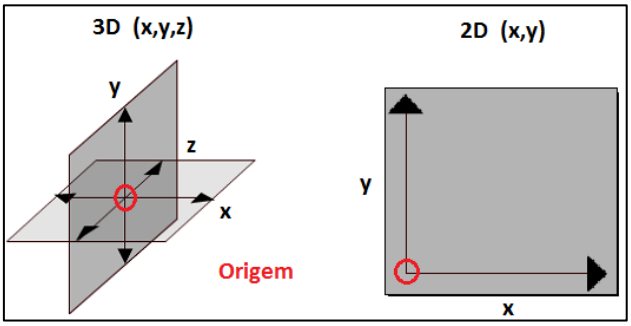
LABORATÓRIO: CRIANDO O JOGO ROLL A BALL

Parte-1

1. Crie um novo projeto 3D e insira uma textura para o céu (*Sunny2-Skybox*) para a cena ficar mais clara (Edit->Render Settings).
2. Criar as seguintes pastas na janela Project: _scenes, _scripts, _materials.
3. Posicionar o objeto Main Camera em (0;3;-6)
4. Adicionar um objeto Diretional Light, renomear para Sol e definir:
    a. A posição como (0; 3; 0)
    b. A rotação como (50; 60; 0)
    c. A propriedade “Shadow Type” para “Hard Shadows“
5. Criar um objeto plano (Plane) com escala (2;1;2)
    a. Renomeá-lo para Chao (será o chão do jogo)
6. Criar uma esfera (Sphere) e posicionar em (0;0.5;0) para ela ficar no topo do plano.
    a. Renomeá-la para Jogador
7. Criar um novo material (janela Project, opção Create->Material) dentro da pasta _materials e mudar sua cor para azul-escuro.
    a. Renomeá-lo para MaterialChao
8. Atribuir o material criado ao plano.
9. Desejamos então movimentar a esfera através do teclado. Para mover e controlar a esfera teremos que usar física (*physics*) e para isso o objeto precisa possuir um componente Rigidbody.
    a. Adicionar um componente Rigidbody à esfera (Add Component->Physics->Rigidbody).
10. Criar um novo script na pasta _scripts e renomear para ControlarJogador.
    a. OBS: Neste caso, iremos inserir o código na função FixedUpdate() e não na função Update(). A função Update() é chamada a cada frame e depende do frame-rate, podendo ser chamada mais ou menos vezes de acordo com o *fps* (*frames per second*) do dispositivo. Já a função FixedUpdate() não é baseada no frame-rate e, por isso, o processamento de algoritmos que envolvem física (*physics*) são realizados de forma mais consistente e constante.
        i. Caso de exemplo: temos um bloco de código criado para aumentar em 10 a força para movimentar um determinado objeto. Caso esse código seja adicionado na função Update(), se o jogo estiver executando a 100 fps, serão adicionados 1000 à força a cada segundo. Mas se o mesmo jogo for executado a 30 fps, serão adicionados somente 300 à força a cada segundo. Se este código for inserido na função FixedUpdate(), o jogo irá se comportar de forma constante não importando o frame-rate do dispositivo.
        ii. Quando realizamos cálculos na função FixedUpdate(), não precisamos usar o “Time.deltaTime” porque esta função possui um temporizador confiável, sendo independente do frame-rate do dispositivo.

11. Pausa para explicarmos a relação entre o espaço dimensional da cena 3D e o da tela 2D onde será executado o jogo. Perceba que o eixo Y da cena 3D refere-se à altura em que o objeto está na cena, o eixo X indica se o objeto está mais à direita ou mais à esquerda, e o eixo Z se o objeto está mais perto ou mais longe da visão da câmera. Na tela 2D, temos apenas as coordenadas X e Y e, portanto, usamos X e Y da tela 2D para definirmos a posição X e Z da cena 3D.



12. Inserir o seguinte código no script ControlarJogador:

```
private var rb: Rigidbody;
public  var velocidade: float = 10;

function Start () {
    //obtem o componente Rigidbody do gameObject
    rb = gameObject.GetComponent.<Rigidbody>();
}

function FixedUpdate () {
    //Obtem as coordenadas dos eixos Horizontal e Vertical mapeadas
    //atraves das teclas de direçao (setas ou ASDW).
    var movHorizontal: float = Input.GetAxis("Horizontal");
    var movVertical: float   = Input.GetAxis("Vertical");

    //Vetor(x,y,z) para aplicar força de movimento na esfera de acordo
    //com as teclas de direçao pressionadas.
    //OBS: lembrando que Y refere-se a altura em que o objeto estah em
    //     relaçao ao "chao" da cena. X e Z devem ser usados para mapear
    //     as coordenadas (x,y) da tela - a qual eh 2D.
    var movimento: Vector3 = new Vector3(movHorizontal, 0, movVertical);

    //aplica a força de movimento a esfera
    rb.AddForce(movimento * velocidade);
}
```

13. Executar o jogo e usar as setas de direção para controlar a esfera.

14. Há um problema com a câmera do jogo, pois ela não está exibindo toda a cena porque está fixa em uma posição. Se afastarmos a câmera poderemos ver toda a cena, mas a parte mais ao fundo ficará muito distante da visão do jogador. Se tentarmos aninhar a Main Camera dentro da esfera, a câmera irá acompanhar a posição da esfera, mas também irá girar como ela (faça isso e veja).

15. Para funcionar corretamente, teremos que controlar a Main Camera via script. Antes vamos posicionar a câmera em (0;7;-10) e rotacionar em (45;0;0). Em seguida, criar um script e renomear para PosicionarCamera.

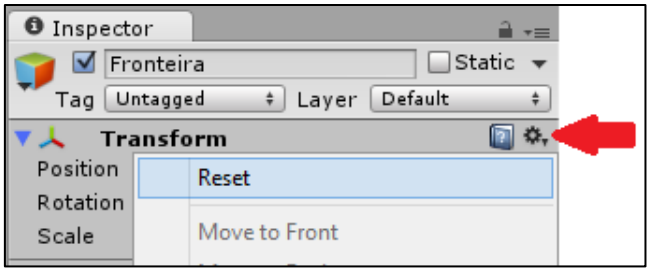
16. Escrever o seguinte código nesse script:

```
//objeto a ser seguido
var objetoAlvo: GameObject;
//distancia da camera ao objetoAlvo
private var distancia: Vector3;

function Start () {
    //calcula a distancia (definida na cena) entre a camera e o objetoAlvo
    distancia = gameObject.transform.position - objetoAlvo.transform.position;
}

//LateUpdate() eh chamada apos todos os Update() dos objetos serem chamados.
//Assim garantimos que o este codigo sera executado no final, pois se colocarmos
//o codigo dentro do Update() da camera, nao saberemos se sera executado sempre
//depois do Update() do objetoAlvo, onde este pode ter sido movido.
function LateUpdate () {
    //posiciona a camera em relaçao a posico do objetoAlvo
    //mantendo a distancia original definida na cena.
    gameObject.transform.position = objetoAlvo.transform.position + distancia;
}
```

17. Adicionar o script PosicionarCamera na Main Camera e definir sua propriedade objetoAlvo para a esfera Jogador (basta arrastar e soltar a esfera dentro desta propriedade).

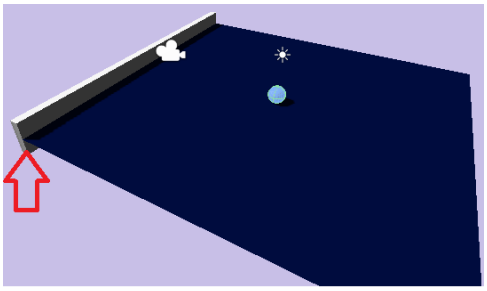
18. Execute o jogo e verifique a câmera acompanhando os movimentos da esfera.

19. Agora iremos adicionar as fronteiras laterais da nossa cena para evitar que a esfera saia do plano e caia no infinito. Vamos organizar nossa fronteira na janela Hierarchy através da criação de um gameObject vazio (menu *Game Object->Create Empty*). Este objeto vazio será utilizado como uma pasta onde iremos inserir os nossos muros que ficarão ao redor do plano, similar às pastas que criamos para organizar o nosso projeto na janela Project (_scenes, _scripts, _materials etc). Após criar o objeto vazio, vamos renomeá-lo para Fronteira e posicioná-lo na origem da cena, em (0;0;0). Podemos fazer isso mais facilmente clicando no ícone de configurações do componente Transform e selecionando Reset (ver imagem).

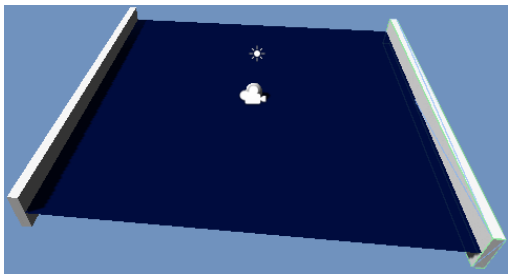


20. Agora criaremos um cubo na cena para representar o muro do lado oeste do plano. Após criar o cubo, renomeie-o para MuroOeste, aninhe-o dentro do objeto vazio Fronteira (criado no passo anterior) e posicione-o na origem.

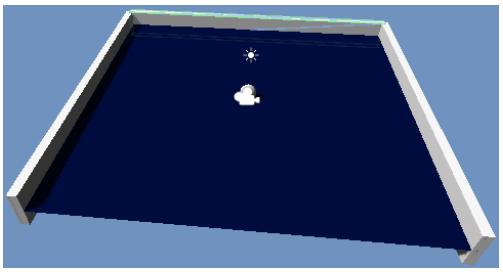
21. Ainda com o objeto MuroOeste selecionado, defina sua escala para (0.5;2;20) e sua posição como (-10;0;0). A cena deverá ficar semelhante à imagem a seguir:



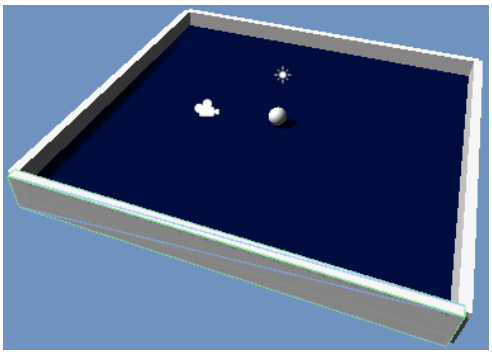
22. Agora vamos duplicar o nosso muro. Selecione-o na janela Hierarchy e pressione “Ctrl+D”. Renomeie o novo objeto para MuroLeste e defina sua posição como (10;0;0). Ficará como a imagem a seguir:



23. Agora duplique novamente o MuroOeste e renomeie-o para MuroNorte. Mude sua escala para (20;2;0.5) e sua posição para (0;0;10). Obteremos a seguinte cena:



24. Por fim, duplique o MuroNorte, renomeie-o para MuroSul, posicione em (0;0;-10). Obtendo a cena a seguir:



25. Execute o jogo e veja como agora a esfera não irá mais sair do plano.

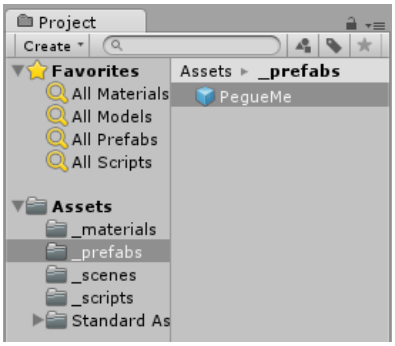
26. Agora vamos criar os objetos que o jogador deverá colidir no jogo para marcar pontos. Adicione um objeto cubo na cena, renomeie-o para PegueMe, posicione-o em (0;0.5;0), altere sua escala para (0.5;0.5;0.5) e defina sua rotação como (45;45;45).

27. Adicione um novo script na pasta _scripts, renomeie-o para GirarCubo e insira-o no objeto PegueMe. Em seguida, insira o código a seguir no script:

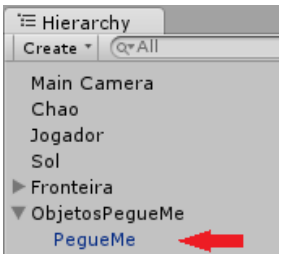
```
function Update () {
    //rotaciona o objeto usando o deltaTime para suavizar a rotaçao, ou seja,
    //torna-la mais lenta.
    gameObject.transform.Rotate(new Vector3(15, 30, 45) * Time.deltaTime );
}
```

28. Execute o jogo e veja que o cubo está animado, fazendo rotações.

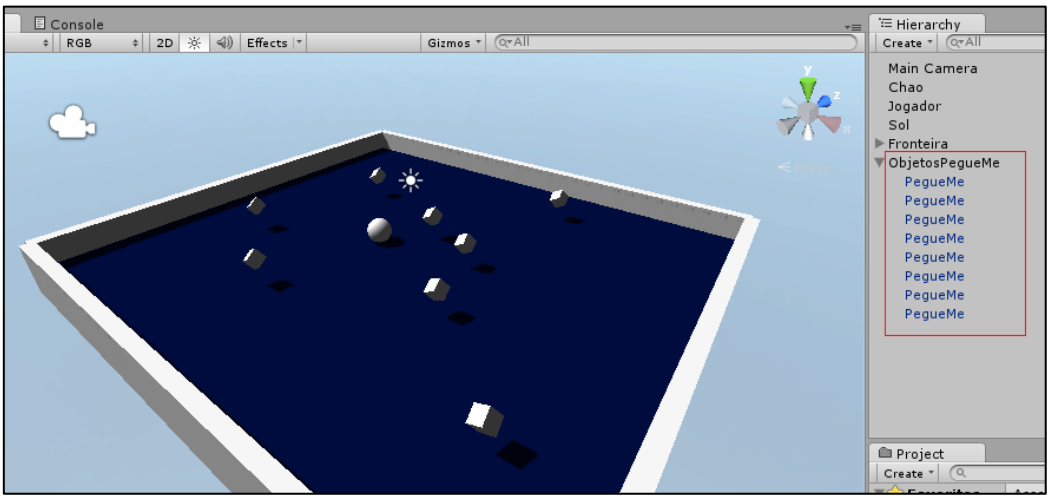
29. Agora iremos transformar o objeto PegueMe em um Prefab, assim poderemos criar vários objetos deste tipo no nosso jogo. Na janela Project, crie uma pasta chamada _prefabs e arraste e solte o objeto PegueMe dentro dela (ver figura). Agora temos um template do objeto PegueMe e podemos criar outros objetos deste tipo a partir deste template.



30. Selecione e delete o objeto PegueMe que foi criado na cena do jogo. Na janela Hierarchy, crie um objeto vazio (Create Empty) e renomeie para ObjetosPegueMe. Arraste e solte o template criado dentro do objeto vazio criado (ver figura).



31. Selecione o objeto PegueMe criado na cena e pressione “Ctrl+D” para duplica-lo na cena. Perceba que o objeto criado se posicionará exatamente em cima do objeto criado anteriormente. Reposicione o objeto recém-criado para tirá-lo de cima do objeto de origem. Repita esse passo até termos na cena 8 objetos do tipo PegueMe (ver figura).



32. Agora vamos mudar a cor dos objetos PegueMe para amarela. Crie um novo material na pasta _materials da janela Project e renomeie para MaterialPegueMe. Em seguida, altere sua cor para amarelo (propriedade Main Color). Agora devemos aplicar o material no ***prefab*** para que todos os objetos PegueMe da cena sejam alterados automaticamente para este material. Para fazer isso, selecione o MaterialPegueMe na janela Project, depois selecione a pasta _prefabs e arraste e solte o material da janela Inspector dentro do prefab PegueMe.

33. Execute e teste o jogo. Mova a esfera para verificar como ela colide com os cubos.