BACKEND. Scripts de Unity

Scripts de la nave principal

VidaNave.cs

Descripción: Maneja la salud y las vidas de la nave del jugador, así como la activación de la invulnerabilidad y los efectos visuales.

Métodos:

- `Start()`: Inicializa referencias y verifica componentes.
- `OnCollisionEnter2D(Collision2D collision)`: Maneja las colisiones y aplica daño si no es invulnerable.
- `AplicarDanio(int cantidad)`: Aplica daño a la nave y activa la invulnerabilidad temporal.
- `IEnumerator ActivarInvulnerabilidad()`: Activa la invulnerabilidad temporalmente.
- `PerderSalud(int cantidad)`: Reduce la salud de la nave y verifica si debe perder una vida.
- `ActualizarBarraDeVida()`: Actualiza la barra de vida según la salud actual.
- `PerderVida()`: Reduce las vidas de la nave y maneja la lógica de reinicio o pérdida del juego.
- `IEnumerator RespawnAndBlink()`: Maneja el respawn de la nave y la animación de parpadeo de invulnerabilidad.
- `SpawnExplosion()`: Genera una explosión en la posición de la nave.
- `SetFieldAlpha(float alpha)`: Establece la transparencia del campo de fuerza.
- `IEnumerator FadeFieldIn(float duration)`: Realiza un fade in en el campo de fuerza.
- `IEnumerator FadeFieldOut(float duration)`: Realiza un fade out en el campo de fuerza.

MovimientoNave.cs

Descripción: Gestiona el movimiento de una nave espacial, incluyendo la rotación, desplazamiento y efectos visuales de los motores.

Métodos:

- `Start()`: Inicializa referencias.
- `Update()`: Calcula la nueva posición y rotación de la nave basada en la entrada de movimiento.
- `OnMove(InputAction.CallbackContext context)`: Método manejado por el Input System para leer la entrada de movimiento.
- `OnCollisionEnter2D(Collision2D collision)`: Método llamado cuando la nave colisiona con otro objeto.
- `ResetearMovimiento()`: Resetea el movimiento y rotación de la nave.

Bullet.cs

Descripción: Gestiona el comportamiento de una bala y el daño que inflige a diferentes tipos de enemigos, además de la puntuación.

Métodos:

- `Start()`: Inicializa referencias y destruye la bala después de su vida útil.
- `OnCollisionEnter2D(Collision2D collision)`: Método llamado cuando la bala colisiona con otro objeto.
- `DamageRock(GameObject rock)`: Aplica daño a una roca.
- `DamageEnemy(GameObject enemy)`: Aplica daño a un enemigo.
- `DamageBoss(GameObject boss)`: Aplica daño a un jefe.
- `PlayHitSound()`: Reproduce el sonido de impacto si está configurado.

BigBullet.cs

Descripción: Gestiona el comportamiento de una bala grande, incluyendo su vida útil, daño, y las interacciones con diferentes tipos de objetos.

Métodos:

- `Start()`: Inicializa referencias y destruye la bala después de su vida útil.
- `OnTriggerEnter2D(Collider2D collision)`: Método llamado cuando la bala colisiona con otro objeto.
- `DamageRock(GameObject rock)`: Aplica daño a una roca y reproduce el sonido de impacto.
- `DamageEnemy(GameObject enemy)`: Aplica daño a un enemigo y reproduce el sonido de impacto.
- `DamageBoss(GameObject boss)`: Aplica daño a un jefe y reproduce el sonido de impacto.
- `PlayHitSound()`: Reproduce el sonido de impacto si está configurado.

LaserShoot.cs

Descripción: Gestiona la mecánica de disparo de un láser y disparos especiales, incluyendo el manejo de puntuaciones y sonidos.

Métodos:

- `Start()`: Inicializa referencias.
- `OnShoot(InputAction.CallbackContext context)`: Método para el disparo normal asociado con una acción de entrada.
- `OnSpecialShoot(InputAction.CallbackContext context)`: Método para el disparo especial asociado con una acción de entrada.
- `Shoot()`: Maneja el disparo normal.
- `ShootSpecial()`: Maneja el disparo especial.
- `ResetIsShooting()`: Método para resetear la bandera de disparo.

Scripts de las naves enemigas

MovimientoEnemigoZigZag.cs

Descripción: Controla el movimiento en zigzag de los enemigos.

Métodos:

- `Start()`: Inicializa la posición y parámetros de movimiento vertical.
- `Update()`: Controla el movimiento y la rotación del enemigo.
- `Move()`: Mueve el enemigo lateralmente y en zigzag vertical.

- `Rotate()`: Ajusta la rotación del enemigo según su movimiento vertical.

SpaceShipSpawner.cs

Descripción: Se utiliza para generar naves espaciales enemigas a intervalos aleatorios.

Métodos:

- `Start()`: Método inicial.
- `StartSpawning()`: Inicia la generación de naves espaciales.
- `StopSpawning()`: Detiene la generación de naves espaciales.
- `SpawnSpaceShipWithRandomInterval()`: Genera una nave solo si el spawner está activo.
- `InstantiateSpaceShip()`: Selecciona un prefab de nave aleatorio y la instancia en una posición inicial.
- `PlaySpawnSound()`: Reproduce el sonido de spawn si está configurado.

BulletEnemigo.cs

Descripción: Controla el comportamiento de las balas enemigas en el juego.

Métodos:

- `Awake()`: Obtiene la referencia al componente AudioSource.
- `OnEnable()`: Programa el retorno de la bala al pool después de su vida útil.
- `ReturnToPool()`: Devuelve la bala al pool.
- `OnCollisionEnter2D(Collision2D collision)`: Aplica daño a la nave del jugador si colisiona con ella.
- `PlayHitSound()`: Reproduce el sonido de impacto si está configurado.

EnemyHealth.cs

Descripción: Gestiona la vida y las colisiones de una nave enemiga, incluyendo sus efectos visuales y sonoros.

Métodos:

- `Start()`: Inicializa referencias y verifica componentes.
- `TakeDamage(int damageAmount)`: Aplica daño al enemigo y maneja su muerte si la salud llega a cero.
- `Die()`: Maneja la muerte del enemigo.
- `SetFieldAlpha(float alpha)`: Establece la transparencia del campo de fuerza del enemigo.
- `IEnumerator FadeFieldIn(float duration)`: Realiza un fade in en el campo de fuerza.
- `IEnumerator FadeFieldOut(float duration)`: Realiza un fade out en el campo de fuerza.

LaserShootEnemy.cs

Descripción: Maneja el disparo de balas enemigas desde la nave enemiga.

Métodos:

- `Start()`: Asigna la fuente de audio para el sonido de disparo.
- `Update()`: Incrementa el temporizador de disparo y dispara si es necesario.
- `Shoot()`: Reproduce el sonido de disparo y dispara una bala enemiga.

Scripts del Boss

BossSpawner.cs

Descripción: Maneja la generación del jefe en el juego.

Métodos:

- `StartSpawning()`: Genera el jefe si no hay uno actualmente.
- `StopSpawning()`: Detiene la generación del jefe.
- `SpawnBoss()`: Genera el jefe en la posición del spawner.

BulletBoss.cs

Descripción: Controla el comportamiento de las balas disparadas por el jefe.

Métodos:

- `Start()`: Inicializa el tiempo de vida de la bala.
- `OnCollisionEnter2D(Collision2D collision)`: Aplica daño a la nave del jugador si colisiona con ella.
- `PlayHitSound()`: Reproduce el sonido de impacto si está configurado.
- `SpawnExplosion()`: Genera la explosión en la posición actual.

LaserShootBoss.cs

Descripción: Maneja el disparo de balas desde el jefe, utilizando múltiples puntos de disparo.

Métodos:

- `Start()`: Inicializa referencias.
- `Update()`: Incrementa el temporizador de disparo y dispara si es necesario.
- `Shoot()`: Dispara una bala desde un punto de disparo aleatorio.

BossMovement.cs

Descripción: Controla el movimiento del jefe, incluyendo avance inicial y oscilación.

Métodos:

- `Start()`: Guarda la posición inicial del jefe.
- `Update()`: Controla el movimiento del jefe, alternando entre avanzar y oscilar.
- `Advance()`: Mueve el jefe hacia la izquierda hasta alcanzar la distancia de avance.
- `Sway()`: Oscilación horizontal y vertical.

LaserBIGShootBoss.cs

Descripción: Maneja el disparo de balas grandes desde el jefe.

Métodos:

- `Start()`: Inicializa referencias.

- `Update()`: Incrementa el temporizador de disparo y dispara si es necesario.
- `Shoot()`: Dispara una bala grande desde un punto de disparo específico.

BossHealth.cs

Descripción: Maneja la vida del jefe y su muerte.

Métodos:

- `Start()`: Inicializa referencias y verifica componentes.
- `TakeDamage(int damageAmount)`: Aplica daño al jefe y maneja su muerte si la salud llega a cero.
- `Die()`: Maneja la muerte del jefe.

Scripts de las rocas

RockMovement.cs

Descripción: Controla el movimiento y la destrucción de las rocas en el juego.

Métodos:

- `Update()`: Mueve la roca y la destruye si se mueve fuera de la pantalla.
- `Move()`: Mueve la roca hacia la izquierda y la rota alrededor del eje Y.

RockSpawner.cs

Descripción: Se utiliza para generar rocas en el juego a intervalos aleatorios.

Métodos:

- `Start()`: Método inicial.
- `StartSpawning()`: Inicia la generación de rocas.
- `StopSpawning()`: Detiene la generación de rocas.
- `SpawnRock()`: Genera una roca en una posición aleatoria.
- `SpawnRocksAtPosition(Vector3 position, int numberOfRocks)`: Genera múltiples rocas en una posición específica.
- `InstantiateRock(Vector3 position)`: Instancia una roca con una escala aleatoria y reproduce un sonido.
- `PlaySpawnSound()`: Reproduce el sonido de spawn si está configurado.

RockHealth.cs

Descripción: Maneja la salud de las rocas y su destrucción, incluyendo la generación de rocas más pequeñas.

Métodos:

- `Start()`: Inicializa la salud de la roca y obtiene el ScoreManager.
- `OnBecameVisible()`: Marca la roca como visible cuando entra en pantalla.
- `OnBecameInvisible()`: Destruye la roca si se vuelve invisible después de haber sido visible.
- `TakeDamage(int damage)`: Aplica daño a la roca y verifica si debe destruirse.

Scripts relacionados con estadísticas

GameStatistics.cs

Descripción: Almacena las estadísticas de una partida, como la fecha, el nombre del mapa, la puntuación, la precisión, el tiempo completado, el daño causado y el daño recibido.

Métodos:

- `GameStatistics(DateTime fecha, string nombreMapa, int puntuacion, float precision, float tiempoCompletado, int danoCausado, int danoRecibido)`: Inicializa los campos con los valores proporcionados.

ScoreManager.cs

Descripción: Gestiona las estadísticas del juego, como la puntuación, los disparos realizados y acertados, el daño infligido y recibido. También permite guardar estas estadísticas y enviarlas a un servidor.

Métodos:

- `Start()`: Inicializa y reinicia las estadísticas de puntuación.
- `ResetScore()`: Reinicia las estadísticas de puntuación.
- `OnEnable()`: Actualiza el texto de la puntuación cuando el objeto es habilitado.
- `AddScore(int amount)`: Añade puntos a la puntuación actual.
- `UpdateScoreText()`: Actualiza el texto con la puntuación actual.
- `RegisterShot()`: Registra un disparo realizado.
- `RegisterHit()`: Registra un disparo acertado.
- `RegisterDamageDealt(int amount)`: Registra el daño infligido.
- `RegisterDamageTaken(int amount)`: Registra el daño recibido.
- `SaveGameStatistics()`: Guarda las estadísticas del juego localmente y las envía al servidor.
- `SendStatisticsToServer(string json, string username, string password)`: Corrutina para enviar las estadísticas al servidor.
- `GetScore()`: Obtiene la puntuación actual.

ResumenPartida.cs

Descripción: Muestra el resumen de la partida, incluyendo estadísticas como la puntuación, precisión, tiempo completado, daño causado y recibido. También maneja la transición a la escena de créditos.

Métodos:

- `Start()`: Obtiene y muestra las estadísticas de la última partida desde PlayerPrefs.
- `OnNextButtonClicked()`: Carga la escena de créditos.

Scripts de menus

LoginManager.cs

Descripción: Gestiona la funcionalidad de inicio de sesión, registro y acceso como invitado en el juego, incluyendo la conexión a un servidor para autenticar y registrar usuarios.

Métodos:

- `Start()`: Inicializa la imagen de fade y configura los campos de entrada.
- `Update()`: Detecta la tecla Enter para iniciar sesión.
- `OnLoginButtonClicked()`: Reproduce el sonido de login e inicia el proceso de login.
- `OnGuestButtonClicked()`: Establece el usuario y contraseña como "guest" y carga la escena siguiente.
- `Login()`: Envía los datos de login al servidor y maneja la respuesta.
- `OnRegisterButtonClicked()`: Inicia el proceso de registro.
- `Register()`: Envía los datos de registro al servidor y maneja la respuesta.
- `FadeOutAndChangeScene()`: Inicia el efecto de fade out y cambia de escena.
- `OnEnterKeyPressed()`: Maneja la tecla Enter para iniciar sesión.

MainMenu.cs

Descripción: Controla el menú principal del juego, incluyendo la navegación, inicio de sesión/cierre de sesión y el inicio o salida del juego.

Métodos:

- `Awake()`: Inicializa los controles cuando se despierta este componente.
- `Start()`: Configura el texto inicial del menú y la navegación entre botones.
- `SetupNavigation()`: Configura la navegación entre los botones usando el teclado y el gamepad.
- `Navigate(InputAction.CallbackContext context)`: Permite navegar entre los elementos con la entrada del controlador.
- `Submit(InputAction.CallbackContext context)`: Permite seleccionar un elemento utilizando la entrada del controlador.
- `OnPointerClick(PointerEventData eventData)`: Métodos de manejo del cursor para clics en pantalla.
- `PlayGame()`: Inicia el juego cargando la escena del mapa.
- `QuitGame()`: Cierra la aplicación del juego.
- `Logout()`: Cierra sesión y carga la escena de inicio de sesión.

CustomCursor.cs

Descripción: Permite personalizar el cursor del juego y ocultarlo tras un período de inactividad.

Métodos:

- `Start()`: Configura el cursor personalizado.
- `Update()`: Detecta si el mouse se mueve y actualiza el tiempo de la última actividad.
- `SetCursor()`: Establece la textura del cursor.
- `ShowCursor()`: Muestra el cursor.
- `HideCursor()`: Oculta el cursor.

FadeInLogo.cs

Descripción: Hace un efecto de fade-in en el logo al inicio del juego.

Métodos:

- `Start()`: Obtiene el componente `SpriteRenderer` del objeto y lo hace transparente al inicio.
- `FadeIn()`: Realiza un fade in gradual en el logo.

OpenURL.cs

Descripción: Abre una URL de registro en el navegador.

Métodos:

- `OpenRegistrationPage()`: Abre la URL en el navegador.

PlaySoundOnType.cs

Descripción: Reproduce un sonido cuando el usuario escribe en un campo de entrada de texto.

Métodos:

- `Start()`: Añade listener para cada entrada de texto en los campos `TMP_InputField`.
- `PlaySound(string input)`: Reproduce el sonido al escribir.

TabNavigation.cs

Descripción: Permite la navegación entre campos de entrada utilizando la tecla Tab.

Métodos:

- `Update()`: Detecta si se presiona la tecla Tab.
- `SelectNextInputField()`: Encuentra el campo de entrada que actualmente tiene el foco y selecciona el siguiente.

Scripts de gameplay

PowerUpMovement.cs (no implementado finalmente)

Descripción: Gestiona el movimiento de los distintos powerups en el juego.

Métodos:

- `Start()`: Inicializa la salud y rota el objeto de forma aleatoria al inicio.
- `Update()`: Mueve el objeto hacia la izquierda, realiza un movimiento en zigzag y lo rota.
- `TakeDamage(int damage)`: Reduce la salud del objeto cuando recibe daño.
- `PlayBreakSound()`: Reproduce un sonido cuando el objeto es destruido.

StageManager.cs

Descripción: Maneja las distintas etapas de un mapa, incluyendo el sistema de pausa y el final de la escena cuando el jefe muere.

Métodos:

- `Start()`: Inicializa y reinicia el sistema de puntuación y spawneo.
- `Update()`: Verifica si se presiona la tecla de pausa y maneja el temporizador de las etapas del juego.
- `PauseGame()`: Pausa el juego y la música, y activa el canvas de pausa.
- `ResumeGame()`: Reanuda el juego y la música, y desactiva el canvas de pausa.
- `StartRestPeriod()`: Inicia un período de descanso entre etapas.

- `EndRestPeriod()`: Finaliza el período de descanso y avanza a la siguiente etapa.
- `AdvanceStage()`: Avanza a la siguiente etapa del juego.
- `UpdateStageSettings()`: Configura la etapa actual del juego.
- `FlyingFortressDestroyed()`: Maneja la destrucción del jefe y guarda las estadísticas del juego.
- `LoadSummaryScene()`: Carga la escena del resumen de la partida.
- `StopAllSpawners()`: Detiene todos los spawners.
- `StopSpawner(MonoBehaviour spawner)`: Detiene un spawner específico.
- `StartSpawner(MonoBehaviour spawner)`: Inicia un spawner específico.
- `ResetStageManager()`: Reinicia el StageManager y los spawners.

BulletPool.cs

Descripción: Gestiona un pool de balas para optimizar la creación y destrucción de balas en el juego.

Métodos:

- `Awake()`: Inicializa la instancia singleton.
- `GetPlayerBullet()`: Obtiene una bala de jugador del pool o crea una nueva si el pool está vacío.
- `ReturnPlayerBullet(GameObject bullet)`: Devuelve una bala de jugador al pool.
- `GetEnemyBullet()`: Obtiene una bala de enemigo del pool o crea una nueva si el pool está vacío.
- `ReturnEnemyBullet(GameObject bullet)`: Devuelve una bala de enemigo al pool.
- `GetSpecialBullet()`: Obtiene una bala especial del pool o crea una nueva si el pool está vacío.
- `ReturnSpecialBullet(GameObject bullet)`: Devuelve una bala especial al pool.

GamepadButtonHandler.cs

Descripción: Maneja el paso a otra escena de manera sencilla cuando se presiona un botón en el gamepad.

Métodos:

- `Update()`: Verifica si se presiona el botón de submit del gamepad y simula un clic en el botón de la interfaz de usuario.

GameManager.cs

Descripción: Gestiona la variable `fullPower`, que indica si el disparo secundario está disponible.

Variables:

- `public static bool fullPower`: Variable estática accesible globalmente.

MovimientoFondo.cs

Descripción: Desplaza una imagen de fondo para crear un efecto de movimiento.

Métodos:

- `Update()`: Calcula la nueva posición de la imagen de fondo y la aplica.

PlayerInputHandler.cs

Descripción: Maneja la entrada del jugador para controlar una nave.

Métodos:

- `Awake()`: Inicializa el manejo de controles y obtiene las referencias a los componentes relacionados.
- `OnEnable()`: Habilita los controles cuando el objeto está activo.
- `OnDisable()`: Deshabilita los controles cuando el objeto ya no está activo.

SceneFader.cs

Descripción: Realiza efectos de fade-in y fade-out entre transiciones de escenas.

Métodos:

- `Start()`: Comienza con un fade in.
- `FadeToScene(string sceneName)`: Inicia el fade out y cambia de escena.
- `FadeIn()`: Realiza un fade in gradual.
- `FadeOut(string sceneName)`: Realiza un fade out gradual y luego carga la nueva escena.

ScrollCredits.cs

Descripción: Desplaza los créditos en la pantalla.

Métodos:

- `Start()`: Obtiene el componente `RectTransform` del objeto.
- `Update()`: Desplaza los créditos hacia arriba a una velocidad constante.

AudioFader.cs

Descripción: Hace un fade out de una fuente de audio.

Métodos:

- `FadeOutAudio()`: Inicia la corrutina para hacer fade out del audio.
- `FadeOutCoroutine()`: Reduce gradualmente el volumen de la fuente de audio hasta silenciarla.

BACKEND. Aplicación flask

app.py

1. **jugar:**
   - Verifica si el usuario ha iniciado sesión y muestra la página para jugar.
2. **home:**
   - Redirige al dashboard si el usuario ha iniciado sesión, sino redirige a la página de login.
3. **perfil_usuario:**
   - Muestra el perfil del usuario seleccionado y sus partidas.
4. **register:**
   - Gestiona el registro de nuevos usuarios, verificando la disponibilidad del nombre de usuario y correo electrónico, y almacenando la nueva cuenta.
5. **login:**
   - Gestiona el inicio de sesión de los usuarios, verificando las credenciales.

6. **dashboard:**
    - Muestra el panel principal del usuario, incluyendo sus mejores partidas.
7. **partidas:**
    - Muestra todas las partidas del usuario.
8. **amigos:**
    - Gestiona la lista de amigos del usuario, permitiendo agregar y eliminar amigos.
9. **logout:**
    - Cierra la sesión del usuario.
10. **perfil:**
    - Permite al usuario actualizar su perfil, borrar partidas o eliminar su cuenta.
11. **guardar_estadisticas:**
    - Guarda las estadísticas de una partida enviada por una aplicación externa, después de verificar las credenciales del usuario.

models.py

1. **Usuario:**
    - Modelo que representa un usuario en el sistema, incluyendo sus datos y relaciones con partidas y amistades.
2. **Mapa:**
    - Modelo que representa un mapa del juego, incluyendo su nombre y las partidas relacionadas.
3. **Partida:**
    - Modelo que representa una partida jugada, incluyendo datos como puntuación, precisión, y tiempo completado.
4. **Amistad:**
    - Modelo que representa una amistad entre dos usuarios, incluyendo la fecha en que se estableció la amistad.