Estructura del proyecto dentro de VSCode:

Captura de pantalla de computadora

Descripción generada automáticamente

Carpeta del proyecto: Grupo-1---Taxi-dev-iu-tkinter

Subcarpeta: Gráficos

Archivos de código en Python (pygame y tkinter):

* GUI Entrada.py
* Check\_passwords.py
* IU\_Class.py
* Readme.md
* Usuarios.csv

Aquí están todos los códigos acordes cada tipo de archivo mencionado arriba:

* **GUI Entrada.py**
* import tkinter as tk
* from tkinter import messagebox
* from Check\_passwords import LogIn, Register, Pregunta, Respuesta, Descuentos, Descuentos\_taxi
* from IU\_Class import init\_game
* class GUI:
* def \_\_init\_\_(self):
* self.root = tk.Tk()
* self.root.title("Inicio de Sesión")
* self.root.geometry('800x600')
* self.root.configure(bg='#541388')
* self.style\_button = {'font': ('Lucida Console', 16), 'bg': '#C8F50A', 'fg': '#541388', 'padx': 20, 'pady': 10, 'bd': 0}
* self.p\_inicio()
* self.root.mainloop()
* def p\_inicio(self):
* self.clear\_screen()
* self.inicio = tk.Button(self.root, text="Iniciar Sesión", command=self.login\_screen, \*\*self.style\_button)
* self.inicio.pack(pady=75)
* self.reg = tk.Button(self.root, text="Registrarse", command=self.reg\_screen, \*\*self.style\_button)
* self.reg.pack(pady=75)
* def clear\_screen(self):
* for widget in self.root.winfo\_children():
* widget.destroy()
* def login\_screen(self):
* self.clear\_screen()
* self.label = tk.Label(self.root, text="Inicie Sesión", font=('Lucida Console', 20), bg='#541388', fg='white')
* self.label.pack(pady=20)
* # Título y Entry para el usuario
* self.label\_user = tk.Label(self.root, text="Usuario:", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label\_user.pack(pady=5)
* self.user = tk.Entry(self.root, font=('Lucida Console', 16))
* self.user.pack(pady=5)
* # Título y Entry para la contraseña
* self.label\_password = tk.Label(self.root, text="Contraseña:", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label\_password.pack(pady=5)
* self.password = tk.Entry(self.root, font=('Lucida Console', 16), show='\*')
* self.password.pack(pady=5)
* self.button = tk.Button(self.root, text="Iniciar Sesión", command=self.check\_password, \*\*self.style\_button)
* self.button.pack(pady=20)
* self.button\_forgot = tk.Button(self.root, text="Olvidé mi contraseña", command=self.res\_pswd, \*\*self.style\_button)
* self.button\_forgot.pack(pady=10)
* self.button\_back = tk.Button(self.root, text="Atrás", command=self.p\_inicio, \*\*self.style\_button)
* self.button\_back.pack(pady=10)
* def reg\_screen(self):
* self.clear\_screen()
* self.label = tk.Label(self.root, text='Registro', font=('Lucida Console', 20), bg='#541388', fg='white')
* self.label.pack(pady=20)
* self.label\_user = tk.Label(self.root, text="Usuario:", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label\_user.pack(pady = 5)
* self.user = tk.Entry(self.root, font=('Lucida Console', 16))
* self.user.pack(pady=10)
* self.label\_password = tk.Label(self.root, text="Contraseña", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label\_password.pack(pady = 5)
* self.password = tk.Entry(self.root, font=('Lucida Console', 16), show='\*')
* self.password.pack(pady=10)
* self.label\_quest = tk.Label(self.root, text="Defina una pregunta secreta para cambiar la contraseña en caso de olvido", font=('Lucida Console', 12), bg='#541388', fg='white')
* self.label\_quest.pack(pady = 5)
* self.quest = tk.Entry(self.root, font=('Lucida Console', 16))
* self.quest.pack(pady=10)
* self.label\_answ = tk.Label(self.root, text="Defina una respuesta para su pregunta secreta", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label\_answ.pack(pady = 5)
* self.answ = tk.Entry(self.root, font=('Lucida Console', 16))
* self.answ.pack(pady=10)
* self.label\_dropdown = tk.Label(self.root, text="Seleccione tipo de conductor:", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label\_dropdown.pack(pady=5)
* # Lista vacía para el menú desplegable
* options = ["Taxista", "VTC"]
* self.selected\_option = tk.StringVar()
* self.selected\_option.set("")
* self.dropdown = tk.OptionMenu(self.root, self.selected\_option, \*options)
* self.dropdown.config(font=('Lucida Console', 16), bg='#C8F50A', fg='#4100A8', width=20)
* self.dropdown.pack(pady=5)
* button\_frame = tk.Frame(self.root, bg='#791E94')
* self.button\_register = tk.Button(button\_frame, text="Registro", command=self.register, \*\*self.style\_button)
* self.button\_register.pack(side=tk.LEFT, padx=10, pady=10)
* self.button\_back = tk.Button(button\_frame, text="Incio de Sesion", command=self.login\_screen, \*\*self.style\_button)
* self.button\_back.pack(side=tk.LEFT, padx=10, pady=10)
* button\_frame.pack(pady=20)
* def res\_pswd(self):
* self.clear\_screen()
* self.label = tk.Label(self.root, text="Reiniciar contraseña", font=('Lucida Console', 20), bg='#541388', fg='white')
* self.label.pack(pady=20)
* self.label\_user = tk.Label(self.root, text="Nombre de usuario", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label\_user.pack(pady = 5)
* self.user = tk.Entry(self.root, font=('Lucida Console', 16))
* self.user.pack(pady=10)
* self.quest = tk.Button(self.root, text="Ver pregunta secreta", command=self.get\_quest, \*\*self.style\_button)
* self.quest.pack(pady=10)
* self.label2 = tk.Label(self.root, text="Escriba su respuesta secreta", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label2.pack(pady=10)
* self.answer = tk.Entry(self.root, font=('Lucida Console', 16))
* self.answer.pack(pady=10)
* self.label3 = tk.Label(self.root, text="Nueva contraseña", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label3.pack(pady=10)
* self.new\_pswd = tk.Entry(self.root, font=('Lucida Console', 16), show='\*')
* self.new\_pswd.pack(pady=10)
* self.submit = tk.Button(self.root, text="Enviar", command=self.change\_pswd, \*\*self.style\_button)
* self.submit.pack(pady=20)
* self.back = tk.Button(self.root, text="Atrás", command=self.login\_screen, \*\*self.style\_button)
* self.back.pack(pady=10)
* def check\_password(self):
* log = LogIn(self.user.get().lower(), self.password.get())
* user = self.user.get()
* if log == False:
* messagebox.showinfo(title = "Error", message = "Nombre de usuario o contraseña equivocados")
* else:
* if log == 'VTC':
* self.discount\_screen(user)
* elif log == 'Taxista':
* self.turno\_screen(user)
* def register(self):
* if Register(self.user.get().lower(), self.password.get(), self.quest.get(), self.answ.get().lower(), self.selected\_option.get()) == True:
* self.login\_screen()
* def get\_quest(self):
* Pregunta(self.user.get().lower())
* def change\_pswd(self):
* Respuesta(self.user.get().lower(), self.answer.get().lower(), self.new\_pswd.get())
* def turno\_screen(self, user):
* self.clear\_screen()
* self.user = user
* self.label = tk.Label(self.root, text="Tarifas", font=('Lucida Console', 20), bg='#541388', fg='white')
* self.label.pack(pady=20)
* self.label1 = tk.Label(self.root, text="Turno", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label1.pack(pady=10)
* options = ["Diurno", "Nocturno"]
* self.turno = tk.StringVar()
* self.turno.set("")
* self.dropdown = tk.OptionMenu(self.root, self.turno, \*options)
* self.dropdown.config(font=('Lucida Console', 16), bg='#C8F50A', fg='#4100A8', width=20)
* self.dropdown.pack(pady=5)

* self.label2 = tk.Label(self.root, text="Porcentaje de aumento de tarifa (solo para noche)", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label2.pack(pady=10)
* self.tarifa\_extra = tk.Entry(self.root, font=('Lucida Console', 16))
* self.tarifa\_extra.pack(pady=10)
* self.submit = tk.Button(self.root, text="Guardar", command=lambda: self.save\_discounts\_t(self.user), \*\*self.style\_button)
* self.submit.pack(pady=20)
* def discount\_screen(self, user):
* self.clear\_screen()
* self.user = user
* self.label = tk.Label(self.root, text="Descuentos", font=('Lucida Console', 20), bg='#541388', fg='white')
* self.label.pack(pady=20)
* self.label1 = tk.Label(self.root, text="Porcentaje de descuento parado", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label1.pack(pady=10)
* self.discount\_stopped = tk.Entry(self.root, font=('Lucida Console', 16))
* self.discount\_stopped.pack(pady=10)
* self.label2 = tk.Label(self.root, text="Porcentaje de descuento movimiento", font=('Lucida Console', 16), bg='#541388', fg='white')
* self.label2.pack(pady=10)
* self.discount\_moving = tk.Entry(self.root, font=('Lucida Console', 16))
* self.discount\_moving.pack(pady=10)
* self.submit = tk.Button(self.root, text="Guardar", command=lambda: self.save\_discounts(self.user), \*\*self.style\_button)
* self.submit.pack(pady=20)
* def save\_discounts(self, user):
* self.user = user.lower()
* stopped\_discount = self.discount\_stopped.get()
* moving\_discount = self.discount\_moving.get()
* stopped\_discount = stopped\_discount if stopped\_discount else 0
* moving\_discount = moving\_discount if moving\_discount else 0
* Descuentos(self.user.lower(), stopped\_discount, moving\_discount)
* messagebox.showinfo(title = "Exito", message = "Descuentos aplicados")
* init\_game(self.user)
* # Aquí puedes agregar la lógica para manejar los valores de descuento ingresados
* def save\_discounts\_t(self, user):
* self.user = user.lower()
* turno = self.turno.get()
* tarifa\_extra = self.tarifa\_extra.get()
* tarifa\_extra = tarifa\_extra if tarifa\_extra else '0'
* Descuentos\_taxi(self.user.lower(), turno, tarifa\_extra)
* messagebox.showinfo(title = "Exito", message = "Descuentos aplicados")
* init\_game(self.user)
* GUI()
* **Check\_passwords.py**
* import pandas as pd
* import hashlib
* import tkinter as tk
* from IU\_Class import init\_game
* #Cuando tengamos una base de datos como tal
* #datos\_usuarios = pd.read\_csv("usuarios.csv")
* def LogIn(username, password):
* datos\_usuarios = pd.read\_csv("Usuarios.csv")
* password\_inp = hashlib.sha256(password.encode('utf-8')).hexdigest()
* password\_local = datos\_usuarios.loc[datos\_usuarios["Usuarios"] == username]["Passwords"].item()
* if username not in datos\_usuarios["Usuarios"].values or password\_inp != password\_local:
* return False
* else:
* return datos\_usuarios.loc[datos\_usuarios["Usuarios"] == username]["Licencia"].item()
* def Register(username, password, s\_quest, s\_answer, conductor):
* datos\_usuarios = pd.read\_csv("Usuarios.csv")
* if datos\_usuarios.Usuarios.isin([username]).any():
* tk.messagebox.showinfo(title = "Error", message = "Nombre de usuario en uso, por favor eliga otro")
* else:
* password\_hash = hashlib.sha256(password.encode('utf-8')).hexdigest()
* df = pd.DataFrame({'Usuarios' : [username], 'Passwords': [password\_hash], 'Pregunta Secreta' : [s\_quest], 'Respuesta Secreta': [s\_answer], 'Licencia': [conductor]})
* datos\_usuarios = pd.concat([datos\_usuarios, df], ignore\_index = True)
* datos\_usuarios.to\_csv('Usuarios.csv', index = False)
* tk.messagebox.showinfo(title = "Registro completado", message = "Registo completado")
* return(True)
* def Pregunta(username):
* datos\_usuarios = pd.read\_csv("Usuarios.csv")
* if not datos\_usuarios.Usuarios.isin([username]).any():
* tk.messagebox.showinfo(title = "Error", message = "Usuario no encontrado")
* else:
* pregunta\_s = datos\_usuarios.loc[datos\_usuarios["Usuarios"] == username]["Pregunta Secreta"].item()
* tk.messagebox.showinfo(title = "Pregunta", message = pregunta\_s)
* def Respuesta(username, answer, new\_pswd):
* datos\_usuarios = pd.read\_csv("Usuarios.csv")
* local\_answ = datos\_usuarios.loc[datos\_usuarios["Usuarios"] == username]["Respuesta Secreta"].item()
* if local\_answ == answer:
* new\_pswd\_hash = hashlib.sha256(new\_pswd.encode('utf-8')).hexdigest()
* datos\_usuarios.loc[datos\_usuarios["Usuarios"] == username, "Passwords"] = new\_pswd\_hash
* datos\_usuarios.to\_csv('Usuarios.csv', index = False)
* tk.messagebox.showinfo(title = "Exito", message = "Contraseña cambiada")
* else:
* tk.messagebox.showinfo(title = "Error", message = "Respuesta Incorrecta")
* def Descuentos(username, stop\_disc, mov\_disc):
* datos\_usuarios = pd.read\_csv("Usuarios.csv")
* datos\_usuarios.loc[datos\_usuarios["Usuarios"] == username, "Descuento Parado"] = int(stop\_disc)
* datos\_usuarios.loc[datos\_usuarios["Usuarios"] == username, "Descuento Movimiento"] = int(mov\_disc)
* datos\_usuarios.to\_csv('Usuarios.csv', index = False)
* def Descuentos\_taxi(username, turno, tarifa):
* datos\_usuarios = pd.read\_csv("Usuarios.csv")
* datos\_usuarios.loc[datos\_usuarios["Usuarios"] == username, "Turno"] = turno
* datos\_usuarios.loc[datos\_usuarios["Usuarios"] == username, "Tarifa extra"] = int(tarifa)
* datos\_usuarios.to\_csv('Usuarios.csv', index = False)
* **IU\_Class.py**
* import pygame
* from sys import exit
* import pygame\_gui
* import time
* from datetime import datetime
* import pandas as pd
* class Game:
* def \_\_init\_\_(self, user):
* self.FPS = 60
* self.S\_Width = 1600
* self.S\_Height = 900
* pygame.init()
* self.screen = pygame.display.set\_mode((self.S\_Width, self.S\_Height))
* self.clock = pygame.time.Clock()
* self.manager = pygame\_gui.UIManager((self.S\_Width, self.S\_Height))
* self.user = user
* self.gameStateManager = gameStateManager('start')
* self.start = Start(self.screen, self.gameStateManager)
* self.intro = Intro(self.screen, self.gameStateManager)
* self.taximetro = Taximetro(self.screen, self.gameStateManager, self.user)
* self.pantalla\_fin = pantalla\_fin(self.screen, self.gameStateManager, self.user)
* self.quit = Quit(self.screen, self.gameStateManager)
* self.states = {'start': self.start,
* 'taximetro': self.taximetro,
* 'intro': self.intro,
* 'pantalla\_fin': self.pantalla\_fin,
* 'quit': self.quit}
* self.gameStateManager.set\_states(self.states)
* def run(self):
* while True:
* for event in pygame.event.get():
* if event.type == pygame.QUIT:
* self.quit.handle\_quit()
* # Manejar eventos específicos del estado actual
* self.states[self.gameStateManager.get\_state()].handle\_events(event)
* self.states[self.gameStateManager.get\_state()].run()
* pygame.display.update()
* self.clock.tick(self.FPS)
* class Start:
* def \_\_init\_\_(self, display, gameStateManager):
* self.display = display
* self.gameStateManager = gameStateManager
* def handle\_events(self, event):
* if event.type == pygame.KEYDOWN:
* if event.key == pygame.K\_e:
* self.gameStateManager.set\_state('level')
* elif event.type == pygame.MOUSEBUTTONDOWN:
* a, b = pygame.mouse.get\_pos()
* if self.quit\_button\_rect.collidepoint((a, b)):
* self.gameStateManager.set\_state('quit')
* elif self.login\_button\_rect.collidepoint((a, b)):
* self.gameStateManager.set\_state('intro')
* def run(self):
* # Variables generales
* a, b = pygame.mouse.get\_pos()
* login\_screen = pygame.image.load('Graficos/start\_p.jpeg')
* font = pygame.font.SysFont('Lucida Console', 70)
* color\_font = (200, 245, 10, 1)
* color\_rect\_hover = (91, 23, 202, 0.8)
* color\_rect\_base = (65, 0, 168, 0.9)
* # Botón Start
* self.login\_button\_rect = pygame.Rect(500, 400, 650, 80)
* login\_text = font.render('Empezar carrera', True, color\_font)
* # Botón Quit
* self.quit\_button\_rect = pygame.Rect(730, 650, 180, 80)
* quit\_text = font.render('Quit', True, color\_font)
* self.display.blit(login\_screen, (0, 0))
* if self.quit\_button\_rect.collidepoint((a, b)):
* pygame.draw.rect(self.display, color\_rect\_hover, self.quit\_button\_rect)
* else:
* pygame.draw.rect(self.display, color\_rect\_base, self.quit\_button\_rect)
* if self.login\_button\_rect.collidepoint((a, b)):
* pygame.draw.rect(self.display, color\_rect\_hover, self.login\_button\_rect)
* else:
* pygame.draw.rect(self.display, color\_rect\_base, self.login\_button\_rect)
* self.display.blit(login\_text, (self.login\_button\_rect.x + 5, self.login\_button\_rect.y + 5))
* self.display.blit(quit\_text, (self.quit\_button\_rect.x + 5, self.quit\_button\_rect.y + 5))
* class Intro:
* def \_\_init\_\_(self, display, gameStateManager):
* self.display = display
* self.gameStateManager = gameStateManager
* def handle\_events(self, event):
* if event.type == pygame.KEYDOWN:
* if event.key == pygame.K\_RETURN:
* self.gameStateManager.set\_state('taximetro')
* # Pasar el estado actual a Taximetro para iniciar el tiempo
* self.gameStateManager.get\_states()['taximetro'].start\_time = time.time()
* def run(self):
* fondo = pygame.image.load('Graficos/tuto.jpeg')
* self.display.blit(fondo, (0, 0))
* texto = pygame.image.load('Graficos/Texto intro.png')
* self.display.blit(texto, (175, 100))
* class Taximetro:
* def \_\_init\_\_(self, display, gameStateManager, user):
* self.user = user
* self.display = display
* self.gameStateManager = gameStateManager
* self.car = pygame.image.load('Graficos/car.png')
* self.car\_position = 20
* self.car\_mov = False
* self.font = pygame.font.SysFont('Lucida Console', 30)
* self.start\_time = None  # Inicializamos start\_time como None
* self.score = 0
* self.datos\_usuarios = pd.read\_csv("Usuarios.csv")
* self.update\_tarifas()
* def update\_tarifas(self):
* user\_info = self.datos\_usuarios[self.datos\_usuarios["Usuarios"] == self.user].iloc[0]
* licencia = user\_info["Licencia"]
* if licencia == 'Taxista':
* turno = user\_info["Turno"]
* if turno == 'Nocturno':
* self.porc = user\_info["Tarifa extra"]
* self.tarifa\_mov = 0.05+(0.05\*(int(self.porc)/100))
* self.tarifa\_par = 0.02+(0.02\*(int(self.porc)/100))
* else:
* self.tarifa\_mov = 0.05
* self.tarifa\_par = 0.02
* else:
* disc\_mov = user\_info["Descuento Movimiento"]
* disc\_stp = user\_info["Descuento Parado"]
* self.tarifa\_mov = 0.05-(0.05\*(int(disc\_mov)/100))
* self.tarifa\_par = 0.02-(0.02\*(int(disc\_stp)/100))
* def create\_csv\_if\_not\_exists(self, filename):
* try:
* pd.read\_csv(filename)  # Intentar cargar el archivo
* except FileNotFoundError:
* # El archivo no existe, crearlo con un DataFrame vacío y guardar
* df = pd.DataFrame(columns=['Usuario', 'Fecha', 'Tiempo\_Minutos', 'Tiempo\_Segundos', 'Precio'])
* df.to\_csv(filename, index=False)
* def handle\_events(self, event):
* if event.type == pygame.KEYDOWN:
* if event.key == pygame.K\_SPACE:
* self.car\_mov = not self.car\_mov
* elif event.key == pygame.K\_p:
* self.gameStateManager.set\_state('start')
* elif event.key == pygame.K\_RETURN:
* self.gameStateManager.set\_state('pantalla\_fin')
* def run(self):
* self.create\_csv\_if\_not\_exists('Carreras.csv')
* first\_screen = pygame.image.load('Graficos/base\_2.jpeg')
* self.display.blit(first\_screen, (0, 0))
* color\_font = (200, 245, 10, 1)
* if self.start\_time is not None:  # Aseguramos que start\_time tenga un valor antes de usarlo
* if self.car\_mov:
* self.car\_position += 5  # Ajusta la velocidad del coche según sea necesario
* if self.car\_position > 1600:  # 1600 es el ancho de la pantalla
* self.car\_position = -self.car.get\_width()  # Aparecer en el otro lado
* self.score += self.tarifa\_mov / 60  # Incrementar la puntuación por segundo en movimiento
* else:
* self.score += self.tarifa\_par / 60  # Incrementar la puntuación por segundo en parado
* self.display.blit(self.car, (self.car\_position, 600))
* # Calcular el tiempo transcurrido en minutos y segundos
* elapsed\_time\_s = time.time() - self.start\_time
* elapsed\_minutes = int(elapsed\_time\_s // 60)
* elapsed\_seconds = int(elapsed\_time\_s % 60)
* clock\_text = self.font.render(f'Tiempo: {elapsed\_minutes:02}:{elapsed\_seconds:02}', True, (color\_font))
* self.display.blit(clock\_text, (50, 50))
* # Mostrar la puntuación
* score\_text = self.font.render(f'Precio: {round(self.score, 2)} €', True, (color\_font))
* self.display.blit(score\_text, (50, 100))
* tarifa\_mov\_text = self.font.render(f'Tarifa en movimiento: {round(self.tarifa\_mov, 2)}', True, (color\_font))
* self.display.blit(tarifa\_mov\_text, (50, 150))
* tarifa\_stp\_text = self.font.render(f'Tarifa en parado: {round(self.tarifa\_par, 2)}', True, (color\_font))
* self.display.blit(tarifa\_stp\_text, (50, 200))
* def reset(self):
* self.start\_time = time.time()
* self.score = 0
* self.car\_position = 20
* self.car\_mov = False
* def get\_score(self):
* return self.score
* def get\_total\_time(self):
* if self.start\_time is None:
* return 0
* return time.time() - self.start\_time
* class gameStateManager:
* def \_\_init\_\_(self, currentState):
* self.currentState = currentState
* self.states = None  # Inicializamos states como None
* def set\_states(self, states):
* self.states = states  # Método para establecer los estados
* def get\_states(self):
* return self.states  # Método para obtener los estados
* def get\_state(self):
* return self.currentState
* def set\_state(self, state):
* self.currentState = state
* class pantalla\_fin:
* def \_\_init\_\_(self, display, gameStateManager, user):
* self.display = display
* self.gameStateManager = gameStateManager
* self.font = pygame.font.SysFont('Lucida Console', 70)
* self.color\_font = (200, 245, 10, 1)
* self.color\_background = (65, 0, 168, 0.9)
* self.final\_price = 0
* self.total\_time = 0
* self.user = user
* self.csv\_updated = False  # Flag para controlar la escritura en el CSV
* self.time\_stopped = False
* def handle\_events(self, event):
* if event.type == pygame.MOUSEBUTTONDOWN:
* a, b = pygame.mouse.get\_pos()
* if self.quit\_button\_rect.collidepoint((a, b)):
* self.gameStateManager.set\_state('quit')
* elif self.login\_button\_rect.collidepoint((a, b)):
* self.gameStateManager.get\_states()['taximetro'].reset()
* self.gameStateManager.set\_state('intro')
* self.reset()
* def precio\_final(self):
* a, b = pygame.mouse.get\_pos()
* login\_screen = pygame.image.load('Graficos/fin.jpeg')
* font = pygame.font.SysFont('Lucida Console', 70)
* color\_font = (200, 245, 10, 1)
* color\_rect\_hover = (91, 23, 202, 0.8)
* color\_rect\_base = (65, 0, 168, 0.9)
* # Botón Start
* self.login\_button\_rect = pygame.Rect(400, 400, 850, 80)
* login\_text = font.render('Empezar otra carrera', True, color\_font)
* # Botón Quit
* self.quit\_button\_rect = pygame.Rect(725, 650, 180, 80)
* quit\_text = font.render('Quit', True, color\_font)
* self.display.blit(login\_screen, (0, 0))
* if self.quit\_button\_rect.collidepoint((a, b)):
* pygame.draw.rect(self.display, color\_rect\_hover, self.quit\_button\_rect)
* else:
* pygame.draw.rect(self.display, color\_rect\_base, self.quit\_button\_rect)
* if self.login\_button\_rect.collidepoint((a, b)):
* pygame.draw.rect(self.display, color\_rect\_hover, self.login\_button\_rect)
* else:
* pygame.draw.rect(self.display, color\_rect\_base, self.login\_button\_rect)
* self.display.blit(login\_text, (self.login\_button\_rect.x + 5, self.login\_button\_rect.y + 5))
* self.display.blit(quit\_text, (self.quit\_button\_rect.x + 5, self.quit\_button\_rect.y + 5))
* price\_text = self.font.render(f'Precio final: {round(self.final\_price, 2)}€', True, self.color\_font)
* minutos = int(self.total\_time // 60)
* segundos = int(self.total\_time % 60)
* time\_text = self.font.render(f'Tiempo total de carrera: {minutos}m:{segundos}s', True, self.color\_font)
* price\_text\_rect = price\_text.get\_rect(center = (800, 250))
* time\_text\_rect = time\_text.get\_rect(center = (800, 350))
* self.display.blit(price\_text, price\_text\_rect)
* self.display.blit(time\_text, time\_text\_rect)
* def run(self):
* if not self.time\_stopped:  # Solo para cuando no está detenido aún
* self.final\_price = self.gameStateManager.get\_states()['taximetro'].get\_score()
* self.total\_time = self.gameStateManager.get\_states()['taximetro'].get\_total\_time()
* self.time\_stopped = True  # Detiene el tiempo al establecerlo la primera vez
* self.precio\_final()
* self.today = datetime.now()
* self.d1 = self.today.strftime("%d/%m/%Y %H:%M:%S")
* if not self.csv\_updated:  # Solo actualiza el CSV si no ha sido actualizado aún
* datos\_usuarios = pd.read\_csv('Carreras.csv')
* df = pd.DataFrame({'Usuario': [self.user], 'Fecha': [self.d1], 'Tiempo\_Minutos': [int(self.total\_time // 60)], 'Tiempo\_Segundos': [int(self.total\_time % 60)], 'Precio': [round(self.final\_price, 2)]})
* datos\_usuarios = pd.concat([datos\_usuarios, df], ignore\_index = True)
* datos\_usuarios.to\_csv('Carreras.csv', index = False)
* self.csv\_updated = True  # Marca el CSV como actualizado
* def reset(self):
* self.final\_price = 0
* self.total\_time = 0
* self.time\_stopped = False
* self.csv\_updated = False
* class Quit:
* def \_\_init\_\_(self, display, gameStateManager):
* self.display = display
* self.gameStateManager = gameStateManager
* def handle\_events(self, event):
* # En esta clase solo manejamos el evento de pygame.QUIT
* if event.type == pygame.QUIT:
* self.gameStateManager.set\_state('quit')
* def handle\_quit(self):
* self.gameStateManager.set\_state('quit')
* def run(self):
* pygame.quit()
* exit()
* def init\_game(user):
* game = Game(user)
* game.run()
* **Readme.md**
* **### Como arrancar el programa**
* Hay que bajarse del github todos los documentos que hay subidos y mantener la misma estructura que tienen.
* Para poder ejecutar el código se necesita instalar las bibliotecas:
* - tkinter
* - pygame
* - pygame\_gui
* Ejecutar el script 'GUI Entrada.py' para comenzar a utilizar la aplicación y seguir los pasos necesarios. Registrarse si es la primera vez y seleccionar tipo de conductor.
* Despues cuando el inicio de sesión es correcto y no presenta errores se podrán definir los descuentos que se quieran aplicar en el caso de licencias 'VTC' y seleccionar el turno en el caso de Taxistas. La tarifa extra solo se aplica durante el horario nocturno pero de momento hay que escribirla hasta que se meta un valor de control ahí para que no salte error.
* **Usuarios.csv**
* Usuarios,Passwords,Pregunta Secreta,Respuesta Secreta,Licencia,Descuento Parado,Descuento Movimiento,Turno,**Tarifa extra**
* Test\_Case, Test\_Case, Test\_Case, Test\_Case, Test\_Case,,,,
* alberto,03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f4,Comida favorita,pizza,Taxista,,,Nocturno,**100.0**
* erika,8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92,erika,erika,Taxista,,,,**0.0**