UNIVERSIDADE DO VALE DO ITAJAÍ - UNIVALI

Data: __16__/__05__/_____2025_

Disciplina: Estrutura de dados
Professor(a): Eduardo Poffo
Acadêmico(a): ___Victor Roberto Viebrantz e Kenzo Gabriel Hashiguti Tanaka_____

Regras e orientações para a realização das provas
1. Leia com atenção cada questão, inclusive aos desenhos (algumas informações importantes são adquiridas nele); **2. As questões desta prova têm C# como a linguagem de programação.** **3. A avaliação é em dupla e permitida a consulta EXCLUSIVAMENTE nas documentações do C# e Unity;** **4. É permitido o uso de calculadoras científicas;** **5. Só será atribuído pontos as questões que apresentarem as respostas de forma organizada e legível e sem interpretação dupla;** **6. Destaque a resolução final, caso seja necessária;** **7. Só será permitida a entrada na sala de aula enquanto ainda ninguém ter entregado a prova;** **8. Não é permitido o uso de celulares, notebooks, livros e outras fontes externas durante a execução da avaliação;** **9. Os últimos 3 alunos deverão sair da sala de aula juntos.** **10. Atribuir-se-á zero aos alunos que usarem de meios fraudulentos para a realização da prova.** **11. Boa sorte!**

1. (Pontuação: 1.5 ponto) Explique com suas próprias palavras:

a) o que é um array em C#. Em seguida, responda:

Conjunto de dados organizados em posições por meio de um índice sendo esses valores de um mesmo tipo com um tamanho fixo.

b) Qual é a principal diferença entre um array e uma List<> em termos de tamanho e flexibilidade?

Array – tamanho limitado e dados do mesmo tipo.

Lista – pode guardar mais dados e pode comportar tipos diferentes

c) Em que situação prática no desenvolvimento de um jogo com Unity você usaria um array em vez de uma lista? Justifique sua resposta.

Numa mecânica de senha onde o player poderia abrir um cofre ou uma porta trancada

Onde cada dígito seria uma posição no array sendo eles do mesmo tipo ou em algum jogo de automação onde só chegar até um inventário items de um certo tipo.

2. **(Pontuação: 1 ponto**) Considere o seguinte código em C# dentro de um script da Unity:

```
public class Exemplo : MonoBehaviour
{
    public int[] numeros = new int[3];

    void Start()
    {
        numeros[0] = 10;
        numeros[1] = 20;
        numeros[2] = 30;
    }
}
```

Com base no código acima, assinale a alternativa correta:

a) [ ] - O array numeros pode ser redimensionado durante a execução para adicionar mais elementos.
b) [ ] - Os valores atribuídos ao array só podem ser acessados dentro do método Start().
c) [ ] - O array numeros armazena três inteiros e cada valor pode ser acessado por seu índice.
d) [ ] - O array numeros é automaticamente convertido em uma lista dinâmica pelo Unity.

3. **(Pontuação: 2.5 pontos**) Você está desenvolvendo um sistema simples de gerenciamento de inimigos para um jogo. Um dos requisitos é armazenar informações básicas dos inimigos (nome, vida e posição inicial) usando uma struct, e instanciá-los no Start() da cena. Analise o código incompleto abaixo e **complete o código** onde indicado, definindo corretamente a struct e instanciando dois inimigos diferentes. Depois, Adapte a struct para incluir um método que calcula a distância entre ele até outro inimigo, usando o campo de posição (Vector3). Demonstre o uso desse método no Start().

```
1   using UnityEngine;
2
3   public class GerenciadorDeInimigos : MonoBehaviour
4   {
5       // 1. Declare aqui a struct 'Inimigo' com: nome (string), vida (int), posição inicial (Vector3
6       //    e um método que recebe outro Inimigo e retorna a distância entre suas posições
7
8       void Start()
9       {
10          // 2. Crie duas variáveis do tipo Inimigo, com valores diferentes
11
12          // 3. Exiba os dados no Console usando Debug.Log
13
14          // 4. Use o método da struct para calcular a distância entre os dois inimigos
15      }
16  }
```

4.

```
public class GerenciadorDeInimigos : MonoBehaviour
{
    5 referências
    public struct Inimigo
    {
        public string nome;
        public int vida;
        public Vector3 posInicial;

        1 referência
        public void PosicoesInimigos(Inimigo other)
        {
            float distancia = Vector3.Distance(other.posInicial, posInicial);
        }
    }
    // Start is called once before the first execution of Update after the MonoBehaviour is created
    Mensagem do Unity | 0 referências
    void Start()
    {
        Inimigo inimigoA = new Inimigo();
        inimigoA.posInicial = new Vector3(5,2,0);

        Inimigo inimigoB = new Inimigo();
        inimigoB.posInicial = new Vector3(8, 3, 0);

        Debug.Log("inimigo A - " + inimigoA.posInicial);
        Debug.Log("inimigo B - " + inimigoB.posInicial);

        inimigoA.PosicoesInimigos(inimigoB);
    }
```

5. **(Pontuação: 5 pontos**) Você deve implementar um mini-jogo inspirado no clássico "Senha (Mastermind)". O objetivo é que o jogador adivinhe uma sequência secreta de 4 cores. Cada tentativa é registrada e comparada com a senha. E então, o jogo exibe se a tentativa foi correta ou não.

Requisitos obrigatórios:

- Crie uma struct CorSenha, que represente uma cor da senha. Ela deve conter:
  - Um índice (int), representando a posição na senha.
  - A senha deve ser armazenada em um array de 4 posições (fixo). Exemplo: [Vermelho, Azul, Verde, Amarelo]
  - A dica (quantas acertou e quantas errou nesta tentativa)
- As tentativas do jogador devem ser armazenadas em uma List<CorSenha>, representando as cores escolhidas na tentativa atual.
- O código deve comparar a tentativa com a senha e exibir:
  - Quantas cores estão na posição correta
  - Quantas cores estão corretas, mas na posição errada
- Mostrar em texto na tela as seguintes informações
  - Quantas tentativas foram realizadas até o momento
  - Qual o resultado da última tentativa.
- Para inserir a senha, o usuário terá 5 botões onde:
  - 1 para cada pino da senha, ao clicar, avança para a próxima cor (ciclando caso chegue no ultimo index)
  - 1 Para aceitar a senha

Critérios de correção:

- Uso correto de struct, array e lista.

- Comparação funcional entre senha e tentativa.
- Organização do código e uso de boas práticas básicas.
- Exibição clara das tentativas e resultados no console.
- O aluno pode usar arrays fixos para representar as tentativas.

Pontos extras:

- Resultados de forma visual (com as bolinhas indicando cada cor da senha)
- Mostrar resultados com bolinhas brancas e pretas (Branca: Cor certa no lugar errado, Preta: Cor certa no lugar certo)
- Mostrar todas as tentativas na tela.