

# 

[**Title of Ownership 2**](#_heading=h.rd2i144uq8bp)

[**Analysis 3**](#_heading=h.w1zeegca9u62)

[**Algorithm (Pseudo code or flowchart) 4**](#_heading=h.ybwofxbic89k)

[**Technical Overview 5**](#_heading=h.1fob9te)

[**Coded Solution 13**](#_heading=h.eik77kdrfnxt)

[**Testing to inform development 35**](#_heading=h.m0ks9gtc2jf9)

[**Testing to inform evaluation 39**](#_heading=h.ytrijijanfdi)

[**Evaluation of Solution 41**](#_heading=h.tyjcwt)

[**Reference list 42**](#_heading=h.mzvguyr20v1h)

# 

# 

# Title of Ownership

I, Saul Ye Zheng , a student at Kaplan International College, am the rightful owner of the intellectual property rights associated with the assignment titled "Programming Assignment 1," completed on April 9th 2023. This work is original and has not been copied or otherwise plagiarised from any other source. As the author of this work, I retain all rights to it, including the right to reproduce, distribute, and display the work. No other person or entity has any claim of ownership or intellectual property rights in this work without my express permission.

# Analysis

Khalid needs to create a hangman game for the company he works for. Therefore, the first step is to analyse what the game must require.

The program must have:

* A login screen that allows the user to register, if he or she does not already have a user, and sign-in if the user is already registered. This login screen should return feedback to the user to let them know if their username or password is invalid.
* User interface
* A way to store usernames and passwords. For this function, text files can be used, but if Khalid desires to take a step further, he can create a database to keep track of the usernames, passwords, and scores with Sqlite3.
* Main menu that allows the player to start a new game. In addition, the main menu can have more functionality by adding more buttons that take the user to different screens. For instance a leader board button might be suitable for the main menu.
* A leaderboard screen. In this screen a chart can be shown with the data stored within the database. The chart must display in order the users with the highest scores.
* A topic selector screen. In this screen the user can choose what topic he wants the secret word to be related to.
* Sounds to make the game more interactive.
* An option to play again when the game ends

Taking into account these requirements, it is optimal to do a decomposition of the back-end of the program and then work on the front-end part of the game.

**Stakeholders:**

Users: The users are the main stakeholders of this project. Users will care mainly about the frontend of the program. Khalid must develop code for a fun game with an attractive front end. In order to satisfy the users, Khalid must create a user friendly UI (User Interface) for the game. Extra features like sounds/music are important to this stakeholder. Also, the game should have the least amount of bugs possible to provide a good user experience and keep this stakeholder satisfied.

Boss/company: This stakeholder cares about efficiency and about the users satisfaction. For this stakeholder, the minimal requirements must be met and extra features are secondary.

# 

# 

# 

# Algorithm (Pseudo code or flowchart)

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Technical Overview

Libraries:

* pygame: python library that allows the creation of games in python. This library was used to create the hangman game.
* pygame\_gui: python library that is used to create a GUI for games created with pygame. This library was used for the UI and front end of the game, since it already contains tools to create buttons and text inputs.
* random: python library used to generate random numbers or choices. For the following code the random.choice() function from the random library was used to randomly choose a word for the hangman game.
* pathlib: python library used to fetch the directory location of files. This library was used to adapt the code for it to be able to run on different computers.
* sys: python library that contains multiple fonts. This library was used to display different messages on the screen.
* sqlite3: python library used to create databases. Sqlite3 was useful to store the usernames, passwords, and their scores. Databases are an extra feature for the program, however they were implemented due to the fact that they are more useful to push and pull data.

Global Variables:

#Path files: These variables are used to fetch the location of different.All of them take the first part pathlib.Path(\_\_file\_\_).resolve().parent/ ‘file\_name’ as a string.

base\_path = pathlib.Path(\_\_file\_\_).resolve().parent

audio\_path = base\_path / 'bonk.mp3'

play\_button\_path = base\_path / 'button\_play.png'

exit\_button\_path = base\_path / 'button\_exit.png'

yes\_button\_path = base\_path / 'button\_yes.png'

no\_button\_path = base\_path / 'button\_no.png'

leader\_board\_path = base\_path / 'button\_score.png'

db\_path = base\_path/ 'logins.db'

image\_folder = base\_path/"hangman"

music\_path = base\_path/''

exit\_sound\_path = base\_path/"exit.mp3"

fail\_sign\_sound\_path = base\_path/"wah.mp3"

omg\_sound\_path = base\_path/"omg.mp3"

spiderman\_sound\_path = base\_path/ "spiderman.mp3"

#Variables that store a tuple that store three integers. This tuple defines the RGB code for each color.

white = (255, 255, 255)

black = (0, 0, 0)

cyan = (204, 255, 255)

# WIDTH and HEIGHT are constants that store the size of the screen for the display. This constants contain integer values

WIDTH, HEIGHT = 800, 500

# The my\_screen variable stores a surface(part of the pygame library) with the size previously stated.

my\_screen = pygame.display.set\_mode((WIDTH, HEIGHT))

#Variable that store the class clock(part of pygame library)

timer = pygame.time.Clock()

#Constants containing integers. These variables are used to genereate the buttons for the game.

ROWS = 3

COLUMNS = 13

GAP = 20

SIZE = 40

A = 65

#Variables to store the Sys fonts. As sys library is used to write different messages on screen, they need to be stored in a variable in order to use them. These variables store the SysFont() function that takes in two parameters. A string e.g. ‘arial’ that specifies the font style and an integer to specify the size of the letter.

btn\_font = pygame.font.SysFont('arial', 30)

game\_font = pygame.font.SysFont('arial', 50)

letter\_font = pygame.font.SysFont('arial', 40)

fail\_font = pygame.font.SysFont('arial', 20)

# These variables are used to store a pygame sound for the game. The variables consists of pygame modules(‘file string’)

doh = pygame.mixer.Sound(audio\_path)

exit\_sound = pygame.mixer.Sound(exit\_sound\_path)

fail\_sign\_sound = pygame.mixer.Sound(fail\_sign\_sound\_path)

omg\_sound = pygame.mixer.Sound(omg\_sound\_path)

spideerman\_sound = pygame.mixer.Sound(spiderman\_sound\_path)

#The word constant is used to store the random word that the user must guess. It is initially set as an empty string. In the word choice menu, this variable is changed into a random word chosen with the random.choice() function.

WORD = ''

#globalUsername is an empty string that initially is set as an empty string. The purpose of the variable is to store the username to fetch the score of the username from the database. When the sign\_in() function is called, the variable will be changed to the user that successfully signed in.

globalUsername = ""

#title variable stores a string. The purpose of this variable is to store the message that will be displayed in the game screen.

title = "Average college student"

#title\_text variable is surface(pygame library) that contains the order to draw/render a message into the screen of the game.

title\_text = game\_font.render(title, True, black)

#title\_rect variable is a rect(pygame library) that sets a rectangle to the title\_text. The variable sets the x and y coordinates for the rectangle. Bothe the surface and the rectangle are needed to draw a message with pygames. Since the .blit function takes two parameters e.g. my\_screen.blit(surface,rect)

title\_rect = title\_text.get\_rect(center=(WIDTH // 2, title\_text.get\_height() // 2))

#Constant SCORE is used to store the player's score. This variable is initially set to 0 and increases when the user guesses a word correctly.

SCORE = 0

Data structures:

#The BOXES list stores the coordinates and dimensions of rectangular boxes on the game screen. These boxes represent the spaces where the letters of the secret word will be displayed.

BOXES = []

#The BUTTONS list stores the buttons for each letter. The BOXES list itself stores the rectangles for the buttons stored in this list.

BUTTONS = []

#The GUESS list stores the letters that the user guessed correctly. If the letter is in the secret word, it will be added to this list.

GUESS = []

#This list stores the images for the hangman. The images are the different stages of the hangman.

IMAGES = []

Class:

The Button class is used to create pygame buttons and rectangles using a png file. The class has a \_\_init\_\_ method that takes x,y coordinates, scale to modify the size of the image, and an image. In addition this class has a function to draw the button into the pygame surface.

class Button():

Functions:

#Name: sign\_up

#Description: this function uses sqlite3 to establish connection to a database. It takes in two parameters and registers them into the ‘logins.db’ database. If the username does not exist inside the database it will push the data and store it in. If the username already exists, the function will return a boolean False and “username already exists”.

#Inputs: (username, password) strings

#Outputs: If successful the function will print “Sign in successful” and return True. Else it will return “username already exists”.

def sign\_up(username, password):

#Name: sign\_in

#Description: This function takes two parameters (username and password) and establishes a connection to the ‘logins.db’ database and searches for the username and password columns. If the inputs are inside the database and are a match, the function returns a True boolean. Else, it prints “invalid username or password.

#Inputs: strings (username, password)

#Outputs: True or “invalid username or password”

def sign\_in(username, password):

#Name: update\_score

#Description: The function uses a sqlite3 query to update a set. The set consists of the username and score. When the function is called the score of the user will be updated.

#Inputs: Score, type: integer

#Output: Booleans True and print “Score successfully updated” if the function works. Else it will print “Error updating score” and give an error in the terminal.

def update\_score(score):

#Name: leaderboard\_screen

#Description: This function pulls data from the logins.db database and uses it to draw a leaderboard using a pygame screen. Within this screen the user can view the highest scores or exit back to the main menu. To achieve this a button is created with a scaled image if the clicked position collides with the coordinates of the button, the user will go to the main menu.

#Inputs: None

#Outputs: None

def leaderboard\_screen():

#Name: draw\_buttons

#Description: This function is used to render all the buttons into the screen. From the BUTTONS lists, it retrieves each letter, creates a rectangle using pygame.get\_rect() function, and draws them into the screen.

#Inputs: None

#Outputs: None

def draw\_buttons():

#Name: draw\_gues()

#Description: The draw\_guess function displays the letters guessed by the user. If the letter is in the secret word, that is stored in the WORD variable, the letter will be added to the GUESS [] list. The items inside the list are rendered into the screen.

#Inputs: None

#Outputs: None

def draw\_guess():

#Name: word\_choice\_menu()

#Description: This function creates a screen where the user can choose the topic for the word. The screen uses pygame\_gui to create two buttons for the topics. If the car brands button is clicked, the game will choose a random word from the car brand list and set the WORD variable to it. Whereas, if the car parts button is clicked the word will be selected from the car parts list. After clicking any of the two buttons, the function will call the game() function to advance into the next screen. Within the main loop the variable called “running” is used to keep this screen running. The running will turn false until one of the buttons is pressed.

#Inputs: None

#Outputs: None

def word\_choice\_menu():

#Name: game()

#Description: The game function holds the loop for the main part of the program. In this screen the buttons, hangman images, secret word, and guessed letters are drawn.The game function starts by defining a “lives” variable that is set to 0 and a game\_is\_over variable set to False. Then, the function enters a while loop until the game is over. Inside the loop every pygame event is handled. With the pygame.event.get() function the user inputs are registered. If the mouse is clicked, the coordinates of the click are registered. Also, if the clicked position collides with a button there are two scenarios. If the button is for a letter that is in the secret word, the letter will be drawn into the screen and added to the GUESS[] list. Else if the letter is not in the secret word an audio will be played to let the user know he lost a life, the hangman image will change into the next stage, and 1 += will be added to the lives counter. The game will end when the user guesses the word or if the life counter reaches 6. After ending the game, a new screen will be shown. This screen will render two buttons to ask the user if he wants to continue playing. If the user desires to continue playing, he will be taken back to the word\_choice\_menu and continue playing. If the user does not want to continue, his score will be updated and will go back to the main menu where he can quit the program.

#Inputs: None

#Outputs: None

def game():

#Name: main\_menu

#Description: This function is for the second screen of the program. In this screen a play, exit, and leaderboard button are drawn using the Button class. These buttons take the user to different screens. The play button moves into the word\_choice\_menu to continue with the game. The exit button goes back into the login menu. And finally the leaderboard button takes the user to the leaderboard screen, where the users with the highest score are displayed. The main loop runs constantly until it receives input from the user. The loop registers the clicked positions and collide points.

#Inputs: None

#Outputs: None

def main\_menu():

#Name: login

#Description: This function handles the login screen for the program. With the implementation of pygmaGUi two text boxes are created to store the input from the user for the username and password. Also, two buttons are drawn into the screen to sign in or to register a new user. If the signin button is clicked, the sign\_in() function is called to check if the username and password are registered.If the username and password are registered, the program moves into the next screen. If the signup button is pressed, the values given from the user are stored into the database and a new user is created. In addition a boolean variable called “fail\_sign” that is initially set to False is used to give feedback to the user. If fail\_sign = True a message saying “Invalid username or password” will be drawn.

#Inputs:None

#Outputs:None

def login():

For loops:

#This for loop is used to append all the images for the different stages of the hangman into the IMAGES list. The directory for all the images is the same, the only value that changes is i, so a for loop and f’ string is used to automate the process instead of appending manually each photo.

for i in range(7):

image = pygame.image.load(f"{image\_folder}{i}.png")

IMAGES.append(image)

# This nested loop generates a grid of rectangular boxes into the screen. The x and y coordinates are calculated within the inner loop and stored into the BOXES list

for row in range(ROWS):

for column in range(COLUMNS):

x = ((column \* GAP) + GAP) + (SIZE \* column)

y = ((row \* GAP) + GAP) + (SIZE \* row) + 360

box = pygame.Rect(x, y, SIZE, SIZE)

BOXES.append(box)

#This loop uses the boxes generated by the previous loop to convert the index into characters(letters) with the chr() function, and also appends it into the BOXES list.

for ind, box in enumerate(BOXES):

letter = chr(A + ind)

button = [box, letter]

BUTTONS.append(button)

Local Variables:

#The conn variable is a local variable that is used inside all the functions that involve setting a connection to the database:

sign\_up

sign\_in

update\_score

leaderboard\_screen

The variable uses the connect() function of sqlite3 and takes in the directory of the file.

conn = sqlite3.connect(str(db\_path))

#cursor variable stores the sqlite cursor, that is used to execute sqlite3 commands

cursor = conn.cursor()

# 

# Coded Solution

import pygame

import sys

import pygame\_gui

import random

import sqlite3

import pathlib

# Paths to make the program run smoothly on any computer

# This line of code was adapted from https://stackoverflow.com/questions/30218802/get-parent-of-current-directory-from-python-script

base\_path = pathlib.Path(\_\_file\_\_).resolve().parent

audio\_path = base\_path / 'bonk.mp3'

play\_button\_path = base\_path / 'button\_play.png'

exit\_button\_path = base\_path / 'button\_exit.png'

yes\_button\_path = base\_path / 'button\_yes.png'

no\_button\_path = base\_path / 'button\_no.png'

leader\_board\_path = base\_path / 'button\_score.png'

db\_path = base\_path/ 'logins.db'

image\_folder = base\_path/"hangman"

music\_path = base\_path/''

exit\_sound\_path = base\_path/"exit.mp3"

fail\_sign\_sound\_path = base\_path/"wah.mp3"

omg\_sound\_path = base\_path/"omg.mp3"

spiderman\_sound\_path = base\_path/ "spiderman.mp3"

# Word lists/topics for the game

cars = ["BMW", "AUDI", "MERCEDES", "LAMBORGHINI", "NISSAN", "FERRARI", "ASTONMARTIN", "MAZDA", "FORD", "CHEVROLET",

"PORSCHE", "SUBARU", "MITSUBISHI", "VOLVO", "LEXUS", "RIMAC", "PONTIAC", "DODGE", "TESLA", "RENAULT",

"MCLAREN", "MASERATI", "HONDA", "SUZUKI"]

car\_mods = ["TURBO", "TRANSMISION", "PISTON", "TIRES", "RIMS", "SUSPENSION", "ENGINE", "ROLLCAGE", "NOS", "SHIFTER",

"WIDEBODY", "OIL", "FILTER", "INTAKE"]

# Colors in RGB

white = (255, 255, 255)

black = (0, 0, 0)

cyan = (204, 255, 255)

# Window for the game

WIDTH, HEIGHT = 800, 500

my\_screen = pygame.display.set\_mode((WIDTH, HEIGHT))

#Set the caption for the game

pygame.display.set\_caption("Hangman")

# setup pygame

#Function to initialeze the pygame module

pygame.init()

#Start the audio mixer

pygame.mixer.init()

#Set timer clock

timer = pygame.time.Clock()

# GUI setup

GUI\_MANAGER = pygame\_gui.UIManager((WIDTH, HEIGHT))

# Buttons for the hangman

ROWS = 3

COLUMNS = 13

GAP = 20

SIZE = 40

BOXES = []

BUTTONS = []

A = 65

# List to store the image

IMAGES = []

#For loop to change the image and store them into the list automatically instead of doing one at a time

for i in range(7):

image = pygame.image.load(f"{image\_folder}{i}.png")

#Append the images into the list

IMAGES.append(image)

#Define a button class

class Button():

#Funtion to create button. Takes x and y coordintaes, scale to modify the size, and a png image

def \_\_init\_\_(self, x, y, scale, image):

#Get the height and width from the image

width, height = image.get\_size()

self.image = image

#scale the image

self.image = pygame.transform.scale(image, (int(width \* scale), int(height \* scale)))

#Give the image a rectangle

self.rect = self.image.get\_rect()

#Set the coordintates

self.rect.topleft = (x, y)

#Draws the images into the surface

def draw(self):

my\_screen.blit(self.image, (self.rect.x, self.rect.y))

# Generate the coordinates x and y for the boxes

for row in range(ROWS):

for column in range(COLUMNS):

x = ((column \* GAP) + GAP) + (SIZE \* column)

y = ((row \* GAP) + GAP) + (SIZE \* row) + 360

box = pygame.Rect(x, y, SIZE, SIZE)

BOXES.append(box)

for ind, box in enumerate(BOXES):

letter = chr(A + ind)

button = [box, letter]

BUTTONS.append(button)

# Audio

# Adapted from https://stackoverflow.com/questions/43845800/how-do-i-add-background-music-to-my-python-game

doh = pygame.mixer.Sound(audio\_path)

exit\_sound = pygame.mixer.Sound(exit\_sound\_path)

fail\_sign\_sound = pygame.mixer.Sound(fail\_sign\_sound\_path)

omg\_sound = pygame.mixer.Sound(omg\_sound\_path)

spideerman\_sound = pygame.mixer.Sound(spiderman\_sound\_path)

# Font

btn\_font = pygame.font.SysFont('arial', 30)

game\_font = pygame.font.SysFont('arial', 50)

letter\_font = pygame.font.SysFont('arial', 40)

fail\_font = pygame.font.SysFont('arial', 20)

# Secret word

WORD = ''

#List to store the letters that the user guessed

GUESS = []

# Username

globalUsername = ""

# Title

title = "Average college student"

title\_text = game\_font.render(title, True, black)

title\_rect = title\_text.get\_rect(center=(WIDTH // 2, title\_text.get\_height() // 2))

#Adapted from https://www.sqlite.org/lang\_insert.html

def sign\_up(username, password):

"""

This function takes in two parameters: Username and password to store them into a database.

"""

# Check if either username or password is empty

if not username or not password:

print("Username or password cannot be empty")

return False

# Establish connection to the database

conn = sqlite3.connect(str(db\_path))

cursor = conn.cursor()

# Command to insert the values into the database

insert\_query = '''

INSERT INTO logins (username, password)

VALUES (?, ?)

'''

try:

# Execute the command

cursor.execute(insert\_query, (username, password))

# Push the data into the database

conn.commit()

# If successful, print successful and return True

print("Sign up successful")

return True

except sqlite3.IntegrityError:

print("Username already exists")

finally:

# Close connection to the database

conn.close()

#Adapted from https://www.sqlite.org/lang\_insert.html

def sign\_in(username, password):

#Establish connection to the database

conn = sqlite3.connect(str(db\_path))

#Create a cursor

cursor = conn.cursor()

#Select the logins and password columns

select\_query = '''

SELECT \* FROM logins WHERE username = ? AND password = ?

'''

cursor.execute(select\_query, (username, password))

#Select one

result = cursor.fetchone()

if result:

print("Sign in successful")

global globalUsername

globalUsername = username

conn.close() # Close the connection

return True

else:

print("Invalid username or password")

conn.close() # Close the connection

def update\_score(score):

conn = sqlite3.connect(str(db\_path))

cursor = conn.cursor()

update\_query = "UPDATE logins SET score=? WHERE username=?"

try:

cursor.execute(update\_query, (score, globalUsername))

conn.commit()

print("Score updated successfully")

return True

except sqlite3.Error as e:

print("Error updating score:", e)

conn.close()

def leaderboard\_screen():

#Establish connection to database

conn = sqlite3.connect(str(db\_path))

#Create a cursor

cursor = conn.cursor()

#Command query to select from the database the pairs of usernames and score by descending order

select\_query = """

SELECT username, score FROM logins

ORDER BY score DESC

"""

#Execute the sqlite3 command

cursor.execute(select\_query)

#Select all the rows

rows = cursor.fetchall()

conn.close() # Close the connection

#Boolean to keep the loop running

running = True

while running:

#Fill the pygame screen with color cyan

my\_screen.fill(cyan)

#Title text for leaderboard screen

leaderboard\_title\_text = game\_font.render("Leaderboard", True, black)

#Rectangle for the title text

leaderboard\_title\_rect = leaderboard\_title\_text.get\_rect(center=(WIDTH // 2, 50))

#Draw the title and the title rectangle

my\_screen.blit(leaderboard\_title\_text, leaderboard\_title\_rect)

y\_pos = 150

#For loop to draw the usernames and scores in the screen

for row in rows:

username, score = row

username\_text = pygame.font.SysFont('arial', 20).render(username, True, black)

score\_text = pygame.font.SysFont('arial', 20).render(str(score), True, black)

username\_rect = username\_text.get\_rect(midleft=(150, y\_pos))

score\_rect = score\_text.get\_rect(midright=(WIDTH - 150, y\_pos))

my\_screen.blit(username\_text, username\_rect)

my\_screen.blit(score\_text, score\_rect)

y\_pos += 40

#Create a button to allow the user to exit this screen

back\_button = Button(340, 400, 0.8, pygame.image.load(str(exit\_button\_path)))

back\_button.draw()

#Update the screen

pygame.display.update()

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

#Get the coordinates of the clicks

if event.type == pygame.MOUSEBUTTONDOWN:

clicked\_position = event.pos

#If statement to see if the user clicks the back buttton and end the loop

if back\_button.rect.collidepoint(clicked\_position):

exit\_sound.play()

running = False

# Function to draw the buttons on the screen

def draw\_buttons():

for box, letter in BUTTONS:

btn\_txt = btn\_font.render(letter, True, black)

btn\_rect = btn\_txt.get\_rect(center=((box.x + 20), (box.y + 20)))

my\_screen.blit(btn\_txt, btn\_rect)

# Function that draws the guess. If the letter is in the secret word it will be displayed.Else just a \_ will be displayed

def draw\_guess():

display\_text = ''

for letter in WORD:

if letter in GUESS:

display\_text += f"{letter} "

else:

display\_text += "\_ "

text = letter\_font.render(display\_text, True, black)

my\_screen.blit(text, (400, 200))

# Menu 3. In this menu the user can choose the topic between car brands or car modifications for the secret word

def word\_choice\_menu():

#Reset the GUI manager to avoid overlapping

GUI\_MANAGER.clear\_and\_reset()

#Create a button for the topic using pygame\_gui

car\_brands\_button = pygame\_gui.elements.UIButton(

relative\_rect=pygame.Rect((275, 200), (110, 70)),

text="SUTUTUTU",

manager=GUI\_MANAGER

)

#Create a button for the topic using pygame\_gui

mods\_button = pygame\_gui.elements.UIButton(

relative\_rect=pygame.Rect((400, 200), (110, 70)),

text="MODS",

manager=GUI\_MANAGER

)

#Create a title for this screen

word\_choice\_title = game\_font.render("Choose a topic", True, black)

word\_choice\_title\_rect = word\_choice\_title.get\_rect(center=(WIDTH // 2, 100))

#Boolean to keep the loop running

running = True

while running:

#refresh rate to limit the FPS(frames per second) to 60

refresh\_rate = timer.tick(60) / 1000

my\_screen.fill(cyan)

#For loop to register the pygame events

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

pygame.quit()

sys.exit()

GUI\_MANAGER.process\_events(event)

#If statement to check if the button is pressed

if car\_brands\_button.check\_pressed():

global WORD

#If the button is pressed, choose a random word from the car brands lists and set it to the secret word

WORD = random.choice(cars)

#Run the game

game()

##If the button is pressed, choose a random word from the car parts lists and set it to the secret word

if mods\_button.check\_pressed():

WORD = random.choice(car\_mods)

#Run the game

game()

#Update the GUI manager at 60 fps

GUI\_MANAGER.update(refresh\_rate)

GUI\_MANAGER.draw\_ui(my\_screen)

my\_screen.blit(word\_choice\_title, word\_choice\_title\_rect)

pygame.display.update()

# Player score count

SCORE = 0

# Game function

# Adapted from https://www.youtube.com/watch?v=UEO1B\_llDnc

def game():

# lives to keep track of the game

lives = 0

game\_is\_over = False

# Main loop to keep pygame running

while True:

for event in pygame.event.get():

if event.type == pygame.QUIT:

exit()

# Event type to register the position where the user clicked

if event.type == pygame.MOUSEBUTTONDOWN:

clicked\_position = event.pos

# Checks if the letter

for button, letter in BUTTONS:

if button.collidepoint(clicked\_position):

if letter not in WORD:

#Play an audio if the letter is not in the secret word

doh.play()

#add 1 to lives

lives += 1

# if the lives reach 6, game\_is\_over will become true and end the game

if lives == 6:

game\_is\_over = True

GUESS.append(letter)

BUTTONS.remove([button, letter])

my\_screen.fill(cyan)

#Draw the hangman images into the screen. The images are stored in a list. Depending on the lives, the idex changes and moves into the next image.

my\_screen.blit(IMAGES[lives], (150, 100))

#Draw the title into the screen

my\_screen.blit(title\_text, title\_rect)

for box in BOXES:

pygame.draw.rect(my\_screen, black, box, 2)

#Draws the buttons for the letters

draw\_buttons()

#Draw the spaces in for the letters in the screen

draw\_guess()

#Boolean to check when the player wins

won = True

#For loop to check if the letter is in the secret word

for letter in WORD:

if letter not in GUESS:

won = False

#If the player won end the game, add 1 to his score and make the message You won

if won:

game\_is\_over = True

global SCORE

SCORE += 1

omg\_sound.play()

game\_message = "You won, play again?"

#If the player loss end the game and make the message You lost

elif lives >= 6:

game\_is\_over = True

game\_message = "You lost, play again?"

pygame.display.update()

timer.tick(60)

if game\_is\_over:

break

#Create yes and no button to ask the player if he wants to continue playing

yes\_button = Button(280, 320, 0.8, pygame.image.load(str(yes\_button\_path)))

no\_button = Button(400, 320, 0.8, pygame.image.load(str(no\_button\_path)))

#While loop for the end of the game

while game\_is\_over:

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

#Register the clicked positons

if event.type == pygame.MOUSEBUTTONDOWN:

clicked\_position = event.pos

if yes\_button.rect.collidepoint(clicked\_position):

# Reset the game variables

GUESS.clear()

BUTTONS.clear()

for ind, box in enumerate(BOXES):

letter = chr(A + ind)

button = [box, letter]

BUTTONS.append(button)

#Reset the game

game\_is\_over = False

#Take the user back to the word\_choice\_menu

word\_choice\_menu()

# check if the no button is pressed

if no\_button.rect.collidepoint(clicked\_position):

spideerman\_sound.play()

#Update the players score

update\_score(SCORE)

#Go back to the main menu

main\_menu()

my\_screen.fill(cyan)

game\_over\_message = game\_font.render(game\_message, True, black)

text\_rect = game\_over\_message.get\_rect(center=(WIDTH // 2, 200))

my\_screen.blit(game\_over\_message, text\_rect)

yes\_button.draw()

no\_button.draw()

pygame.display.update()

timer.tick(60)

# Second menu. In this menu the user can click two buttons. One to start playing and one to exit

def main\_menu():

"""

Function for the second menu. In the main menu two buttons are drawn. A play and exit button

"""

#Boolean to keep the loop running until its changed to False

running = True

#Main loop

while running:

#Fill the surface with cyan color

my\_screen.fill(cyan)

#Create the buttons with the Button class previously defined.

# The class takes x,y coordinates, scales the image. and loads the image.

play\_button = Button(340, 100, 0.8, pygame.image.load(str(play\_button\_path)))

exit\_button = Button(340, 200, 0.8, pygame.image.load(str(exit\_button\_path)))

leader\_board\_button = Button(340,300,0.7, pygame.image.load(str(leader\_board\_path)))

#Draw the buttons into the screen

play\_button.draw()

exit\_button.draw()

leader\_board\_button.draw()

#Update the screen

pygame.display.update()

timer.tick(60)

#Keep the loop running. The program will end if the close button is pressed

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

pygame.quit()

sys.exit()

#Check if the button is clicked

if event.type == pygame.MOUSEBUTTONDOWN:

#Gets the coordinates of the click

clicked\_position = event.pos

#Check if the coordinates of the click collides with the rect of the play button

if play\_button.rect.collidepoint(clicked\_position):

running = False

# Reset the game variables

GUESS.clear()

BUTTONS.clear()

for ind, box in enumerate(BOXES):

letter = chr(A + ind)

button = [box, letter]

BUTTONS.append(button)

#If the play button rect collides with the clicked position move on into the next menu.

word\_choice\_menu()

# If the leader board buttons is clicked, call the function for the leaderboard screen.

elif leader\_board\_button.rect.collidepoint(clicked\_position):

leaderboard\_screen()

# If the exit button rect collides with the clicked position, the programe ends.

elif exit\_button.rect.collidepoint(clicked\_position):

exit\_sound.play()

login()

def login():

"""

Function for login screen. This function takes a username and a password. The user

can sign in if he has a user or create a new user.

"""

fail\_sign = False

#Reset the GUI manager to avoid items to overlap

GUI\_MANAGER.clear\_and\_reset()

#Entry line to store the username

username = pygame\_gui.elements.UITextEntryLine(

relative\_rect=pygame.Rect((310, 100), (200, 50)),

manager=GUI\_MANAGER,

object\_id="#username\_txt\_entry",

placeholder\_text="Enter username"

)

#Entry line to store the password

password = pygame\_gui.elements.UITextEntryLine(

relative\_rect=pygame.Rect((310, 250), (200, 50)),

manager=GUI\_MANAGER,

object\_id="#username\_txt",

placeholder\_text="Enter password"

)

#Sign in button

sign\_in\_button = pygame\_gui.elements.UIButton(

#Give the button a rectangle, coordinate, and size

relative\_rect=pygame.Rect((WIDTH / 2 - 100, HEIGHT / 2 + 100), (100, 40)),

text="Sign In",

manager=GUI\_MANAGER

)

#Sign up button

sign\_up\_button = pygame\_gui.elements.UIButton(

#Give the button a rectangle, coordinate, and size

relative\_rect=pygame.Rect((WIDTH / 2 + 10, HEIGHT / 2 + 100), (100, 40)),

text="Sign Up",

manager=GUI\_MANAGER

)

#Boolen to keep the loop running until it changes to False

running = True

while running:

refresh\_rate = timer.tick(60) / 1000

#Set the background color for the surface

my\_screen.fill(cyan)

for event in pygame.event.get():

if event.type == pygame.QUIT:

running = False

pygame.quit()

sys.exit()

GUI\_MANAGER.process\_events(event)

#Gets the text that user typed in the username box

username\_txt = username.get\_text()

global globalUsername

globalUsername = username\_txt

#Gets the text that user typed in the password

password\_txt = password.get\_text()

# Check if the sign in button got pressed

if sign\_in\_button.check\_pressed():

#If the button is pressed called the sign\_in function previuosly defined

if sign\_in(username\_txt, password\_txt):

#Change the Boolean to False to stop running the login screen

running = False

#If the login is succesfull the program will move on into the main menu screen

main\_menu()

else:

fail\_sign = True

if fail\_sign == True:

fail\_sign\_sound.play()

fail\_msg = fail\_font.render("Invalid username or password", True, black)

fail\_msg\_rect = fail\_msg.get\_rect(center=(WIDTH // 2, 80))

my\_screen.blit(fail\_msg, fail\_msg\_rect)

#Check if the sign up button was pressed

if sign\_up\_button.check\_pressed():

#If it got pressed the sign up function will be called and store the username and password into the database

sign\_up(username\_txt, password\_txt)

#Updates the screen

GUI\_MANAGER.update(refresh\_rate)

#Draw all the pygame\_ui elements into a surface.

GUI\_MANAGER.draw\_ui(my\_screen)

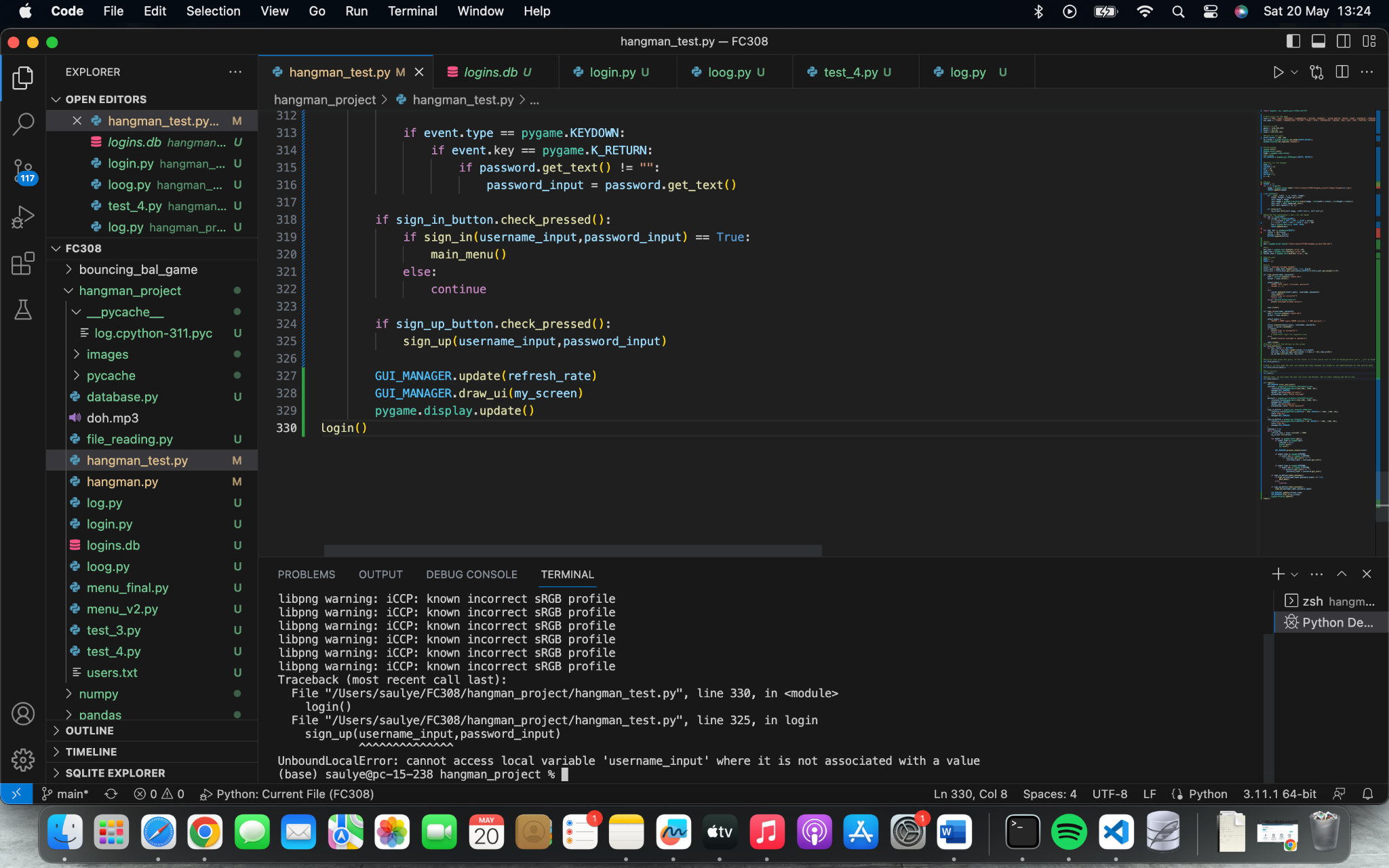
pygame.display.update()

login()

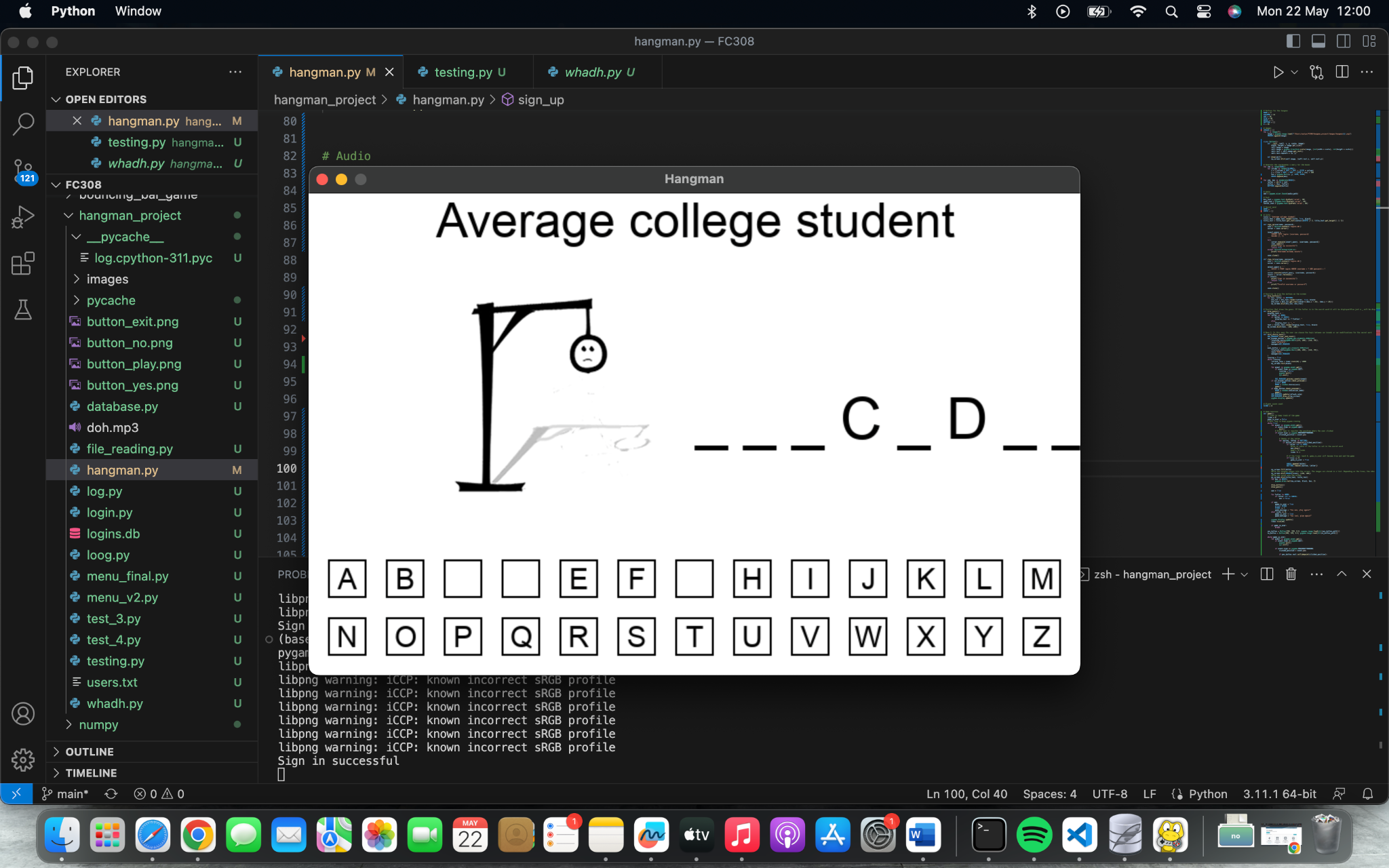
# Testing to inform development

Glitch#1

A test was done for the sign\_up() function in the image below. As it is shown, the username\_input could not be accessed since it was a local variable. Independently the sign\_up function was working, however when it got implemented into the login screen the function did not save the user and his password in the database. To fix this, a new variable called username\_txt was created and it was set equal to the input stored in the username UITextEntryLine with the .get\_text() function. The same was implemented to the password.



Glitch#2

This test was done to verify if the letters were correctly drawn into the screen. All the letters were drawn correctly as shown in the image below, but the letters were too big. If the word is too long, the guessed letters would go off the screen. Initially the letters were set to Arial 60, so to fix the bug the font was changed into a smaller size. The letter font was changed into Arial 40 to avoid the letters going off the screen.

Glitch#3

In this glitch the logins.db database could not be accessed and the username input could not be read since the database was not in the same directory as the hangman program. To fix this the database was moved to the same directory and the program had to be executed from the terminal using python3 command. Also before executing the program the directory needs to be changed in the terminal using the cd command.

# 

# 

# 

# 

# 

# 

# 

# 

# 

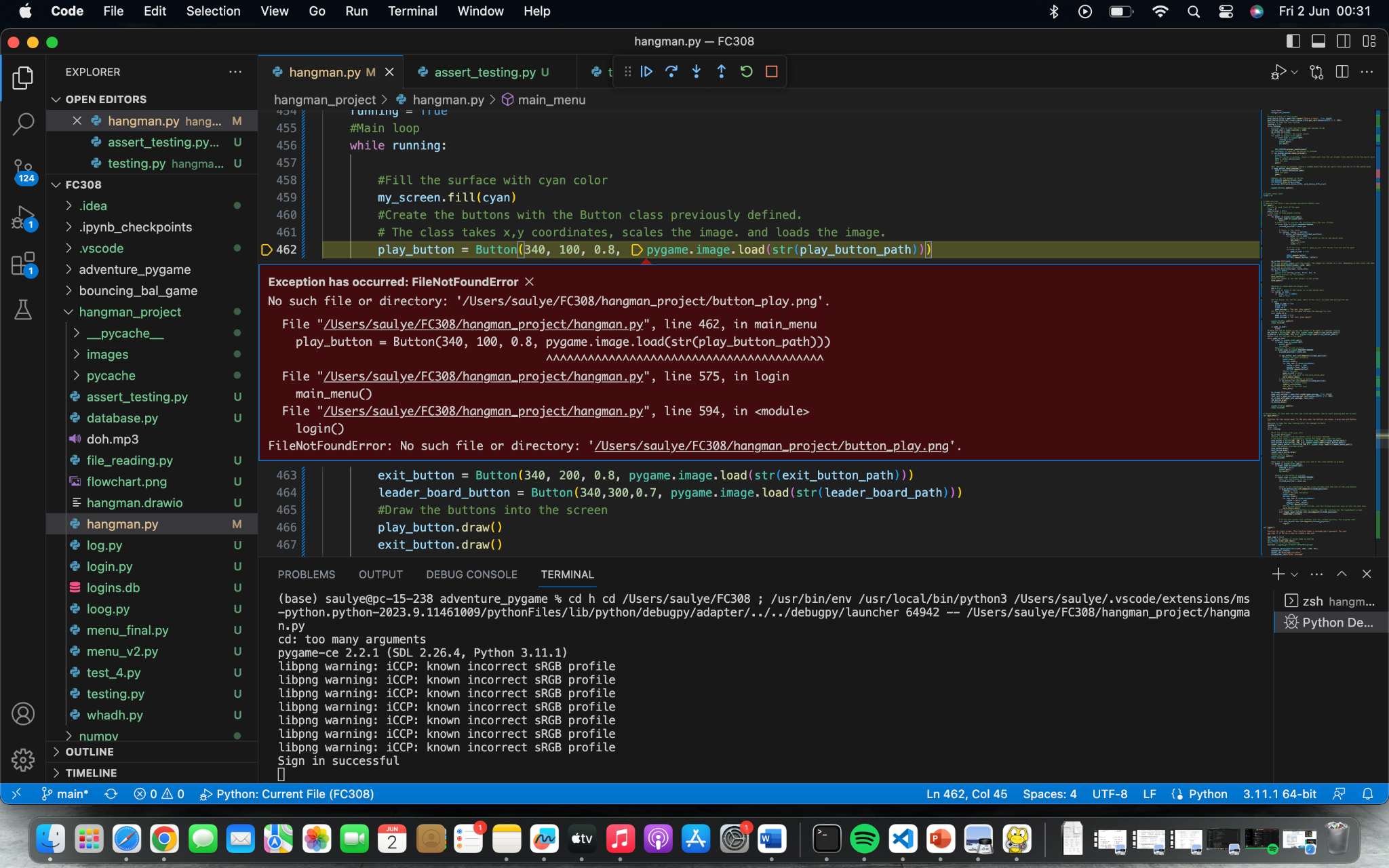
# 

# 

# 

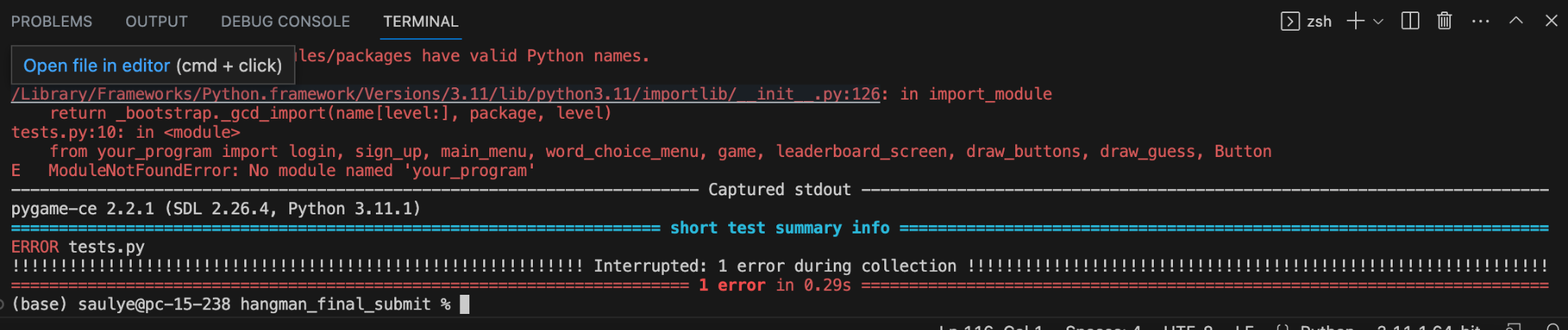
Glitch#4

Another glitch involving directory problems. To fix this the pathlib library was used to find the correct file path. Using pathlib is more effective since the literal path is not required to find the file. Pathlib allows the program to run in multiple computers, since typing in manually the path just works in the computer in which the program was written in



# Testing to inform evaluation

To ensure that the functions created worked properly a unit test was created for each function. With the use of pytest and assert testing, different scenarios were created.



from hangman import \*

import pygame

import pytest

# Import the functions from the code

def test\_sign\_up():

# Test case for successful scenario

assert sign\_up("test\_user", "test\_password") == True

# Test case when the user already exists

assert sign\_up("test\_user", "test\_password") == None

def test\_sign\_in():

# Case for succesful sign in

assert sign\_in("test\_user", "test\_password") == True

# Case for when the user enters an invalid password

assert sign\_in("test\_user", "wrong\_password") == None

def test\_update\_score():

# Test case for successful score update

assert update\_score(10) == True

# Test case for invalid username

assert update\_score(10) == None

# Run the tests

if \_\_name\_\_ == '\_\_main\_\_':

pygame.init()

pygame.display.set\_mode((1, 1))

pytest.main()

# Evaluation of Solution

In conclusion, the minimal requirements for both stakeholders were met. The game runs effectively and contains the basic requirements. The program contains a login menu, main menu, leaderboard chart, and the algorithm for the game. When it comes to the main stakeholder (the players) the program complies in satisfying them since extra features such as a colourful graphical user interface, and sounds were implemented. The GUI is interactive and allows the user to interact with the program.

On the other hand, the interests of the second stakeholder (boss/company) were met. The program runs smoothly and can be used in any computer, since pathlib was used to manage files.

Some of the extra features are the following:

* Sqlite3 databases. Implementing a database increases the efficiency of pushing and pulling data. A text file would satisfy the basic function of storing a username and its password, however using this method would be harder to search if the username already exists; keeping track of the score would also be harder. Using a database ensures that all the information is kept in the same file and primary keys are useful to match the username,password,and score.
* Pathlib directories. This module is useful to find the directories of files required for the program to run. Raw writing the directory of each file is not efficient and does not allow the program to run on different devices.
* Pygame Mixer. Sounds were used to make the game more interactive and attractive to the user.
* Pygame\_gui. Implementing a GUI is useful to get inputs from the user. The GUI from this library provides built-in objects like buttons and text boxes that allow the user to interact with the program. Overall, implementing pygam\_gui improves the UX(user experience) substantially.

Furthermore, the program could be benefited with more extra features. It is worth noting that these additional features are complementary, but could be implemented in the future. Features such as an encryption system to keep the username and password safe from external parties might be benefitial. Also, a varied selection of topics for the game could be implemented. The game programmed above is focused on cars and car culture. It targets a niche group of individuals that are interested in the topic. More topics could be an extra implementation.

Finally, the program meets the requirements and contains extra features. When it comes to reaching its goal, it is adequate for the scenario presented to Khalid.

# Reference list

Arne, G. (2023). *Python 3’s pathlib Module: Taming the File System – Real Python*. [online] realpython.com. Available at: https://realpython.com/python-pathlib/.

Lawrence, D. (n.d.). *Documentation - Home Page — Pygame GUI 0.6.9 documentation*. [online] pygame-gui.readthedocs.io. Available at: https://pygame-gui.readthedocs.io/en/v\_069/ [Accessed 2 Jun. 2023].

Pygame Org (n.d.). *Pygame Front Page — pygame v2.0.0.dev15 documentation*. [online] www.pygame.org. Available at: https://www.pygame.org/docs/.

Python Org (2022). *sys — System-specific parameters and functions — Python 3.7.3 documentation*. [online] Python.org. Available at: https://docs.python.org/3/library/sys.html.

Python, R. (n.d.). *Python’s assert: Debug and Test Your Code Like a Pro – Real Python*. [online] realpython.com. Available at: https://realpython.com/python-assert-statement/.

Shahid, M. (2023). *Personal Class*. 23 Mar.

Sqlite Org (2022). *INSERT*. [online] www.sqlite.org. Available at: https://www.sqlite.org/lang\_insert.html [Accessed 2 Jun. 2023].

Tech With Tim (2020). *Python Hangman Tutorial #1 - Learn to Make Games with Pygame*. *YouTube*. Available at: https://www.youtube.com/watch?v=UEO1B\_llDnc [Accessed 26 Nov. 2020].

‌