**FUNDAÇÃO ESCOLA DE COMÉRCIO ALVARES PENTEADO**
**CURSO DE CIÊNCIA DA COMPUTAÇÃO**

**GUSTAVO ARCHANGELO**
**ANTONIO GABRIEL**
**LUIZ ANTONIO**
**PAULO GUILHERME**
**NICOLAS ARAUJO**

# ALGORITMOS E LÓGICA DE PROGRAMAÇÃO

**SÃO PAULO**
2025

# Sumário

**SCRIPTS USADOS PARA O BOTÃO DO MENU**

```
using UnityEngine; // Importa a biblioteca principal do Unity para manipulação de objetos e componentes.
using UnityEngine.SceneManagement; // Importa a biblioteca para gerenciamento de cenas no Unity.

public class BOTAOMENU : MonoBehaviour
{
    // Método chamado uma vez quando o script é ativado
    void Start()
    {
        // Geralmente usado para inicializar variáveis ou configurar o estado inicial do objeto.
    }

    // Método chamado a cada quadro (frame) enquanto o script estiver ativo
    void Update()
    {
        // Usado para verificar entradas do usuário, movimentação de objetos, etc., em tempo real.
    }

    // Método público que pode ser chamado para iniciar o jogo
    public void IniciaJogo()
    {
        // Carrega a cena com o índice 1, geralmente a primeira cena do jogo após o menu inicial.
        SceneManager.LoadScene(1);
    }
}
```

**SCRIPTS USADOS PARA O BOTÃO DE VELOCIDADE**

```
using UnityEngine; // Importa a biblioteca principal do Unity para manipulação de objetos e componentes.
```

```
using TMPro; // Importa a biblioteca para trabalhar com o TextMeshPro, um sistema avançado de texto no Unity.

public class ControladorVelocidade : MonoBehaviour
{
    public TextMeshProUGUI textoBotao; // Referência ao componente de texto do botão para exibir a velocidade atual.

    private float[] velocidades = { 1f, 2f, 4f }; // Array que armazena os diferentes valores de velocidade.
    private int indiceAtual = 0; // Índice que controla a velocidade atual.

    // Método chamado para alternar entre as velocidades definidas no array.
    public void AlternarVelocidade()
    {
        indiceAtual = (indiceAtual + 1) % velocidades.Length; // Atualiza o índice para o próximo valor no array, retornando ao início se necessário.
        Time.timeScale = velocidades[indiceAtual]; // Define a velocidade do jogo com base no valor atual do índice.
        AtualizarTexto(); // Atualiza o texto do botão para refletir a nova velocidade.
    }

    // Método chamado uma vez quando o script é ativado.
    void Start()
    {
        AtualizarTexto(); // Atualiza o texto do botão na inicialização.
    }

    // Método responsável por atualizar o texto do botão com a velocidade atual.
    void AtualizarTexto()
    {
        if (textoBotao != null) // Verifica se a referência ao componente de texto não é nula.
        {
            textoBotao.text = velocidades[indiceAtual].ToString() + "x"; // Define o texto do botão para mostrar a velocidade atual.
        }
```

```
    }
}
```

**SCRIPT PARA A ENERGIA DAS CASAS**

```
using UnityEngine; // Importa a biblioteca principal do Unity para manipulação de objetos e componentes.

public class CasaEnergia : MonoBehaviour
{
    [SerializeField]
    public string nomeCasa; // Nome da casa, pode ser atribuído via Inspector.

    public float energiaConsumida = 0f; // Total de energia consumida pela casa.
    public float aguaConsumida = 0f; // Total de água consumida pela casa.

    void Awake()
    {
        // Se o nome não for atribuído no Inspector, usa o nome do GameObject.
        if (string.IsNullOrWhiteSpace(nomeCasa))
        {
            nomeCasa = gameObject.name;
        }
    }

    // Método chamado uma vez por dia no jogo para consumir recursos.
    public void ConsumirRecursos()
    {
        // Gera valores aleatórios de consumo diário.
        float consumoEnergia = Random.Range(1f, 5f);  // energia em kWh
        float consumoAgua = Random.Range(2f, 6f);      // água em litros

        // Soma ao total acumulado.
        energiaConsumida += consumoEnergia;
        aguaConsumida += consumoAgua;
    }
```

```
}
```

**SCRIPT USADO PARA O CONTADOR DOS DIAS**

```
using UnityEngine; // Importa a biblioteca principal do Unity para manipulação de objetos e componentes.
using TMPro; // Usado para manipular textos com TextMeshPro
using UnityEngine.SceneManagement; // Permite trocar de cenas
using System.Collections; // Necessário para usar corrotinas (IEnumerator)

public class ContadorDias : MonoBehaviour
{
    public TMP_Text textoDias; // Referência ao texto que exibe o dia atual
    public int totalDias = 30; // Quantidade total de dias no jogo
    public float intervaloPorDia = 30f; // Tempo em segundos que representa 1 dia no jogo

    private int diaAtual = 0; // Contador de dias
    private float tempoAcumulado = 0f; // Acumula o tempo passado
    private bool jogoFinalizado = false; // Indica se o jogo terminou

    void Start()
    {
        AtualizarTexto(); // Atualiza o texto na interface no início do jogo
    }

    void Update()
    {
        if (jogoFinalizado) return; // Se o jogo terminou, para a execução

        tempoAcumulado += Time.deltaTime; // Soma o tempo desde o último frame

        if (tempoAcumulado >= intervaloPorDia)
        {
            tempoAcumulado = 0f; // Reseta o tempo acumulado
            diaAtual++; // Passa para o próximo dia
```

```
            AtualizarTexto(); // Atualiza o texto com o novo dia

            // Se o número de dias chegou ao limite, finaliza o jogo
            if (diaAtual >= totalDias)
            {
                jogoFinalizado = true;
                textoDias.text = $"Fim do jogo - Dia {diaAtual}";
                StartCoroutine(CarregarCenaFim()); // Inicia a corrotina para trocar de cena
            }
        }
    }

    void AtualizarTexto()
    {
        textoDias.text = $"Dia: {diaAtual}"; // Mostra o dia atual na interface
    }

    // Corrotina que espera 3 segundos antes de carregar a cena de fim
    IEnumerator CarregarCenaFim()
    {
        yield return new WaitForSeconds(3f);
        SceneManager.LoadScene("FimDoJogo"); // Carrega a cena chamada "FimDoJogo"
    }
}
```

**SCRIPT USADO PARA O GERENCIADOR DE ENERGIA DA CIDADE**

```
using System.Collections.Generic;  // Importa a biblioteca que permite trabalhar com listas genéricas, como List<T>
using UnityEngine;                 // Importa a biblioteca principal do Unity para manipulação de objetos e componentes
using TMPro;                       // Importa a biblioteca do TextMeshPro para manipular textos no Unity
using UnityEngine.SceneManagement; // Permite trocar de cenas no Unity
```

```
public class TabelaEnergia : MonoBehaviour
{
    [Header("Referências")]
    public CasaEnergia[] casas;             // Array de casas que participam da simulação de consumo
    public TMP_Text[] camposEnergia;        // Campos de texto que mostrarão o consumo de energia de cada casa
    public TMP_Text rankingTexto;           // Campo de texto que exibirá o ranking das casas (pode ser ocultado se não for necessário)

    [Header("Configuração do tempo")]
    public int totalDias = 30;              // Número total de dias para a simulação
    public float tempoPorDia = 30f;         // Tempo (em segundos) que representa um "dia" no jogo

    private int diaAtual = 0;               // Contador de dias da simulação
    private float tempoAcumulado = 0f;      // Acumula o tempo para controlar a passagem dos dias
    private bool jogoFinalizado = false;    // Flag para verificar se o jogo foi finalizado

    // Método chamado no início para inicializar as variáveis
    void Start()
    {
        // Se o texto do ranking foi atribuído, oculta o ranking na tela inicialmente
        if (rankingTexto != null)
        {
            rankingTexto.gameObject.SetActive(false); // Oculta ranking na tela se não for usado aqui
        }
    }

    // Método chamado a cada quadro (frame) enquanto o jogo estiver em andamento
    void Update()
    {
        // Se o jogo já foi finalizado, não faz nada no Update
```

```
        if (jogoFinalizado) return;

        // Acumula o tempo que passa durante o jogo
        tempoAcumulado += Time.deltaTime;

        // Verifica se o tempo acumulado passou do tempo necessário para completar 1 "dia"
        if (tempoAcumulado >= tempoPorDia)
        {
            tempoAcumulado = 0f;  // Reseta o tempo acumulado para iniciar o próximo "dia"
            diaAtual++;           // Avança para o próximo dia da simulação

            // Simula o consumo de energia para cada casa de forma aleatória entre 1 e 5 kWh
            foreach (CasaEnergia casa in casas)
            {
                float energia = Random.Range(1f, 5f);  // Gera um consumo de energia aleatório
                casa.energiaConsumida += energia;     // Soma o valor gerado ao consumo total da
casa
            }

            AtualizarTabela(); // Atualiza a tabela exibida com os novos consumos

            // Quando o número de dias atingir o total de dias definidos, finaliza o jogo
            if (diaAtual >= totalDias)
            {
                jogoFinalizado = true; // Marca o jogo como finalizado
                MostrarRanking();      // Exibe o ranking das casas
                StartCoroutine(CarregarCenaRanking(5f)); // Chama a corrotina para carregar a cena
do ranking após 5 segundos
            }
        }
    }

    // Método responsável por atualizar a tabela de consumo de energia na interface
    public void AtualizarTabela()
    {
```

```
        // Para cada casa, atualiza o campo de texto correspondente com o consumo de energia acumulado
        for (int i = 0; i < casas.Length && i < camposEnergia.Length; i++)
        {
            // Atualiza o texto de cada campo para mostrar o nome da casa e o consumo de energia
            camposEnergia[i].text = $"{casas[i].nomeCasa}: {casas[i].energiaConsumida:F1} kWh";
        }
    }

    // Método para exibir o ranking de casas baseado no consumo de energia
    void MostrarRanking()
    {
        // Cria uma lista com todas as casas e ordena por menor consumo de energia (casas que consumiram menos no topo)
        List<CasaEnergia> ranking = new List<CasaEnergia>(casas);
        ranking.Sort((a, b) => a.energiaConsumida.CompareTo(b.energiaConsumida));

        // Limpa as listas de nomes e consumos de energia dos rankings anteriores
        DadosDoJogo.nomesRanking.Clear();
        DadosDoJogo.energiaRanking.Clear();

        // Armazena os dados do ranking em ordem do maior para o menor consumo de energia
        for (int i = ranking.Count - 1; i >= 0; i--)
        {
            DadosDoJogo.nomesRanking.Add(ranking[i].nomeCasa);              // Adiciona o nome da casa
            DadosDoJogo.energiaRanking.Add(ranking[i].energiaConsumida);    // Adiciona o consumo de energia da casa
        }
    }

    // Corrotina que aguarda o tempo especificado (delay) antes de carregar a cena do ranking
    System.Collections.IEnumerator CarregarCenaRanking(float delay)
    {
        yield return new WaitForSeconds(delay); // Espera o tempo definido
```

```
        SceneManager.LoadScene("FimDoJogo");     // Carrega a cena de fim de jogo
    }
}
```

**SCRIPT USADO PARA QUICK EVENTS**

```
using System.Collections;  // Importa a biblioteca que permite usar corrotinas.
using System.Collections.Generic;  // Importa a biblioteca que fornece suporte a coleções genéricas, como List<T>.
using UnityEngine;  // Importa a biblioteca principal do Unity para manipulação de objetos e componentes.
using UnityEngine.UI;  // Importa o namespace necessário para trabalhar com UI (Interface de usuário) no Unity.
using TMPro;  // Importa o TextMeshPro, utilizado para manipular texto na interface.

public class QuickEventManager : MonoBehaviour
{
    [Header("Referências")]
    public List<QuickEvent> eventList;  // Lista de eventos rápidos disponíveis no jogo (cada evento pode ter uma pergunta e respostas).
    public GameObject quickEventUI;  // UI que aparece para mostrar a pergunta e opções.
    public TMP_Text questionText;  // Texto que exibe a pergunta do evento.
    public Button[] optionButtons;  // Botões que representam as opções de resposta.
    public CasaEnergia casaEnergia;  // Referência à casa de energia, onde o consumo será ajustado dependendo da resposta.
    public TabelaEnergia tabelaEnergia;  // Referência à tabela de energia, para atualizar a UI depois de um evento.
    public float penalidadeEnergia = 3f;  // Penalidade de energia a ser adicionada caso o jogador escolha a resposta errada.
    public float reducaoEnergia = 2f;  // Redução de energia a ser subtraída caso o jogador escolha a resposta correta.
    public float timeBetweenEvents = 60f;  // Tempo entre um evento e outro, em segundos.
```

```
    private void Start()
    {
        quickEventUI.SetActive(false);  // Inicialmente a UI do evento rápido está oculta.
        StartCoroutine(EventRoutine());  // Inicia a corrotina que controla os eventos rápidos.
    }

    // Corrotina que fica aguardando o tempo entre eventos e depois chama um evento aleatório.
    IEnumerator EventRoutine()
    {
        while (true)  // Laço infinito para criar eventos continuamente.
        {
            yield return new WaitForSeconds(timeBetweenEvents);  // Espera pelo tempo entre eventos.
            TriggerRandomEvent();  // Dispara um evento aleatório.
        }
    }

    // Método que escolhe um evento aleatório da lista de eventos e o exibe.
    void TriggerRandomEvent()
    {
        if (eventList.Count == 0) return;  // Se a lista de eventos estiver vazia, sai do método.

        QuickEvent evt = eventList[Random.Range(0, eventList.Count)];  // Escolhe um evento aleatório da lista.
        ShowEvent(evt);  // Exibe o evento escolhido.
    }

    // Método que exibe o evento na UI, pausa o jogo e exibe a pergunta e as opções.
    void ShowEvent(QuickEvent evt)
    {
        Time.timeScale = 0f;  // Pausa o tempo do jogo enquanto o evento está ativo (o jogador não pode mover-se, por exemplo).

        quickEventUI.SetActive(true);  // Exibe a UI do evento.
        questionText.SetText(evt.question);  // Define o texto da pergunta.
```

```
        questionText.ForceMeshUpdate();  // Atualiza o texto na interface.

        // Para cada botão de opção, define o texto e o comportamento ao clicar.
        for (int i = 0; i < optionButtons.Length; i++)
        {
            int index = i;  // Armazena o índice do botão.
            TMP_Text btnText = optionButtons[i].GetComponentInChildren<TMP_Text>();  // Pega o texto do botão.
            btnText.SetText(evt.options[i]);  // Define o texto da opção do botão.
            btnText.ForceMeshUpdate();  // Atualiza o texto do botão.

            optionButtons[i].interactable = true;  // Habilita a interação com o botão.

            // Reseta a cor da imagem do botão para branco (cor padrão).
            Image btnImage = optionButtons[i].GetComponent<Image>();
            btnImage.color = Color.white;

            // Remove qualquer ouvinte anterior de clique (para evitar múltiplos cliques em eventos antigos).
            optionButtons[i].onClick.RemoveAllListeners();
            optionButtons[i].onClick.AddListener(() =>
            {
                bool acertou = (index == evt.respostaCorretaIndex);  // Verifica se a resposta clicada é a correta.

                // Muda a cor do botão clicado para verde (se acertou) ou vermelho (se errou).
                Image clickedImage = optionButtons[index].GetComponent<Image>();
                if (acertou)
                {
                    clickedImage.color = Color.green;  // Resposta correta: verde.
                }
                else
                {
                    clickedImage.color = Color.red;  // Resposta errada: vermelho.
                }
```

```
        // Desativa todos os botões para que o jogador não possa clicar em mais de uma opção.
        foreach (var btn in optionButtons)
            btn.interactable = false;

        // Se houver uma referência à casa de energia, ajusta o consumo com base na resposta do jogador.
        if (casaEnergia != null)
        {
            if (acertou)
            {
                casaEnergia.energiaConsumida = Mathf.Max(0f, casaEnergia.energiaConsumida - reducaoEnergia);  // Subtrai a energia (não deixa ser negativa).
                Debug.Log($"Resposta correta! -{reducaoEnergia} kWh. Total: {casaEnergia.energiaConsumida} kWh");
            }
            else
            {
                casaEnergia.energiaConsumida += penalidadeEnergia;  // Adiciona penalidade de energia.
                Debug.Log($"Resposta errada! +{penalidadeEnergia} kWh. Total: {casaEnergia.energiaConsumida} kWh");
            }

            // Se houver uma referência à tabela de energia, atualiza a tabela com os novos valores.
            if (tabelaEnergia != null)
            {
                tabelaEnergia.AtualizarTabela();
            }
        }

        // Invoca qualquer ação associada à opção selecionada.
        if (evt.onOptionSelected[index] != null)
        {
            evt.onOptionSelected[index].Invoke();
```

```
            }

            // Chama a corrotina para fechar o evento depois de um pequeno delay.
            StartCoroutine(FecharEventoDepoisDeDelay(1f));
        });
    }
}

// Corrotina que espera um tempo real (ignorando o "Time.timeScale" pausado) e depois fecha o evento.
IEnumerator FecharEventoDepoisDeDelay(float delay)
{
    yield return new WaitForSecondsRealtime(delay);  // Espera o delay em tempo real, pois o jogo está pausado.

    quickEventUI.SetActive(false);  // Fecha a UI do evento.
    Time.timeScale = 1f;  // Retorna o tempo do jogo ao normal.

    // Reseta os botões para o próximo evento (deixa eles interativos e com a cor padrão).
    foreach (var btn in optionButtons)
    {
        btn.interactable = true;  // Torna os botões novamente interativos.

        // Reseta a cor da imagem para branco.
        Image img = btn.GetComponent<Image>();
        img.color = Color.white;
    }
}
}
```

**SCRIPT PARA RANKING FINAL**

```
using System.Collections.Generic;  // Importa a biblioteca que fornece suporte a coleções genéricas como List<T>.
using UnityEngine;  // Importa a biblioteca principal do Unity para manipulação de objetos e componentes.
using TMPro;  // Importa o TextMeshPro, utilizado para manipular texto na interface de usuário.

public class RankingFinal : MonoBehaviour
{
    public TMP_Text rankingCompletoTexto;  // Referência ao componente de texto onde o ranking final será exibido.

    // Método chamado automaticamente quando o objeto é inicializado no jogo (como o Start do Unity).
    void Start()
    {
        ExibirRanking();  // Chama o método que exibe o ranking quando o jogo começa.
    }

    // Método para exibir o ranking final de consumo de energia.
    void ExibirRanking()
    {
        List<string> nomes = DadosDoJogo.nomesRanking;  // Lista de nomes das casas no ranking (salvo em outro script).
        List<float> consumos = DadosDoJogo.energiaRanking;  // Lista de consumos de energia das casas (salvo em outro script).

        // Verifica se as listas de nomes ou consumos estão vazias ou nulas, e mostra uma mensagem de erro se necessário.
        if (nomes == null || consumos == null || nomes.Count == 0 || consumos.Count == 0)
        {
            rankingCompletoTexto.text = "Ranking indisponível.";  // Caso as listas estejam vazias ou nulas, exibe uma mensagem de erro.
            return;  // Sai do método sem tentar gerar o ranking.
        }
```

```
        // Cria o título para o ranking final.
        string texto = "<b>Ranking Final de Consumo</b>\n\n";

        // Determina o total de casas a ser exibido (no máximo 3).
        int total = Mathf.Min(3, nomes.Count);  // A quantidade de casas a ser exibida será no máximo 3, ou o total de casas disponíveis, se houver menos de 3.

        // Laço que monta a string com o ranking, começando pelo 1º lugar.
        for (int i = total - 1; i >= 0; i--)  // Laço que começa do último colocado (menor consumo) para o primeiro (maior consumo).
        {
            int posicao = total - i;  // Calcula a posição no ranking (1º, 2º, 3º).
            texto += $"{posicao}º - {nomes[i]}: {consumos[i]:F1} kWh\n";  // Adiciona a casa e o consumo à string do ranking.
        }

        // Atualiza o componente de texto da interface para exibir o ranking completo.
        rankingCompletoTexto.text = texto;
    }
}
```

**SCRIPT PARA PAUSAR O JOGO**

```
using UnityEngine;  // Importa a biblioteca principal do Unity para manipulação de objetos e componentes.
using UnityEngine.UI;  // Importa a biblioteca para trabalhar com a interface do usuário, como Sliders e Buttons.
using UnityEngine.SceneManagement;  // Importa a biblioteca para manipulação de cenas no Unity.

public class Pausar : MonoBehaviour
{
    public GameObject pausemenu;  // Referência ao painel de pause (UI) que será exibido quando o jogo for pausado.
    public Slider volume;  // Referência ao componente Slider que ajusta o volume do jogo.
```

```
    public AudioSource audioSource;  // Referência ao AudioSource responsável pela reprodução do áudio no jogo.

    private bool isPaused = false;  // Controla se o jogo está pausado ou não.

    // Método chamado quando o jogo começa.
    void Start()
    {
        pausemenu.SetActive(false);  // Inicializa o jogo sem o painel de pausa visível.
        volume.onValueChanged.AddListener(Volume);  // Adiciona um ouvinte para o evento de mudança de valor do Slider de volume.
        volume.value = AudioListener.volume;  // Inicializa o Slider com o volume atual do áudio global.
    }

    // Método chamado a cada frame.
    void Update()
    {
        // Verifica se a tecla "Escape" foi pressionada.
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (isPaused)
                DespausarJogo();  // Se o jogo estiver pausado, chama o método para despausar.
            else
                PausarJogo();  // Se o jogo não estiver pausado, chama o método para pausar.
        }
    }

    // Método para pausar o jogo.
    public void PausarJogo()
    {
        isPaused = true;  // Marca o jogo como pausado.
        Time.timeScale = 0f;  // Pausa o tempo do jogo, fazendo com que tudo pare (movimento, animações, física, etc).
        pausemenu.SetActive(true);  // Torna o painel de pausa visível.
    }
```

```
    // Método para despausar o jogo.
    public void DespausarJogo()
    {
        isPaused = false;  // Marca o jogo como não pausado.
        Time.timeScale = 1f;  // Restaura o tempo normal do jogo (tempo volta a correr normalmente).
        pausemenu.SetActive(false);  // Torna o painel de pausa invisível.
    }

    // Método que ajusta o volume do jogo quando o Slider é movido.
    public void Volume(float value)
    {
        AudioListener.volume = value;  // Define o volume global do áudio do jogo de acordo com o valor do Slider.
    }
}
```

**FUNÇAO DO BOTAO PARA VOLTAR POR MENU INICIAL**

```
using UnityEngine;  // Importa a biblioteca principal do Unity para manipulação de objetos e componentes.
using UnityEngine.SceneManagement;  // Importa a biblioteca para manipulação de cenas no Unity.

public class VoltarMENU : MonoBehaviour
{
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    void Start()
    {
        // Neste caso, o método Start não realiza nenhuma ação. Ele pode ser removido se não for necessário.
    }

    // Update is called once per frame
    void Update()
```

```
    {
        // O método Update também não contém nenhuma lógica no momento, mas ele é chamado a cada frame.
    }

    // Método público que é chamado para carregar a cena principal (menu).
    public void VoltarMenu()
    {
        // Carrega a cena com índice 0 (normalmente o menu principal, se ele estiver configurado como a cena inicial).
        SceneManager.LoadScene(0);
    }
}
```