KNIGHT'S NIGHT TRAVEL

RAÚL TOLEDO PROCOPIO
DESARROLLO DE APLICACIONES MULTIPLATAFORMA
TFG

Índice

Índice de tablas e imágenes

Imágenes:

Tablas:

Introducción

La introducción del juego se encuentra en el documento de diseño del juego, adjuntado al proyecto como GDD.pdf o pulsando este enlace.

Descripción

Al ejecutar el juego, pasaremos a una pantalla de login, en la que tendemos que logearnos o registrarnos en caso de que no tengamos cuenta. Una vez hecho esto, pasaremos al menú principal para decidir si queremos jugar una nueva partida o continuar con la que ya tenemos.

Además de los menús, tenemos 10 escenas principales que conforman los niveles jugables.

El juego cuenta con una estructura de carpetas limpia y clara, para poder movernos por ella de una manera simple y ordenada. Entre ellas se distinguen carpetas para los niveles, los personajes, los enemigos, los proyectiles...

Instalación

Para instalar el juego, solo necesitamos descargarlo de itch.io en el siguiente enlace:

https://github.com/RaulToledoProcopio/JuegoPSP

o descargarnos Godot, .NET y clonar el siguiente repositorio a nuestro pc:

https://github.com/RaulToledoProcopio/JuegoPSP

Documentación de código

Toda la documentación de la introducción y de la API, están adjuntadas en al misma carpeta de este documento tanto en su formato original como en PDF, además puedes acceder a los PDF desde los enlaces proporcionados.

Vamos a realizar la documentación de código, para ello vamos a centrarnos en primer lugar, en la parte más importante de los scripts que sean relevantes.

Player.cs – Este es el script del personaje principal

Puntos clave:

[Export] Speed, JumpVelocity, Gravity, CrouchSpeed, MaxJumps - Parámetros ajustables que controlan movimiento, salto, agachado...

```
[Export] public float Speed = 200f;          // Velocidad de movimiento del personaje.
[Export] public float JumpVelocity = -500f;  // Velocidad aplicada al personaje al saltar.
[Export] public float Gravity = 1500f;       // Fuerza de gravedad aplicada al personaje.
[Export] public float speedDagger = 300f;    // Velocidad a la que se lanza la daga.
[Export] public float CrouchSpeed = 100f;    // Velocidad de movimiento cuando el personaje está agachado.
[Export] public int MaxJumps = 2;             // Número máximo de saltos permitidos para el doble salto.
```

Ilustración 1 Parámetros de Player

_Ready() – Cacheo de nodos - Se obtienen referencias a nodos de animación, UI y sonidos para no buscarlos en cada frame.

```
public override void _Ready()
{

    // Referencias a los nodos necesarios
    animation = GetNode<AnimatedSprite2D>("AnimatedSprite2D");
    dagger = GD.Load<PackedScene>("res://Proyectiles/Daga/daga.tscn");
    _espada = GetNode<Espada>("Espada");
    _gameOverTimer = GetNode<Timer>("Timer");
    _gameOverTimer.Stop();  // Detiene el temporizador inicialmente.
    _healthBar = GetNode<ProgressBar>("../UI/HealthBar");
    _healthBar.Value = GameState.Health;  // Inicializa la barra de salud.
    _ammoLabel = GetNode<Label>("../UI/Ammo");
    _idleCollisionShape = GetNode<CollisionShape2D>("idleCollisionShape2D");
    _crouchCollisionShape = GetNode<CollisionShape2D>("crouchCollisionShape2D");
    _idleCollisionShape.Disabled = false;  // Colisión de pie activa.
    _crouchCollisionShape.Disabled = true; // Colisión agachado desactivada.

    jumpSoundPlayer = GetNode<AudioStreamPlayer>("Jump");
    throwSoundPlayer = GetNode<AudioStreamPlayer>("Throw");
    swordSoundPlayer = GetNode<AudioStreamPlayer>("Sword");
    deathPlayer = GetNode<AudioStreamPlayer>("Death");
    hitPlayer = GetNode<AudioStreamPlayer>("Hit");
}
```

Ilustración 2 Ready() Player

_PhysicsProcess(delta) – Movimiento y lógica principal - Aplica gravedad, lee input (mover, saltar, agacharse, atacar), lanza proyectiles y actualiza la animación.

```
public override void _PhysicsProcess(double delta)
{
    if (_isDead)
        return;  // Si el personaje está muerto, no hace nada.

    // Prioriza la animación de daño si sigue reproduciéndose
    if (animation.Animation == "Hit" && animation.IsPlaying())
        return;

    Vector2 velocity = Velocity;  // Velocidad actual del personaje.

    _healthBar.Value = GameState.Health;
    _ammoLabel.Text = GameState.Ammo.ToString();
```

Ilustración 3 PhysicsProcess Player

```
// Comprobar si está agachado
isCrouching = Input.IsActionPressed("ui_down");
_idleCollisionShape.Disabled = isCrouching;      // Ajusta colisiones según estado.
_crouchCollisionShape.Disabled = !isCrouching;

// Aplicar gravedad si está en el aire
if (!IsOnFloor())
    velocity.Y += Gravity * (float)delta;
else
    jumpCount = 0;  // Reinicia contador al tocar suelo.

// Entrada de movimiento horizontal
Vector2 direction = Input.GetVector("ui_left", "ui_right", "ui_up", "ui_down");
```

Ilustración 4 PhysicsProcess1 Player

```
// Ataque con espada
if (Input.IsActionJustPressed("ui_attack") && !isAttacking && !isCrouching)
{
    isAttacking = true;  // Marca estado de ataque.
    Attack();            // Lanza método de ataque.
    animation.Play("Attack"); // Reproduce animación de ataque.
    swordSoundPlayer.Play();
}

// Ataque con daga
if (GameState.Ammo > 0 && Input.IsActionJustPressed("ui_daga") && !isAttacking && !isCrouching)
{
        var instDagger = (Daga)dagger.Instantiate();
        instDagger.Position = GlobalPosition + new Vector2(animation.FlipH ? -15 : 15, -15);
        instDagger.RotationDegrees = RotationDegrees;
        instDagger.speedDagger = animation.FlipH ? -speedDagger : speedDagger;
        if (animation.FlipH)
        instDagger.Scale = new Vector2(-Mathf.Abs(instDagger.Scale.X), instDagger.Scale.Y);
        GetParent().AddChild(instDagger);
        throwSoundPlayer.Play();

        GameState.ChangeAmmo(-1);
}
```

Ilustración 5 PhysicsProcess2 Player

Attack() – Activación temporal del collider de la espada, permite detectar colisiones de ataque durante un breve instante.

```
// Activa el área de la espada durante un ataque
public void Attack()
{
    _espada.ActivarEspada();
    GetTree().CreateTimer(0.1f).Timeout += () => _espada.DesactivarEspada();
}
```

Ilustración 6 Attack() Player

UpdateAnimation(direction) – Cambia el estado y reproduce la animación adecuada según salud, ataque, salto, etc.

```
// Decide qué animación reproducir según el estado actual.
private void UpdateAnimation(Vector2 direction)
{
    PlayerState state;

    if (GameState.Health <= 0 && IsOnFloor())
    {
        state = PlayerState.Death;
        if (!_isDead)
        {
            _gameOverTimer.Start(5.5f);  // Inicia temporizador de GameOver
            _isDead = true;
            var session = GetNode<SessionManager>("/root/SessionManager");
            var saveManager = GetNode<SaveManager>("/root/SaveManager");
            saveManager.DeleteGame(session.Username);
        }
    }
    // Animación de daño
    else if (animation.Animation == "Hit" && animation.IsPlaying())
        state = PlayerState.Hit;
    // Animación de ataque
    else if (animation.Animation == "Attack" && animation.IsPlaying())
        state = PlayerState.Attack;
    // En el aire
    else if (!IsOnFloor())
        state = PlayerState.Jump;
    // Agachado
    else if (isCrouching)
        state = direction.X != 0 ? PlayerState.CrouchWalk : PlayerState.Crouch;
    // Corriendo
```

Ilustración 7 UpdateAnimation(direction) Player

Enemy1 – Ejemplo Script de enemigos de tier1, aplicable a Enemy3 y 7.
Puntos clave:

Variables - Velocidad, vida y daño base del enemigo, así como todas las variables exportables o constantes.

```
public const float Speed = 100.0f;  // Velocidad de movimiento del enemigo.
private AnimatedSprite2D animation;  // Referencia a AnimatedSprite2D para animaciones.
private Timer _deathTimer;  // Temporizador para eliminar al enemigo después de morir.
private Vector2 _movementDirection = new Vector2(-1, 0);  // Dirección inicial del movimiento (izquierda).
private int hp = 100;  // Puntos de vida del enemigo.
private bool _timerStarted = false;  // Bandera para evitar que se reinicie el temporizador.
[Export] public int damage = 10; // Daño que inflinge el enemigo
private AudioStreamPlayer deathSound;
private AudioStreamPlayer hitSound;
private Vector2 _knockbackVelocity = Vector2.Zero;
private bool _isKnockedBack = false;
private const float KnockBackSpeed    = 200f;  // fuerza horizontal
private const float KnockBackUpForce  = 100f;  // fuerza vertical
private const float Gravity           = 1000f;  // gravedad
```

Ilustración 8 Variables Enemies Tier1

_Ready() – Iniciar animación y sonidos, arranca el “Walk” y cachea AnimatedSprite2D, Timer, AudioStreamPlayer.

```
public override void _Ready()
{
    // Obtención de los nodos necesarios
    animation = GetNode<AnimatedSprite2D>("AnimatedSprite2D");
    _deathTimer = GetNode<Timer>("Timer");

    animation.Play("Walk");// Reproducir la animación de caminar al inicio
    _deathTimer.Stop(); // Asegurar de que el Timer no esté corriendo al principio
    deathSound = GetNode<AudioStreamPlayer>("Dead");
    hitSound = GetNode<AudioStreamPlayer>("Hit");
}
```

Ilustración 9 Ready() Enemies Tier1

_PhysicsProcess(delta) – Movimiento con rebote - Se desplaza en línea recta, invierte dirección al chocar con muros y aplica knockback cuando recibe daño.

```gdscript
public override void _PhysicsProcess(double delta)
{
    if (_isKnockedBack)
    {
        // Aplica gravedad al retroceso
        _knockbackVelocity.Y += Gravity * (float)delta;
        Velocity = _knockbackVelocity;
        MoveAndSlide();

        // Cuando tocas el suelo, terminas el knockback
        if (IsOnFloor())
        {
            _isKnockedBack = false;
            _knockbackVelocity = Vector2.Zero;
        }
        return; // no ejecutas el movimiento normal mientras haces knockback
    }

    // ... Aquí va tu movimiento normal
```

Ilustración 10 _PhysicsProcess Enemies Tier1

TakeDamage(damage) – Retroceso y detección de muerte - Reduce hp, reproduce hitSound, aplica fuerza de retroceso y marca _isKnockedBack.

```gdscript
public void TakeDamage(int damage)
{
    hp -= damage;

    // Calcula la dirección: 1 si el player está a la izquierda, -1 si está a la derecha
    var player = GetNode<Player>("../Player");
    float direction = (player.Position.X < Position.X) ? 1f : -1f;

    hitSound?.Play();
    _knockbackVelocity = new Vector2(direction * KnockBackSpeed, -KnockBackUpForce);
    _isKnockedBack = true;
}
```

Ilustración 11 TakeDamage(damage) Enemies Tier1

Enemy2 – Ejemplo Script de enemigos de tier2, aplicable a Enemy5
Puntos clave:

Además de tener algunos métodos en común con el anterior añadimos la opción de lanzar un proyectil basado en una escena que se lanza desde una posición relativa al propio enemigo.

FireBallScene + fireRate - Referencia a escena de proyectil y cadencia de disparo entre fases de “Idle” y “Attack”.

```
[Export] public PackedScene FireBallScene;  // Referencia a la escena de la bola de fuego.
private float fireRate = 1.5f;  // Tiempo entre disparos.
```

Ilustración 12 FireBallScene Enemies Tier2

ShootFireBall() – Instanciación de proyectil - Ajusta posición y velocidad del FireBallScene según orientación.

```
// Método para disparar la bola de fuego.
private void ShootFireBall()
{
    // Instanciamos la bola de fuego a partir de la escena.
    FireBall fireBallInstance = (FireBall) FireBallScene.Instantiate();
    // Ajusta la dirección de la velocidad de la bola de fuego según si la animación está volteada horizontalmente.
    fireBallInstance.speedDagger = (animation.FlipH ? -fireBallInstance.speedDagger : fireBallInstance.speedDagger);
    // Hacer un cast a Node2D para poder usar la propiedad Position.
    Node2D fireBallNode = fireBallInstance as Node2D;
    // Establece la posición de la bola de fuego en relación con la posición actual del personaje.
    fireBallNode.Position = Position + new Vector2(animation.FlipH ? 15 : -15, -15);
    // Añadimos la bola de fuego al árbol de nodos del juego.
    GetParent().AddChild(fireBallInstance);
}
```

Ilustración 13 ShootFireBall() Enemies Tier2

Enemy4 – Ejemplo Script de enemigos de tier3, aplicable a Enemy6

Puntos clave:

En estos enemigos, la mecánica que añadimos es que tienen un área de detección que detecta al player, cuando esto sucede dejan de patrullar y hacen una animación de ataque al mismo.

State enum {Patrol, Attack, Death} - Define el comportamiento: patrullar, atacar al detectar jugador, o morir.

```
private enum State { Patrol, Attack, Death }
private State _state = State.Patrol;
```

Ilustración 14 State enum {Patrol, Attack, Death} Enemies Tier3

DetectionArea + Timer de ataque - Usa un Area2D para entrar/salir de rango y un Timer para cadencia de golpes.

```
_detectionArea.Connect("body_entered", new Callable(this, nameof(OnDetectionBodyEntered)));
_detectionArea.Connect("body_exited",  new Callable(this, nameof(OnDetectionBodyExited)));
_attackTimer.Connect("timeout",         new Callable(this, nameof(OnAttackTimerTimeout)));
```

Ilustración 15 DetectionArea Enemies Tier3

PatrolBehavior() - mueve y rebota en muros.

```
private void PatrolBehavior()
{
    Velocity = new Vector2(_patrolDir.X * Speed, 0);
    MoveAndSlide();

    if (IsOnWall())
    {
        _patrolDir.X = -_patrolDir.X;
        _anim.FlipH  = !_anim.FlipH;
    }

    if (_anim.Animation != "Walk")
        _anim.Play("Walk");

    _weaponArea.Monitoring = false;
    _weaponShape.Disabled  = true;
}
```

Ilustración 16 PatrolBehavior() Enemies Tier3

AttackBehavior() - detiene, invierte sprite, activa el collider del ataque.

```
private void AttackBehavior()
{
    Velocity = Vector2.Zero;
    _anim.FlipH = (_player.Position.X < Position.X) ? false : true;

    float x = Mathf.Abs(_weaponOriginalPosition.X);
    _weaponShape.Position = _anim.FlipH
        ? new Vector2(-x, _weaponOriginalPosition.Y)
        : new Vector2( x, _weaponOriginalPosition.Y);

    if (_anim.Animation != "Attack")
        _anim.Play("Attack");

    _weaponArea.Monitoring = true;
    _weaponShape.Disabled  = false;
}
```

Ilustración 17 AttackBehavior() Enemies Tier3

Boss – Script de enemigos de tier4, aplicable parcialmente a Miniboss ya que es

una mezcla de mecánicas de todos los enemigos anteriores.

En este caso, hemos copiado las mecánicas del resto de enemigo, con la novedad de que cada ciertos segundos el boss baja y hace un barrido a diferente altura.

State enum {PatrolHigh, MovingDown, PatrolLow, MovingUp, Death} - Controla patrones de patrulla vertical y ataques de rayo.

```
private enum State { PatrolHigh, MovingDown, PatrolLow, MovingUp, Death }
private State _state = State.PatrolHigh;
```

Ilustración 18 State enum Boss

Lógica de patrulla vertical - Cambia State tras HighPatrolDuration y mueve hacia LowY, regresando luego arriba.

```
case State.PatrolHigh:
    vel = new Vector2(_patrolDir.X * Speed, 0);
    if (_stateTimer >= HighPatrolDuration)
    {
        _state = State.MovingDown;
        _stateTimer = 0f;
        _lightningTimer.Stop();
    }
```

Ilustración 19 Lógica de patrulla vertical Boss

Ahora vamos a proceder a la documentación de otra parte importante del código, como son los singleton. Los singleton son scripts que son cargados automáticamente y son accesible globalmente desde cualquier parte del juego.

AudioManager.cs - Controlar la música de fondo del juego en función del nivel.

Puntos clave:

_tracks : Guarda todas las pistas en memoria para evitar cargas repetidas.

```
private readonly Dictionary<int, AudioStream> _tracks = new()
{
    { 1, ResourceLoader.Load<AudioStream>("res://BSO/1-Intro.mp3")! },
    { 2, ResourceLoader.Load<AudioStream>("res://BSO/2-Fase1.mp3")! },
    { 3, ResourceLoader.Load<AudioStream>("res://BSO/3-Fase2.mp3")! },
    { 4, ResourceLoader.Load<AudioStream>("res://BSO/11-Fase10.mp3")! },
    { 5, ResourceLoader.Load<AudioStream>("res://BSO/4-Fase3.mp3")! },
    { 6, ResourceLoader.Load<AudioStream>("res://BSO/10-Fase9.mp3")! },
    { 7, ResourceLoader.Load<AudioStream>("res://BSO/12-Créditos.mp3")! },

};
```

Ilustración 20 Track AudioManager

_Ready() y _player : Se cachea la referencia a AudioStreamPlayer para optimizar llamadas.

```
private AudioStreamPlayer _player = null!;

public override void _Ready()
{
    _player = GetNode<AudioStreamPlayer>("Audio");
}
```

Ilustración 21 Ready() AudioManager

PlayForLevel() : Llama a GetGroupForLevel para saber qué pista corresponde al nivel y solo cambia la música si el grupo es distinto al actual.

```
public void PlayForLevel(int level)
{
    int group = GetGroupForLevel(level);
    if (group == _currentGroup)
        return;

    _currentGroup = group;
    if (_tracks.TryGetValue(group, out var stream))
    {
        _player.Stream = stream;
        _player.Play();
    }
}
```

Ilustración 22 PlayForLevel() AudioManager

GetGroupForLevel() : Centraliza la lógica de asignación nivel→grupo, fácil de ajustar.

```
private int GetGroupForLevel(int level)
{
    if (level == 0)
    return 1;

    if (level == 2 || level == 3)
    return 2;

    if (level == 4 || level == 5)
    return 3;

    if (level == 6 || level == 7)
    return 4;

    if (level == 8 || level == 9)
    return 5;

    if (level == 10)
    return 6;

    if (level == 99)
    return 7;

    return 0;
}
```

Ilustración 23 GetGroupForLevel() AudioManager

GameState.cs - Mantener el estado global del jugador entre escenas.
Puntos clave:

Variables:

Health - Vida global del jugador, persistente entre escenas.
Ammo - Munición global del jugador, persistente.
DiedByFall - Indica si el jugador murió por caída, para gestionar efectos de sonido.
CurrentUI - Referencia al controlador de la interfaz para actualizar valores.

```
// Vida y munición globales persistentes
public static float Health = 100f;
public static int Ammo = 10;
public static bool DiedByFall = false; // Quiero controlar esto para evitar que el sonido de muerte se solape con el de caída.
public static UI CurrentUI; // Para que el UI pueda refrescarse al cambiar valores
```

Ilustración 24 Variables GameState

ChangeHp(amount) - Ajusta Health con clamp entre 0–100 y notifica a la UI.

```
public static void ChangeHp(float amount)
{
    Health = Mathf.Clamp(Health + amount, 0, 100);
    CurrentUI?.UpdateHealth(Health);
}
```

Ilustración 25 ChangeHp(amount) GameState

ChangeAmmo(amount) - Ajusta Ammo con mínimo cero y actualiza la UI.

```
public static void ChangeAmmo(int amount)
{
    Ammo = Mathf.Max(Ammo + amount, 0);
    CurrentUI?.UpdateAmmo(Ammo);
}
```

Ilustración 26 ChangeAmmo(amount) GameState

Reset() - Revierte Health, Ammo y DiedByFall a valores iniciales.

```
public static void Reset()
{
Health = 100f;
Ammo = 10;
DiedByFall = false;
}
```

Ilustración 27 Reset() GameState

SaveManager.cs - Gestionar el guardado y carga de partidas en disco.
Puntos Clave:

SaveGame(username, levelPath) - Serializa SaveData a JSON y lo escribe en user://save_{username}.json.

```
public void SaveGame(string username, string levelPath)
{
    var saveData = new SaveData
    {
        Health = GameState.Health,
        Ammo = GameState.Ammo,
        LevelPath = levelPath
    };

    string json = JsonSerializer.Serialize(saveData);
    string savePath = $"user://save_{username}.json";
    using var file = FileAccess.Open(savePath, FileAccess.ModeFlags.Write);
    file.StoreString(json);
    GD.Print("Juego guardado en: " + savePath);
}
```

Ilustración 28 SaveGame(username, levelPath) SaveManager

LoadGame(username) - Lee el JSON de user://save_{username}.json y deserializa a SaveData.

```
public SaveData LoadGame(string username)
{
    string savePath = $"user://save_{username}.json";
    if (!FileAccess.FileExists(savePath))
        return null;

    using var file = FileAccess.Open(savePath, FileAccess.ModeFlags.Read);
    string json = file.GetAsText();
    return JsonSerializer.Deserialize<SaveData>(json);
}
```

Ilustración 29 LoadGame(username) SaveManager

DeleteGame(username) - Elimina el archivo de guardado si existe.

```
public void DeleteGame(string username)
{
    var dir = DirAccess.Open("user://");
    if (dir != null && dir.FileExists($"save_{username}.json"))
        dir.Remove($"save_{username}.json");
}
```

Ilustración 30 DeleteGame(username) SaveManager

SessionManager.cs - Almacenar el nombre de usuario durante la sesión.
Puntos clave:

Username - Almacena el nombre de usuario de la sesión actual.
Acceso global - Permite leer el usuario desde cualquier nodo.

```
public partial class SessionManager : Node
{
    public string Username { get; set; } = "";
}
```

Ilustración 31 SessionManager

Crono.cs - Gestionar el cronómetro de juego y enviar puntuaciones online.

elapsedTime - Acumula el tiempo transcurrido mientras el cronómetro corre.
timerRunning - Controla si el cronómetro debe actualizarse en _Process.
canSubmitScore - Indica si está permitido enviar la puntuación al detener el cronómetro.

```
private float elapsedTime = 0f;
private bool timerRunning = false;
private bool canSubmitScore = false;  // Controla si se puede enviar la puntuación
```

Ilustración 32 Variables Crono

_Process(delta) - Formatea elapsedTime a MM:SS:DS y actualiza el Label.

```
public override void _Process(double delta)
{
    if (timerRunning)
    {
        elapsedTime += (float)delta;

        // Formateamos el tiempo en MM:SS:DS
        int minutes = (int)(elapsedTime / 60);
        int seconds = (int)(elapsedTime % 60);
        int decimas = (int)((elapsedTime % 1) * 10);

        var tiempoLabel = GetNode<Label>("/root/CronometroUI/CronoUI");
        if (tiempoLabel != null)
            tiempoLabel.Text = $"{minutes:D2}:{seconds:D2}:{decimas:D2}";
    }
}
```

Ilustración 33 _Process(delta) Crono

StopTimer(submitScore) - Detiene el cronómetro y, si procede, envía la puntuación.

```
public void StopTimer(bool submitScore = false)
{
    timerRunning = false;
    if (submitScore && canSubmitScore)  // Solo enviar puntuación si se debe
    {
        GD.Print("Cronómetro detenido: " + elapsedTime + " segundos.");
        _ = SubmitScoreAsync(elapsedTime);
    }
}
```

Ilustración 34 StopTimer(submitScore) Crono

SubmitScoreAsync(timeSeconds) - Obtiene Username de SessionManager, construye DTO y usa HttpClient para POST.

```
private async Task SubmitScoreAsync(float timeSeconds)
{
    // Recuperar el username del Autoload SessionManager
    var session = GetNode<SessionManager>("/root/SessionManager");
    string username = session?.Username;
    if (string.IsNullOrEmpty(username))
    {
        GD.PrintErr("SessionManager.Username vacío, no se envía puntuación.");
        return;
    }

    // Construir el DTO anónimo y enviarlo
    var scoreDto = new
    {
        username = username,
        timeSeconds = timeSeconds
    };
```

Ilustración 35 SubmitScoreAsync(timeSeconds) Crono

Diario de desarrollo

Tabla 1 Diario de desarrollo

	Tarea	Fecha
1	Diseño del documento	20/27 marzo
2	Creación de personaje	20/27 marzo
3	Creación de los primeros niveles	27 marzo/3 abril
4	Demo viable	3/10 abril
5	Persistencia de datos para el login y el registro	10/24 abril
6	Persistencia de datos para las puntuaciones	10/24 abril
7	Diseño de la BSO y efectos de sonido	24 abril/8 mayo
8	Inclusión de nuevos niveles	8/22 mayo
9	Depuración de bugs	22/29 mayo
10	Inclusión de nuevos enemigos	29 mayo/10 junio
11	Documentación	5/12 junio

Pruebas

Al ser un videojuego, he realizado las pruebas manuales sobre el login y el registro, de las cuales adjuntos capturas y están documentadas en la documentación de la API, incluida en esta misma carpeta en el documento API.PDF o bien en este enlace y he distribuido el juego a varias personas de confianza para que hagan de betatesters.

Distribución

Juego distribuido de manera gratuita mediante Itch.io

Conclusiones

Comparativa:

La jugabilidad final del producto ha resultado ser exactamente la esperada; el juego se ha desarrollado tal y como lo había concebido desde un principio.

Durante el proceso, he aprendido nuevas herramientas y técnicas, como el uso de TileMapLayers y TileSets para pintar niveles, algo que no dominaba en la primera entrega.

También he logrado implementar nuevas mecánicas en los enemigos, algunas con mayor complejidad técnica, como activar un Area2D que haga daño solo durante una animación específica, y que además se oriente hacia la posición del jugador.

Mejoras futuras:

Dentro del propio juego, en el menú final, hay una sección dedicada a las mejoras futuras previstas. Entre ellas, me gustaría destacar la intención de añadir una historia más desarrollada con un lore definido, así como cinemáticas más elaboradas que complementen la experiencia actual.

Bibliografía

https://docs.godotengine.org
https://docs.godotengine.org/es/4.x/classes/index.html#csharp-api
https://learn.microsoft.com/es-es/dotnet/csharp/
https://learn.microsoft.com/es-es/dotnet/api/
https://chatgpt.com/
https://itch.io/
https://suno.com/home
https://www.bing.com/images/create?setlang=es