Introdução

Neste projeto foi desenvolvido um jogo da memória em Unity utilizando linguagem C#. O código está dividido em três scripts principais (CardRandomizer, Card, CardFlip), que trabalham em conjunto para sortear cartas, controlar a virada, verificar pares e aplicar regras do jogo.

O objetivo deste relatório é demonstrar como o projeto aplica conceitos essenciais de algoritmos e lógica de programação, incluindo estruturas de decisão, estruturas de repetição, vetores, funções e corrotinas, dentro do contexto do Unity.

Estruturas Utilizadas no Código

Switch-case no sorteio das cartas (CardRandomizer)

Cada carta é gerada conforme o valor aleatório rnd, garantindo que cada tipo apareça no máximo duas vezes:

```
switch (rnd)
{
    case 0:
        if (a < 2) { ... }
        break;
    case 1:
        if (b < 2) { ... }
        break;
    ...
}
```

Condicionais para impedir cliques inválidos (CardFlip)

```
if (isAnimating || !gameManager.canClick)
    return;

if (cardToFlip == gameManager.cardFliped1)
    return;
```

Essas decisões garantem que o jogador não vire mais cartas do que o permitido.

Verificação de pares (CardFlip)

```
if (c1.cardId == c2.cardId)
{
    Destroy(card1);
    Destroy(card2);
```

```
}
else
{
   StartCoroutine(FlipAnimationObject(card1));
}
```

O JOGO UTILIZA ESTRUTURAS DE REPETIÇÃO E AQUI ESTÁ O FUNCIONAMENTO:

Loop da animação da carta

```
while (elapsedTime < flipDuration)
{
   card.transform.rotation = Quaternion.Slerp(...);
   elapsedTime += Time.deltaTime;
   yield return null;
}
```

Esse loop executa pequenos passos por frame, simulando a virada suave da carta.

Loop de geração até encontrar carta disponível

```
while (!cartaGerada)
{
  rnd = Random.Range(0, cards.Length);
  ...
}
```

Ele garante que cada tipo de carta apareça somente 2 vezes.

VETORES E MATRIZES
O projeto utiliza vetores e matrizes em várias partes do jogo, mas selecionei partes bem específicas e essenciais do mesmo para demonstrar nesse relatório.

Vetor de cartas e slots (CardRandomizer)

```
[SerializeField]
private GameObject[] cards;
[SerializeField]
private Transform[] slots;
```

Definem quais cartas podem ser instanciadas e onde elas estarão posicionadas no tabuleiro.

Embaralhamento usando LINQ

```
cards = cards.OrderBy(x => Random.value).ToArray();
slots = slots.OrderBy(x => Random.value).ToArray();
```

Funções e Procedimentos• CardRandomizer

- Start() — inicializa e chama geração de cartas

- GerarCartas(int i) — escolhe carta aleatória e instancia
- SetupCards() — organiza cartas e anima virada inicial (corrotina)
- FlipCard() — executa flip inicial

• Card

- MarkMatched() — marca carta correta e desativa colisores
- GetRenderers() — retorna renderizadores da carta

• CardFlip

- Update() — captura clique do jogador
- FlipAnimation() — anima virada de carta clicada
- StartFlipBack() — desvira carta via GameManager
- CheckMatchCoroutine() — verifica se as duas cartas abertas são iguais

As funções são utilizadas para dividir todo o andamento do jogo.

Conclusão

O projeto demonstra de maneira clara a aplicação prática de conceitos de algoritmos e lógica de programação no Unity.

Foram utilizados:

- Estruturas de decisão para controle de fluxo
- Loops para animações e sorteio
- Vetores para organizar cartas e slots
- Funções e corrotinas para modularizar a lógica do jogo

O código exemplifica um sistema real, onde diferentes estruturas de programação foram utilizadas para criar o jogo.