Carpeta Proyecto: GRUPO-1—TAXI

Subcarpeta: Graficos

Todos los archivos de código abajo están dentro de la carpeta de proyecto GRUPO-1—TAXI

Archivo: 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:

        tk.messagebox.showinfo(title = "Error", message = "Usuario o contraseña incorrecta")

    else:

        init\_game(username)

def Register(username, password, s\_quest, s\_answer):

    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})

        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")

Archivo: Clases taxi.py

'''

Definimos una clase principal conductor donde incluimos como atributos las tarifas estándar para todos los vehiculos tipo taxi.

Después con herencias, establecemos dos clases, VTC (uber, cabify, etc.) que heredan las tarifas de conductor pero que permiten establecer atributos distintos.

En el caso de los VTC, como quieren competir con los taxis, establecen el descuento que consideren necesario a sus tarifas, tanto en parado como en movimiento, pasando como argumentos los porcentajes de descuento para cada uno de ellos. Por defecto no hacen descuento.

En el caso de los taxi, su tarifa es fija, sin embargo por la noche cobran plus de nocturnidad, por lo tanto, cada taxista puede ajustar cuanto quiere cobrar extra por la noche. Para ello el primer argumento define si está en horario nocturno mediante True o False, y de ser noche, establece el procenatje (que por defecto es 0) que se debe sumar a la tarifa total.

'''

class conductor:

    tarifa\_parado = 0.02

    tarifa\_movimiento = 0.05

class VTC(conductor):

    def \_\_init\_\_(self, descuento\_p = 0, descuento\_m = 0):

        self.tarifa\_movimiento += self.tarifa\_movimiento \* (descuento\_m/100)

        self.tarifa\_parado += self.tarifa\_parado \* (descuento\_p/100)

        self.tarifa\_movimiento = round(self.tarifa\_movimiento, 2)

        self.tarifa\_parado = round(self.tarifa\_parado, 2)

class taxista(conductor):

    def \_\_init\_\_(self, noche, prcnt = 0):

        if noche:

            self.tarifa\_parado += self.tarifa\_parado \* (prcnt/100)

            self.tarifa\_movimiento += self.tarifa\_movimiento \* (prcnt/100)

            self.tarifa\_parado = round(self.tarifa\_parado, 2)

            self.tarifa\_movimiento = round(self.tarifa\_movimiento, 2)

taxi = taxista(True, 20)

taxi.tarifa\_parado

uber = VTC(20, 30)

uber.tarifa\_parado

Archivo: Creador de logs.py

import logging # Biblioteca para generar los logs

def creador\_logs():

    '''

    Se crea una función que va a iniciar el proceso de logear los sucesos.

    getLogger es una función de la biblioteca logging, a la que le asignamos \_\_name\_\_ que asigna a la variable el nombre del modulo a emplear

    basicConfig permite configurar los logs que se van a producir

    Por último devemos devolver el logger que hemos creado

    '''

    logger = logging.getLogger(\_\_name\_\_)

    logging.basicConfig(filename = 'archivo\_logs.log', #nombre del archivo

                        level = logging.INFO,  #nivel mínimo para el que va a generarse logs (mirar tabla https://docs.python.org/3/library/logging.html#levels)

                        format='%(asctime)s - %(levelname)s - %(message)s', #formato de presentación del log, asctime será la hora, levelname el nombre del nivel del mensaje y message el mensaje que nosotros asignemos

                        datefmt='%d/%m/%Y %I:%M:%S: %p') #formato de fecha, puesto para día, mes, año, hora, minuto, segundo y AM o PM.

    return logger

def taximetro():

    logger = creador\_logs() #llamamos a la función previa

    '''

    El código a continuación es meramente una prueba para ver como funcinaría el logger

    Los diferentes niveles se establecen con un .level(.info/.error) que saldrán en el archivo del log

    '''

    logger.info("Inicio del log")

    movimiento = 0.05

    parado = 0.02

    try: # Para guardar los errores utilizar un try/except

        movimiento/0

        logger.info("Funcionando")

        logger.info("Final")

    except ZeroDivisionError:

        logger.error("Se ha producido un error")

taximetro()

Archivo: Estructura base.py

import time

class conductor:

    tarifa\_parado = 0.02

    tarifa\_movimiento = 0.05

class VTC(conductor):

    def \_\_init\_\_(self, descuento\_p = 0, descuento\_m = 0):

        self.tarifa\_movimiento += self.tarifa\_movimiento \* (descuento\_m/100)

        self.tarifa\_parado += self.tarifa\_parado \* (descuento\_p/100)

        self.tarifa\_movimiento = round(self.tarifa\_movimiento, 2)

        self.tarifa\_parado = round(self.tarifa\_parado, 2)

class taxista(conductor):

    def \_\_init\_\_(self, noche, prcnt = 0):

        if noche:

            self.tarifa\_parado += self.tarifa\_parado \* (prcnt/100)

            self.tarifa\_movimiento += self.tarifa\_movimiento \* (prcnt/100)

            self.tarifa\_parado = round(self.tarifa\_parado, 2)

            self.tarifa\_movimiento = round(self.tarifa\_movimiento, 2)

def tipo():

    #LogIn()

    conductor = input("Indica 'Taxista' o 'VTC'")

    if conductor.lower() == 'taxista':

        noche = input('¿Es de noche?(si/no)')

        if noche.lower() == 'si':

            tasa = input('Indica el procentaje extra de tarifa nocturna')

            mov = taxista(True, int(tasa))

        else:

            mov = taxista(False)

    else:

        desc\_mov = input("Indica la tasa de descuento en movimiento")

        desc\_par = input("Indica la tasa de descuento en parado")

        mov = VTC(int(desc\_mov), int(desc\_par))

    return(mov)

def precio(movimiento):

    tasas = tipo()

    precio = 0

    taximetro = True

    while taximetro and precio <= 1:

        if movimiento:

            precio += tasas.tarifa\_movimiento

            time.sleep(1)

            print(precio)

        else:

            precio += tasas.tarifa\_parado

            time.sleep(1)

            print(precio)

    return(round(int(precio), 2))

Archivo: GUI Entrada.py

import tkinter as tk

import pandas as pd

import hashlib

from tkinter import messagebox

from Check\_passwords import LogIn, Register, Pregunta, Respuesta

class GUI:

    def \_\_init\_\_(self):

        self.root = tk.Tk()

        self.title = self.root.title("Inicio de Sesion")

        self.root.geometry('500x500')

        self.root.configure(bg = '#824AB5')

        self.inicio = tk.Button(self.root, text = "Iniciar Sesion", font = ('Lucida Console', 16), command = self.login\_screen)

        self.inicio.pack()

        self.reg = tk.Button(self.root, text = "Registrarse", font = ('Lucida Console', 16), command = self.reg\_screen)

        self.reg.pack()

        self.root.mainloop()

    def login\_screen(self):

        for widget in self.root.winfo\_children():

            widget.destroy()

        self.label = tk.Label(self.root, text = "Inicie Sesion", font = ('Lucida Console', 16))

        self.user = tk.Entry(self.root, font = ('Lucida Console', 16))

        self.user.pack()

        self.password = tk.Entry(self.root, font =('Lucida Console', 16), show = '\*')

        self.password.pack()

        self.button = tk.Button(self.root, text = "Iniciar Sesion", font = ('Lucida Console', 16), command = self.check\_password)

        self.button.pack()

        self.button = tk.Button(self.root, text = "Olvidé mi contraseña", font = ('Lucida Console', 16), command = self.res\_pswd)

        self.button.pack()

    def reg\_screen(self):

        for widget in self.root.winfo\_children():

            widget.destroy()

        self.label = tk.Label(self.root, text = 'Registro', font = ('Lucida Console', 16))

        self.user = tk.Entry(self.root, font = ('Lucida Console', 16))

        self.user.pack()

        self.password = tk.Entry(self.root, font = ('Lucida Console', 16), show = '\*')

        self.password.pack()

        self.quest = tk.Entry(self.root, font = ('Lucida Console', 16))

        self.quest.pack()

        self.answ = tk.Entry(self.root, font = ('Lucida Console', 16))

        self.answ.pack()

        self.button = tk.Button(self.root, text = "Registro", font = ('Lucida Console', 16), command = self.register)

        self.button.pack()

    def res\_pswd(self):

        for widget in self.root.winfo\_children():

            widget.destroy()

        self.label = tk.Label(self.root, text = "Reiniciar contraseña", font = ('Lucida Console', 16))

        self.label.pack()

        self.user = tk.Entry(self.root, font = ('Lucida Console', 16))

        self.user.pack()

        self.quest = tk.Button(self.root, text = "Ver pregunta secreta", font = ('Lucida Console', 16), command = self.get\_quest)

        self.quest.pack()

        self.label2 = tk.Label(self.root, text = "Respuesta secreta", font = ('Lucida Console', 16))

        self.label2.pack()

        self.answer = tk.Entry(self.root, font = ('Lucida Console', 16))

        self.answer.pack()

        self.label3 = tk.Label(self.root, text = "Nueva contraseña", font = ('Lucida Console', 16))

        self.label3.pack()

        self.new\_pswd = tk.Entry(self.root, font = ('Lucida Console', 16))

        self.new\_pswd.pack()

        self.submit = tk.Button(self.root, text = "Enviar", font = ('Lucida console', 16), command = self.change\_pswd)

        self.submit.pack()

        self.back = tk.Button(self.root, text = "Atrás", font = ('Lucida Console', 16), command = self.login\_screen)

        self.back.pack()

    def check\_password(self):

        LogIn(self.user.get().lower(), self.password.get())

    def register(self):

         if Register(self.user.get().lower(), self.password.get(), self.quest.get(), self.answ.get().lower()) == 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())

GUI()

Archivo: initApp.py

import pandas as pd

import hashlib

import traceback

import datetime

# Especifica la ruta del archivo CSV donde se almacenan los datos de los usuarios.

path = "Usuarios.csv"

def register\_action(mensaje):

# Función para registrar acciones y errores.

# Registra mensajes de acciones y errores en un archivo de registro (registro.txt) con una marca de tiempo.

    with open("registro.txt", "a") as archivo:

        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        archivo.write(f"{timestamp} - {mensaje}\n")

def load\_data():

# Función que intenta cargar los datos del archivo CSV especificado en path.

# Si el archivo no existe, crea uno nuevo con las columnas Usuarios, Passwords, Pregunta Secreta y Respuesta Secreta.

    try:

        # Intentar leer el archivo CSV

        datos\_usuarios = pd.read\_csv(path)

    except FileNotFoundError:

        # Si el archivo no existe, crear uno nuevo

        print(f"El archivo {path} no se encontró. Creando una nueva base de datos.")

        datos\_usuarios = pd.DataFrame(columns=['Usuarios', 'Passwords', 'Pregunta Secreta', 'Respuesta Secreta'])

        datos\_usuarios.to\_csv(path, index=False)

    except Exception as e:

        print(f"Ha ocurrido un error inesperado al cargar los datos: {e}")

        datos\_usuarios = pd.DataFrame(columns=['Usuarios', 'Passwords', 'Pregunta Secreta', 'Respuesta Secreta'])

    return datos\_usuarios

def request\_input(mensaje, validacion=None):

# Solicita una entrada del usuario y valida la entrada usando una función de validación opcional

    while True:

        entrada = input(mensaje).strip()

        if validacion and not validacion(entrada):

            print("La contreña ha de tener como mínimo 8 caracteres. Inténtalo de nuevo.")

        else:

            return entrada

def validate\_password\_format(contrasena):

# Valida que la contraseña tenga al menos 8 caracteres

    return len(contrasena) >= 8

def request\_password(mensaje):

# Solicita una contraseña del usuario y valida que tenga al menos 8 caracteres

    return request\_input(mensaje, validate\_password\_format)

def register\_user(datos\_usuarios):

    '''

    Esta función registra un nuevo usuario:

    1. Solicita un nombre de usuario y verifica que no esté en uso.

    2. Solicita una contraseña, pregunta secreta y respuesta secreta.

    3. Guarda los datos del nuevo usuario en el archivo CSV.

    '''

    try:

        print("Iniciando el registro de usuario")

        user = request\_input("Escriba un nombre de usuario: ").lower()

        while user in datos\_usuarios["Usuarios"].str.lower().values:

            print("Ese nombre de usuario ya está seleccionado.")

            user = request\_input("Escriba un nombre de usuario diferente: ").lower()

        password = request\_password("Escribe una contraseña segura (mínimo 8 caracteres alfanuméricos): ")

        password\_hash = hashlib.sha256(password.encode('utf-8')).hexdigest()

        secret\_q = request\_input("Escriba una pregunta que solo usted conozca la respuesta: ").lower()

        secret\_a = hashlib.sha256(request\_input("Escriba la respuesta a su pregunta secreta: ").encode('utf-8')).hexdigest()

        nuevo\_usuario = pd.DataFrame({'Usuarios': [user], 'Passwords': [password\_hash], 'Pregunta Secreta': [secret\_q], 'Respuesta Secreta': [secret\_a]})

        datos\_usuarios = pd.concat([datos\_usuarios, nuevo\_usuario], ignore\_index=True)

        datos\_usuarios.to\_csv(path, index=False)

        print("Usuario registrado correctamente.")

        register\_action(f"Nuevo usuario registrado: {user}")

    except Exception as e:

        print(f"Ha ocurrido un error al registrar usuario: {e}")

        register\_action(f"Error al registrar usuario: {e}")

def validate\_password(datos\_usuarios, usuario):

    '''

    Esta función valida que la contraseña introducida por el usuario es correcta.

    En caso que no se introduzca la contraseña correcta en los 6 intentos, se pide al usuario que reinicie el programa

    '''

    intentos = 0

    password\_local = datos\_usuarios.loc[datos\_usuarios["Usuarios"].str.lower() == usuario, "Passwords"].values[0]

    while intentos < 6:

        password\_inp = request\_password("Escriba su contraseña: ")

        password\_inp\_hash = hashlib.sha256(password\_inp.encode('utf-8')).hexdigest()

        if password\_inp\_hash == password\_local:

            print("Bienvenido")

            register\_action(f"Usuario autenticado correctamente: {usuario}")

            return

        else:

            print("Contraseña incorrecta")

            register\_action(f"Contraseña incorrecta para usuario: {usuario}")

        intentos += 1

    print("Demasiados intentos fallidos, reinicie el programa")

def change\_password(datos\_usuarios, usuario):

    '''

    Cambia la contraseña del usuario:

    Verifica la respuesta a la pregunta secreta (hasta 3 intentos).

    Solicita una nueva contraseña que debe ser diferente a la actual.

    '''

    secret\_q = datos\_usuarios.loc[datos\_usuarios["Usuarios"].str.lower() == usuario, "Pregunta Secreta"].values[0]

    true\_answ = datos\_usuarios.loc[datos\_usuarios["Usuarios"].str.lower() == usuario, "Respuesta Secreta"].values[0]

    intentos = 0

    while intentos < 3:

        secret\_answ = hashlib.sha256(request\_input(secret\_q).encode('utf-8')).hexdigest()

        if secret\_answ == true\_answ:

            current\_pass\_hash = datos\_usuarios.loc[datos\_usuarios["Usuarios"].str.lower() == usuario, "Passwords"].values[0]

            new\_pass = request\_password("Introduce tu nueva contraseña (mínimo 8 caracteres alfanuméricos): ")

            new\_pass\_hash = hashlib.sha256(new\_pass.encode('utf-8')).hexdigest()

            while new\_pass\_hash == current\_pass\_hash:

                print("La nueva contraseña no puede ser igual a la actual.")

                new\_pass = request\_password("Introduce tu nueva contraseña diferente a la actual: ")

                new\_pass\_hash = hashlib.sha256(new\_pass.encode('utf-8')).hexdigest()

            datos\_usuarios.loc[datos\_usuarios["Usuarios"].str.lower() == usuario, "Passwords"] = new\_pass\_hash

            datos\_usuarios.to\_csv(path, index=False)

            print("Contraseña cambiada")

            register\_action(f"Contraseña cambiada para usuario: {usuario}")

            return

        else:

            print("Respuesta incorrecta")

            register\_action(f"Respuesta secreta incorrecta para usuario: {usuario}")

        intentos += 1

    print("Demasiados intentos fallidos, reinicie el programa")

    register\_action(f"Demasiados intentos fallidos para cambiar contraseña de usuario: {usuario}")

def login\_user(cambio=False):

    '''

    Gestiona el inicio de sesión o cambio de contraseña:

    Carga los datos de usuarios.

    Si cambio es True, cambia la contraseña; si no, valida la contraseña.

    '''

    datos\_usuarios = load\_data()

    try:

        usuario = request\_input("Escriba su nombre de usuario: ").lower()

        if cambio:

            if usuario not in datos\_usuarios["Usuarios"].str.lower().values:

                print("Este usuario no existe")

                register\_user(datos\_usuarios)

                register\_action(f"Intento de cambio de contraseña para usuario no existente: {usuario}")

            else:

                change\_password(datos\_usuarios, usuario)

        else:

            if usuario not in datos\_usuarios["Usuarios"].str.lower().values:

                print("Usuario no registrado.")

                register\_user(datos\_usuarios)

            else:

                validate\_password(datos\_usuarios, usuario)

    except Exception as e:

        register\_action(f"Error: {e}")

        register\_action(traceback.format\_exc())

        print("Ha ocurrido un error. Por favor, revise el registro.")

def start():

    '''

    Llama a la función login\_user con el parámetro cambio

    Si cambio=True para start el proceso de cambio de contraseña al ejecutar el programa.

    Si cambio=False se inicia el proceso para start sesion en la aplicación

    '''

    #login\_user(False)

    login\_user(True)

start()

Archivo: IU\_Class.py

import pygame

from sys import exit

import pygame\_gui

import time

from datetime import date

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.quit = Quit(self.screen, self.gameStateManager, self.user)

        self.states = {'start': self.start,

                       'taximetro': self.taximetro,

                       'intro': self.intro,

                       '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/base\_2.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/base\_2.jpeg')

        self.display.blit(fondo, (0, 0))

        texto = pygame.image.load('Graficos/Intro\_text (Mediana).png')

        self.display.blit(texto, (100, 100))

class Taximetro:

    def \_\_init\_\_(self, display, gameStateManager):

        self.display = display

        self.gameStateManager = gameStateManager

        self.car = pygame.image.load('Graficos/car1.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

    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('quit')

    def run(self):

        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 += 0.05 / 60  # Incrementar la puntuación por segundo en movimiento

            else:

                self.score += 0.02 / 60  # Incrementar la puntuación por segundo en parado

            self.display.blit(self.car, (self.car\_position, 700))

            # 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))

    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 Quit:

    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

    def handle\_quit(self):

        self.gameStateManager.set\_state('quit')

    def precio\_final(self):

        self.display.fill(self.color\_background)

        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, 350))

        time\_text\_rect = time\_text.get\_rect(center = (800, 450))

        self.display.blit(price\_text, price\_text\_rect)

        self.display.blit(time\_text, time\_text\_rect)

        pygame.display.update()

        pygame.time.wait(3000)

    def run(self):

        self.final\_price = self.gameStateManager.get\_states()['taximetro'].get\_score()

        self.total\_time = self.gameStateManager.get\_states()['taximetro'].get\_total\_time()

        self.tiempo\_minutos = self.total\_time // 60

        self.tiempo\_segundos = self.total\_time % 60

        self.precio\_final()

        self.today = date.today()

        self.d1 = self.today.strftime("%d/%m/%Y")

        datos\_usuarios = pd.read\_csv('Carreras.csv')

        df = pd.DataFrame({'Usuario' : [self.user], 'Fecha': [self.d1], 'Tiempo\_Minutos' : [self.tiempo\_minutos], 'Tiempo\_Segundos': [self.tiempo\_segundos],'Precio': [round(self.final\_price, 2)]})

        datos\_usuarios = pd.concat([datos\_usuarios, df], ignore\_index = True)

        datos\_usuarios.to\_csv('Carreras.csv', index = False)

        pygame.quit()

        exit()

def init\_game(user):

    game = Game(user)

    game.run()

Archivo: LogIn Screen.py

import pygame

from sys import exit

pygame.init()

#Tamaño de pantalla del juego

screen = pygame.display.set\_mode((800, 400))

#Texto que figura en la ventana ejecutada

pygame.display.set\_caption('Taximetro Interactivo')

#Reloj interno, para configurar la tasa de refreso del juego (fps)

clock = pygame.time.Clock()

first\_screen = pygame.image.load('Graficos/road 2.jpg')

#Boton de LogIn

#Fuente para el boton de LogIn

login\_font = pygame.font.SysFont('Lucida Console', 45)

#Rectangulo para el boton, (posicion ancho, posicion alto, tamaño ancho, tamaño alto)

login\_button\_rect = pygame.Rect(585, 70, 170, 62)

#Superficie donde se coloca el boton

login\_text = login\_font.render('Log In', True, (128, 245, 10, 1))

#Boton de registo

reg\_font = pygame.font.SysFont('Lucida Console', 45)

reg\_but\_rect = pygame.Rect(560, 170, 220, 62)

reg\_text = reg\_font.render('Registro', True, (128, 245, 10, 1))

#Boton de quit

quit\_font = pygame.font.SysFont('Lucida Console', 45)

quit\_but\_rect = pygame.Rect(610, 270, 120, 62)

quit\_text = quit\_font.render('Quit', True, (128, 245, 10, 1))

#Bucle de ejecución del juego

while True:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            pygame.quit()

            exit()

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

            if quit\_but\_rect.collidepoint(event.pos):

                pygame.quit()

                exit()

    a,b = pygame.mouse.get\_pos()

    if login\_button\_rect.x <= a <= login\_button\_rect.x + 190 and login\_button\_rect.y <= b <= login\_button\_rect.y + 70:

        pygame.draw.rect(screen, (91 ,23 ,202, 0.8), login\_button\_rect) #sin hover

    else:

        pygame.draw.rect(screen, (65, 0, 168, 0.9), login\_button\_rect) #con hover

    if reg\_but\_rect.x <= a <= reg\_but\_rect.x + 210 and reg\_but\_rect.y <= b <= reg\_but\_rect.y + 70:

        pygame.draw.rect(screen, (91 ,23 ,202, 0.8), reg\_but\_rect)

    else:

        pygame.draw.rect(screen, (65, 0, 168, 0.9), reg\_but\_rect)

    if quit\_but\_rect.x <= a <= quit\_but\_rect.x + 210 and quit\_but\_rect.y <= b <= quit\_but\_rect.y + 70:

        pygame.draw.rect(screen, (91 ,23 ,202, 0.8), quit\_but\_rect)

    else:

        pygame.draw.rect(screen, (65, 0, 168, 0.9), quit\_but\_rect)

    screen.blit(first\_screen, (0, 0))

    screen.blit(login\_text, (login\_button\_rect.x + 5, login\_button\_rect.y+5))

    screen.blit(reg\_text, (reg\_but\_rect.x + 5, reg\_but\_rect.y+5))

    screen.blit(quit\_text, (quit\_but\_rect.x + 5, quit\_but\_rect.y+5))

    pygame.display.update()

    clock.tick(60)

Archivo: Password database.py

import pandas as pd

import hashlib

#Cuando tengamos una base de datos como tal

#datos\_usuarios = pd.read\_csv("usuarios.csv")

def LogIn(cambio = False):

    '''

    LogIn va a permitir un argumento 'cambio' que por defecto será falso, para poder cambiar la contraseña.

    La función en primer lugar va a pedir un nombre al usuario y comprobará mediante condicional if si existe ese nombre en nuestra base de datos.

    Si el usuario no existe, entendemos que es un usuario nuevo y le pedimos que nos de un nombre

    La función comprueba que el nombre no esté ya en uso, y mientras no se de un nombre nuevo seguira pidiendolo en bucle.

    Después se le pide una contraseña "segura", podríamos definir que significa que su contraseña sea segura e incluso solicitar una longitud concreta (o carácteres)

    Una vez tenemos la contraseña, esta se hashea utilizando sha-256 y se almacena de forma encriptada en nuestra base de datos, para aumentar la seguridad.

    Para permitir cambiar la contraseña, se introduce una pregunta y respuesta secreta definidas por el usuario, y que deberá responder correctamente en el caso de necesitar cambiarla. Se permiten 3 intentos para acertar la respuesta.

    Finalmente el usuario, su contraseña encriptada y la pregunta y respuesta secretas se añaden a la base de datos de usuarios.

    Si el usuario si existe, se le pedirá que introduzca su contraseña y está se buscara en la base de datos en su forma encriptada.

    Si hay coincidencias el programa de taximetro se inicia, si no, se le pide que repita la contraseña, hasta un máximo de 6 intentos.

    '''

    datos\_usuarios = pd.read\_csv("Usuarios.csv")

    if cambio:

        usuario = input("Escriba su nombre de usuario: ")

        if usuario not in datos\_usuarios["Usuarios"].values:

            print("Este usuario no existe")

        else:

            secret\_q = datos\_usuarios.query('Usuarios == @usuario')["Pregunta Secreta"][1]

            secret\_answ = hashlib.sha256(input(secret\_q).encode('utf-8')).hexdigest()

            true\_answ = datos\_usuarios.query('Usuarios == @usuario')["Respuesta Secreta"][1]

            intentos = 0

            if secret\_answ == true\_answ:

                new\_pass = input("Introduce tu nueva contraseña: ")

                new\_pass\_hash = hashlib.sha256(new\_pass.encode('utf-8')).hexdigest()

                datos\_usuarios.query('Usuarios == @usuario')["Pregunta Secreta"][1]

                datos\_usuarios.loc[datos\_usuarios["Usuarios"] == usuario, "Passwords"] = new\_pass\_hash

                datos\_usuarios.to\_csv('Usuarios.csv', index = False)

                print("Contraseña Cambiada")

            else:

                while (secret\_answ != true\_answ):

                    if intentos == 4:

                        print("Demasiados intentos fallados, reinicie el programa")

                        break

                    else:

                        print("Respuesta incorrecta")

                        secret\_answ = input(secret\_q)

                        intentos += 1

    else:

        usuario = input("Escriba su nombre de usuario: ")

        if usuario not in datos\_usuarios["Usuarios"].values:

            print("Usuario no registrado.")

            user = input("Escriba un nombre de usuario")

            while user in datos\_usuarios.Usuarios.isin([user]):

                user = input("Ese nombre de usuario ya está seleccionado. Escriba un nombre de usuario.")

            password = input("Escribe una contraseña segura")

            password\_hash = hashlib.sha256(password.encode('utf-8')).hexdigest()

            secret\_q = input("Escriba una pregunta que solo usted conozca la respuesta")

            secret\_a = hashlib.sha256(input("Escriba la respuesta a su pregunta secreta").encode('utf-8')).hexdigest()

            df = pd.DataFrame({'Usuarios': [user], 'Passwords': [password\_hash], 'Pregunta Secreta': secret\_q, 'Respuesta Secreta': secret\_a})

            datos\_usuarios = pd.concat([datos\_usuarios, df], ignore\_index= True)

            datos\_usuarios.to\_csv('Usuarios.csv', index = False)

        else:

            password\_inp = input("Escriba su contraseña: ")

            password\_inp\_hash = hashlib.sha256(password\_inp.encode('utf-8')).hexdigest()

            intentos = 0

            password\_local = datos\_usuarios.query('Usuarios == @usuario')["Passwords"][1]

            while(password\_inp\_hash != password\_local):

                if intentos == 6:

                    print("Demasiados intentos fallados, reinicie el programa")

                    break

                else:

                    print("Contraseña incorrecta")

                    password\_inp = input("Escriba su contraseña: ")

                    password\_inp\_hash = hashlib.sha256(password\_inp.encode('utf-8')).hexdigest()

                    intentos += 1

            else:

                print("Bienvenido")

                #taximetro()

def Iniciar(cambio = False):

    LogIn(cambio)

#Iniciar()

#Iniciar(True) Cambio de contraseña

Archivo: Taxi.ipynb

import time

class Taximetro:

    def \_\_init\_\_(self):

        self.costo\_total = 0.0

        self.en\_carrera = False

        self.en\_movimiento = False

        self.ultimo\_tiempo = None

        self.costo\_por\_segundo\_parado = 0.02

        self.costo\_por\_segundo\_movimiento = 0.05

    def iniciar\_carrera(self):

        self.en\_carrera = True

        self.en\_movimiento = False

        self.ultimo\_tiempo = time.time()

        print("La carrera ha comenzado. El taxi está parado.")

    def terminar\_carrera(self):

        self.actualizar\_costo()

        self.en\_carrera = False

        print(f"La carrera ha terminado. El total es {self.costo\_total:.2f} Euros.")

        self.reiniciar()

    def comenzar\_movimiento(self):

        if self.en\_carrera and not self.en\_movimiento:

            self.actualizar\_costo()

            self.en\_movimiento = True

            self.ultimo\_tiempo = time.time()

            print("El taxi está en movimiento.")

    def detener\_movimiento(self):

        if self.en\_carrera and self.en\_movimiento:

            self.actualizar\_costo()

            self.en\_movimiento = False

            self.ultimo\_tiempo = time.time()

            print("El taxi está parado.")

    def actualizar\_costo(self):

        if self.en\_carrera:

            tiempo\_actual = time.time()

            tiempo\_transcurrido = tiempo\_actual - self.ultimo\_tiempo

            if self.en\_movimiento:

                self.costo\_total += tiempo\_transcurrido \* self.costo\_por\_segundo\_movimiento

            else:

                self.costo\_total += tiempo\_transcurrido \* self.costo\_por\_segundo\_parado

            self.ultimo\_tiempo = tiempo\_actual

    def reiniciar(self):

        self.costo\_total = 0.0

        self.en\_carrera = False

        self.en\_movimiento = False

        self.ultimo\_tiempo = None