INSTITUTO POLITÉCNICO DE VIANA DO CASTELO

ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

ENGENHARIA INFORMÁTICA

TECNOLOGIAS MULTIMÉDIA

2022/2023

Corre Guedes

Aluno:
Rafael André - 28234
Diogo Reis - 28239

Docente:
Beatriz Miranda

Instituto Politécnico
de Viana do Castelo

12 de junho de 2023

Conteúdo

Capítulo 1

Objetivos

1.1 Introdução

Este trabalho foi desenvolvido no âmbito da Unidade Curricular de Tecnologias Multimédia, onde o grupo decidiu criar um jogo do gênero *Endless Runner*, utilizando o Unity e a suas funcionalidades, e desenvolvendo o jogo em C#.

É possível acompanhar o processo de criação do jogo através do seguinte link do GitHub: link .

O GitHub é uma ferramenta que permite criar um repositório, e com a ajuda da aplicação desktop, é possível ir atualizando sempre o código, facilitando o trabalho em equipa, e evitando derivados envios de ficheiros entre o grupo.

Capítulo 2

Desenvolvimento do jogo

2.1 Manual de Instruções

No ecrã inicial aparece o nome do jogo, e para o iniciar basta clicar no botão **START**. Para mover para os lados e saltar utiliza-se as setas esquerda e direita, e a barra de espaço, respetivamente. Pressionar a tecla **ESC** pausa o jogo.

Por fim, quando colidir com um obstáculo, irá aparecer um ecrã de *Game Over* e basta pressionar o botão de recomeçar o jogo, carregando com o **botão esquerdo** do rato.

2.2 Configuração de ambientes

Para o desenvolvimento do nosso jogo trabalhamos á volta de dois cenas para o nosso jogo, uma para o Menu de início do jogo, e outra onde corre o resto do jogo.

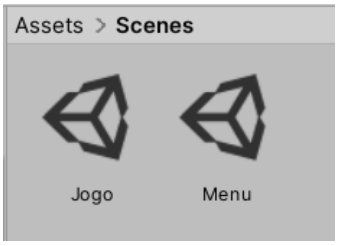

FIGURA 2.1: Scenes

2.3 Scripts

O nosso jogo apresenta os seguintes scripts:

FIGURA 2.2: Scripts

2.3.1 Game Manager

Nesta script é onde se verifica o estado de jogo, além de também incrementar pontuação conforme as moedas apanhadas.

```
public void IncrementScore()
{
    if (!jogoIniciado) return;  // Se o jogo não foi iniciado, não incremente a pontuação

    score++;
    scoreText.text = "MOEDAS: " + score;
    scoreText.color = new Color(1f, 1f, 1f);
    playerMovement.speed += (playerMovement.speedIncreasePerPoint/3);


}
```

FIGURA 2.3: Função para incrementar score

2.3.2 GroundSpawn

Para o nosso jogo ser um *Endless Runner*, é necessário que tenha-mos um plataforma infinita, para a criação dessa plataforma usamos então o sprite *GroundSpawn*.

Conforme a geração da plataforma irá gerar também moedas e obstáculos.

```
public void SpawnTile (bool spawnItems)
{

    GameObject temp = Instantiate(groundTile, nextSpawnPoint, Quaternion.identity);
    nextSpawnPoint = temp.transform.GetChild(1).transform.position;

    if (spawnItems) {
        temp.GetComponent<GroundTile>().SpawnObstacle();
        temp.GetComponent<GroundTile>().SpawnCoins();
    }
}

private void Start () {
    for (int i = 0; i < 15; i++) {
        if (i < 4) {
            SpawnTile(false);

        } else {
            SpawnTile(true);

        }

    }
}
```

FIGURA 2.4: Função para gerar o chão

2.3.3 GroundTile

Para controlar a geração de moedas e obstáculos, como uma parede alta que só pode ser evitada se o jogador se desviar, e uma parede baixa que pode ser evitada se o jogador pular ou se desviar, utilizamos o script chamado *GroundTile*.

```
public void SpawnObstacle ()
{
    //escolher qual dar spawn
    GameObject obstacleToSpawn = obstaclePrefab;
    float random = Random.Range(0f, 1f);

    if(random < 0.35){
        obstacleToSpawn = tallObsPrefab;
    }

    // Choose a random point to spawn the obstacle
    int obstacleSpawnIndex = Random.Range(2, 5);
    Transform spawnPoint = transform.GetChild(obstacleSpawnIndex).transform;

    // Spawn the obstace at the position
    Instantiate(obstacleToSpawn, spawnPoint.position, Quaternion.identity, transform);
}
```

FIGURA 2.5: Função para gerar as paredes

```
public void SpawnCoins ()
{
    float random = Random.Range(0f, 1f);
    if(random > 0.15f){
        int coinsToSpawn = 5;
        for (int i = 0; i < coinsToSpawn; i++) {
            float random2 = Random.Range(0f, 1f);
            if(random2 > 0.65f){
                GameObject temp = Instantiate(coinPrefab, transform);
                temp.transform.position = GetRandomPointInCollider(GetComponent<Collider>());
            }
        }
    }
}
```

FIGURA 2.6: Função para gerar as moedas

2.3.4 PlayerMovement

Também necessitamos de um player e que ele consiga realizar acções, para isso, nesta script, definimos as setas para os movimentos horizontais do player, e o espaço para o player consiguir pular.

```
private void FixedUpdate()
{
    if (!alive) return;

    Vector3 forwardMove = transform.forward * speed * Time.fixedDeltaTime;
    Vector3 horizontalMove = transform.right * horizontalInput * Time.fixedDeltaTime * horizontalMultiplier * 10;

    // Define os limites do movimento horizontal
    float minX = -4.5f;
    float maxX = 4.5f;
    float clampedX = Mathf.Clamp(rb.position.x + horizontalMove.x, minX, maxX); // Aplica o limite

    // Atualiza o movimento horizontal com o valor limitado
    horizontalMove = new Vector3(clampedX - rb.position.x, 0f, 0f);

    rb.MovePosition(rb.position + forwardMove + horizontalMove);
}
```

FIGURA 2.7: Função para definir os movimentos horizontais

```
void Jump()
{
    // Verificar se já estamos pulando
    if (isJumping) return;

    // Verificar se estamos no chão
    if (Mathf.Abs(rb.velocity.y) < 0.1f)
    {
        // Se estamos no chão, pular
        rb.AddForce(Vector3.up * jumpForce);
        isJumping = true;
    }
}
```

FIGURA 2.8: Função para fazer o player pular

Além disso também temos uma função que é chamada para o caso de o jogador perder.

```
public void Die()
{
    alive = false;
    gameObject.SetActive(false);
    gameManager.GameOver();
}
```

FIGURA 2.9: Função que acaba com o jogo

2.3.5 Obstacle

Nesta script definimos o tipo de interação que o player tem com as paredes, neste caso, sempre que o jogador colidir num a parede o jogador perde.

```
private void OnCollisionEnter (Collision collision)
{
    if (collision.gameObject.name == "Player") {
        // Kill the player

        AudioSource.PlayClipAtPoint(colisaoSound, transform.position);
        playerMovement.Die();
    }
}
```

FIGURA 2.10: Criação de gasolina

2.3.6 Coin

Tal como na script que abordamos anteriormente, esta script trata do tipo de interação que o player tem com as moedas, ou seja, sempre que o player encosta numa moeda, esta desapareçe e aumenta a contagem do score.

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.GetComponent<Obstacle>() != null)
    {
        Destroy(gameObject);
        return;
    }

    if (other.gameObject.name != "Player")
    {
        return;
    }

    // Reproduz o som da moeda quando há colisão com o jogador
    AudioSource.PlayClipAtPoint(coinSound, transform.position);

    GameManager.inst.IncrementScore();
    Destroy(gameObject);
}
```

FIGURA 2.11: Colisões

2.4 Prefabs

Usando as ferramentas do Unity criamos também estes 4 prefas, respetivamente, uma moeda, um quadrado para o chão, uma parede baixa e uma parede alta.

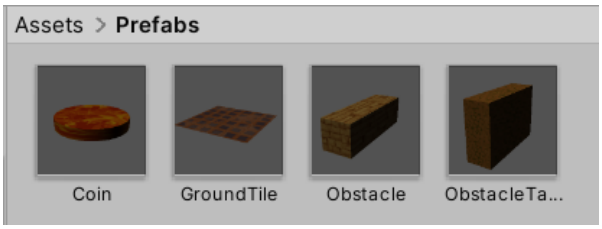

FIGURA 2.12: Colisões

2.5 Player

Para o nosso "personagem principal" utilizamos a assets store do Unity, e escolhemos um modelo de minotauro.

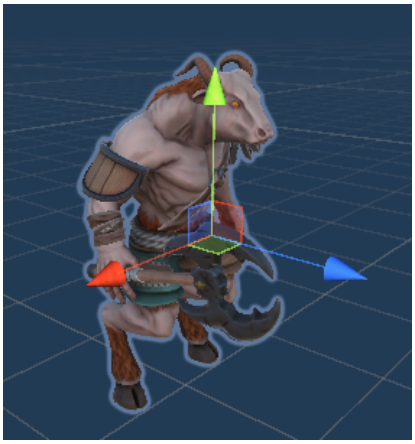

FIGURA 2.13: Modelo do minotauro

Capítulo 3

Dificuldades

Inicialmente houveram algumas dificuldades na utilização do git, pois o repositório que criavamos não adicinavamos ficheiro *.gitignore*. O grupo aproveitou e acompanhou um tutorial para fazer este jogo, coisa que ajudou no surgimento de alguma dificuldade. Além disso também tivemos problemas na parte de fazer *build* no jogo, e de o publicar no *itch.io*. .

De resto, o jogo fez se relativamente bem, foi-se trabalhando para que todos os pormenores estivessem alinhados e da maneira que o grupo desejava.

Capítulo 4

Conclusões

Concluindo este projeto / jogo, o grupo acredita que correspondeu com o esperado, e com os objetivos delimitados.

Deste modo foi possível obter um conhecimento mais profundo da aplicação do Unity, seja a gerenciar objetos ou modelos 3D, orientação de luz, câmera, animações e por ai em diante . Sendo C# a linguagem mais utilizada para o desenvolvimento deste jogo.