

Politécnico do Porto
Escola Superior de Media Artes e Design

Carlos Pinto Guedes – 9170138
Rodrigo Daniel Lopo Queirós - 9170312

Where is Light

Licenciatura em Tecnologias de Sistemas de Informação para a Web

Desenvolvimento de Jogos

Orientação: Prof. Gustavo Carneiro

Vila do Conde, janeiro de 2020

Abstract

This report aims to document all the stages in carrying out the project “Where is Light”. “Where is Light” is the game created by my group in the Game Development discipline. In the course of the report, we will explain the process of creating the game, from the definition of the idea, to the final result.

Resumo

Este relatório tem como objetivo documentar todas as etapas na realização do projeto “Where is Light”. “Where is Light” é o jogo criado pelo meu grupo na disciplina de Desenvolvimento de Jogos. No decorrer do relatório, iremos explicar o processo de criação o jogo, desde a definição da ideia, até ao resultado final.

SUMÁRIO

LISTA DE FIGURAS	3
0 INTRODUÇÃO	5
1 CONCEITO DO JOGO.....	6
2 OBJETIVOS.....	7
3 PROCESSO DE DESENVOLVIMENTO	8
3.1 COMPONENTES	8
3.1.1 PLAYER.....	9
3.1.2 PLAYER UI	12
3.1.3 NÍVEIS E ESTADOS	13
3.1.4 GAME STATE MANAGER.....	16
3.1.5 CÂMERA	18
3.1.6 INIMIGOS.....	18
3.2 MÚSICA E SOM.....	23
3.2.1 MÚSICA	23
3.2.1 SONS.....	24
CONCLUSÃO	25
REFERÊNCIAS BIBLIOGRÁFICAS	26

LISTA DE FIGURAS

FIGURA 1 - COMPONENTES DO JOGO.....	8
FIGURA 2 – PLAYER.....	9
FIGURA 3 - SALTO SIMPLES (FICHEIRO MOVEPLAYER.CS)	10
FIGURA 4 - MELHORAMENTO DO SALTO (FICHEIRO BETTERJUMPPLAYER.CS)	10
FIGURA 5 - PLAYER ANIMATOR.....	11
FIGURA 6 - INFORMAÇÃO DO PLAYER (PLAYER UI)	12
FIGURA 7 – MENU.....	13
FIGURA 8 - NÍVEL 1.....	13
FIGURA 9 - NÍVEL 2.....	14
FIGURA 10 - COROUTINES PLATAFORMAS	14
FIGURA 11 - NÍVEL 3.....	15
FIGURA 12 - COMO JOGAR.....	15
FIGURA 13 – PAUSA	15
FIGURA 14 - GAME OVER	16
FIGURA 15 - GAME STATE MANAGER.....	16
FIGURA 16 - VERIFICAÇÃO DE WAVES.....	17
FIGURA 17 - STATE MACHINE	17
FIGURA 18 - INSTANTIATE ENEMIES	18
FIGURA 19 – NÍVEL 2 LOADER	18

FIGURA 20 - ENEMY4.....	19
FIGURA 21 - ENEMY4 SHOT	19
FIGURA 22 - COMPOSIÇÃO DO ENEMY4.....	19
FIGURA 23 - FUNÇÃO DE LANÇAMENTO DOS ESPINHOS	19
FIGURA 24 - ENEMY2.....	20
FIGURA 25 – GHOST	20
FIGURA 26 – SPELL.....	20
FIGURA 27 - COMPOSIÇÃO DO ENEMY5.....	21
FIGURA 28 – SPERM	21
FIGURA 29 - FUNÇÃO PARA OLHAR PARA O JOGADOR	21
FIGURA 30 - FUNÇÃO PARA SEGUIR A POSIÇÃO DO JOGADOR.....	21
FIGURA 31 - FINAL BOSS.....	22
FIGURA 32 – FIREBALL.....	22
FIGURA 33 – GEISER	22
FIGURA 34 – METEORO.....	22
FIGURA 35 - GESTÃO DOS ATAQUES DO FINAL BOSS	23
FIGURA 36 - TEMA INICIAL NO LOGIC PRO X	23
FIGURA 37 - PASTA SONS E MÚSICA	24

O INTRODUÇÃO

Ao início vieram muitos jogos à ideia, desde 3D shooters até jogos mais para mobile. Mas devido à influencia por jogos que gostamos, decidimos optar por um jogo que nos desse para ir um bocado a todas as áreas do desenvolvimento de jogos (arte, história, cutscenes, música, effects, animation, gameplay, etc).

Foi por isso que optamos por uma ideia difícil e grande, um metrovania. A história surgiu por influência de animes e cultura japonesa, mas sendo ela original e nunca antes vista.

1 CONCEITO DO JOGO

Baseado em jogos plataforma e metrovania, “Where is light” é um plataformer arena com waves de monstros, isto é, a medida que se vai derrotando os monstros presentes na arena vão surgindo mais e vamos passando entre arenas até chegar e derrotar o último boss.

A ideia não começou assim. No início, o jogo era suposto permitir a exploração de mapas, desbloquear skills e permitir a evolução da personagem, sendo bastante parecido a jogo “Hollow Knight” e “Ori and the Blind Forest” neste sentido. Era suposto termos também história no jogo, contada em cutscenes antes do início do jogo e no seu final, tendo também não sido possível fazer o desfecho da história como planeado.

No relatório do Rodrigo Queirós será melhor retratada a história que envolve o jogo.

2 OBJETIVOS

Um dos principais objetivos que tínhamos em mente para este projeto eram:

- Fazer um jogo 2D e meio, composto por várias layers;
- Todos os componentes do jogo serem originais (música, sons, arte), sendo assim, o jogo 100% nosso;
- Conseguir finalizar o jogo com todos os elementos que se definiram no início (waves de inimigos, final boss, etc.);
- Conseguir transmitir uma mensagem (história), com a ajuda de Cutscenes. No final, não conseguimos fazer (não só por questões de tempo, mas também por questões de organização). Este é um dos aspectos que tivemos a infelicidade de não conseguir finalizar.
- Uma das nossas maiores inspirações foi o Hollow Knight, o que nos fez querer atingir um nível de Programação e Arte (dentro dos nossos conhecimentos) o mais perto possível do jogo.

No final, esses objetivos foram concretizados, onde o jogo possui todos os elementos definidos no início. Notamos também que, no final do projeto, conseguimos reter imensa informação relativa a C#, o que nos levou a fazer os nossos próprios métodos para conseguir atingir determinados objetivos.

3 PROCESSO DE DESENVOLVIMENTO

O projeto foi dividido em três vertentes:

- Arte;
- Programação;
- Música e Som;

O Rodrigo ficou com uma parte muito importante do jogo, a Identidade Visual, e um pouco da programação.

A minha parte, foi maioritariamente a Programação e a parte da Música e Som.

Ficou assim decidido pois o Rodrigo tem muito mais jeito no que eu no que toca a arte, por isso foi o melhor das soluções.

3.1 COMPONENTES

Maioritariamente, tratei da constituição dos níveis (level design), dos estados do jogo (State Machine), do Player e dos seus envolventes, e de maior parte dos inimigos.

Explicar como fiz é um pouco complicado, por isso, vou enumerar os componentes existentes no jogo, e explicar cada um deles:

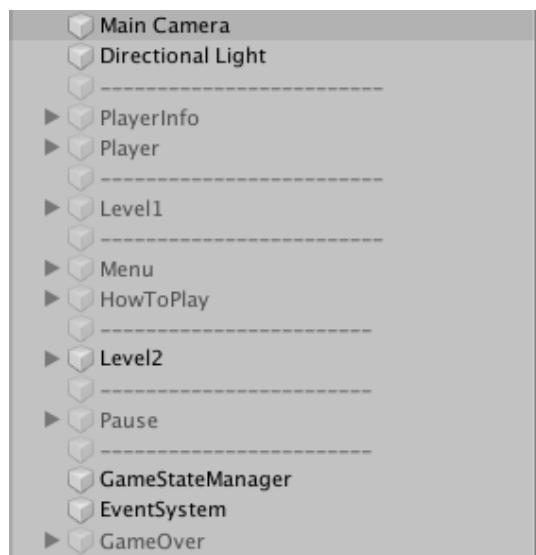


Figura 1 - Componentes do Jogo

O jogo é composto por 3 níveis e um Menu, um Player, e um componente que contém a informação do mesmo.

3.1.1 PLAYER

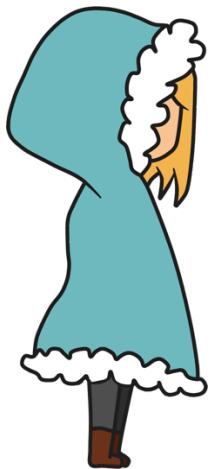


Figura 2 – Player

O Player tem 8 estados:

- Atacar;
- Morto;
- Idle;
- Salto;
- Ataque Vertical;
- Feitiço;
- Andar;
- Ultimate;

É o componente principal do jogo, por isso decidiu-se investir mais tempo nas características que possui.

Uma das decisões tomadas de melhoria foi o salto da personagem. Este, numa primeira fase, era um salto simples. O que se verificou foi que era necessário melhorar o salto (com a Física). Então, além do básico Salto da personagem, criou-se outro Script para ajudar a lidar com as questões da Física.

```

public void Jump(string typeOfJump)
{
    //Jump
    if (typeOfJump == "normal")
    {
        animator.SetBool("Jumping", true);
        isJumping = true;
        playerRb.velocity = Vector2.up * jumpVelocity;
        //animator.SetBool("Jump", true);
    }

    if (typeOfJump == "vertical")
    {
        isJumping = true;
        animator.SetBool("Jumping", true);
        animator.SetBool("VerticalAttack", true);
        playerRb.velocity = Vector2.up * jumpVelocity;
    }
}

```

Figura 3 - Salto Simples (Ficheiro MovePlayer.cs)

```

if (rb.velocity.y < 0)
{
    rb.velocity += Vector2.up * Physics2D.gravity.y * (fallMultiplier - 1) * Time.deltaTime;
}
else if (rb.velocity.y > 0 && !Input.GetButtonDown("Jump"))
{
    rb.velocity += Vector2.up * Physics2D.gravity.y * (lowJumpMultiplier - 1) * Time.deltaTime;
}

if (rb.velocity.y < -0.1)
{
    animator.SetBool("IsFalling", true);
    isFalling = true;
}
else
{
    animator.SetBool("IsFalling", false);
    isFalling = false;
}

```

Figura 4 - Melhoramento do Salto (Ficheiro BetterJumpPlayer.cs)

Vou mencionar também a funcionalidade de dar Freeze a um inimigo com o Spell.

Ao invocar um Spell, o jogador congela um inimigo durante 2 segundos. Isto é possível com dois métodos simples.

```

IEnumerator FrozenEnemy()
{
    gameObject.GetComponent<Renderer>().material.color = Color.cyan;

    yield return new WaitForSeconds(frozedTime);
    isFrozen = false;
    anim.enabled = true;
    speed = initialSpeed;
    gameObject.GetComponent<Renderer>().material.color = Color.white;
}

```

Figura 5 - Função de Congelar

```

IEnumerator EnemyMoveLeft()
{
    for (i = 1; i < 70; i++)
    {
        if (!isFrozen)
        {
            transform.position += transform.right * -speed;
        }
        else
        {
            i -= 1;
            speed = 0;
        }
        yield return new WaitForSeconds(0.1f);
    }

    StartCoroutine("EnemyMoveRight");
    Rotate();
}

```

Figura 6 - Movimento Inimigo

Com a função de FrozenEnemy, conseguimos que a variável isFrozen fique com o valor de true durante dois segundos, onde depois mudamo-la para false, que faz com que o inimigo continue o seu movimento.

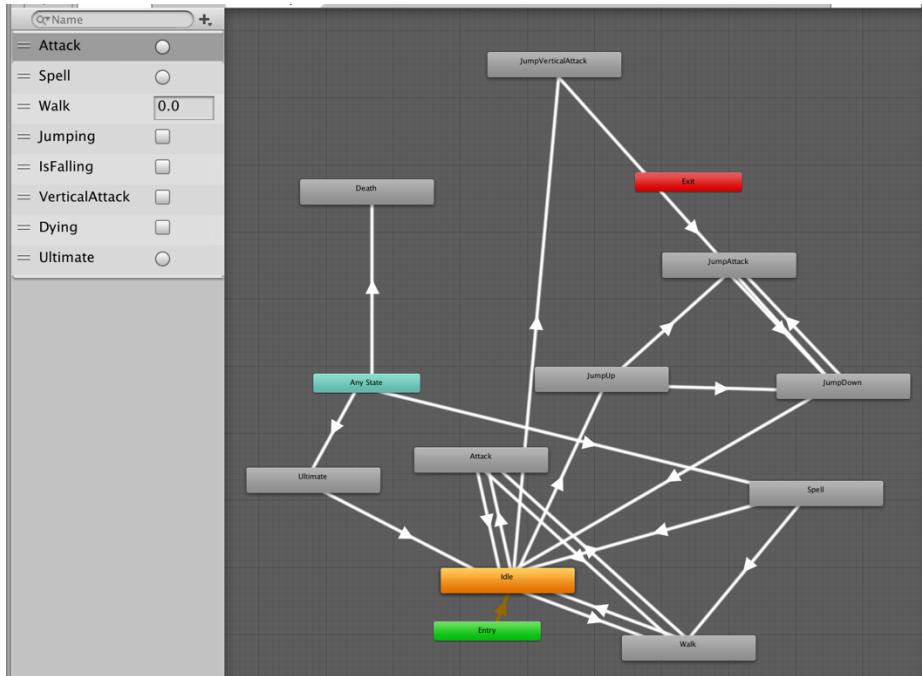


Figura 7 - Player Animator

Aqui está representado o animador do Player.

Contém todos os diferentes estados que o Player possui, como também todas as variáveis que fazem as verificações possíveis.

3.1.2 PLAYER UI

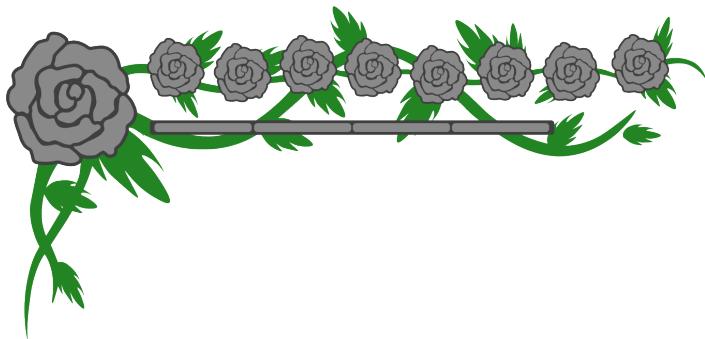


Figura 8 - Informação do Player (Player UI)

Esta informação serve para o utilizador estar sempre a par do estado da Personagem Principal (Player).

A rosa maior indica o estado do Ultimate do jogador. Essa rosa divide-se em 10 partes, e só quando essa rosa estiver totalmente preenchida, é que o jogador pode lançar o seu Ultimate.

As oito rosas demonstram as 8 vidas do jogador, quando as 8 vidas acabam, o jogador perde o jogo.

A barra em baixo retrata a Mana que o jogador tem. Para o jogador lançar um feitiço precisa de, pelo menos, 2 barras de Mana, onde cada feitiço também gasta 2 barras de Mana.

Para ganhar progresso de Mana e de Ultimate, o jogador tem que derrotar os diversos inimigos que irão aparecer ao longo do jogo.

3.1.3 NÍVEIS E ESTADOS

O conceito foi criar um objeto para cada componente, onde que, para mudar do Nível 1 para outro Estado do Jogo qualquer, era só ativar e desativar os Objetos.

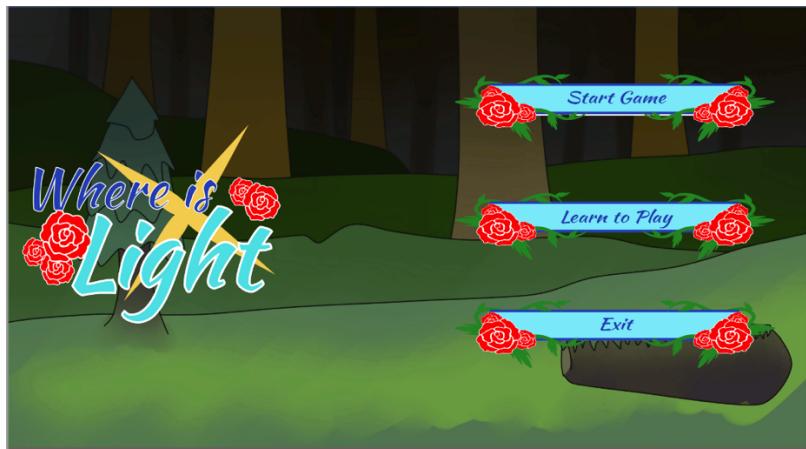


Figura 9 – Menu



Figura 10 - Nível 1

No primeiro nível, as plataformas são estáticas, onde que no segundo nível, não são.

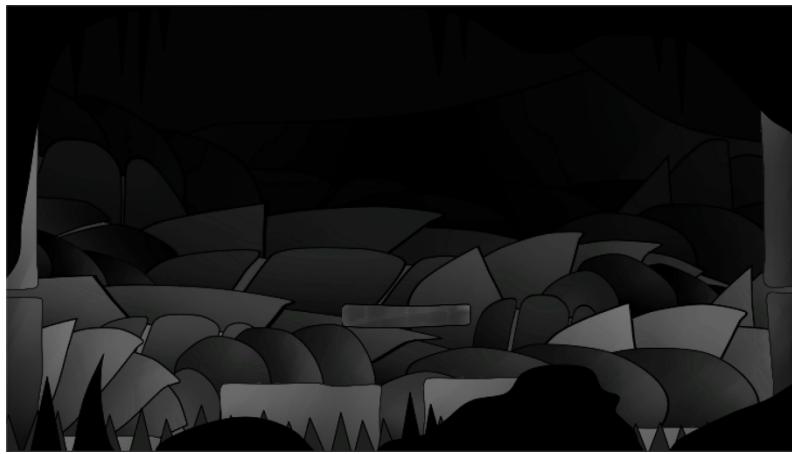


Figura 11 - Nível 2

Assim, cada plataforma é animada com duas Coroutines, onde uma desce e outra sobe a posição da plataforma.

```
IEnumerator MoveUp()
{
    for (i = 1; i < 20; i++)
    {
        transform.position += Vector3.up * speed;
        yield return new WaitForSeconds(0.1f);
    }
    StartCoroutine("MoveDown");
}

IEnumerator MoveDown()
{
    for (i = 1; i < 20; i++)
    {
        transform.position += Vector3.up * -speed;
        yield return new WaitForSeconds(0.1f);
    }
    StartCoroutine("MoveUp");
}
```

Figura 12 - Coroutines Plataformas

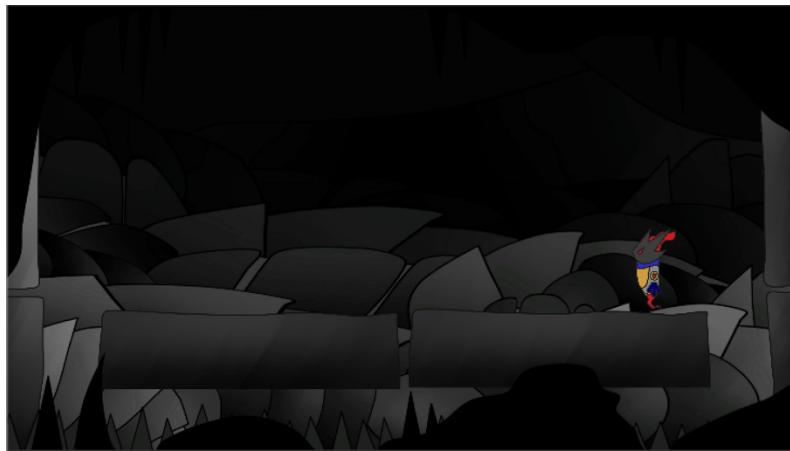


Figura 13 - Nível 3

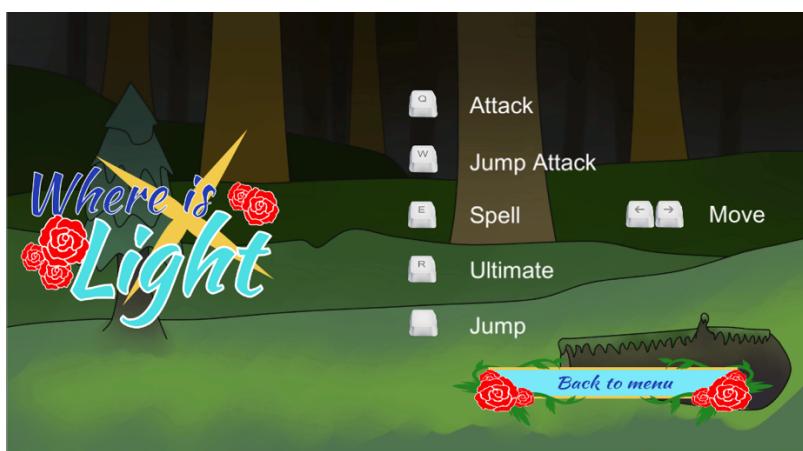


Figura 14 - Como Jogar



Figura 15 – Pausa



Figura 16 - Game Over

3.1.4 GAME STATE MANAGER

No Game State Manager, tentou-se que o mesmo tomasse conta de todas as variáveis importantes e relevantes para o decorrer do jogo.



Figura 17 - Game State Manager

Estes botões são os botões que encontramos no Menu, na Pause, etc.

No Script encontram-se várias funções de gestão de jogo. Uma delas é mesmo a gestão das Waves.

Essa mesma função vai verificando quantos inimigos o jogador já eliminou. À medida que vai matando, o progresso de waves vai aumentando.

```

private void checkEnemiesKilled()
{
    if (enemiesKilled == 2 && level1loader.currentWave == 1)
    {
        level1loader.currentWave++;
        instantiateEnemies(GameObject.Find("Level1"));
    }
    if (enemiesKilled == 4 && level1loader.currentWave == 2)
    {
        level1loader.currentWave++;
        instantiateEnemies(GameObject.Find("Level1"));
    }
    if (enemiesKilled == 8 && level1loader.currentWave == 3)
    {
        level1loader.currentWave++;
        instantiateEnemies(GameObject.Find("Level1"));
    }
    if (enemiesKilled == 10 && level1loader.currentWave == 4)
    {
        level1loader.currentWave++;
        instantiateEnemies(GameObject.Find("Level1"));
    }
    if (enemiesKilled == 13 && level1loader.currentWave == 5)
    {
        level1loader.currentWave++;
        instantiateEnemies(GameObject.Find("Level1"));
    }
    if (enemiesKilled == 15 && level1loader.currentWave == 6)
    {
        level1loader.currentWave++;
        instantiateEnemies(GameObject.Find("Level1"));
    }
    if (enemiesKilled == 21 && level1loader.currentWave == 7)
    {
        level1loader.currentWave++;
        instantiateEnemies(GameObject.Find("Level1"));
    }
    if (enemiesKilled == 28 && level1loader.currentWave == 8)
    {

```

Figura 18 - Verificação de Waves

Ao todo, o jogo tem 17 waves e 3 níveis, onde a wave final é o [Final Boss](#).

Neste script também se encontra a State Machine, que ajuda na percepção em que parte do jogo nos encontramos.

```

public void setGameState(GameState _state)
{
    if (curGameState == _state)
    {
        return;
    }

    curGameState = _state;
    switch (curGameState)
    {
        case GameState.Menu:
            canIPause = false;
            OnMenu();
            break;
        case GameState.Level1:
            canIPause = true;
            OnLevel1();
            break;
        case GameState.Level2:
            canIPause = true;
            OnLevel2();
            break;
        case GameState.Level3:
            canIPause = true;
            OnLevel3();
            break;
        case GameState.GameOver:
            canIPause = false;
            OnGameOver();
            break;
        case GameState.GameWon:
            canIPause = false;
            OnGameWon();
            break;
        case GameState.HowToPlay:
            canIPause = false;
            OnhowToPlay();
            break;
    }
}

```

Figura 19 - State Machine

Para finalizar este script, demonstro aqui a função de Instanciar Inimigos (Instantiate Enemies). Esta função vai a cada nível, e instancia os inimigos correspondentes à CurrentWave.

```
private void instantiateEnemies(GameObject lvl)
{
    if (lvl.name == "Level1") { lvl.GetComponent<Level1Loader>().instantiateLevel(); };
    if (lvl.name == "Level2") { lvl.GetComponent<Level2Loader>().instantiateLevel(); };
}
```

Figura 20 - Instantiate Enemies

A cada nível está associado um Loader, onde contém a composição de cada wave.

```
public void instantiateLevel()
{
    if (currentWave == 1) {
        enemy5Position = new Vector3(6f, 3f, -2f);
        var newObj = Instantiate(enemy5, enemy5Position, transform.rotation);
        newObj.transform.parent = GameObject.Find("Enemies2").transform;
    }
    if (currentWave == 2)
    {
        enemy6Position = new Vector3(6f, 3f, -2f);
        var newObj = Instantiate(enemy6, enemy6Position, transform.rotation);
        newObj.transform.parent = GameObject.Find("Enemies2").transform;

        enemy6Position = new Vector3(-6f, 3f, -2f);
        newObj = Instantiate(enemy6, enemy6Position, transform.rotation);
        newObj.transform.parent = GameObject.Find("Enemies2").transform;

        enemy6Position = new Vector3(0f, 3f, -2f);
        newObj = Instantiate(enemy6, enemy6Position, transform.rotation);
        newObj.transform.parent = GameObject.Find("Enemies2").transform;
    }
}
```

Figura 21 – Nível 2 Loader

3.1.5 CÂMERA

Usamos uma câmera ortográfica devido a ser um jogo em 2D e meio. Está sempre estática, pois o que muda sempre é o que a câmera vê, nunca a sua posição.

3.1.6 INIMIGOS

Todos os inimigos são “Prefabs” pois, como o nosso tipo de jogo é por waves de inimigos, tem mais sentido darmos “Instantiate” aos inimigos do que criá-los todos e mantê-los em memória.

Cada um tem três vidas, menos o [Enemy6](#) (Sperm) pois o ataque dele é considerado “suicídio”.

3.1.6.1 SHOTGUN (ENEMY4)



Figura 22 - Enemy4



Figura 23 - Enemy4 Shot

Este inimigo, a cada 2 segundos, roda aleatoriamente, e depois dispara 4 espinhos. Tenho 4 ShotPoints que servem de ponto inicial e de direção de cada espinho. Assim, todos os espinhos sairão direitos dos orifícios.

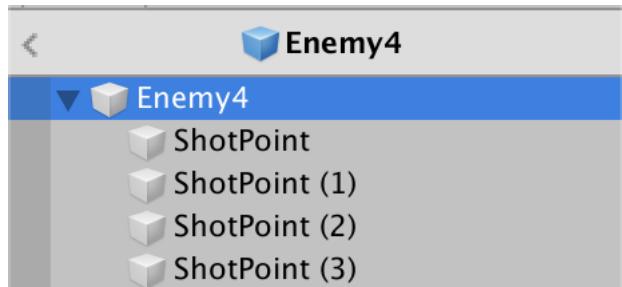


Figura 24 - Composição do Enemy4

```
void Shoot()
{
    foreach (Transform child in gameObject.transform)
    {
        if (child.transform.name != "HurtSound") {
            var newObj = Instantiate(shootPrefab, child.position, child.rotation);
            newObj.transform.parent = GameObject.Find("EnemiesAttacks").transform;
        }
    }
}
```

Figura 25 - Função de Lançamento dos espinhos

Aqui está representada a função do lançamento dos espinhos. Percorro todos os filhos do Enemy4, e verifico se não são o “HurtSound”. Se não, lançam o espinho.

O espinho é lançado, mas é inserido no objeto “EnemiesAttacks”. Fiz isto pois assim o jogo fica com uma organização maior do que andarem Clones fora do nível presente.

3.1.6.2 ENEMY2



Figura 26 - Enemy2

Este inimigo não ataca, o conceito dele é ir saltando do solo da terra. Aplico uma força vertical para ele saltar. É dos inimigos mais simples do jogo.

3.1.6.3 GHOST (ENEMY5)



Figura 27 – Ghost



Figura 28 – Spell

A cada 2.5 segundos, este inimigo lança 3 “Spells” a partir dos seus ShotPoints. Cada Spell leva uma direção diferente.

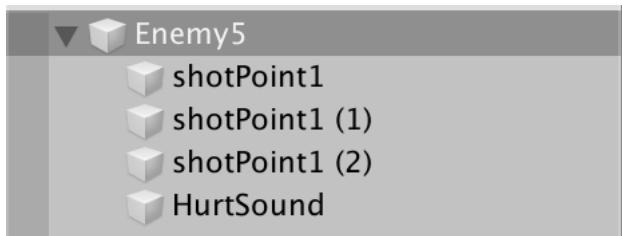


Figura 29 - Composição do Enemy5

3.1.6.4 SPERM (ENEMIE6)



Figura 30 – Sperm

Este inimigo não tem ataque. Ele segue o jogador, e quando existe colisão, morre depois de danificar o jogador.

Um dos problemas que não consegui resolvi foi o facto de não conseguir pôr o “Sperm” a olhar diretamente para o jogador. Tenho uma função que tenta fazer isso, mas malsucedida.

```
void LookAt()
{
    Vector3 targ = target.transform.position;
    targ.z = 0f;

    Vector3 objectPos = transform.position;
    targ.x = ... targ.x - objectPos.x;
    targ.y = ... targ.y - objectPos.y;

    float angle = Mathf.Atan2(targ.y, targ.x) * Mathf.Rad2Deg;
    transform.rotation = Quaternion.Euler(new Vector3(0, 0, angle));
}
```

Figura 31 - Função para olhar para o Jogador

```
void Move()
{
    speed = 2;
    float step = speed * Time.deltaTime;

    transform.position = Vector3.MoveTowards(transform.position, target.transform.position, step);
}
```

Figura 32 - Função para seguir a posição do Jogador

3.1.6.5 FINAL BOSS



Figura 33 - Final Boss

O Final Boss é o inimigo mais forte do jogo, onde possui dez vidas e três ataques:

- FireBall: Dispara 10 Fireballs seguidas;



Figura 34 – Fireball

- Geiser: Invoca 5 Geisers;



Figura 35 – Geiser

- Meteoro: Invoca 20 meteoros que caem do céu.



Figura 36 – Meteoro

Estes três ataques acontecem de 12 em 12 segundos, alternadamente. Para isto acontecer, criei uma Coroutine que faz a gestão dos ataques do Final Boss.

Claro que depois, cada prefab de cada Ataque (FireBall, Geiser e Meteoro) tem o seu próprio comportamento. Isto acontece ao longo de todo o jogo.

```

IEnumerator Attacks()
{
    yield return new WaitForSeconds(12f);
    StartCoroutine("Attack" + curAttack);
    if (curAttack == 1)
    {
        anim.SetTrigger("Attack1");
    }
    if (curAttack == 2)
    {
        anim.SetTrigger("Attack2");
    }
    if (curAttack == 3)
    {
        anim.SetTrigger("Attack3");
    }
    curAttack++;
}

StartCoroutine("Attacks");
}

```

Figura 37 - Gestão dos ataques do Final Boss

3.2 MÚSICA E SOM

3.2.1 MÚSICA

Como referi anteriormente, toda a música e sons foram criados por nós.

Para a criação de música, usei o Logic Pro X.

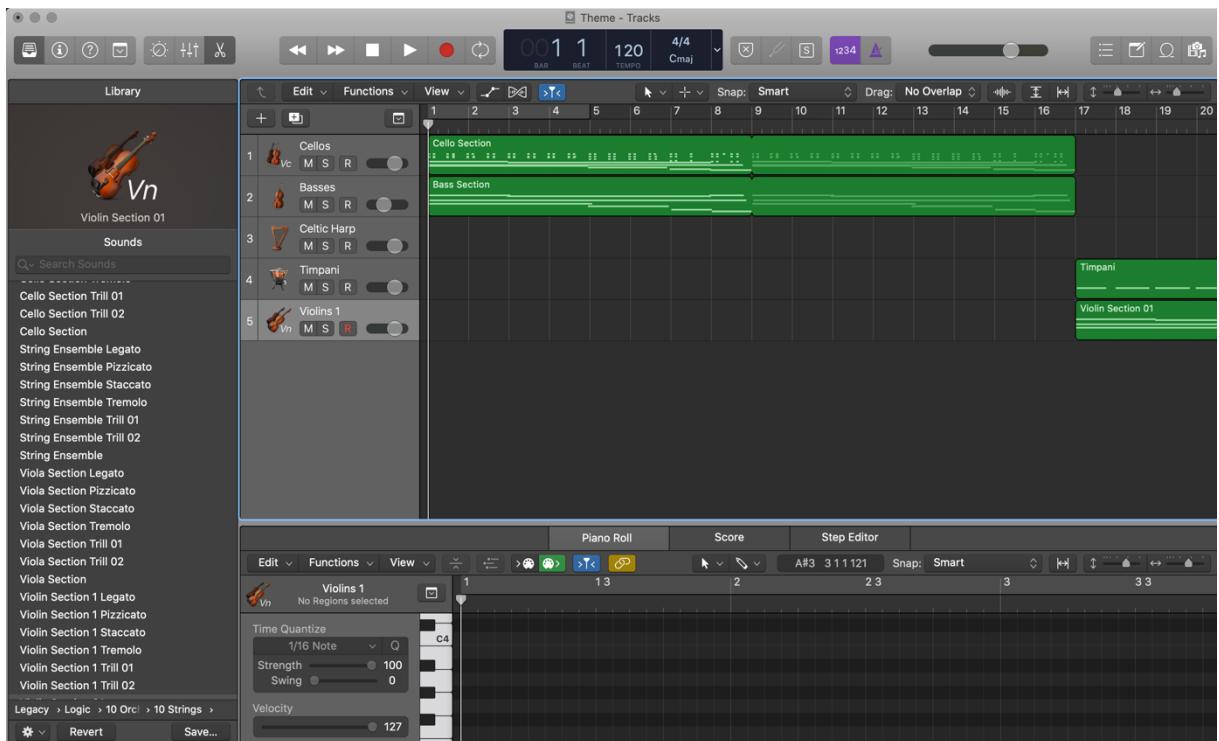


Figura 38 - Tema Inicial no Logic Pro X

Uma das minhas maiores referências foi o soundtrack do [Hollow Knight](#).

Assim, criei dois temas: o Theme, que é o tema do jogo (aparece no Menu e no 1º Nível); e o Dark Theme (que aparece no 2º Nível).

Cada theme tenta transmitir a sensação presente em cada nível como, por exemplo, o 2º Nível é mais sombrio, criei um tema mais triste.

3.2.1 SONS

Os sons foram gravados com o auxílio de uma rapariga de 14 anos. Decidimos assim pois teria a voz perfeita para representar a nossa personagem.

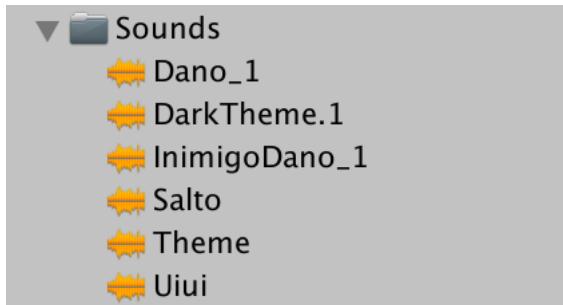


Figura 39 - Pasta Sons e Música

CONCLUSÃO

Ao longo deste projeto fui reparando que as minhas capacidades de C# aumentaram imenso, onde já não precisava de consultar o fórum de Unity. Isso ajudou imenso para quando precisava de código rápido e simples.

No início tínhamos decidido que o jogo ia ter determinadas funcionalidades. Claro que ao decorrer do desenvolvimento do projeto, tive que reduzir em alguns inimigos e algumas funcionalidades, mas isso não invalidou o resultado final, pois está um jogo conciso, fácil e perto do que tínhamos definido desde o início.

Estou contente com o resultado que eu e o Rodrigo conseguimos obter.

REFERÊNCIAS BIBLIOGRÁFICAS

Hollow Knight Soundtrack: <https://www.youtube.com/watch?v=0HbnqjGirEg>;

Unity Forum: <https://forum.unity.com/>;