Escola Superior de Tecnologia e Gestão

Licenciatura em Engenharia Informática

Escola Superior de Tecnologia e Gestão

Instituto Politécnico de Viana do Castelo

Tecnologias Multimédia

2023/2024

Minigolf 3D

João Lima Araújo Nº 24682

Escola Superior de Tecnologia e Gestão

Índice

Introdução

O jogo 'Minigolf 3D' foi desenvolvido no âmbito da unidade curricular Tecnologias Multimédia do curso de Engenharia Informática do Instituto Politécnico de Viana do Castelo. Este jogo foi criado em Unity. O objetivo deste projeto é aplicar os conhecimentos sobre Unity e C# adquiridos ao longo das aulas práticas da unidade curricular. Ao longo do relatório, vão ser apresentados os detalhes do desenvolvimento do jogo, desde os *scripts*, os *GameObjects*, os níveis do jogo entre outros, até à conceção final do jogo.

Objetivo e instruções do jogo

Tal como qualquer jogo de Minigolf, o objetivo deste jogo é colocar a bola no buraco. Para isso o jogador deverá de utilizar o rato para apontar em que direção quer que a bola seja lançada, através do auxílio duma seta. A potência com que a bola é disparada é definida pela distância entre a bola e a posição do rato. Quando a bola entra no buraco, o jogador passa para o nível seguinte e assim sucessivamente até chegar ao fim do nível. Em cada nível, são contados o número de tacadas feitas pelo jogador até conseguir colocar a bola no buraco.

Desenvolvimento do jogo

Bola de golf

A bola de golf é a peça fundamental do jogo pois é em torno da mesma que o jogo se desenvolve, isto é, associada à bola existe um script responsável pelo comportamento da bola que permite que o jogo se desenrole dependentemente dos comportamentos da mesma.

Script BallControll

void Start ()

```
void Start()
{
    rb.maxAngularVelocity = 1000;
    isAiming = false;
    powerSlider.gameObject.SetActive(false);
    arrow.SetActive(false);
}
```

Figura 1 - Método Start

Esta função é chamada antes da primeira atualização do jogo, ou seja, antes da função 'Update ()' ser chamada e serve para inicializar as variáveis e configurar objetos. A linha 'rb.maxAngularVelocity = 1000;' controla a velocidade máxima de rotação da bola; variável booleana 'isAiming' é inicializada como falsa e os objetos 'powerSlide' e 'arrow' são inativados inicialmente.

void Update ()

```
void Update()
{
    if (rb.velocity.magnitude < stopForce)
    {
        ProcessAim();

        if (Input.GetMouseButtonDown(0))
        {
            if (isIdle) isAiming = true;
        }

        if (Input.GetMouseButtonUp(0))
        {
            isShooting = true;
        }
    }
    shotsText.text = "SHOTS: " + GetNumberOfShots().ToString();
}
```

Figura 2 - Método Update

Quando a velocidade da bola é menor que a *stopForce* é chamada a função *ProcessAim ()*. Seguidamente, verifica se o botão do rato foi pressionado (*if (Input.GetMouseButtonDown(0))*) e altera a variável *isAiming* para verdade. Quando o botão do rato deixa de ser pressionado a variável booleana *isShooting* altera para verdade. Esta verificação é importante pois vai permitir que, na função *FixedUpdate ()*, seja chamada a função *Shoot ()* que é responsável pela execução da tacada sobre a bola. Por fim, é atualizada na variável *shotsText* o número de tacadas.

void FixedUpdate()

```
void FixedUpdate()
{
    if (rb.velocity.magnitude < stopForce)
    {
        Stop();
    }
    if (isShooting && worldPosition.HasValue)
    {
        Shoot(worldPosition.Value);
        numberOfShots++;
        isShooting = false;
    }
}
```

Figura 6 - Método FixedUpdate

O evento *FixedUpdate ()* é chamado a cada *frame* por segundo. Neste evento, primeiramente é verificado se a velocidade da bola é menor que a *stopForce,* caso se verifique é chamada a função *Stop ()*, que tal como o nome indica é responsável por fazer parar a bola.

```
private void Stop()
{
    rb.velocity = Vector3.zero;
    rb.angularVelocity = Vector3.zero;

    isIdle = true;
    powerSlider.gameObject.SetActive(false);
}
```

Figura 7 - Método Start

Nesta função, o *powerSlide* é desaparece do ecrã e como a bola está parada, a variável *isIdle* passa a ser verdade.

Continuando no *FixedUpdate ()*, caso não se confirme a primeira condição é feita outra verificação, mas desta vez é verificado se a variável *isShooting* é verdadeira e se *worldPosition* tem algum valor. À variável *worldPosition* é atribuído valor na função *ProcessAim ()* que foi chamada anteriormente no evento *Update ()*. O valor é atribuído à variável *worldPosition* caso, quando o jogador clica no botão do rato, este atinja um objeto do ecrã do jogo. Esta atribuição de valor e verificação é feita pela função *CastMouseClickRay ()*. Se não atingir, o método retorna imediatamente e não é executado o restante código. Ainda no *ProcessAim ()*, no caso de *worldPosition* ter valor, é chamado o método *UpdateArrowDirection ()*, que atualiza a direção e a rotação da seta associada à bola e determina a potência com que esta vai ser disparada.

```
private void ProcessAim()
{
    if (isAiming && !isIdle) return;

    worldPosition = CastMouseClickRay();

    if (!worldPosition.HasValue) return;

    UpdateArrowDirection();
}

1 reference
private void UpdateArrowDirection()
{
    Vector3 lineDirection = worldPosition.Value - transform.position;

    arrow.SetActive(true);
    powerSlider.gameObject.SetActive(true);

    Quaternion rotation = Quaternion.LookRotation(lineDirection.normalized, Vector3.up);
    Vector3 eulerAngles = rotation.eulerAngles;
    eulerAngles.x = 90;
    arrow.transform.rotation = Quaternion.Euler(eulerAngles);

    float power = Mathf.Clamp01(lineDirection.magnitude / maxLineLength);
    powerSlider.value = power;
}

1 reference
private Vector3? CastMouseClickRay()
{
    Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);
    RaycastHit hit;

    if (Physics.Raycast(ray, out hit))
    {
        return hit.point;
    }
    return null;
}
```

Figura 8 - Método ProcessAim

Voltando ao *FixedUpdate ()*, no caso de se verificar a segunda condição, é chamado o método *Shoot ()* que é responsável por fazer com que a bola seja disparada. Este método tem como parâmetro um *point* do tipo *Vector3*. Neste caso, vai ser passado como parâmetro o valor do *worldPosition*. Além disso são contados o número de tacadas e a variável *isShooting* passa para falso.

void Shoot ()

```
private void Shoot(Vector3 point)
{
    isAiming = false;
    arrow.SetActive(false);

    Vector3 horizontalWorldPosition = new Vector3(point.x, transform.position.y, point.z);
    Vector3 direction = (transform.position - horizontalWorldPosition).normalized;

    float distance = Vector3.Distance(horizontalWorldPosition, transform.position);
    rb.AddForce(-direction * distance * shootForce);
}
```

Figura 9 - Método Shoot

A função *Shoot(Vector3 point)* é responsável por disparar a bola na direção do ponto de destino especificado. Após o disparo, a função desativa a seta de mira associada e atualiza a variável *isAiming* para indicar que o jogador não está a mirar. Para determinar a direção do disparo, a função calcula um vetor apontando da posição atual da bola para o ponto de destino no plano horizontal, mantendo a altura da bola. Em seguida, calcula a distância entre a posição atual da bola e o ponto de destino. Usando essa direção e distância, a função aplica uma força à bola na direção oposta à direção calculada, multiplicada pela distância e pela força de disparo, garantindo que a bola seja lançada na direção do ponto de destino com a força necessária para alcançá-lo.

void OnTriggerEnter (Collider other)

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag(floorTag))
    {
        StartCoroutine(RestartGameCoroutine());
    }
    if (other.CompareTag(holeTag))
    {
        StartCoroutine(LoadNextScene());
    }
```

Figura 10 - Método OnTriggerEnter

A função *OnTriggerEnter(Collider other)* é responsável por detetar quando a bola colide com outro objeto no jogo. Quando isso acontece, a função verifica se o objeto colidido tem uma *tag* específica. Se o objeto tiver a *tag* correspondente à variável *floorTag*, a função inicia uma *coroutine* chamada *RestartGameCoroutine()*. Esta *coroutine* reinicia o jogo 2 segundos, o que é útil quando a bola de golfe cai fora do campo de jogo. Por outro lado, se o objeto colidido tiver a *tag: holeTag*, a função inicia uma *coroutine* chamada *LoadNextScene()*. Esta *coroutine* carrega a próxima cena do jogo após 2 segundos igualmente, indicando que o jogador conseguiu fazer a bola entrar no buraco, avançando assim para o próximo nível do jogo. Essa função é essencial para controlar os eventos que ocorrem quando a bola colide com certos objetos no jogo e permite uma transição suave entre os diferentes estados do jogo.

As variáveis de booleanas desempenham um papel crucial no controlo do fluxo de execução. Estas são particularmente importantes para determinar o estado do jogo e controlar o comportamento da bola.

Script MainMenu

Esta script é responsável por controlar o menu principal do jogo e gerir as transições entre as diferentes cenas. O *MainMenu* contém um conjunto de métodos que são executados por botões ou outros elementos da interface do jogo. Estes métodos possuem nomes que descrevem exatamente quais as suas funções quando executados.

```
public class MainMenu : MonoBehaviour
{
    0 references
    public void PlayGame()
    {
        SceneManager.LoadSceneAsync(1);
    }

    0 references
    public void QuitGame()
    {
        Application.Quit();
    }

    0 references
    public void LoadLevels()
    {
        SceneManager.LoadSceneAsync(5);
    }
    0 references
    public void LoadMainMenu()
    {
        SceneManager.LoadSceneAsync(0);
    }

    0 references
    public void LoadLevel1()
    {
        SceneManager.LoadSceneAsync(1);
    }

    0 references
    public void LoadLevel2()
    {
        SceneManager.LoadSceneAsync(2);
    }

    0 references
    public void LoadLevel3()
    {
        SceneManager.LoadSceneAsync(3);
    }

    0 references
    public void RestartLevel()
    {
        SceneManager.LoadSceneAsync(SceneManager.GetActiveScene().buildIndex);
    }
}
```

Figura 11 - Script MainMenu

Configurações dos GameObjetcs

Bola de golfe

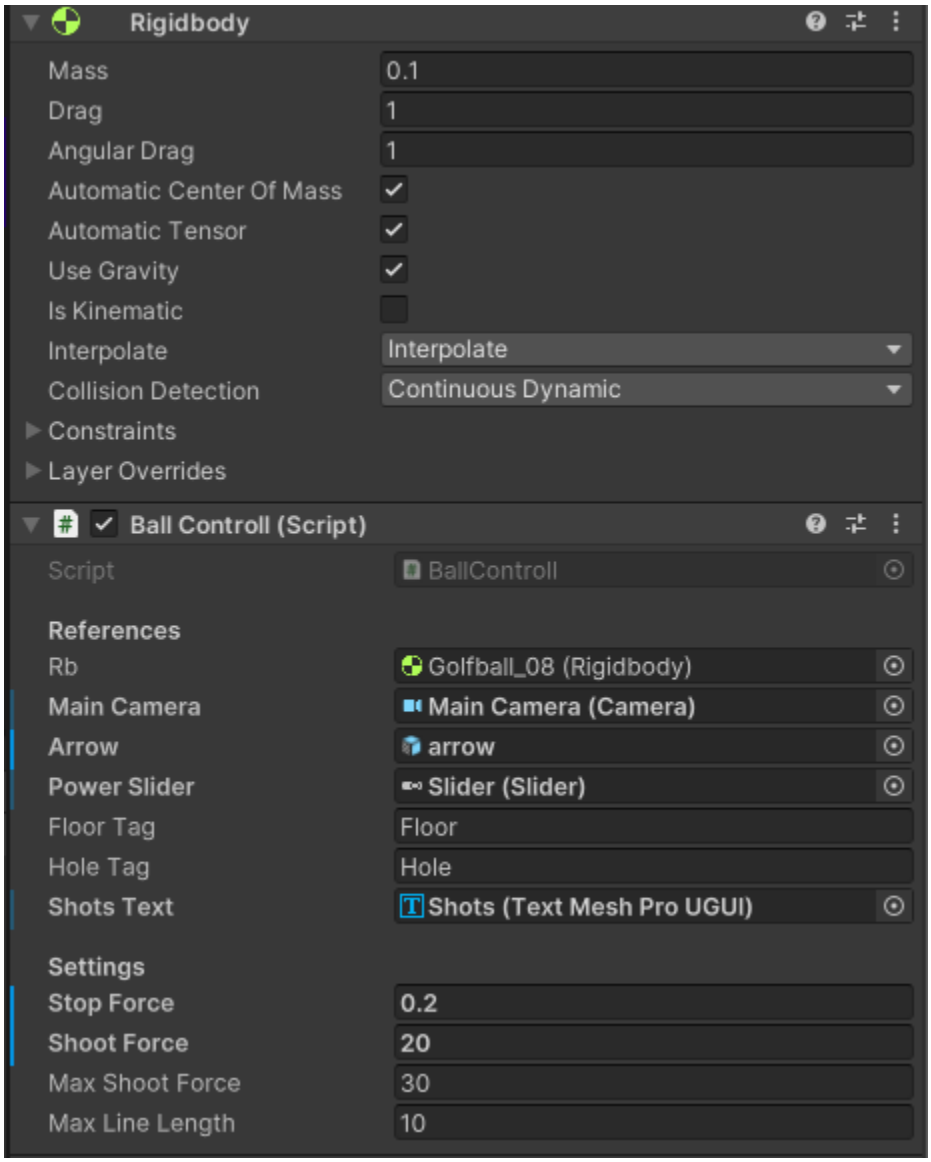


Figura 12 - Inspector da Bola de Golfe

Câmara principal

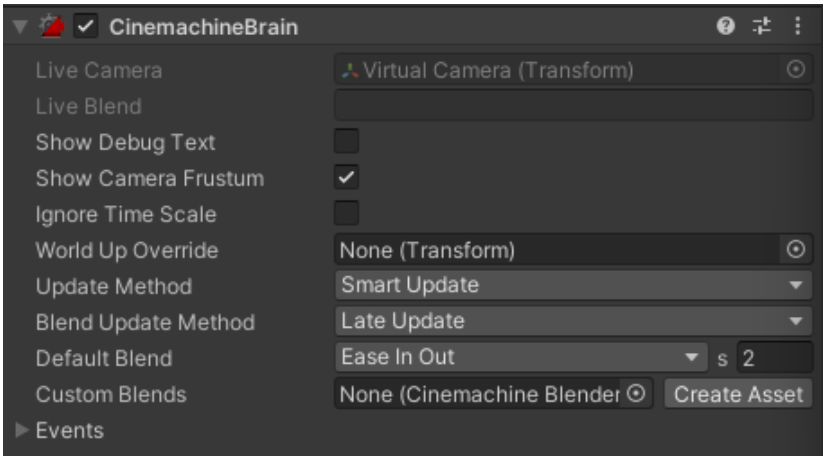


Figura 13 - Inspector da Câmara Principal

Câmara virtual

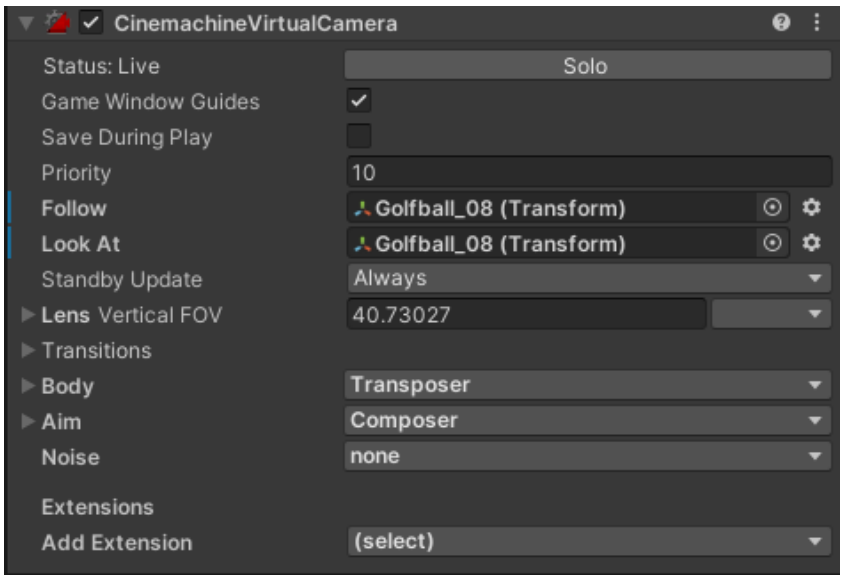


Figura 14 - Inspector da Câmara Virtual

Chão de colisão

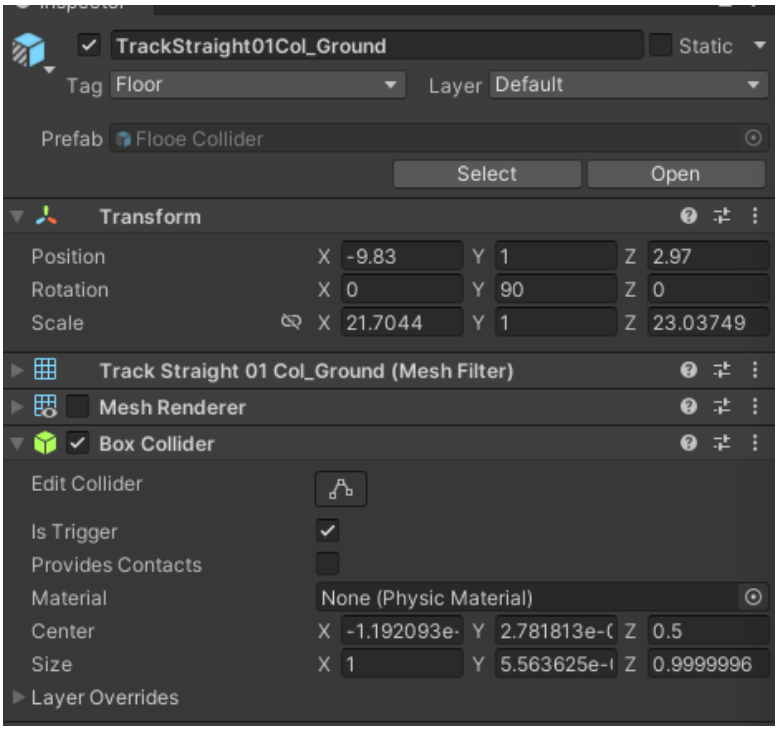


Figura 15 - Inspector do Chão de colisão

Buraco

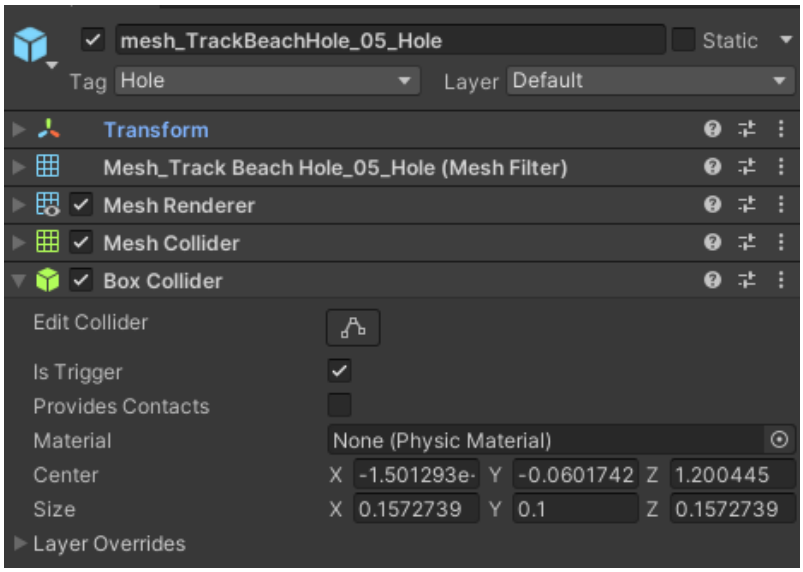


Figura 16 - Inspector do buraco

Níveis e interface

Nível 1

Figura 17 - Nível 1

Nível 2

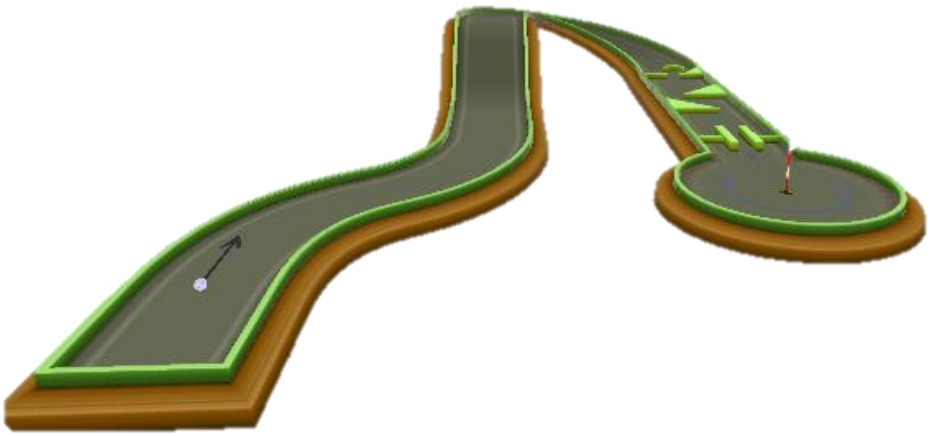


Figura 18 - Nível 2

Nível 3

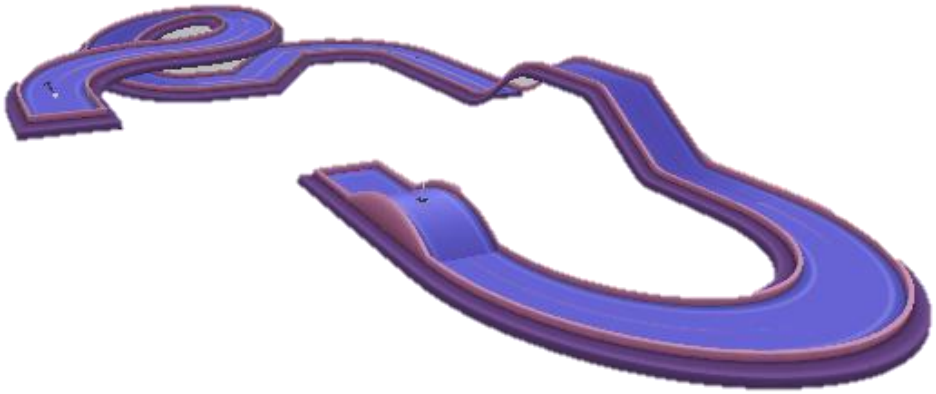
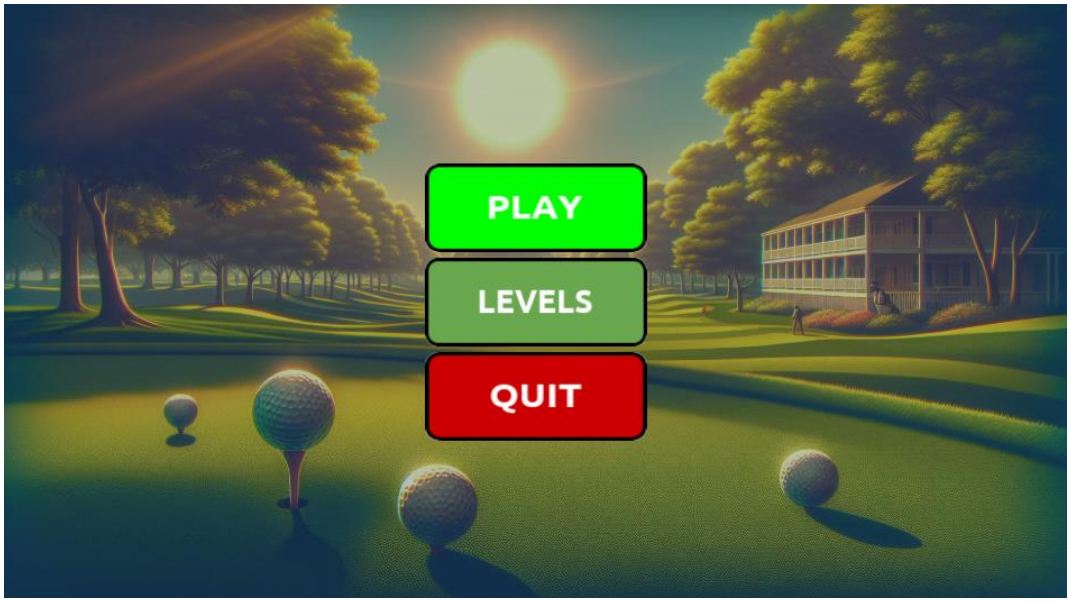
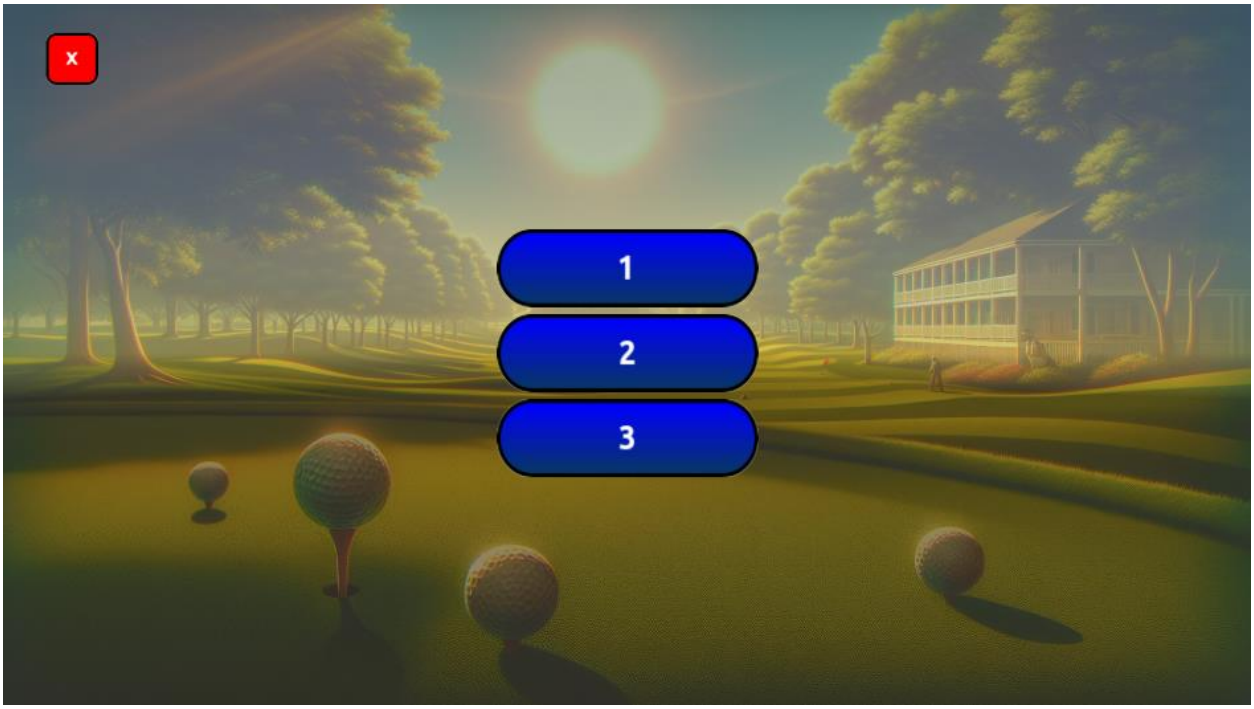


Figura 19 - Nível 3

Menu principal



Níveis



GameOver

Conclusão

Em conclusão, o desenvolvimento do jogo apresentou desafios significativos, mas também oportunidades de aprendizagem. A implementação dos scripts, como *BallControll* e MainMenu, foram fundamentais para criar uma experiência de jogo funcional. O menu principal permitiu uma navegação intuitiva pelo jogo. Além disso, a funcionalidade de reiniciar o nível foi importante para melhorar a experiência do jogador, permitindo que eles tentassem novamente caso falhassem em um desafio.

Um dos desafios mais significativos enfrentados durante o desenvolvimento foi a configuração da seta na bola de golfe. A necessidade de garantir que a seta estivesse corretamente alinhada e visível na posição correta da bola, independentemente da sua orientação e movimento, exigiu uma cuidadosa manipulação das transformações e rotações. Isso incluiu o cálculo preciso da direção do disparo, a rotação da seta para apontar na direção correta e a sincronização dessas operações com o movimento da bola.