PROFESSOR:Adriano Felix Valente	
CURSO: Ciência da Computação	
DISCIPLINA: Algoritmos e Lógica de Programação	
TURMA: 1 CCOMP	DATA: 13/10/2025
ALUNOS(AS): Lucas de Freitas Soares, Emilly Oliveira dos Santos, Guilherme Belcastro de Medeiros, Kaike Cavalcante dos Santos	

Algoritmos e Lógica de Programação

Projeto 4 – Quarto Fobo

Dessa forma, visando demonstrar o funcionamento prático da lógica aplicada a programação de desenvolvimento de jogos, este presente relatório descreverá os códigos utilizados para a criação do jogo Quarto Fobo, jogo este que está em desenvolvimento e apresenta o conceito de um jogo Escape Room 3D com visão geométrica e uma temática voltada para o psicológico, possuindo em suas "entranhas diversos códigos para a execução das ações feitas no jogo.

Script - Outline

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

public enum ObjectType      //cria uma categoria para o tipo de objeto
{
    ground, door, text, dialogue, collectable, none, interactable
}

public class Outline : MonoBehaviour        //permite que outros códigos encontrem essa classificação
{
    public ObjectType objectType;
}
```

Este script cria uma categoria para cada objeto a qual ele é vinculado, sendo elas: ground, door, text, dialogue, collectable, none e interactable.

FECAP

Script – ItemSlot

```
using System;
using UnityEngine;
using UnityEngine.EventSystems;

public class ItemSlot : MonoBehaviour, IPointerClickHandler     //IPointerClickHandler permite a interação com objetos 2D como a UI do inventário
{
    [Header("UI")]
    public GameObject selectedShader;       //função que verifica se o painel de destaque (SelectedPanel) está ativo
    public bool thisItemSelected;       //função que verifica se o item está selecionado ou não
    public Item Item;

    [Header("Runtime")]
    public Inventory inventory;     //pode arrastar no inspector ou será buscado no Start

    private void Awake()
    {
        //garante que o shader comece invisível
        if (selectedShader != null)
            selectedShader.SetActive(false);
    }

    private void Start()
    {
        //tenta resolver o inventory automaticamente se não foi definido no inspector
        if (inventory == null)
        {
            inventory = GetComponentInParent<Inventory>();
            if (inventory == null)
            {
                inventory = FindObjectOfType<Inventory>();
            }
        }

        Debug.Log($"ItemSlot.Start - name: {gameObject.name} | inventory: {(inventory != null ? inventory.name : "null")} | selectedShader: {(selectedShader != null ? selectedShader.name : "null")}");
    }

    //faz o inventário poder ser clicável
    public void OnPointerClick(PointerEventData eventData)
    {
        Debug.Log($"ItemSlot.OnPointerClick on {gameObject.name} button: {eventData.button}");
        if (eventData.button == PointerEventData.InputButton.Left)
            OnLeftClick();
        else if (eventData.button == PointerEventData.InputButton.Right)
            OnRightClick();
```

```
    //faz a verificação se o item está no slot ou se é um slot vazio
    private void OnLeftClick()
    {
        if (inventory == null)
        {
            Debug.LogError($"ItemSlot {gameObject.name}: inventory está null ao clicar. Verifique referência.");
            return;
        }

        //se o slot já estava selecionado, desseleciona tudo
        if (thisItemSelected)
        {
            inventory.DeselectAllSlots();
            return;
        }

        //desseleciona todos os slots e seleciona este
        inventory.DeselectAllSlots();
        Select();
    }

    private void OnRightClick()
    {

    }

    public void Select()
    {
        if (selectedShader != null)
            selectedShader.SetActive(true);

        thisItemSelected = true;

        //atualiza o slot selecionado no Inventory
        if (inventory != null)
        {
            inventory.SetSelectedSlot(this);
        }
    }


    public void Deselect()
    {
        if (selectedShader != null)
            selectedShader.SetActive(false);
        thisItemSelected = false;
        // Debug.Log($"ItemSlot.Deselect -> {gameObject.name}");
    }

    internal void ClearSlot()
    {
        throw new NotImplementedException();
    }
}
```

Este script é responsável pelo funcionamento dos slots do inventário, permitindo a visualização dos itens e da seleção dentro da UI inventory.

Script - Item

```
using UnityEngine;

[CreateAssetMenu(fileName = "NewItem", menuName = "Item")]      //permite criar um asset com as características deste  código
public class Item : ScriptableObject        //ScriptableObject determina que é um objeto modificável
{
    [Header("Identificação única")]     //título no menu da Unity
    public string itemID;       //cria a classe nomeável para o ID do Item

    public string itemName;     //cria a classe nomeável para o nome do Item
    public Sprite itemImage;        //cria a classe que permite colocar a imagem do item
    [TextArea]
    public string description;      //cria a classe nomeável para a descrição do Item
}
```

Este script categoriza os objetos vinculados a ele como ScriptableObject, podendo assim, interagir com este objeto através de outros scripts e podendo incrementar o mesmo no inventário.

Script - Inventory

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

public class Inventory : MonoBehaviour
{
    public List<Item> items = new List<Item>();     //função que cria uma lista de itens
    public List<ItemSlot> itemSlots = new List<ItemSlot>();     //função que permite atualizar essa lista com cada item selecionado

    public static Inventory instance;       //torna o inventário uma instância que pode ser usada em outros scripts

    private void Awake()        //função chamada antes do primeiro frame
    {
        //permite que o inventário seja salvo para ser mantido o mesmo independente da cena
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
            return;
        }
    }

    private void Start()
    {
        // Tenta pegar slots filhos primeiro
        itemSlots = new List<ItemSlot>(GetComponentsInChildren<ItemSlot>());

        // Se nada foi encontrado, tenta buscar globalmente (fallback)
        if (itemSlots.Count == 0)
        {
            ItemSlot[] found = FindObjectsOfType<ItemSlot>();
            itemSlots = new List<ItemSlot>(found);
            Debug.LogWarning($"Inventory: nenhum ItemSlot filho encontrado, usando fallback. Slots achados = {itemSlots.Count}");
        }
        else
        {
            Debug.Log($"Inventory: slots encontrados como filhos = {itemSlots.Count}");
        }
    }

    public static void SetItem(Item item)       //função que adiciona o item ao inventário
    {
        if (instance == null) return;       //se não tiver nada, retorna

        instance.items.Add(item);       //adiciona o item ao inventário
        UiManager.SetInventoryImage(item);      //coloca a imagem desse item no UI do inventário
    }

    public static bool HasItem(Item item)       //função para a verificação se o item existe no inventário
    {
        if (instance == null) return false;     //se não tiver nada, retorna false
        return instance.items.Contains(item);       //retorna a quatidade e se contém o item
    }


    public ItemSlot selectedSlot;       //função para slots do inventário selecionados

    //SELECIONA O SLOT
    public void SetSelectedSlot(ItemSlot slot)
    {
        selectedSlot = slot;
    }


    //DESSELECIONA O SLOT
    public void DeselectAllSlots()      //função para desselecionar os slots
    {
        //faz a contagem dos slots
        for (int i = 0; i < itemSlots.Count; i++)
        {
            //verifica se o slot está selecionado e desseleciona ele
            var slot = itemSlots[i];
            if (slot != null)
            {
                slot.Deselect();
            }
        }
    }
}
```

Este script trata da lógica por trás do funcionamento do inventário interativo, fazendo com que toda a lógica de guardar, selecionar e usar item seja aplicada diretamente nele.

Script - Interactable

```
using UnityEngine;
using UnityEngine.UI;

[RequireComponent(typeof(Outline))]     //az o objeto necessitar do componete script "Outline"
public abstract class Interactable : MonoBehaviour
{
    public ObjectType objectType;       //função que determina o tipo de objeto

    public bool isInteracting;      //função que determina se está interagindo
    public Item conditionalItem;        //função que determina o item como condicional

    public Outline outline;     //permite acessar o script Outline

    private void Awake()        //é chamado automaticamente quando um objeto de jogo é instanciado na cena, antes que o primeiro frame seja atualizado
    {
        //tenta encontrar o componente outline e deixa ele como desativado
        try
        {
            outline = GetComponent<Outline>();
            outline.enabled = false;
        }
        //informa se o componete não está presente
        catch
        {
            Debug.LogError("Objeto " + gameObject.name + " precisa de um Outline");
        }

    }

    public abstract void Interact();        //função que permite o uso desse script
}
```

Este script torna qualquer objeto a qual é aplicado um objeto interativo, por exemplo, podendo ser um objeto que executa uma ação ou coletável, sendo guardado no inventário.

Script - CollectableItem

```
using UnityEngine;

public class CollectableItem : Interactable     //Determina que o script pertence ao tipo interativo
{
    public Item item;       //permite vincular com o script "Item"

    public override void Interact()     //função de interação para o item coletável
    {
        if (item == null)
        {
            Debug.LogWarning($"O objeto {gameObject.name} não possui um Item definido!");
            return;
        }

        // Adiciona o item ao inventário
        Inventory.SetItem(item);
        Debug.Log($"Coletou {item.name}");

        // Encontra todos os objetos do tipo, inclusive desativados
        CollectableItem[] allItems = Resources.FindObjectsOfTypeAll<CollectableItem>();

        //Indica o que ocorre com cada item coletado
        foreach (CollectableItem obj in allItems)
        {
            //determina que se o objeto é defierente de null e é igual a item
            if (obj != null && obj.item == item)
            {
                if (obj.gameObject.scene.IsValid()) // ignora prefabs fora da cena
                {
                    Destroy(obj.gameObject);        //destrói o objeto e seus clones dentro da cena
                }
            }
        }
    }
}
```

Este script trata da lógica por trás dos itens guardados no inventário, ele torna os objetos “Interactables” em objetos coletáveis, permitindo assim o seu armazenamento no inventário e destruindo o objeto presente na cena.

Script - ClickManager

```
using UnityEngine;

public class ClickManager : MonoBehaviour
{
    [SerializeField] private Camera mainCamera; // arraste a Main Camera aqui no inspetor
    [SerializeField] private float clickRange = 100f; // distância máxima do clique

    void Update()
    {
        if (Input.GetMouseButtonDown(0)) // clique esquerdo do mouse
        {
            //determina o "raio" que o mouse estava clicando e se o objeto era um com interação
            Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;

            if (Physics.Raycast(ray, out hit, clickRange))      //verifica se o sinal foi recebido como true
            {
                Interactable interactable = hit.collider.GetComponent<Interactable>();// verifica se o objeto clicado tem um Interactable

                if (interactable != null)       //determina que se o objeto interativo existe, ou seja é diferente de null(nada)
                {
                    interactable.Interact(); // executa o comportamento (no caso, coletar e destruir)

                }
            }
        }
    }
}
```

Este script se refere a leitura dos clicks do mouse, permitindo que essa interação ocorra com os objetos presentes no cenário, além de ser aplicado principalmente para a interação do mouse com as UI presentes no jogo, como o inventário por exemplo.

Script - UsableObject

```
using UnityEngine;
using UnityEngine.Events;

//PRECISA MONTAR O CÓDIGO DA MANEIRA CORRETA

public class UsableObject : Interactable
{
    [Header("Item Required")]
    public Item requiredItem; // qual item precisa para interagir
    public bool consumeItem = true; // se true, remove do inventário ao usar

    [Header("Events")]
    public UnityEvent onUseItem; // eventos visuais ou lógicos

    public override void Interact()
    {
        if (InventoryHasSelectedItem())
        {
            Debug.Log($"Usando {Inventory.instance.selectedSlot.Item.itemName} em {gameObject.name}");

            // dispara evento
            onUseItem.Invoke();

            // remove do inventário
            if (consumeItem)
            {
                InventoryUseSelectedItem();
            }

            //Para a seleção
            Inventory.instance.DeselectAllSlots();
        }
        else
        {
            Debug.Log($"Não tem o item correto selecionado para usar em {gameObject.name}");
        }
    }

    private bool InventoryHasSelectedItem()
    {
        if (Inventory.instance.selectedSlot == null || Inventory.instance.selectedSlot.Item == null)
            return false;

        Item selectedItem = Inventory.instance.selectedSlot.Item;
        Debug.Log($"Selecionado: {selectedItem.itemID}, Requerido: {requiredItem.itemID}");

        return selectedItem.itemID == requiredItem.itemID;
    }


    private void InventoryUseSelectedItem()
    {
        Item usedItem = Inventory.instance.selectedSlot.Item;
        Inventory.instance.selectedSlot.Deselect();
        Inventory.instance.selectedSlot = null;

        Inventory.instance.items.Remove(usedItem);
        UiManager.RemoveInventoryImage(usedItem);
    }
}
```

Este script ainda está em estado de correção, mas sua funcionalidade tratará da interação dos itens armazenados no inventário com algum objeto determinado no cenário.

Script – Rotação da sala

```
    //FUNCIONAMENTO DO ARRASTAR DO MOUSE
    if (dragging)
    {
        Vector3 delta = Input.mousePosition - lastMousePosition;        //calcula quanto o mouse se moveu no eixo X desde o último frame.
        float rotationAmount = -delta.x * dragRotationSpeed * Time.deltaTime;       //converte isso em um valor de rotação.

        float currentY = transform.eulerAngles.y;       //obtêm o ângulo atual de y

        // Limita a rotação entre +-90°
        float minAngle = (currentSnapAngle - 90f + 360f) % 360f;
        float maxAngle = (currentSnapAngle + 90f) % 360f;

        //garante que o movimento não ultrapasse o limite de 90° por giro
        float proposedY = (currentY + rotationAmount + 360f) % 360f;
        bool allowRotation;

        if (minAngle < maxAngle)
            allowRotation = proposedY >= minAngle && proposedY <= maxAngle;
        else
            allowRotation = proposedY >= minAngle || proposedY <= maxAngle;

        if (allowRotation)
        {
            transform.Rotate(0f, rotationAmount, 0f);
            lastMousePosition = Input.mousePosition;        //determina o último posicionamento do mouse enquanto estava rodando o objeto

            //determina que o ângulo escolhido para rodar será o mais próximo do múltiplo de 90°, sendo um movimento maior que 45° vai pro próximo ângulo e menor volta para o que estava
            float angleDiff = Mathf.DeltaAngle(currentSnapAngle, transform.eulerAngles.y);
            if (Mathf.Abs(angleDiff) >= 45f)
            {
                nextSnapAngle = (angleDiff > 0)
                    ? (currentSnapAngle + 90f) % 360f
                    : (currentSnapAngle - 90f + 360f) % 360f;
            }
        }

        AtualizarParedes();     //Atualiza a função com estas condições
    }

    //ROTAÇÃO DA SALA ATRAVÉS DAS TECLAS/SETAS
    if (Input.GetKeyDown(KeyCode.RightArrow) && !isRotating)
    {
        currentSnapAngle = Mathf.Round(transform.eulerAngles.y / 90f) * 90f;
        nextSnapAngle = (currentSnapAngle - 90f + 360f) % 360f;
        targetRotation = Quaternion.Euler(0, nextSnapAngle, 0);
        isRotating = true;
    }
    else if (Input.GetKeyDown(KeyCode.LeftArrow) && !isRotating)
    {
        currentSnapAngle = Mathf.Round(transform.eulerAngles.y / 90f) * 90f;
        nextSnapAngle = (currentSnapAngle + 90f) % 360f;
        targetRotation = Quaternion.Euler(0, nextSnapAngle, 0);
        isRotating = true;
    }

    AtualizarParedes();     //Atualiza a função com estas condições
}

//VISIBILIDADE DAS PAREDES
void AtualizarParedes()     //Função para ativar ou desativar as paredes de acordo com a angulação
{
    float angulo = transform.eulerAngles.y;


    angulo = (angulo + 360f) % 360f;    // Normaliza o ângulo entre 0 - 360
```

```
using UnityEngine;

public class RotacaoDaSala : MonoBehaviour
{
    [Header("Configurações de Rotação")]        //cria um *título no menu da Unity
    [SerializeField] float dragRotationSpeed = 100f;     // Velocidade da movimentação com o mouse
    [SerializeField] float smoothRotationSpeed = 300f;   // Velocidade da rotação com as setas

    [Header("Referências das Paredes")]     //cria um *título no menu da Unity
    public GameObject Parede1;      //determina qual parede vai ser ativado a consição de aparecer ou sumir
    public GameObject Parede2;
    public GameObject Parede3;
    public GameObject Parede4;

    private bool dragging = false;      //indica se o mouse está sendo movimentado/arrastado
    private bool isRotating = false;        //indica se o objeto está rodando
    private Vector3 lastMousePosition;      //grava a última posição do mouse

    private Quaternion targetRotation;      //determia a rotação que o objeto deve atingir, com o Quaternion garantindo que a ângulação seja mais livre
    private float currentSnapAngle;     //demonstra o ângulo atual (arredondando para os múltiplos do 90°)
    private float nextSnapAngle;        //determina o próxima ângulo (mantendo os múltiplos de 90°)

    void Start()        //Determina tudo o que vai ocorrer no início do jogo
    {
        currentSnapAngle = 0f;      //determina o ângulo inicial da sala
        targetRotation = Quaternion.Euler(0, currentSnapAngle, 0); //determina o tipo de rotação da sala, sendo feita através de ângulos
        transform.rotation = targetRotation;    //atualiza a rotação conforme o próximo ângulo
        nextSnapAngle = currentSnapAngle;       //determina que o ângulo atual vai ser o próximo ângulo

        AtualizarParedes();     //Atualiza a função com estas condições
    }

    void Update()       //determina o que vai acontecer a cada frame
    {
        //MOVIMENTAÇÃO DO MOUSE
        if (Input.GetMouseButtonDown(0) && !isRotating)     //quando o botão esquerdo do mouse é pressionado sem estar rodando o objeto
        {
            dragging = true;        //ativa a função de arrastar o mouse
            lastMousePosition = Input.mousePosition;

            float rounded = Mathf.Round(transform.eulerAngles.y / 90f) * 90f;   //garante que o ângulo vai ser arredondado para o mais próximo múltiplo de 90°
            currentSnapAngle = rounded;
            nextSnapAngle = currentSnapAngle;
        }

        if (Input.GetMouseButtonUp(0) && dragging && !isRotating)       //quando o botão esquerdo do mouse é para de ser pressionado cancela todas as funções
        {
            dragging = false;
            targetRotation = Quaternion.Euler(0, nextSnapAngle, 0);
            isRotating = true;
        }

        // MOVIMENTAÇÃO SUAVE DE ÂNGULO EM ÂNGULO
        if (isRotating)
        {
            transform.rotation = Quaternion.RotateTowards(      //limita quantos graus vai ser rodado por frame
                transform.rotation,     //determina a rotação em si do objeto
                targetRotation,     //determina o objetivo final da rotação
                smoothRotationSpeed * Time.deltaTime        //define a velocidade (em graus por segundo) da rotação, utilizando o Time.deltaTime para ocorrer a cada frame do computador
            );

            if (Quaternion.Angle(transform.rotation, targetRotation) < 0.1f)        //calcula o ângulo de diferença (em graus) entre duas rotações.
            {
                transform.rotation = targetRotation;
                isRotating = false;     //finaliza a rotação
            }

            AtualizarParedes();     //Atualiza a função com estas condições
            return; //garantia de que somente este script seja usado quando for ativado
        }
```

```
        // INTERVALOS

        // 0° = entre 315° e 45°
        if (angulo >= 315f || angulo < 45f)
        {
            Parede1.SetActive(true);
            Parede2.SetActive(false);
            Parede3.SetActive(false);
            Parede4.SetActive(true);
        }

        // 90° = entre 45° e 135°
        else if (angulo >= 45f && angulo < 135f)
        {
            Parede1.SetActive(true);
            Parede2.SetActive(false);
            Parede3.SetActive(true);
            Parede4.SetActive(false);
        }

        // 180° = entre 135° e 225°
        else if (angulo >= 135f && angulo < 225f)
        {
            Parede1.SetActive(false);
            Parede2.SetActive(true);
            Parede3.SetActive(true);
            Parede4.SetActive(false);
        }

        // 270° = entre 225° e 315°
        else if (angulo >= 225f && angulo < 315f)
        {
            Parede1.SetActive(false);
            Parede2.SetActive(true);
            Parede3.SetActive(false);
            Parede4.SetActive(true);
        }
    }
}
```

Este script trata de uma das mecânicas principais da sala, sendo a rotação da sala através de ângulos, limitando para que a sala seja rodada somente uma vez para a esquerda ou para a direita a cada 90º. Além disso, ele também faz com que as paredes da sala se tornem visíveis ou invisíveis para o jogado de acordo com a angulação da sala.

Script - menuPrincipal

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class menuPrincipal : MonoBehaviour
{

    //funções que criam os elementos do menu
    //SerializeField permite acessar essa função no espectro da unity, mas não permite editar ele com outros scripts
   [SerializeField] private GameObject painelmenuinicial;
   [SerializeField] private GameObject painelopcoes;
   [SerializeField] private GameObject MenudoNewGame;

    //função que determina se o menu do jogo vai ser ativado
    public void Iniciar()
    {
       MenudoNewGame.SetActive(true);
    }

    //função que carrega a fase escolhida caso o botão seja pressionado
    public void NovoJogo()
    {
        SceneManager.LoadScene("MapaT");        //LoadScene muda para a cena determinada
    }

    //função que permite retornar ao menu inicial
    public void VoltarAoMenu()
    {
       painelmenuinicial.SetActive(true);       //ativa o painel
        MenudoNewGame.SetActive(false);     //desativa a função menunewgame
    }

    //função que permite acessar o menu de opções do jogo ao pressionar o botão
    public void AbrirOpcoes()
        {
        painelmenuinicial.SetActive(false);     //desativa o painel menuinicial
        painelopcoes.SetActive(true);       //ativa o painel
    }


    //função que fecha o menu de opções do jogo ao pressionar o botão
    public void FecharOpcoes()

    {
    painelmenuinicial.SetActive(true);      //ativa o painel
        painelopcoes.SetActive(false);      //desativa o painel de opções

    }

    //função que permite sair do jogo ao pressionar o botão
    public void Sair()
    {
        Debug.Log("Sair");      //escreve no console o que foi solicitado
        Application.Quit();     //fecha o jogo
    }


}
```

Este script gerencia o menu principal do jogo, permitindo que os botões presentes nele funcionem de maneira adequada, seja iniciando o jogo, mudando alguma configuração do mesmo ou até fechando o jogo.