Trabalho de Realidade Aumentada

Aluno: Mateus Gonçalez Etto - **RA:** 121022943

Objetivo do Jogo

O objetivo deste jogo é acertar alvos virtuais usando uma arma virtual. Os alvos são gatos que se movimentam em cima de uma mesa (real), sendo que a mesa é detectada por um marcador. Para atirar, é usado outro marcador que representa uma arma. Ao esconder a parte inferior do marcador da arma, um botão virtual é ativado e o tiro é disparado. O jogador possui 5 munições, que ao acabar pode ser recarregado retirando o dedo da parte inferior do marcador da arma. Os alvos (gatos) aparecem na mesa a cada 7 segundos, e podem aparecer até 3 gatos simultaneamente.

Marcadores

Foram usados dois marcadores neste projeto. Um foi usado para encontrar a mesa, e o outro para colocar a arma no ambiente de Realidade Aumentada.

Marcador da mesa

O motivo da escolha deste marcador foi que sua performance é muito boa, muito usada em diversos exemplos na internet.

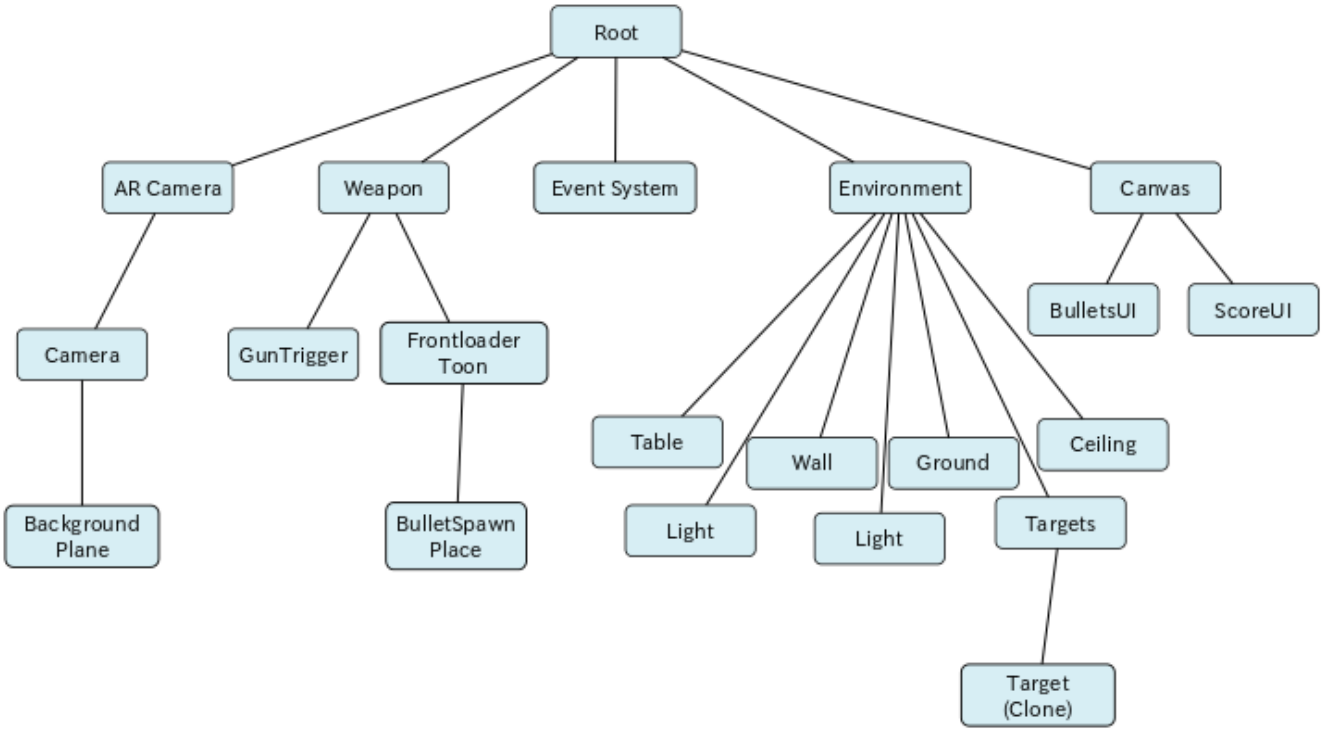
Marcador da arma

O marcador que localiza a posição da arma é o que se encontra na imagem a seguir:

O objetivo de usar tal marcador era testar marcadores feitos à mão, e seu desempenho. Pode-se adiantar que o marcador, mesmo sendo feito à mão, teve um ótimo desempenho, tendo nota 5 (máximo) no site do Vuforia.

Grafo dos Objetos em Cena

A cena do jogo foi criada na seguinte estrutura:



Alguns objetos possuem componentes importantes, que estão citados a seguir:

Weapon

Estes 2 componentes são usados para o rastreamento do marcador da arma:

- Image Target Behaviour
- Default Trackable Event

Estes outros 2 permitem uso do Audio e do algoritmo de disparar tiros:

- AudioSource
- FireBullet.cs

GunTrigger

Seu componente implementa o botão virtual:

- Virtual Button Behaviour

FrontloaderToon

Objeto que representa a arma do jogador:

- Animator

BulletSpawnPlace

Este objeto possui um componente que dispara uma animação de partículas.

- ParticleSystem

Environment

Possui os componentes para rastreamento do marcador da mesa:

- Image Target Behaviour
- Default Trackable Event
- RastreioMarcador.cs

Target

Possui o script que instancia novos alvos.

- TargetSpawner.cs

Target(Clone)

É o objeto com animação de gato que anda pelo mapa e deve ser atingido:

- Animator
- Rigidbody
- BoxCollider
- Cat.cs

BulletUI

Carrega o texto da quantidade de munições disponíveis.

- Text
- Shadow

ScoreUI

Carrega o texto com a pontuação do jogador.

- Text
- Shadow
- ScoreManager.cs

bullet

Não aparece no grafo, mas é instanciado sempre que o botão (virtual) de atirar da arma é acionado.

- SphereCollider
- Rigidbody
- Bullet.cs

Scripts

Foram criados 6 scripts para a criação do jogo. O nome dado a eles é: *Bullet.cs*, *Cat.cs*, *FireBullet.cs*, *RastreioMarcador.cs*, *ScoreManager.cs* e *TargetSpawner.cs*.

O código deles está a seguir:

Bullet.cs

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

public class Bullet : MonoBehaviour {

    // Variável com ponteiro do script que administra a pontuação
    private ScoreManager scoreManager;

    void Start()
    {
        // Buscando o script que administra a pontuação
        scoreManager = GameObject.Find("Canvas/ScoreUI").GetComponent<ScoreManager>();
    }
```

```
    void OnCollisionEnter(Collision col)
    {
        // Se colidir com o alvo
        if (col.gameObject.name == "Target(Clone)")
        {
            // Ganha ponto e destroi ambos os objetos
            scoreManager. AddScore();
            Destroy(gameObject);
            Destroy(col.gameObject, 0.5f);
        }
    }
}
```

Cat.cs

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

public class Cat : MonoBehaviour {

    // Variável para manter um ponteiro ao componente de animação
    private Animator catAnimator;
    // Movimento do gato
    private float move = 0f;
    // Estado do gato (pode ser parado, rotacionando ou em movimento)
    private int catState = 0;

    // Use this for initialization
    void Start ()
    {
        // Buscando o ponteiro do componente de animação
        catAnimator = GetComponent<Animator>();
        // Faz uma decisão a cada 3 segundos
        InvokeRepeating("MakeDecision", 2f, 3f);
    }

    /// <summary>
    /// Random decision
    /// </summary>
    void MakeDecision()
    {
        // Se o gato está tombado, levantá-lo
        if (transform.rotation.x != 0 || transform.rotation.z != 0)
            transform.eulerAngles = new Vector3(0, transform.eulerAngles.y, 0);
        // Decisão sobre o que o gato vai fazer
        catState = Random. Range(0, 6);
```

```
        switch (catState)
        {
            case 0: // parar
                ChangeMovement(0);
                break;
            case 1: // rotacionar
                Rotating(-90);
                break;
            case 2: // rotacionar
                Rotating(90);
                break;
            case 3: // andar
                ChangeMovement(1);
                break;
            case 4: // andar
                ChangeMovement(1);
                break;
            case 5: // andar
                ChangeMovement(1);
                break;
        }
    }

    private void Update()
    {
        // Move o gato para frente se a variável move != 0
        transform.position += transform.forward * move * Time.deltaTime;
        // Se o gato cair, destrui-lo
        if (transform.position.y < - 1)
            Destroy(gameObject);
    }

    void Rotating(float angle)
    {
        // Rotaciona o gato
        catAnimator. SetFloat("Speed", 0);
        transform. Rotate(transform.up * angle );
        move = 0f;
    }

    void ChangeMovement(int moveSpeed)
    {
        // Altera movimento do gato
        catAnimator. SetFloat("Speed", moveSpeed);
        move = moveSpeed * 0.04f;
    }
}
```

FireBullet.cs

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

// Para fazer o som de atirar, é necessário ter o AudioSource
[RequireComponent (typeof(AudioSource ))]
public class FireBullet : MonoBehaviour, IVirtualButtonEventHandler
{
    // Prefab da bala
    public GameObject bulletPrefab;
    // Local onde o tiro será instanciado
    private GameObject bulletSpawnPlace;
    // Botão (virtual) que dispara o tiro
    private GameObject gunTrigger;
    // Animação quando é feito um disparo
    private ParticleSystem fireAnimation;
    // UI informando quantas munições ainda tem
    private Text bulletsNumberUI;
    // Fonte de som para executar som do disparo
    private AudioSource fireSound;

    // Variáveis da arma e do tiro
    private int bulletSpeed = 80;
    private int numberBullets = 5;
    private bool reloading = false;
    private bool firing = false;

    // Use this for initialization
    void Start ()
    {
        // Buscando o Game Object do botão virtual que dispara o tiro
        gunTrigger = transform. FindChild("GunTrigger").gameObject;
        // Buscando o Game Object do local a ser instanciado o tiro
        bulletSpawnPlace = transform. FindChild("FrontloaderToon/bulletSpawnPlace ")
                           .gameObject;
        // Buscando a UI que apresenta o texto de quantas munições a arma tem
        bulletsNumberUI = GameObject. Find("Canvas/BulletsUI ")
                          .GetComponent<Text>();
        // Buscando o Particle System para animação no tiro
        fireAnimation = transform. FindChild("FrontloaderToon/bulletSpawnPlace ")
                        .GetComponent<ParticleSystem>();
        // Ponteiro para o componente que irá executar o som do disparo
        fireSound = GetComponent<AudioSource>();

        // Registra o botão virtual
        gunTrigger.GetComponent<VirtualButtonBehaviour>().  RegisterEventHandler (this);
    }
```

```
public void OnButtonPressed(VirtualButtonAbstractBehaviour  vb)
{
    // Cancela qualquer outra chamada à função que efetua o disparo
    CancelInvoke("Fire");
    // Se for possível atirar, atira
    if (!reloading && !firing && numberBullets >=  1)
    {
        firing =  true;
        InvokeRepeating("Fire", 0f, 0.7f);
    }
}

public void OnButtonReleased(VirtualButtonAbstractBehaviour  vb)
{
    // Não está mais apertando o botão de disparar
    firing =  false;
    CancelInvoke("Fire");
    // Se a arma está descarregada, recarrega-a
    if (numberBullets <  1)
    {
        reloading =  true;
        Invoke("Reload", 2f);
        UpdateUI(string.Format("Bullets: {0}¥nReloading...", numberBullets));
    }
}

private void Fire()
{
    // Verifica se tem munição suficiente para efetuar o disparo
    if (numberBullets <  1)
    {
        CancelInvoke("Fire");
        return;
    }
    // Efetua o disparo
    var bulletInstance =  Instantiate(bulletPrefab,
                          bulletSpawnPlace.transform.position  ,
                          Quaternion.identity  ) as GameObject;
    bulletInstance.GetComponent<Rigidbody>().  AddForce(bulletSpawnPlace
      .transform.forward * bulletSpeed );
    numberBullets--;
    // Ativa som e animação
    fireAnimation. Play();
    fireSound. Play();
    // Atualiza UI com o número de munições disponíveis
    UpdateUI(string.Format("Bullets: {0}", numberBullets));
}

private void Reload()
{
```

```
        // Recarrega as munições, atualiza na UI e permite que novos disparos
        // sejam efetuados
        numberBullets = 5;
        UpdateUI(string.Format("Bullets: {0}", numberBullets));
        reloading = false;
    }

    private void UpdateUI(string text)
    {
        // Atualiza o texto com o número de munições
        bulletsNumberUI.text = text;
    }
}
```

RastreioMarcador.cs

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

public class RastreioMarcador : MonoBehaviour, ITrackableEventHandler {

    // Variável para pegar o estado do marcador
    private TrackableBehaviour mTrackableBehaviour;

    void Start ()
    {
        // Carrega o componente que verifica o estado do marcador
        mTrackableBehaviour = GetComponent<TrackableBehaviour>();
        if (mTrackableBehaviour)
        {
            // Permite alterar a conduta de cada estado do marcador
            mTrackableBehaviour.RegisterTrackableEventHandler(this);
        }
    }

    public void OnTrackableStateChanged(TrackableBehaviour.Status previousStatus,
      TrackableBehaviour.Status newStatus)
    {
        if (newStatus == TrackableBehaviour.Status.DETECTED ||
            newStatus == TrackableBehaviour.Status.TRACKED ||
            newStatus == TrackableBehaviour.Status.EXTENDED_TRACKED)
        {
            // Se o marcador está visível, chama o método OnTrackingFound
            OnTrackingFound();
        }
```

```
    else
    {
        // Se o marcador não está visivel, chama o método OnTrackingLost
        OnTrackingLost();
    }
}

private void OnTrackingFound()
{
    // Pegando os componentes de renderização e colisão de todos os "filhos"
    Renderer[] rendererComponents = GetComponentsInChildren<Renderer>();
    Collider[] colliderComponents = GetComponentsInChildren<Collider>();

    // Percorre todos os "filhos" e os ativa
    for (int i = 0; i < this.transform.GetChildCount(); ++i)
    {
        Debug.Log("Ativando os filhos");
        this.transform.GetChild(i).gameObject.SetActive(true);
    }

    // Ativa os renderizadores de todos os "filhos"
    foreach (Renderer component  in rendererComponents)
    {
        component.enabled =  true;
    }

    // Ativa os colisores de todos os "filhos"
    foreach (Collider component  in colliderComponents)
    {
        component.enabled =  true;
    }

    Debug.Log("Rastreamento de " + mTrackableBehaviour.TrackableName +  " encontrado");
}

private void OnTrackingLost()
{
    // Pegando os componentes de renderização e colisão de todos os "filhos"
    Renderer[] rendererComponents = GetComponentsInChildren<Renderer>();
    Collider[] colliderComponents = GetComponentsInChildren<Collider>();

    // Percorre todos os "filhos" e os desativa
    for (int i = 0; i < this.transform.GetChildCount(); ++i)
    {
        Debug.Log("Ativando os filhos");
        this.transform.GetChild(i).gameObject.SetActive(false);
    }

    // Desativa os renderizadores de todos os "filhos"
    foreach (Renderer component  in rendererComponents)
    {
```

```
            component.enabled =  false;
        }

        // Desativa os colisores de todos os "filhos"
        foreach (Collider component  in colliderComponents)
        {
            component.enabled =  false;
        }

        Debug.Log("Rastreamento de " + mTrackableBehaviour.TrackableName +  " perdido");
    }
}
```

ScoreManager.cs

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

public class ScoreManager : MonoBehaviour {

    // Componente que escreve a pontuação no UI
    private Text scoreUI;
    // Contador de pontos
    private int score = 0;

    void Start()
    {
        // Referência ao componente que escreve no local da pontuação
        scoreUI = gameObject.GetComponent<Text>();
    }

    public void AddScore()
    {
        // Atualiza os pontos, considerando que conseguiu mais um
        scoreUI.text =  string.Format("Score: {0}", ++score);
    }
}
```

TargetSpawner.cs

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

```
public class TargetSpawner  : MonoBehaviour
{

    // Prefab do alvo
    public GameObject targetPrefab;

    void Start ()
    {
        // Fica instanciando alvos a cada 7 segundos
        InvokeRepeating("IntantiateTarget", 7f, 7f);
    }

    void IntantiateTarget()
    {
        // Instancia até 3 alvos
        if (transform.childCount <  3)
        {
            // Instancia um alvo na posição do objeto vazio Targets
            GameObject targetInstance =  Instantiate(targetPrefab,
                                        transform.position  ,
                                        Quaternion.identity  ) as GameObject;
            // Deixa o gato olhando para a câmera
            targetInstance.transform. Rotate(0, 180, 0);
            // Deixa o gato (alvo) como filho do objeto vazio Targets
            targetInstance.transform.parent = transform;
        }
    }
}
```