Entrega 01

Algoritmos e Lógica de Programação

Profª Lucy

Entrega 2

404

Not Found

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

Grupo 5

Script de inicialização dos parâmetros(moradores, consumo):

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

// script que representa o comportamento de uma casa individual no jogo
public class CasaBehaviour : MonoBehaviour
{
    // variável do número de moradores da casa
    public int moradores;

    // lista dos tipos de consumo de cada morador (1 = baixo, 2 = médio, 3 = alto)
    public List<int> tiposDeConsumo = new List<int>();

    // consumo total de água da casa
    public float consumoAguaTotal;

    // consumo total de energia da casa
    public float consumoEnergiaTotal;

    // média dos tipos de consumo dos moradores
    public double mediaTipos;

    // quantidade de problemas identificados na casa
    public int totalProblemas;

    // lista de problemas atribuídos à casa
    public List<string> problemaCasa = new List<string>();

    // lista de soluções associadas aos problemas da casa
    public List<string> solucao = new List<string>();

    // método chamado para inicializar os dados da casa
    public void Inicializar(
        int moradores,
        List<int> tipos,
        double mediaTiposParam,
        int totalProblemasParam,
        float consumoAguaParam,
        float consumoEnergiaParam,
        List<string> problemaCasa,
        List<string> solucao
    )
    {
```

```
        // atribui os parâmetros recebidos às variáveis de cada classe
        this.moradores = moradores;
        this.tiposDeConsumo = tipos;
        this.mediaTipos = mediaTiposParam;
        this.totalProblemas = totalProblemasParam;
        this.consumoAguaTotal = consumoAguaParam;
        this.consumoEnergiaTotal = consumoEnergiaParam;
        this.problemaCasa = problemaCasa;
        this.solucao = solucao;
    }
}
```

Função responsável por carregar a cena, editar a porcentagem conforme o progresso do loading e ativar a cena do jogo:

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

public class LoadingManager : MonoBehaviour
{
    // nome da cena a ser carregada
    public string cenaParaCarregar = "jogo";

    // referência ao texto que exibirá a porcentagem de carregamento
    public TMP_Text textoPorcentagem;

    void Start()
    {
        // inicia a contagem que carrega a cena
        StartCoroutine(CarregarCena());
    }

    // coroutine para carregar a cena e atualizar a UI
    IEnumerator CarregarCena()
    {
        // começa a carregar a cena especificada
        AsyncOperation operacao = SceneManager.LoadSceneAsync(cenaParaCarregar);

        // evita que a cena seja ativada automaticamente quando o carregamento atingir 90%
        operacao.allowSceneActivation = false;

        // enquanto o carregamento não estiver completo
        while (operacao.isDone ==  false)
        {
            // calcula o progresso normalizado entre 0 e 1
            float progresso = Mathf.Clamp01(operacao.progress / 0.9f);

            // atualiza o texto da UI com a porcentagem de progresso, se o texto existir
            if (textoPorcentagem != null)
                textoPorcentagem.text = (progresso * 100f).ToString("F0") + "%";

            // quando o progresso atingir 90% (estado pronto para ativar)
            if (operacao.progress >= 0.9f)
            {
```

```
            // mostra 100% na UI
            textoPorcentagem.text = "100%";

            // espera 1 segundo
            yield return new WaitForSeconds(1f);

            // permite ativar a cena, finalizando o carregamento e mudando para ela
            operacao.allowSceneActivation = true;
        }

        // espera até o próximo frame antes de continuar o loop
        yield return null;
    }
  }
}
```

Função responsável por editar o texto exibido no menu de volume:

```
using UnityEngine;
```

```
using UnityEngine.UI;
using TMPro;

public class audioNum : MonoBehaviour
{
    // referência ao componente Slider da UI
    public Slider slider;

    // referência ao texto da UI que exibirá o valor em porcentagem
    public TMP_Text valorTexto;

    void Start()
    {
        // adiciona um listener para reagir quando o valor do slider mudar
        slider.onValueChanged.AddListener(UpdateValorTexto);

        // atualiza o texto com o valor inicial do slider
        UpdateValorTexto(slider.value);
    }

    // método para atualizar o texto de valor quando o slider é movimentado
    void UpdateValorTexto(float valor)
    {
        // converte o valor (de 0 a 1) para porcentagem e atualiza o texto
        valorTexto.text = Mathf.RoundToInt(valor * 100) + "%";
    }
}
```

Função responsável por fechar um dos menus que o jogador pode abrir durante a gameplay:

```
using UnityEngine;

public class fechar : MonoBehaviour
{
    // referência ao painel do menu que será ativado/desativado
    public GameObject menuPanel;

    // referência à câmera principal para acessar o componente RayCast
    public Camera cam;

    // método público para fechar o menu
    public void FecharMenu()
    {
        // verifica se o mouse está ativo no script RayCast associado à câmera
        if (cam.GetComponent<RayCast>().mouseAtivo == true)
        {
            // desativa o mouse ativo
            cam.GetComponent<RayCast>().mouseAtivo = false;

            // trava o cursor no centro da tela
            Cursor.lockState = CursorLockMode.Locked;
        }

        // marca que o painel está desativado no script RayCast
        cam.GetComponent<RayCast>().painelDesativo = true;

        // desativa o painel do menu
        menuPanel.SetActive(false);

    }
}
```

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].consumoEn
ergiaTotal}");
    }
  }
}
```

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
    }
}
```