TFG PokeFarm

1. Introducción..... 3
1.1 Motivación del Proyecto..... 3
1.2 Objetivo del proyecto..... 3
2. ¿Qué es PokeFarm?..... 4
2.1 ¿Para qué sirve?..... 4
2.2 ¿Cómo funciona?..... 4
2.3 ¿Requisitos para utilizarlo?..... 5
3.1 Funcionamiento..... 6
3.2 Python y librerias..... 7
3.2.1 OpenCV..... 7
3.2.2 PyAutoGUI..... 8
3.2.3 Keyboard..... 8
3.2.4 Threading..... 8
3.2.5 NumPy..... 9
3.2.6 MSS..... 9
3.2.7 Time..... 9
3.2.8 Tkinter..... 10
3.3 Supabase..... 10
3.4 React + Tailwind..... 11
4.1 Layout principal..... 11
4.2 Pestaña de configuración..... 12
5. Back-End..... 14
5.1 Funciones principales..... 14
5.2 Funciones secundarias..... 16
5.3 Flujo de Ejecución..... 19
6. Problemas y Soluciones..... 20
6.1 Solapamiento de movimientos..... 21
6.2 Detección por pantalla..... 21
6.3 Manejo de errores..... 21
7. Futuras mejoras..... 22
7.1 Añadir nuevos scripts..... 22
7.2 Mejorar la eficiencia..... 22
8. Licencia..... 22

1. Introducción

1.1 Motivación del Proyecto.

Antes de entrar al Grado Superior de Desarrollo de Aplicaciones Multiplataforma jugaba a un juego llamado PokeMMO. Este es un juego basado en Pokémon pero Multijugador Masivo, es decir, juegas con otras personas de todo el mundo. Jugando me di cuenta que habían acciones muy repetitivas que consumen mucho tiempo y empecé a pensar en alguna idea de automatizarlo.

De esa idea empecé a programar en Python un script que automatiza ciertas acciones del juego, este era un script básico, muy sucio y con errores pero funcionaba lo suficientemente bien como para utilizarlo. A partir de este pequeño programa que hice decidí entrar al grado para seguir aprendiendo programación y por eso decidí desarrollar esto como proyecto final.

1.2 Objetivo del proyecto.

- Crear una aplicación simple, eficaz y personalizable para automatizar ciertas acciones del juego
- Hacer un análisis de requisitos previos para luego codificar el software de la manera más efectiva posible.
- Crear una aplicación simple, eficaz y personalizable para automatizar ciertas acciones del juego
- Crear un manual de usuario gracias al cual cualquier usuario sea capaz de usar la aplicación

- Diseñar una interfaz intuitiva que permita al usuario usar la aplicación en su plenitud sin tener ningún conocimiento avanzado.

2. ¿Qué es PokeFarm?

PokeFarm es una aplicación desarrollada en **Python** que automatiza acciones en el videojuego **PokeMMO** mediante reconocimiento de imágenes en pantalla y simulación de entradas de teclado. Para ello, utiliza una carpeta con patrones de imagen que busca en la pantalla y ejecuta funciones que reproducen los movimientos correspondientes dentro del juego.

2.1 ¿Para qué sirve?

Permite automatizar tareas tediosas en **PokeMMO**, como capturar Pokémon o entrenarlos hasta subir de nivel. Estas acciones son altamente repetitivas y pueden llegar a ocupar horas si se realizan manualmente. Con esta aplicación, basta con pulsar un botón para ejecutarlas de forma automática, simplificando así la experiencia de juego.

2.2 ¿Cómo funciona?

La aplicación es muy sencilla de usar: el jugador únicamente debe introducir su configuración de PokeMMO y, a continuación, elegir una de las dos opciones principales:

- **Iniciar EXP**: pone en marcha un bucle automático que entrena a los Pokémon hasta subirlos de nivel.

- **Iniciar Farm**: ejecuta un ciclo de captura continua de Magikarps.

Internamente, el programa dispara un flujo de funciones que combinan la simulación de entradas de teclado (por ejemplo, pulsar WASD para mover al personaje) con el reconocimiento de pantalla para interpretar lo que sucede en el juego (por ejemplo, detectar si la captura ha tenido éxito). Este proceso se repite de forma automática hasta alcanzar los objetivos configurados.

2.3 ¿Requisitos para utilizarlo?

PokeMMO instalado con las siguientes zonas desbloqueadas:

- **Zona Safari**: necesaria para la función de “Farm” (captura de Pokémon).
- **Región de Teselia**: imprescindible para la función de “Iniciar EXP” (entrenamiento).

Para “Iniciar EXP”

- Se recomienda contar con un Pokémon que disponga de **Surf** u otra habilidad de área que permita eliminar hordas de enemigos de un solo golpe.
- Esto acelera enormemente el ciclo de combate y optimiza el flujo de entrenamiento automático.

Para “Iniciar Farm” (Magikarp)

- Basta con tener la **Zona Safari** desbloqueada y la **caña vieja** en el inventario.

- No se requieren objetos adicionales; el script gestionará automáticamente cada intento de captura.

3. Tecnologías utilizadas

Aquí cubriremos qué tecnologías y metodología he utilizado para llevar a cabo la aplicación y porque he elegido esas y no otras. Cubriremos tanto lenguaje de programación principal y secundarios, base de datos, librerías, metodología de la aplicación…

3.1 Funcionamiento

La primera aproximación a la automatización fue la **lectura directa de la memoria** del juego. Este método permitía conocer con exactitud nuestra posición, el entorno y el estado de la partida, reduciendo el margen de error al mínimo. Sin embargo, PokeMMO es un emulador basado en **Java**, lo que complica enormemente el acceso y la interpretación de sus estructuras de memoria. Por ello, opté por técnicas alternativas basadas en **simulación de entradas** y **reconocimiento de pantalla**: conceptualmente más sencillas de implementar, aunque exigen un mejor manejo de las excepciones .

La aplicación se basa en dos mecánicas principales:

1. **Simulación de entradas de teclado**

Utilizando librerías como **keyboard** y **PyAutoGUI**, el programa envía pulsaciones al juego (por ejemplo, “pulsar W”, “soltar W” o “pulsar O”). Gracias a esto, puede reproducir la mayoría de los movimientos y acciones del personaje sin intervención manual.

2. **Reconocimiento de pantalla**

Con librerías como **OpenCV** se capturan fragmentos de la

ventana de PokeMMO, se convierten a escala de grises y se comparan con imágenes guardadas en la carpeta **assets**. De este modo, el sistema detecta en qué punto del flujo de automatización se encuentra (por ejemplo, al localizar la imagen “¡Pokémon ha sido capturado!” sabe que puede reiniciar el ciclo).

3.2 Python y librerias

He elegido Python como lenguaje para desarrollar la aplicación ya que es simple, tiene todas las librerías necesarias para llevar a cabo la aplicación y ya tenía conocimientos previos que me han ayudado a llevar a cabo la aplicación.

En cuanto a librerías hemos utilizado varias que nos han ayudado a hacer la aplicación, dentro de estas tenemos OpenCV, PyAutoGUI, Keyboard, Threading, Numpy…

3.2.1 OpenCV

Biblioteca de visión por computador que utilizamos para:

- **Capturar** y procesar imágenes del juego.
- **Convertir** capturas a escala de grises o aplicar umbrales para facilitar la comparación.
- **Ventajas**: alta velocidad en operaciones de imagen y gran variedad de funciones de procesamiento.
- **Ejemplos** de uso son **cv2.cvtColor** para cambiar a escala de grises o **cv2.matchTemplate** para saber el porcentaje de similitud

3.2.2 PyAutoGUI

Herramienta para **simular** entradas de ratón y teclado:

- **Mover** y hacer click con el cursor a coordenadas específicas dentro de la ventana de PokeMMO.
- **Ventajas**: multiplataforma, fácil de usar y con control de pausas/retardos que ayudan a emular comportamiento humano.
- **Ejemplos** de uso son **pyautogui.leftClick()** para hacer click izquierdo o **pyautogui.moveTo()** para mover el ratón a una posición en concreto.

3.2.3 Keyboard

Librería ligera para **gestionar** eventos de teclado:

- **Escuchar** simultáneamente combinaciones de teclas (por ejemplo, “pausa automática con Ctrl+P”).
- **Pulsar,** mantener o soltar ciertas en especifico.
- **Ventajas**: muy precisa para captura y envío de eventos, complementa a PyAutoGUI cuando necesitamos un control más fino de teclado.
- **Ejemplos** de uso son **keyboard.press(x)** x siendo el boton que queremos presionar o **keyboard.release(x)** x siendo el boton que queremos dejar de presionar.

3.2.4 Threading

Módulo estándar de Python para **concurrencia**:

- **Ejecutar** en paralelo el bucle de captura de pantalla y la simulación de entradas, evitando bloqueos.
- **Permitir** interrupciones limpias (por ejemplo, detectar una tecla de “emergencia” mientras continúa el farmeo).

3.2.5 NumPy

Biblioteca de arrays y operaciones numéricas, empleada sobre todo en el procesamiento de imagen:

- **Convertir** regiones de la captura de pantalla a arrays para acelerar comparaciones pixel a pixel.
- **Ventajas**: optimización de cálculos matriciales frente a listas nativas de Python.
- **Ejemplos** de uso son **np.array(x)** para convertir una imagen a array of **np.where(**condición) para hacer una especie de condicional

3.2.6 MSS

Librería especializada en **captura rápida de pantalla**:

- **Grabación** de frames a alta velocidad directamente desde el framebuffer, con menos sobrecarga que métodos basados en PIL.
- **Ventajas**: muy baja latencia, ideal para aplicaciones que necesitan actualizar el estado del juego decenas de veces por segundo.
- **Ejemplos** de uso son **mss.grab(mss.monitor[x])** para hacer una captura a la pantalla.

3.2.7 Time

Módulo estándar para **gestión de temporización**:

- **Introducir** pausas (time.sleep) entre acciones para simular comportamiento humano y evitar detecciones anti-bot.
- **Ventajas**: imprescindible para controlar flujos de ejecución y coordinarlos con eventos de juego.

3.2.8 Tkinter

Módulo simple para crear interfaces en Python.

- **Crear** de manera simple una interfaz funcional en Python.
- **Ventajas**: simple de utilizar y aprender comparado a otras bibliotecas centradas en la creación de interfaces.

3.3 Supabase

Supabase es una alternativa a Firebase que te proporciona varias herramientas como autenticación, base de datos… para el backend de tu web o aplicación.

En nuestro caso hemos utilizado su método de autenticación mediante gmail y contraseña para iniciar sesión dentro de la aplicación o registrarse en la web.

La conexión se hace mediante tu URL y tu código Anon que te proporciona SupaBase. Con estos dos haces la conexión en el código de la aplicación.

3.4 React + Tailwind

Para la web de descargas hemos elegido **React** con **TypeScript** por su arquitectura de componentes y tipado estático, lo que facilita el desarrollo y mantenimiento. Para el diseño visual empleamos **Tailwind CSS**, aprovechando su enfoque utility-first para crear interfaces limpias y adaptativas sin necesidad de escribir CSS personalizado.

- **React + TypeScript**: componentes reutilizables, detección temprana de errores y librerías como React Router y React Query para navegación y gestión de datos.

- **Tailwind CSS**: clases utilitarias para márgenes, colores y tipografías…

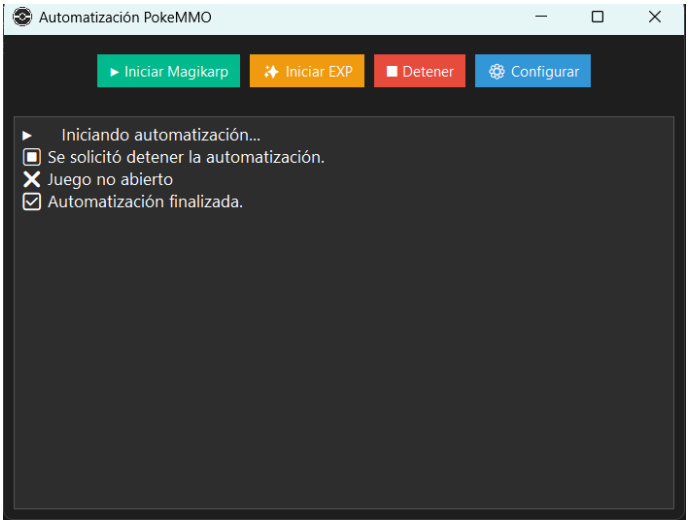
Gracias a esta combinación obtenemos una web ligera, rápida de iterar y fácil de mantener, sin complicarnos con archivos CSS extensos ni configuraciones complejas.

4. Interfaz de Usuario

4.1 Layout principal

Este es el layout principal que vemos nada más iniciar sesión en la aplicación. Podemos ver 4 botones principales y destacados.

1. **Iniciar Magikarp** que automáticamente inicia el farmeo de Magikarps
2. **Iniciar EXP** que automáticamente empieza el farmeo de Experiencia explicado anteriormente.
3. **Detener** es el botón que pulsaremos cuando queramos detener el código en cualquier momento.
4. **Configurar** nos abre un panel donde podemos personalizar los botones que usa la aplicación y los assets.



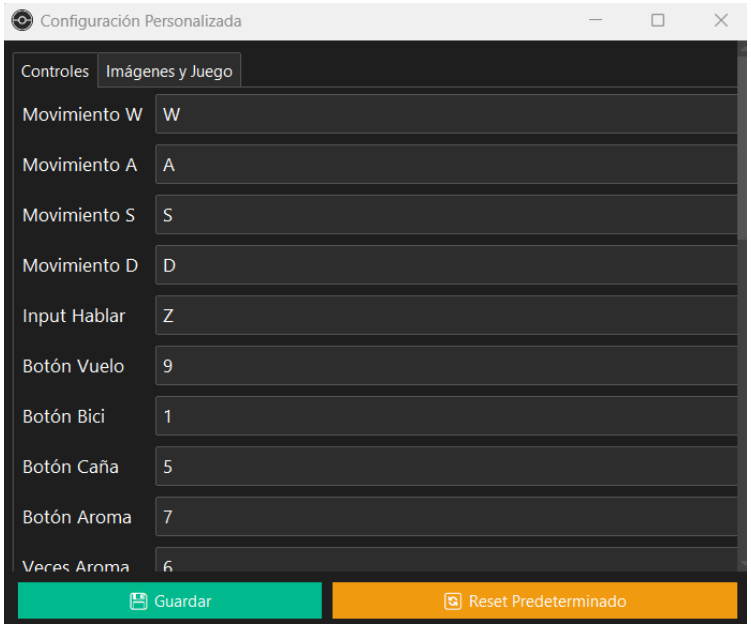
También vemos un panel debajo de los botones, esta es una consola

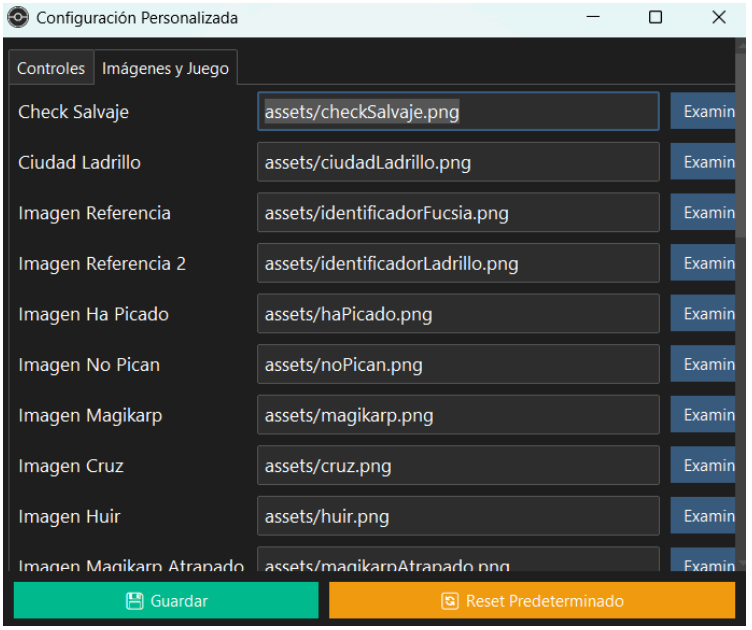
que nos va indicando qué está pasando en el código.

4.2 Pestaña de configuración

Al pulsar la **configuración** esto es lo primero que nos aparece, aquí el usuario pondrá sus controles del juego para que el programa pueda utilizarlos y de esta forma llevar a cabo el script, si el usuario usa otros **controles** a los predeterminados tendrá que modificarlos aquí.

En la pestaña de **“Imágenes y Juego”** el usuario también podrá **modificar** los assets que usa el script por si quiere usar otros diferentes porque utiliza una **versión modificada del juego**.





Si le damos a **examinar** nos abrirá el explorador de archivos y podemos seleccionar el archivo que **queramos.**

Aparte de esto también tenemos un botón de **“Guardar”** para guardar la configuración y el botón “**Reset**

Predeterminado” que permite al usuario poner todos los assets predeterminados en caso de que algo haya dejado de funcionar.

5. Back-End

5.1 Funciones principales

El código del juego se basa en **3 pilares**, los inputs por teclado, movimiento del ratón y reconocimiento por pantalla.

El input por teclado lo hacemos con keyboard con funciones como **keyboard.press y keyboard.release**. **El movimiento** de ratón lo hacemos mediante **PyAutoGui** con las funciones pyautogui.moveTo y pyautogui.click. **Y el reconocimiento** por pantalla lo hacemos con **mss** que se encarga de hacer las capturas en tiempo real, **cv2** los pasa a grises y **numpy** los compara con un umbral de acierto.

Gracias a estas funciones que nos dan las bibliotecas de **python** he podido crear **mis propias funciones** que agilizan el proceso de automatización en gran medida, entre estas tenemos las siguientes.

Detectar imagen: durante la ejecución del programa se hacen decenas de detecciones por pantalla y por tanto hemos creado una función que recibe una imagen de referencia y el

umbral que elijamos y buscará esa imagen por pantalla

```python
def detectar_imagen(imagen_ref, umbral):
    with mss.mss() as sct:
        screenshot = sct.grab(sct.monitors[1])      You, 2 weeks ago • first commit
        img_pantalla = np.array(screenshot)
        img_pantalla_gris = cv2.cvtColor(img_pantalla, cv2.COLOR_BGRA2GRAY)
        img_ref_gris = cv2.imread(imagen_ref, cv2.IMREAD_GRAYSCALE)
        if img_ref_gris is None:
            log_message(f"❌ Error: No se encontró la imagen {imagen_ref}")
            return False
        resultado = cv2.matchTemplate(img_pantalla_gris, img_ref_gris, cv2.TM_CCOEFF_NORMED)
        loc = np.where(resultado >= umbral)
        if len(loc[0]) > 0:
            log_message(f"✅ Imagen detectada: {imagen_ref}")
            return True
        else:
            return False
```

Click imagen: es muy parecida a detectar imagen solo que este se hace en bucle hasta encontrar la imagen y cuando la encuentra busca en qué parte de la pantalla está, busca su centro, y le hace click. Muy útil para decirle al programa “espera hasta que esto aparezca en pantalla y cuando lo haga le das click” así nos ahorramos time.sleeps y hacemos el código más eficiente.

```python
def click_imagen(ruta_imagen, umbral=0.8):
    with mss.mss() as sct:
        while running:
            screenshot = sct.grab(sct.monitors[1])
            pantalla = np.array(screenshot)
            pantalla = cv2.cvtColor(pantalla, cv2.COLOR_RGB2GRAY)
            plantilla = cv2.imread(ruta_imagen, cv2.IMREAD_GRAYSCALE)
            if plantilla is None:
                log_message(f"❌ Error: No se pudo cargar '{ruta_imagen}'")
                return None
            resultado = cv2.matchTemplate(pantalla, plantilla, cv2.TM_CCOEFF_NORMED)
            _, max_val, _, max_loc = cv2.minMaxLoc(resultado)
            if max_val >= umbral:
                x, y = max_loc
                x += plantilla.shape[1] // 2
                y += plantilla.shape[0] // 2
                pyautogui.moveTo(x, y)
                if not sleep_check(0.5):
                    return None
                pyautogui.click()
                log_message(f"✅ Click en imagen en posición ({x}, {y})")
                return (x, y)
            log_message("🔍 Imagen no encontrada, reintentando...")
            if not sleep_check(0.2):
                return None
        return None
```

5.2 Funciones secundarias

A partir de las **funciones principales**, tenemos las funciones secundarias que son las que son un conjunto de todas las principales. Es decir, mediante el uso de las funciones click_imagen, keyboard.press, pyautogui.moveTo… hacen **todo el movimiento** y **acciones** dentro del juego. Estas funciones las podemos dividir en **movimientos, acciones y detecciones**.

Movimientos son acciones que principalmente tienen **inputs por teclado** que lo único que hacen es colocar al personaje en una posición en concreto como posicionSafari(). Aquí aprieta el botón de la bici para acelerar el movimiento y luego ejecuta mediante inputs y sleeps todos los movimientos necesarios.

```python
# Va a la posicion del safari elegida para pescar
def posicionSafari():
    if not running:
        return
    keyboard.press_and_release(config.botonBici)
    if not sleep_check(0.2):
        return
    keyboard.press(config.MovimientoW)
    if not sleep_check(1):
        return
    keyboard.release(config.MovimientoW)
    if not sleep_check(0.2):
        return
    keyboard.press(config.MovimientoD)
    if not sleep_check(0.52):
        return
    keyboard.release(config.MovimientoD)
    if not sleep_check(0.2):
        return
    keyboard.press(config.MovimientoW)
    if not sleep_check(0.5):
        return
    keyboard.release(config.MovimientoW)
```

Acciones son funciones que hacen una **función en concreto** dentro del juego como puede ser tirar una pokeball, cerrar un popup…

```python
def tirarBall():
    while running:
        if click_imagen(config.imagen_tirarBall, 0.8):
            log_message("⚽ Tirando pelota")
            return True
        log_message("❌ No se encuentra tirar pelota")
        if not sleep_check(0.5):
            return False
    return False

# Cierra el popup después de la captura
def cerrarPopup():
    while running:
        if click_imagen(config.imagen_cruz, 0.9):
            log_message("❎ Cerrando popup…")
            return True
        log_message("❌ No se encuentra popup…")
        if not sleep_check(0.5):
            return False
    return False
```

Por último tenemos **detecciones**, estas funciones se utilizan para **ver qué está sucediendo en el juego**, por ejemplo, cuando le tiramos una piedra a un pokemon este se puede enfadar o se puede escapar, entonces creamos una función que detecta si se enfada o se escapa

```python
def enfado():
    for _ in range(15):
        if not running:
            return False
        if detectar_imagen(config.imagen_enfadado, 0.8):
            log_message("😡 Se encontró enfado")
            return True
        log_message("😕 No se encuentra enfado")
        if not sleep_check(0.2):
            return False
    return False
```

5.3 Flujo de Ejecución

Ahora llega la parte **más importante**, el flujo de ejecución, aquí es donde se juntan todas las funciones y mediante condiciones manejamos **todos los posibles eventos del juego.** Una de estas seria pescar:

```
def pescar():
    while running:
        if not sleep_check(2):
            return
        keyboard.press_and_release(config.botonCaña)
        if not running:
            return
        if detectarPicado():
            log_message("🎣 Pokémon ha picado, lanzando piedra...")
            if not sleep_check(1):
                return
            keyboard.press_and_release(config.inputHablar)
            if not sleep_check(0.2):
                return
            if tirarPiedra():
                log_message("✅ Piedra lanzada, intentando tirar ball")
                if not sleep_check(0.2):
                    return
                if tirarBall():
                    log_message("✅ Ball lanzada, cerrando popup")
                    if cerrarPopup():
                        log_message("🔄 Revisando Ding Dong...")
                        if dingdong():
                            log_message("🔔 Ding Dong")
                            continue
                        else:
                            log_message("🔄 Ding dong no encontrado. Reiniciando ciclo...")
                            continue
                    else:
                        log_message("❌ Error cerrando popup")
                else:
                    log_message("❌ Error al tirar Ball")
            else:
                log_message("🔄 Revisando Ding Dong...")
                if dingdong():
                    log_message("🔔 Ding Dong")
```

```
            else:
                log_message("🔄 Ding dong no encontrado. Reiniciando ciclo...")
                continue
        else:
            log_message("❌ No ha picado, reiniciando proceso...")
            if not sleep_check(0.2):
                return
            keyboard.press_and_release(config.inputHablar)
            if not sleep_check(0.2):
                return
            continue
```

Aquí podemos ver **todas las funciones** y condiciones que influyen en la acción de pescar dentro del farmeo de Magikarps, todo con sus respectivos logs para saber por dónde vamos y cada condición llamando a otra función para hacer este bucle infinito.

6. Problemas y Soluciones

Durante el proceso de creación de la aplicación se han resuelto **varios errores de código** donde podemos encontrar los siguientes:

6.1 Solapamiento de movimientos

Al principio al pedirle al código **presionar ciertos botones** este lo hacía tan rápido que los empezaba a **mezclar**, por tanto decidí poner un pequeño **tiempo de espera** de algunos milisegundos para que no suceda esto, si aumentara el tiempo de ejecución del programa pero también aumenta la fiabilidad de este.

6.2 Detección por pantalla

Estuve pensando en que usar para las **detecciones por pantalla**, si **detectar texto** o directamente **detectar imágenes,** tras varias pruebas vi que la detección por texto no era muy de fiar a que se inventaba palabras o las leía mal y la detección por imagen con un pequeño margen de error es mucho más fiable.

6.3 Manejo de errores

Al ser un código **totalmente automático** y en bucle hay que manejar **todas las posibilidades** que pueden ocurrir. Decidimos crear varias funciones que hacen ciertas acciones y crear un flujo principal que mediante condicionales if/else en un while va llamando a los funciones que llevan a cabo la automatización.

7. Futuras mejoras

7.1 Añadir nuevos scripts

Hay **muchas más acciones** que se pueden automatizar en el juego cada uno con sus diferentes dificultades y manejo de errores por tanto con el tiempo se irían añadiendo más scripts para englobar el máximo de acciones posibles.

7.2 Mejorar la eficiencia

Los scripts con el tiempo también se pueden **optimizar** mejorando los tiempos y puliendolos para que se ejecuten lo más rápido posible sin dar lugar a fallos o añadiendo lectura de memoria del juego para acciones como el movimiento lo que reduciría en gran medida posibles fallos.

8. Licencia

[Licencia de Creative Commons CC BY SA]
https://creativecommons.org/licenses/by/4.0/deed.es