

```
def check_ai_button_selection(self, button:AIButton, mouse_pos:()):
        """Method to check for AI selection, when one of the AI buttons
are pressed"""
        if button != None and game_functions.mouse_on_button(button,
mouse_pos):
            self.__selected_button = button

            updated_selection:bool = not
self.__selected_button.is_selected()

            if updated_selection:       #If button is selected, disable
every other button
                for button in self.__ai_buttons:
                    if button != self.__selected_button:
                        button.set_selected(False)

            self.__selected_button.set_selected(updated_selection)

self.get_ai_details_display_screen().set_image(self.__selected_button.ge
t_ai().get_image_info_screen_path())

self.prep_ai_idle_image(self.__selected_button.get_ai().get_image())

            self.__choose_button.set_enabled(updated_selection)
            if updated_selection == False:
                self.__selected_button = None
            else:

self.__selected_button.get_ai().get_image_idle_rect().center = 
self.get_screen().get_rect().center
```

Here is a breakdown of the algorithm:

```
if button != None and game_functions.mouse_on_button(button, mouse_pos):
            self.__selected_button = button

            updated_selection:bool = not 
self.__selected_button.is_selected()
            self.__selected_button.set_selected(updated_selection)
```

If one of the AIButton objects is pressed, set that button to be the active selected button. Then in whatever state that button is, reverse it. So if the button has not been selected, select that button. Otherwise, deselect it.

```
if updated_selection:       #If button is selected, disable every other 
button
                for button in self.__ai_buttons:
                    if button != self.__selected_button:
                        button.set_selected(False)
```

Now if the button is selected, all of the other AIButton objects are deselected. This will ensure that only one AI is chosen.

```
self.get_ai_details_display_screen().set_image(self.__selected_button.ge
t_ai().get_image_info_screen_path())

self.prep_ai_idle_image(self.__selected_button.get_ai().get_image())
```

This will display the images associated with the AI.

```
self.__choose_button.set_enabled(updated_selection)
            if updated_selection == False:
                self.__selected_button = None
            else:
```

```
self.__selected_button.get_ai().get_image_idle_rect().center =
self.get_screen().get_rect().center
```

Now the Choose button needs to be updated. If no AI is selected, it needs to be disabled. If an AI has been selected, enable it. This is done by the code above.

Since this panel involves storing the currently selected AIButton, it would be game-breaking if the user’s choice of AI is retained on the round of the game. This means that the selection will need to be cleared. To achieve this, the reset_defaults() function inherited from the Panel class is utilized.

```
def reset_defaults(self):
        super().reset_defaults()

        #If there's an existing selection, turn it off and set the
selection to None
        if (self.__selected_button != None):
            self.__selected_button.set_selected(False)
            self.__selected_button = None
```

If the Choose button is enabled and the player clicks on it, the game will store the currently selected AI into a global variable so that the next stages of the game can utilize that information.

```
def check_choose_button(self, mouse_pos:()):
        """Method to check when the choose button is pressed"""
        if gf.mouse_on_button(self.__choose_button, mouse_pos) and
self.__choose_button.is_enabled():
            globals.active_ai = self.__selected_button.get_ai()
            globals.panel_index += 1
```

The function above does the needed operation of assigning the currently selected AI to the active_ai global variable and updating the panel_index variable so that the screen can move on to the next panel.

The DrawPanel class is next up the chain. The main feature of this panel is providing the player the ability to draw on the screen. In the class, there is a variable called __colored_pixels which is a set object. It’s purpose is to keep track of the pixels drawn by the player. With it being a set object, there will be no duplicates. The function that handles the drawing operation is detailed below:

```
def draw_components(self):
        super().draw_components()
        #Draw sub title
        if self._draw_sub_title_label:
            self.__sub_title_label.draw(self.get_screen())

        #Draw art
        if not self._guess_button_pressed:
            self.draw_art()
        #Draw buttons
        for button in self._buttons_to_draw:
            button.draw()

        if not self._guess_button_pressed:
            pygame.display.flip()
        else:
            self.get_screen().fill(self.get_settings().background_color)
            self.draw_art()
            self._guess_button_pressed = False
```

```
def draw_art(self):
        """Method to draw the mouse click art"""
        if globals.mouse_left_pressed:
            x, y = gf.get_mouse_position()
            collide_with_button = False
            for button in self._buttons_to_draw:
                collide_with_button = button.get_rect().collidepoint(x,
y)  #Check if the drawing will collide with the buttons
                if collide_with_button:
                    break
            if not collide_with_button and not
self._guess_button_pressed:
                self.__colored_pixels.add((x, y))
        for pixel in self.__colored_pixels:
            pygame.draw.rect(self.get_screen(),
                             self.get_settings().pen_color,
                             Rect(pixel[0],
                                  pixel[1],
                                  self.get_settings().pen_size,
                                  self.get_settings().pen_size))
```

The draw_art() function does the drawing, while calling it in the draw_components() function makes sure that the player's art is displayed on the screen.

```
if globals.mouse_left_pressed:
```

The above code in the draw_art() function ensures that the game will only draw if the left mouse button is pressed.

```
x, y = gf.get_mouse_position()
            collide_with_button = False
            for button in self._buttons_to_draw:
                collide_with_button = button.get_rect().collidepoint(x,
y)  #Check if the drawing will collide with the buttons
                if collide_with_button:
                    break
            if not collide_with_button and not
```

```
self._guess_button_pressed:
                self.__colored_pixels.add((x, y))
```

The above code snippet make sures that the game will only draw on the screen area that will not cover the buttons displayed on the panel. Once the checking is done, the program loops through the __colored_pixel set object in order to display the drawing:

```
for pixel in self.__colored_pixels:
            pygame.draw.rect(self.get_screen(),
                             self.get_settings().pen_color,
                             Rect(pixel[0],
                                  pixel[1],
                                  self.get_settings().pen_size,
                                  self.get_settings().pen_size))
```

Since the player's drawing is to be fed into their chosen AI's neural network, an image of their drawing is necessary. Preferably without the buttons being displayed. This operation is conducted when the user presses the Guess button in the panel and the following function is executed:

```
def check_guess_button(self, mouse_pos:()):
        """Check for guess button presses"""
        if game_functions.mouse_on_button(self.get_guess_button(),
mouse_pos):
            self.set_guess_button_pressed(True)
            #Remove the sub title label, exit, guess, and clear buttons
to capture only the drawn image and then restore it
            self.get_buttons_to_draw().clear()
            self.set_draw_sub_title_label(False)
            self.draw_components()  #Redraw the panel without the label
and buttons, but do not update the screen to prevent flickering
            pygame.image.save(self.get_screen(),
constants.pygame_image_path + constants.pygame_test_image_name)
#Take a screenshot of the game screen window and save it to disk
            self._buttons_to_draw = [self.get_exit_button(),
self.get_guess_button(), self.get_clear_button()] #Add the buttons back
            self.set_draw_sub_title_label(True)   #Draw the sub title
label again

            #Connect to loading screen
            globals.loading_active = True
            globals.panel_index += 1
```

Pay attention to these two lines of code:

```
self.set_guess_button_pressed(True)
```

```
self.draw_components()  #Redraw the panel without the label and buttons,
but do not update the screen to prevent flickering
```

When the Guess button is clicked, the program stores that fact in a boolean. The draw_component() function is called after that. With the program notified of the press, the following lines of code in the draw_component() function comes to live:

```
if not self._guess_button_pressed:
            pygame.display.flip()
        else:
            self.get_screen().fill(self.get_settings().background_color)
            self.draw_art()
            self._guess_button_pressed = False
```

In order for the game to basically take a screenshot of the game screen with only the user's drawing, it fills the entire screen with the background color, and then renders that drawing only. The _guess_button_pressed identifier is then updated, the screenshot taken, and the game goes back to normal.

Before the game moves on to the next panel by way of the code

```
            #Connect to loading screen
            globals.loading_active = True
            globals.panel_index += 1
```

(which is the globals.panel_index += 1 command), the globals.loading_active = True command is necessary in order to actually start the loading progress. The reason for that will be explained next.

After the player presses the Guess button, the game moves on to the loading screen panel, which is an instance of the LoadingScreen class. This class is very unique, as it is the only class to use multithreading. Multithreading is necessary in order to display the hourglass animation (alongside the loading progress text), show a random character, and render a random sentence regarding neural networks. All the while the player's chosen AI's neural network is being loaded and their drawing predicted. In the LoadingScreen class, the threads are first declared like so:

```
self.__thread_progress:Thread = None
self.__thread_ai_image:Thread = None
self.__thread_loading_hints:Thread = None
self.__thread_nn:Thread = None
```

The threads are initialized with None values because they are not supposed to do anything yet when the constructor is called (which is done in the main_game module). It waits until the loading_active global variable to turn True to start doing their magic.

In order to check for the status of that global variable continuously, the panel's draw_component() function is utilized.

```
def draw_components(self):
        super().draw_components()

        if globals.loading_active:
            #Check AI:
            if self.__ai == None:
                self.__ai = globals.active_ai

self.__ai_image.set_image(self.__ai.get_image_processing_path())
```

```
                #Positioning ai image
                self.__ai_image.get_rect().centery = 
self.get_screen().get_rect().centery
                self.__ai_image.get_rect().right = 
self.get_screen().get_rect().centerx - self.__padding

            #Check Thread status
            if self.__thread_loading_hints == None or not 
self.__thread_loading_hints.is_alive():
                self.__thread_loading_hints = 
Thread(target=self.display_loading_hints)
                self.__thread_loading_hints.start()
            if self.__thread_ai_image == None or not 
self.__thread_ai_image.is_alive():
                self.__thread_ai_image = 
Thread(target=self.display_guess)
                self.__thread_ai_image.start()
            if self.__thread_progress == None or not 
self.__thread_progress.is_alive():
                self.__thread_progress = 
Thread(target=self.display_loading_progress)
                self.__thread_progress.start()
            #Check neural network thread
            if self.__thread_nn == None:
                self.__thread_nn = 
Thread(target=self.load_neural_network_and_predict)
                self.__thread_nn.start()

            #Draw components
            #Display loading hints
            self.__loading_hint_label.draw(self.get_screen())
            #Display loading progress and hourglass
            self.__loading_progress_label.draw(self.get_screen())
            self.__hourglass.draw(self.get_screen())
            #Display guess
            self.__ai_image.draw(self.get_screen())
            self.__ai_guess_display.draw(self.get_screen())
            self.__ai_guess_label.draw(self.get_screen())
            #Display AI Image
            self.__ai_image.draw(self.get_screen())

        pygame.display.flip()
```

So when globals.loading_active = True, it first checks if the AI has been specified. If not, it assigns itself from the active_ai global variable, which was updated in the ai selection panel.

After checking for the AI, it then checks for the threads.

```
  #Check Thread status
            if self.__thread_loading_hints == None or not 
self.__thread_loading_hints.is_alive():
                self.__thread_loading_hints = 
Thread(target=self.display_loading_hints)
                self.__thread_loading_hints.start()
            if self.__thread_ai_image == None or not 
self.__thread_ai_image.is_alive():
                self.__thread_ai_image = 
Thread(target=self.display_guess)
                self.__thread_ai_image.start()
            if self.__thread_progress == None or not 
self.__thread_progress.is_alive():
                self.__thread_progress = 
Thread(target=self.display_loading_progress)
                self.__thread_progress.start()
            #Check neural network thread
            if self.__thread_nn == None:
                self.__thread_nn = 
Thread(target=self.load_neural_network_and_predict)
                self.__thread_nn.start()
```

It checks if the threads are either None or is no longer active (had finished their function), it will start a new instance of that thread.

```
            #Check neural network thread
            if self.__thread_nn == None:
                self.__thread_nn = 
Thread(target=self.load_neural_network_and_predict)
                self.__thread_nn.start()
```

Take note that the __thread_nn thread does not check if it had finished its process, because we only need to load the neural network once. In fact, once the neural network process is completed, the game moves on to the next panel. However, while that task is not yet complete, the game will continue to do the following processes:

```
def display_loading_hints(self):
        """Method to display the loading hints"""
        msg:str = random.choice(self.__loading_hints)   #Pick a text by 
random
        self.__loading_hint_label.set_text(msg, True)
        self.__loading_hint_label.get_rect().centerx = 
self.get_screen().get_rect().centerx #Update label position
        time.sleep(5.0)
```

This function is called by __thread_loading_hints. It will randomly choose a sentence regarding neural network from the __loading_hints list every 5 seconds. This idea is inspired by various games that displays random texts during loading screens, such as Skyrim, Fallout 4, and Total War.

```
def display_guess(self):
        """Method to display AI's guess display screen"""
        #Pick guess by random
        guess:str = random.choice(constants.char_list) if not
self.__loading_finished else ":)"
        self.__ai_guess_label.set_text(guess, True)
        self.__ai_guess_label.get_rect().centerx =
self.__ai_guess_display.get_rect().centerx
        time.sleep(0.2)
```

The random character display is conducted by this function, which is linked to the __thread_ai_image thread. While the loading process is not yet finished, it randomly chooses a random character from constants.char_list. The contents of that list are inside constants, a subsidiary module.

```
char_list:[] = [str(x) for x in range(10)] + [chr(x) for x in range(65,
91)] + [chr(x) for x in range(97, 123)]       #A list consisting of the
characters 0-9, A-Z, and a-z
```

Otherwise, it would display the text ":)" instead to announce to the player that the loading is complete.

```
def display_loading_progress(self):
        """Method to display the loading progress messages and hourglass
animation"""
        progress:str = globals.loading_progress
        if progress != self.__loading_progress_label.get_text():
#Only update when needed
            self.__loading_progress_label.set_text(progress, True)

            #Update positions
            self.__loading_progress_label.get_rect().centerx =
self.get_screen().get_rect().centerx - self.__hourglass.get_rect().width
            self.__hourglass.get_rect().left =
self.__loading_progress_label.get_rect().right
            self.__hourglass.get_rect().centery =
self.__loading_progress_label.get_rect().centery

        self.__hourglass.next()     #Update hourglass animation
        time.sleep(0.08)
```

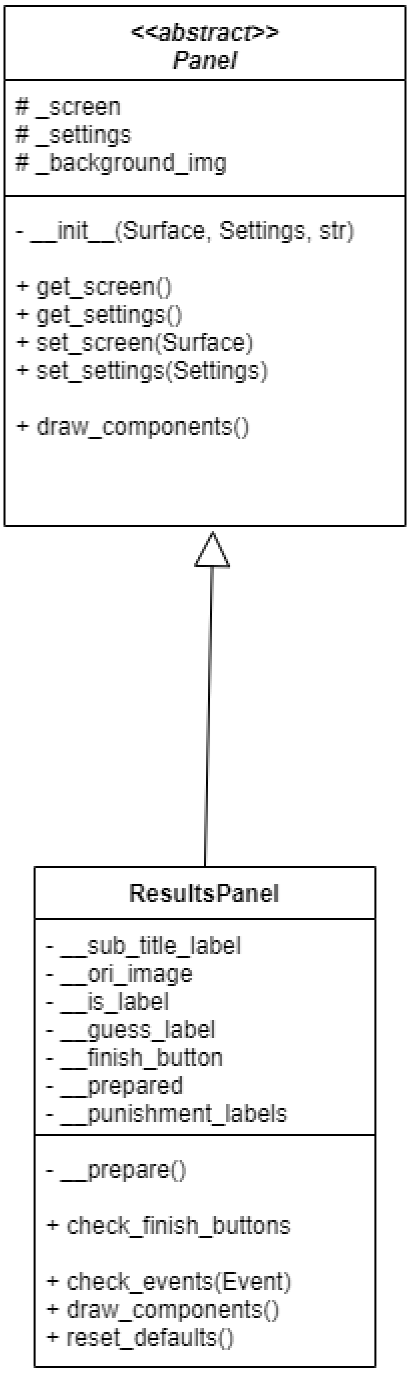
While loading is not yet finished, its progress (alongside the hourglass animation) is displayed by the function above. It checks for an update in the loading_progress global variable. If there is an update, the text is updated. There is no need to update the text every single time as assigning the text requires repositioning the text and its Rect object alongside the hourglass all over again. Hence, it should only be done when necessary. The hourglass animation is updated by simply calling the next() function and the time.sleep() function gives 0.08 seconds gap for each hourglass image. It is basically tells how long is one frame.

The final thread, __thread__nn, executes the following function:

```
def load_neural_network_and_predict(self):
        """Loads the neural network of the specified AI"""
        #Load neural network
        self.__loading_finished = False
        methods.update_loading_progress("Loading neural network...")
        nn:NeuralNetwork = self.__ai.load_neural_network()

        #Load input
        methods.update_loading_progress("Loading input...")
        image = file_operation.load_image(constants.pygame_image_path +
constants.pygame_test_image_name)
        methods.update_loading_progress("Formatting input...")
        image = methods.prepare_images_for_mlp_input([image])

        #Prediction
        methods.update_loading_progress("Making prediction...")
        prediction = nn.predict(image[0].reshape(1,
constants.image_width,
                                                constants.image_height,

constants.color_channels), False)

        methods.update_loading_progress("Prediction complete!")
        globals.prediction = prediction
        self.__loading_finished = True
        time.sleep(2.0)
        globals.panel_index += 1
        globals.show_results = True
```

This function updates the loading_progress global variable in every step of the neural network loading and prediction making procedure. Once a prediction is made, the prediction and show_results global variables are updated (these global variables are necessary for the next panel). The __loading_finished flag is also updated accordingly to tell the other threads that the loading is finished. The thread is paused for 2 seconds so that the game does not go directly to the next panel once the loading is finished (to add a slight suspense)

With the loading screen panel finished, the final panel is the results panel. This panel is the place where the AI's guess is displayed. Below is an outline of the class:



The __guess_label field stores the AI's prediction. However, similar to the thread back in the LoadingScreen class, it does not have any initial values as it depends on a global variable to load it. The global variable this time is show_results. It regularly checks for show_results to have a value of True by utilizing the draw_components() function.

```
def draw_components(self):
        super().draw_components()

        if globals.show_results:
            if not self.__prepared:
                self.__prepare()
            #Draw components
            self.__sub_title_label.draw()
            self.__ori_image.draw(self.get_screen())
            self.__is_label.draw()
            self.__guess_label.draw()
            self.__finish_button.draw()
            for label in self.__punishment_labels:
                label.draw()

        pygame.display.flip()
```

So when the show_results global variable is set to True (which is updated in the LoadingScreen class), it first check if it is the first iteration in the draw_components() function loop (recall that this function is called by the update_screen() function in game_functions module). If so, the __prepare() function is called where it initializes the components to be drawn on the screen.

```
def __prepare(self):
        """Method to prepare the components for display"""
        self.__ori_image.set_image(image = 
pygame.transform.scale(pygame.image.load(constants.pygame_image_path + 
constants.pygame_test_image_name), (313, 313)))
        self.__guess_label.set_text(globals.prediction)

        #Configure placements
        #Sub title
        self.__sub_title_label.get_rect().top = 
self.get_screen().get_rect().top + 30
        self.__sub_title_label.get_rect().centerx = 
self.get_screen().get_rect().centerx
        self.__sub_title_label.set_text(globals.active_ai.get_name() + " 
guessed..." if isinstance(globals.active_ai, AI) else 
globals.active_ai.get_nicer_name() + " guessed...")
        #Ori image
#          self.__ori_image.get_rect().right = 
self.__is_label.get_rect().left - 50
        self.__ori_image.get_rect().centerx = 
self.get_screen().get_rect().centerx//2
        self.__ori_image.get_rect().centery = 
2*self.get_screen().get_rect().centery//3
        #Is Label
        self.__is_label.get_rect().centery = 
self.__ori_image.get_rect().centery
        self.__is_label.get_rect().centerx = 
self.get_screen().get_rect().centerx
        self.__is_label.prep_msg(self.__is_label.get_text())
        #AI Guess label
        self.__guess_label.get_rect().centerx =
self.get_screen().get_rect().centerx +
self.get_screen().get_rect().centerx//2
        self.__guess_label.get_rect().centery =
self.__is_label.get_rect().centery
        self.__guess_label.prep_msg(self.__guess_label.get_text())
        #Punishment labels
        max_width:int = 0
        for label in self.__punishment_labels:
            width = label.get_rect().width
            if width > max_width:
                max_width = width
        rect:Rect = Rect(0, 0, max_width, 1)
        rect.top = self.__ori_image.get_rect().bottom + 20
        rect.left = self.get_screen().get_rect().left + 50
        self.__punishment_labels[0].get_rect().left = rect.left
        self.__punishment_labels[0].get_rect().top = rect.top

self.__punishment_labels[0].prep_msg(self.__punishment_labels[0].get_tex
t())
        for a in range(1, len(self.__punishment_labels)):
            self.__punishment_labels[a].get_rect().top =
self.__punishment_labels[a-1].get_rect().bottom
            self.__punishment_labels[a].get_rect().left =
self.__punishment_labels[a-1].get_rect().left

self.__punishment_labels[a].prep_msg(self.__punishment_labels[a].get_tex
t())

        self.__prepared = True
```

The preparation involves setting the user's drawing up for display and showing the AI's guess. This is done by the following two lines of code in the function above:

```
self.__ori_image.set_image(image =
pygame.transform.scale(pygame.image.load(constants.pygame_image_path +
constants.pygame_test_image_name), (313, 313)))
        self.__guess_label.set_text(globals.prediction)
```

Below that are just code to configure how to place the components. Once the __prepare() function finishes its task by updating the value of __prepared from False to True (which is important so that the components do not get reconfigured after the first loop), the components are drawn on the screen in the draw_components() function. This concludes the lengthy discussion regarding the implementation of graphical user interface (GUI) in the game.

Data Drivers:

Data drivers are classes created with the sole purpose of storing information regarding a particular object. There are two main data driver classes used for this project:

- NeuralNetwork

- AI
- Settings

These data drivers (alongside their UML diagram) has been shown in earlier discussions so they won't be explained here in great detail again.

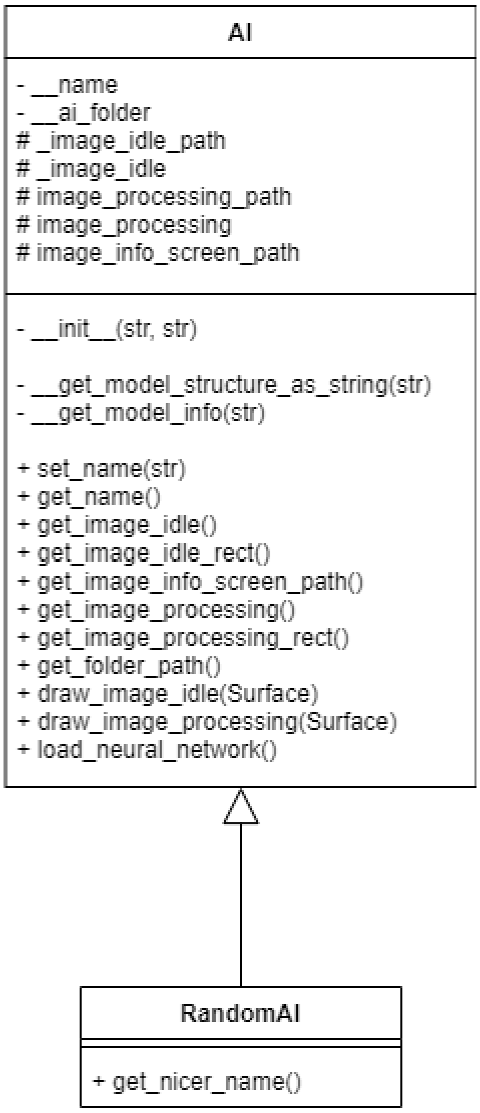
To recap, the NeuralNetwork class is used to serve an easier syntax when trying to do a few operations on Keras' neural network model. For example, the predict() automatically converts the neural network output into the actual predicted character. By default, the neural network outputs a tensor with the probability for each output. This function skips a number of operations in order for the player to see and read what the output actually meant.

```
def predict(self, image, show_pred_graph:bool = False) -> str:
        """Method to predict what character is the image, returns the
image."""
        #If show_pred_graph = True, it will draw the image and the
prediction in a matplotlib graph
        prediction = self.get_model().predict(image)
#         for pred in prediction:
#             print(pred)
        prediction_argmax = prediction.argmax()
#         print("Argmax:", prediction_argmax)
        predLabel:str =
constants.char_list[int(prediction_argmax.__str__())]

        if show_pred_graph:
            methods.show_prediction_graph(image.reshape(128, 128, 3),
predLabel)

        return predLabel
```

So the user will see "A" (for example) as the neural network output as opposed to [0.00021, 0.330, 0.00021, ….].

While the NeuralNetwork class handles Keras, the AI class is used to represent the game's AI characters as an object with its primary feature in storing information regarding its neural network model without loading it into memory until needed. This prevents the user from having a full RAM when playing the game.

One important note regarding the AI class is that it has a child class called RandomAI. This class' UML diagram is quite simple:

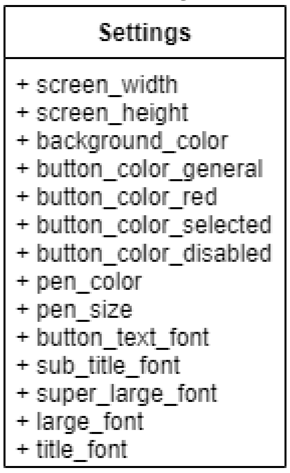


It only has one extra function in addition to everything else from its parent class. However, it is initialized quite differently.

```
class RandomAI(AI):

    #Constructor
    def __init__(self):
        #Randomize AI Folder
        root_folder:str = "ai_files"
        folders = [folder for folder in os.listdir(root_folder) if
os.path.isdir(os.path.join(root_folder, folder))]
        super().__init__("??????????", root_folder + "/" +
random.choice(folders) + "/")

        self._image_idle_path = constants.path_img_mystery_ai
        self._image_processing_path = constants.path_img_mystery_ai
        self.image = pygame.image.load(self._image_idle_path)
        self._image_processing =
pygame.image.load(self._image_processing_path)
```

```
        self._image_info_screen_path = constants.path_img_ai_info_empty

    #Other Methods
    def get_nicer_name(self):
        return "AI"
```
To put it simply, the AI class looks for its ai folder that contains its images and neural network model, however for this class, that folder location is randomized. An instance of this class may get an Cintra, Echo, Glados, or others as an AI, but the user would not know. In fact, their names are nothing more than “?????????”. The get_nicer_name() function is only called in the ResultsPanel class, so that the label does not say ““?????????? guessed....” but “AI guessed…”. With its random and uncertain nature, RandomAI is only used for the Random Chaos game mode.

Settings is the only data driver class that has no functions.



Settings

+ screen_width
+ screen_height
+ background_color
+ button_color_general
+ button_color_red
+ button_color_selected
+ button_color_disabled
+ pen_color
+ pen_size
+ button_text_font
+ sub_title_font
+ super_large_font
+ large_font
+ title_font

Since a lot of components in the game, especially the GUI components, needs to access several settings about the game, it was decided to keep them all in one class as one object. The Settings class stores some predefined configurations about the game, such as the font general size, the background color, and how big of a rectangle is drawn in the draw panel.

Subsidiary Modules:

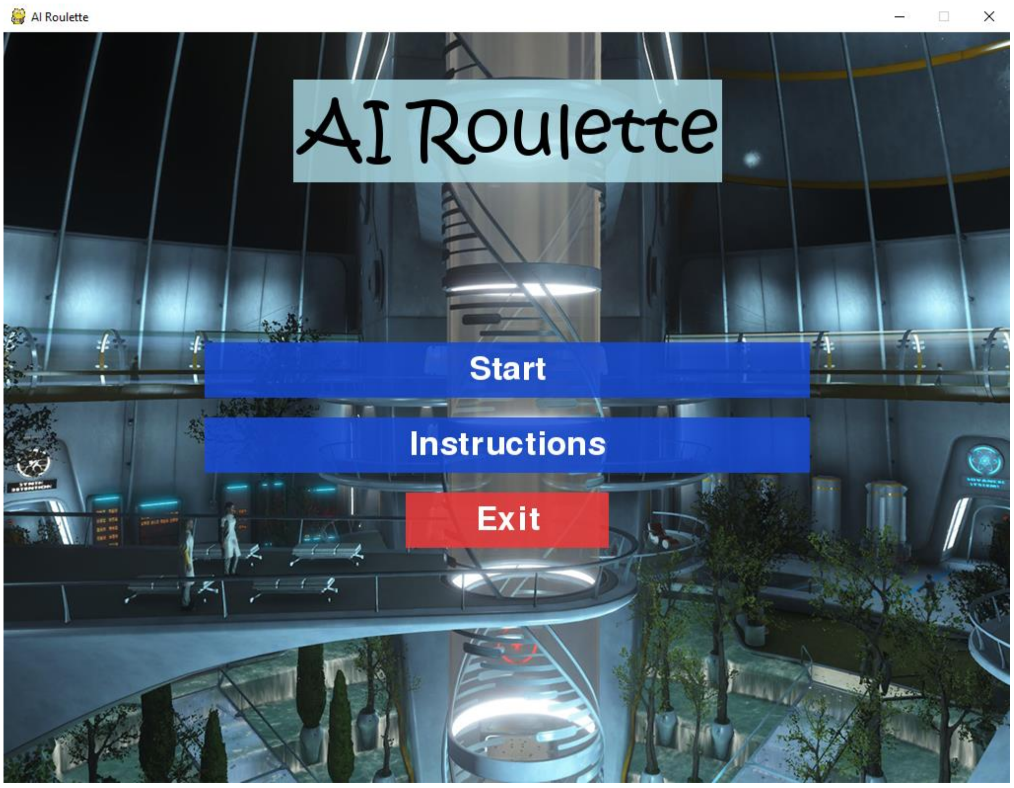
While most modules in this project has a class, some modules only contain functions or variables. The purpose of these modules is to provide easy access to values and execute functions without needing to instantiate an object instance of the class. The subsidiary modules used for this project, alongside their purpose, is detailed below:

Module	**Purpose**
file_operation	A collection of functions that handles file operations such as reading from files and loading images.
game_functions	A collection of functions to be used in the making of the game. These include updating the game screen, checking for user input, and getting the current mouse position.

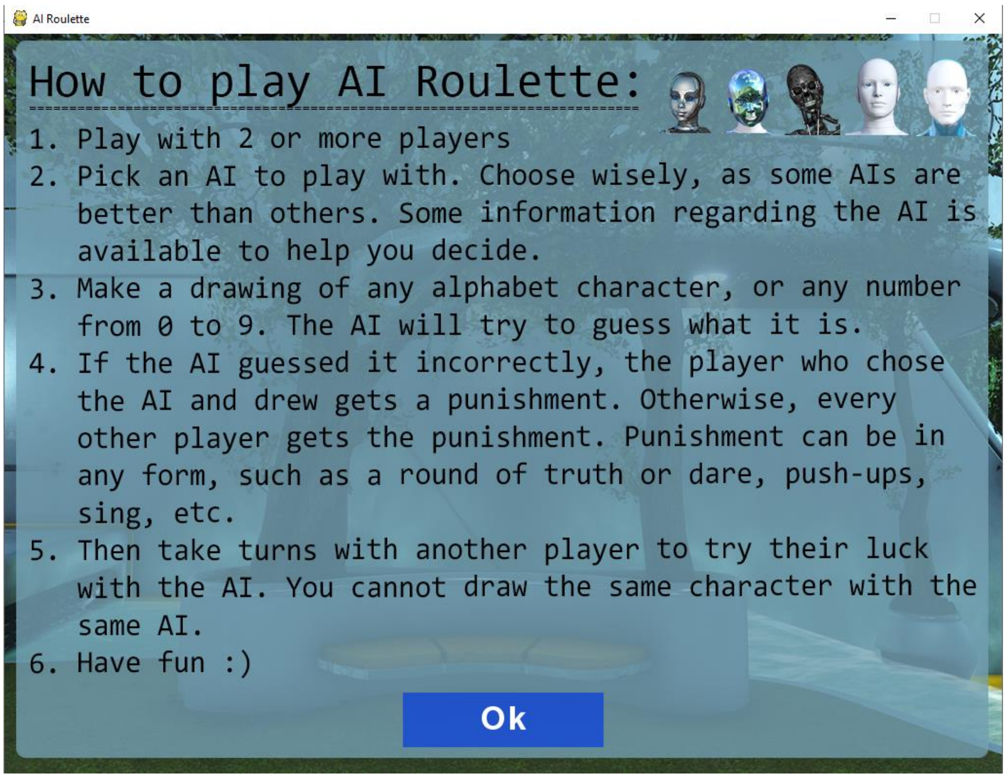
methods	A collection of utility functions that do not fit in any of the two previous modules. Most of the functions in this module are used for the neural network, such as formatting the input images, creating the neural network model, and training the neural network and saving them in periodic iterations.
constants	A collection of variables that should not be altered in any way at any point as the game runs. Some of these variables include a list of characters that the AI can detect, the dimensions of the image that the neural network processes, and the expected amount of color channels in the image.
globals	A collection of variables that are accessed and modified as the game progresses. A number of classes make use of these variables to function. Some of these variables include panel_index, loading_progress, and reset_defaults.

Evidence of Working Program

Main Menu



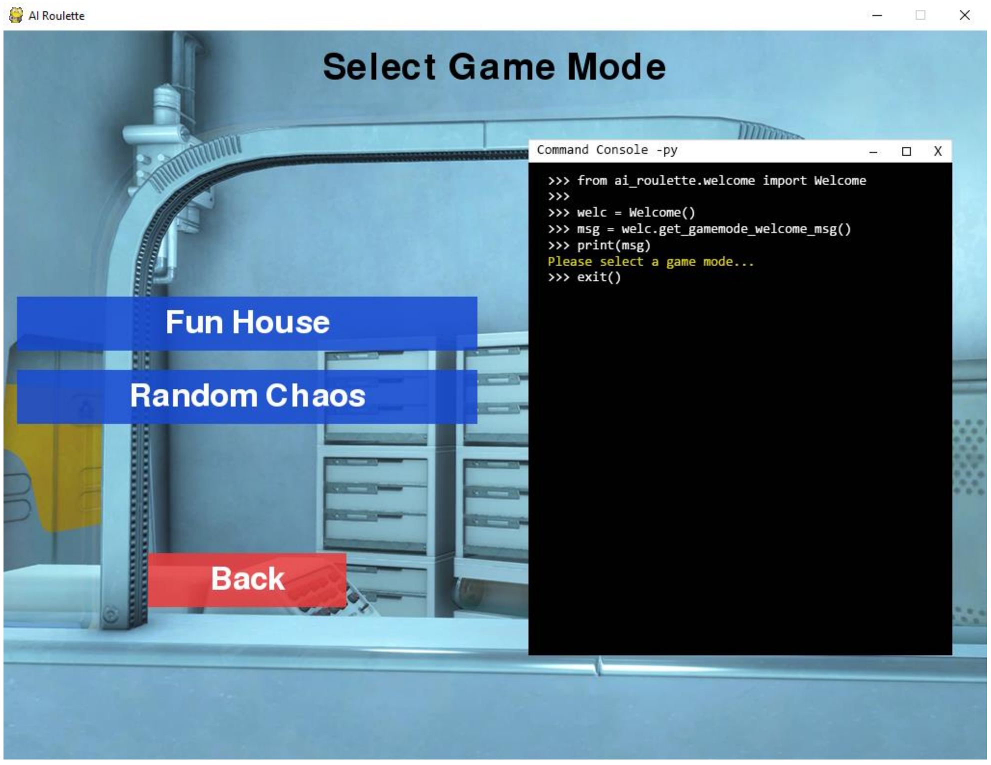
Instructions Panel



How to play AI Roulette:

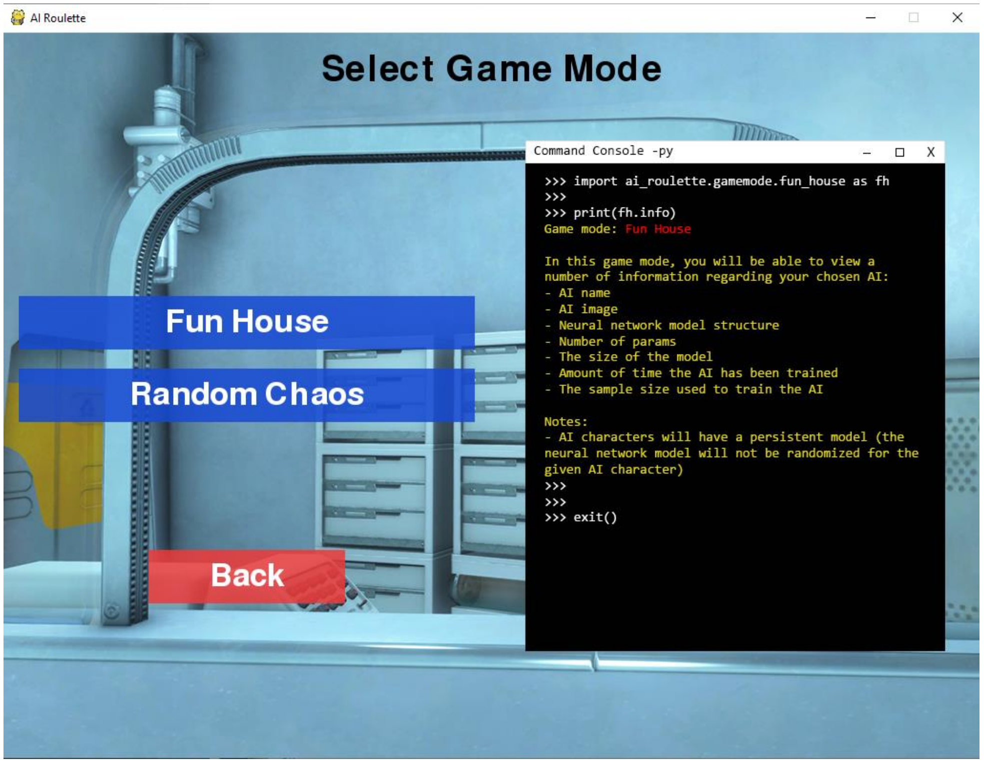
1. Play with 2 or more players
2. Pick an AI to play with. Choose wisely, as some AIs are better than others. Some information regarding the AI is available to help you decide.
3. Make a drawing of any alphabet character, or any number from 0 to 9. The AI will try to guess what it is.
4. If the AI guessed it incorrectly, the player who chose the AI and drew gets a punishment. Otherwise, every other player gets the punishment. Punishment can be in any form, such as a round of truth or dare, push-ups, sing, etc.
5. Then take turns with another player to try their luck with the AI. You cannot draw the same character with the same AI.
6. Have fun :)

Game Mode Selection



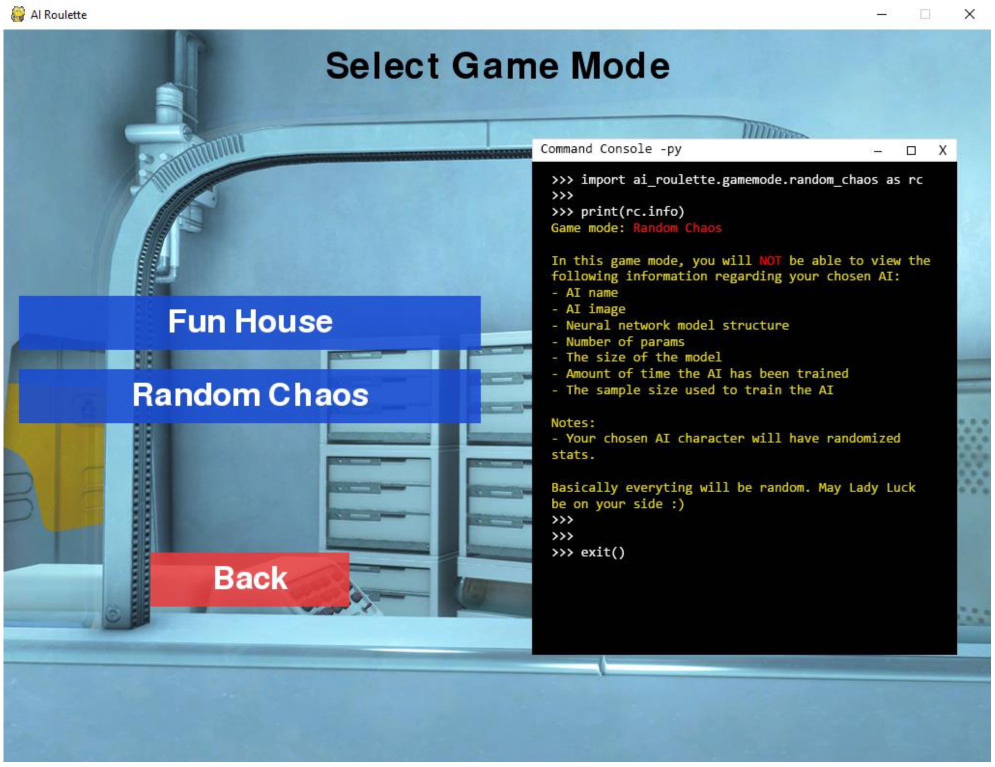
```
>>> from ai_roulette.welcome import Welcome
>>>
>>> welc = Welcome()
>>> msg = welc.get_gamemode_welcome_msg()
>>> print(msg)
Please select a game mode...
>>> exit()
```

When the mouse hovers over the Fun House button:



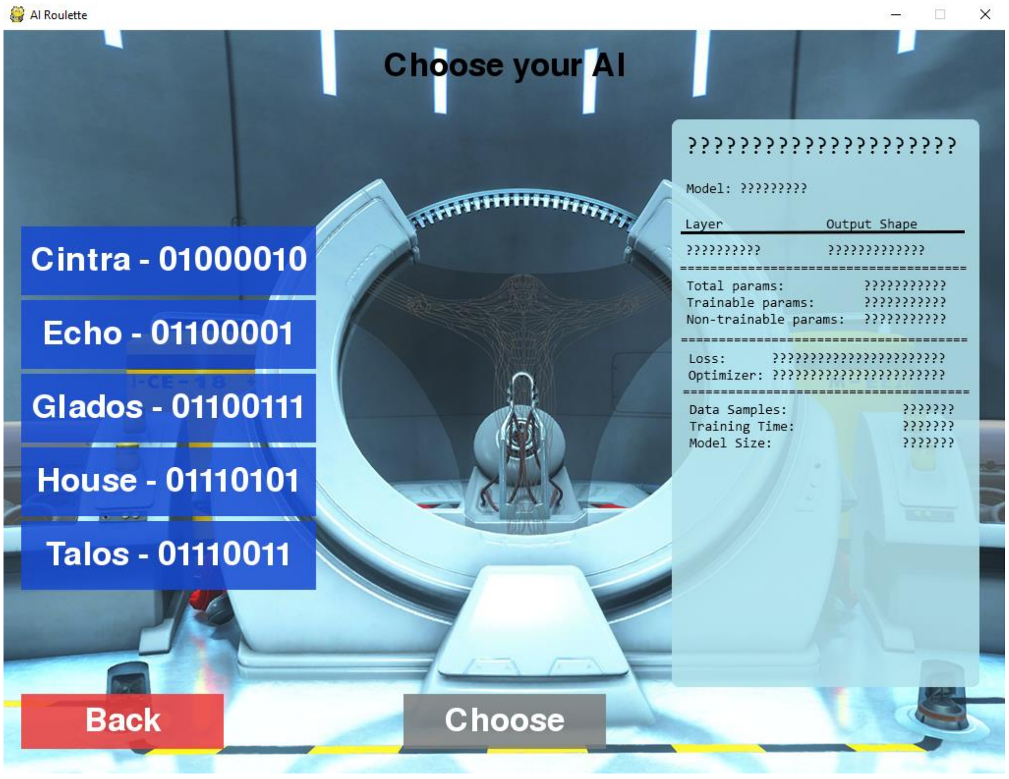
```
>>> import ai_roulette.gamemode.fun_house as fh
>>>
>>> print(fh.info)
Game mode: Fun House

In this game mode, you will be able to view a
number of information regarding your chosen AI:
- AI name
- AI image
- Neural network model structure
- Number of params
- The size of the model
- Amount of time the AI has been trained
- The sample size used to train the AI

Notes:
- AI characters will have a persistent model (the
neural network model will not be randomized for the
given AI character)
>>>
>>>
>>> exit()
```

When the mouse hovers over the Random Chaos button:



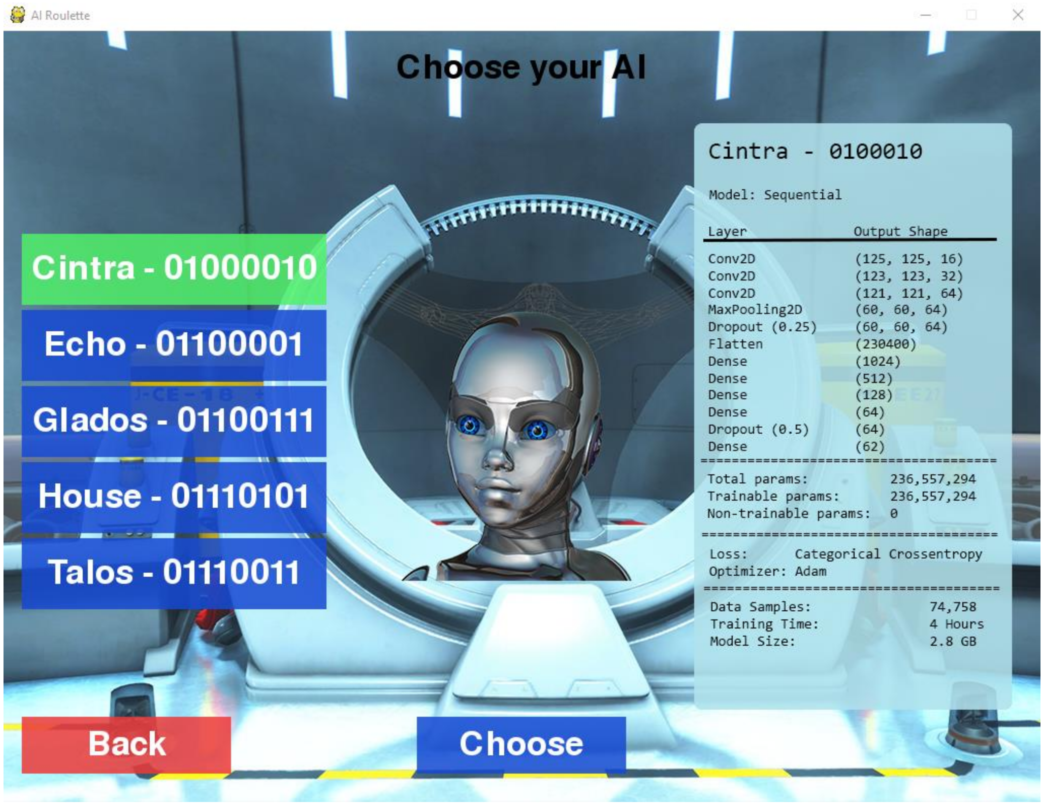
```
>>> import ai_roulette.gamemode.random_chaos as rc
>>>
>>> print(rc.info)
Game mode: Random Chaos

In this game mode, you will NOT be able to view the
following information regarding your chosen AI:
- AI name
- AI image
- Neural network model structure
- Number of params
- The size of the model
- Amount of time the AI has been trained
- The sample size used to train the AI

Notes:
- Your chosen AI character will have randomized
stats.

Basically everyting will be random. May Lady Luck
be on your side :)
>>>
>>>
>>> exit()
```

AI Selection



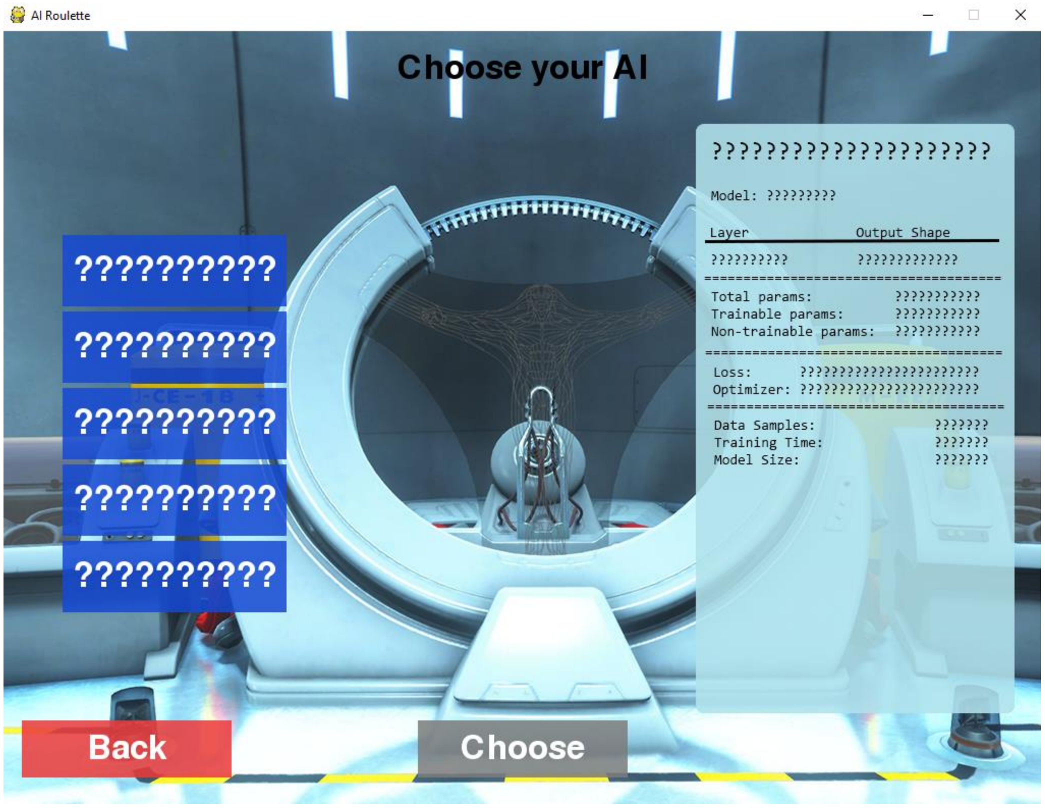
```
?????????????????????

Model: ?????????

Layer                   Output Shape
?????????????           ??????????????
=========================================
Total params:               ???????????
Trainable params:           ???????????
Non-trainable params:       ???????????
=========================================
Loss:       ??????????????????????
Optimizer:  ??????????????????????
=========================================
Data Samples:                   ???????
Training Time:                  ???????
Model Size:                     ???????
```

When an AI is selected:



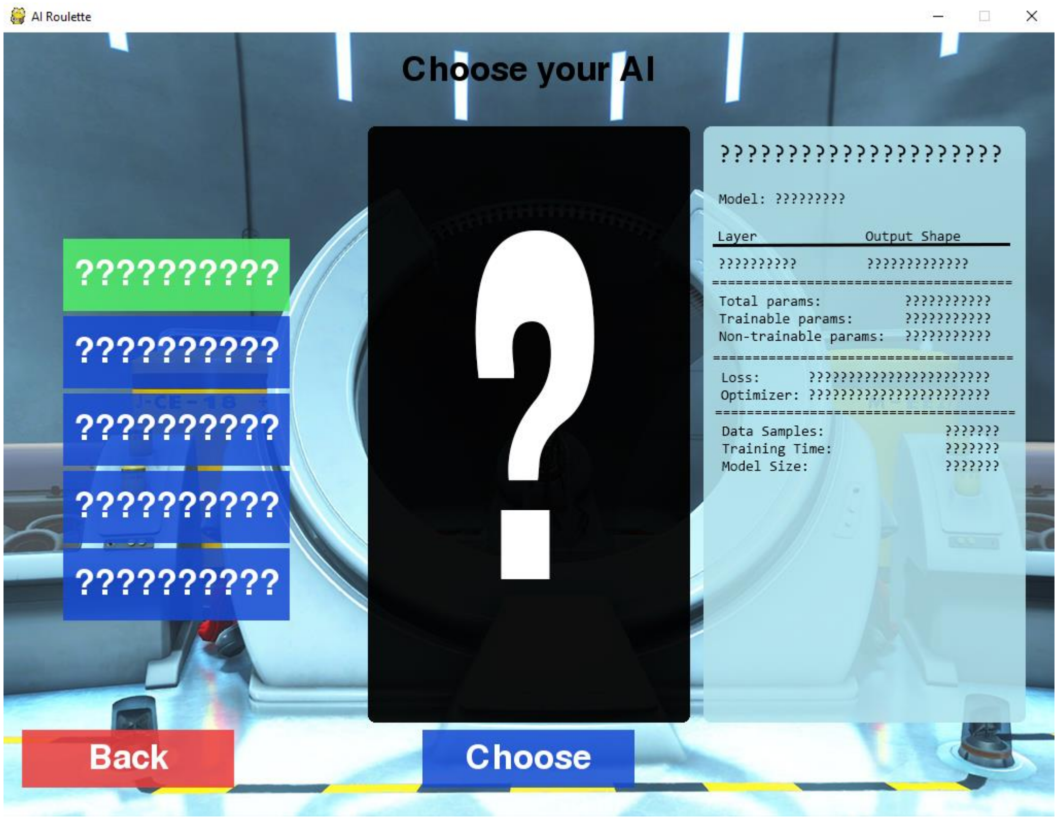
```
Cintra - 0100010

Model: Sequential

Layer                    Output Shape
Conv2D                   (125, 125, 16)
Conv2D                   (123, 123, 32)
Conv2D                   (121, 121, 64)
MaxPooling2D             (60, 60, 64)
Dropout (0.25)           (60, 60, 64)
Flatten                  (230400)
Dense                    (1024)
Dense                    (512)
Dense                    (128)
Dense                    (64)
Dropout (0.5)            (64)
Dense                    (62)
=====================================
Total params:            236,557,294
Trainable params:        236,557,294
Non-trainable params:    0
=====================================
Loss:      Categorical Crossentropy
Optimizer: Adam
=====================================
Data Samples:                 74,758
Training Time:                4 Hours
Model Size:                   2.8 GB
```

<u>AI Selection (Random)</u>



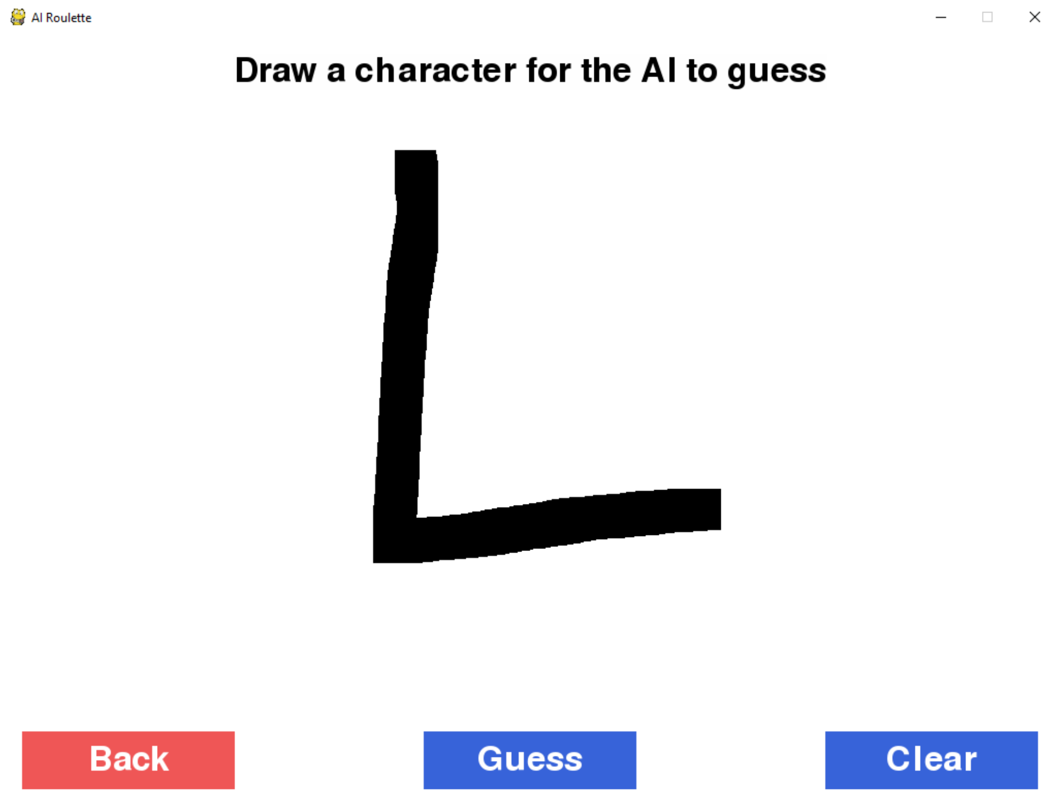
```
????????????????????

Model: ?????????

Layer                    Output Shape
?????????                ????????????
=====================================
Total params:            ???????????
Trainable params:        ???????????
Non-trainable params:    ???????????
=====================================
Loss:      ??????????????????????
Optimizer: ??????????????????????
=====================================
Data Samples:                ???????
Training Time:               ???????
Model Size:                  ???????
```

When an AI is selected:



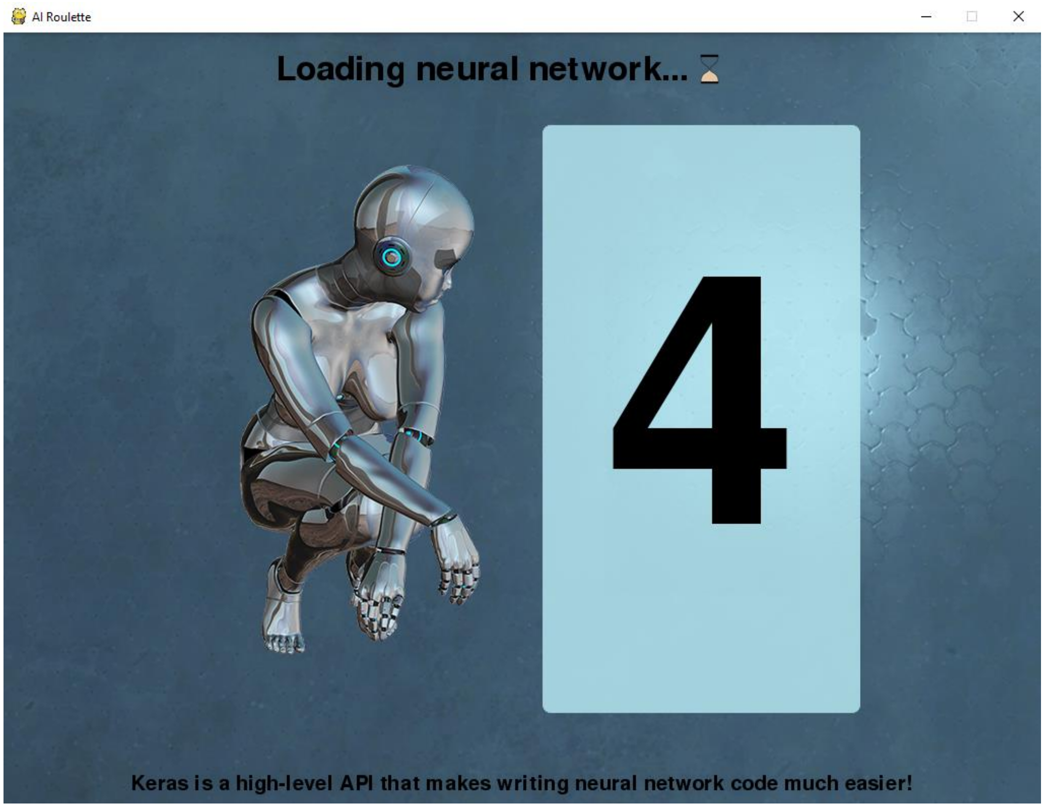
```
?????????????????????

Model: ?????????

Layer                Output Shape
??????????           ??????????????
==========================================
Total params:              ???????????
Trainable params:          ???????????
Non-trainable params:      ???????????
==========================================
Loss:       ???????????????????????
Optimizer:  ???????????????????????
==========================================
Data Samples:                    ???????
Training Time:                   ???????
Model Size:                      ???????
```

Draw Panel (for both the normal and random version)



Loading Screen

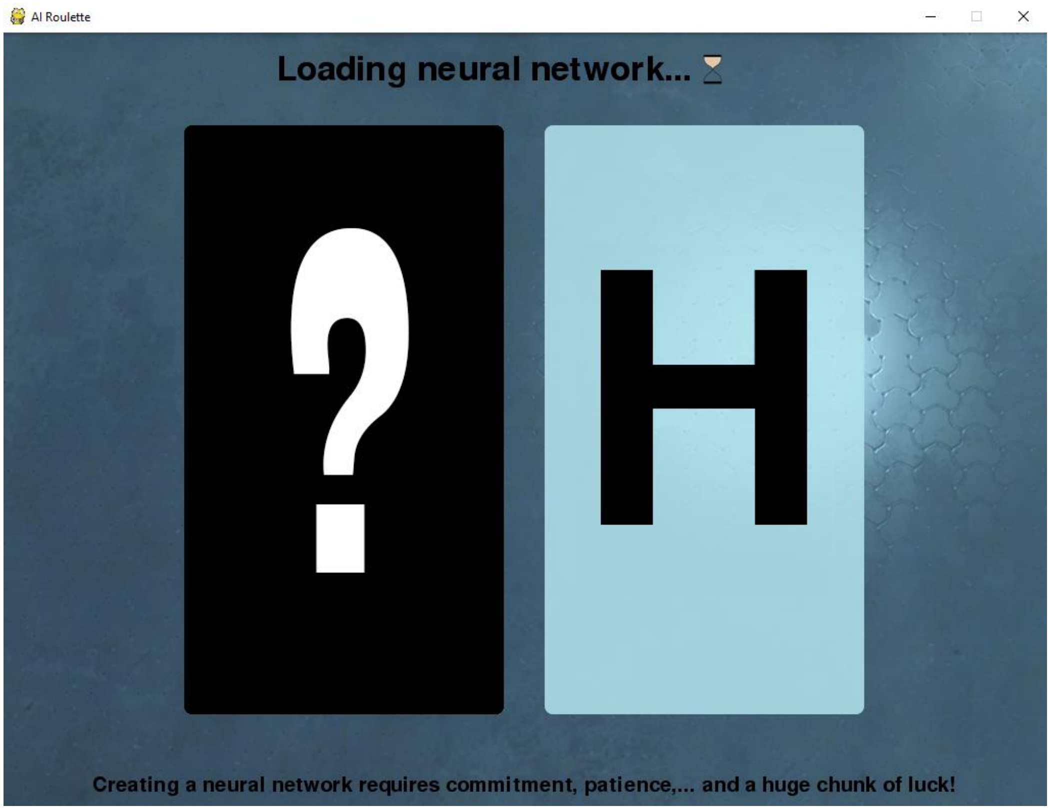
For Fun House:



Loading neural network... ⌛

Keras is a high-level API that makes writing neural network code much easier!

For Random Chaos:

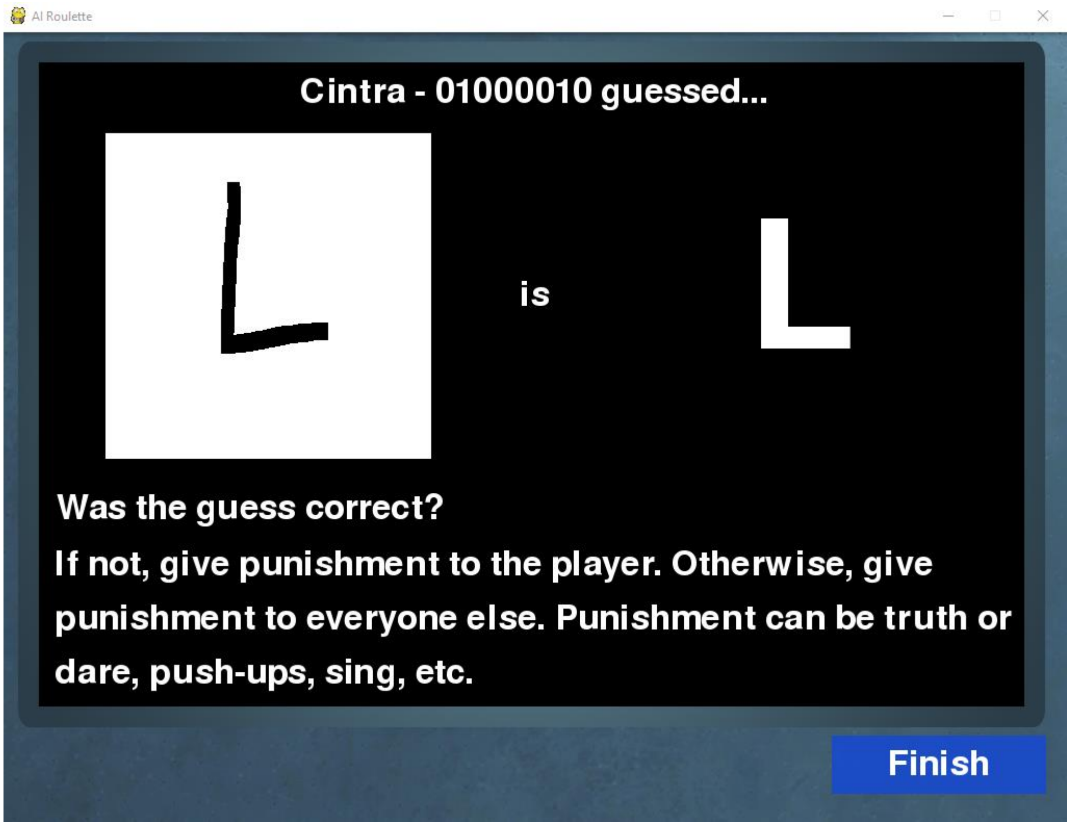


Loading neural network... ⌛

Creating a neural network requires commitment, patience,... and a huge chunk of luck!

Results Panel

For Fun House:



AI Roulette

Cintra - 01000010 guessed...

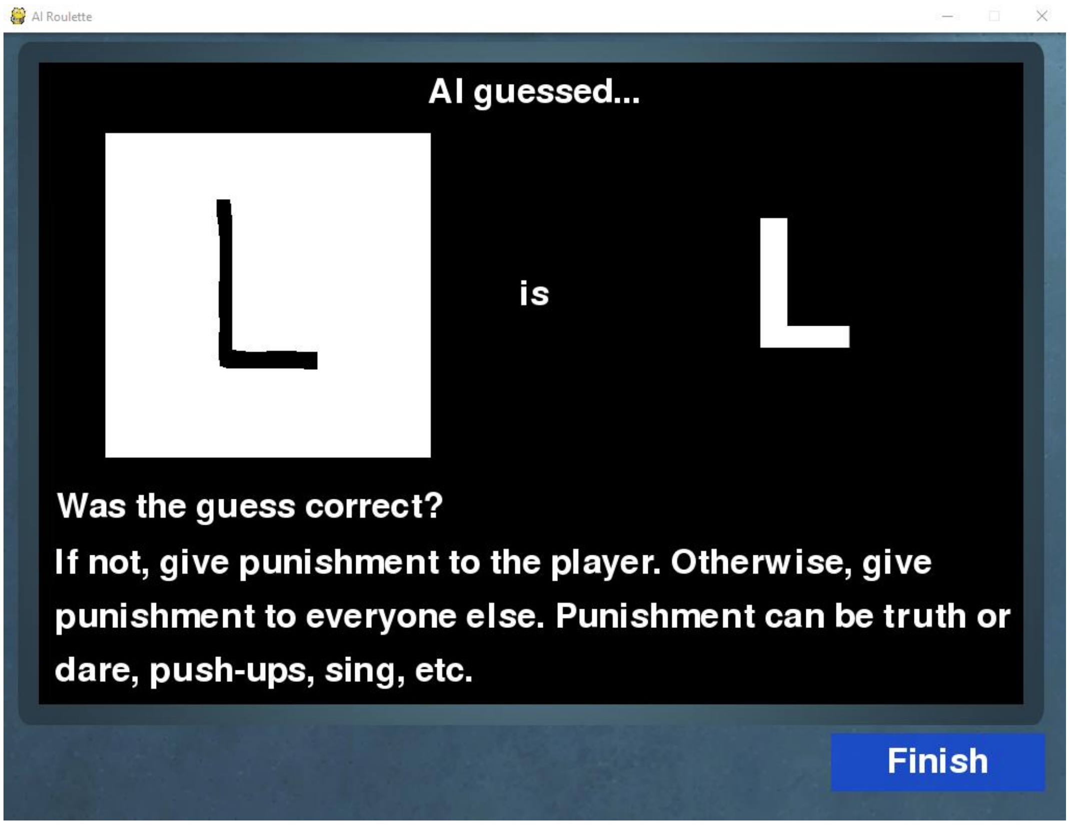
is

L

Was the guess correct?

If not, give punishment to the player. Otherwise, give punishment to everyone else. Punishment can be truth or dare, push-ups, sing, etc.

For Random Chaos:



AI Roulette

AI guessed...

is

L

Was the guess correct?

If not, give punishment to the player. Otherwise, give punishment to everyone else. Punishment can be truth or dare, push-ups, sing, etc.

Credits

- Barroa_Artworks
  - Glados AI images
    - https://pixabay.com/illustrations/robot-cyborg-scifi-android-robotic-3226893/
- Bethesda Game Studios
  - Fallout 4 images used for panel background
- Botlibre
  - Talos AI images
    - https://pixabay.com/illustrations/robot-android-face-intelligence-3431313/
    - https://pixabay.com/illustrations/robot-android-intelligence-3431317/
- Clipart Library
  - Hourglass image
    - http://clipart-library.com/clipart/179228.htm
- DrSJS
  - Cintra AI images
    - https://pixabay.com/illustrations/girl-woman-side-robot-cyborg-320267/
    - https://pixabay.com/illustrations/girl-woman-posing-front-robot-320264/
- Matplotlib
- OpenCV
- Pandas
- Pygame
- Tensorflow
- TheDigitalArtist
  - Echo AI images
    - https://pixabay.com/illustrations/robot-child-future-science-kid-1583675/
    - https://pixabay.com/illustrations/robot-cyborg-futuristic-artificial-1665620/
  - House AI images
    - https://pixabay.com/photos/futuristic-robot-cyborg-3308094/
    - https://pixabay.com/photos/robot-cyborg-futuristic-android-3310192/