IPRJ – TÓPICOS ESPECIAIS EM ENGENHARIA DE SOFTWARE (JOGOS II)

LISTA DE EXERCÍCIOS – UNITY 3D

1. O que é um *Prefab* na Unity3D? Descreva pelo menos um exemplo de uso.

2. Ao criar uma cena na Unity3D é possível associar objetos hierarquicamente. Qual a utilidade de tal hierarquia? Descreva pelo menos um caso de uso.

3. Na Unity3D, todos objetos de um jogo são *GameObjects*. *GameObjects* podem ser compostos por inúmeros componentes. Qual componente existe por padrão em qualquer *GameObject*? Descreva as informações armazenadas por esse componente e qual a sua utilidade na criação de um jogo.

4. A Unity3D possui 4 tipos de luzes que podem ser utilizado de diferentes maneiras na criação de uma cena. Descreva o funcionamento de cada tipo de luz e cite pelo menos um exemplo de uso para cada tipo.

5. O que é um Skybox? Descreva o conceito e apresente pelo menos um exemplo de uso.

6. Uma das principais propriedades que define como as sombras são geradas pela Unity3D é o tipo de sombra (*Shadow Type*), o qual pode ser *Soft Shadows* ou *Hard Shadows*. Qual a diferença entre estes dois tipos de sombra?

7. O que é um *Material* na Unity3D?

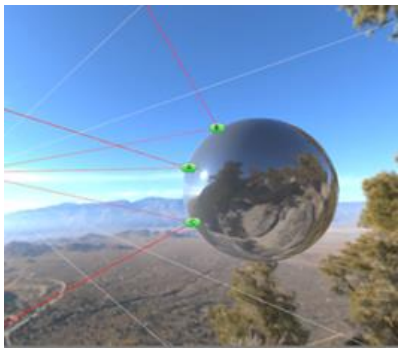
8. O que é um *Shader*?

9. O que é uma textura na Unity3D?

10. Uma das principais propriedades do *Standard Shader* é o modo de rendeização (*Rendering Mode*), o qual pode ser *Opaque*, *Cutout*, *Transparent* ou *Fade*. Descreva a utilidade de cada um destes modos.

11. Descreva os passos necessários para criar um material para representar uma janela de vidro quebrada e transparente na Unity 3D (como a mostrada na figura abaixo).

12. Quais propriedades de um material devem ser alteradas para se criar um objeto reflexivo (como mostrado na figura abaixo)?



13. O que é *Bump Mapping*/*Normal Mapping*? Descreva o funcionamento da técnica e apresente pelo menos um exemplo de uso.

14. Qual diferença entre *Normal Mapping* e *Height maping*?

15. Para que serve o *Occlusion Map* do *Standard Shader*?

16. A propriedade *Emission* do *Standard Shader* é usada para definir a cor e intensidade da luz emitida pela superfície do material. Essa luz é capaz de afetar outros objetos?

17. Para que servem as propriedades *Secondary Maps* e *Detail Mask* do *Standard Shader*?

18. Descreva os passos necessários para adicionar vento às árvores de um terreno.

19. O que é um *Billboard*? Descreva o conceito e apresente pelo menos um exemplo de uso.

20. Descreva o funcionamento do editor de árvores da Unity3D apresentando os conceitos de *Branch Groups*, *Branch Levels* e *Leaf Groups*.

21. O que é um sistema de partículas? Descreva pelo menos dois exemplos de uso.

22. Utilize os conceitos e as propriedades básicas de um sistema de partículas para descrever o ciclo de vida de uma partícula dentro de um sistema de partículas.

23. O que são *Sub Emitters* em um sistema de partículas? Descreva o conceito e apresente pelo menos um exemplo de uso.

24. Qual a função da propriedade *Rotation over Lifetime* de um sistema de partículas? Apresente pelo menos um exemplo de uso desta propriedade.

25. Um item possui as seguintes características:

    - O item possui um estado que determina se ele está ativo ou não ativo;
    - O item possui um tipo (vida, munição ou poder extra);
    - Todo item possui uma quantidade. No caso da vida, essa quantidade representa a quantidade de vida. No caso da munição, representa a quantidade de munição. E no caso do poder extra, representa a quantidade de poder extra.
    - O item pode ser coletado totalmente ou parcialmente. Ao ser coletado parcialmente, a quantidade do item é reduzida de acordo com a quantidade coletada. Ao ser coletado totalmente, a quantidade do item é reduzida a zero.
    - Se a quantidade do item for menor ou igual à zero, o item muda o seu estado atual para não ativo.
    - Um item somente pode ser coletado se ele estiver ativo.

    Crie uma classe em C# para representar o item, incluindo todos os atributos e métodos que possibilitem o seu funcionamento. Em seguida, instancie um objeto do tipo item e teste seus métodos.

26. É comum que um script/componente precise acessar outros objetos. Tal acesso pode ser realizado através de *links usando variáveis*, através de uma *busca por nome* ou *busca por tag*. Supondo que você está criando uma classe para representar um inimigo e precisa que esse inimigo tenha acesso a posição atual do jogador, crie três versões da classe inimigo: a primeira deve acessar o jogador através de *links usando variáveis*; a segunda deve acessar o jogador através de uma *busca por nome*; e a terceira deve acessar o jogador através de uma *busca por tag*;

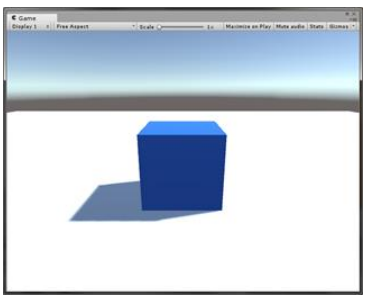
27. A variável *Time.deltaTime* é extremamente importante no desenvolvimento de aplicações usando a Unity3D. Em relação a essa variável, responda:

    a) O que a variável *Time.deltaTime* representa?

    b) Explique qual é a principal utilidade da variável *Time.deltaTime* no desenvolvimento de uma aplicação em Unity3D.

28. Quais as diferenças entre as funções de evento Update() e FixedUpdate() na Unity3D?

29. Considerando a existência de uma cena como a ilustrada na figura abaixo:



Levando em consideração que o cubo azul possui um *Rigidbody*, crie um script para controlar o cubo azul pelo teclado. O cubo deve mover-se em 4 direções (para direita, para esquerda, para frente e para traz). Considere que o script será associado ao cubo azul.

30. Considerando a existência de um *Prefab* de um cubo, crie um script para instanciar uma parede formada por 100 cubos (10 x 10).

31. Considerando a existência de um jogo que utiliza 3 tipos de itens que podem ser coletados pelo jogador: vida, munição e poder extra. Crie um script para instanciar e gerenciar os itens do cenário dinamicamente:
    - O script será associado a um *GameObject* vazio chamado *ItemManager*;
    - Os objetos que representam os itens devem ser associados ao script através de 3 *Prefabs* (um para cada tipo de item);
    - O script deve permitir que as posições dos itens sejam definidas manualmente através da interface da Unity3D. A quantidade de itens também deve ser especificada pela interface;
    - Quando o *ItemManager* for instanciado, os itens devem ser instanciados em suas devidas posições do cenário;
    - O script deverá ter um método chamado "*DestruirItens*", o qual deve destruir todos os itens quando executado.

32. Implemente um sistema de moedas coletáveis para um jogo. O sistema deve funcionar da seguinte maneira: sempre que o jogador aproximar-se de uma moeda, essa moeda deve desaparecer e o número de moedas do jogador deve ser incrementado em 1.
    - Devem ser implementados dois scripts/classes: um para ser associado às moedas e um para ser associado ao jogador;
    - O número de moedas coletadas deve permanecer armazenado na classe do jogador;
    - O objeto que representa o jogador possui a *tag* "*Player*";
    - Não é necessário exibir o número de moedas coletadas na tela.

33. Para solucionar um quebra-cabeça, o jogador precisa encontrar um determinado número de pistas espalhadas pelo cenário do jogo. Crie uma classe genérica para gerenciar as pistas do jogo:
    - Em cada fase do jogo existirá uma instancia do script criado;
    - Cada fase terá um determinado número de pistas;
    - O script deve armazenar as pistas encontradas pelo jogador em uma estrutura de dados do tipo lista;
    - Internamente, as pistas são representadas por palavras (*strings*);
    - O script deve possuir um método chamado "*PistaEncontrada*" que recebe como parâmetro uma pista e a armazena na lista de pistas;
    - O script deve possuir um método chamado "*JaFoiEncontrada*" que recebe como parâmetro uma pista e retorna: *true* caso a pista já tenha sido encontrada; e *false* caso a pista ainda não tenha sido encontrada;

- O script deve possuir um método chamado "*EncontrouTodas*" que retorna: *true* caso todas as pistas da fase já tenha sido encontradas; e *false* caso ainda existam pistas não encontradas.

34. Ao implementar o inimigo em um jogo de tiro em primeira pessoa é necessário saber se o jogador está dentro do campo de visão do inimigo para que ele atire no jogador. Implemente um método chamado "*PossoVer*", o qual deve receber com parâmetro o *GameObject* do jogador e retornar: *true*, caso o jogador esteja no campo de visão do inimigo; ou *false*, caso não esteja. Considere que o campo de visão do inimigo é de 90 graus. Considere também que este método será implementado na classe Inimigo, a qual está associada ao *GameObject* do inimigo.

35. Em um jogo de estratégia o jogador controla diversas unidades simultaneamente e é capaz de atribuir mais de uma tarefa para uma mesma unidade. Implemente um script/classe para gerenciar as tarefas realizadas por uma unidade.
    - Internamente, as pistas podem ser representadas por palavras (*strings*);
    - A classe possui um atributo que determina a tarefa atual sendo executada;
    - A classe deve possuir um método chamado "*AdicionarTarefa*", o qual deve adicionar uma tarefa ao conjunto de tarefas da unidade;
    - A classe deve possuir um método chamado "*ExecutaProximaTarefa*", o qual deve retirar a próxima tarefa a ser executada do conjunto de tarefas e atribui-la a tarefa atual. O método também deve exibir o nome da tarefa que esta sendo executada (pode ser no console);
    - As tarefas devem ser executadas na ordem em que elas foram atribuídas a unidade;

36. Qual a função de um *Rigidbody* em um objeto? Descreva a função e apresente pelo menos um exemplo de uso.

37. Um *GameObject* com um *Rigidbody* e com a opção *IsKinematic* desabilitada deve ser movido pelo método *Translate* do seu *Transform*? Por quê?

38. Em quais casos *Mesh Colliders* devem ser usados?

39. Em quais casos *Mesh Colliders* não devem ser usados?

40. Como a área de colisão de um objeto 3D com uma malha complexa pode ser representada sem utilizar um *Mesh Collider*?

41. É correto movimentar objetos que possuem um *Collider*, mas não possuem um *Rigidbody*? Por quê?

42. Em quais casos a propriedade *IsKinematic* de um *Rigidbody* deve ser habilitada?

43. Descreva os passos necessários para aumentar fricção de uma superfície na Unity3D.

44. Qual diferença entre *Dynamic Friction* e *Static Friction* em um material físico (*Physics Material*)?

45. O que é um *Trigger*? Cite pelo menos dois exemplos de uso.

46. Quando ocorre uma colisão entre dois objetos, a Unity 3D executa automaticamente três funções de eventos em todos os scripts anexados aos objetos envolvidos. Descreva quais são essas funções de evento e em quais momentos da colisão cada uma delas é executada.

47. Em um jogo de tiro em primeira pessoa é necessário implementar o comportamento das balas disparadas pelas armas. Quando instanciadas, as balas devem mover-se para frente até que algum objeto físico seja atingido. Quando a bala atingir um objeto físico, ela deve ser destruída. Implemente um script para ser associado a um *Prefab* que representa as balas. Esse *Prefab* será instanciado sempre que um disparo for realizado. O seu script deve controlar apenas o comportamento da bala.

48. O que é um *Character Controller*? Descreva o conceito e apresente pelo menos um exemplo de uso.

49. O que é um *Joint*? Descreva o conceito e apresente exemplos de uso.

50. Qual diferença entre o *Hinge Joint* e o *Spring Joint*? Cite exemplos de uso para os dois tipos de *Joints*.

51. Supondo a existência de um personagem com um *Character Controller* associado e configurado, implemente um script para permitir que o jogador o jogador possa controlar esse personagem com as setas direcionais do teclado (andar para frente, andar para traz, virar para a direita e virar para a esquerda).

52. Qual a função do componente *Constant Force*? Cite exemplos de uso.

53. Supondo a existência de um cubo com um *Rigidbody* associado a ele, implemente um script para fazer com que o cubo gire para a direta quando a seta da direita do teclado for pressionada e gire para a esquerda quando a seta da esquerda for pressionada. Considere que o script será associado ao cubo.

54. Descreva o processo de criação de uma animação para um personagem.

55. O que é um *Animator Controller*?

56. Na Unity 3D, as animações dos personagens são controladas por máquinas de estados. Explique como as animações são representadas em uma máquina de estados e como a máquina de estados funciona.

57. Considerando a existências de um modelo 3D de um personagem com as seguintes animações: parado, andando, correndo, pulando e atacando.

    a) Crie uma máquina de estados para controlar os comportamentos e animações do personagem. A máquina de estados deve ser capaz de controlar todas as animações do personagem (parado, andando, correndo, pulando e atacando).

    b) Implemente um script para utilizar a máquina de estados criada para controlar e animar o personagem. Considere que o seu script será associado ao personagem e que ele já contém um *Character Controller*.

58. O *Canvas* utilizado para a criação de interfaces com o usuário possui 3 modos de renderização (*Render Mode*): (1) *Screen Space – Overlay*; (2) *Screen Space – Camera*; e (3) *World Space*. Descreva as diferenças entre estes 3 modos de renderização e cite exemplos de situações onde cada modo é mais adequado.

59. Supondo a existência de uma interface com um botão (*Button*), um campo de texto (*Input Field*) e um texto (*Text*). Crie um script para copiar o conteúdo digitado no campo de texto para o texto no momento que o botão for pressionado. Considere que o script será associado ao botão.

60. Descreva o funcionamento do sistema de áudio da Unity 3D, incluindo as definições de *Audio Source* e *Audio Listener*.

61. A forma como um arquivo de áudio é carregado pela Unity 3D depende da propriedade *Load Type* definida no clipe de áudio, a qual pode ser: (1) *Decompress On Load*; (2) *Compressed In Memory*; ou (3) *Streaming*. Descreva as diferenças entre esses 3 modos de carregamento de áudio.

62. Implemente um script para fazer com que um determinado personagem emita um som quando o jogador se aproximar dele. Considere que o personagem possui um *Audio Source* já configurado e que o jogador possui a *tag* "*Player*".

63. A inteligência artificial em jogos busca criar inimigos que sejam invencíveis. Verdadeiro ou Falso? Justifique sua resposta.

64. O que é a "ilusão de inteligência" na inteligência artificial aplicada a jogos?

65. O que são *Waypoints* e qual a sua aplicação na inteligência artificial de jogos?

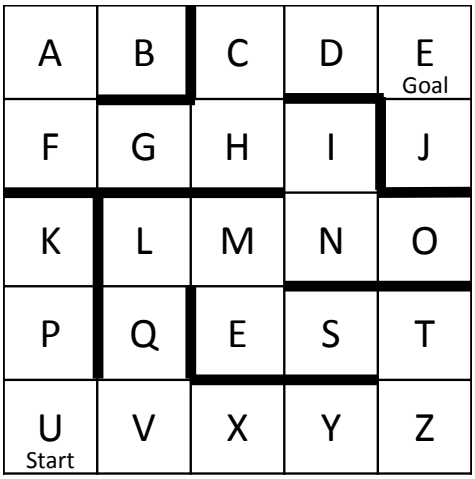
66. Explique o processo de criação de *Waypoints* baseados em Pontos de Visibilidade. Quais as vantagens e desvantagens dessa técnica?

67. Explique o processo de criação de *Waypoints* baseados em Tiles/Grid. Quais as vantagens e desvantagens dessa técnica?

68. Em um labirinto, mostrado na figura a seguir, um robô é colocado na célula inicial indicada por "U" e deve encontrar um caminho até a saída, denotada pela letra "E". O robô não pode se mover na diagonal, somente acima, abaixo, direita e esquerda. Ele também não pode atravessar paredes (as linhas mais grossas da grade) ou as bordas do labirinto, de modo que ele é forçado a contornar obstáculos. Felizmente, o robô possui um mapa do ambiente. A solução é o caminho mais curto até a saída e todos os movimentos do robô possuem os mesmos custos.

    (a) Defina uma função heurística para ser utilizado pelo algoritmo A*.

    (b) Realize uma busca utilizando o algoritmo A* para encontrar o melhor caminho para chegar a letra E partindo da letra U. Construa a árvore de busca criada pela execução do algoritmo apresentando os valores de $f(n)$, $g(n)$ e $h(n)$ para cada nó.



69. Como uma máquina de estados finitos pode ser utilizada para codificar o comportamento autônomo de um personagem? Descreva a técnica e apresente as vantagens e desvantagens de sua utilização.

70. Máquinas de estados finitos são uma das formas mais usadas para implementar o comportamento autônomo de personagens em jogos. Porem, à medida que a complexidade dos comportamentos dos personagens aumenta, as máquinas de estados finitos tendem a crescer de forma descontrolada. Como é possível reduzir essa complexidade?