

Ingeniería informática
2023-2024



Criptografía y seguridad informática

“Práctica 1”

Grupo 11

Fernando Consiglieri Alcántara
100472111@alumnos.uc3m.es

David Andrés Yáñez Martínez
100451958@alumnos.uc3m.es

24 de octubre de 2023

REPOSITORIO: [Enlace al repositorio para la primera entrega de la práctica](#)

PROPÓSITO DE LA APLICACIÓN

La aplicación se llama “*Titanomachy*” es un RPG asíncrono de combate por turnos en el que los usuarios se registran (Véase [ANEXO 1](#)), escogen su personaje (Véase [ANEXO 2](#)) y a través de un buscador de partida combaten contra otros jugadores (Véase [ANEXO 3](#)).

Una vez dentro de la partida, el jugador al que le toca podrá elegir entre varios ataques de su personaje para atacar, defenderse o curarse. Cuando sea el turno del otro jugador, este podrá atacar y se repetirá esta secuencia hasta que uno de los dos jugadores se quede sin puntos de salud (Véase [ANEXO 4](#)).

Se notifica el ganador de la partida y ambos jugadores podrán volver a buscar un nuevo combate (Véase [ANEXO 5](#)).

La aplicación está hecha a partir del módulo Tkinter que nos permite diseñar de forma sencilla una interfaz gráfica.

AUTENTICACIÓN DE USUARIOS

El proceso comienza con el registro del usuario (Véase [ANEXO 1](#)). El usuario proporciona un nombre de usuario o "nickname" y una contraseña. Verificamos que el nombre no exista ya en el archivo JSON de usuarios y que la contraseña cumpla con una política de contraseñas fuertes. Una vez registrado, almacenamos su nombre y un hash de su contraseña, que se convierte a hexadecimal. Este proceso protege al usuario, ya que el hash es unidireccional y no puede ser revertido para obtener la contraseña original.

Para la autenticación del usuario (Véase el [ANEXO 6](#)), el proceso es más sencillo. Al iniciar sesión, el usuario proporciona su contraseña. Entonces, volvemos a generar el hash de la contraseña utilizando la misma “salt” que se usó originalmente. Si el resultado coincide con el nombre almacenado en el archivo JSON, se completa la autenticación (Véase [ANEXO 7](#)).

Algoritmos Utilizados

Para el "hasheo" de la contraseña, hemos implementado una función HMAC (Véase [ANEXO 8](#)), incluida en la librería "Cryptography". A continuación, se detalla el algoritmo paso a paso:

- Generamos una “salt” de 16 bytes de valor aleatorio.
- Convertimos la contraseña proporcionada en una secuencia de bytes.
- Creamos un objeto llamado “kdf”, donde se especifica el algoritmo de hash, SHA256; la longitud de la clave derivada, 32; el valor de la “salt” y la cantidad de iteraciones que se realizarán, 10000.
- Utilizamos el algoritmo PBKDF2 para derivar una clave secreta a partir de la contraseña y la “salt”, aumentando así su seguridad.
- La clave secreta se convierte en una cadena hexadecimal antes de ser devuelta.

En resumen, esta función toma una contraseña en texto plano, la procesa mediante un algoritmo de hash seguro (PBKDF2 con SHA-256) y devuelve una versión hexadecimal de la contraseña para almacenarla en el archivo JSON. Además, también almacenamos la “salt” generada en hexadecimal, que será vital para la autenticación (Véase [ANEXO 9](#)).

CIFRADO SIMÉTRICO

Dentro del videojuego, es importante que un jugador no sea capaz de saber el movimiento que va a hacer su rival antes de tiempo porque podría anticiparse a este poniéndole en una posición de ventaja. Por ese motivo, el nombre del ataque se encripta antes de que la otra persona reciba el golpe. Al mantener un secreto esta información, se fomenta un ambiente de juego justo y estratégico donde la toma de decisiones se basa en la habilidad y adaptación en tiempo real enriqueciendo así la experiencia de juego.

Nuestro enfoque en el contexto de criptografía se basa en el uso del módulo “cryptography” que proporciona una implementación del algoritmo Fernet para el cifrado. Hemos utilizado Fernet porque simplifica en gran medida la implementación de la criptografía sin comprometer la solidez de su seguridad ni la integridad de sus datos.

En cuanto a la gestión de claves, para garantizar la seguridad, generamos una clave aleatoria utilizando ‘Fernet.generate_key()’, esta clave se utiliza tanto para el cifrado como para el descifrado de los datos. Una vez generada, creamos un objeto Fernet utilizando esta clave, el ataque se cifra utilizando la clave y se almacena como una cadena hexadecimal en una variable denominada ‘token’. Tanto esta variable como la clave se guardan en un archivo JSON ya que son las que nos permitirán descifrar los datos más tarde, si perdiésemos alguno de estos, nos sería imposible recuperar el nombre del ataque utilizado por el usuario (Véase [ANEXO 10](#)) (Véase [ANEXO 11](#)).

AUTENTICACIÓN

Todo lo relacionado con el autenticado de mensajes es una función que ya viene implementada en fernet, es decir mientras ejecutamos el cifrado simétrico de los mensajes se está autenticando. Para la gestión de claves fernet proporciona un formato estándar para las claves cifradas, lo que facilita su almacenamiento y distribución de manera segura. Toda esta información viene explicada en su página oficial, [aquí](#).

FUTURAS POSIBLES IMPLEMENTACIONES PARA MEJORAR EL PROYECTO:

- Encriptado del JSON para saber si ha sido modificado desde la última vez que se abrió la aplicación
- Para una mayor seguridad en el cifrado simétrico hemos decidido encriptar la “key” para que no sea tan vulnerable. Pues la pérdida o el robo de esa clave supondría una exposición total de los datos.

ANEXOS

ANEXO 1

Titanomachy

Registro

Username

Password

Login

ANEXO 2

Titanomachy


Elige un personaje


Nombre del personaje:


SUSANOO


Finalizar


Buscar Partida











Ataques del personaje:

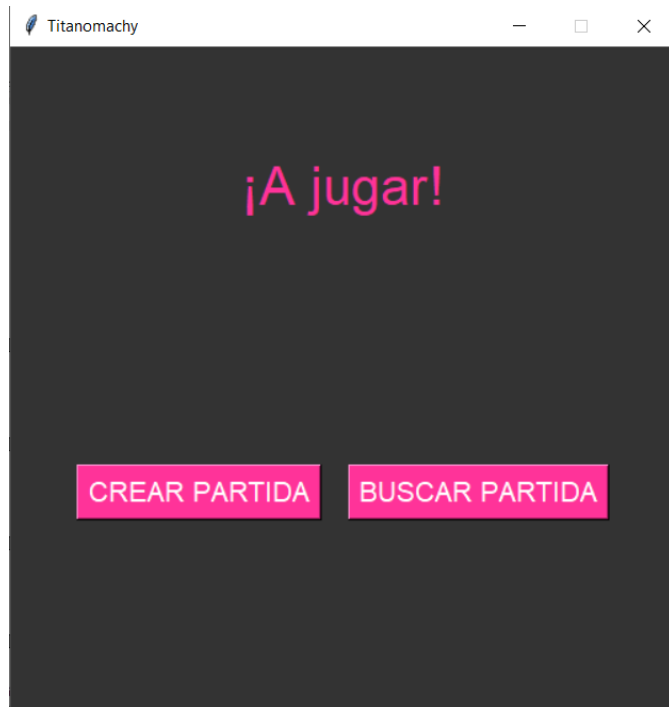
CORTE ACUATICO

PURIFICACION

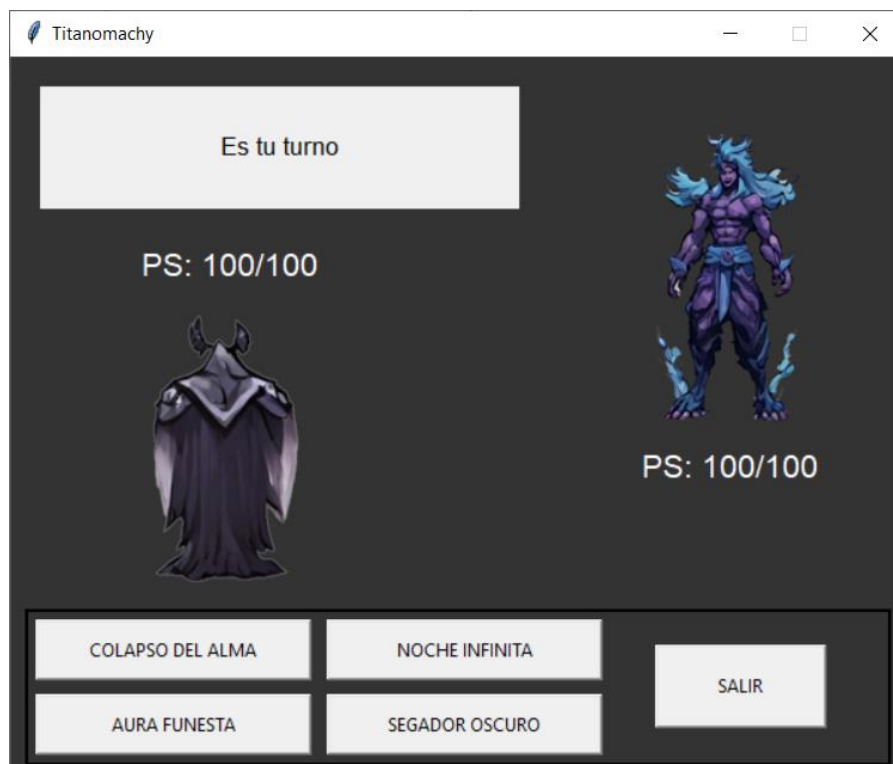
TORMENTA ELECTRICA

SEISMO

ANEXO 3



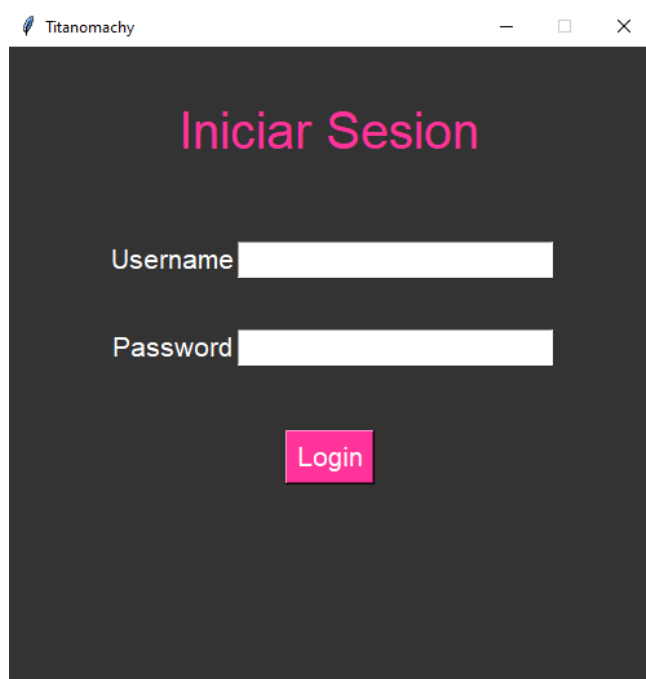
ANEXO 4



ANEXO 5



ANEXO 6



ANEXO 7

```
def input_handler(self, username_entry, password_entry):
    """Funcion que comprueba si el log in se encuentra en el json"""
    self.user = username_entry.get()
    self.password = password_entry.get()
    if self.register.user_exists(self.user):
        salt = self.register.get_salt_from_user(self.user)
        if self.register.get_password_from_user(self.user) == self.register.hash_password(self.password, salt):
            if self.register.get_character_from_user(self.user) == None:
                messagebox.showinfo("Info", "Falta elegir personaje")
                CharacterChooser(self.window, self.user)
            else:
                messagebox.showinfo("Info", "Usuario logeado correctamente")
                SearchMatch(self.window, self.user, self.register.get_character_from_user(self.user))
        else:
            messagebox.showerror("Error", "Contraseña incorrecta")
    else:
        messagebox.showerror("Error", "El usuario no existe")
```

ANEXO 8

```
def hash_password(self, password, salt):
    """Funcion que aplica una funcion hash a la contraseña"""
    password = password.encode()
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
    )
    key = kdf.derive(password)
    return key.hex()

def save_data(self, user, password):
    """Funcion que guarda los datos en el json"""
    # Agregar el nuevo usuario a la lista de usuarios
    self.load_data()
    #hash password and save
    salt = os.urandom(16)
    password = self.hash_password(password, salt)
    salt_hex = salt.hex()
    new_user = {"user": user, "password": password, "salt": salt_hex, "character": None}
    self.data.setdefault("Client Register", []).append(new_user)
    self.save()
```

ANEXO 9

```
{
  "user": "David123@",
  "password": "93014667efb3056937db0bd7da79557f44e9a6a8c10eb60a9f683f97a7e34ed3",
  "salt": "e32fe22f6b7800b546d36377503ec7a4",
  "character": "VISHNU"
},
{
  "user": "David11@",
  "password": "310413e71bac3c17e866b18b18bd024f0be9a83e1760f1a7552c9cb7785ab31f",
  "salt": "5686809271af878872b7704e1bcace2a",
  "character": "VISHNU"
},
{
  "user": "ferjo",
  "password": "1fb83d75aa17b01983835491679752d1ef4e5576f8502fbab31f8114528ac452",
  "salt": "56344dc83bfb0818f4b902014f3492a",
  "character": "THANATOS"
},
{
  "user": "ferjodio",
  "password": "0d1c646e825e64e45280d072a3381afde8371261e7ca84ea2da037132d1e7807",
  "salt": "1c22708868c94737869ed97149364a4a",
  "character": "VISHNU"
},
```

ANEXO 10

```
def encryptar_ataque(self, ataque, game):
    key = Fernet.generate_key()
    f = Fernet(key)
    token = f.encrypt(ataque.encode('utf-8'))

    game["cripto"]["token"] = token.hex()
    game["cripto"]["key"] = key.hex()
    return game

def get_atack_from_token(self, game):
    if game["cripto"]["key"] != "":
        key = bytes.fromhex(game["cripto"]["key"])
        token = bytes.fromhex(game["cripto"]["token"])
        f = Fernet(key)
        return f.decrypt(token).decode('utf-8')
    else:
        return ""
```

ANEXO 11

```
{
    "id_partida": 12,
    "id_jugador1": "ferjos",
    "id_jugador2": "ferjudio",
    "juego_activo": true,
    "cripto": {
        "token": "674114141414126c4e39646e486a65756e2d5039433677f52414c69482d5",
        "key": "3733576f454158354776b5f655272695942736b2d6f5651426b335a7ab344"
    },
    "datos_juego": {
        "personaje1": "THANATOS",
        "personaje2": "SUSANOO",
        "turno": "Jugador 1"
    },
    "stats1": {
        "VIDA": 100,
        "DEFENSA": 35,
        "RESISTENCIA": 35,
        "EVASION": 10,
        "ATAQUE": 15
    },
    "stats2": {
        "VIDA": 100,
        "DEFENSA": 15,
        "RESISTENCIA": 15,
        "EVASION": 10,
        "ATAQUE": 40
    }
}
```