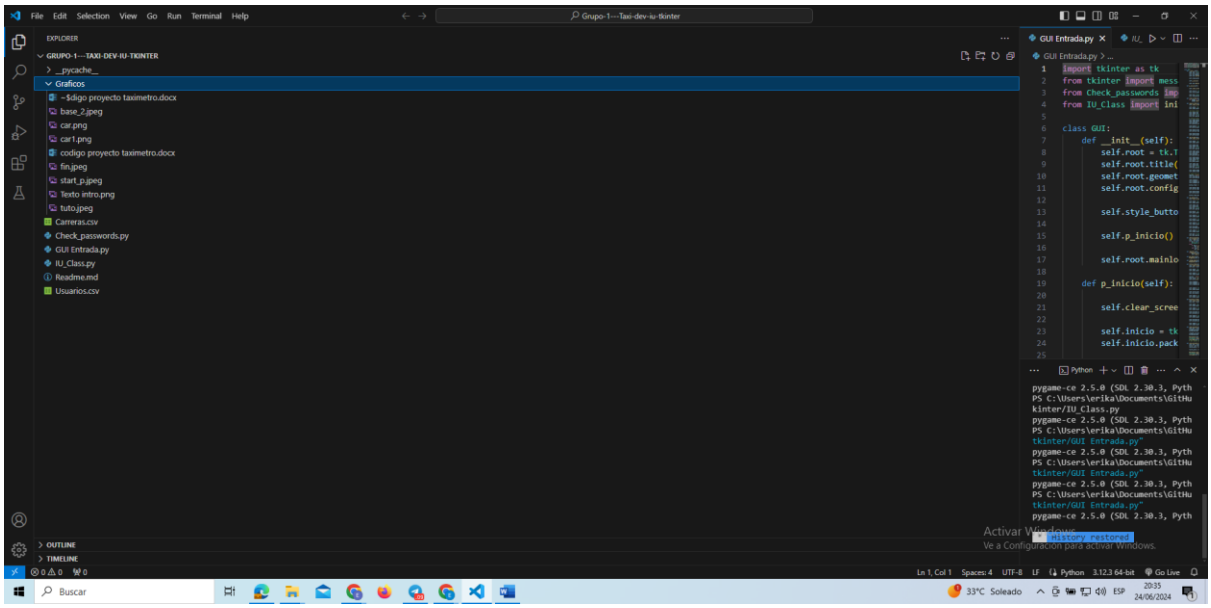
Estructura del proyecto dentro de VSCode:



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_e
    vents(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_t
    ime = 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,03ac674216f3e15c761ee1a5e255f067953623c8b388b4459e13f978d7c846f
    4,Comida favorita,pizza,Taxista,,,Nocturno,100.0
```

```
-   erika,8d969eef6ecad3c29a3a629280e686cf0c3f5d5a86aff3ca12020c923adc6c92,
    erika,erika,Taxista,,,,0.0
```