PROFESSOR: Adriano Felix Valente	
CURSO: Ciência da Computação	
DISCIPLINA: Algoritmos e Lógica de Programação	
TURMA: Matutino	DATA: 22/04/2025
ALUNOS: Analice Coimbra Carneiro, Harry Zhu, João Pedro Da Silva, Rafaela Florêncio Morais	

TÍTULO DA ATIVIDADE:

Análise da Aplicação de Algoritmos com Estruturas de Decisão, Repetição, Vetores e Matrizes nos Códigos Fornecidos

Serão analisados os scripts "*Menu.cs*", "*MotionDetector.cs*", "*PlayerCasting.cs*", "*Timer.cs*", "*UIController.cs*", "*cameraRenderScript.cs*", "*flashlight.cs*", "DoorOpen.cs,"*PickUpScript.cs*" e *"BarradeSanidade.cs"* identificando o uso dessas estruturas algorítmicas e seus resultados dentro do contexto do projeto.

Estruturas de Decisão

Estruturas de decisão permitem que o programa execute diferentes blocos de código com base em uma condição. Nos scripts fornecidos, a estrutura de decisão fundamental utilizada é o *if*.

MotionDetector.cs: Este script utiliza estruturas de decisão nos métodos OnTriggerEnter e OnTriggerExit.

No OnTriggerEnter, um if verifica se o objeto que entrou no trigger possui a tag "Player" (ou "Intruder"). Se a condição for verdadeira, uma mensagem é logada no console (*Debug.Log("Movimento detectado!");*), a cor da luz spotLight é alterada para vermelho (se spotLight não for nulo), e o som de alerta é reproduzido (se alertSound não for nulo). O resultado dessa decisão é uma mudança no estado visual e sonoro do jogo em resposta à detecção do jogador.

Similarmente, no OnTriggerExit, um if verifica se o objeto que saiu do trigger possui a tag "Player". Se verdadeiro, a mensagem "Área limpa." é logada e a cor da spotLight é alterada para verde (se não for nulo). O resultado é o retorno ao estado de "alerta desligado".

Timer.cs: Este script emprega várias estruturas de decisão no método Update.

Um if verifica se o tempo de jogo em minutos (*currentGameTimeInMinutes*) atingiu ou ultrapassou 1440 (24 horas). Se sim, subtrai 1440 para simular a passagem para o dia seguinte (00:00) e uma variável sceneLoaded é definida como true. O resultado é a implementação de um ciclo de 24 horas no tempo do jogo.

Outro if verifica se o tempo de jogo atingiu ou passou de 360 minutos (6:00) e se a cena foi carregada (*sceneLoaded*). Em caso afirmativo, a cena de índice 3 é carregada (*SceneManager.LoadScene(3)*). O resultado é a transição automática para uma nova fase do jogo em um horário específico.

Um terceiro if verifica se o número de minutos atual (minutes) é um múltiplo de 5 e diferente do último minuto exibido (*lastDisplayedMinute*). Se ambas as condições forem verdadeiras, o texto do timer na interface do usuário (*timerText*) é atualizado com o novo tempo formatado, e lastDisplayedMinute é atualizado. O resultado é a atualização do timer na tela apenas a cada 5 minutos do tempo de jogo, otimizando a atualização da interface.

UIController.cs: No método Update, uma estrutura de decisão if-else controla a visibilidade dos elementos da interface do usuário.

O if verifica se a variável booleana estática *uiActive* é verdadeira. Se for, as caixas de ação (*actionBox*), de comando (*commandBox*) e a mira de interação (*interactCross*) são ativadas (*SetActive(true)*) e seus textos são atualizados com os valores das variáveis estáticas *actionText* e commandText. O resultado é a exibição da interface quando *uiActive* está ativo.

O else é executado quando *uiActive* é falso, desativando os mesmos elementos da interface (*SetActive(false)*). O resultado é o desaparecimento da interface quando *uiActive* está inativo.

cameraRenderScript.cs: No método Start, um if-else verifica se o componente Camera e a renderTexture foram encontrados.

O if é executado se ambos existirem, definindo a renderTexture da câmera como o alvo de renderização (camera.targetTexture = renderTexture). O resultado é a configuração da câmera para renderizar para uma textura específica.

O else é executado se algum deles não for encontrado, logando uma mensagem de erro no console (*Debug.LogError("Não encontrei");*). O resultado é a indicação de um problema de configuração no editor Unity.

flashlight.cs: O script flashlight.cs emprega estruturas if e if ! dentro do bloco que verifica a pressão da tecla "F". Essas decisões alternam o estado da lanterna (ligada ou desligada) e a visibilidade dos GameObjects ON e OFF, controlando o efeito de ligar e desligar a luz.

PickUpScript.cs: O PickUpScript.cs emprega extensivamente estruturas de decisão. Por exemplo, na função *Update*, um if verifica se a tecla "E" foi pressionada e, dentro dele, outro if verifica se nenhum objeto está sendo segurado (*heldObj == null*). Essa decisão determina se uma tentativa de pegar um objeto deve ser iniciada. Da mesma forma, outro if verifica se um objeto está sendo segurado (*heldObj != null*) para então executar funções de mover e rotacionar o objeto. Essas decisões controlam o ciclo de pegar, segurar, rotacionar, largar e arremessar objetos. Além disso, dentro da função *RotateObject*, um if verifica se a tecla "R" está sendo pressionada para habilitar a rotação, e outro if else controla se o objeto pode ser largado (*canDrop*). Essas decisões governam a interação de rotação e a possibilidade de largar o objeto durante a rotação. A função *StopClipping* também utiliza uma estrutura de decisão (*if (hits.Length > 1)*) para verificar se um raycast atingiu mais do que o objeto carregado, ajustando a posição do objeto para evitar clipping. O resultado dessa decisão é a prevenção de problemas visuais quando o objeto é largado ou arremessado perto de outros objetos.

DoorOpen.cs: Utiliza estrutura de decisões e variáveis booleanas para usar a tecla “E” quando estiver na área do trigger(OnTriggerEnter) e quando a porta estiver fechada(doorOpen = false) para abrir a porta e “OnTriggerExit” para a porta fechar quando estiver fora da área(doorOpen = true).

BarradeSanidade.cs: O script utiliza estruturas de decisão para verificar o estado do ambiente e atualizar a sanidade do personagem. No método Update, o *if* verifica se o jogador está dentro da casa (*if (estaDentroDaCasa)*), ativando o sistema de verificação de sanidade. Isso significa que a lógica da sanidade é verificada constantemente, garantindo que o jogo reaja dinamicamente a cada mudança do ambiente. Caso esteja, o código vai analisar as condições do ambiente e modificar a sanidade do personagem com base nisso. Alguns fatores que modificam a barra de sanidade do jogador, são apresentados e dentro do próximo if, o script analisa se algum desses fatores (*monstroNaJanela, janelaVazia, ouvindoBarulhos e semLuz*) está ativo. Essas estruturas de decisão controlam a entrada de dados booleanos *(true/false)* e determinam a resposta lógica do sistema.

Estruturas de Repetição

Estruturas de repetição (como for e while) permitem executar um bloco de código múltiplas vezes. Porém, nenhum dos scripts feitos contém exemplos explícitos da utilização dessas estruturas de repetição como for ou while,mas a função Update() presente em muitos scripts atua como um loop implícito, pois é chamada repetidamente a cada frame do jogo.

Vetores

Em programação, vetores (ou arrays) são estruturas de dados que armazenam uma coleção de elementos do mesmo tipo sob um único nome, acessíveis por um índice.

PlayerCasting.cs: Neste script é utilizada a estrutura Vector3, que é um tipo de dado fundamental na Unity para representar vetores tridimensionais (posição, direção, etc.). No entanto, não há uma aplicação de vetores no sentido de arrays sendo manipulados para implementar algoritmos de iteração ou processamento de múltiplos dados. A variável *transform.position* e a função *transform.TransformDirection(Vector3.forward)* utilizam a representação vetorial para realizar o raycast, que é um processo de lançar um raio em uma direção e verificar se ele atinge algum objeto. O resultado é a obtenção da distância até o objeto atingido, armazenada nas variáveis toTarget e distanceFromTarget.

PickUpScript.cs: No script PickUpScript.cs, vetores são utilizados para diversas funcionalidades. Um exemplo é o uso de *transform.TransformDirection(Vector3.forward)* para realizar o raycast na direção para a frente do objeto que está realizando o lançamento do raio. Além disso, vetores são empregados para manter o objeto segurado na posição de um ponto específico chamado *holdPos*, como em *heldObj.transform.position = holdPos.transform.position.* No momento de arremessar o objeto, utiliza-se transform.forward para aplicar força na direção para a frente. Também é feito o cálculo da distância entre o objeto segurado e a posição da câmera usando *Vector3.Distance*, e, para evitar o problema de clipping, aplica-se um pequeno deslocamento na posição do objeto com *new Vector3(0f, -0.5f, 0f).*

Nos demais scripts, não há utilização explícita de estruturas de dados vetoriais (arrays) para fins algorítmicos.

Matrizes

Matrizes são estruturas de dados bidimensionais (tabelas) que armazenam elementos do mesmo tipo, acessíveis por dois índices (linha e coluna). Nenhum dos scripts feitos demonstram a utilização de estruturas de dados matriciais ou operações matriciais para implementar algoritmos.

Conclusão

A análise dos códigos feitos revela que a principal estrutura algorítmica utilizada é a de decisão (if), empregada para controlar o fluxo de execução com base em condições como detecção de movimento, tempo decorrido no jogo, estado da interface e configuração da câmera. O uso de vetores é crucial para manipular objetos no espaço tridimensional, permitindo interações como detecção, movimentação e arremesso. Embora matrizes não sejam manipuladas diretamente nos scripts, elas estão intimamente envolvidas nas operações de transformação espacial realizadas pela Engine do Unity