Final Project

Program Design and Methods

Project Name: **TYPO**

Name: **Sri Kalyan Rohan**

Student ID: **2440090266**

Class: **L1BC**

Project Specification

The purpose of this project is to create a typing game that can help people measure and better their typing skills. It may seem simple at first but having a better typing skill can help do tasks with a higher efficiency as nowadays especially during this pandemic, everything is done online and through the computer. The user can choose how many sentences they want to type in and then they can start the game by typing in the input box. The sentences will be randomly generated from an external text file. On the bottom left of the screen, there will be a Typo rate which is the percentage of wrong character input that the user enters from the entire character input. For example, if there are 100 characters and the wrong input made is 2 then the typo rate will show 2 percent. At the end of the game, the user will be brought to a game over window where it shows the statistics of the user from the typo rate, accuracy, words per minute, number of characters and time taken. Based on the statistics, the user will be given a rank accordingly and it will be indicated with a gem. Even though, typing games are fairly simple, I think that nowadays games that are minimalistic and are user friendly can last for quite some time and not only that, you also get to better your typing skills!

Input

1. User's keystroke when typing in the sentences.
2. Number of sentences

Output

1. Accuracy
2. Words typed per minute
3. Number of characters typed
4. Typo Rate
5. Rank
6. Time taken

Solution Design

1. Main Menu
2. How to play
3. Set up (How many sentences)
4. Main Game
5. Results

Main Menu

The main menu is pretty straight forward. You can use it to navigate yourself through the game like going to the How to Play window to look at how the game is played or you can go to the Instruction window to setup and play the game.

How to Play

Another straight forward window that shows the user how to play the game. It also shows the guide of the ranks or tiers that the user can get as a result.

Set up

This window sets up the game of the user by asking the user for input for the number of sentences that the user wants to play with.

Main Game

This window is where all of the game play happen. The user can track their typo rate through the bottom left of their screen. This window starts with a guide text on how the user can start playing the game. The user then starts typing the sentences in the input box

Results

This window will show the user their statistics. It shows the accuracy, speed, time taken, number of characters and the rate of typo of the user. It also shows which tier or rank the user ends up with.

Implementation and explanation of code

For this project, I'm only using two modules which are the arcade module and the random module. There are 6 classes that I created in this project. 5 of them are used for showing the different views or windows of the game and the last 1 is for the GUI element. For the view classes (Main Menu Class, How to Play class, Game View class, Game Over class and Setup View class) I am inheriting from the arcade class module called ac.View(). This view basically allows me to have multiple views or windows as I can easily navigate through them. For the UI class (Button class), I will be inheriting from the arcade GUI class module called ac.gui.UIFlatButton(). This class already is a built in button class that I can use to create my buttons later on.

Main Menu Class

```
def __init__(self):
    super().__init__()
    # this ui manager is from the arcade module and it controls our UI such as buttons.
    self.ui_manager = UIManager()
    #drawing the background image
    self.background_image=ac.sprite.Sprite('computer-overhead-dribbb_1.gif',
                                           center_x=self.window.width//2,
                                           center_y=self.window.height//1.4,
                                           scale=0.3)
```

In the init method is where I initialize the attributes that I are going to use to make the view. All throughout the proejct self.ui_manager will always be equal to UIManager(). This UIManager() comes from the arcade module and it is used to manage the UI elements of the view such as buttons. I also initialize the background image by using the built in sprite class from the arcade module so that I can load the image sprite.

```
"""The on_draw method is called whenever we draw text,sprites and etc to the current view."""
def on_draw(self):
    ac.start_render()
    # drawing the title
    ac.draw_text(SCREEN_TITLE,(self.window.width//2-100),self.window.height//1.2,ac.color.WHITE,72,align='center')
    # drawing the bg image
    self.background_image.draw()
```

The on_draw method is used to draw the elements of the View. The start render function is called to render the elements that I want to draw. The draw_text function is also called to draw text to the view. I also use the draw sprite method the draw the background_image sprite that I intialized earlier.

```
""" Called once when view is activated. """
def on_show_view(self):
    #calling the setup function whenever we switch to this view to setup our view.
    self.setup()
    #setting the bg color
    ac.set_background_color(ac.color.BLUEBERRY)

""" Called once when view is deactivated. """
def on_hide_view(self):
    #removes the handlers of this view when we switch to another view from this view.
    self.ui_manager.unregister_handlers()
```

The on_show_view and on_hide_view method is called when I change to and from the view respectively. For the on_show_view method, whenever I switch to the view, I want to call the setup() method to setup the view instantly so that the elements in the view will be already setted up once I switch to this view. I also set the background color here. For the on_hide_view(), I use the UI Manager() built in method called the unregister_handlers() method, to unregister the handlers of the current view, so that the gui elements from different views don't overlap with each other.

```
""" Everything under this function will run everytime we switch to this view  """
def setup(self):
    #Removes all UIElements which were added to the UIManager()
    self.ui_manager.purge_ui_elements()
    # Loads the background image
    self.background = arcade.load_texture(":resources:images/backgrounds/abstract_1.jpg")
    y=self.window.height//2
    x=self.window.width//2
    #play button gui
    play_button=Button(text="Play",center_x=x,center_y=y,width=200,height=100,align='center')
    #using the inbuilt method in the UIManager() to add gui elements
    self.ui_manager.add_ui_element(play_button)
    #how to playy button gui
    how_to_play_button= Button(text="How To Play", center_x=x,center_y=y-150,width=200,height=100,align='center')
    self.ui_manager.add_ui_element(how_to_play_button)
    #quit button gui
    quit_button=Button(text="Quit",center_x=x,center_y=y-300,width=200,height=100,align='center')
    self.ui_manager.add_ui_element(quit_button)
```

The setup function is always called first whenever I change to the view thanks to the on_show_view() function. The first thing I did was to purge the UI elements. In other words, I am removing the UI elements that were added to the UI manager. I then load the background image by using the load_texture method. I set half the window height to y and half the window width to x so that it will be easier to adjust positions of elements later on. I then created the Play, how to play and quit button by using the Button class that I will be discussing later on. I then use the UI manager to add these buttons int to the view by using the add_ui_element() method.

```
"""This is the view for setting up the number of sentences for the game"""
class SetupView(ac.View):
    def __init__(self):
        super().__init__()
        # We use an input box from the arcade module to get the user's input
        self.number_of_sentences=ac.gui.UIInputBox(self.window.width//2,self.window.height//2,200)
        # same ui manager as the above. Note that we will be using the same thing for all of the views.
        self.ui_manager=UIManager()
        #initializing our start button
        self.start=None
        #initializing our back button
        self.back=None
```

The initializer for the Setup view. Here I used the same name for the UI Manager and basically for all the views. I set the self.start and self.back as None as I am going to be creating the start and back buttons using those respectively.

```
""" Called once when view is activated. """
def on_show_view(self):
    #setting the bg color
    ac.set_background_color(ac.color.ORANGE_PEEL)
    #calling the setup function when this view activates
    self.setup()

""" Called once when view is deactivated. """
def on_hide_view(self):
    #removes the handlers of this view when we switch to another view from this view.
    self.ui_manager.unregister_handlers()
```

Same on_show_view and on_hide_view function as the previous view except for the color of the background.

```
"""The on_draw method is called whenever we draw text,sprites and etc to the current view."""
def on_draw(self):
    ac.start_render()
    # Draw text
    ac.draw_text("Please enter the number of sentences.", self.window.width/2,
                 self.window.height/2+200,ac.color.WHITE,
                 font_size=50,
                 anchor_x="center")
    # If a non integer is entered, it will result in an error. The following text will only be drawn once there is an error.
    if error==True:
        ac.draw_text("Please enter an integer!.", self.window.width/2,
                     self.window.height/2+150,ac.color.RED,
                     font_size=30,
                     anchor_x="center")
```

On_draw function for the Setup View. Here I drew the necessary texts using the draw_text method. The second draw_text method is under a condition and that condition is if the global variable error is equal to true. I did this so that whenever the user inputs a non-integer to the number of sentences that they want, this text will be drawn to show that the users input is invalid.

```
""" Everything under this function will run everytime we switch to this view  """
def setup(self):
    #Removes all UIElements which were added to the UIManager()
    self.ui_manager.purge_ui_elements

    y=self.window.height//2
    x=self.window.width//2
    #set the start button
    self.start= Button(text="Start",center_x=x,center_y=y//1.5,width=200,height=100,align='center')
    self.ui_manager.add_ui_element(self.start)
    #set the back button
    self.back=Button(text="Back",center_x=x,center_y=y//4,width=200,height=100,align='center')
    self.ui_manager.add_ui_element(self.back)
    #adding the input box
    self.ui_manager.add_ui_element(self.number_of_sentences)
```

The setup() method for the Setup view. For the setup method of all views, you will notice that all of the GUI elements such as the button are added here. I will not elaborate much as I have elaborated already on the previous setup() method as I feel like it will be redundant.

```
""" All the logic, changes and input is written under here.
delta_time is a built in parameter that allows for real time changes to occur.  """
def on_update(self,delta_time):
    global number
    # Storing the input from the user in the global variable called number.
    number=self.number_of_sentences.text
```

The on_update function takes in one parameter and that is delta_time. Delta_time is essentially real time and this allows us to create changes and interact with the view. This is why all of the logic and user interface in the view is under this. For this on_update function I am just setting the input of the number of sentences from the user equal to the global variable "number". This global variable will be the number of sentences in the game view later on.

```
""" This class is ued to show the guide of the game"""
class HowToPlay(ac.View):
    def __init__(self):
        super().__init__()
        self.ui_manager=UIManager()
        self.emerald=ac.sprite.Sprite('Emerald Gem05.png',center_x=self.window.width//1.5,center_y=self.window.height//3)
        self.ruby=ac.sprite.Sprite('Ruby Gem_5.png',center_x=self.window.width//4.5,center_y=self.window.height//10.5)
        self.sapphire=ac.sprite.Sprite('Sapphire Gem05.png',center_x=self.window.width//4.5,center_y=self.window.height//4.5)
        self.topaz=ac.sprite.Sprite('Topaz Gem10.png',center_x=self.window.width//1.5,center_y=self.window.height//4.5)
        self.blue_gem=ac.sprite.Sprite('Aquamarine Gem05.png',center_x=self.window.width//4.5,center_y=self.window.height//3)
```

Initializer for the How To Play class(). Here I load the image sprites for the ranks using the Sprite method that is built into arcade.

```
    """ Called once when view is activated. """
    def on_show_view(self):
        ac.set_background_color(ac.color.RED)
        self.setup()

    """ Called once when view is deactivated. """
    def on_hide_view(self):
        self.ui_manager.unregister_handlers()
```

On_show_view() and on_hide_view() for the how to play class. The same as before but only changing the color to Red.

```
""" Everything under this function will run everytime we switch to this view  """
def setup(self):
    #Removes all UIElements which were added to the UIManager()
    self.ui_manager.purge_ui_elements()

    y=self.window.height//2
    x=self.window.width//2
    # add back button
    back_button=Button(text="Back",center_x=x,center_y=y-300,width=200,height=100,align='center')
    self.ui_manager.add_ui_element(back_button)
    # add play button
    play_button=Button(text="Play",center_x=x,center_y=y-150,width=200,height=100,align='center')
    self.ui_manager.add_ui_element(play_button)
```

Setup() method for the How to Play view. Here I added the play and back button using the same Button class that allows me to create an interactive button gui.

```
"""The on_draw method is called whenever we draw text,sprites and etc to the current view."""
def on_draw(self):
    ac.start_render()
    # drawing the guide text and sprites to the view
    ac.draw_text("How to Play",self.window.width/2,
                 self.window.height/2+270,ac.color.WHITE,40,align="center",anchor_x="center")
    ac.draw_text("1. Enter the number of sentences that you want.",self.window.width/2,
                 self.window.height/2+200,ac.color.WHITE,30,align="center",anchor_x="center")
    ac.draw_text("2. Start Typing in the box to start the game.",self.window.width/2,
                 self.window.height/2+150,ac.color.WHITE,30,align="center",anchor_x="center")
    ac.draw_text("3. If the color of the box becomes red, then you have made a TYPO! \n If the box turns green then, you have inputte
    ac.draw_text("4. You can see your typo percentage bottom left, The lower it is, the better. \n There are tiers or ranks for your
    self.emerald.draw()
    self.ruby.draw()
    self.blue_gem.draw()
    self.topaz.draw()
    self.sapphire.draw()
    ac.draw_text("SSS Tier",self.window.width//4.5+100,self.window.height//3,ac.color.WHITE,20,align="center",anchor_x="center")
    ac.draw_text("S Tier",self.window.width//4.5+100,self.window.height//4.5,ac.color.WHITE,20,align="center",anchor_x="center")
    ac.draw_text("A Tier",self.window.width//4.5+100,self.window.height//10.5,ac.color.WHITE,20,align="center",anchor_x="center")
    ac.draw_text("B Tier",self.window.width//1.5+100,self.window.height//3,ac.color.WHITE,20,align="center",anchor_x="center")
    ac.draw_text("C Tier",self.window.width//1.5+100,self.window.height//4.5,ac.color.WHITE,20,align="center",anchor_x="center")
```

The on_draw function for the How to Play view. Here is where I draw all the necessary sprites and text for the player guide. This whole view is essentially to give the player guidance and help on how to play the game.

```
""" The main game. This is where the user plays."""
class GameView(ac.View):

    def __init__(self):
        super().__init__()
        #initializing all the necessary attributes
        self.time_taken = 0
        self.typo=0
        self.wrong_input=0
        # the sentences that the user will be typing in. It is randomly generated from an external .txt file
        self.text=random.choice(sentences)
        self.ui_manager=UIManager()
        # the user will be typing in the sentences inside this UI Input Box from the arcade module.
        self.input=ac.gui.UIInputBox(self.window.width//2,self.window.height//3,1200)
        self.empty=''
        #the number of sentences will be the input from the user on the SetupView(). We change to int as the input is stored in a str.
        self.sentences=int(number)
        self.wpm=0
        self.start=False
        self.accuracy=0
        self.char=0
        self.correct_input=0
        # loading the typing sfx
        self.sfx=ac.load_sound(path='typingsfx.mp3',streaming=True)
        self.background = None
        self.start_text=""
```

This view is where the main game happens. Here I intialize the attributes for the statistics of the player. Such attributes are everything that is set to zero. I also initialize the user input as the same input box from the arcade module. I also intialize the sentence that the user will type using the random.choice() from the random module. This method chooses a random sentence from the "sentences" list that was set into a global variable. I also initialize the number of sentences as the global variable "number" and since this global variable is a string (because of the input from the Setup View), I use the int() method to change it to an integer. I also loaded the sfx sound here. I set the self.start to False as the game don't start instantly.

```
    """ Everything under this function will run everytime we switch to this view  """
    def setup(self):
        self.start_text="Start Typing to Start!"
        self.char=len(self.text)
        self.input.text= self.empty
        self.ui_manager.purge_ui_elements()
        self.input.cursor_index = len(self.input.text)
        self.input.set_style_attrs(font_size=14)
        self.ui_manager.add_ui_element(self.input)
```

Setup() method for the game view. Again here I added the gui elements. The set_style_attrs() method allows me to set the style of the GUI element. I also set the input text box as an empty string and the input box cursor equal to the length of the input text the user is going to put in. This means the cursor will only go as long as the input text.

```
    """ Called once when view is activated. """
    def on_show_view(self):
        ac.set_background_color(ac.color.BLACK)
        self.setup()

    """ Called once when view is deactivated. """
    def on_hide_view(self):
        self.ui_manager.unregister_handlers()
```

On_show_view and on_hide_view for the game view but this time the color is BLACK

```
"""The on_draw method is called whenever we draw text,sprites and etc to the current view."""
def on_draw(self):
    x=self.window.width
    y=self.window.height
    ac.start_render()
    # Put the sentence that the user should type on the screen.
    ac.draw_text(self.text,x//2,y//2,ac.color.WHITE,font_size=16,anchor_x='center')
    typo = f"Typo: {self.typo}%"
    # drawing the rate of typo to the bottom of the view
    ac.draw_text(typo, 10, 10, ac.color.WHITE, 14)
    # a guide text to tell the users how to start the game. This will be gone once the game starts.
    ac.draw_text(self.start_text,x//2,y//1.5,ac.color.WHITE,font_size=16,anchor_x='center')
```

On_draw method() for the game View that draws all the necessary elements. This includes the all of the text and labels.

```
""" This function will be called when the user inputs any kind of keystroke """
def on_key_press(self,symbols,modifiers):
        # once the user starts typing, we start the game
        self.start=True
        # deleting the guide text
        self.start_text=""
        # whenever there is a keystroke, the sfx will play
        ac.play_sound(self.sfx,1.0)

        # iterating the sentences and user input.
        for i,c in enumerate(self.text):
            # using the try block to allow iteration of multiple inputs and tries
            try:
                # checking whether the character inputted is not the same with the one in the sentence
                if self.input.text[i]!=c:
                    # excluding the backspace key so that it doesn't trigger any events. Backspace key is stored as 65288
                    if symbols!=65288:
                        """ if backspace is not entered and the input doesn't match, we will add 1 to the wrong_input
                        and set the input box color to red to indicate that a mistake have been made.
                        """
                        self.wrong_input+=1
                        self.input._set_color(ac.color.RED)
                    # we break the loop
                    break
                # the loop then continues once the user enters the correct character.
                elif self.input.text[i]==c:
                    # whenever the input is correct, the box will stay green.
                    self.input._set_color(ac.color.GREEN)
            except:
                pass
```

The on_key_press() is called whenever the user enters a key_stroke. It is an built in function from the arcade module. Under this function is where I set the self.start = True and this means that when the user starts typing, the game starts. I also play the sfx sound here so that whenever the user types, the sound plays. The for loop here essentially checks whether the user enters the correct input and when the input is wrong, the wrong_input will be added by 1. The condtion "if symbols != 65288" basically means if the backspace is not entered, then we count it as wrong. The reason for this is that hitting backspace is also counted as a keystroke. I used the try block so that whenever the user input's wrongly, the program will not crash. A flow chart of this for loop is shown later on this report.

The on_update function for the game view.

```
""" All the logic, changes and input is written under here.
delta_time is a built in parameter that allows for real time changes to occur.  """
def on_update(self, delta_time):
    # all of the code will only run when the user enters a keystroke.
    if self.start==True:
        # to make life easier, instead of using the time module, I use delta time which is a running clock in seconds.
        self.time_taken += delta_time
        # I will keep udpating the rate of typo. I use the percentage of wrong input from the total input or characters.
        self.typo=round(self.wrong_input/self.char*100,2)

         # iterating the sentences and user input.
        for i,c in enumerate(self.text):
            # using the try block to allow iteration of multiple inputs and tries
            try:
                # checking whether the character inputted is not the same with the one in the sentence
                if self.input.text[i]!=c:
                    self.input._set_color(ac.color.RED)
                    # we break the loop
                    break
                # the loop then continues once the user enters the correct character.
                elif self.input.text[i]==c:
                    # whenever the input is correct, the box will stay green.
                    self.input._set_color(ac.color.GREEN)
            except:
                pass
```

The very first thing is the condition, self.start==True. This function will run if and only if the self.start==True and self.start is only equal to true if the on_key_pressed() function is called by entering a key stroke. Here I also equate the time_taken to delta_time as delta_time is already real time. I was going ot use the time module but, it will take more coding which can affect the performance. For simplicity, delta_time is used and anyways, it still gives the same thing. We also type in the formula for the rate of typo which is the percentage of the wrong input from the total number of characters. The rate of typo will be shown constantly thorughout the gameplay. The for loop follows the previous for loop for the on_key_pressed function. This for loop will keep iterating through the inputted text and the sentence shown to check if they are correct or wrong. If the inputted text is wrong, the input box will stay red. If it is the correct then the input box will be green.

```
# when the user enters the same number of characters as the sentence and the sentence is not empty
if len(self.input.text)==len(self.text) and self.text!=self.empty:
    # I will use the same iteration to go through the sentence and input text
    for i,c in enumerate(self.text):
        # using the try block to allow iteration of multiple inputs and tries
        try:
            # check whether the input matches the characters in the sentence.
            if self.input.text[i]==c:
                # if it matches we add one to correct input
                self.correct_input+=1
        # dont want any error so.....
        except:
            pass
    # reset the input box to an empty string
    self.input.text=self.empty
    # reset the text
    self.text=self.empty
    # subtrac one from the number of sentences
    self.sentences-=1
```

The next sub condition is if the number of characters inputted by the user is the same as the number of characters of the sentence shown and the sentence is not empty. If this condition is met, we will again iterate through the characters of the user_input and compare them with the characters in the sentence. Now this might seem like the same for loop as the one in the on_key_pressed function. However this is different as the one in the on_key_pressed function runs everytime the user enters a keystroke but this for loop runs when the user has inputted all of the characters for the sentence. In this for loop, we are just counting the number of correct input by checking again whether the characters inputted by the user is the same as the sentence. We also then reset the input text to an empty string and the sentence to an empty string. We also deduct 1 from the number of sentences. The flowchart for this for loop is also shown later on this report.

```
# when everything is resetted
if self.text==self.empty and self.sentences!=0:
    # we randomize again the sentence
    self.text=random.choice(sentences)
    # drawing the next sentence
    GameView().on_draw()
    # resetting the color of the input box
    self.input._set_color(ac.color.WHITE)
    # adding the length of characters of the sentence to the total number of characters
    self.char+=len(self.text)
```

This code is to make sure that whenever a sentence is done and the sentence left is not zero, the next sentence will be drawn into the view but before that we randomize the sentence again. This is why we set the self.text equal to random.choice(sentences) again to get a new randomized sentence. We also set the input box color to white and then add the number of characters of the sentence to the self.char attribute.

```
        # when the game is over
        if self.sentences==0:
            # calculating the speed of typing
            self.wpm=(self.char/5)//(self.time_taken/60)
            # calculating the accuracy of typing
            self.accuracy=(self.correct_input//self.char)*100
            game_over_view = GameOverView()
            # setting the attributes' values for the game_over_view with the already calculated attribute from the game view.
            game_over_view.time_taken = self.time_taken
            game_over_view.wpm=self.wpm
            game_over_view.accuracy=self.accuracy
            game_over_view.char=self.char
            game_over_view.correct_input=self.correct_input
            game_over_view.typo=self.typo
            # change view to the game over view
            self.window.show_view(game_over_view)
```

This code runs when there are no more sentences left, meaning the game is over. When this happens, we calculate the speed (self.wpm) and the accuracy (self.accuracy). It is said that the average word has 5 characters and that's why we divided the number of characters by 5 to get the number of words. I then divide this by the time_taken in minutes. For the accuracy I just simply count the percentage of correct_input from the total number of characters typed in. I also set the attributes of the gameover view to the attributes of the finished game view so that the game over view can display the statistics of the user. I also change the view to the game over view by using the show_view() method.

```
class GameOverView(ac.View):
    def __init__(self):
        super().__init__()
        # since we have set the values to be the same as the one in the game view, they will later on change accordingly.
        self.time_taken = 0
        self.wpm=0
        self.accuracy=0
        self.char=0
        self.correct_input=0
        self.typo=0
        # setting up the sprites for the rank
        self.emerald=ac.sprite.Sprite('Emerald Gem05.png',center_x=self.window.width//2,center_y=self.window.height//2)
        self.ruby=ac.sprite.Sprite('Ruby Gem_5.png',center_x=self.window.width//2,center_y=self.window.height//2)
        self.sapphire=ac.sprite.Sprite('Sapphire Gem05.png',center_x=self.window.width//2,center_y=self.window.height//2)
        self.topaz=ac.sprite.Sprite('Topaz Gem10.png',center_x=self.window.width//2,center_y=self.window.height//2)
        self.rainbow=ac.sprite.Sprite('round_gem.png',center_x=self.window.width//2,center_y=self.window.height//1.75,scale=0.25)
        self.blue_gem=ac.sprite.Sprite('Aquamarine Gem05.png',center_x=self.window.width//2,center_y=self.window.height//2)
```

Initializing the elements and sprites for the game over view. We set the values of the attributes as 0 at first but this will change because we set the attributes of these equal to the ones of the Game View().

```
""" Called once when view is activated. """
def on_show(self):
    ac.set_background_color(ac.color.BLACK)

""" function used to type the rank easier since all of them have the same format """
def rank(self,text):
    ac.draw_text(text,self.window.width/2,self.window.height//1.45,ac.color.NEON_CARROT,font_size=15,anchor_x='center',align='center')
```

Usually whenver there is a on_show_view() function, there will be the on_hide_view() function. This time I didn't add the on_hide_view() function because there are no GUI elements in this view. The rank function is used to write down the rank that the user obtained in a much easier way as the text will be drawn in the same place with the same formats all the time.

```
    """The on_draw method is called whenever we draw text,sprites and etc to the current view."""
    def on_draw(self):
        ac.start_render()
        # rounding off the numbers to two decimal places
        time_taken_formatted = f"{round(self.time_taken, 2)} seconds"
        typo_formatted = f"{round(self.typo, 2)} %"

        # drawing the necessary text to show the results
        ac.draw_text("Game Over",self.window.width//2, self.window.height//1.25, ac.color.WHITE, 54,anchor_x='center')
        ac.draw_text(f"Typo Rate: {typo_formatted}",self.window.width/2,self.window.height//3,ac.color.RED,font_size=15,anchor_x="center")
        ac.draw_text("Click to restart", self.window.width//2, self.window.height//2.5, ac.color.WHITE, 24,anchor_x='center')
        ac.draw_text(f"Time taken: {time_taken_formatted}",self.window.width/2,self.window.height//3.5,ac.color.GREEN,font_size=15,anchor_x="center")
        ac.draw_text(f"Words per minute (WPM): {self.wpm}",self.window.width/2,self.window.height//4.5,ac.color.BLUEBONNET,font_size=15,anchor_x='center')
        ac.draw_text(f"Accuracy: {self.accuracy}%",self.window.width/2,self.window.height//6,ac.color.YELLOW,font_size=15,anchor_x='center')
        ac.draw_text(f"Characters: {self.char}",self.window.width/2,self.window.height//9.5,ac.color.NEON_CARROT,font_size=15,anchor_x='center')
```

The on_draw method for the game_over view. Here is where all of the statistics are drawn. I also rounded off the time and the rate of typo to 2 decimal places so that they don't recur.

```
        # if typo rate is smaller or equal to 30 and typing speed smaller or equal to 40 but not 0
        if self.typo<=30.00 and 40>=self.wpm>0:
            # draw the emerald sprite
            self.emerald.draw()
            # draw the respective texts
            GameOverView().rank("Rank: Emerald \n Not bad, You are actually decent.")
        # if typo rate is smaller or equal to 20 and typing speed smaller or equal to 60 but not 0
        elif self.typo<=20.00 and 60>=self.wpm>0:
            # draw the topaz sprite
            self.topaz.draw()
            GameOverView().rank("Rank: Topaz \n What emerald????")
        # if typo rate is smaller or equal to 10 and typing speed smaller or equal to 70 but not 0
        elif self.typo<=10.00 and 70>=self.wpm>0:
            # draw the ruby sprite
            self.ruby.draw()
            GameOverView().rank("Rank: Ruby \n Hey slow down there Mcqueen. You have good typing skills ngl.")
        # if typo rate is smaller or equal to 5 and typing speed smaller or equal to 80 but not 0
        elif self.typo<=5.00 and 80>=self.wpm>0:
            # draw the sapphire sprite
            self.sapphire.draw()
            GameOverView().rank("Rank: Sapphire \n I wonder what are the things that you do to have these speedy fingers.")
        # if typo rate is smaller or equal to 3 and typing speed smaller or equal to 100 but not 0
        elif self.typo<=3.00 and 100>=self.wpm>0:
            # draw the blue_gem sprite
            self.blue_gem.draw()
            GameOverView().rank("Rank: Blue Gem \n Ok time to dropout and join a pro team.")
        # if typo rate is equal to 0 and typing speed greater or equal to 120
        elif self.typo==0.00 and self.wpm>=120:
            # draw the rainbow sprite
            self.rainbow.draw()
            GameOverView().rank("Rank: Diamond Fingers \n Ayy what this god doing down on earth yeeeeeee. Congrats you have found the missing gem!")

    # changing the view to the Setup View when the mouse is clicked
    def on_mouse_press(self, _x, _y, _button, _modifiers):
        self.window.show_view(SetupView())
```

This is also part of the on_draw function. This essentially draws a gem sprite according to the rank of the user. A text will also be drawn to show which rank the user obtained. This is why I use the rank function that draws text because the text for this will have the same format and this makes it easier and more efficient. The rank is as follows:

1. Blue Gem → self.typo<=3.00 and 100>=self.wpm>0
2. Sapphire →self.typo<=5.00 and 80>=self.wpm>0
3. Ruby →self.typo<=10.00 and 70>=self.wpm>0
4. Topaz →self.typo<=20.00 and 60>=self.wpm>0
5. Emerald →self.typo<=30.00 and 40>=self.wpm>0
6. Hidden Rainbow Rank →self.typo==0.00 and self.wpm>=120

The on_mouse_press() function is called whenever the user dicks on this mouse. This will change the view to the Setup view

```
#arcade gui button class that we are going to be using for all of the buttons in the game
class Button(ac.gui.UIFlatButton):
    #This function triggers when the buttons are pressed
    def on_click(self):
        #using conditionals to trigger different events when different buttons are pressed
        #play button
        if self.text.lower()=="play":
            ac.View().window.show_view(SetupView())
        #quit button
        elif self.text.lower()=='quit':
            ac.View().window.close()
        #start button
        elif self.text.lower()=="start":
            """This start button is placed in the Instruction window where the user set up the number of sentences.
            By using the try block, I can check wether the user input is in the correct format.
            I want the number of sentences to be in a integer fromat and that's why I used the isdigit() method to check that.
            Once the correct format is inputted, then the event will be triggered
            """
            try:
                if number.isdigit()==True:
                    # if the user input is an integer, switch to the game view or scene.
                    ac.View().window.show_view(GameView())
                else:
                    # if the user input is not an integer, the global varibale error will be set to true. I will explain what this do late on.
                    global error
                    error=True
            except ValueError:
                #when a noninteger is inputted, python will cause an error, but I want the program to keep running. Hence the 'pass'.
                pass

        #back button
        elif self.text.lower()=="back":
            ac.View().window.show_view(MainMenu())

        #how to play button
        elif self.text.lower()=="how to play":
            ac.View().window.show_view(HowToPlay())
```

This is the built in button class that allows me to add GUI elements such as the button in order for users to interact with my program. There is only the on_click function and this on_click function triggers events whenever the buttons are pressed. In order to contrd the events being triggered, I used conditionals to check which button is pressed by comparing the text written on the button. If the text says 'play' we switch to the setup view, if the text is 'quit' then we close the game and so on. The text is change to a lower case using the lower() method so that we don't have to worry about any capitalizations. For the start_button we have a special try block that is used to check whether the user have inputted an integer to the input box in the Setup View where it is asked to input the number of sentences. If it is an integer, we switch the view to the game view. If it is not an integer then we change the global variable, "error" to True. When the

global variable “error” is set to True the on_draw method of the Setup view will draw the error message to the view. The value error is used for the except block because the value error indicates that a wrong datatype have been inputted.

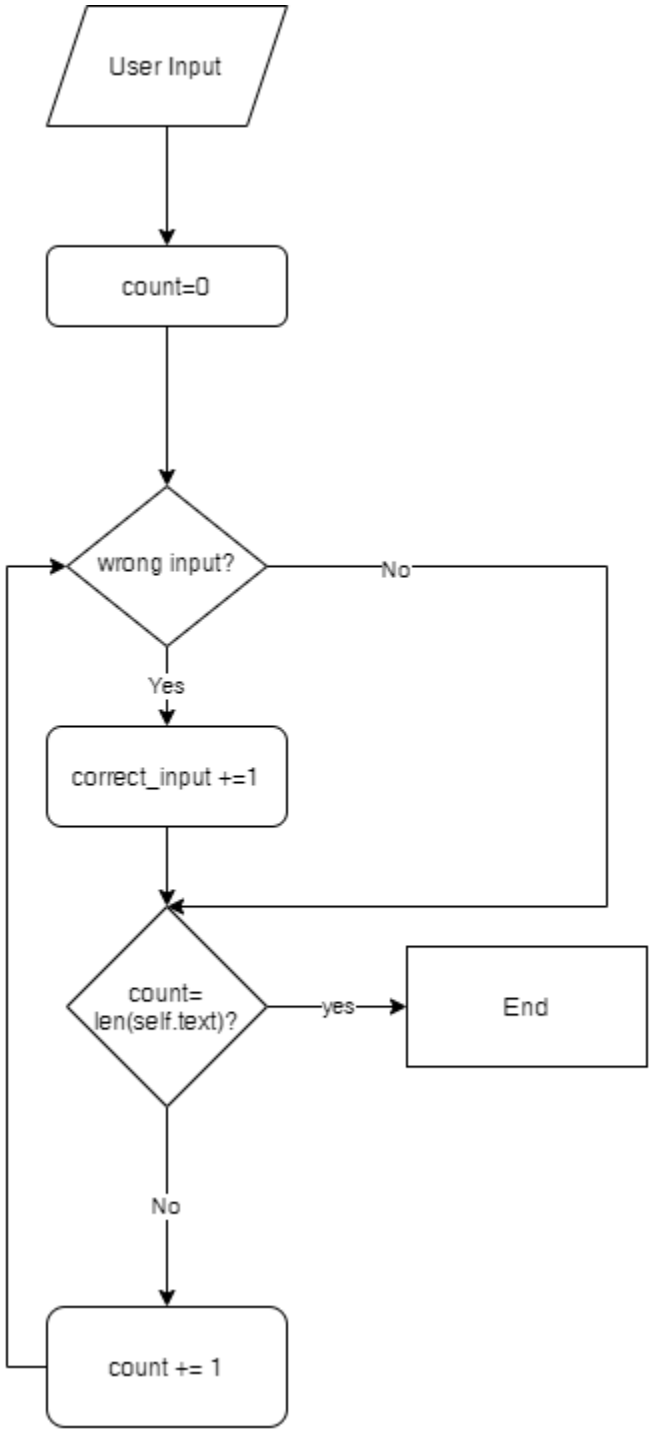
```
# running the program
if __name__ == '__main__':
    window = ac.Window(SCREEN_WIDTH,SCREEN_HEIGHT,SCREEN_TITLE,resizable=False)
    view = HowToPlay()
    window.show_view(view)
    ac.run()

f.close()
```

Running the program using the arcade module. Ac.Window is used to set up a window and this takes in the width, height and title of the window we want to create. The resizable attribute is just for whether we want the window to be resizable or not and in this case I set it to False. We then use the show_view method to show the wanted view and then use ac.run() to run the program.

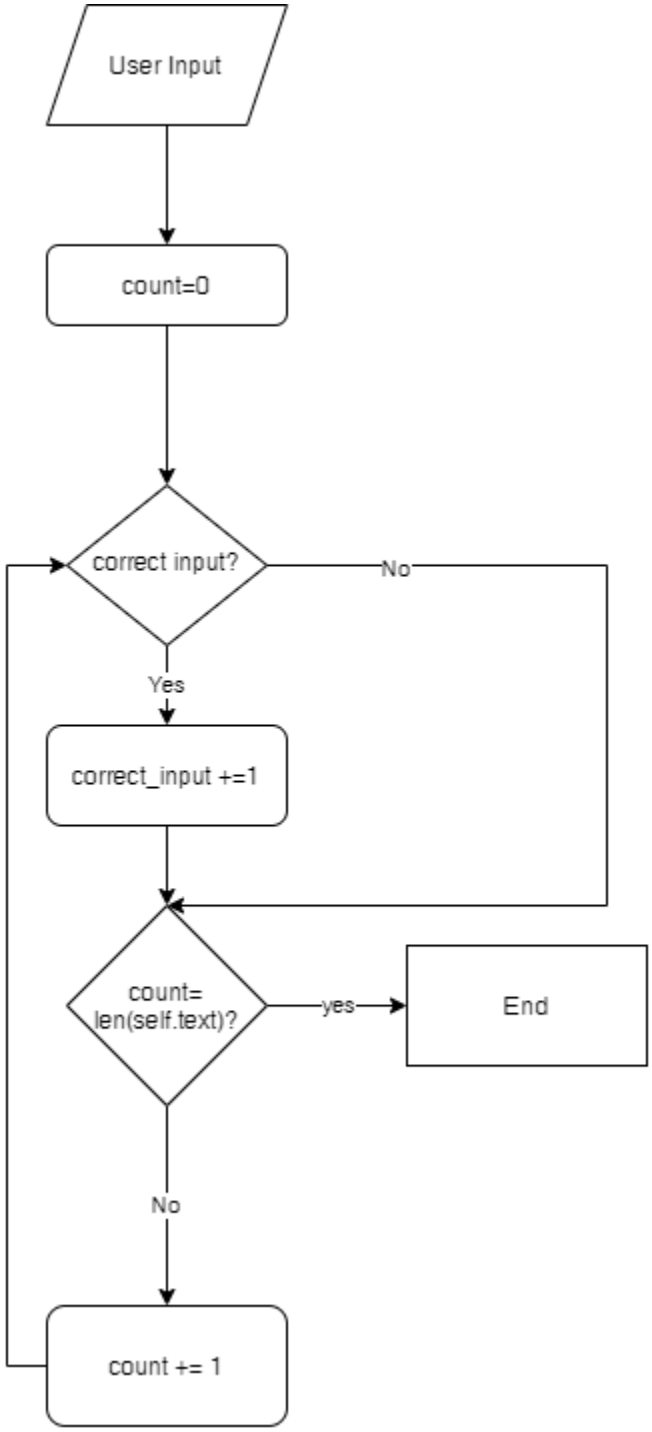
```
import arcade as ac
import random
import arcade.gui
from arcade.gui import UIManager
from arcade.gui.ui_style import UIStyle

SCREEN_WIDTH = 1460
SCREEN_HEIGHT = 720
SCREEN_TITLE = "TYPO"
# opening the external text file for the sentences
f=open('sentences.txt',mode='r',encoding='utf-8')
# split the text into a list of sentences
sentences=f.read().split('\n')
# global variable that will store the number of sentences inputted by the user
number=""
# the global variable that will show whether the user has inputted an error
error=False
```

These are the global variables and constants that are used in this program. I used the sentences.txt file which contains 100 random sentences for randomizing and drawing the sentence for the game view. I also used the split() method to split the sentences into a list so that it will be more accessible later on when I will be getting the data.

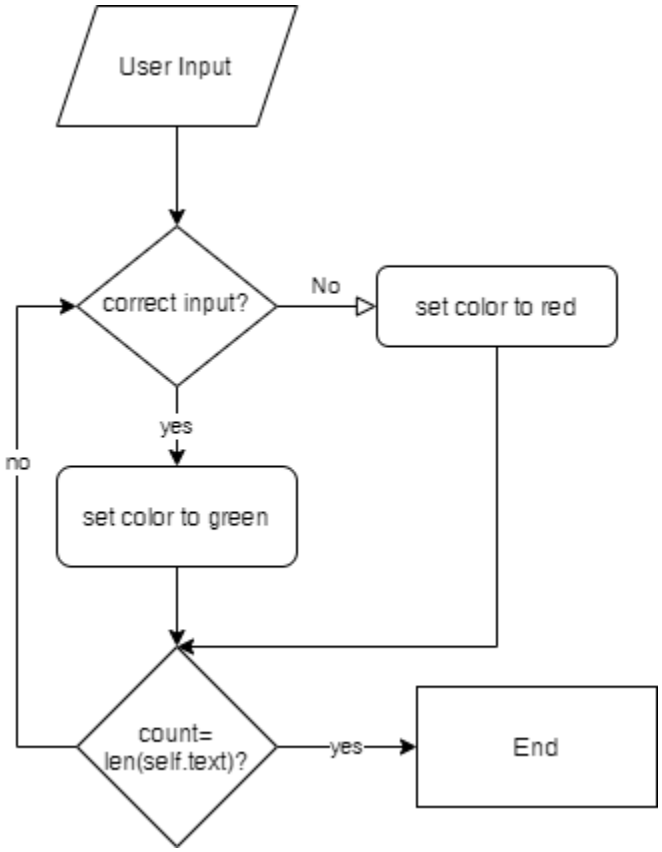
Flowchart for counting the wrong input:



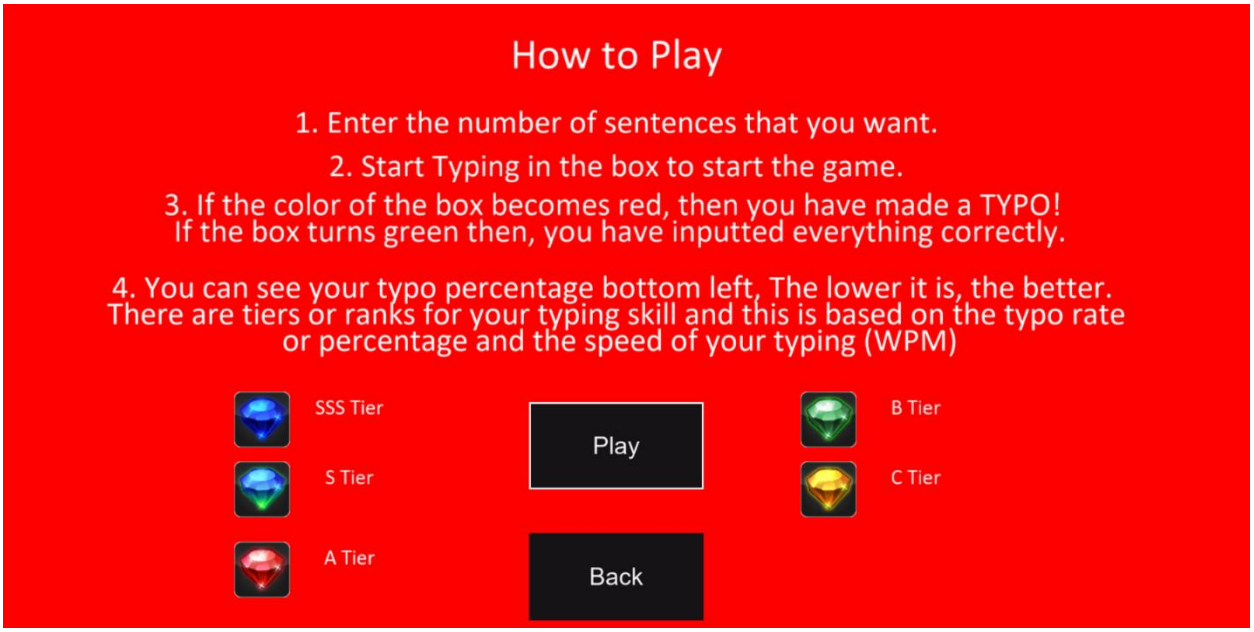
Flowchart for counting the correct input:

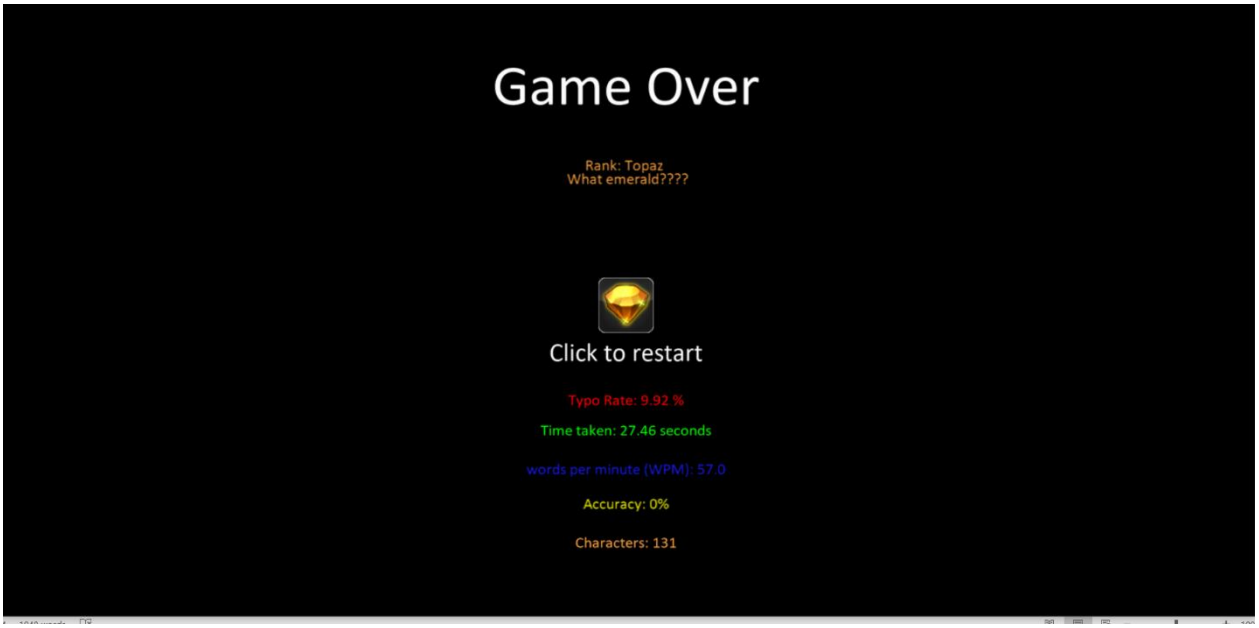


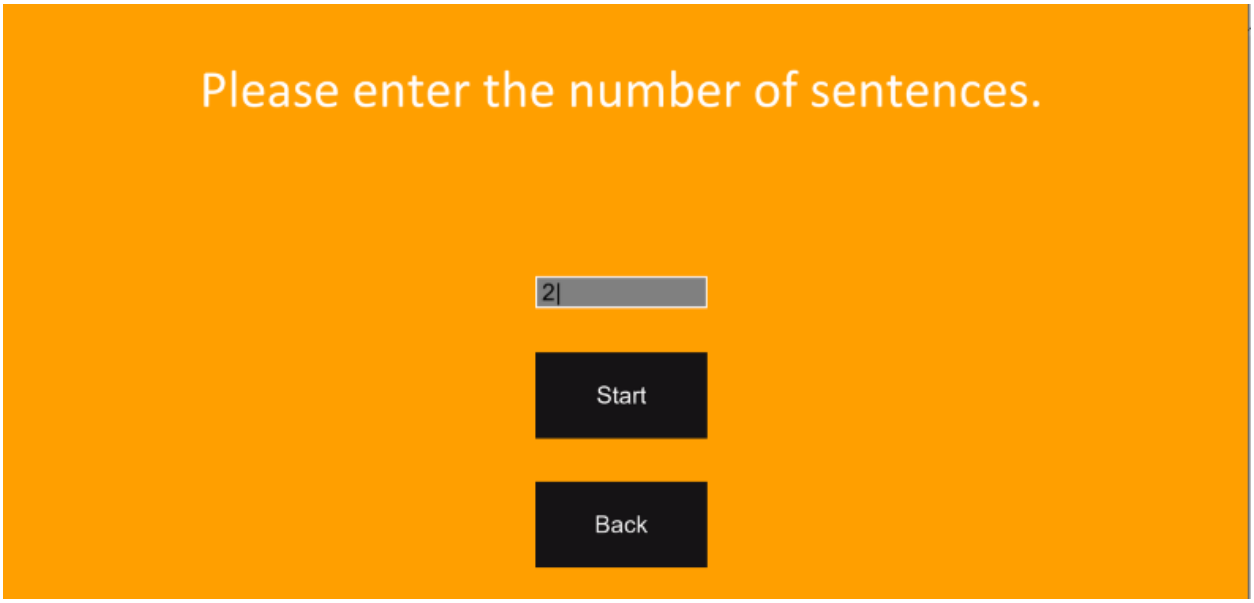
Flowchart for changing the input box color:

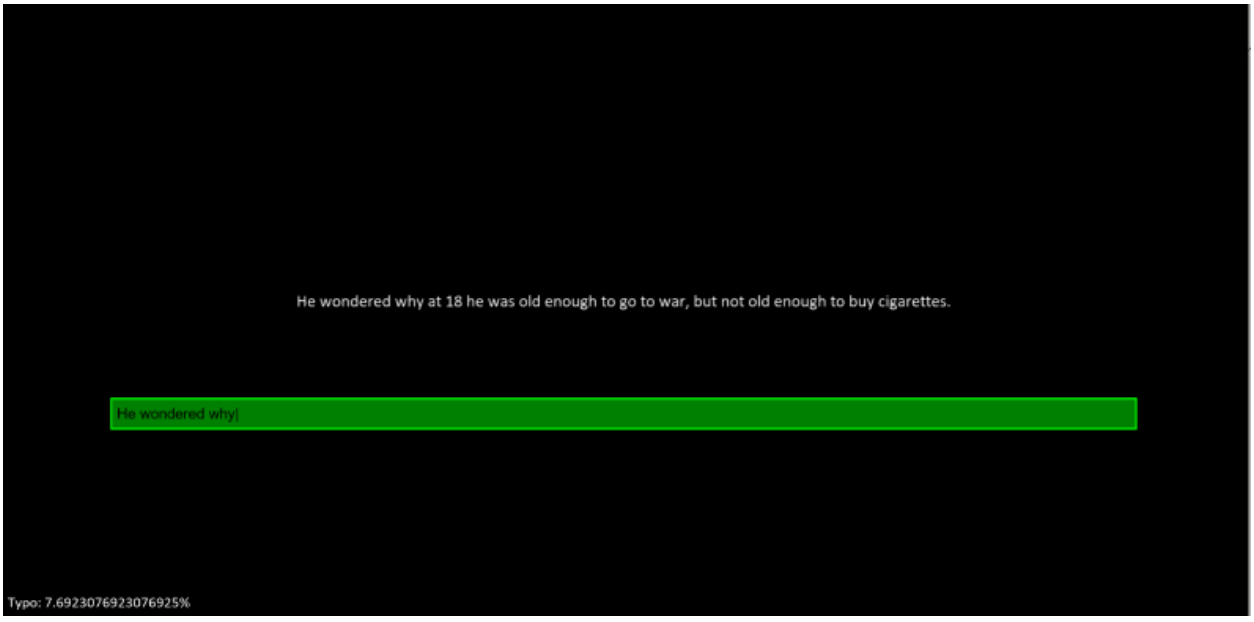


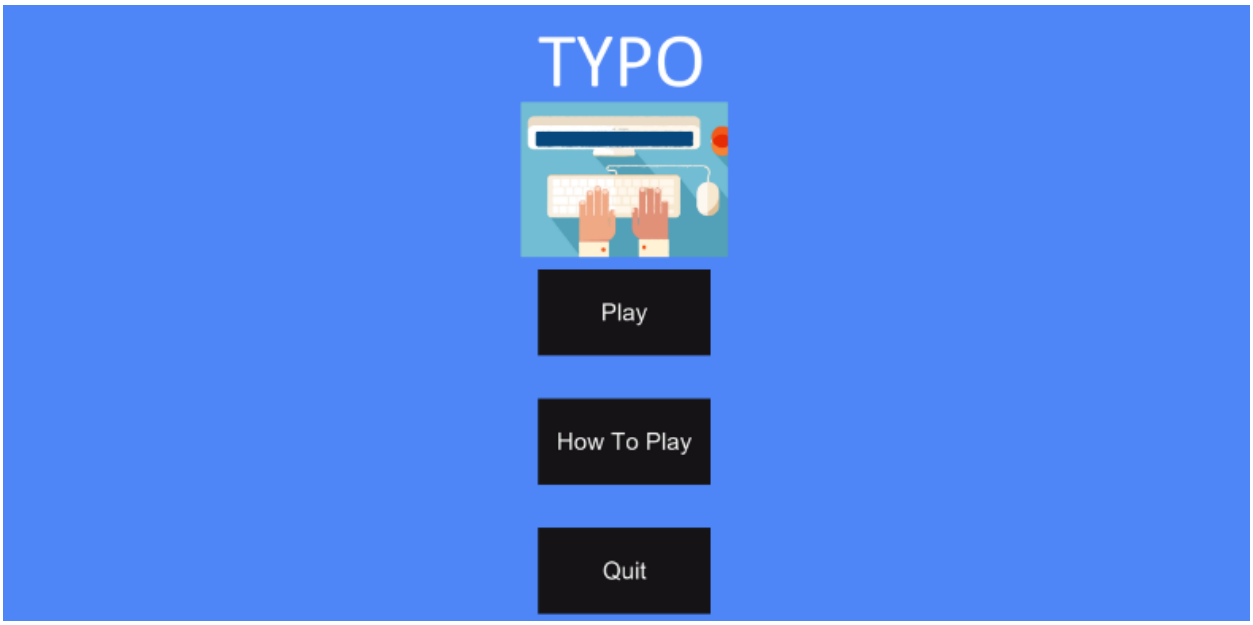
PROOF OF WORKING PROGRAM






Please enter the number of sentences.
2
Start
Back


He wondered why at 18 he was old enough to go to war, but not old enough to buy cigarettes.
He wondered why
Typo: 7.6923076923076925%



Reflection and Experience:

At first glance, this project was hard for me, firstly because I couldn't find a doable idea that matches my skill set as a programmer at that time. Sophisticated ideas definitely came into mind but I didn't know whether I was capable enough to do the challenge. I thought of a simple type racer but added my twist to it which is the typo rate. I realized that I had to add something to make this game more intriguing as a type racer on it's own isn't that fun to play. The arcade documentation at first glance seems pretty straightforward and I didn't have any problems untill my arcade for some reason encountered an error when I tried running my program. A simple reinstall is what I thought would solve this issue but it turned into a night mare as I couldn't reinstall the arcade module from pip. I tried using the virtual environment as suggested by Sir Baguz but it still didn't work. I tried to dig it to the internet but I can't seem to find anything. I tried analyzing the error and turns out that the error triggered when it was trying to install the numpy module. I just uninstalled the numpy module and let the installing of the arcade install the numpy version I needed for arcade. Turns out it still wasn't working and I even tried reinstalling my numpy and it also didn't work. I soon realize maybe the version of pip and python wasn't compatible with the older version of numpy the system was trying to isntall. So, I deleted my python and downloaded an older version which also gave an older version of pip and everything was back to normal.

I was definitely frustrated at this time but, I figured that you should always try to as calm as possible in order to come up with a solution to fix the problem. Another issue that I faced was the input box was blinking in and out of the window. I tried to check my code for the setting up of the input box and I can't seem to find an issue. I then realized that I have put the finish render function after all of the elements drawn in the on_draw function which means that the setup function where I setup the input box doesn't get rendered hence resulting in the blinking. I just deleted the finish_render function so that it keeps rendering all of the elements being put on the screen. I also put a ranking system at the end to grade people's typing skills based on their speed, time taken, typo rate and accuracy. It was these small features that I added to the game to make it more interesting but at the same time achieve the minnamhistic design and mechanics that I was trying to achieve.

Overall, this project have put me through ups and downs, but I guess one big lesson that I got from this porject is how to handle and deal with the downs which creates the whole programming experiencing more realistic and enjoyable once you cope with it. Sir Baguz has also been a really great professor and definitely have taught me well in class which I am really appreciative of because a big part of my comprehensive understanding of the programming world comes from his class.

References:

https://app.diagrams.net/

https://arcade.academy/

https://opengameart.org/content/basic-gems-icon-set

https://www.thewordfinder.com/random-sentence-generator/

https://www.fesliyanstudios.com/royalty-free-sound-effects-download/keyboard-typing-6

https://id.pinterest.com/pin/375206212681302825/

https://www.w3schools.com/