FORMANDO/A: TIAGO FILIPE SOUSA CARVALHO		
CURSO: GESTÃO E PROGRAMAÇÃO DE SISTEMAS INFORMÁTICOS	**TURMA:** 3ºGPSI	**TRIÉNIO: 2021/ 2024**
DESIGNAÇÃO DO PROJETO: DEATH ESCAPE		**DATA: 10/11/2023**

Figura 1 – Logo

Índice

Índice de Ilustrações

Introdução

Chamo-me Tiago Carvalho e frequento o curso de Gestão e Programação de Sistemas Informáticos na escola profissional de Valongo. A PAP (Prova de Aptidão Profissional) é uma avaliação importante e a final do curso profissional que pretende avaliar os conhecimentos obtidos pelo respetivo aluno ao longo do curso em forma de projeto final e realizarei a mesma no período de outubro de 2023 a março de 2024.

Eu criei um jogo na Unity, uma plataforma de criação de jogos que nunca usei, o jogo será uma demo com apenas um nível e esse nível 1 consiste num ambiente pós-apocalíptico com vários “fantasmas”, um caos sobrenatural que foi começado nos estados unidos por um culto que criou um ritual para que fantasmas surgissem do chão como plantas. O ritual acidentalmente corrompeu cada vez a terra dos campos e tornou-os num cemitério de onde os fantasmas apareciam frequentemente. Desesperados, o governo cercou completamente com muralhas a área afetada, deixando-a em quarentena, porem, com a pressa toda, eles não se importaram com quem ficava lá.

O jogador sendo uma das pessoas naquele local ficou preso e agora é o único sobrevivente, o objetivo será sair com vida da cidade. O jogador terá armas e irá destruir as hostilidades que o mesmo encontrará pelo mapa, haverá dois fantasmas chefes que terão muita mais vida que os outros e darão uma chave enferrujada ao jogador após a sua derrota, essas chaves serão usadas para puder passar por um suposto misterioso portão grande e negro que pelo meio do espesso nevoeiro foi avistado. O jogo acaba quando o jogador for derrotado ou conseguir sair com vida da cidade.

O nome que darei ao jogo será “Death Escape, (em português: Fuga da morte)” dando adrenalina, suspense e um pouco de receio ou medo que o jogo pretende obter dos jogadores.

A ideia do projeto surgiu pelo meu gosto de jogos de sobrevivência. Não será fácil pelo facto de que nunca trabalhei com Unity, terei que aprender tudo do zero, mas os meus objetivos são fazer um jogo simples de entender e desafiante tendo fases cada vez mais difíceis para despertar competitividade no jogador. O jogo pretende causar diversão, stress e adrenalina no jogador como efeitos principais, por isso tentarei tornar algo bastante ativo e stressante para que o jogador fique stressado e ansioso, mas ao mesmo tempo divertido.

Mês de Outubro

Neste mês não fiz muito de especial, instalei a unity e dediquei-me a estudar sobre a unity e a criar ideias de como faria o jogo e o que poderia adicionar mais.

Mês de Novembro

1ª Semana:

Mapa de teste

Neste mês nesta primeira semana comecei a trabalhar, fiz um pequeno mapa para começo do ambiente. Comecei por colocar alguns prédios neste momento com apenas algumas texturas simples.

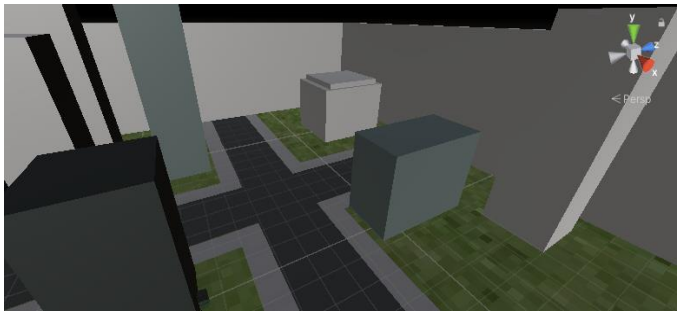


Figura 2 - Mapa versão 1

2ª Semana:

Adicionar Assets

Na segunda semana comecei a adicionar alguns assets para me ajudar com a criação do jogo para ter tempo para acabar o projeto pois preciso de aprender e fazer ao mesmo tempo.

Figura 3 - primeiro teste de movimento

Por exemplo o “Modular First Person Controller”. Para instalá-lo e pôr em ação o mesmo fazemos o seguinte:

Na asset store da unity, baixamos o asset e importamos para a unity.

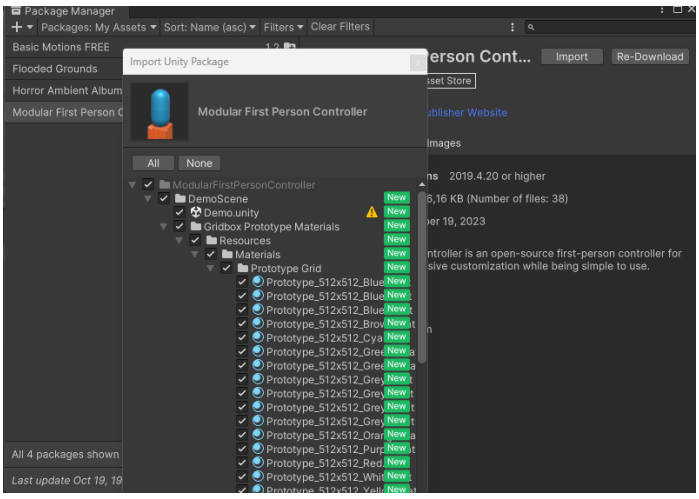


Figura 4 - importar asset

3ª Semana:

Colocar Asset de personagem no Jogo

Esta semana comecei a pôr os assets em ação.

Por exemplo para pôr o asset acima descrito vamos na pasta dele e procuramos pelo script dele.

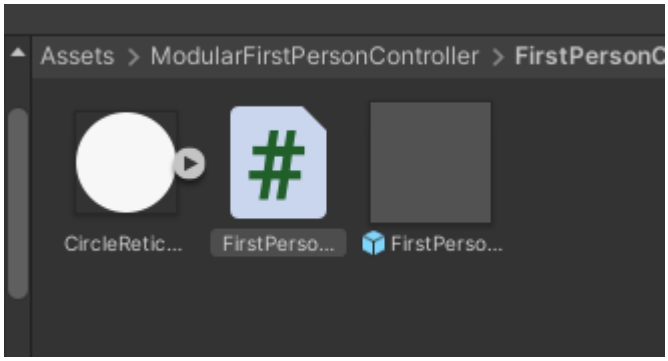


Figura 5 - Script do movimento teste

Depois disso arrastamos o script para o jogo.

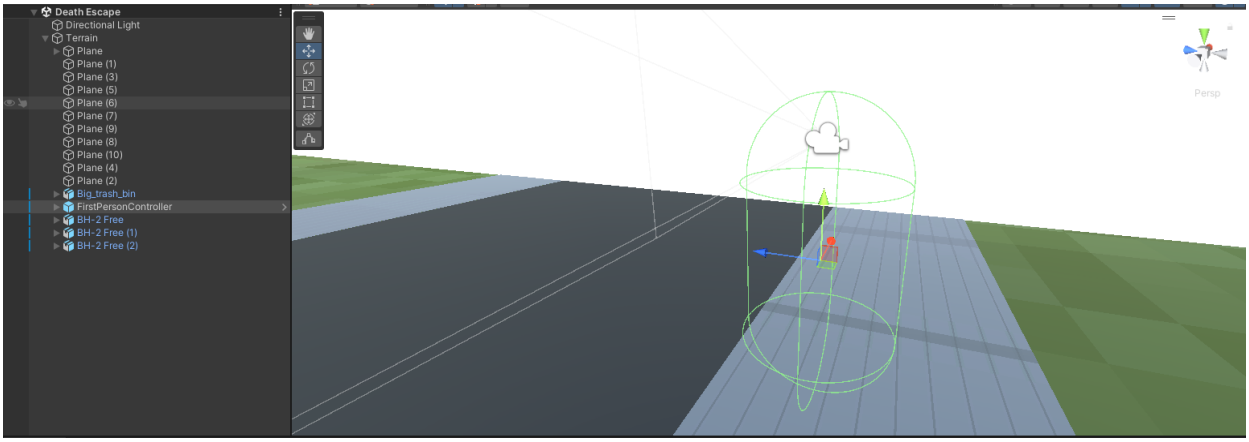


Figura 6 - primeiro teste em cena

Modificação de script

Depois de adicionar o script, modifiquei-o aos meus gostos. Para isso fui no “Inspector”.

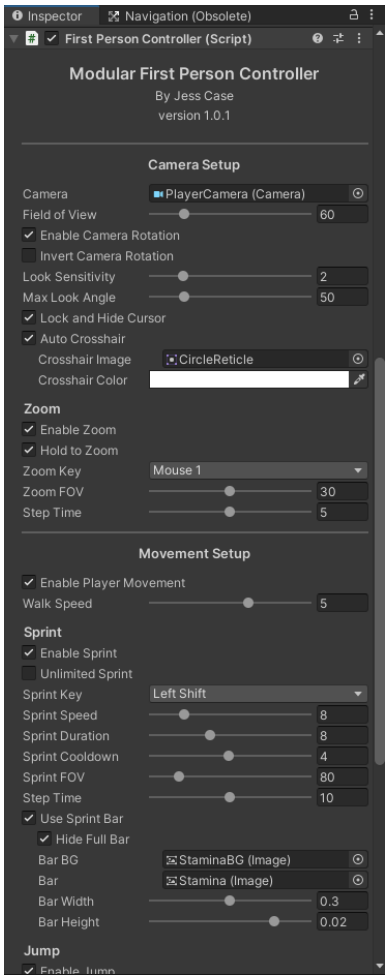


Figura 7 - Modificação do 1º asset teste

Adicionamento de componentes ao personagem para colisão e interação

Depois disso voltei no “Inspector” e cliquei em “Add Component” e adicionei o “Rigidbody” para o corpo do player ter massa e puder interagir com o ambiente e “Capsule Collider” para o player ter colisão e não passar por objetos ou paredes.

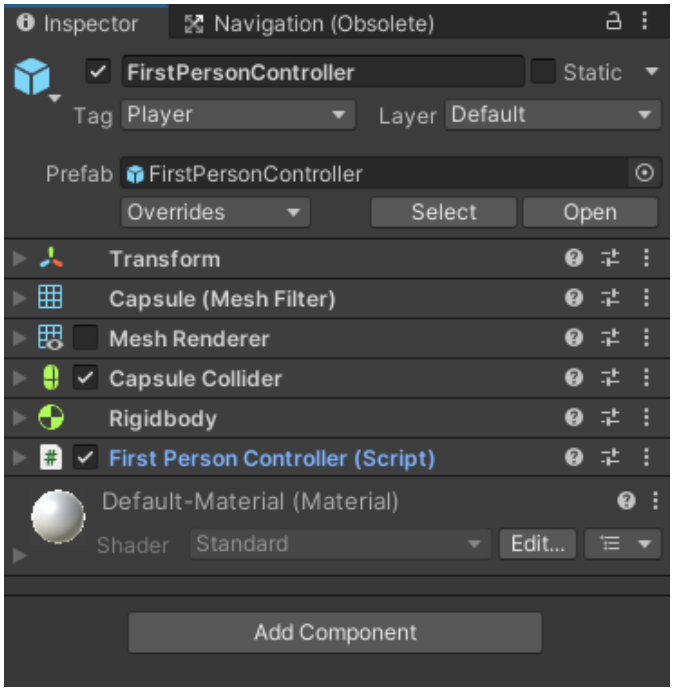


Figura 8 - Componentes do player

4ª Semana:

Adicionamento de um asset do corpo e animação de teste de inimigo

Agora adicionei alguns inimigos de testes com um asset baixado para o corpo e animação deles.

Adicionei também um Rigidbody e capsule Collider para os mesmos efeitos do player.

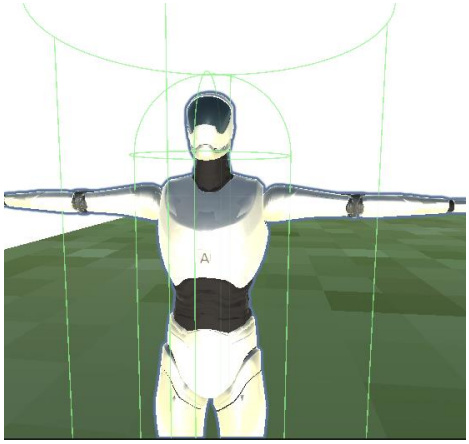


Figura 9 - Inimigo de teste

Adicionamento de animação

Adicionei o componente “Animator” e adicionei a animação e o avatar que seria usado para tal.

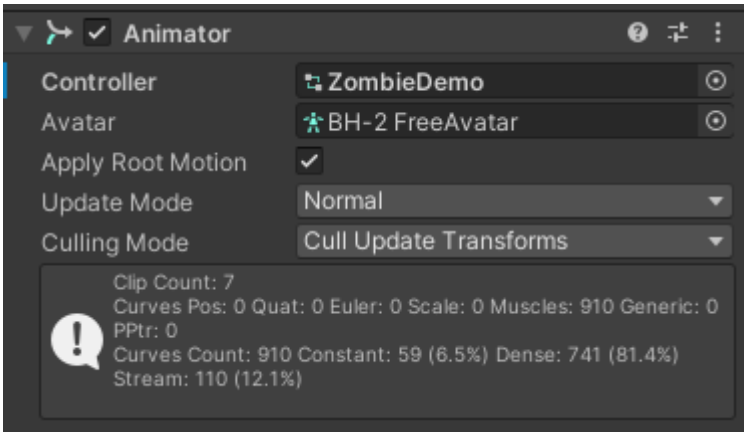


Figura 10 - Animação do inimigo de teste

Adição de script para o inimigo seguir o jogador

Agora preciso que os inimigos venham atras de mim e para isso coloquei um script simples no inimigo chamado “Enemyfollow”.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.AI; // adicionar AI para o movimento do objeto.

public class enemyfollow : MonoBehaviour
{
    // adicionar locais para colocar qual o objeto player e qual o inimigo
    public NavMeshAgent enemy;
    public Transform player;

    void Start()
    {

    }


    void Update()
    {
        enemy.SetDestination(player.position); // Colocar o destino do inimigo em direção à minha posição
    }
}
```

Figura 11 - Script para inimigo seguir o player

Depois disso identifiquei o player e o inimigo.

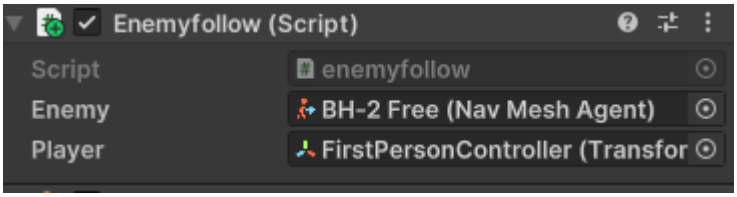


Figura 12 - Inspector do script Enemyfollow

Depois disse basta ir em window > AI > Navigation (Obsolete).

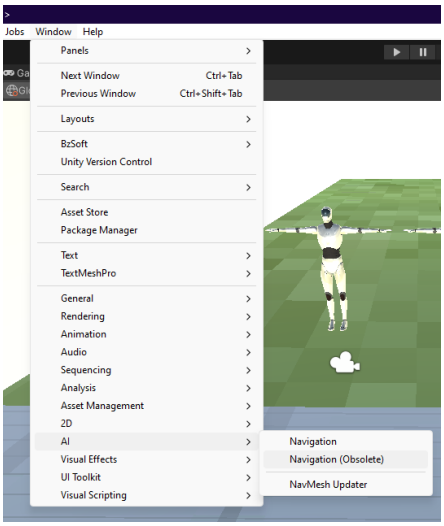


Figura 13 - Tutorial para a Navigation

Em seguida vamos em Navigation (Obsolete) e clicamos em Bake e mais uma vez em baixo em Bake para criar a plataforma de navegação do objeto. Caso contrário o inimigo movia-se sem orientação e colava-se ao player.

Depois disse colocamos o componente "NavMeshAgent" para a modificação da velocidade e etc…

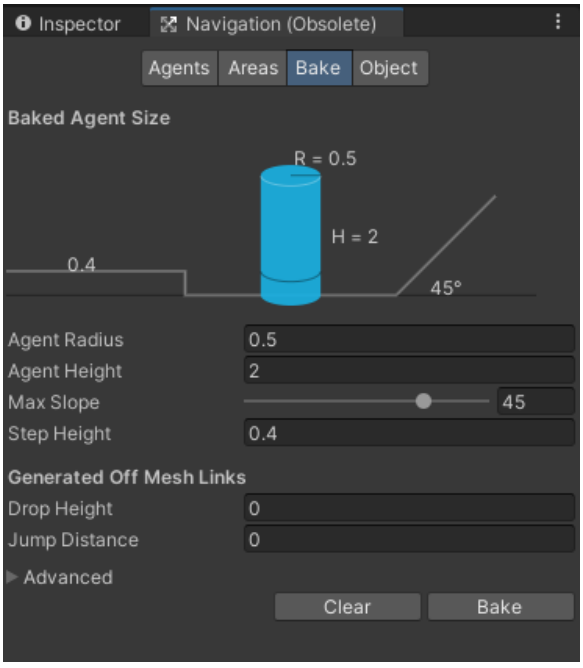


Figura 14 -Bake no Navigation

Figura 15 - NavMeshAgent

Mês de Dezembro

1ª Semana:

Mapa novo

Comecei a modelar um mapa novo mais bonito e detalhado. Comecei pelo relvado criando uma textura com uma imagem de relva da internet. Para o fazer: Baixei a imagem da internet, na unity fui em “all materials” e cliquei com o botão direito, cliquei em “create” e depois “material”.

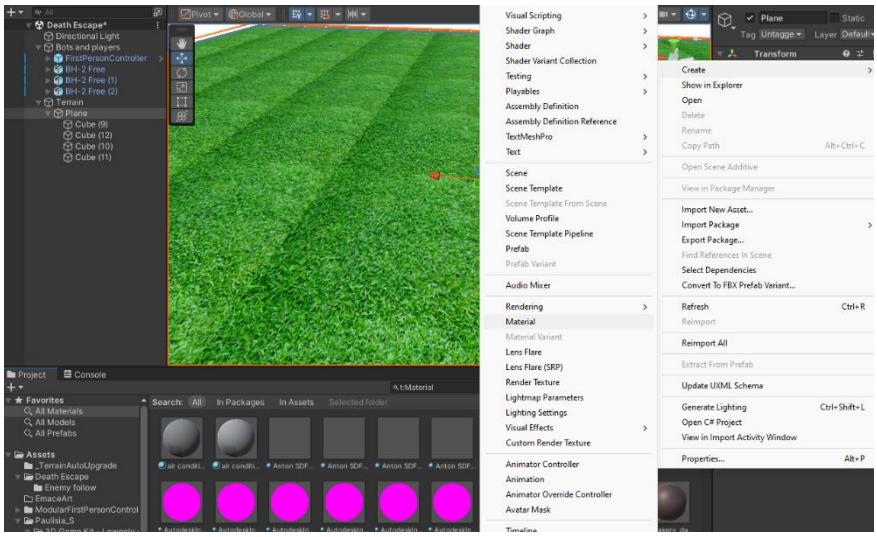


Figura 16 - Como adicionar Material 1

Depois disso nomeei o material para “grass”. Depois no inspector do material fui no canto superior direito em shader e selecionei “Legacy Shaders” e depois “diffuse”.

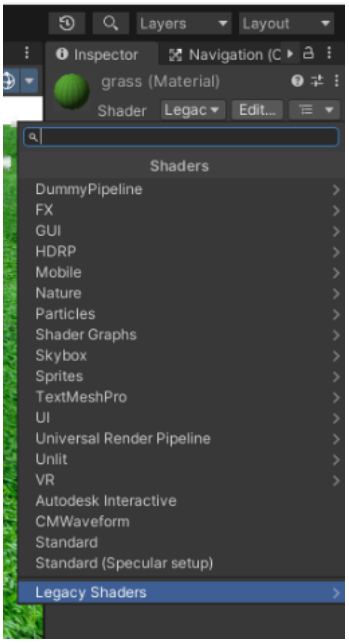


Figura 17 - Shaders Material

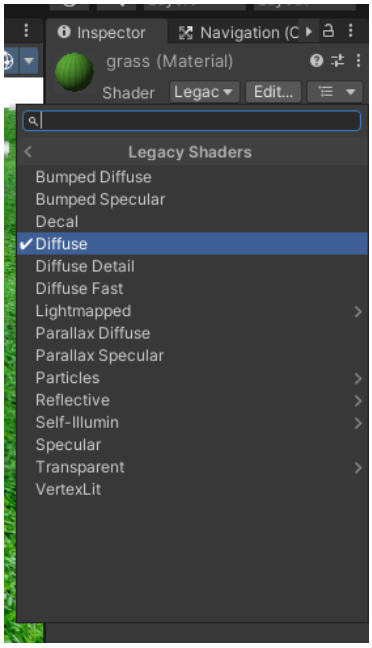


Figura 18 - Shader Diffuse

Depois disso adicionamos a imagem na unity.

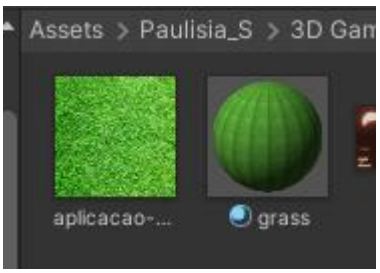


Figura 19 - Imagem do material relva

Em seguida, levamos a imagem até ao canto superior direito e largamos no quadrado ao lado de base. Mudei também o Tiling pois se colocasse a textura no plano grande a relva se expanderia e desfocaria toda, então fiz apenas com que se repetisse em quadrados. Por fim é só puxar a textura e largarmos para o plano que queremos colocar.

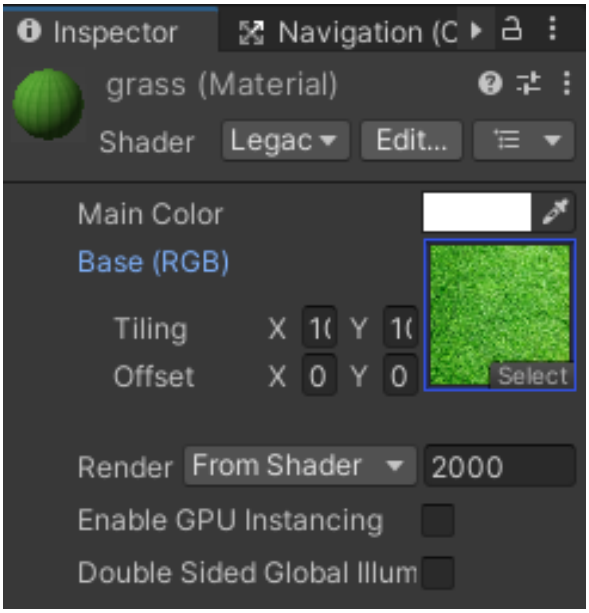


Figura 20 - Material relva

2ª Semana:

Continuação do mapa

Depois da textura de relva eu fui adicionando estradas, prédios, carros e outros objetos de um asset baixado ao mapa para o tornar mais urbano.

Figura 21 - Novo mapa em Construção

Mês de Janeiro

1ª Semana:

Detalhamento do mapa

Nesta semana tenho o mapa quase terminado adicionei mais detalhes urbanos bem como mais detalhes para parecer mais devastado.

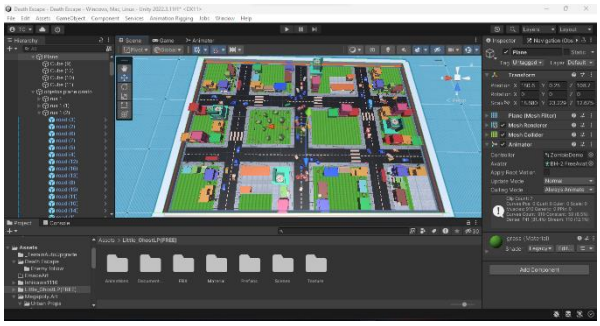


Figura 23 - Continuação do mapa

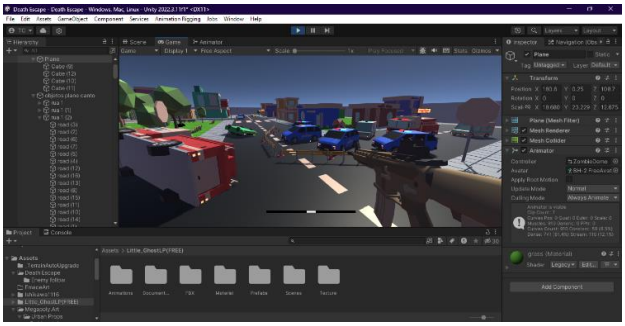


Figura 22 - Detalhes no Mapa

2ª Semana:

Mapa Completo

Completei o mapa em geral, melhorarei futuramente. Vou começar a implementar armas e o resto da lógica de jogo FPS.

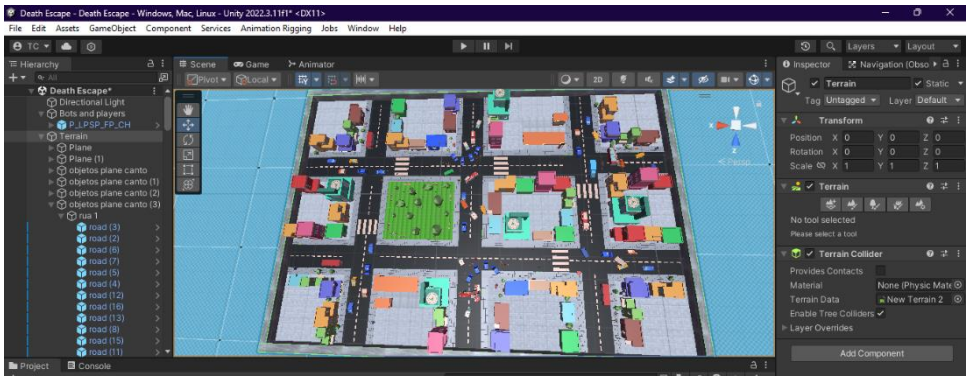


Figura 24 - Mapa versão final

3ª Semana:

Lógica de jogo FPS

Fiz download de algumas armas, braços, animações, scripts dos mesmos e coloquei-os em ação. Temos o script principal do personagem, o script do personagem kinematics (coisas mais especificas do principal), o script de movimento e do inventário. Troquei o script de movimento pelo novo e todo o personagem foi modificado para usar estes scripts novos.

4ª Semana:

Função de Salto

Esta semana adicionei a função de saltar ao player. Para isso fui no script de “Movement”, vim na função “Update()” e adicionei “Jump()” para chamar a função “Jump()”. Depois em baixo da função criei então a função Jump. Coloquei um if, se clica-se no espaço e o player estivesse no chão (na primeira tentativa o player saltava infinitamente, portanto, adicionei que só saltaria caso tambem estivesse no chão) adicionaria força no rigidBody para ir para cima a multiplicar por 5 para ter um salto moderado.

```
void Jump()
{
    if (Input.GetKeyDown(KeyCode.Space) && grounded)
    {

        rigidBody.velocity = transform.up * 5;
    }
}
```

Figura 25 - Função Salto

Após isso vim na função “MoveCharacter()” e adicionei a velocity do rigidBody o novo vector3 com as cordenadas x,y e z, sendo Y altura então em vez de moviment coloquei a velocity do rigidBody.

```
//atualizar velocity
rigidBody.velocity = new Vector3(movement.x, rigidBody.velocity.y, movement.z);
```

Figura 26 - Atualizar Velocity

Problemas**:**

O salto estava infinito então coloquei para o player só saltar caso esteja no chão.

Modifiquei o código da velocity pois saltava, mas não se movia horizontalmente. Coloquei em vez de “Movement.y”, “rigidBody.velocity.y” para especificar o movimento vertical do Rigidbody com a velocity.

Criação de Barra de Energia

Agora vou adicionar uma Stamina Bar ao player para fazê-lo parar de correr certo tempo quando a energia se gastar. Para isso adicionei ao player “canvas”, com o botão direito em cima do objeto do player > UI > Canvas Apos isso vou em cima da vista do projeto e clico “2D” para ir para a vista 2d do projeto e trabalhar na interface.

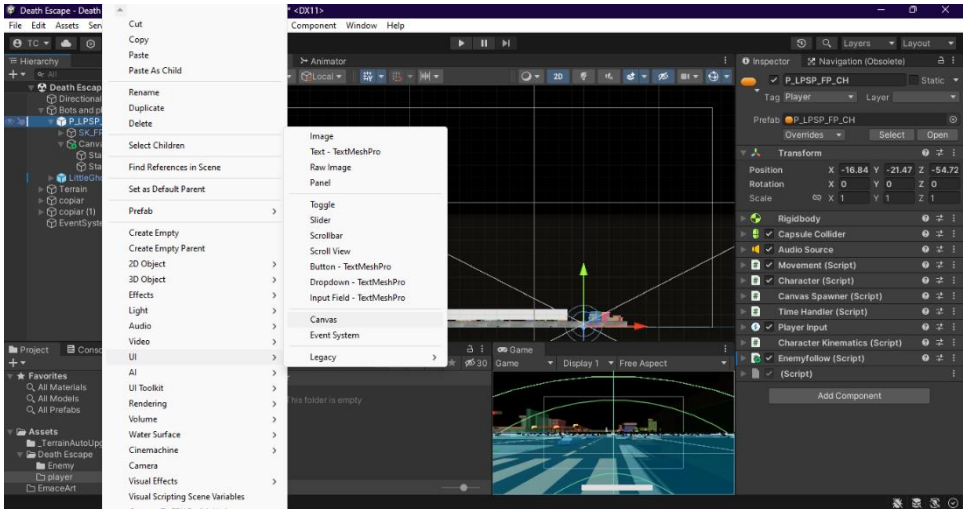


Figura 27 - Criação UI

Após isso adicionei nos canvas uma “image” pelo mesmo caminho: botão direito em canvas > UI > image. A seguir fui buscar imagens de barras para colocar na minha, como esta aqui, mas como tinha cor tive que retirar o fundo da cor. Adicionei mais uma image para ser a stamina em si que descerá conforme usada.

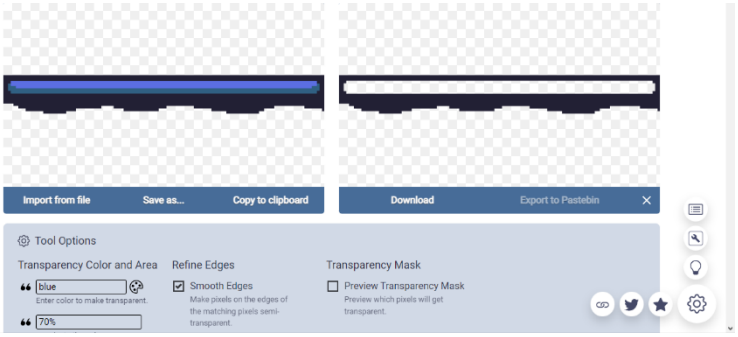


Figura 28 - Retiramento de fundo da barra

Depois disso, adicionamos a imagem no projeto, vamos no inspector dela e mudamos a texture type para Sprite

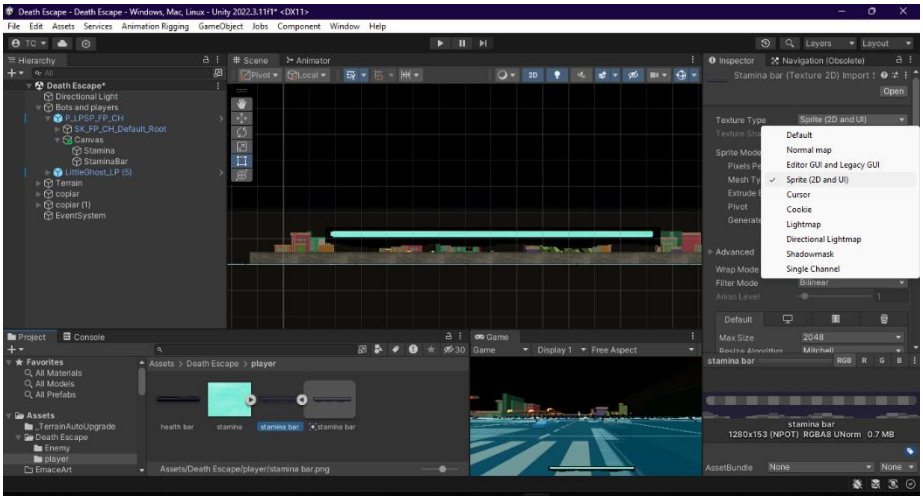


Figura 29 - Imagem em Sprite

Após isso é so arrastar a imagem para o campo “source image” no componente image, no meu caso,no componente image da stamina já que estou adicionar na print a imagem com a cor dela(o mesmo processo é feito para a image da barra).

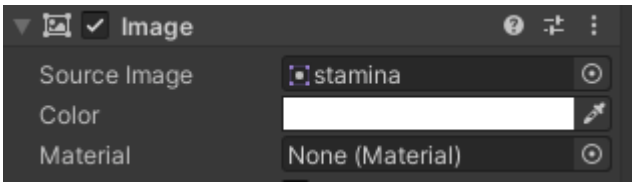


Figura 30 - Colocação da imagem da barra

Agora coloco no canvas 2d da interface do player no local que quiser um em cima do outro de modo à stamina com aquela cor ficar no buraco que tornei png na imagem da barra.

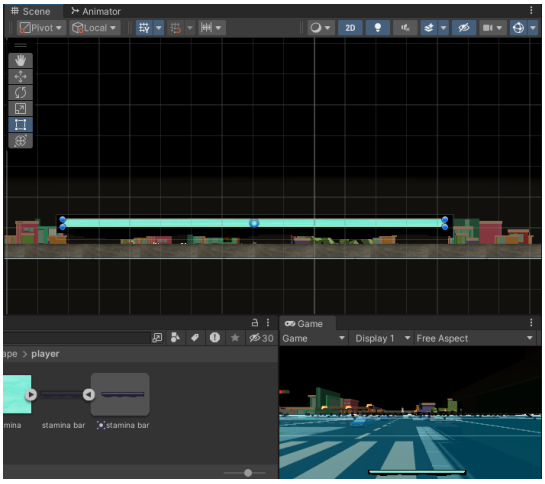


Figura 31 - Colocação de cor na barra

Temos que colocar a image da stamina como tipo “Filled” e método como “Horizontal” para poder modificar a quantidade de preenchimento na barra.

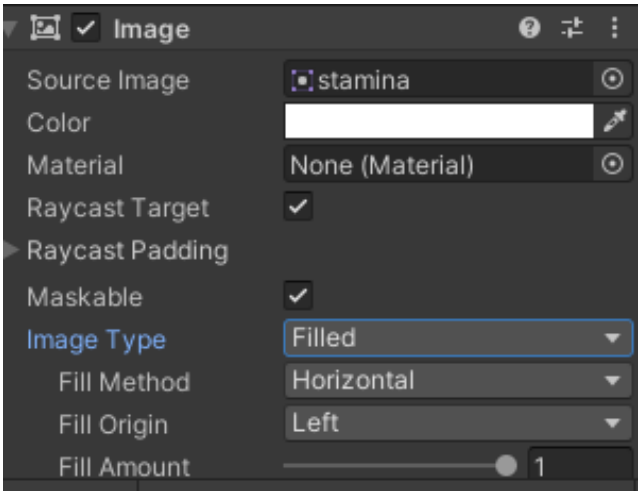


Figura 32 - Tipo de imagem Filled

Depois disso, vamos para o código da barra para que o personagem corra e gaste energia. Começo por definir as variáveis, no caso: a stamina máxima, a quantidade de stamina no momento, a velocidade que a stamina desce e a velocidade que ela regenera.

```
public class StaminaBar : MonoBehaviour
{
    public float maxStamina = 100f;
    public float currentStamina;
    public float staminaDecreaseSpeed = 5f;
    public float staminaRegenSpeed = 2f;
```

Figura 33 - Variáveis da barra de stamina

Em start coloco apenas que a stamina do momento é a máxima para começar com a stamina no máximo. E coloco uma referência ao script de movimento para poder ir buscar variáveis do mesmo.

```
void Start()
{
    currentStamina = maxStamina;

    // Obtém uma referência ao script de movimento do jogador
    playerMovement = FindObjectOfType<Movement>();
}
```

Figura 34 - Função start da stamina

No Update coloco a função de atualizar a stamina e em baixo crio essa mesma função, em que coloco a variável “isRunning” que é igual ao movimento do player se não for nulo e o player estiver a correr e se estiver a correr, ativo a função de diminuir stamina, caso contrário ativo a função de regenerar stamina. No final ativo a função de atualizar a stamina visualmente.

```
void Update()
{
    UpdateStamina();
}

void UpdateStamina()
{
    bool isRunning = playerMovement != null && playerMovement.IsRunning();

    if (isRunning)
    {
        // Diminui a stamina enquanto o jogador está a correr
        DecreaseStamina(staminaDecreaseSpeed);
    }
    else
    {
        // Regenera a stamina quando o jogador não está a correr
        RegenerateStamina(staminaRegenSpeed);
    }

    // Atualiza a exibição da barra de stamina
    UpdateStaminaUI();
}
```

Figura 35 - Funções de atualizar stamina

Aqui temos as funções de diminuir e regenerar a stamina. Na função de diminuir retiramos stamina da quantidade de stamina do momento e prevenimos que a stamina não desça abaixo de zero. Na função de aumentar o mesmo, mas ao contrário e impeço que a stamina ultrapasse o máximo.

```
void DecreaseStamina(float amount)
{
    currentStamina -= amount * Time.deltaTime;

    // Impede que a stamina fique abaixo de zero
    currentStamina = Mathf.Max(currentStamina, 0f);
}

void RegenerateStamina(float amount)
{
    currentStamina += amount * Time.deltaTime;

    // Impede que a stamina ultrapasse o máximo
    currentStamina = Mathf.Min(currentStamina, maxStamina);
}
```

Figura 36 - Funções de lógica da stamina

Mês de Fevereiro

1ª Semana:

Skybox

Esta semana, comecei por adicionar uma skybox para ter um céu mais bonito. Para isso baixei uma imagem, adicionei ao projeto e como fiz com a relva criei um material porem, no Shader eu escolho skybox > cubemap. Depois disso adiciono a imagem como fiz na relva e arrasto para o céu do projeto.

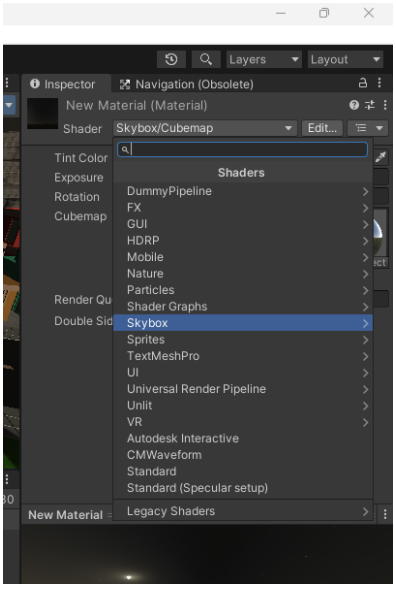


Figura 38 - Shaders do material

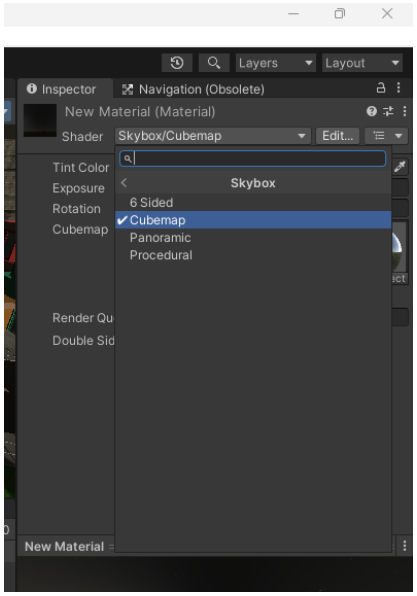


Figura 37 - Shader Cubemap

Barra de Vida

Comecei a fazer a barra de vida parecida à de stamina. Fiz exatamente o mesmo processo para a colocar visualmente e em código fiz quase o mesmo também, apenas com algumas mudanças de logica, como é obvio.

Aqui começo por colocar variaveis, vida maxima, vida atual, e o dano que o inimigo dará por segundo caso toque no jogador, bem como a referença à imagem que coloquei de vida.

```
public class HealthBar : MonoBehaviour
{
    public float maxHealth = 100f;
    public float currentHealth;
    public float damagePerSecond = 10f; // Quantidade de dano por segundo ao ser tocado por um inimigo

    public UnityEngine.UI.Image healthFill;
```

Figura 39 - Variáveis da barra de vida

No start, coloco que a vida atual corresponde à vida máxima para que o jogador comece com a vida cheia. No Update, coloco um “If” que se a função “isTouchingEnemy” for chamada ou seja se o player estiver a ser tocado pelo inimigo, chama a função de levar dano multiplicado a variável de dano por segundo para ir retirando vida ao longo do tempo. Por fim chamo a função de atualizar a barra de vida visualmente.

```
void Start()
{
    currentHealth = maxHealth;
}

void Update()
{
    // Diminui a vida se estiver a ser tocado por um inimigo
    if (IsTouchingEnemy())
    {
        TakeDamage(damagePerSecond * Time.deltaTime);
    }

    // Atualiza a barra de vida
    UpdateHealthUI();
}
```

Figura 40 - Lógica para a barra de vida

Depois fiz a função de levar dano, começo por colocar um código para me certificar que a vida não passa abaixo de zero. Crio a função para verificar se o inimigo e o player se tocam usando os “colliders” dos dois e coloco um If para verificar se o objeto que tocou o player é da Tag “enemy” que será uma Tag que irei criar para identificar os objetos que sejam inimigos ou não para levar dano apenas dos inimigos e não outros objetos que me toquem.

```
void TakeDamage(float amount)
{
    currentHealth -= amount;

    // Garante que a vida não caia abaixo de zero
    currentHealth = Mathf.Max(currentHealth, 0f);
}

bool IsTouchingEnemy()
{
    // Verifica se o jogador está a ser tocado por um inimigo
    Collider[] colliders = Physics.OverlapCapsule(transform.position, transform.position + Vector3.up * 2.0f, 0.5f);
    foreach (Collider collider in colliders)
    {
        if (collider.gameObject.CompareTag("Enemy"))
        {
            // Se estiver a tocar num inimigo
            return true;
        }
    }

    return false;
}
```

Figura 41 - Verificação de dano e toque

Por fim colocamos a função de atualizar a barra de vida visualmente verificando se a vida não é null e fazendo a proporção da imagem de vida descer horizontalmente para a esquerda.

```
void UpdateHealthUI()
{
    // Atualiza a barra de vida visualmente
    if (healthFill != null)
    {
        float fillAmount = currentHealth / maxHealth;
        healthFill.fillAmount = fillAmount;
    }
}
```

Figura 42 - Atualização visual de vida

2ª Semana:

Matar Inimigos

Agora comecei a colocar os inimigos a desaparecer disparando tiros neles. Para isso temos um script chamado TargetScript do conjunto que baixei anteriormente.

Começando pelo importante, adiciono duas variáveis: Hits, HitsToDestroy. Sendo Hits para contar os hits que são dados nos inimigos, ou seja, os tiros e HitsToDestroy a "vida" dos inimigos, mais especificamente os hits necessários para os derrotar.

```
public int hits = 0;
public int hitsToDestroy = 3;
```

Figura 43 -Variáveis de Hits

Depois na função "start()" adiciono uma referência ao componente do meu script para o inimigo seguir o player. Para poder segui-lo caso ele dispare contra o inimigo.

```
private void Start()
{
    // Obtém a referência ao componente enemyfollow
    enemyfollow = GetComponent<enemyfollow>();
}
```

Figura 44 - Função start de matar inimigos

Após isso na função "Update()" coloco se o número de tiros disparado for maior ou igual que o número necessário para derrotar o inimigo faço "Destroy(gameObject)" para que o inimigo desapareça. Depois coloco caso o inimigo levar um hit verifico se o script Enemyfollow não é nulo e faço o inimigo ter o destino da posição do player para o seguir.

```
//se levar hits suficientes para ser derrotado
if (hits >= hitsToDestroy)
{
    Destroy(gameObject);
}

// If the target is hit
if (hits > 0)
{
    // Verifica se enemyfollow não é nulo antes de acessá-lo
    if (enemyfollow != null)
    {
        // Define a posição de destino do inimigo para seguir o jogador
        enemyfollow.enemy.SetDestination(enemyfollow.player.position);
    }
```

Figura 45 - Verificação de matar ou acertar

Por fim, coloco também a função de levar Hit como é obvio. Aumento o contador de hits e ser for os hits necessários para derrotar o inimigo volto a chamar o método de destruir o objeto.

```
// hit no alvo
public void TakeHit()
{
    hits++; // aumenta o contador de hits

    if (hits >= hitsToDestroy)
    {

        Destroy(gameObject);
    }
}
```

Figura 46 - Função TakeHit

3ª Semana:

Adição de inimigos à cena

Agora adicionei vários inimigos espalhados pelo mapa, uns com mais vida que outros. Criei também secções difirentes para cada tipo de dificuldade dos inimigos: Easy, medium e Hard. Cada um adicionei o script enemyfollow que também atualizei para andarem aleatoriamente pelo mapa e apenas me seguirem quando estiver no campo de visão deles bem como fazer me seguirem caso dispare neles.

Figura 47 - Adição de inimigos à cena

Atualização de script enemyfollow

Adicionei estas 4 variaveis novas:

```
private Vector3 randomDestination; // Destino aleatório para os inimigos andarem
private float timeToChangeDestination = 5f; // Tempo para mudar o destino aleatório
private float visionRadius = 10f; // Raio de visão dos inimigos
public bool isChasingPlayer = false; // Indica se o inimigo está perseguir o jogador
```

Figura 48 - Variáveis novas do Enemyfollow

Na função de Start, chamo a função para atualizar a localização aleatória dos inimigos. Na função Update, verifico se a distancia do player é menor ou igual ao raio de visão do inimigo, se for, a variável que indica que o inimigo está a perseguir o jogador é ativada, caso contrário, desativada. Se estiver ativada o inimigo vai para a localização do jogador.

```
void Update()
{
    if (player != null)
    {
        float distanceToPlayer = Vector3.Distance(transform.position, player.position);
        if (distanceToPlayer <= visionRadius)
        {
            isChasingPlayer = true;
        }
        else
        {
            isChasingPlayer = false;
        }

        if (isChasingPlayer)
        {
            enemy.SetDestination(player.position);
        }
    }
}
```

Figura 49 -Função Update do Enemyfollow

Por fim temos a função de atualizar o destino aleatório e a localização aleatória do inimigo. Em que aviso caso o jogador se afaste o suficiente do raio de visão do inimigo, o inimigo retoma a uma posição aleatória e para de seguir o jogador.

```
IEnumerator UpdateRandomDestination()
{
    while (true)
    {
        if (!isChasingPlayer)
        {
            // Calcula um novo destino aleatório dentro de um raio em torno do inimigo
            randomDestination = RandomNavmeshLocation(transform.position, 10f);
            enemy.SetDestination(randomDestination);
        }
        yield return new WaitForSeconds(timeToChangeDestination);
    }
}

// Método para encontrar um ponto aleatório no NavMesh
Vector3 RandomNavmeshLocation(Vector3 origin, float radius)
{
    Vector3 randomDirection = UnityEngine.Random.insideUnitSphere * radius;
    randomDirection += origin;
    NavMeshHit hit;
    NavMesh.SamplePosition(randomDirection, out hit, radius, NavMesh.AllAreas);
    return hit.position;
}
```

Figura 50 - Atualizar localização aleatória

Problemas:

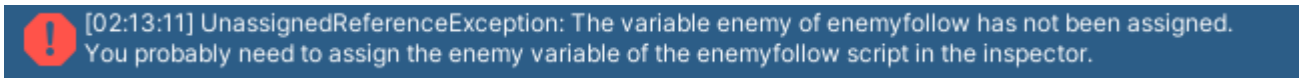


Figura 51 - Problema de variável

Este é um problema muito simples, está simplesmente a dizer que não assinalei nenhum objeto à variável enemy do script Enemyfollow e para corrigir isso basta ir ao objeto onde esta adicionado o script e assinalar o objeto a variável enemy.

4ª Semana:

Interfaces

Agora vou criar interfaces do jogador, nomeadamente, logo, menu incial, menu de pausa e ecrã de morte do jogador. Começando pelo logo, precisamos criar uma scene nova para isso vamos no local que queremos, botão direito > create > scene e nomeamos como quisermos. Após isso, vamos no canto superior esquerdo > file > build settings e clicamos “add Open Scenes” para adicionar scenes abertas na lista e por fim colocar a logo que será a primeira coisa a aparecer em primeiro lugar, no lugar 0.

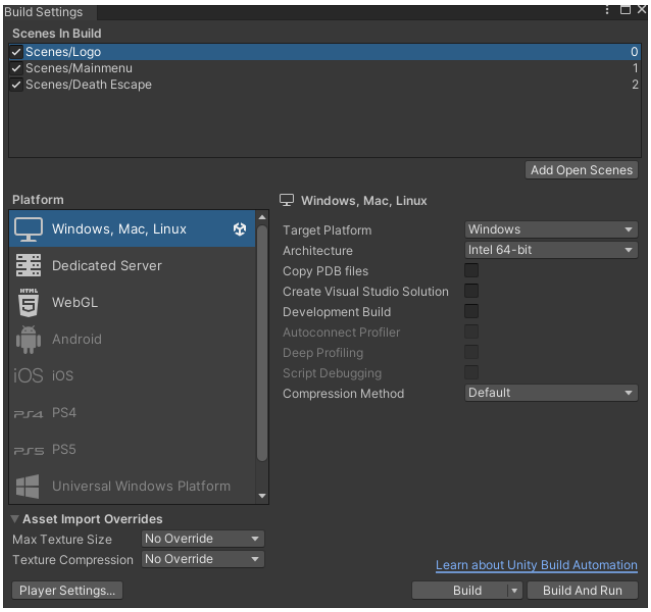


Figura 52 - Build das cenas

Main Menu

Depois da scene criada, criamos um gameObject para adicionar nele as canvas para entrarmos no 2D.

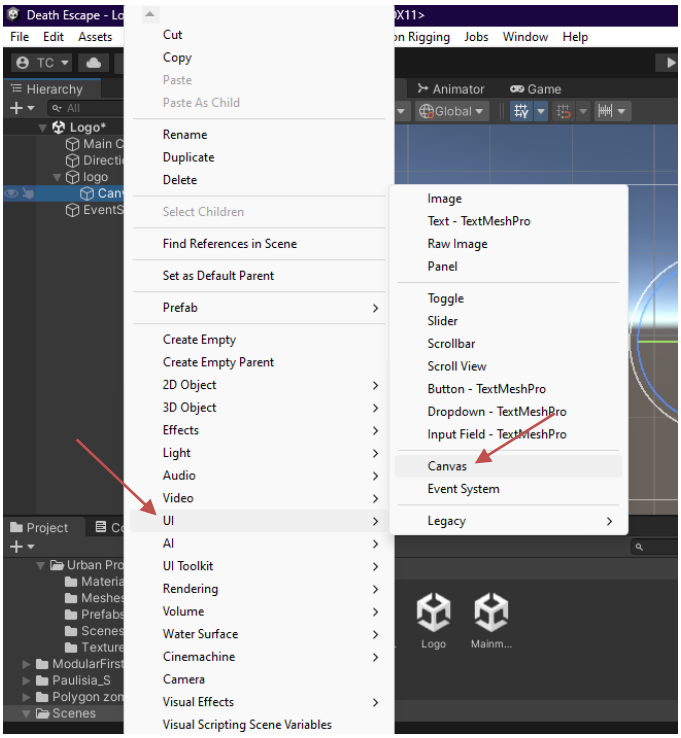


Figura 53 - Adição de canvas

Agora adicionamos a imagem da logo no projeto e tornamos a imagem em sprite.

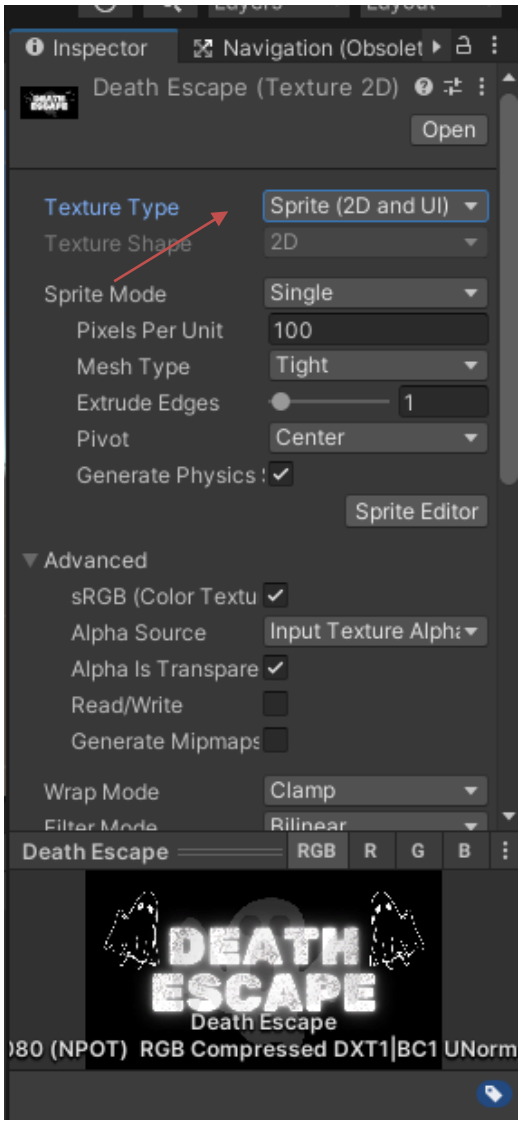


Figura 54 - Sprite na imagem

Após isso adicionamos nas canvas um panel.

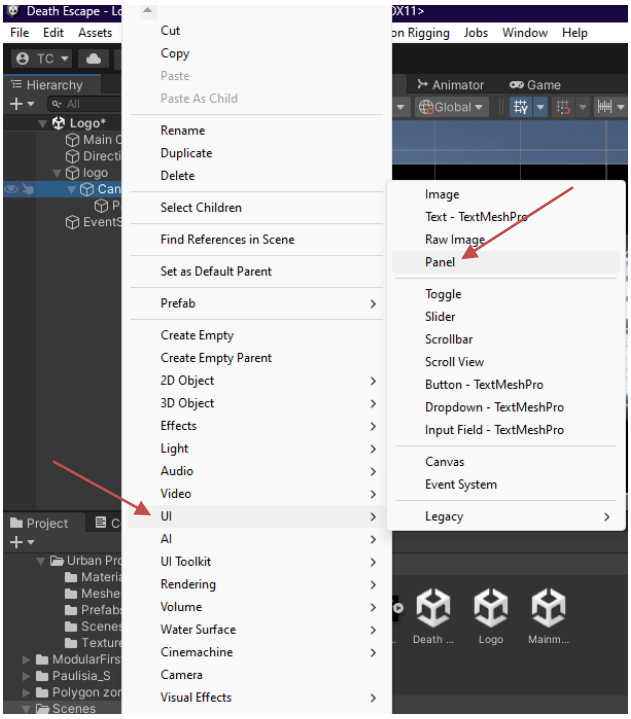


Figura 55 - Panel

Depois disso, adiciono a imagem a “Source Image”.

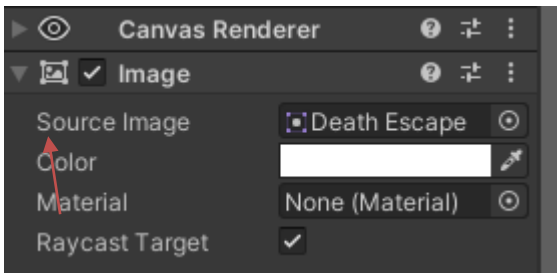


Figura 56 - imagem para o panel

Agora vou adicionar os botões de “play” e “quit” para entrar ou sair do jogo. Vamos em gameObject > UI > Button.

Figura 57 - Botões de play e quit

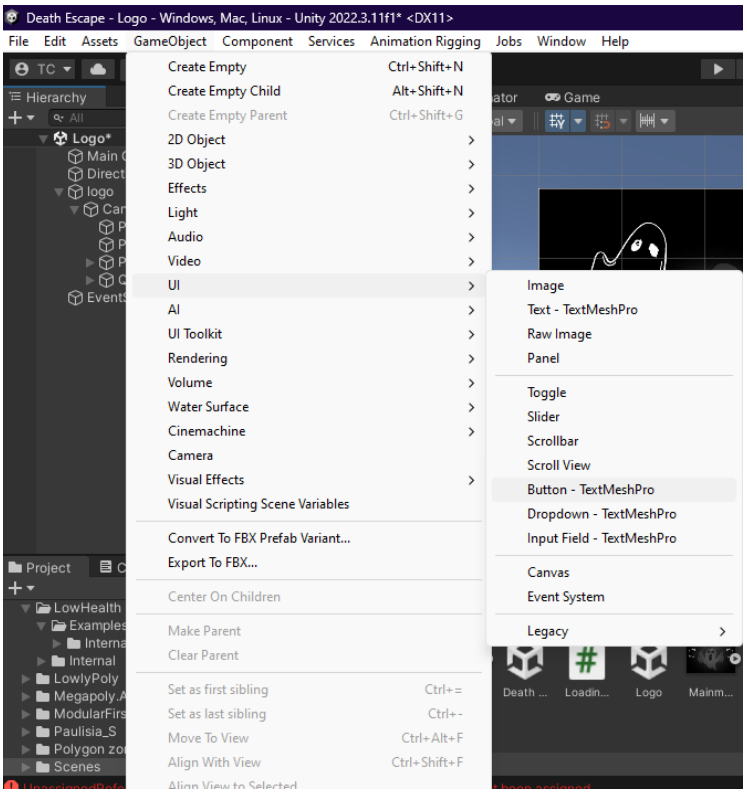


Figura 58 - Como adicionar botões

Depois disso coloquei uma imagem no botão da mesma maneira que coloquei no panel acima.

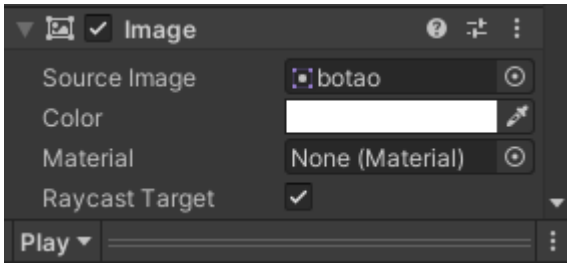


Figura 59 - Imagem do botão

Após isso, fiz o código para os botões. Criei um script chamado Menu e coloquei-o no object “logo” que é o object que tem os canvas.

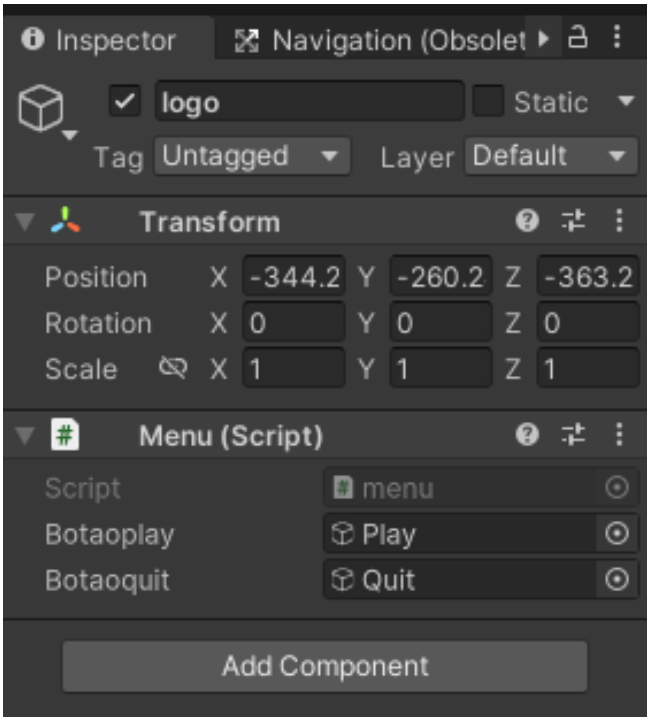


Figura 60 - Script menu

No script, coloquei o namespace “UnityEngine.SceneManagement” para puder ir de uma cena para outra, criei duas variáveis para identificar os botões e criei duas funções uma de play e outra de quit para os respetivos botões.

```
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class menu : MonoBehaviour
{
    public GameObject botaoplay;
    public GameObject botaoquit;
```

Figura 61 - Variáveis do menu

Na função de Play, ou seja, quando o botão play for clicado, desativei os botões para desaparecerem aparecendo “Loading” que estava atrás deles e coloquei para ativar a função de carregar a próxima cena dois segundos depois.

```
public void Play()
{
    botaoplay.SetActive(false);
    botaoquit.SetActive(false);
    StartCoroutine(LoadNextSceneAfterDelay(2f));
}
```

Figura 62 - Função play do menu

Na função Quit, coloquei apenas para sair da aplicação assim que clicar no botão. Na função para carregar a próxima cena coloquei um “yield return” para ter um certo “delay” que é definido na função play que são 2 segundos e coloco para adicionar mais um ao contador de cenas indo para a cena que está a seguir.

```
public void Quit()
{
    Application.Quit();
}

IEnumerator LoadNextSceneAfterDelay(float delay)
{

    yield return new WaitForSeconds(delay);

    SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
}
```

Figura 63 - Função Quit e de esperar

Agora arrastamos os objects dos botões nos respetivos sítios do script.

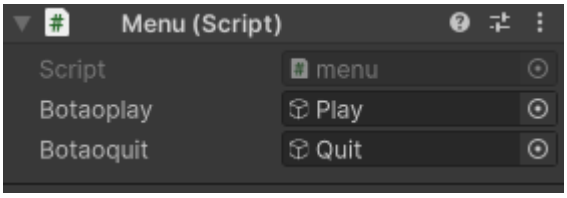


Figura 64 - Atribuição das variáveis

Por fim, preciso de ir nos botões, no inspector em button e onde diz “OnClick”, ou seja, quando é clicado, adicionamos uma condição e arrastamos o object “logo” onde coloquei o script. Depois onde diz “no Function” clicamos e vamos em menu > play() e vice versa para o Quit() no botão quit.

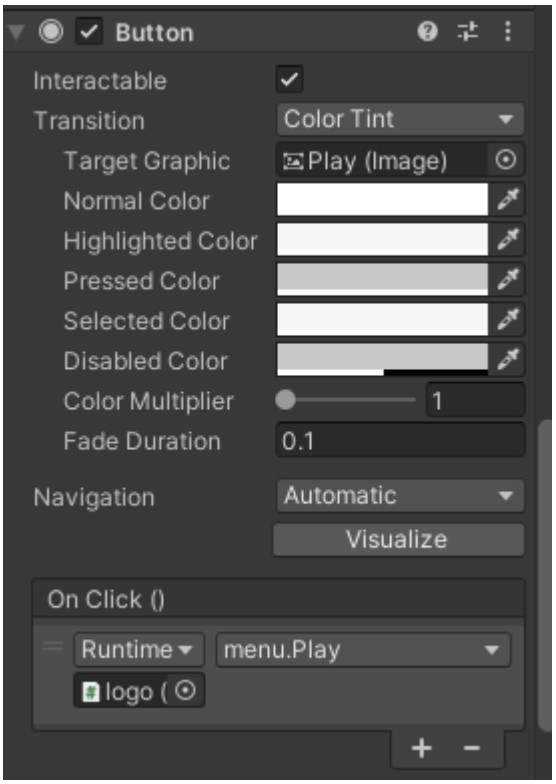


Figura 65 - Configuração de clique

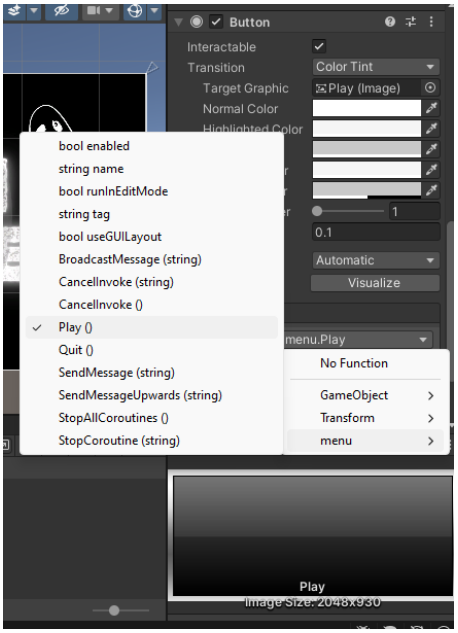


Figura 66 - Função de play no clique

Problemas:

Quando abria a cena depois de clicar no Play a luminosidade desaparecia e ficava tudo escuro. Para resolver tal problema, vamos na cena que ocorre o problema, vamos na barra superior em window > Rendering > lighting e clicamos em auto generator quando abrir a aba de lighting e aguardo gerar a iluminação global da scene como na imagem.

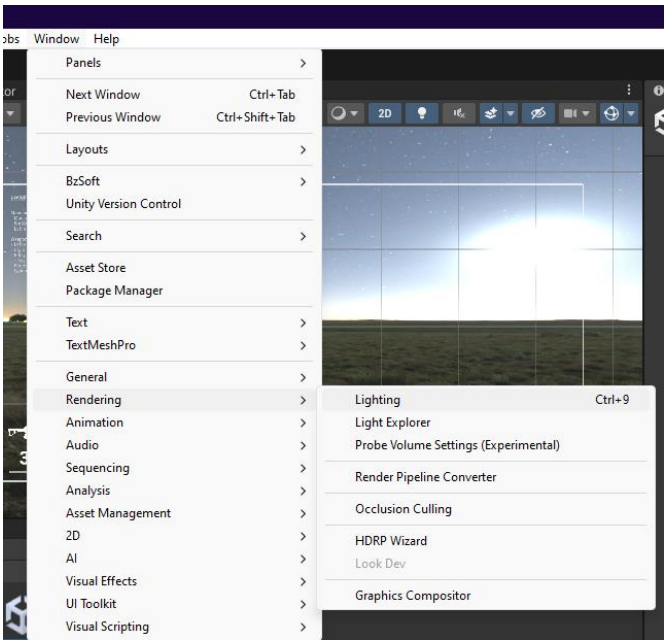


Figura 67 - Adição do menu de luz

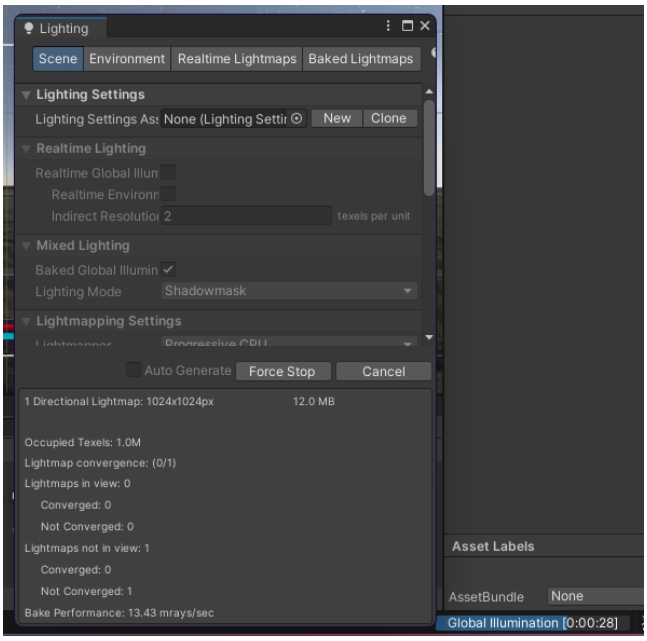


Figura 68 - Menu de luz

Adaptar a qualquer Resolução

Agora preciso de adaptar a interface do jogo a qualquer resolução para isso usei a ancoragem da unity. Coloquei os objetos da UI como membros do panel com o background e fui em “Rect Transform” em cada um deles e no quadradinho coloquei para os objetos ficarem em baixo centrados enquanto no panel de background coloquei-o para esticar para todos os lados.

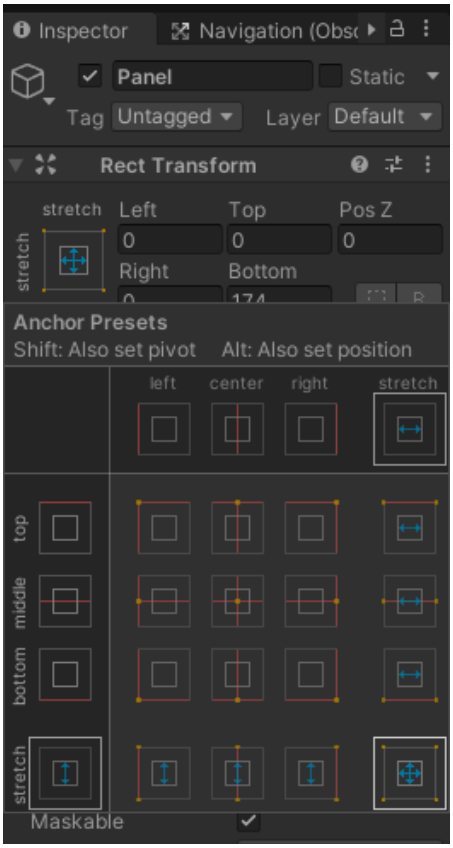


Figura 69 - Ancoragem do panel

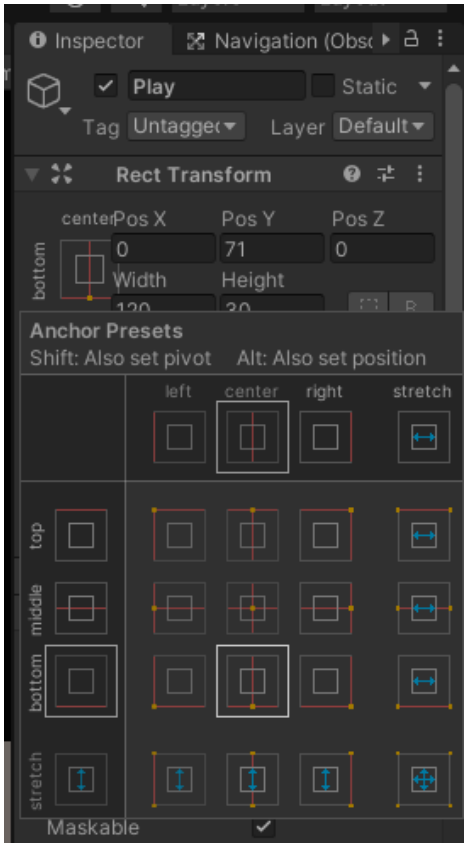


Figura 70 - Ancoragem dos botões

Mudança de botões

Mudei os botões para uns transparentes e por isso também tive que colocar o “Loading” invisível e fiz com que aparecesse depois de clicar no “Play”.

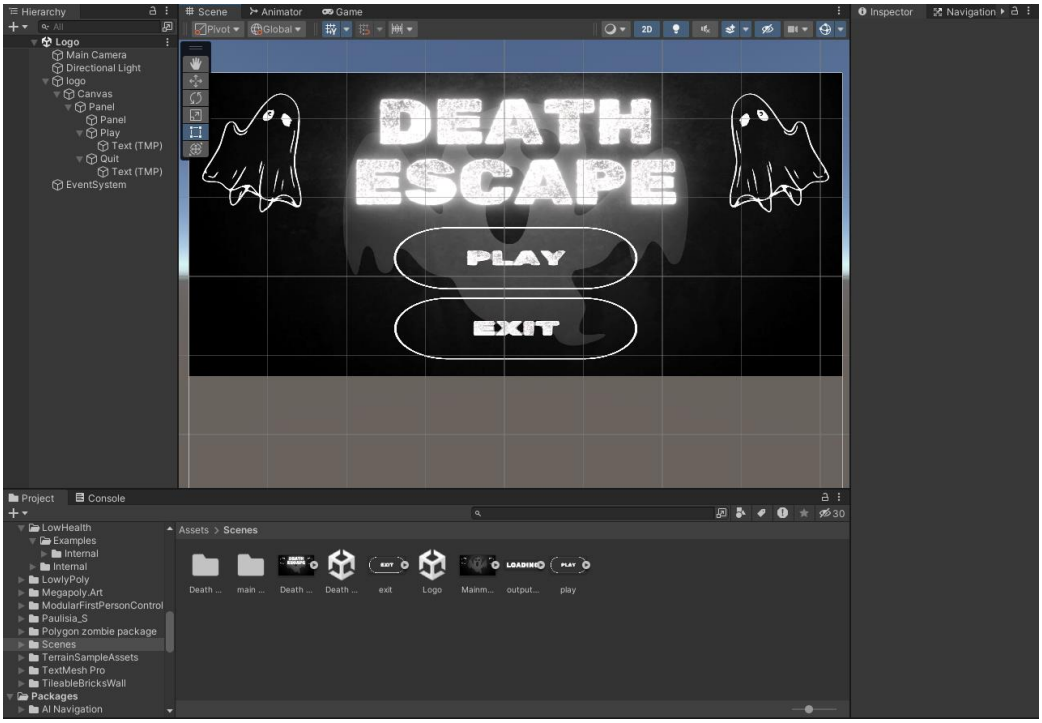


Figura 71 - Novos botões

Para isso deixei o “Loading” transparente indo no panel de Loading > inspector > image > color e mudei para transparência total. Depois disso mudei o código “menu” e adicionei uma variável para identificar o panel e a o nível de transparecia que ia ficar após clicar em play. Na função de play coloquei para ativar a função de modificar a cor para colocar visível novamente.

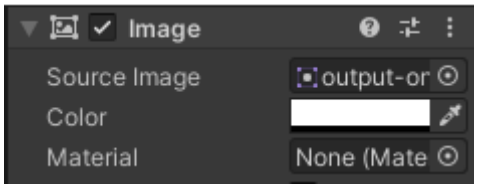


Figura 74 - Imagem dos botões

```
public GameObject loading;

public Color corAtiva = new Color(1f, 1f, 1f, 1f);

public void Play()
{
    botaoplay.SetActive(false);
    botaoquit.SetActive(false);
    SetLoadingPanelColor(corAtiva);
    StartCoroutine(LoadNextSceneAfterDelay(2f));
}
```

Figura 73 - Transição para o Loading

```
private void SetLoadingPanelColor(Color color)
{

    UnityEngine.UI.Image loadingImage = loading.GetComponent<Image>();
    if (loadingImage != null)
    {

        loadingImage.color = color;
    }

}
```

Figura 72 - Retira a transparência

Mês de Março

1ª Semana:

Menu de Pausa

Comecei a fazer o menu de pausa para quando o jogador clicar na tecla “esc” aparecer um menu e pausar o jogo. Para isso criei no canvas um objeto vazio chamado “main menu” e la dentro coloquei as coisas do menu como botões e assim.

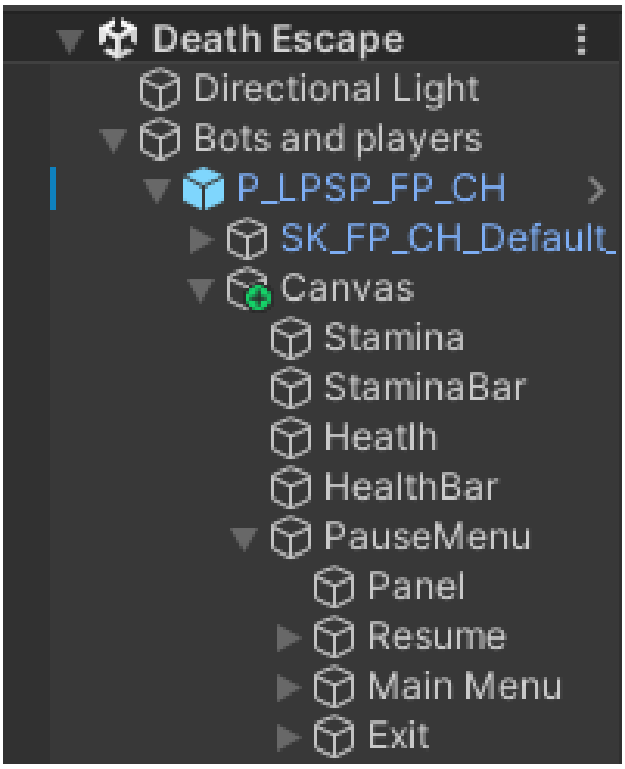


Figura 75 - Organização do canvas

No objeto do canvas adicionei um novo script chamado “PauseMenu”. Comecei por adicionar duas variáveis, uma para dizer se o jogo está pausado ou não e uma para associar ao objeto do menu de pausa. Na função de Update coloquei para verificar se o esc está a ser clicado e se estiver verifica se o jogo está pausado, se estiver chamo a função “Resume” e se não chamo a função “Pause”. Na função de Resume, coloco para desativar o menu de pausa e a variável do jogo pausado em falso, bem como o “timescale” em 1 para o tempo voltar ao normal e não estar pausado. Enquanto na função de Pause faço o contrário.

```
public static bool GameIsPaused = false;
public GameObject PauseMenuUI;

void Update()
{
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if (GameIsPaused)
        {
            Resume();
        }
        else
        {
            Pause();
        }
    }
}
```

Figura 76 - Update do menu pausa

```
void Resume()
{
    PauseMenuUI.SetActive(false);
    Time.timeScale = 1f;
    GameIsPaused = false;
}

void Pause()
{
    PauseMenuUI.SetActive(true);
    Time.timeScale = 0f;
    GameIsPaused = true;
}
```

Figura 77 - Funções Resume e pause

Depois disso fiz os códigos para os botões “exit”, “menu”, e “resume” que no caso já está feito na função resume. No código do menu coloquei para continuar o tempo e abrir a scene do menu inicial e no exit coloquei para sair do jogo.

```
public void loadmenu()
{
    Time.timeScale = 1f;
    SceneManager.LoadScene("Logo");
}

public void exit()
{
    Application.Quit();
}
```

Figura 78 - Função loadmenu e exit

Criei um design igual ao dos botões do menu inicial para os botões do menu de pausa. Coloquei as imagens dos botões em sprite coloquei no componente image deles como fiz anteriormente.

Fui agora no componente button dos botões para adicionar as funções do código para quando eu clico neles. Para isso, como já fiz anteriormente, vou no evento "OnClick" do componente button e adiciono um evento. Depois escolho o objeto com o código que no meu caso é o objeto do canvas e adiciono a Function que quero, por exemplo, no botão de menu quero a função loadmenu.

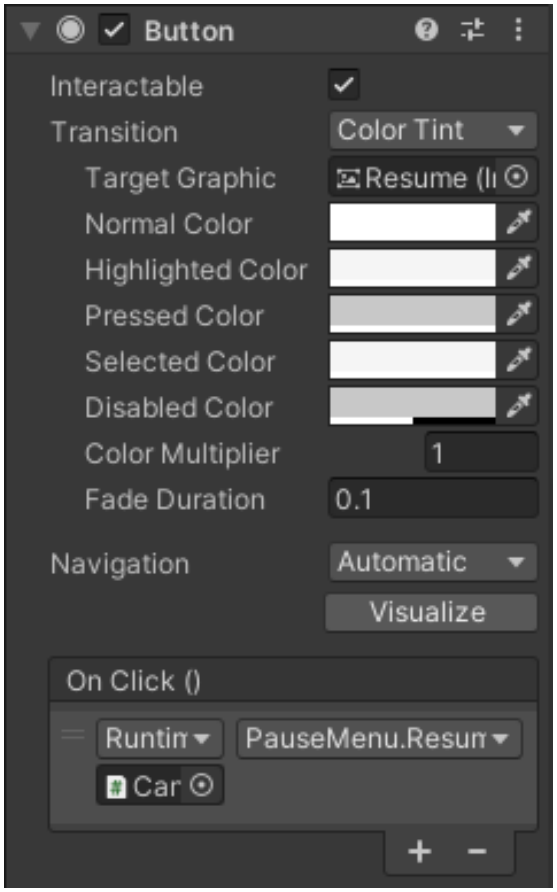


Figura 79 - Evento clique do botão do menu

Efeitos na Camara

Agora coloquei efeitos na camara para tornar o ambiente mais bonito. Para isso usei o “Post Process”, fui na parte superior em window > pack manager, coloquei para aparecer todos os packs da unity indo em packages e selecionando os packages no registo da unity. Depois, procuro na barra de pesquisa “Post Processing” e instalo.

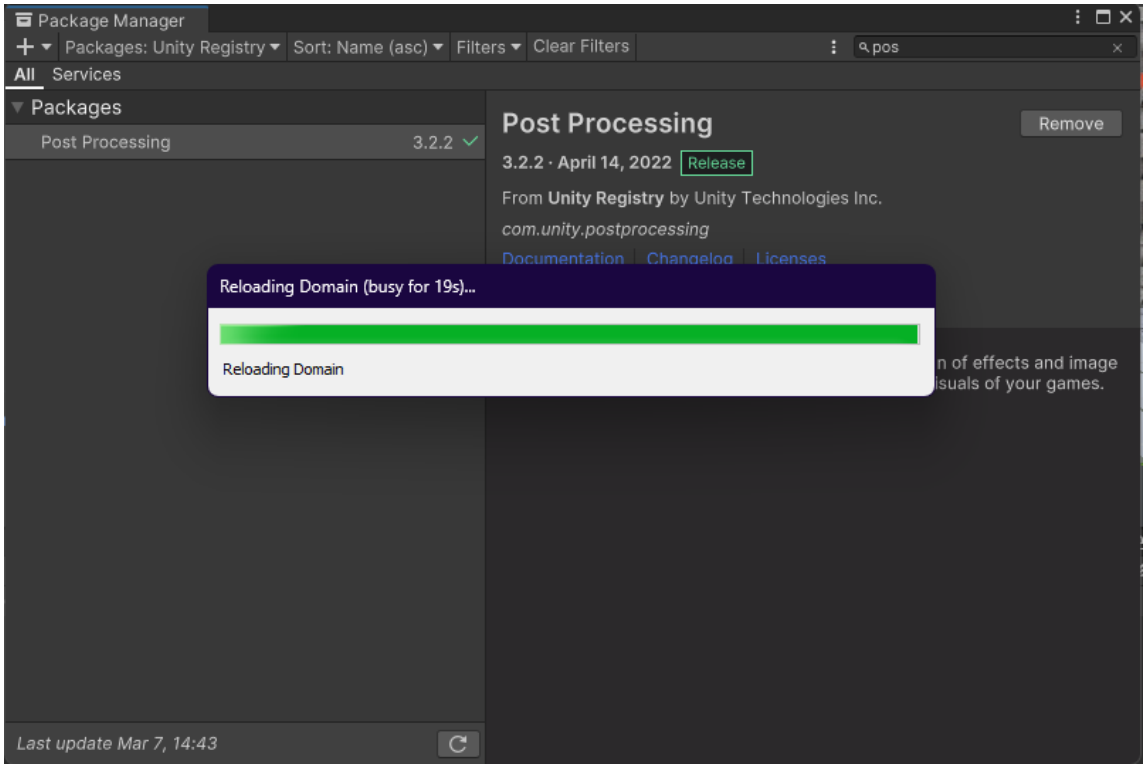


Figura 80 - Instalação de Post Processing

Após a instalação fui na camara do player e adicionei um novo componente chamado “Post Processing Layer”. Criei um layer novo chamado “PostProcessing” indo em layer no inspector da camara e clicando para adicionar novo.

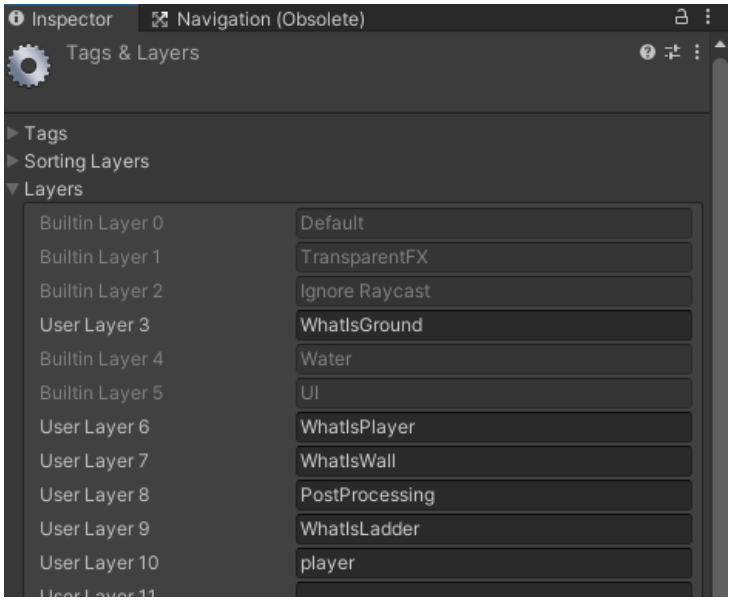


Figura 81 - Nova layer PostProcessing

No Post Processing Layer coloquei o trigger como a camara onde está o componente e o layer para ser ativado coloquei o PostProcessing para não ativar em mais nenhum objeto aleatório.

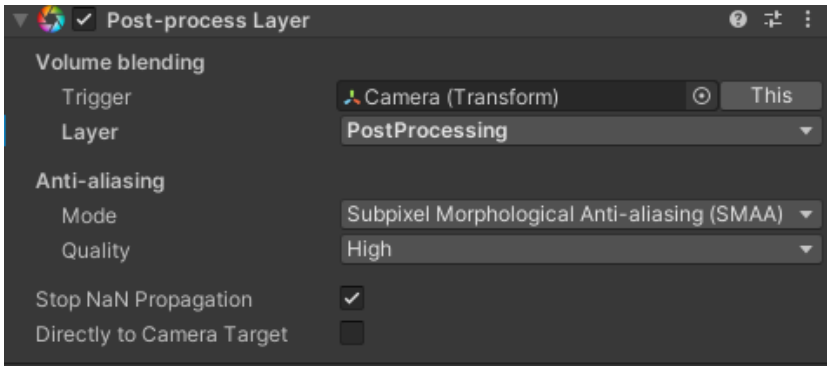


Figura 82 - Layer do Post Processing

Agora, criei um objeto vazio ao lado da camara, ou seja, dentro do jogador, coloquei o layer como Post Processing e adicionei o componente Post Processing Volume. A partir daqui comecei a clicar para adicionar efeitos e fui adaptando e escolhendo os efeitos que queria.

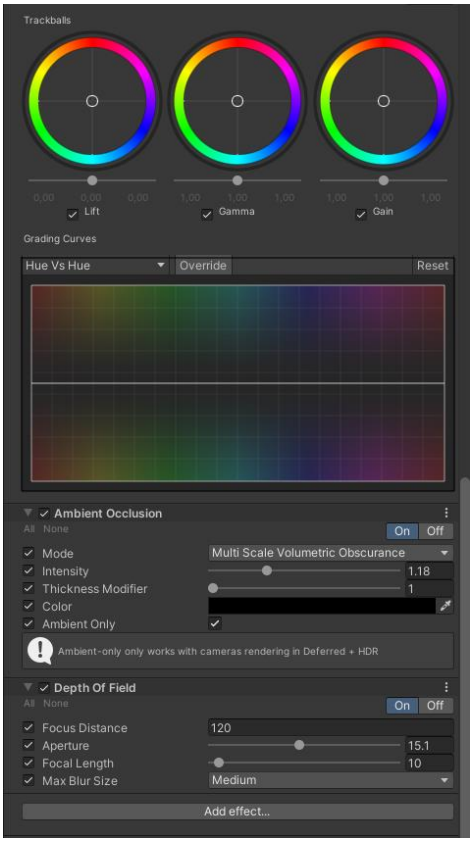


Figura 83 - Efeitos do Post Processing

Por fim coloquei um nevoeiro e para isso foi na parte superior em window > Rendering > lighting.

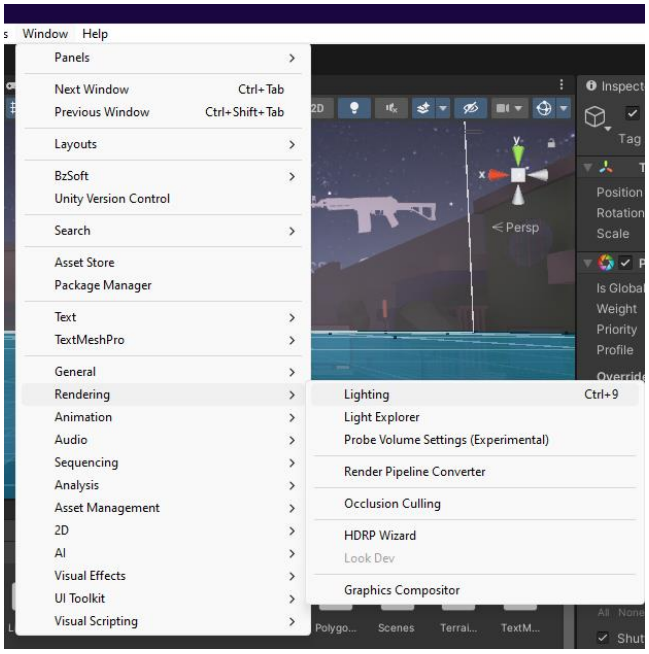


Figura 84 - Como colocar nevoeiro

Depois fui em “Environment” e ativei o “fog” e personalizei-o a meu gosto.

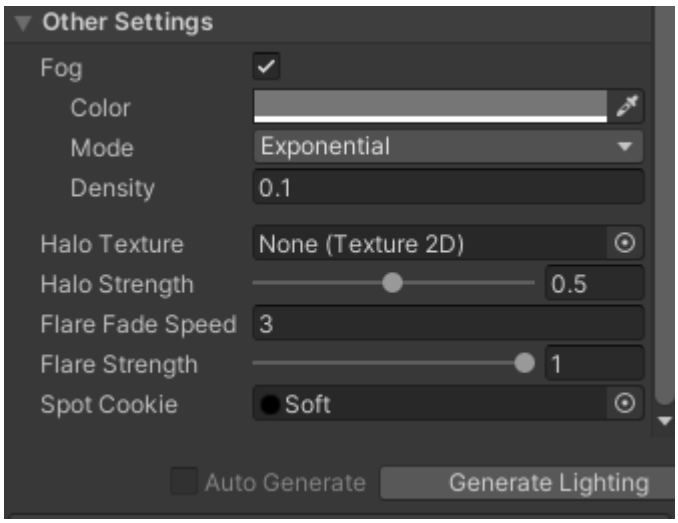


Figura 85 - Nevoeiro

Antes e depois:

Figura 87 - Jogo antes

Figura 86 - Jogo Depois

Tela de Morte

Agora fiz uma tela de morte para quando o jogador morre. Criei um objeto vazio no canvas e chamei de “DeathScreen” e desenhei a interface. Para desenhar a interface usei o website canva como já usei no resto do design baixei e tirei o fundo das imagens, coloquei na unity e coloquei image tipo sprite.

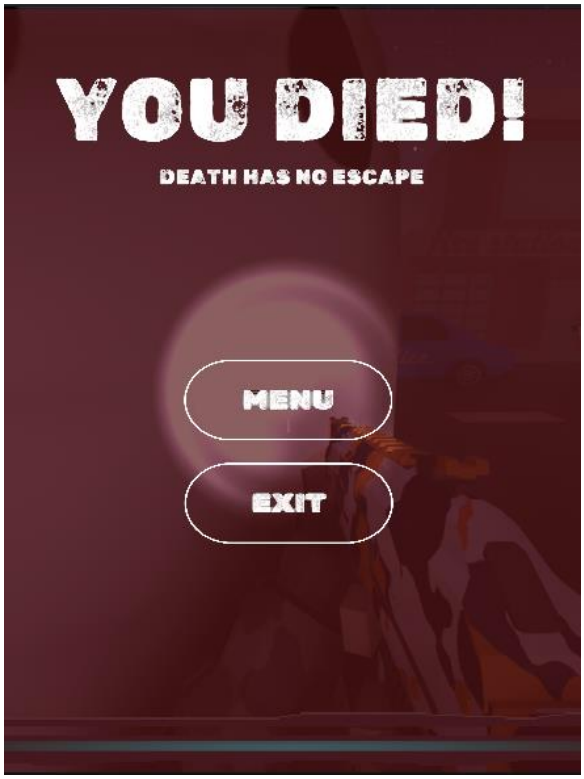
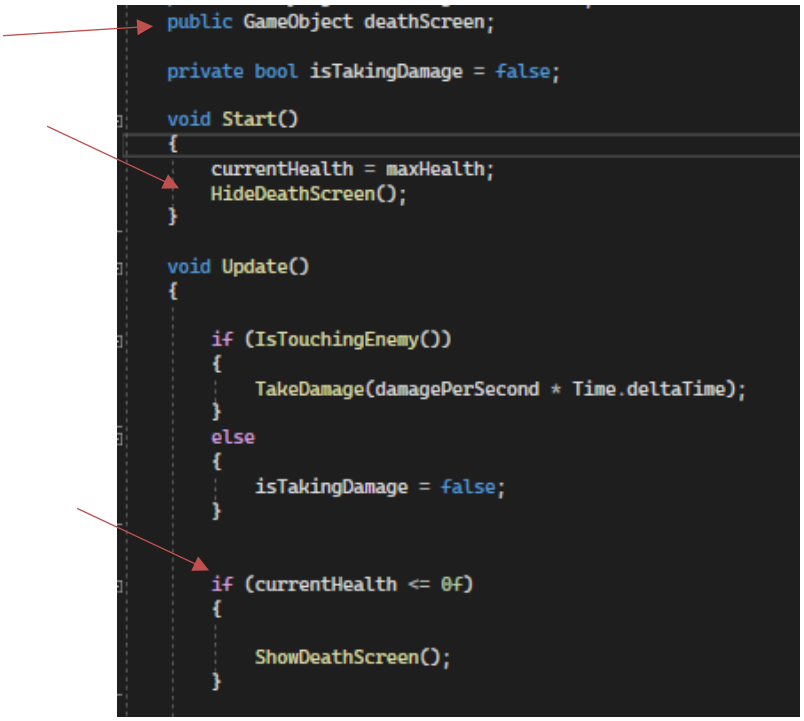


Figura 88 - Tela de morte

Agora tenho de criar o script dos botões, mas antes, modifiquei o script de vida do jogador para a tela de morte aparecer apenas quando o jogador morre. Criei uma variável para arrastar a tela de morte para lá, coloquei no start para estar desativada e apenas ativar caso a vida atual for 0.



```
public GameObject deathScreen;

private bool isTakingDamage = false;

void Start()
{
    currentHealth = maxHealth;
    HideDeathScreen();
}

void Update()
{

    if (IsTouchingEnemy())
    {
        TakeDamage(damagePerSecond * Time.deltaTime);
    }
    else
    {
        isTakingDamage = false;
    }


    if (currentHealth <= 0f)
    {

        ShowDeathScreen();
    }
```

Figura 89 - Como esconder Tela de morte

```
void ShowDeathScreen()
{
    // Ativa a tela de morte
    if (deathScreen != null)
    {
        deathScreen.SetActive(true);
    }
}

void HideDeathScreen()
{
    // Oculta a tela de morte
    if (deathScreen != null)
    {
        deathScreen.SetActive(false);
    }
}
```

Figura 90 - Funções da tela de morte

Tela de Vitória

Agora criei a vitória, para se ganhar eu fiz 2 chefes inimigos onde cada um oferece uma chave após a morte e com essas duas chaves podemos procurar pelo grande portão e indo para ele com as duas chaves podemos assim vencer o jogo. Para fazer isso criei a tela de vitória primeiro. Criei um objeto vazio e chamei de “WinScreen”, a partir daí fiz igual as outras telas, adicionei uns textos por imagens que criei no canva e por fim adicionei o código para aparecer um texto, depois outro e de seguida ir para o menu inicial. Comecei por adicionar o namespace para controlar as scenes para me mover para outra, criei 3 variáveis, duas para identificar as imagens e uma para identificar a tela de vitoria.

```
using System.Collections;
using System.Collections.Generic;
using System.Threading;
using UnityEngine;
using UnityEngine.SceneManagement;

public class WinScreen : MonoBehaviour
{
    public GameObject Winscreen;

    public GameObject wonImage;
    public GameObject continuedImage;
```

Figura 91 - Variáveis da tela de vitória

Depois disso, no Start ativei uma imagem, desativei a outra e ativei a função de trocar de imagem. Na funçao de trocar de imagem espero 4 segundos e ativo a que estava desativada e vice-versa. Depois espero mais 4 segundos e vou para a scene do menu inicial.

```
void Start()
{
    wonImage.SetActive(true);
    continuedImage.SetActive(false);

    StartCoroutine(SwitchImages());
}

IEnumerator SwitchImages()
{

    yield return new WaitForSeconds(4f);

    wonImage.SetActive(false);
    continuedImage.SetActive(true);

    yield return new WaitForSeconds(4f);

    SceneManager.LoadScene("Logo");
}
```

Figura 92 - Função Start e trocar imagens

Chave

Baixei um modelo de uma chave enferrujada e coloquei-a com os chefes desativada e apenas se ativa quando o chefe é destruído. Para fazer isso e para puder pegá-la alterei o “targetscript” e o “healthbar”.
No targetscript, no Update adicionei a função nova “DropChave()” quando os hits que o inimigo leva for maior ou igual aos hits necessário para destruí-los.

```
private void Update()
{
    // valores random de tempo max e minimo
    randomTime = UnityEngine.Random.Range(minTime, maxTime);


    if (hits >= hitsToDestroy)
    {
        DropChave();
        Destroy(gameObject);

    }
```

Figura 93 - Função DropChave no Update

Na função de DropChave() eu encontro o objeto da chave e retiro a ligação dela do chefe para não ser destruída com ele e depois ativo a chave.

```
 private void DropChave()
{
    Transform chave = transform.Find("rust_key");
    if (chave != null)
    {

        chave.parent = null;

        chave.gameObject.SetActive(true);
    }
}
```

Figura 94 - Função DropChave

No “Healthbar” adicionei duas variáveis uma para contar a chaves coletadas e outra para ativar caso as duas tenham sido apanhadas para permitir a entrada no portão.

```
private int chavesColetadas = 0;
private bool todasAsChavesForamColetadas = false;
```

Figura 95 - Variáveis de chave

Na função Update verifico se o player esta a tocar na chave com uma função que também adicionei. Se sim ativo a função de coletar chave. Depois coloquei que se o número de chaves coletadas for maior ou igual a 2 e tiver a tocar no portão (chamando também uma função que verifica o mesmo) abre a scene de vitória.

```
void Update()
{
    if (IsTouchingChave(out GameObject chaveTocada))
    {
        ColetarChave(chaveTocada);
    }

    if (chavesColetadas >= 2 && IsTouchingFinalGate())
    {
        SceneManager.LoadScene("Vitoria");
    }
```

Figura 96 - Funções da chave no Update

Utilizei na função de tocar a chave e tocar o portão método que usei para detetar se o jogador tocava o inimigo com os colliders e se o collider for da tag “Chave” e no portão da tag “FinalGate” então retornam verdadeiros e passamos a função de coletar chave que aumenta o número de chaves coletadas e se for maior que 2 ou igual a 2 ativa a variável de todas as chaves coletadas e desativa a chave para que ela desapareça.

```
bool IsTouchingChave(out GameObject chaveTocada)
{
    chaveTocada = null;
    Collider[] colliders = Physics.OverlapSphere(transform.position, detectionRadius);
    foreach (Collider collider in colliders)
    {
        if (collider.gameObject.CompareTag("Chave"))
        {
            chaveTocada = collider.gameObject;
            return true;
        }
    }
    return false;
}

void ColetarChave(GameObject chave)
{
    chavesColetadas++;
    if (chavesColetadas >= 2)
    {
        todasAsChavesForamColetadas = true;
    }
    chave.SetActive(false);
}

bool IsTouchingFinalGate()
{
    Collider[] colliders = Physics.OverlapSphere(transform.position, detectionRadius);
    foreach (Collider collider in colliders)
    {
        if (collider.gameObject.CompareTag("FinalGate"))
        {
            return true;
        }
    }
    return false;
}
```

Figura 97 - Funções da chave

Chave a rodar

Coloquei a chave a rodar quando é deixada pelo inimigo. Para isso criei um script para fazê-la girar onde criei uma variável chamada spinspeed para controlar a velocidade da chave a girar. No Update criei um código para ativar a rotação da chave a cada frame que passa.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class spin : MonoBehaviour
{
    public float spinSpeed = 50f;

    void Update()
    {

        transform.Rotate(Vector3.right, spinSpeed * Time.deltaTime);
    }
}
```

Figura 98 - Código da Rotação da Chave

Áudio de terror Ambiente

Agora coloquei um áudio ambiente de terror para isso baixei um áudio da internet puxei para dentro da unity, foi por exemplo no personagem e adicionei um áudio Source a adicionei o áudio baixado e deixei em Loop para não acabar.

Figura 99 - Áudio de terror Ambiente

Áudio de Vitória

Coloquei agora áudio de vitoria, fiz o mesmo processo e adicionei um áudio Source no canvas da scene de vitoria.

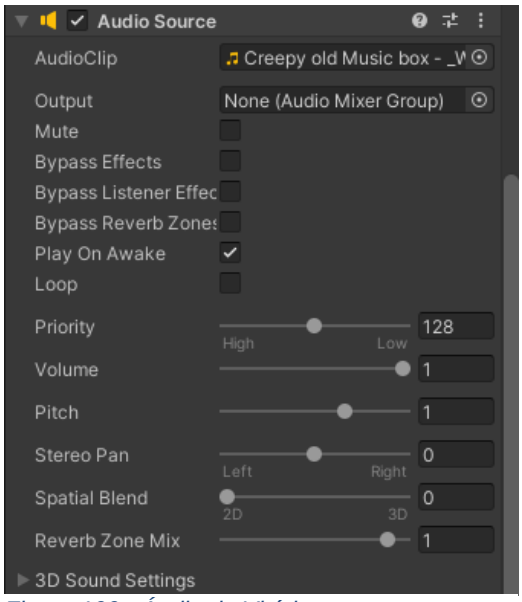


Figura 100 - Áudio de Vitória

Áudio de Início

Coloquei também áudio de início, fiz o mesmo processo do anterior e adicionei um áudio Source no canvas da scene de menu inicial.

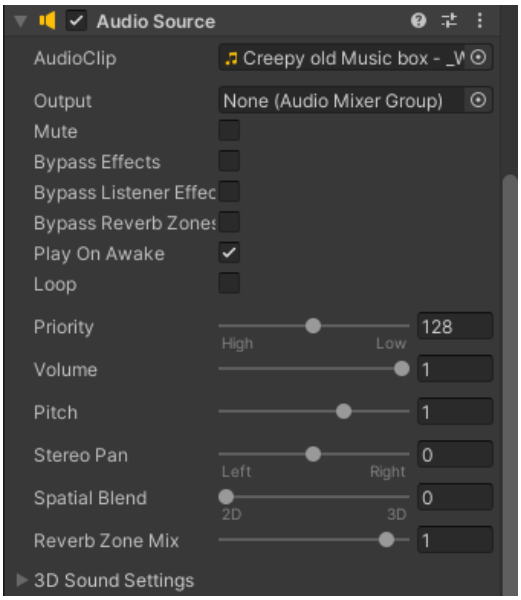


Figura 101 - Áudio de Início

Mudança de cor nos botões

Coloquei para quando passar o cursor pelos botões, fiquem vermelhos. Para isso fui no inspector dos botões e fui nas cores para mudar o Highlighted Color.

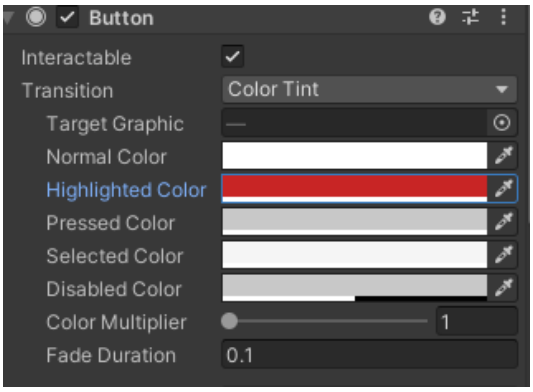


Figura 102 - Mudança de cor nos botões

Modificação de tela de morte

Mudei a tela de morte, coloquei fundo escuro e agora aparece “oh no” e seguidamente “you died” e o menu de escolhas com um áudio novo também.

```
void Start()
{
    Cursor.lockState = CursorLockMode.None;
    Cursor.visible = true;

    SetDeathScreen();

    StartCoroutine(SwitchImages());
}


void SetDeathScreen()
{
    menub.SetActive(false);
    exitb.SetActive(false);
    ohno.SetActive(true);
    died.SetActive(false);

}


IEnumerator SwitchImages()
{
    yield return new WaitForSeconds(5f);

    menub.SetActive(true);
    exitb.SetActive(true);
    ohno.SetActive(false);
    died.SetActive(true);
}
```

Figura 103 - Mudança de tela de morte (Código)

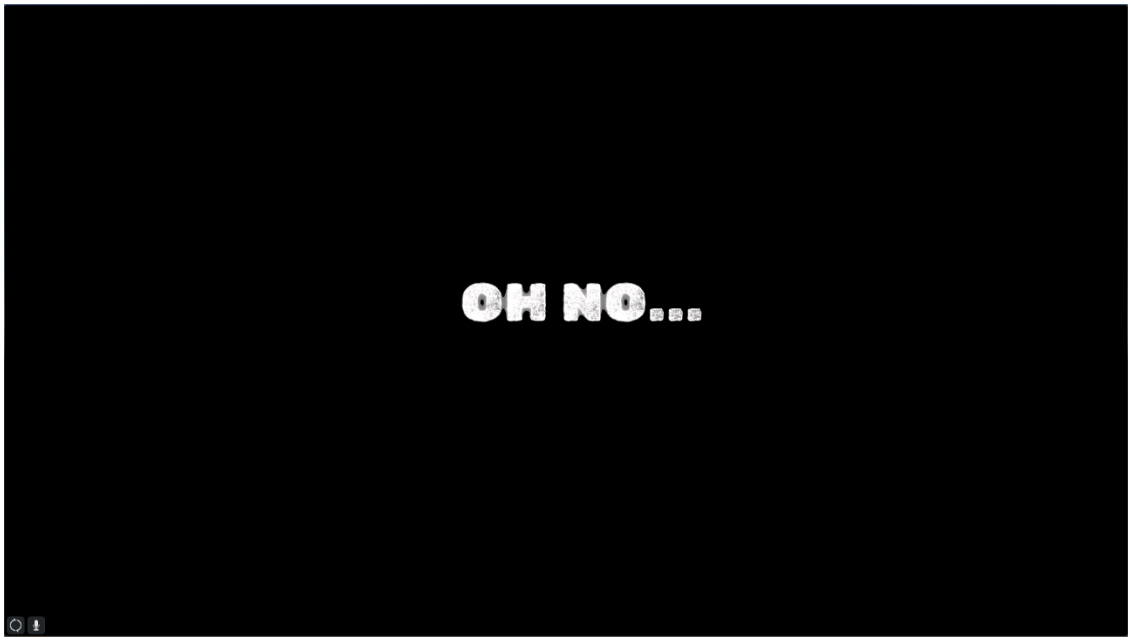


Figura 104 - Tela de morte oh no

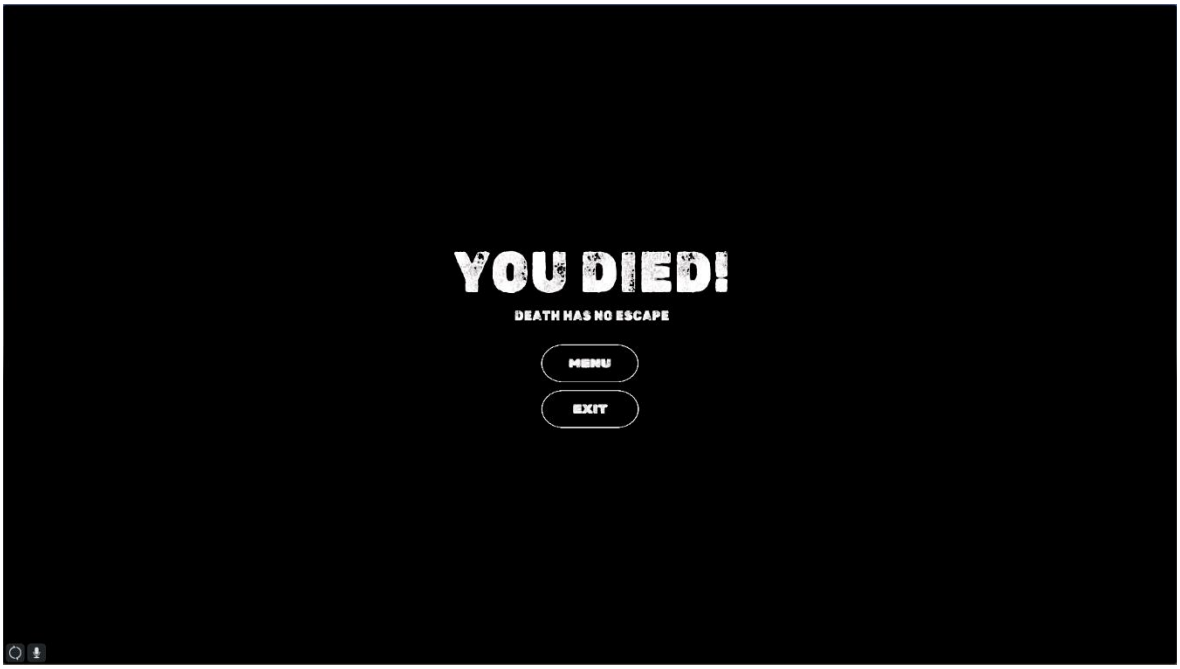


Figura 105 - Tela de morte you died

Colocar Jogo em aplicação executável

Agora coloquei o jogo como uma aplicação executável para puder jogar normalmente. Para isso fui em file > Build Settings > build e seleciono onde quero colocar a pasta com o jogo executável e começa a fazer a pasta.

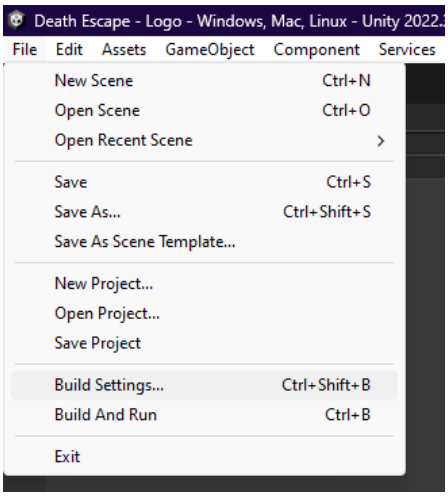


Figura 107 - Build settings

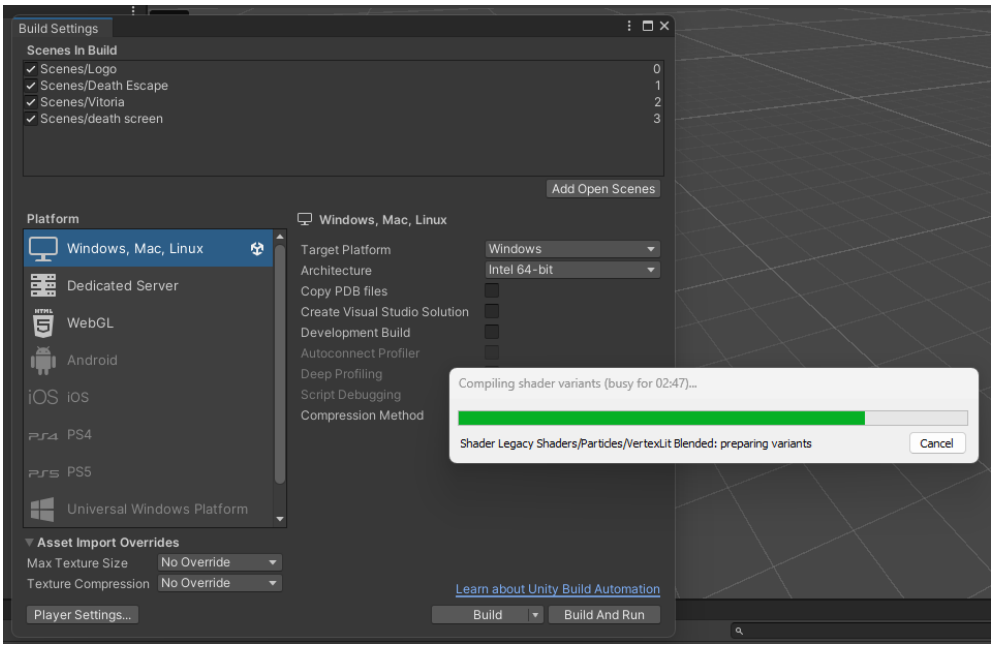


Figura 106 - Build

Website

Criei por fim um website para informação e download do jogo. Para isso usei o wordpress e criei um design a partir daí. Link: https://deathescape.wordpress.com

Figura 109 - Website (1)

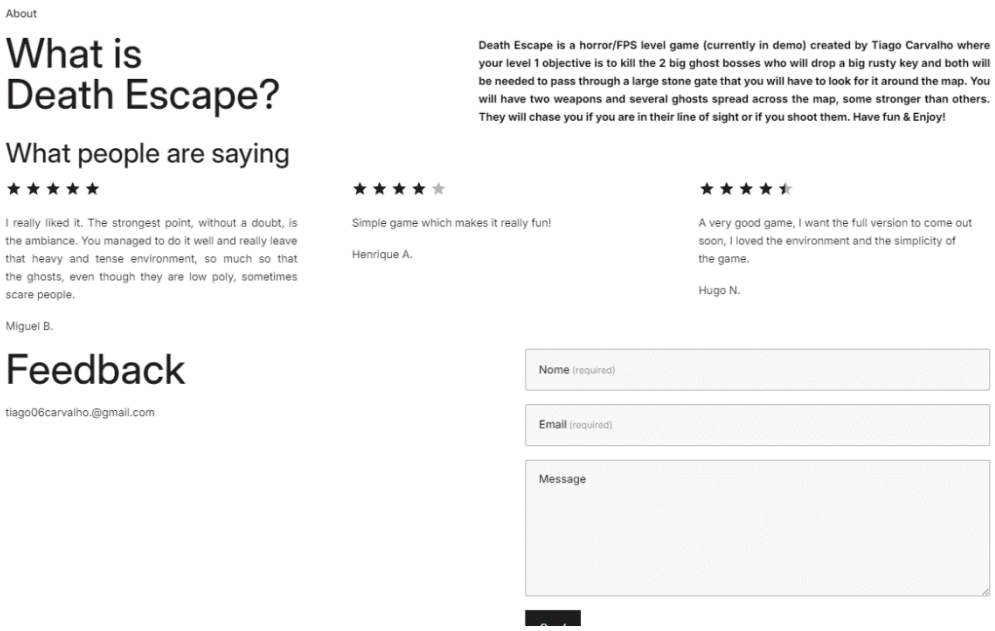


Figura 108 - Website (2)

Link para o jogo

Fiz agora um link para o jogo para adicionar no website para isso utilizei o mega. Fui no mega passei a pasta do jogo para lá, fui em links e criei novo link com a pasta, copiei o link e coloquei-o no botão de download do site.

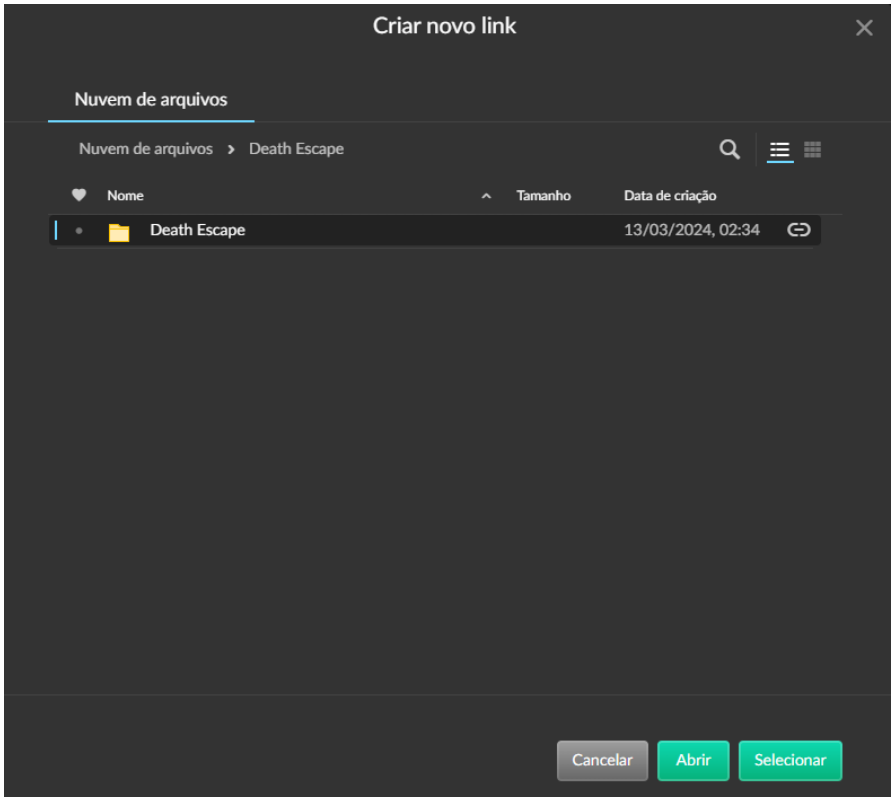


Figura 110 - Criação do Link para o jogo

Adicionei a função num botão para baixar o meu jogo, ou seja, quando clico no botão sou levado para um link no mega para dar download do meu jogo.

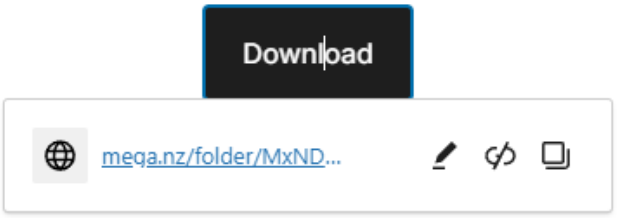


Figura 111 - Link para baixar o jogo

Conclusão

A criação do jogo 'Death Escape' representa um desafio emocionante para mim, Tiago Carvalho, um estudante do curso de Gestão e Programação de Sistemas Informáticos na Escola Profissional de Valongo. Esta Prova de Aptidão Profissional é uma oportunidade para demonstrar a minha dedicação e empenho adquiridos ao longo do curso através de um projeto final significativo. O meu objetivo é oferecer diversão, tensão e uma pitada de competitividade, levando os jogadores a uma jornada inesquecível através do desconhecido. Com dedicação e paixão, enfrentei este desafio e dei vida ao meu jogo.

Webgrafia

https://www.youtube.com

https://chat.openai.com

Manual de Utilizador

Figura 112 - Logo (2)

Intrdução

Aqui temos um breve resumo e explicações sobre como funciona o jogo e como jogar o mesmo. Falarei dos controlos do jogo, como se ganha, como se perde e o que se pode fazer nele.

O que fazer no Death Escape?

No Death Escape podemos explorar o mapa, matar fantasmas (uns mais fortes que outros), e tentar concluir o primeiro nível que requer a morte de dois fantasmas enormes sendo eles os chefes do nível 1.

Figura 113 - Explicação HUD

Arma usada e número de balas disponíveis.

Barra vermelha de vida e barra azul de stamina.

Créditos

Controlos

Dentro do jogo podemos segurar a tecla “Tab” para ver um breve resume dos controlos.

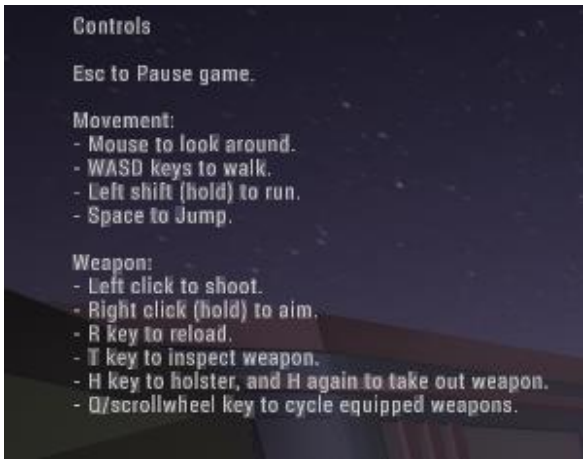


Figura 114 - Menu Tab

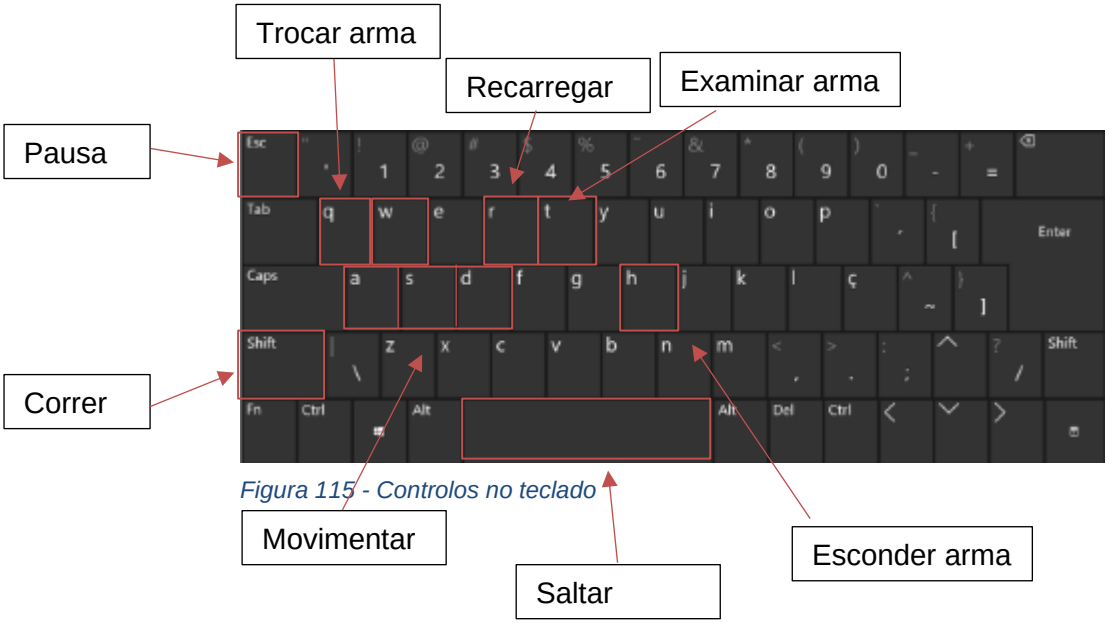


Figura 115 - Controlos no teclado

Como ganhar?

Para vencer o nível 1 é necessário matar dois chefes para obter duas chaves que desbloquearão um portão. Após isso é só procurar e ir até ao portão para sair da cidade.

Figura 116 - Portão final

Figura 117 - Boss

Como perder?

Para perder é muito simples bastar ser morto pelos inimigos, assim que eles encostarem no jogador, o jogador perderá vida continuamente até morrer.

No menu de morte temos o botão de menu e exit. O menu volta para o menu inicial e exit, sai do jogo.

No menu de pausa a mesma coisa também com o botão resume que volta ao jogo.

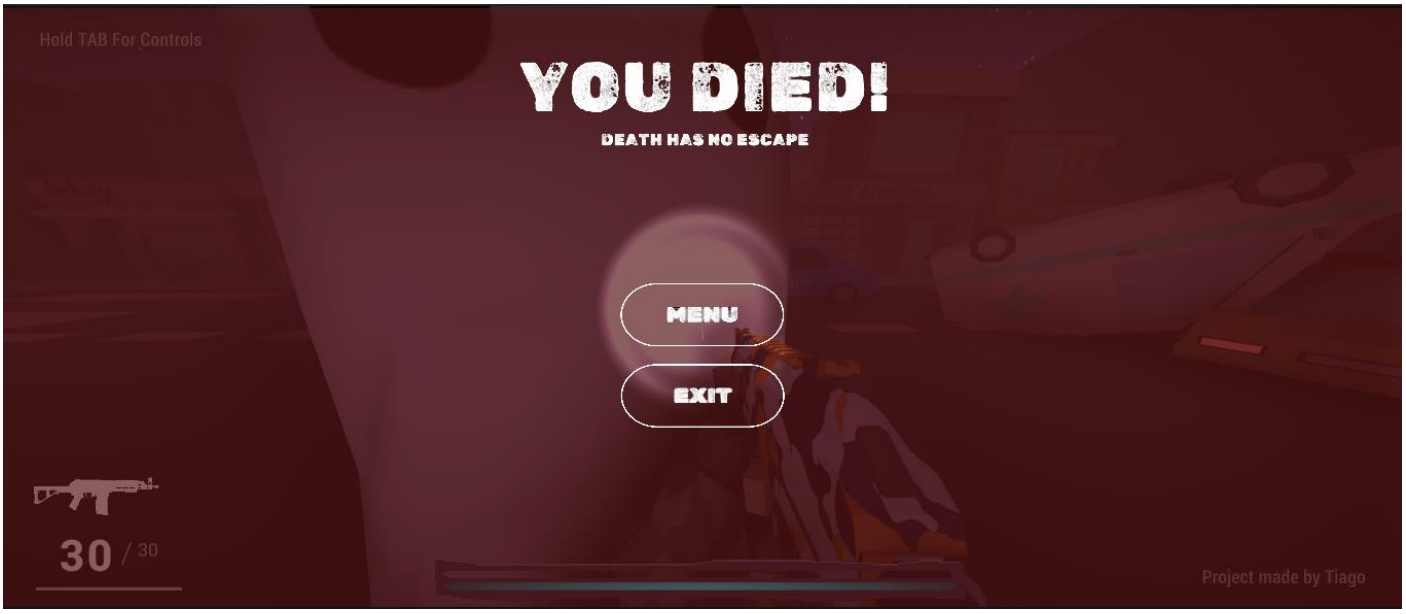


Figura 118 - Tela de morte

Conclusão

Isto é tudo o que precisão de saber sobre Death Escape Demo. Obrigado por jogarem.