DESARROLLO DE VIDEOJUEGO CON SISTEMA DE ESTADISTICAS EN LA NUBE

Trabajo realizado por Sebastián Juan Melgar Marín

IES FUENGIROLA Nº1

INTRODUCCIÓN

En el contexto del desarrollo de un videojuego, nos enfrentamos a dos áreas fundamentales que requieren atención:

- **Mejora de la experiencia del jugador:** Implementación de características básicas faltantes.
- **Integración de un sistema de estadísticas:** Enriquecer la interacción del jugador con el juego.

Principales problemas anteriores al proyecto:

- Carencia de un sistema de estadísticas
- Falta de un menú de login
- Inexistencia de un final apropiado con créditos.
- Interfaz visualmente poco atractiva.
- Carencia de transiciones de sonido y sucesos coherentes.
- Sistema de control mejorable

¿Que aporta un sistema de estadísticas?

- Recopilación de datos sobre el desempeño del jugador (daño recibido, infligido, tiempo de juego, porcentaje de aciertos).
- Información valiosa para el jugador y desarrolladores.
- Mejora de la experiencia del jugador y ajuste del juego en futuras iteraciones.

ANTECEDENTES

El desarrollo de videojuegos es un campo en constante evolución. La atención a los detalles y la satisfacción del jugador son cruciales para el éxito de un título.

- **Deficiencias anteriores:** La falta de ciertas funcionalidades básicas y la ausencia de un sistema de estadísticas pueden afectar negativamente la experiencia del jugador y limitar el potencial del juego.
- **Enfoque del proyecto:** Abordar estas deficiencias y mejorar la experiencia del usuario.

Anteriormente el juego solo disponía de estas dos escenas

OBJETIVOS DEL PROYECTO

Objetivo principal:

- Mejorar la experiencia del jugador mediante la implementación de características faltantes y la integración de un sistema de estadísticas en el videojuego.

Objetivos específicos:

- Ampliar las funcionalidades del juego (menú de login, final con créditos, transiciones coherentes, interfaz vistosa y transiciones de sonido y añadir varios mapas).
- Implementar un sistema de estadísticas que recoja datos sobre el desempeño del jugador (daño recibido, infligido, enemigos destruidos, tiempo de partida, porcentaje de aciertos) y poder consultarlos via web.

REQUISITOS FUNCIONALES

- Implementación de un sistema de estadísticas al final de cada mapa para mostrar datos relevantes sobre el desempeño del jugador.
- Sistema de consulta de estadísticas de partidas vía web:
  - Registro.
  - Login.
  - Registro de partidas.
  - Sistema de amigos.
- Creación de un final con créditos.
- Diseño de una interfaz de usuario atractiva y visualmente agradable.
- Implementación de un menú de opciones para personalizar la configuración del juego.
- Desarrollo de varios mapas con diferentes niveles de dificultad y desafíos.

REQUISITOS NO FUNCIONALES

- Eficiencia en el rendimiento del juego para garantizar una experiencia fluida.
- Seguridad en el almacenamiento y gestión de datos del jugador.

DESCRIPCIÓN DEL PROYECTO

UNITY

Descripción General:

- El videojuego es un shooter espacial desarrollado en Unity.
- Permite al jugador controlar una nave espacial, combatir enemigos, y progresar a través de diferentes mapas y niveles.

Público objetivo y motivación detrás del proyecto:

- Dirigido a jugadores aficionados a los shooters espaciales y juegos de arcade.
- Motivación: Crear un juego envolvente y desafiante que combine jugabilidad clásica con características modernas.

FLASK

Descripción General:

- Una página web con diseño acorde a la temática del videojuego
- Permite al usuario ver sus estadísticas y las de otros jugadores

TECNOLOGÍAS UTILIZADAS

- **Unity:** Usado para el desarrollo del videojuego debido a su robustez y facilidad para crear gráficos y mecánicas de juego.
- **Flask:** Elegido como backend por su simplicidad y eficiencia en manejar peticiones y gestionar la lógica del servidor, junto a Jinja2 como motor de plantillas.
- **MySQL:** Base de datos seleccionada por su fiabilidad y compatibilidad con PythonAnywhere.
- **SQLAlchemy:** ORM utilizado para facilitar la interacción con la base de datos MySQL.

CARACTERÍSTICAS DEL JUEGO

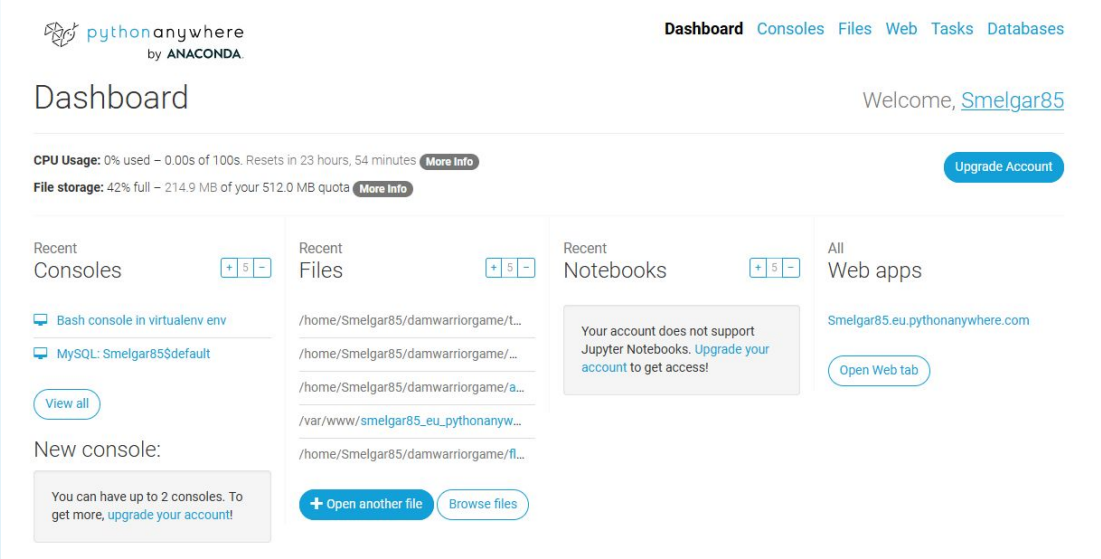
Características Principales:

- **Controles del juego:**
  - Compatible con teclado y gamepad. Uso de mouse en la pantalla de login.
- **Descripción de la jugabilidad:**
  - El jugador navega una nave espacial, enfrentándose a oleadas de enemigos y jefes finales a través de distintos mapas.

ARQUITECTURA DEL BACKEND

Estructura del Backend de la web de estadísticas:

- **Uso de Flask:** Framework ligero y eficiente para manejar las peticiones y la lógica del servidor.
- **Arquitectura monolítica:** Simplifica el desarrollo y el despliegue del proyecto.
- **Base de datos MySQL:** Almacenada en PythonAnywhere, utilizada para guardar información de usuarios y estadísticas del juego.
- **Sistema de autenticación:** Utiliza hashes de contraseña para asegurar las cuentas de usuario.



API DEL BACKEND

Registro de Usuarios:
Endpoint: /register
Método: POST
Descripción: Permite a los nuevos usuarios registrarse en la plataforma.
Parámetros: nombre_usuario, email, contrasena
Respuesta: Mensaje de éxito o error.

Inicio de Sesión:
Endpoint: /login
Método: POST
Descripción: Autentica a los usuarios y establece una sesión.
Parámetros: nombre_usuario, contrasena
Respuesta: Redirige al dashboard en caso de éxito, mensaje de error en caso de fallo.

Guardar Estadísticas:
Endpoint: /guardar_estadisticas
Método: POST
Descripción: Recibe las estadísticas del juego desde Unity y las almacena en la base de datos.
Parámetros: username, password, stats (JSON con las estadísticas de la partida)
Respuesta: Mensaje de éxito o error.

Dashboard:
Endpoint: /dashboard
Método: GET
Descripción: Muestra el panel principal del usuario con las mejores partidas de todos los tiempos.
Parámetros: Ninguno.
Respuesta: Renderiza la plantilla dashboard.html con la información del usuario.

Perfil de Usuario:
Endpoint: /perfil
Método: GET, POST
Descripción: Muestra y permite editar el perfil del usuario.
Parámetros: nombre_usuario, email (para actualizar), accion (para determinar la acción a realizar)
Respuesta: Renderiza la plantilla perfil.html con la información del usuario.

EJEMPLO PRÁCTICO

1. Preparación de los Datos en Unity.

En Unity, se deben recolectar las estadísticas del juego y convertirlas en un formato que se pueda enviar fácilmente a través de una solicitud HTTP, en nuestro caso, hemos escogido JSON.

```
public void SaveGameStatistics()
{
    // Creamos una instancia de GameStatistics con los datos del juego
    GameStatistics stats = new GameStatistics(
        DateTime.Now,
        mapName,
        score,
        totalShots > 0 ? (float)shotsHit / totalShots : 0,
        Time.time - startTime,
        damageDealt,
        damageTaken
    );

    // Se convierten las estadísticas a JSON
    string json = JsonUtility.ToJson(stats);

    //Guardamos las estadísticas localmente usando PlayerPrefs
    PlayerPrefs.SetString("ultimaPartida", json);
    PlayerPrefs.Save();

    // Se envían las estadísticas al servidor siempre y cuando el usuario no sea "guest"
    string username = PlayerPrefs.GetString("username", "guest");
    if (username != "guest")
    {
        StartCoroutine(SendStatisticsToServer(json, username, PlayerPrefs.GetString("passwor
    }
}
```

2. Enviar los Datos desde Unity a Flask.

Se utiliza una corrutina en Unity para enviar los datos a través de una solicitud HTTP POST. Esto permite manejar la solicitud de forma asíncrona sin que el juego se bloquee.

```
private IEnumerator SendStatisticsToServer(string json, string username, string password)
{
    // Creamos un formulario para la solicitud POST que enviar al servidor
    WWWForm form = new WWWForm();
    form.AddField("username", username);
    form.AddField("password", password);
    form.AddField("stats", json);

    // Aquí enviamos la solicitud POST al servidor Flask
    using (UnityWebRequest www = UnityWebRequest.Post("http://smelgar85.eu.pythonanywhere.co
    {
        yield return www.SendWebRequest();

        // Comprobamos la respuesta del servidor (solo para debug)
        if (www.result != UnityWebRequest.Result.Success)
        {
            Debug.LogError("Error al enviar las estadísticas: " + www.error);
        }
        else
        {
            Debug.Log("Estadísticas enviadas correctamente");
        }
    }
}
```

Código en Flask (app.py):

3. Recepción de los Datos en Flask.

En el servidor Flask, se configura un endpoint para recibir la solicitud POST con las estadísticas del juego.

```python
@app.route('/guardar_estadisticas', methods=['POST'])
def guardar_estadisticas():
    username = request.form['username']
    password = request.form['password']
    stats_json = request.form['stats']

    # Buscamos el usuario en la base de datos
    user = Usuario.query.filter_by(nombre_usuario=username).first()
    if not user or not check_password_hash(user.contrasena, password):
        return jsonify({'status': 'error', 'message': 'Usuario o contraseña incorrectos'}),

    # Convertimos el JSON de estadísticas en un diccionario
    stats = json.loads(stats_json)

    # Se crea una nueva instancia de Partida con los datos recibidos
    partida = Partida(
        usuario_id=user.id,
        mapa_id=1,  # Esto (y lo que conlleva) habrá que modificarlo si se añaden mapas adic
        fecha=datetime.datetime.strptime(stats['fecha'], '%Y-%m-%d %H:%M:%S'),
        puntuacion_destruccion=stats['puntuacion'],
        estadisticas_precision=stats['precision'] * 100,
        tiempo_completado=stats['tiempoCompletado'],
        dano_recibido=stats['danoRecibido'],
        dano_infligido=stats['danoCausado']
    )

    # Se guarda la partida en la base de datos
    db.session.add(partida)
    db.session.commit()

    return jsonify({'status': 'success', 'message': 'Estadísticas guardadas correctamente'})
```

DESAFÍOS ENFRENTADOS

- **Unity.** Es un sistema complejo con cientos de configuraciones posibles, que no siempre dan un resultado esperado si no se conoce bien cómo funciona el editor.
- **Problemas con la configuración de controles:** Anteriormente se usaba la librería Input Manager, que cumplía los requisitos de manejo con teclado. Ésta librería fue sustituida por Input System, más versátil. El problema es que gran parte del código estaba pensado para usar Input Manager. Se intentó migrar toda la lógica de control a Input System pero interfería con el uso del ratón. Finalmente se usó un sistema mixto
- **Almacenado de estadísticas dentro del juego:** En un principio desconocía cómo almacenar las estadísticas dentro de Unity, hasta que descubrí la clase PlayerPrefs que permite almacenar texto. Esto se usó para almacenar las estadísticas en forma de json, que se envían al final de cada partida al servidor Flask mediante la API.
- **Mantenimiento de sesiones de usuario:** Asegurar sesiones estables y seguras para los usuarios. En un principio me encontré con el problema de que si se dejaba la página en reposo, la siguiente vez que se accedía a ella daba error, pues la conexión con la base de datos se cerraba pasado un tiempo. Añadiendo esta configuración a SQLAlchemy se evitaba el error:

```
'pool_pre_ping': True,  # Verifica la conexión antes de usarla
'pool_recycle': 280  # Recicla las conexiones para evitar desconexiones por timeout
```

CONCLUSIÓN

Creo que con el presente proyecto se ha cumplido con los objetivos principales del mismo, que eran desarrollar un sistema de estadísticas que fuese consultable vía web, si bien es cierto que muchos objetivos secundarios no se han visto implementados por falta de tiempo.

Opino que el resultado final es vistoso a la vista, sencillo y práctico.

Por supuesto, existe muchísmo márgen de mejora.

- **Flask:** Migración a estructura modular del código, implementar imágenes de perfil, confirmación de cuenta por email, cambio de contraseña, solicitudes de amistad.

- **Unity:** Más mapas y enemigos, pantalla de selección de naves, sistema narrativo con viñetas, mejoras en transiciones entre escenas, opciones de sonido y dificultad, menú de pausa, sistema "Continue" al ser derrotado.

FIN