



**UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
BACHARELADO EM ENGENHARIA DA COMPUTAÇÃO**

**RESUMO SOBRE A LGPD:
Lei Geral de Proteção de Dados Pessoais**

**BELÉM - PARÁ
2022**

LEANDRO HENRIQUE FERREIRA PINHEIRO - 201806840069
ANDRÉ LUCAS MENDONÇA SANTANA - 201806840095
ALLAN JORGE MENDONÇA FERREIRA MENDES - 201806840096
EMANOEL CRISTHIAN LEÃO MARINHO - 201806840098
VICTOR HIERRO MORAES FERREIRA - 201806840071

RESUMO SOBRE A LGPD:
Lei Geral de Proteção de Dados Pessoais

Documento apresentado como requisito total para o 1º Trabalho da disciplina de Legislação na Engenharia de Telecomunicações da Faculdade de Engenharia da Computação e Telecomunicações - UFPA.

Professora orientadora: Dra. Rafaela Teixeira Sena Neves.

Conteúdo

1	Introdução	2
1.1	Funcionalidades do Jogo	2
2	Ferramentas Utilizadas	2
2.1	Unity	3
2.2	Visual Studio Code	3
3	Implementação do Projeto	3
3.1	Implementação do Ambiente 3D	3
3.1.1	Objeto Player	4
3.1.2	Objeto Island	4
3.1.3	Objeto Hazard	5
3.1.4	Objeto Cenário	6
3.2	Scripts	6
3.2.1	Player	6
3.2.2	Hazard	9
3.2.3	Lerp	9
3.2.4	AutoDestroyer	10
3.2.5	MainMenu	11
3.2.6	GameOverMenu	12
3.2.7	GameManager	12
4	Técnicas Utilizadas nos Objetos	15
4.1	Rotação	15
4.2	Translação	15
4.3	Escala	15
4.4	Iluminação, Sombra	15
4.5	Algoritmos pontuais e geométricos para visibilidade	15
5	Considerações Finais	16

1 Introdução

O presente projeto tem por objetivo implementar um ambiente gráfico em 3D que contemple a definição de objetos 3D e as respectivas transformações geométricas nesses objetos 3D, tais quais: translação, rotação e escala. Não só isso como também objetiva adicionar neste ambiente modelos de iluminação, cor, textura e sombra.

Tendo em vista o objetivo deste trabalho, a equipe decidiu em conjunto que o tema do projeto será o desenvolvimento do jogo Chicken Box, que apesar de não ser muito conhecido, abrange a implementação de todas as funcionalidades desejadas para os objetivos deste projeto. Como não foi definida ferramenta obrigatória para o desenvolvimento, a equipe escolheu utilizar a ferramenta Unity, grandemente conhecida no mercado de jogos e ambientes gráficos. A ideia partiu do nosso entendimento acerca do atual cenário do mundo dos jogos, da sua importância deste mundo para a sociedade e sua disponibilidade de vagas quanto ao mercado de trabalho, portanto aprofundar-se em desenvolvimento de jogos no Unity, em especial os jogos 3D, é extremamente benéfico para a equipe como um todo como futuros profissionais da área.

1.1 Funcionalidades do Jogo

Para os não conhecedores deste jogo a ser desenvolvido, basicamente a ideia do jogo se tratará de um personagem (player) que iniciará o jogo ao centro de uma plataforma ambiente e terá por objetivo desviar das caixas que caem do céu. Além disso, a cada caixa que colide com o chão, o sistema de score aumentará a pontuação do usuário em 1 ponto.

O personagem principal (player) se movimenta apenas no eixo horizontal e as caixas ao colidirem com o ambiente sofrem a transformação da escala e se subdividem em caixas menores como um efeito de destruição, o mesmo ocorre com o player quando o mesmo é atingido pelas caixas, caracterizando assim a derrota do jogo. Tudo isso por meio de um sistema de partículas implementado ao cenário.

Logo, as funcionalidades do jogo serão: a movimentação do player, objetivando esquivar das caixas que irão cair; as caixas que caem do céu, que serão responsáveis por destruir o player e o sistema de score, para definir a pontuação de cada jogador.

2 Ferramentas Utilizadas

Neste projeto, foi utilizado a plataforma unity para a criação do ambiente abrangendo a implementação de modelos de iluminação, sombra, cor e textura; sistema de score; organização do jogo; alocação de objetos e as transformações geométricas realizadas nos mesmos; além de trabalhar também com a interação dos objetos com o player. Essas interações e transformações se fazem por meio de scripts na linguagem C# para uma melhor comunicação com os componentes que a Unity oferece. Por fim, as texturas foram encontradas na web então neste projeto não utilizamos o software Photoshop para o desenvolvimento de texturas. As ferramentas serão melhores descritas abaixo.

2.1 Unity

Em primeiro, temos a ferramenta principal para o desenvolvimento do jogo, a ferramenta Unity. O Unity é uma plataforma de desenvolvimento de jogos, sendo esses disponíveis para diversas ambientações e dispositivos. Para o nosso jogo utilizamos a versão 2020.3.31f1 do editor.

Além do mais, a Unity fornece componentes e funções que auxiliam em projetos, sejam eles 2D ou 3D. Visto que o presente projeto se trata de um jogo 3D, os componentes utilizados, as funções e os objetos utilizados neste jogo foram todos especificamente desenvolvidos para ambientes 3D.

2.2 Visual Studio Code

Por fim, não foi citado antes, mas esta também foi uma ferramenta utilizada, porém, para o projeto qualquer editor de texto seria o suficiente. O VS Code é um editor de código-fonte criado pela Microsoft para a construção e desenvolvimento de aplicativos.

O Visual Studio Code é incorporado à Unity, dessa forma, pode-se editar e testar os códigos fontes instantaneamente na plataforma, o que propiciou uma experiência mais imersiva e versátil durante a criação do jogo. Nesta ferramenta que foram desenvolvidos os scripts na linguagem C#.

3 Implementação do Projeto

3.1 Implementação do Ambiente 3D

Para a criação do nosso ambiente 3D, os pontos a que deveríamos nos atentar são: iluminação do ambiente; cor, sombra e textura dos objetos. Tratando-se de iluminação, por meio do unity adicionamos uma luz ambiente (Directional Light) que simula uma luz solar podendo ser colocada em qualquer posição do nosso ambiente, desta forma garantimos a iluminação do ambiente e a cor dos objetos de forma correta.



Figura 1: Tela inicial do jogo demonstrando o ambiente.

Após isso, criamos um objeto cubo, nomeamos como “Island” e alteramos sua escala nos eixos x, y e z para obtermos uma plataforma que servirá como uma base para o nosso player e para os itens do nosso ambiente. Quanto às cores, foi criado uma pasta em Assets chamada “Materials” em que colocamos as cores dos nossos objetos. Os assets que compõem nosso cenário foram baixados gratuitamente da internet nos sites: Quaternius e Kenney. Além do mais, adicionamos um pacote do unity chamado “cinemachine”, que possibilita criar câmeras virtuais. Dentro desse leque de opções temos o “Cinemachine Impulse Source”, que dá um efeito de “shake” na câmera e outra opção que denominamos de MainVCam, em que a câmera irá seguir o personagem na horizontal. É importante destacar que para chegar no produto final do nosso cenário algumas transformações foram feitas, tais como de escala, rotação e translação dos objetos.

Em seguida, criamos os nossos objetos que fazem parte do jogo Chicken Box, como a galinha (denominamos de “player”), o hazard (as caixas caindo), o cenário (Objeto cenário) e a ilha principal (Objeto Island).

3.1.1 Objeto Player

Para a criação do objeto player, foi adicionado um objeto cubo, que é disponibilizado pelo unity, para ser uma base para o nosso personagem. A fim de preparar o player para ter aspecto de objeto do nosso cenário 3D, adicionamos alguns componentes como: *Mesh Renderer*, *Box Collider*, *RigidBody* e um script para o movimento do nosso personagem que veremos mais à frente. Além disso, para ambientar ao cenário do jogo, adicionamos uma textura de galinha para o nosso player.

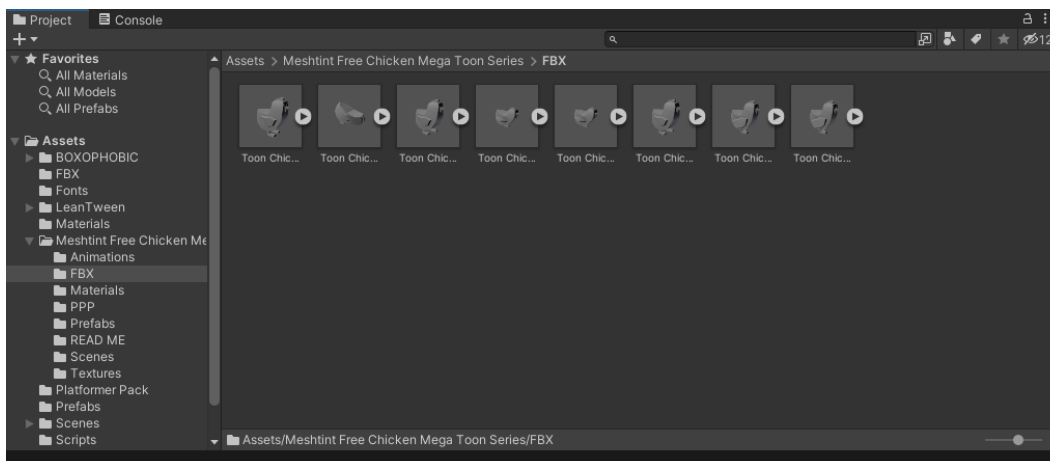


Figura 2: Imagem dos objetos que formam o objeto Galinha (Player).

3.1.2 Objeto Island

O objeto Island é o objeto que serve como uma plataforma para o nosso player. A princípio, a criação dele é parecida com a do player, em que é adicionado um objeto cubo, com os mesmos componentes utilizados no objeto player, porém a sua escala é alterada para obtermos um objeto em forma de plataforma. Além disso, para a ambientação colocamos uma textura em tom terroso para este objeto. A figura de número 3 demonstra o ambiente como um todo, o

nosso objeto island refere-se ao bloco em que os componentes do cenário estão em cima, o bloco marrom que sustenta a ilha flutuante.

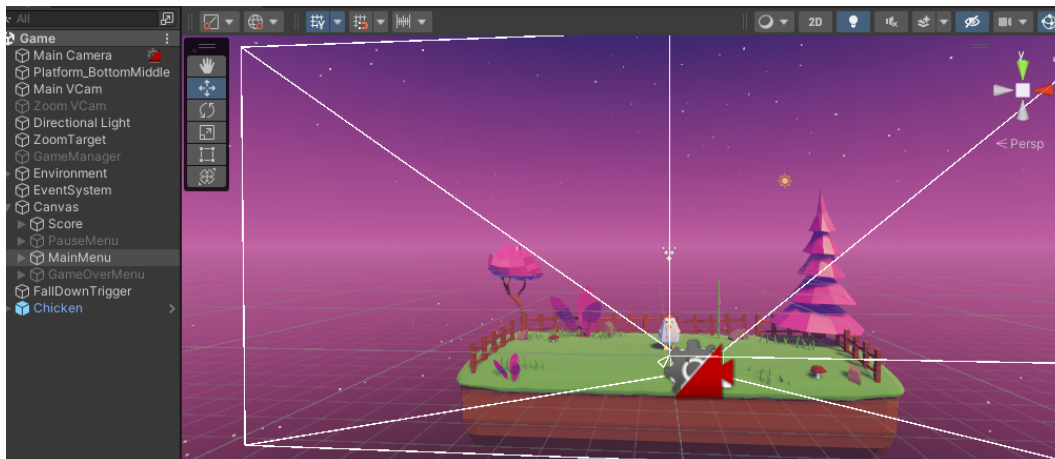


Figura 3: Imagem dos objetos Cenário e Island.

3.1.3 Objeto Hazard

O objeto Hazard, no contexto do jogo é o nosso obstáculo, e seu propósito é dificultar o acúmulo de pontos do player. Para o processo de criação desse obstáculo, primeiramente criamos um cubo e renomeamos como “Hazard”, resetamos o que estava pré-setado no transform dele, também criamos um materials para ele e adicionamos um Rigidbody. Em “Tag”, nós adicionamos uma tag (addTag) denominada “Hazard” (o uso da TAG pode ser melhor compreendido no script do Player). Em suma, toda vez que o obstáculo colide com o player o jogo acaba, e caso o player desvie do obstáculo e esse obstáculo atingir nosso solo, então o Hazard é “desintegrado” por meio do sistema de partículas proveniente do Unity. Posteriormente, prezando a estética do jogo, utilizamos um asset disponibilizado gratuitamente na internet e substituímos pelo cubo inicial conservando todas suas características de escala e outras propriedades.

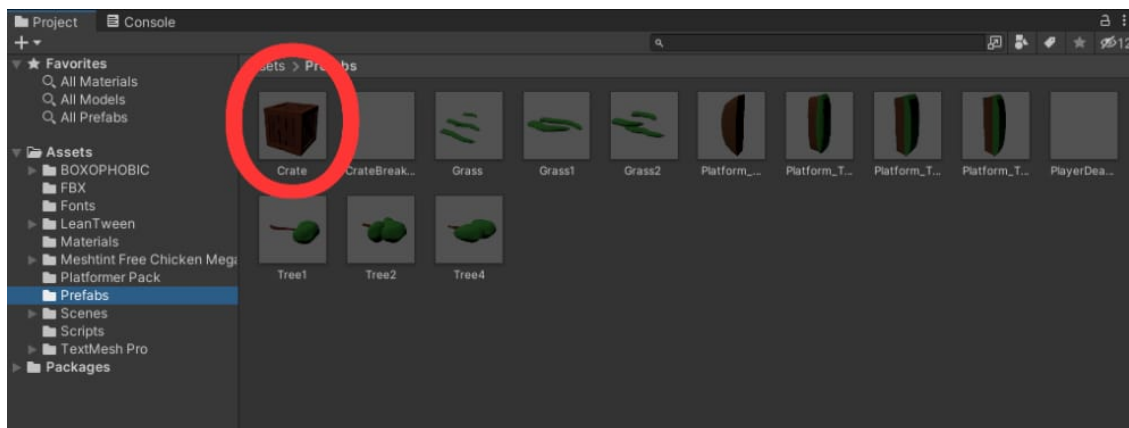


Figura 4:: Imagem demonstrando o objeto Hazard (elemento circulado em vermelho).

3.1.4 Objeto Cenário

O objeto cenário contempla o total do jogo, isto é, é a junção de todos os objetos, assets e outros responsáveis pela estética do cenário.

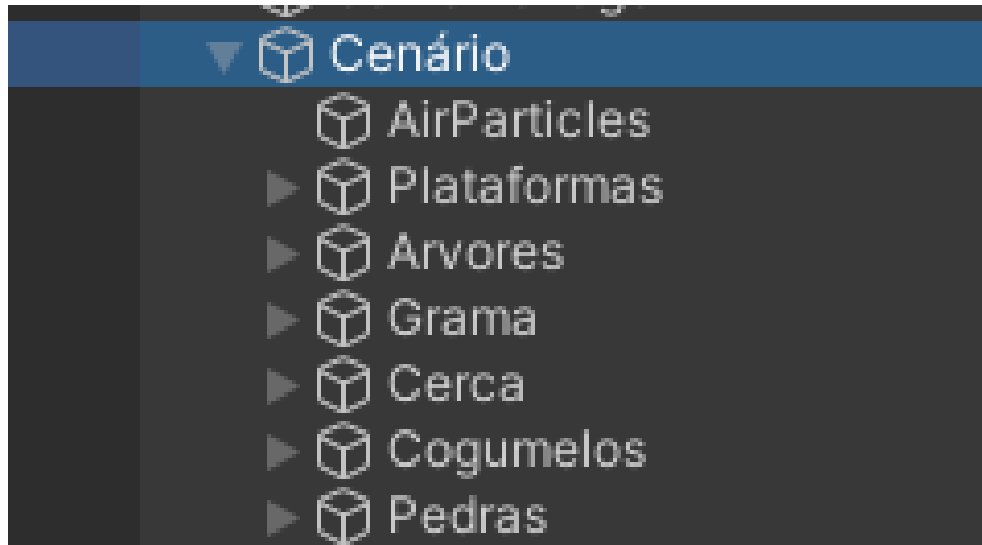


Figura 5: Imagem demonstrando os assets do Cenário.

3.2 Scripts

Foram utilizados sete scripts durante o desenvolvimento de todo o nosso jogo, estes scripts por sua vez são: script `AutoDestroyer.cs`; script `GameManager.cs`; script `GamerOverMenu.cs`; script `Hazard.cs`; o script `Lerp.cs`; o script `MainMenu.cs` e o script `Player.cs`. Conforme é exibido na figura 5.

3.2.1 Player

Neste primeiro script, tem-se toda a formulação, definição de funções, métodos, movimentações e a ação de fim do jogo referentes à galinha (player) controlada pelo usuário. A figura 6 logo abaixo retrata o script inteiramente.


```

MainMenus.cs  C# Player.cs x
Scripts > C# Player.cs
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Cinemachine;
5
6  public class Player : MonoBehaviour
7  {
8      [SerializeField]
9      private float forceMultiplier = 3f;
10     [SerializeField]
11     private float maximumVelocity = 3f;
12     [SerializeField]
13     private ParticleSystem deathParticles;
14
15     private Rigidbody rb;
16     private CinemachineImpulseSource cinemachineImpulseSource;
17
18     void Awake()
19     {
20         rb = GetComponent<Rigidbody>();
21         cinemachineImpulseSource = GetComponent<CinemachineImpulseSource>();
22     }
23
24     void Update()
25     {
26         if (GameManager.Instance == null)
27         {
28             return;
29         }
30
31         float horizontalInput = 0;
32
33         if (Input.GetMouseButton(0))
34         {
35             var center = Screen.width / 2;
36             var mousePosition = Input.mousePosition;
37             if (mousePosition.x > center)
38             {
39                 horizontalInput = 1;
40             }
41             else if (mousePosition.x < center)
42             {
43                 horizontalInput = -1;
44             }
45         }
46         else
47         {
48             horizontalInput = Input.GetAxis("Horizontal");
49         }
50
51         if (rb.velocity.magnitude <= maximumVelocity)
52         {
53             rb.AddForce(new Vector3(horizontalInput * forceMultiplier * Time.deltaTime, 0, 0));
54         }
55     }
56
57     private void OnEnable()
58     {
59         transform.position = new Vector3(0, 0.75f, 0);
60         transform.rotation = Quaternion.identity;
61         rb.velocity = Vector3.zero;
62     }
63
64     private void OnCollisionEnter(Collision collision)
65     {
66         if (collision.gameObject.CompareTag("Hazard"))
67         {
68             GameOver();
69
70             Instantiate(deathParticles, transform.position, Quaternion.identity);
71             cinemachineImpulseSource.GenerateImpulse();
72         }
73
74         if (collision.gameObject.CompareTag("FallDown"))
75         {
76             GameOver();
77         }
78     }
79

```

```

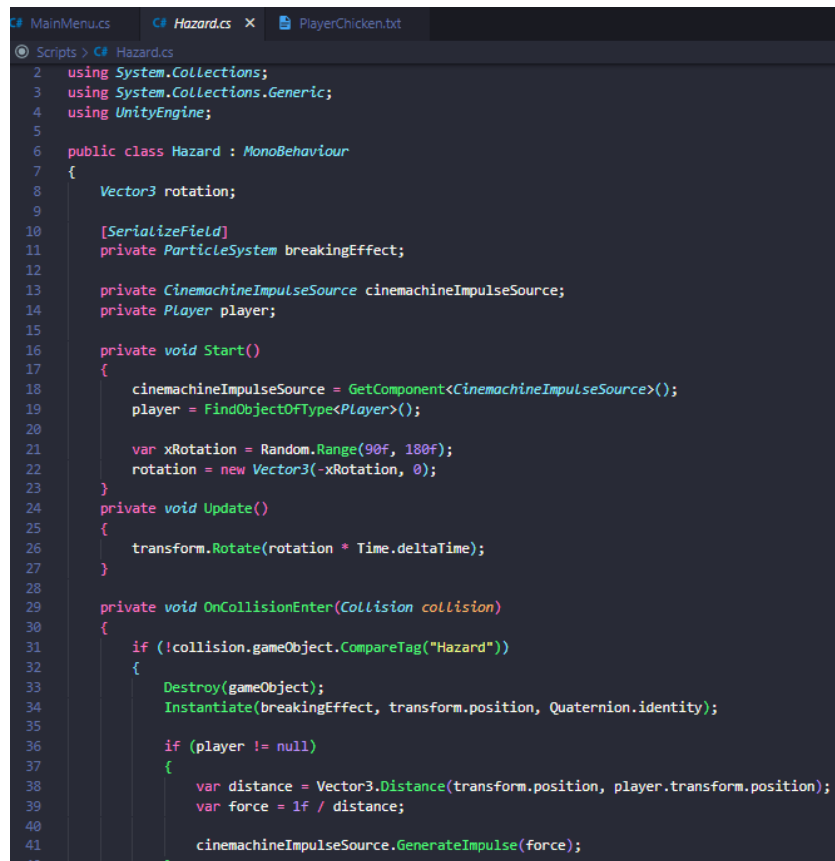
79
80     private void OnTriggerExit(Collider other)
81     {
82         if (other.CompareTag("FallDown"))
83         {
84             GameOver();
85         }
86     }
87
88     private void GameOver()
89     {
90         GameManager.Instance.GameOver();
91
92         gameObject.SetActive(false);
93     }
94 }
95

```

Figura 6: Script player.cs

Basicamente neste script, temos: a definição do objeto Player como um rigidbody, que por padrão vem como um componente 3D; atribuição de duas teclas para a movimentação horizontal do player definidos na função Update(); definição da velocidade do objeto, das partículas geradas no fim de jogo e da chamada para a função de fim de jogo (GameOver). Lembrando que o fim de jogo é chamado no instante em que o objeto Hazard colide com o objeto Player. Por fim, neste script também é chamado a função de autodestruição do player, porém a função só é descrita no script AutoDestroyer.cs

3.2.2 Hazard



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5
6 public class Hazard : MonoBehaviour
7 {
8     Vector3 rotation;
9
10    [SerializeField]
11    private ParticleSystem breakingEffect;
12
13    private CinemachineImpulseSource cinemachineImpulseSource;
14    private Player player;
15
16    private void Start()
17    {
18        cinemachineImpulseSource = GetComponent<CinemachineImpulseSource>();
19        player = FindObjectOfType<Player>();
20
21        var xRotation = Random.Range(90f, 180f);
22        rotation = new Vector3(-xRotation, 0);
23    }
24    private void Update()
25    {
26        transform.Rotate(rotation * Time.deltaTime);
27    }
28
29    private void OnCollisionEnter(Collision collision)
30    {
31        if (!collision.gameObject.CompareTag("Hazard"))
32        {
33            Destroy(gameObject);
34            Instantiate(breakingEffect, transform.position, Quaternion.identity);
35
36            if (player != null)
37            {
38                var distance = Vector3.Distance(transform.position, player.transform.position);
39                var force = 1f / distance;
40
41                cinemachineImpulseSource.GenerateImpulse(force);
42            }
43        }
44    }
45 }
```

Figura 7: Script Hazard.cs

Na figura de número 7 temos o script que será responsável por definir o objeto Hazard. Além de apenas definir o objeto, além de instanciar o mesmo, também é configurado o evento de rotação no objeto, esta transformação geométrica é definida para ocorrer de maneira aleatória, assim como a posição na qual o objeto surge. A velocidade do objeto foi outro ponto definido neste script, a velocidade de cada objeto hazard é aleatória, portanto varia de mais rápida a mais lenta arbitrariamente, porém sempre obedecendo um teto máximo, isto é, a apesar de aleatória, estabelecemos um limite para o qual não queremos que a velocidade do objeto ultrapasse.

Por fim também é estabelecido o evento de Fim de Jogo (GameOver) ao haver a colisão entre o objeto Hazard e o objeto Player e a chamada da função de auto destruição ao haver a colisão entre o objeto Hazard e o solo.

3.2.3 Lerp

Na figura 8 temos o script responsável pela interpolação entre pontos pré-estabelecidos. Basicamente, este script irá definir um padrão de transformações geométricas de um objeto entre um ponto e outro, neste projeto este script foi utilizado para definir as rotações do objeto hazard. O ponto inicial do método lerp seria acima do cenário principal e o ponto final seria o solo, portanto são definidos as rotações do objeto e a velocidade do movimento dele entre esses pontos.

```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Lerp : MonoBehaviour
6  {
7      public Transform A;
8      public Transform B;
9
10     [Range(0f, 1f)]
11     public float t;
12
13     public Color ColorA;
14     public Color ColorB;
15
16     public float duration = 10f;
17     private float elapsedTime = 0;
18
19     private MeshRenderer meshRenderer;
20
21     private void Start()
22     {
23         meshRenderer = GetComponent<MeshRenderer>();
24     }
25
26     void Update()
27     {
28         elapsedTime += Time.deltaTime;
29         t = elapsedTime / duration;
30
31         transform.position = Vector3.Lerp(A.position, B.position, t);
32         transform.rotation = Quaternion.Lerp(A.rotation, B.rotation, t);
33
34         meshRenderer.material.color = Color.Lerp(ColorA, ColorB, t);
35     }
36 }
37

```

Figura 8: Script Lerp.cs

3.2.4 AutoDestroyer

O objetivo deste script é fazer com que o gameObject seja destruído depois de um certo tempo, no nosso caso, faz com que as partículas desapareçam gradualmente. A construção desse Script é bem simples, primeiramente temos a criação de um campo público do tipo float chamado "delay", no método "Start()" utilizamos o "Destroy()" e passamos como parâmetro o objeto a ser destruído (partículas) e um tempo antes de destruir o objeto (delay) que agora pode ser modificado pela própria interface do Unity.

```

C# AutoDestroyer.cs X
Scripts > C# AutoDestroyer.cs
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class AutoDestroyer : MonoBehaviour
6 {
7     [SerializeField]
8     private float delay;
9
10    void Start()
11    {
12        Destroy(gameObject, delay);
13    }
14 }
15

```

Figura 9: Script AutoDestroyer.cs

3.2.5 MainMenu

O script MainMenu trata do menu do jogo e também da pontuação, por meio dele temos acesso ao início do jogo.

```

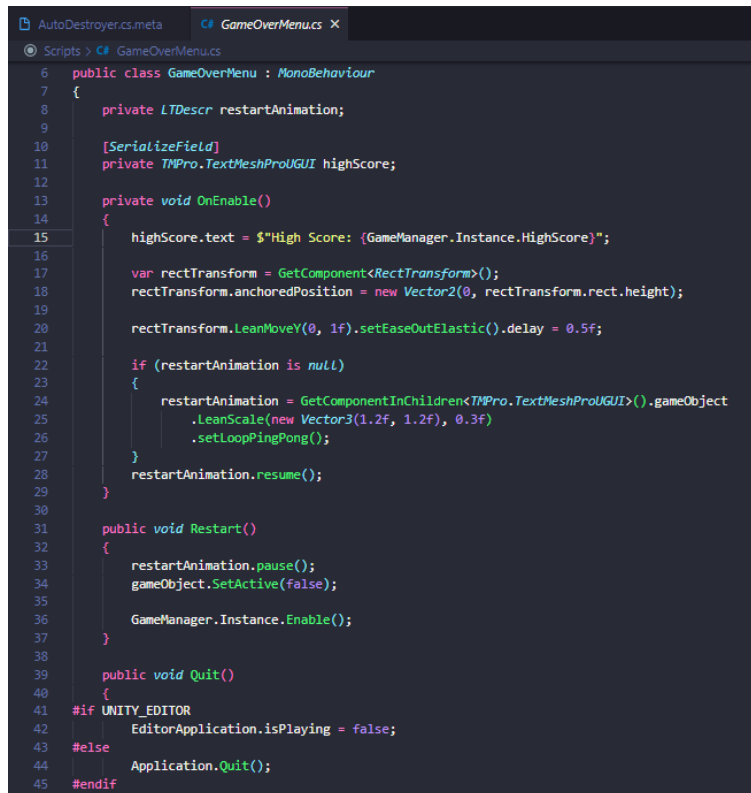
C# MainMenu.cs X
Scripts > C# MainMenu.cs
1 using System;
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5
6 public class MainMenu : MonoBehaviour
7 {
8     [SerializeField]
9     private GameManager gameManager;
10
11    [SerializeField]
12    private RectTransform scoreRectTransform;
13
14    private void Start()
15    {
16        scoreRectTransform.anchoredPosition = new Vector2(
17            scoreRectTransform.anchoredPosition.x, 0);
18
19        GetComponentInChildren<TMPPro.TextMeshProUGUI>().gameObject
20            .LeanScale(new Vector3(1.2f, 1.2f), 0.3f)
21            .setLoopPingPong();
22    }
23
24    public void Play()
25    {
26        GetComponent<CanvasGroup>().LeanAlpha(0, 0.2f)
27            .setOnComplete(OnComplete);
28    }
29
30    private void OnComplete()
31    {
32        scoreRectTransform.LeanMoveY(-72f, 0.75f).setEaseOutBounce();
33
34        gameManager.Enable();
35        Destroy(gameObject);
36    }
37 }
38

```

Figura 10: Script MainMenu.cs

3.2.6 GameOverMenu

Quanto ao script GameOverMenu, no instante em que a classe GameOverMenu é chamada ela salva o valor do highscore no texto da variável highscore para ser atribuída durante a chamada no GameManager; reinicia a animação do jogo exibindo a tela com o highscore, um botão de reiniciar e um botão para sair do jogo e gera a função reiniciar e a função quit; o botão de reiniciar retorna a animação para o script MainMenu e o botão sair ativa a função Quit() que exerce a função de sair do jogo.



```
6 public class GameOverMenu : MonoBehaviour
7 {
8     private LDescr restartAnimation;
9
10    [SerializeField]
11    private TMPro.TextMeshProUGUI highScore;
12
13    private void OnEnable()
14    {
15        highScore.text = $"High Score: {GameManager.Instance.HighScore}";
16
17        var rectTransform = GetComponent<RectTransform>();
18        rectTransform.anchoredPosition = new Vector2(0, rectTransform.rect.height);
19
20        rectTransform.leanMoveY(0, 1f).setEaseOutElastic().delay = 0.5f;
21
22        if (restartAnimation is null)
23        {
24            restartAnimation = GetComponentInChildren<TMPPro.TextMeshProUGUI>().gameObject
25                .LeanScale(new Vector3(1.2f, 1.2f), 0.3f)
26                .setLoopPingPong();
27        }
28        restartAnimation.resume();
29    }
30
31    public void Restart()
32    {
33        restartAnimation.pause();
34        gameObject.SetActive(false);
35
36        GameManager.Instance.Enable();
37    }
38
39    public void Quit()
40    {
41        #if UNITY_EDITOR
42            EditorApplication.isPlaying = false;
43        #else
44            Application.Quit();
45        #endif
46    }
47 }
```

Figura 11: Script GameOverMenu.cs

3.2.7 GameManager

Como o último script temos o GameManager.cs que faz exatamente o que o nome diz, gerencia o jogo como um todo. As funcionalidades implementadas por este script são as seguintes: Definir, iniciar e incrementar o highscore (sistema de pontuação) do jogo; gerar (spaw) os objetos do tipo hazard a partir da classe hazard; definição das funções de Pause e unPause para pausar e despausar o jogo e chamar a classe Game Over, definida no script GameOver.cs, exibindo assim o valor do highscore alcançado na tela.

```

GameManager.cs X
Scripts > C# GameManager.cs
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4 using UnityEngine.UI;
5
6 public class GameManager : MonoBehaviour
7 {
8     [SerializeField]
9     private GameObject hazardPrefab;
10    [SerializeField]
11    private int maxHazardsToSpawn = 3;
12
13    [SerializeField]
14    private TMPro.TextMeshProUGUI scoreText;
15    [SerializeField]
16    private Image backgroundMenu;
17
18    [SerializeField]
19    private GameObject player;
20
21    [SerializeField]
22    private GameObject mainVCam;
23    [SerializeField]
24    private GameObject zoomVCam;
25
26    [SerializeField]
27    private GameObject gameOverMenu;
28
29    private int highScore;
30    private int score;
31    private float timer;
32    private Coroutine hazardsCoroutine;
33
34    private bool gameOver;
35
36    private static GameManager instance;
37
38    private const string HighScorePreferenceKey = "HighScore";
39
40    public static GameManager Instance => instance;
41
42    public int HighScore => highScore;
43
44    void Start()
45    {
46        instance = this;
47        highScore = PlayerPrefs.GetInt(HighScorePreferenceKey);
48    }
49
50    private void OnEnable()
51    {
52        player.SetActive(true);
53
54        zoomVCam.SetActive(false);
55        mainVCam.SetActive(true);
56
57        gameOver = false;
58        scoreText.text = "0";
59        score = 0;
60        timer = 0;
61
62        hazardsCoroutine = StartCoroutine(SpawnHazards());
63    }
64
65    private void Update()
66    {
67        if (Input.GetKeyDown(KeyCode.Escape))
68        {
69            if (Time.timeScale == 0)
70            {
71                UnPause();
72            }
73            if (Time.timeScale == 1)
74            {
75                Pause();
76            }
77        }
78    }

```

```

78
79     if (gameOver)
80         return;
81
82     timer += Time.deltaTime;
83
84     if (timer >= 1f)
85     {
86         score++;
87         scoreText.text = score.ToString();
88
89         timer = 0;
90     }
91 }
92
93 private void Pause()
94 {
95     LeanTween.value(1, 0, 0.75f)
96         .setOnUpdate(SetTimeScale)
97         .setIgnoreTimeScale(true);
98     backgroundMenu.gameObject.SetActive(true);
99 }
100
101 private void UnPause()
102 {
103     LeanTween.value(0, 1, 0.75f)
104         .setOnUpdate(SetTimeScale)
105         .setIgnoreTimeScale(true);
106     backgroundMenu.gameObject.SetActive(false);
107 }
108
109 private void SetTimeScale(float value)
110 {
111     Time.timeScale = value;
112     Time.fixedDeltaTime = 0.02f * value;
113 }
114
115 private IEnumerator SpawnHazards()
116 {
117     var hazardToSpawn = Random.Range(1, maxHazardsToSpawn);
118
119     for (int i = 0; i < hazardToSpawn; i++)
120     {
121         var x = Random.Range(-7, 7);
122         var drag = Random.Range(0f, 2f);
123
124         var hazard = Instantiate(hazardPrefab, new Vector3(x, 11, 0), Quaternion.identity);
125         hazard.GetComponent<Rigidbody>().drag = drag;
126     }
127
128     var timeToWait = Random.Range(0.5f, 1.5f);
129     yield return new WaitForSeconds(timeToWait);
130
131     yield return SpawnHazards();
132 }
133
134 public void GameOver()
135 {
136     StopCoroutine(hazardsCoroutine);
137     gameOver = true;
138
139     if (Time.timeScale < 1)
140     {
141         UnPause();
142     }
143
144     if (score > highScore)
145     {
146         highScore = score;
147         PlayerPrefs.SetInt(HighScorePreferenceKey, highScore);
148     }
149
150     mainVCam.SetActive(false);
151     zoomVCam.SetActive(true);
152
153     gameObject.SetActive(false);
154     gameOverMenu.SetActive(true);
155 }
156
157
158 public void Enable()
159 {
160     gameObject.SetActive(true);
161 }
162

```

Figura 12: Script gameManager.cs

4 Técnicas Utilizadas nos Objetos

4.1 Rotação

A técnica de rotação é aplicada principalmente nos obstáculos (Hazard), que para não serem tão estáticos, rotacionam todos no mesmo sentido, o que dá impressão de "queda-livre" desses objetos.

4.2 Translação

Apesar de simples, o jogo conta com bastante movimentação, seja por parte do player e algumas características próprias do mesmo (ciscar, bater asa, etc) ou por parte dos caixotes quem caem do céu. No script Hazard e também no script Player é possível identificar trechos que remetem à expressões de movimentos e suas variáveis. Ademais, na aba lateral do Unity, já mencionada anteriormente, podemos acompanhar o status de movimentação dos objetos do jogo.

4.3 Escala

No jogo, as mudanças de escala foram essenciais para compor o cenário, por exemplo, temos diferentes tamanho de pedras, gramas, árvores, entre outros. Ademais, ao utilizar o sistema de partículas no player e nos obstáculos, é perceptível o uso da escala, uma vez que esses objetos se desintegram em partículas menores caso colidam.

4.4 Iluminação, Sombra

Tratando-se da iluminação, por meio do unity adicionamos uma luz ambiente (Directional Light) que simula uma luz solar podendo ser colocada em qualquer posição do nosso ambiente, desta forma garantimos a iluminação do ambiente e a cor dos objetos. Além disso, utilizamos modelos globais de rendering, em que temos as sombras dos objetos, suas reflexões, etc. Quanto a geometria da fonte de luz, temos a emissão de luz em todas as direções uniformemente, portanto, pontual. Para as sombras temos a técnica de Real-Time Shadow Volumes em que a sombra é calculada a cada frame do jogo, como não temos muitos objetos que se movem, então não temos grandes problemas.

4.5 Algoritmos pontuais e geométricos para visibilidade

Quanto ao Sistema de Coordenadas de Câmera, definimos a origem da nossa câmera um pouco a cima e direcionada de frente ao ambiente para obtermos uma projeção de perspectiva geral do cenário. Além disso, é utilizada a ideia dos algoritmos de culling, pois temos objetos volumétricos, sendo assim, não visualizamos todas as suas faces, somente as visíveis pela câmera, eliminando as faces traseiras dos objetos, também temos, o Z-buffer e occlusion culling, eliminando os objetos que ficam a frente de outros na câmera de visualização.

5 Considerações Finais

Por meio deste trabalho fomos capazes de fortalecer nosso conhecimento com a ferramenta Unity, além de implementar um ambiente 3D interativo e que contemplasse tudo o que nos foi proposto. Todos os resultados obtidos são fruto de total empenho, estudo e discussões da equipe para com a disciplina de Computação Gráfica. Ademais, a busca por conhecimento em outras fontes de informação nos instigou a entender mais a fundo sobre aplicações 3D.