Entrega 01

Algoritmos e Lógica de Programação

Profª Lucy

404

Not Found

Breno Groba de Azevedo, Igor Almeida, João Victor Ferian, Kaio Inglez e Gustavo Diniz

Grupo 5

Scripts da Câmera:

- **Movimentação da câmera:**

```
using UnityEngine;

public class PlayerCameraTopDown : MonoBehaviour
{
   public float velocidade = 20.0f;
   public float velocidadeCorrendo = 40.0f; // velocidade quando está com o
Control pressionado
   public Transform cameraTransform; // referência à câmera principal

   void Update()
   {  // Movimentação da Câmera
      float horizontal = Input.GetAxis("Horizontal");
      float vertical = Input.GetAxis("Vertical");

      Vector3 forward = cameraTransform.forward;
      Vector3 right = cameraTransform.right;

      forward.y = 0f;
      right.y = 0f;
      //  Código para as direções de movimento ficarem de acordo com a
câmera
      forward.Normalize();
      right.Normalize();

      Vector3 moveDirection = (forward * vertical + right *
horizontal).normalized;

        // Se Control estiver pressionado, a velocidade aumenta.
      float velocidadeAtual = Input.GetKey(KeyCode.LeftControl) ||
Input.GetKey(KeyCode.RightControl)
         ? velocidadeCorrendo
         : velocidade;

      transform.position += moveDirection * velocidadeAtual * Time.deltaTime;
   }
```

- **Zoom da câmera:**

```
using UnityEngine;
using static UnityEngine.GraphicsBuffer;

public class NewMonoBehaviourScript : MonoBehaviour
{
   public Camera cam; // Referência à câmera que será controlada
   public float zoomSpeed = 5f; // Velocidade do zoom
   public float minFOV = 50f; // Menor campo de visão permitido (Zoom IN)
   public float maxFOV = 120f; // Maior campo de visão permitido (Zoom
OUT)

   void Update()
   {
      float scroll = Input.GetAxis("Mouse ScrollWheel");
      // Captura a rolagem do scroll do mouse

      if (scroll != 0) // Se houve movimento no scroll do mouse
      {
         cam.fieldOfView -= scroll * zoomSpeed;
         // Ajusta o FOV (quanto menor o FOV, mais zoom)

         cam.fieldOfView = Mathf.Clamp(cam.fieldOfView, minFOV, maxFOV);
         // Garante que o FOV fique dentro dos limites definidos

      }
   }
}
```

- **Alvo da câmera e velocidade de movimento:**

```
using Unity.VisualScripting;
using UnityEngine;

public class CameraTopDown : MonoBehaviour
{
   //declarei a variável que controla a velocidade da câmera
   public float velocidade = 5.0f;
   // acessa o objeto que a câmera precisa seguir
   public Transform target;
   // distância dos eixos entre a câmera e o player
   public Vector3 offset = Vector3.up;


   void Update()
   {   //faz a movimentação de forma suave
      transform.position = Vector3.Lerp(transform.position,
target.position + offset, velocidade*Time.deltaTime);
   }
}
```

Rotação da câmera ao clicar no Mouse 0:

```
using UnityEngine;

public class MouseClick : MonoBehaviour
{

   public Vector2 turn;// declaração da variável turn com um vetor 2D (x e y)
   public float sensitivity = .5f; // sensibilidade do mouse

   void Start()
   {
      Cursor.lockState = CursorLockMode.Locked; // comando para o cursor
não aparecer na tela durante o jogo
   }
   void Update()
   {  // movimentação do mouse ao pressionar o botão direito
      if (Input.GetMouseButton(1))
      {
         turn.x += Input.GetAxis("Mouse X") * sensitivity; // Multiplicação da
sensibilidade do mouse pelo movimento no eixo X
         turn.y += Input.GetAxis("Mouse Y") * sensitivity; // Multiplicação da
sensibilidade do mouse pelo movimento no eixo Y
         transform.localRotation = Quaternion.Euler(-turn.y, turn.x, 0); //
comando para realizar a rotação de um objeto convertendo valores de um
ângulo


      }
   }
}
```

Script Raycast:

```
using UnityEngine;

public class Raycast : MonoBehaviour
{
   RaycastHit hitInfo;  // Armazena informações do objeto atingido pelo
raycast
   GameObject lastHighlighted = null; // Guarda o último objeto que foi
destacado
   Material originalMaterial; // Guarda o material original antes de aplicar o
highlight
   public Material highlightMaterial; // Coloca um material de destaque no
Inspector
   private Predios predioSelecionado = null; // Referência para o prédio
atualmente selecionado

   void Update()
   {  // Cria um raio que parte do centro da tela
      Ray ray = Camera.main.ViewportPointToRay(new Vector3(0.5f, 0.5f, 0));
      GameObject hitObject = null;

      // Raycast para visualização e highlight. Verifica se colidiu com algo
      if (Physics.Raycast(ray, out hitInfo, 100f))
      {
         hitObject = hitInfo.collider.gameObject; // Obtém o GameObject
atingido

         Debug.Log(hitObject.name); // Mostra o nome do objeto atingido no
console

         // Se mudou de objeto, reseta o anterior
         if (lastHighlighted != null && lastHighlighted != hitObject)
         {
            lastHighlighted.GetComponent<Renderer>().material =
originalMaterial;
         }

         // Salva o material original e aplica o destaque
         if (hitObject.CompareTag("Construcao"))
         {
            if (hitObject.TryGetComponent(out Predios predio))
```

```
            {
                predio.ativo = true;
                predio.TrocaMaterial();
            }
        }// Caso o objeto não tenha a tag "Construcao", mas ainda seja do tipo
Predios
        else if (hitObject.TryGetComponent(out Predios predio))
        {
            predio.ativo = false;
        }
    }

    // Detecta clique com o botão esquerdo do mouse
    if (Input.GetMouseButtonDown(0))
    {
        if (hitObject != null && hitObject.CompareTag("Construcao"))
        {
            if (hitObject.TryGetComponent(out Predios novoPredio))
            {
                // Se já havia um prédio selecionado e é diferente do novo,
desseleciona o anterior
                if (predioSelecionado != null && predioSelecionado !=
novoPredio)
                {
                    predioSelecionado.Desselecionar();
                }

                // Seleciona o novo prédio e atualiza a referência
                novoPredio.Selecionar();
                predioSelecionado = novoPredio;
            }
        }
        else
        {
            // Clicou no vazio ou em algo que não é construção
            if (predioSelecionado != null)
            {
                predioSelecionado.Desselecionar();
                predioSelecionado = null;
            }
        }
    }
```

```
    }
}
```

Script dos prédios:

```
using UnityEngine;
using System.Collections;

public class Predios : MonoBehaviour
{
    public bool ativo; // Indica se o prédio está ativo (destacado)
    public bool selecionado; // Indica se o prédio está selecionado
    private Vector3 posicaoOriginal; // Armazena a posição original do prédio
    private Renderer rendererPredio; // Referência ao componente Renderer
do prédio (para trocar cor/material)

    void Start()
    {
        ativo = false; // Inicialmente não está ativo
        selecionado = false; // Inicialmente não está selecionado
        posicaoOriginal = transform.position; // Salva a posição original para
depois voltar
        rendererPredio = GetComponent<Renderer>(); // Pega o Renderer do
objeto
    }

    // Método para ativar o efeito de destaque no prédio
    public void TrocaMaterial()
    {
        StopCoroutine("TrocaCor"); // Para qualquer coroutine anterior que
esteja mudando a cor
        if (ativo)
        {
            // Altera a cor de emissão para destacar
            rendererPredio.material.SetColor("_EmissionColor", Color.white *
0.22f);
            StartCoroutine("TrocaCor"); // Inicia a coroutine que desativa o
destaque após um tempo
        }
    }

    // Coroutine que aguarda 0.1 segundo antes de apagar o brilho e desativar
o "ativo"
    IEnumerator TrocaCor()
    {
```

```
        yield return new WaitForSeconds(0.1f);
        ativo = false; //Desativa o estado "ativo"
        rendererPredio.material.SetColor("_EmissionColor", Color.black);
    }

    // Método para selecionar o prédio
    public void Selecionar()
    {
        if (!selecionado)
        {
            selecionado = true; // Marca como selecionado
            transform.position += new Vector3(0, 0.7f, 0); // Move o prédio para
cima (efeito visual de seleção)

        }
    }

    // Método para desselecionar o prédio
    public void Desselecionar()
    {
        if (selecionado)
        {
            selecionado = false;  // Marca como não selecionado
            transform.position = posicaoOriginal; // Volta à posição original

        }
    }
}
```

Script da quantidade de moradores em cada casa:

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

// Classe que representa uma casa
public class Casa
{
   public int moradores; // Classe que representa a quantidade de moradores da casa
   public List<int> tiposDeConsumo = new List<int>(); // Lista com os tipos de consumo de cada morador (1:Econômico, 2:Médio, 3:Alto)
   public int consumoAguaTotal; /// Consumo total de água da casa (em litros por dia)
   public int consumoEnergiaTotal; // Consumo total de energia da casa (em kWh por mês)
}

public class Program
{
   public static void Main()
   {
      List<Casa> casas = new List<Casa>(); // Lista para armazenar todas as casas
      int numeroDeCasas = 32; // Total de casas que serão geradas

      Random random = new Random();  // Objeto para gerar números aleatórios

      // Laço para gerar cada casa
      for (int i = 0; i < numeroDeCasas; i++)
      {
         Casa novaCasa = new Casa(); // Cria uma nova casa
         novaCasa.moradores = random.Next(1, 6); // Define um número aleatório de moradores entre 1 e 5
         novaCasa.consumoAguaTotal = 0; // Inicializa o consumo de água
         novaCasa.consumoEnergiaTotal = 0; // Inicializa o consumo de energia

         // Laço para cada morador da casa
         for (int j = 0; j < novaCasa.moradores; j++)
         {
```

```
            int tipo = random.Next(1, 4); // Tipo de consumo: 1 (Econômico), 2 (Médio) ou 3 (Alto)
            novaCasa.tiposDeConsumo.Add(tipo); // Adiciona o tipo de consumo à lista da casa

            // Adiciona os valores correspondentes de consumo com base no tipo
            switch (tipo)
            {
                case 1: // Econômico
                    novaCasa.consumoAguaTotal += 110;
                    novaCasa.consumoEnergiaTotal += 80;
                    break;
                case 2: // Médio
                    novaCasa.consumoAguaTotal += 166;
                    novaCasa.consumoEnergiaTotal += 155;
                    break;
                case 3: // Alto
                    novaCasa.consumoAguaTotal += 300;
                    novaCasa.consumoEnergiaTotal += 300;
                    break;
            }
        }

        casas.Add(novaCasa); // Adiciona a casa completa à lista
    }

    // Exibe um resumo de todas as casas geradas
    Console.WriteLine("Resumo por Casa:");
    Console.WriteLine("C\tM\tÁgua\tEnergia");
    for (int i = 0; i < casas.Count; i++)
    {
        // Exibe o número da casa (C), número de moradores (M), consumo de água e energia
        Console.WriteLine($"{i + 1}\t{casas[i].moradores}\t{casas[i].consumoAguaTotal}\t\t{casas[i].consumoEnergiaTotal}");
    }
  }
}
```

Script da passagem dos dias:

```
using UnityEngine;

public class SkyboxDayCycle : MonoBehaviour
{
    [Header("Duração do ciclo (em segundos)")]
    public float fullDayLength = 180f; // Duração total do ciclo dia/noite (180 segundos por padrão)


    private Material skyboxMat; // Referência ao material do skybox atual
    private float currentTime = 0f; // Tempo decorrido desde o início do ciclo

    void Start()
    {
        skyboxMat = RenderSettings.skybox; // Obtém o material de skybox configurado no RenderSettings
    }

    void Update()
    {
        currentTime += Time.deltaTime; // Incrementa o tempo com o tempo que passou desde o último frame

        // Calcula uma fração (de 0 a 1) do tempo atual em relação ao ciclo completo
        float t = (currentTime % fullDayLength) / fullDayLength;

        float blend; // Variável que vai controlar a transição entre céu diurno e noturno

        if (t < 0.4f)
        {
            // DIA
            blend = 0f;
        }
        else if (t < 0.5f)
        {
            // ENTARDECER (10%)
            float p = (t - 0.4f) / 0.1f; // Progresso dentro da transição
```

```
            blend = Mathf.Lerp(0f, 1f, p); // Mistura entre os SkyBoxes do céu
diurno e noturno
        }
        else if (t < 0.9f)
        {
            // NOITE (40%)
            blend = 1f;
        }
        else
        {
            // AMANHECER (10%)
            float p = (t - 0.9f) / 0.1f; // Progresso dentro da transição
            blend = Mathf.Lerp(1f, 0f, p); // Mistura entre os SkyBoxes do céu
noturno e diurno
        }

        skyboxMat.SetFloat("_Blend", blend); // Aplica o valor de blend ao
material do skybox
        DynamicGI.UpdateEnvironment(); // Atualiza a iluminação global com
base na mudança do skybox
    }
}
```