Desenvolvimento com Motores de Jogos II

Aula 03 - Jogo Polygonal Rescue - Parte 1

Material Didático do Instituto Metrópole Digital - IMD
Versão 5.0 - Todos os Direitos reservados

Apresentação

Olá, pessoal! Sejam bem-vindos à aula 3 da nossa disciplina de Motores de Jogos II! Já aprendemos a configurar e utilizar o sistema de entradas do Unity, bem como adicionar um movimento básico a um objeto da cena. Nesta aula vamos melhorar o sistema de movimentos que iniciamos, configurar a velocidade do objeto através do script C# e utilizar Rigidbody para tratar com movimentos mais realísticos.

Preparados para a criação do nosso primeiro jogo? Ele se chamará “Polygonal Rescue”. Nele, o jogador controla um herói que se trata de uma esfera, e esta salvará seus outros amigos polígonos, os quais ficaram presos em um labirinto no mundo 3D.

Objetivos

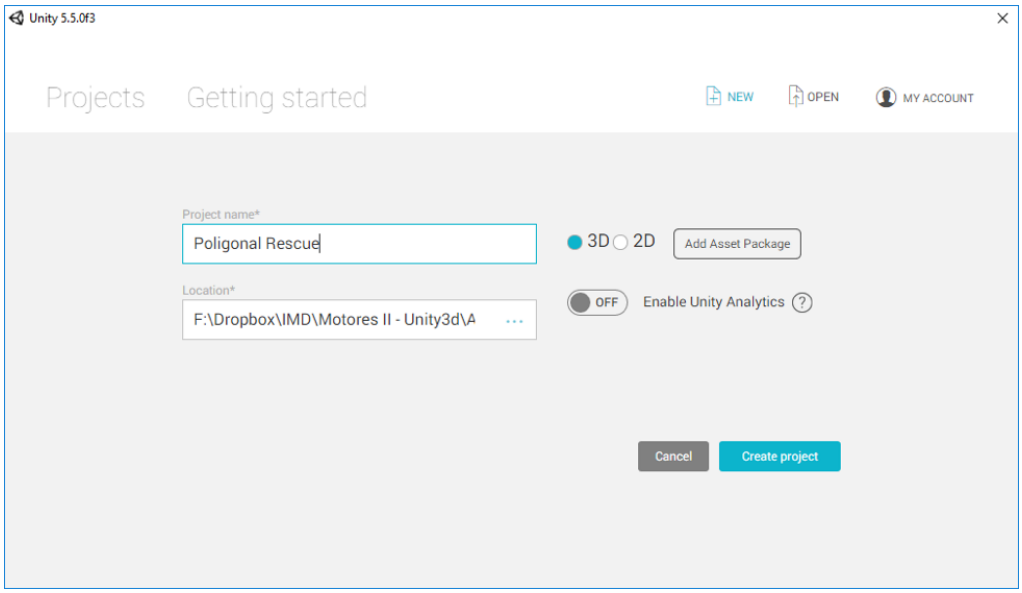
Ao final desta aula, você deverá ser capaz de:

- Conhecer as técnicas e procedimentos para criar o jogo “Polygonal Rescue”;
- Compreender o que são Materials e como utilizá-los;
- Aprender a utilizar o Rigidbody e os movimentos com física.

Preparando o Projeto do Jogo

Para o nosso jogo criaremos um novo projeto chamado “Polygonal Rescue”, veja na Figura 1.

Figura 01 - Criando um novo projeto no Unity.



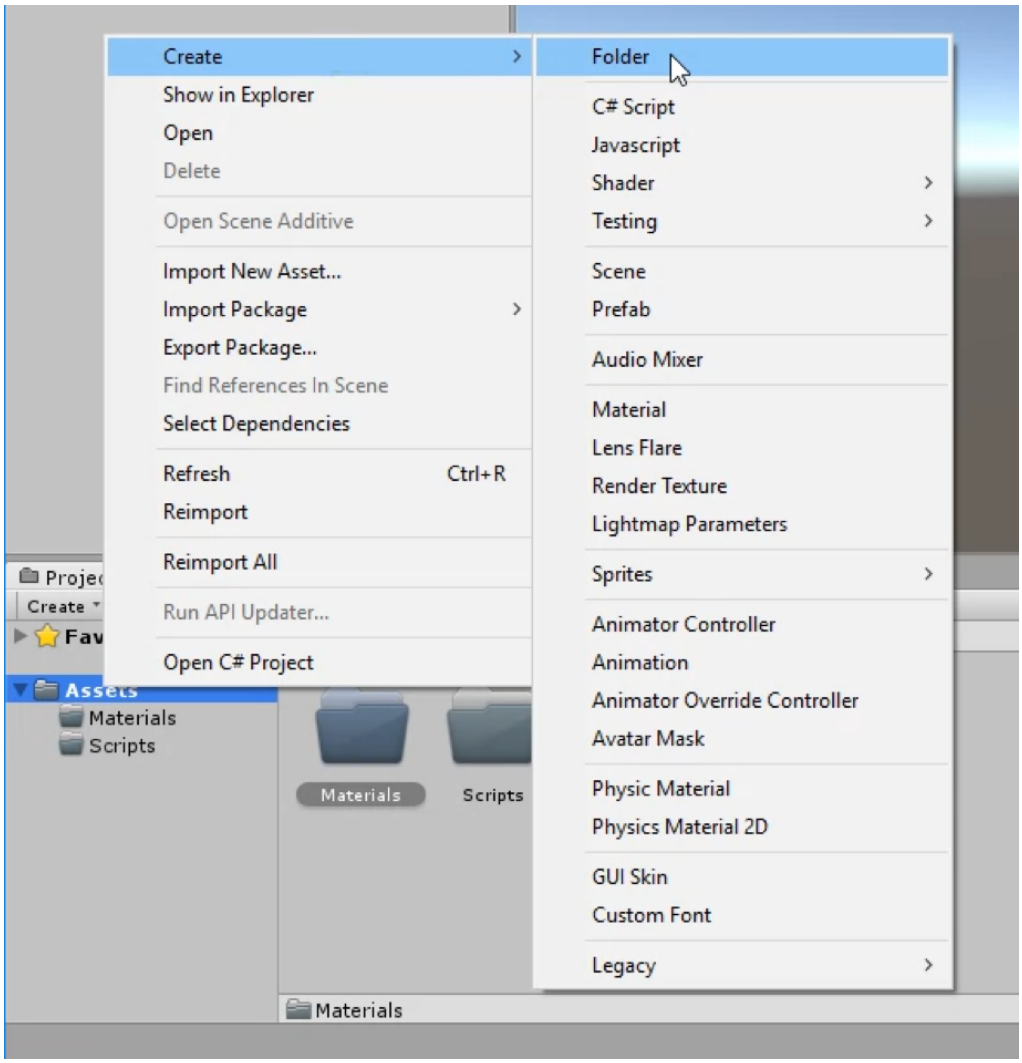
Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Com o projeto criado vá no painel Project e dentro da pasta Assets adicione outras 3 pastas:

1. Pasta “**Scripts**”: será utilizada para guardar nossos scripts C#.
2. Pasta “**Materials**”: é onde salvaremos nossos Materiais, que são elementos do Unity com informações de cores e texturas os quais podem ser aplicados a objetos para mudar sua aparência.
3. Pasta “**Scenes**”: é a pasta onde ficarão as cenas do projeto. Em outras aulas aprenderemos a criar várias fases para o jogo e cada uma delas será uma cena nessa pasta.

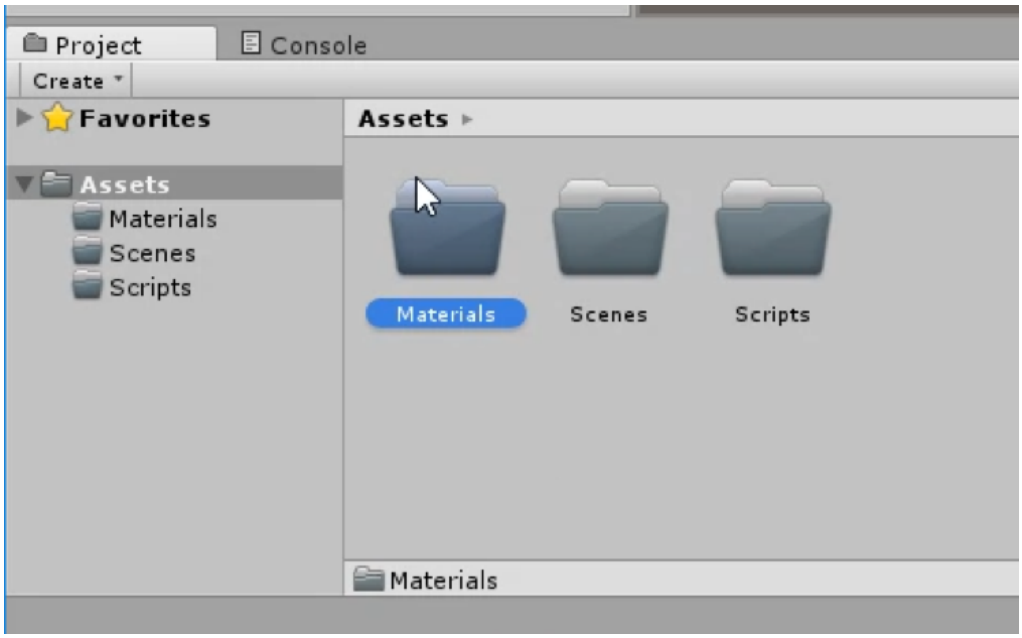
Na Figura 2 você pode ver como se criam as pastas do projeto e, depois de criadas, elas devem estar como na Figura 3.

Figura 02 - Criando as pastas do projeto.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Figura 03 - Projeto com as pastas criadas.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Aproveite para salvar a cena atual dentro da pasta Scenes com o nome “Principal” e pronto, já criamos nosso projeto e as pastas que utilizaremos por enquanto.

Criando a Base do Ambiente

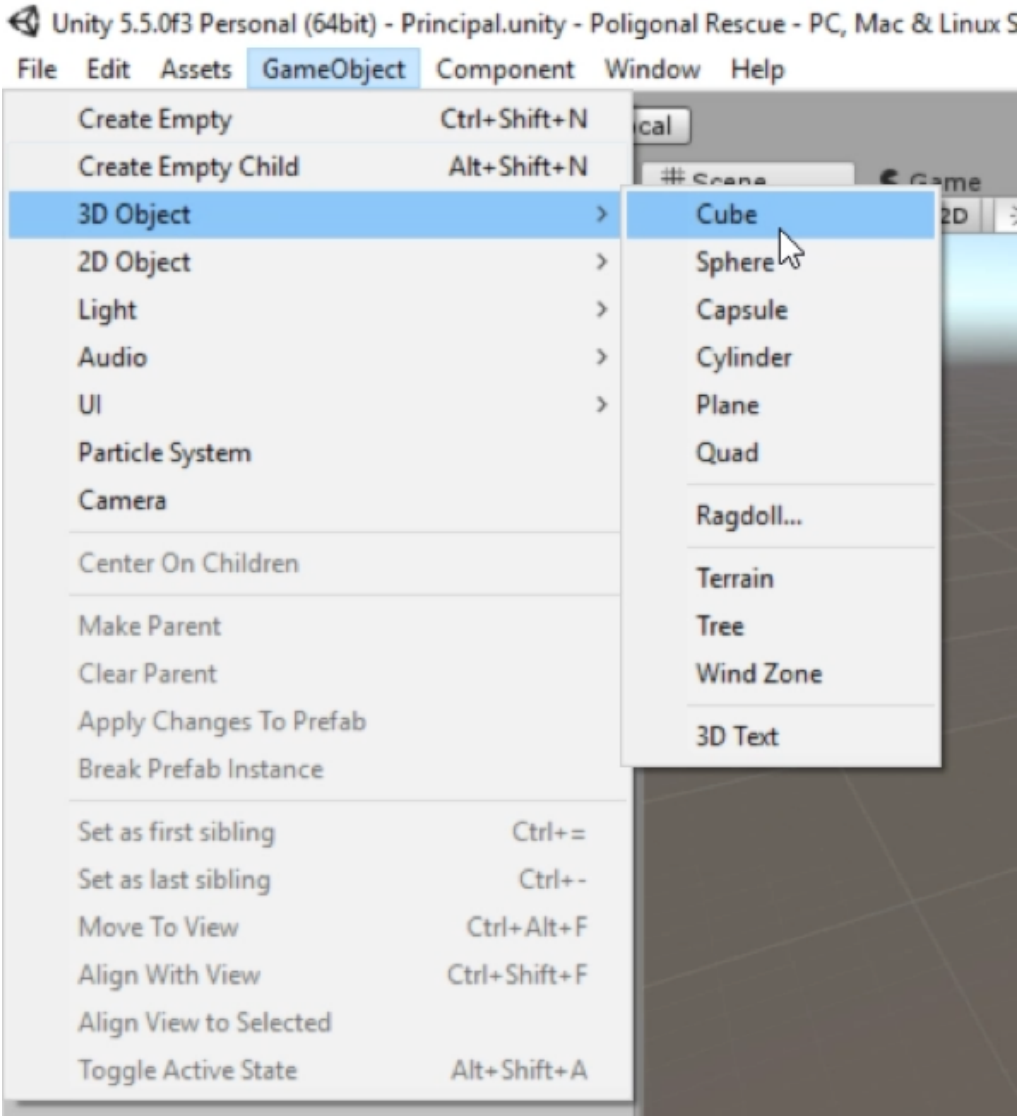
Como dito anteriormente, o jogo Polygonal Rescue se passa em um labirinto onde o jogador controla o personagem principal, que é uma esfera. Nesse labirinto existem outros polígonos que precisam ser salvos e você pode escolher salvá-los ou não. Nas próximas aulas, adicionaremos outras mecânicas ao jogo, como tempo limite para completar cada fase. Mas não se preocupe! Por enquanto esse limite não existirá.

A base do ambiente da nossa primeira fase será um labirinto simples, somente para o jogador aprender a controlar o personagem e avançar para a próxima fase. Para criar o labirinto vamos utilizar **Cubos** com sua escala modificada, tanto para o piso como para as paredes.

Criando o piso do labirinto

Para criarmos o piso do labirinto, inicialmente adicionaremos um **Cubo** na cena para o piso. Para isso clique primeiramente no nome da cena (Principal) no Hierarchy e escolha a opção no menu GameObject->3D Object->Cube, assim como na Figura 4.

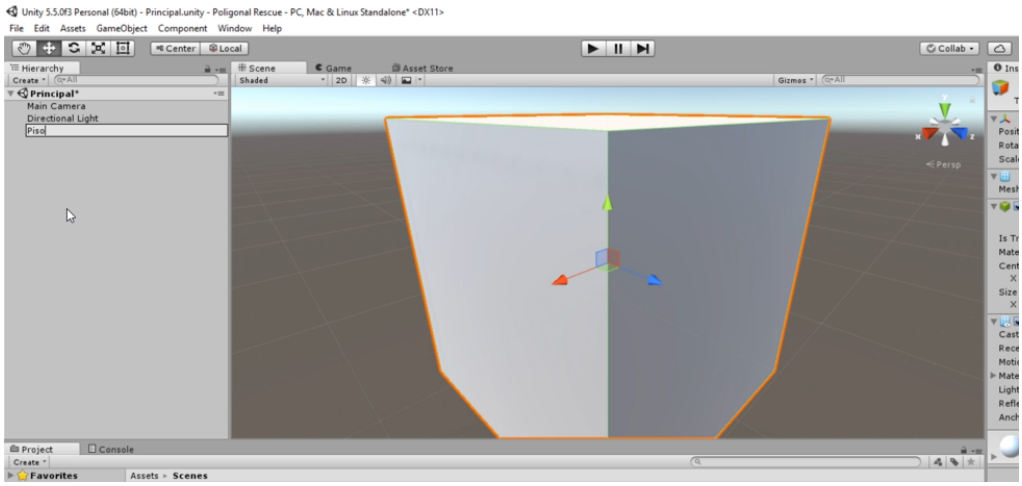
Figura 04 - Adicionando o cubo para o piso.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

É importante clicar inicialmente no nome da cena para que o Cubo fique na raiz da cena, ou seja, não seja filho de nenhum objeto previamente selecionado. Renomeie o Cubo para “Piso”, assim como na Figura 5.

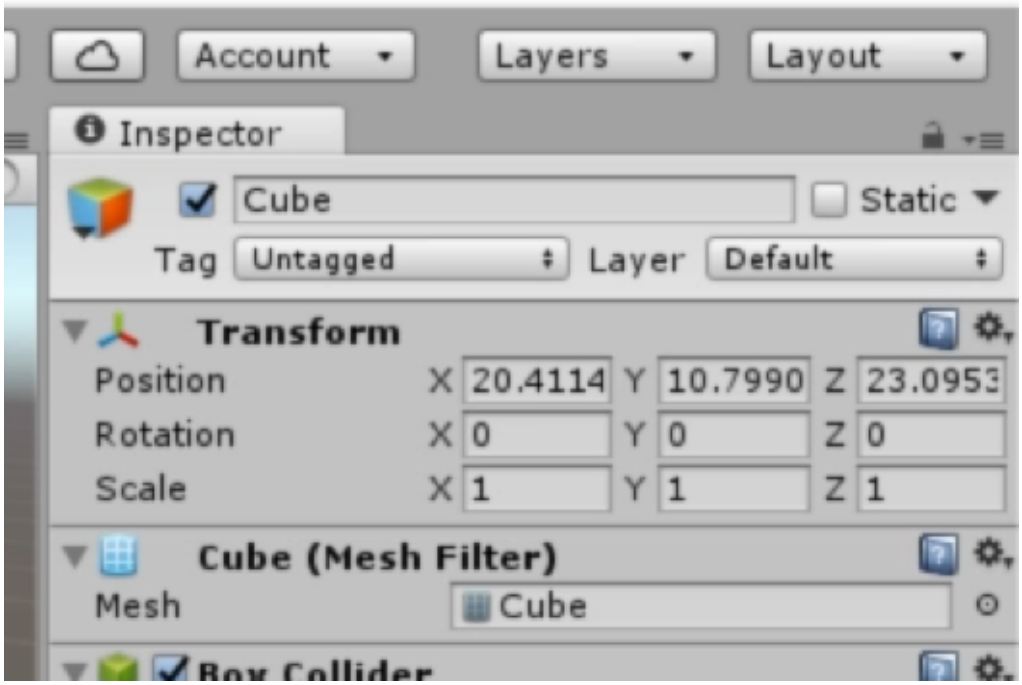
Figura 05 - Renomeando o Cubo para que ele represente nosso piso.



Fonte: Captura de tela do Unity – Game Engine. Disponível em:https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Quando você adiciona GameObjects na cena eles geralmente são criados em posições que nem sempre são realmente desejadas, como é possível verificar clicando no Piso e observando sua propriedade Transform position no Inspector, veja na Figura 6.

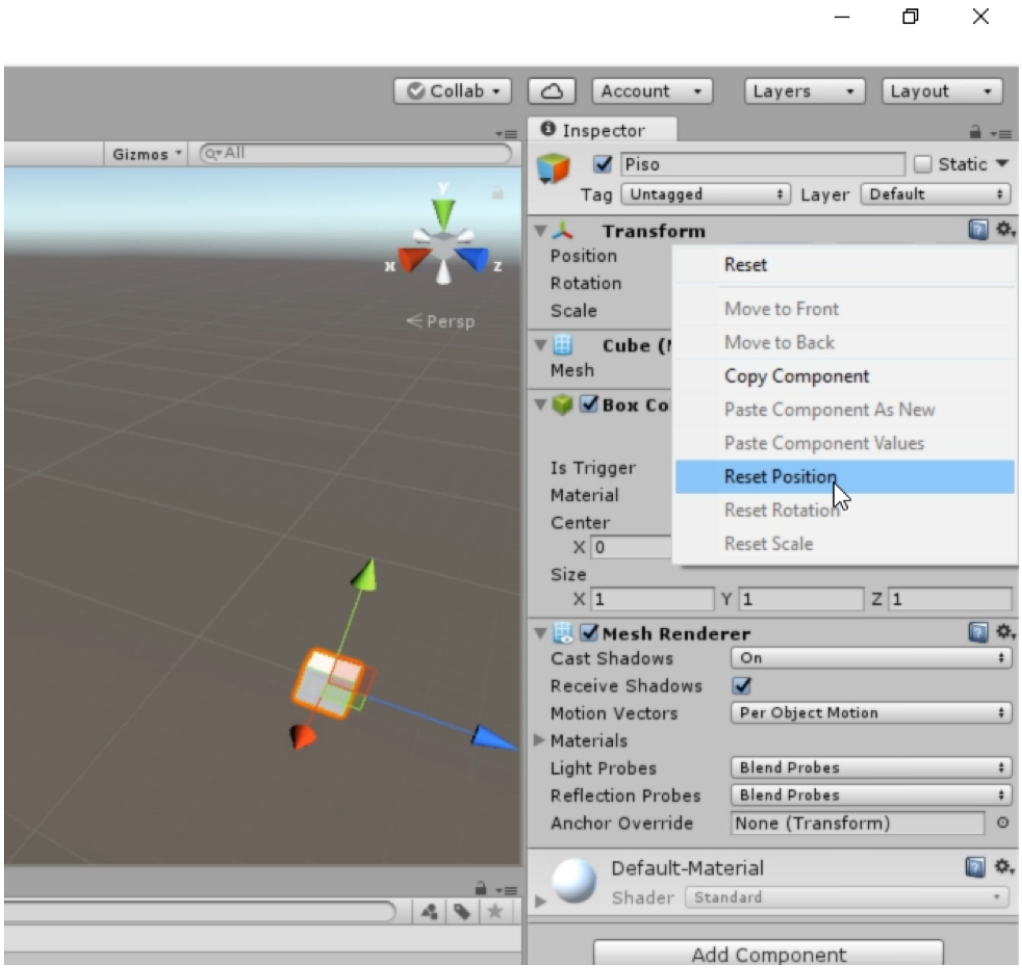
Figura 06 - Posição original do Cubo que representa o piso assim que ele é adicionado na cena.



Fonte: Captura de tela do Unity – Game Engine Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Você pode corrigir isso simplesmente movendo-os, porém algumas vezes é melhor levá-los inicialmente para a origem da cena no intuito de facilitar o seu trabalho. Para tanto, clique no ícone que se assemelha com uma engrenagem no canto superior direito da propriedade **Transform**, escolha a opção **Reset position** e seu Cubo irá para a origem da cena (Posição 0, 0, 0). Veja na Figura 7.

Figura 07 - Cubo sendo levado à origem da cena através da opção Reset Position do Transform.



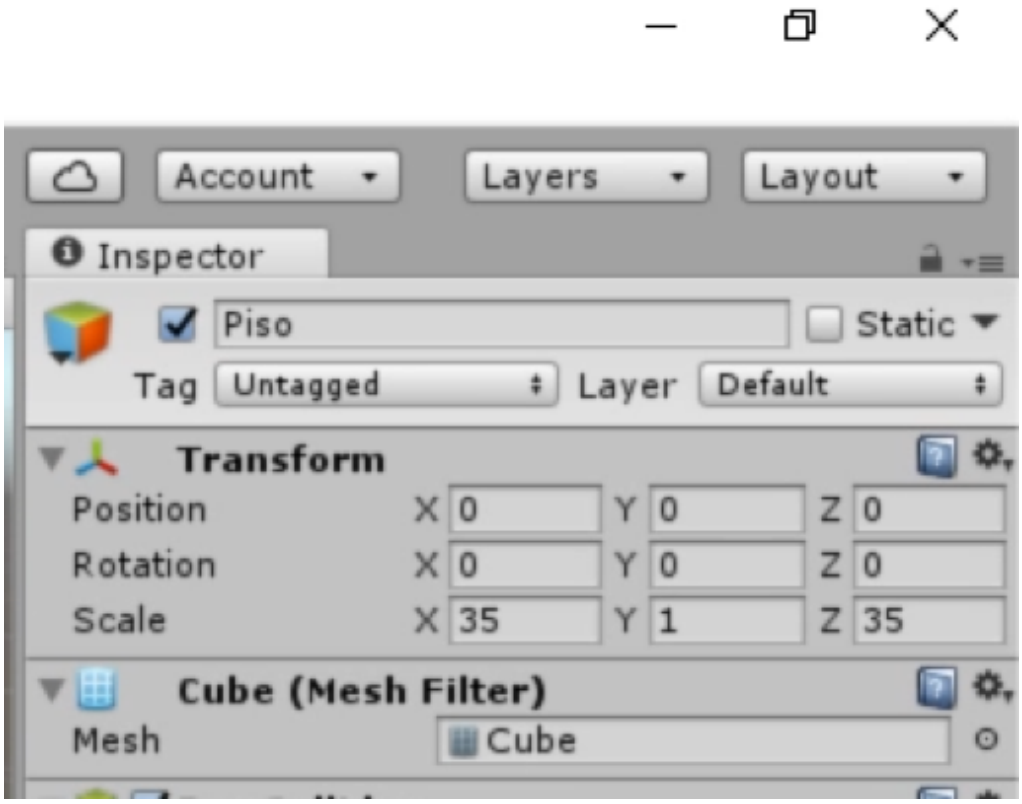
Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Atenção

Às vezes, quando você muda o posicionamento de um **GameObject** dessa maneira, ele sai da sua área de visão. Isso não é exatamente um problema, já que você pode navegar na cena segurando o botão direito do mouse e utilizando as teclas W, S, A, D para navegar e chegar até ele novamente. Uma forma de atalho seria localizar o **GameObject** (Piso no nosso caso) no hierarchy e acionar um duplo clique nele. Isso fará com que a visão da cena centralize e se aproxime do **GameObject** em questão.

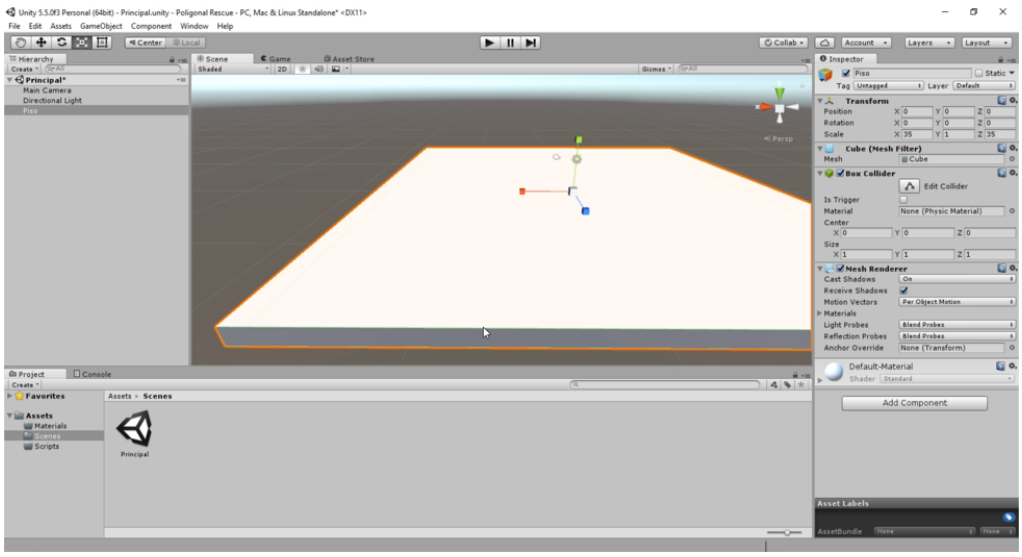
A escala padrão do cubo que renomeamos para "Piso" é de 1,1,1. Para criar o nosso piso modifique esses valores para 35, 1, 35, como demonstra a Figura 8. Isso fará com que ele fique maior e com um aspecto achatado, conforme a Figura 9.

Figura 08 - Modificando as dimensões do Piso.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Figura 09 - Imagem do piso com suas novas dimensões na cena.

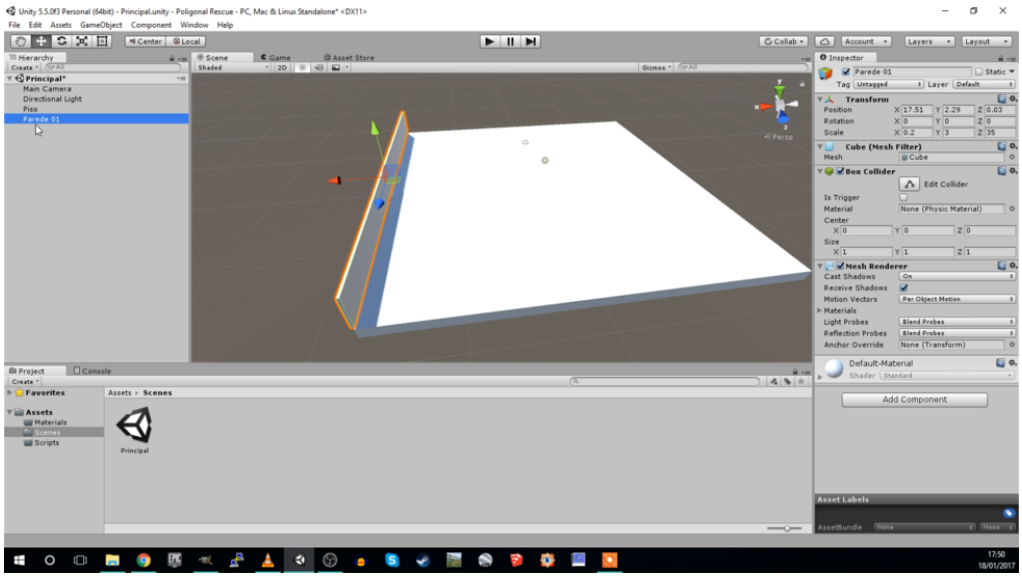


Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Criando as Paredes do Labirinto

Utilizaremos o **mesmo objeto base (Cubo)** para criar as paredes do labirinto, porém com dimensões diferentes. Adicione um novo cubo, modifique a sua escala para **1, 3, 35** e depois mova-o para uma das laterais do piso, representando uma parede, veja na Figura 10. Aproveite para renomear esse cubo para **"Parede 01"**.

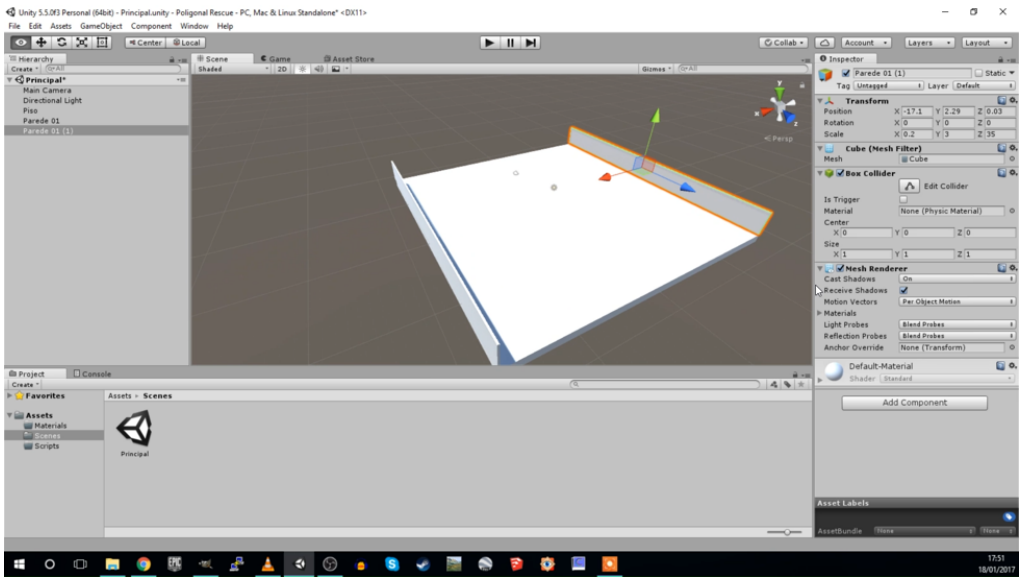
Figura 10 - Criando a primeira parede.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Para criar a parede do lado oposto você pode realizar o procedimento similar ao da primeira parede ou então duplicar a parede atual e mover a sua cópia para o outro lado do labirinto. Para duplicar um **GameObject**, primeiramente selecione-o e depois utilize o comando **Control+D** no teclado. A nova cópia surgirá na cena e já estará selecionada, pronta para ser reposicionada, então você deve movê-la para a outra extremidade, conformea Figura 11.

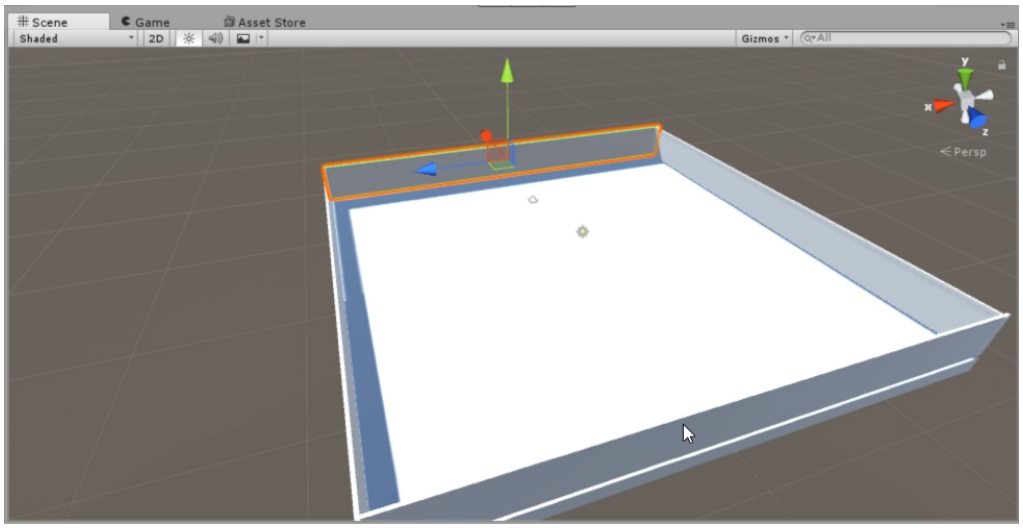
Figura 11 - Segunda parede externa criada a partir da primeira e movida para a outra extremidade do labirinto.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

E aí, até aqui tudo bem? Estamos criando inicialmente as quatro paredes externas do labirinto. As duas primeiras já estão prontas. As outras duas também podem ser criadas a partir de uma duplicata de uma dessas, entretanto você precisa girar a cópia 90 graus no eixo Y para que ela fique na orientação correta, veja na Figura 12.

Figura 12 - Terceira parede criada a partir de uma cópia e sendo girada até 90 graus no eixo Y.

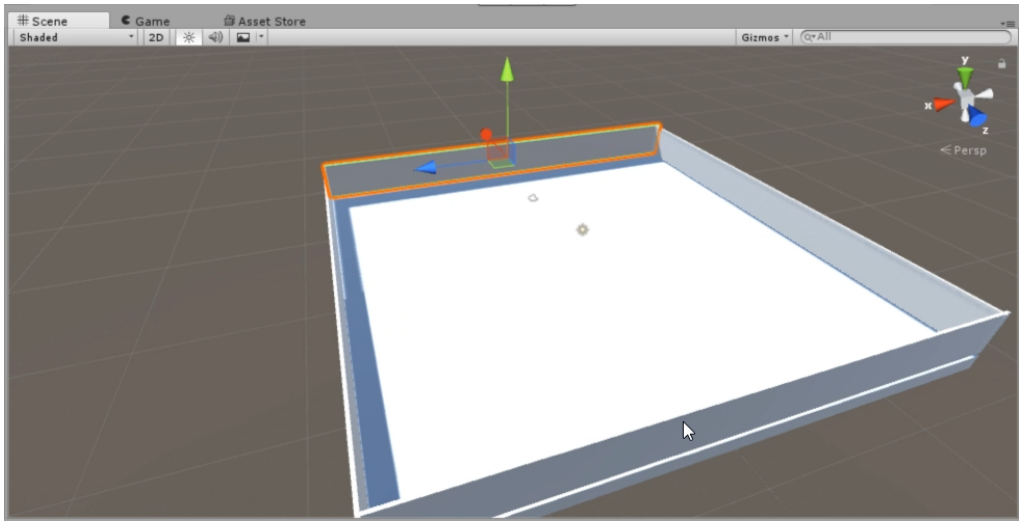


Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Para facilitar a rotação da parede você pode realizar esse procedimento de duas maneiras: uma é digitando o valor 90 na componente Y da propriedade **Rotation** e a outra é rotacionando a parede com a ferramenta de rotação com o mouse, porém segurando a tecla **Control**. Isto fará com que o **GameObject** vá girando de 15 em 15 graus.

Faça o mesmo para a última parede e teremos todas as quatro paredes externas criadas com sucesso, criando um ambiente completamente fechado inicialmente, como demonstra a Figura 13.

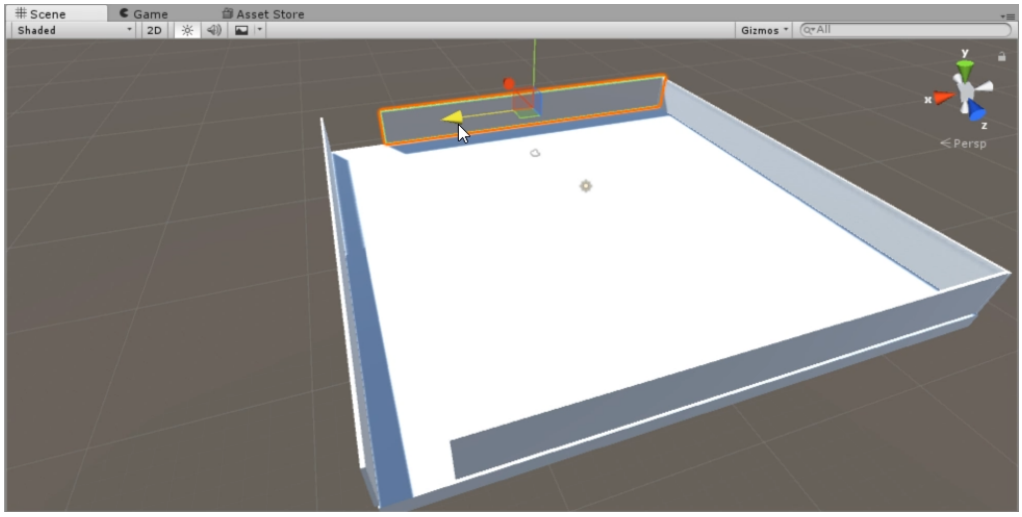
Figura 13 - Quatro paredes externas e piso do labirinto criados.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Precisamos agora criar a entrada e a saída do labirinto. Para isso modifique a escala de duas paredes opostas e as reposicione para que se crie um espaço onde o personagem poderá passar. Veja como ficou na Figura 14.

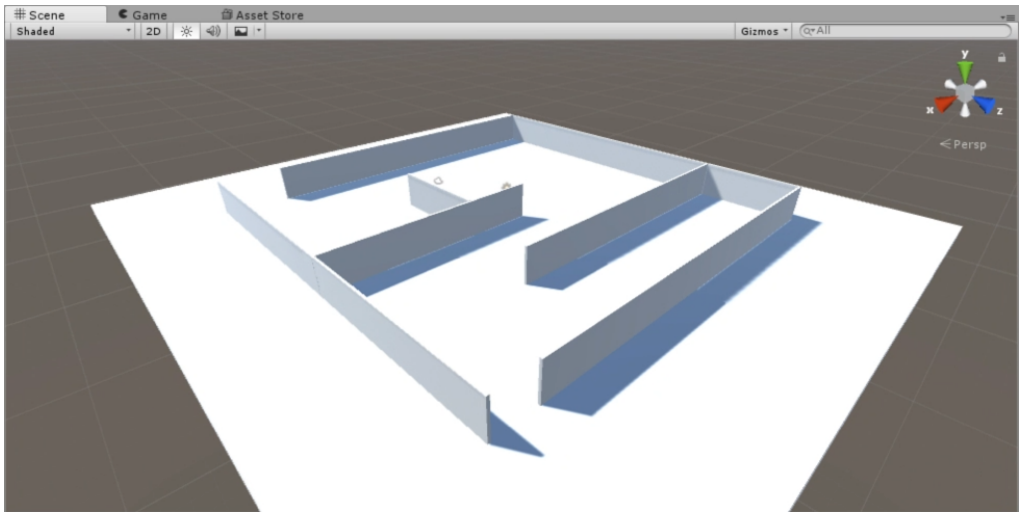
Figura 14 - Ajustes da escala no eixo X de duas paredes externas opostas para criar a entrada e a saída do labirinto.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Com as paredes externas criadas, faremos agora as paredes internas para que a nossa cena fique mais parecida com um labirinto de verdade. Utilize as mesmas técnicas de criação das paredes externas para adicionar as paredes internas do modo que achar melhor, porém é necessário existir pelo menos um caminho da entrada até a saída. Na Figura 15 tem um simples exemplo de como um labirinto pode ser criado. Aumentar um pouco a largura e a profundidade (eixos X e Z) do piso para que este fique um pouco mais amplo e o personagem não inicie no seu entorno é um ajuste que também pode ser realizado.

Figura 15 - Algumas paredes internas adicionadas e piso ampliado.

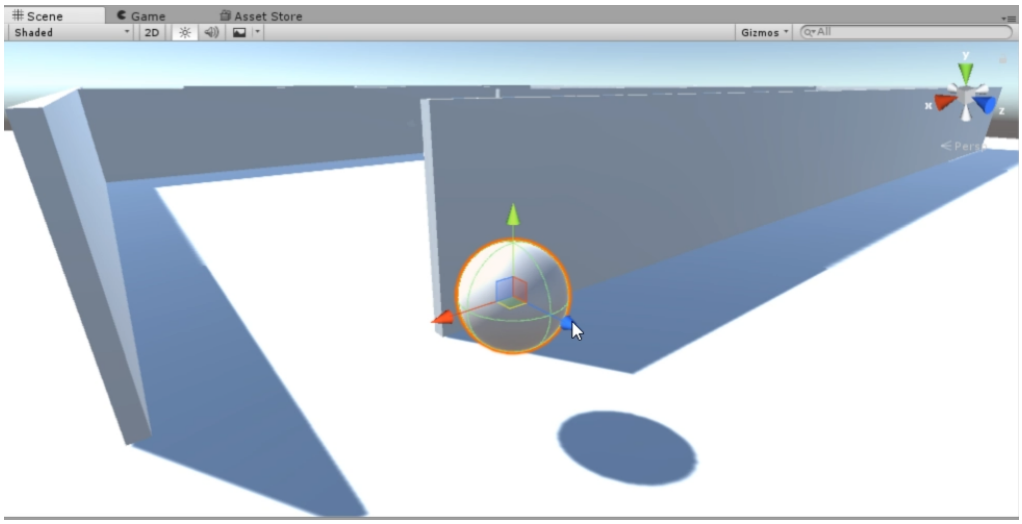


Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Criação do Personagem

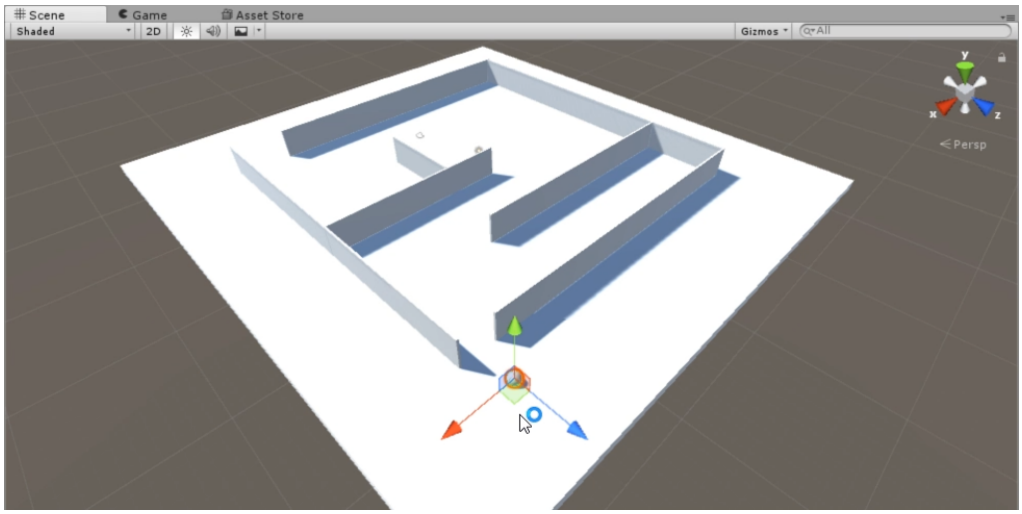
Como já desenvolvemos o ambiente para jogo Polygonal Rescue. Criaremos agora o personagem adicionando uma simples esfera na cena (GameObject->3D Object->Sphere) e movendo-a para a proximidade da entrada do labirinto, veja nas Figuras 16 e 17.

Figura 16 - Personagem (Sphere) adicionado na entrada do labirinto.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Figura 17 - Visão de cima da base da nossa cena criada com o personagem selecionado.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Renomeie a esfera que foi criada para **"Personagem"** e pronto! Temos a base da nossa primeira fase do jogo criada. Porém ainda não temos um jogo. Como? Calma, precisamos melhorar vários aspectos e adicionar outros recursos também. Um dos aspectos a ser melhorado é o fato de todos os elementos da cena apresentarem a mesma cor. Isso é muito confuso e dificulta que o jogador se localize. Exploraremos agora outro recurso para auxiliar na mudança de cor dos objetos, chamado **Materials.**

Fonte: Autoria própria.

Materials

Materials são elementos do Unity que guardam informações de cor, textura, reflexão, etc. Um Material é criado como um Asset no Unity e pode ser adicionado a vários **GameObjects**, como Cubos, Esferas, Cilindros, Modelos 3D customizados, etc.

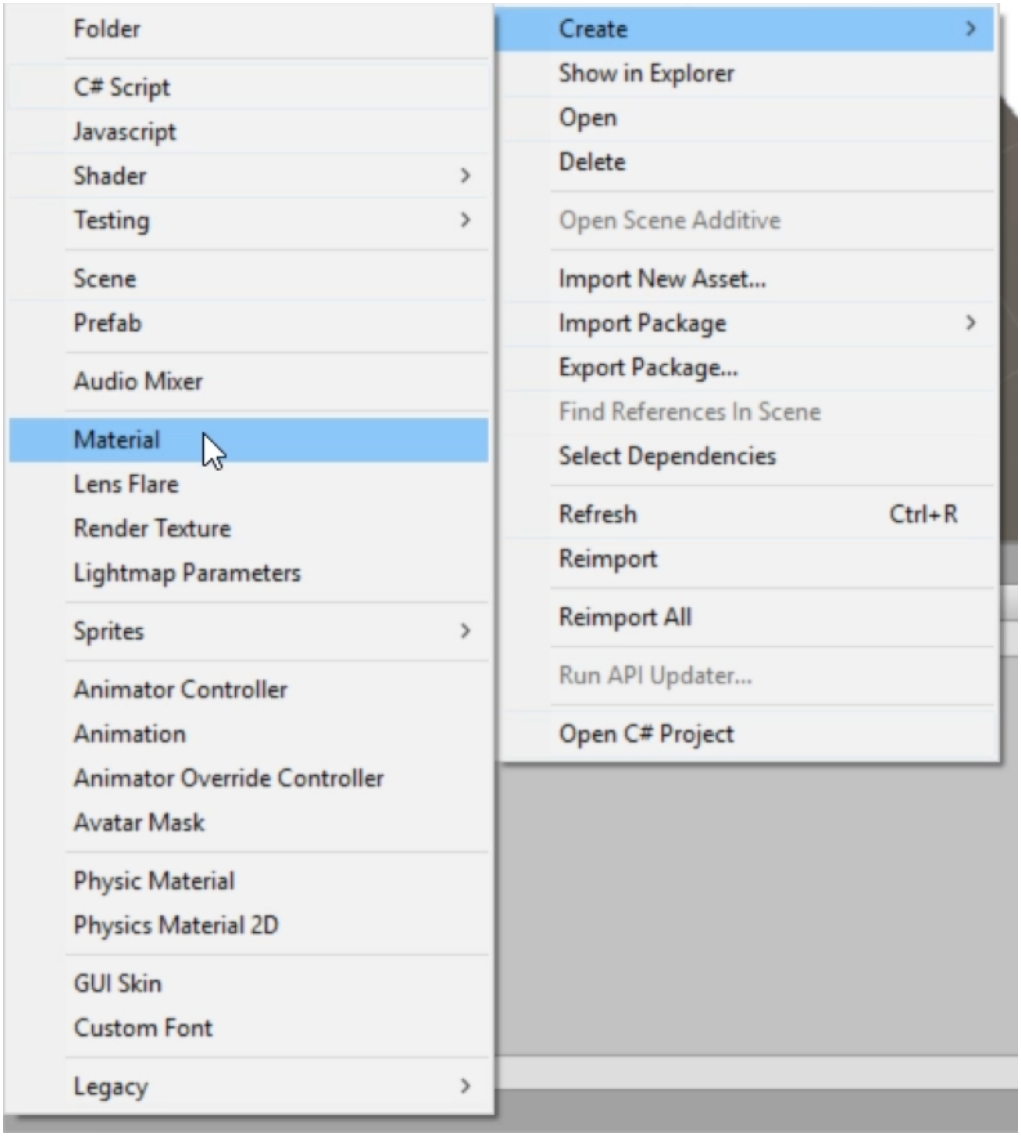
No Unity, um **GameObject**, por exemplo um Cubo, não tem simplesmente uma cor, na verdade ele tem um Material associado que apresenta as propriedades de aparência da superfície do objeto ao qual ele está ligado. A vantagem dessa abordagem é que um Material pode ser aplicado em diversos objetos ao mesmo tempo e qualquer mudança nesse Material se reflete em todos os objetos associados a ele.

Nas próximas aulas estudaremos sobre os **Materials** de forma mais avançada, entretanto para o nosso primeiro jogo utilizaremos a forma mais simples de **Material**, com apenas uma cor específica.

Precisaremos de 3 **Materials:** um é para personagem; outro é para as paredes e um último é para o piso.

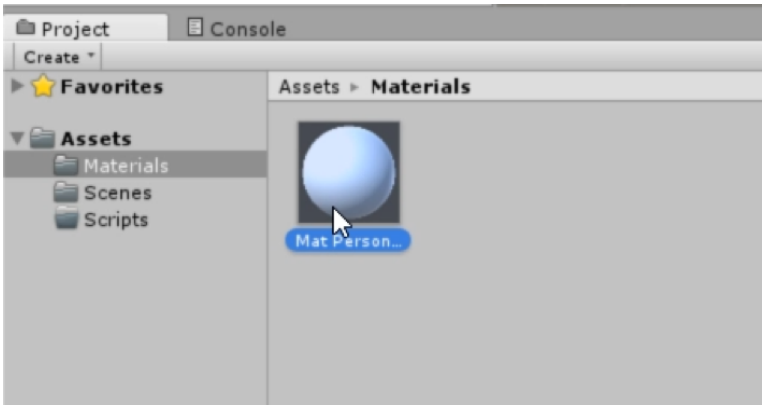
Para criar um **Material** clique na pasta **Materials** que criamos no início do projeto, depois clique com o botão direito do mouse no seu conteúdo vazio e escolha a opção Create->Material (Figura 18). Renomeie o **Material** para **“Mat Personagem”**. Vamos utilizar esse padrão de nomenclatura com o **“Mat”** como prefixo dos **Materials**, mas isso não é algo obrigatório. A Figura 20 mostra como o **Asset Material** criado ficará no nosso projeto.

Figura 18 - Criando o Material para o personagem.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

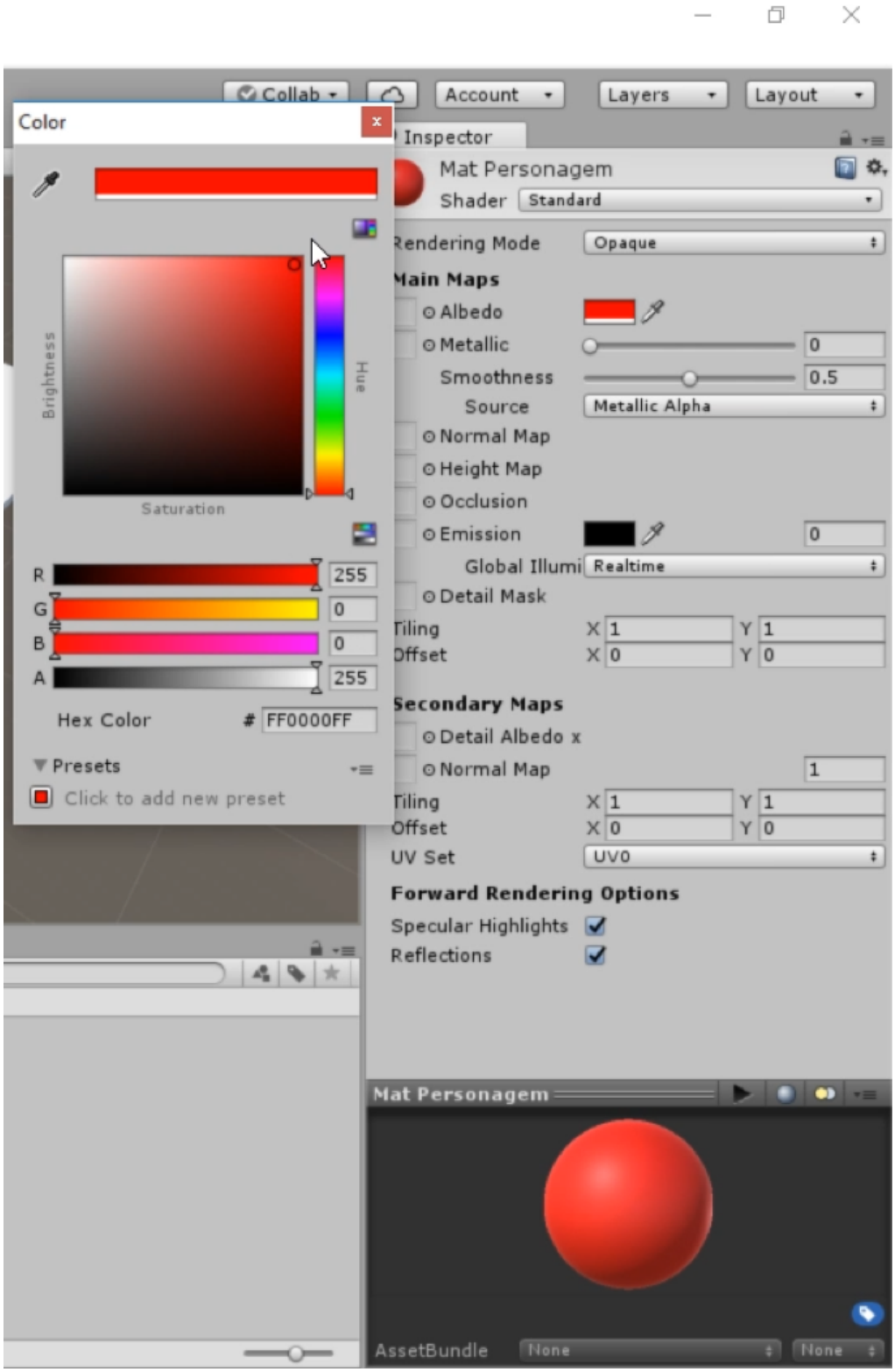
Figura 19 - Material do personagem devidamente criado e renomeado.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Com esse **Material** selecionado, observe no **Inspector** que a cor padrão dele é branco. Clique nessa cor e escolha uma cor vermelha na janela de seleção de cor que irá aparecer (Figura 20).

Figura 20 - Selecionando uma nova cor base para o Material do personagem.

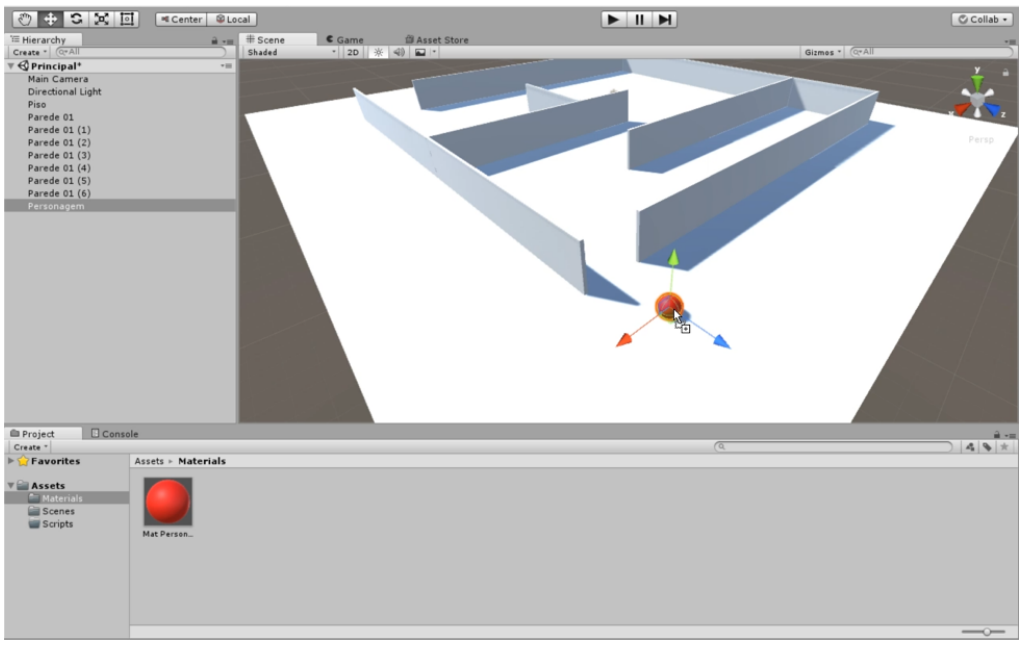


Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Repare que os **Materials** têm várias propriedades as quais podem ser ajustadas. Você pode livremente experimentar mudar algumas delas, como o **Metallic** e **Smoothness**, e conferir o que ocorre. Ainda exploraremos melhor algumas dessas propriedades, não se preocupe!

O Material do personagem está pronto. No entanto a única coisa que fizemos foi criar um novo Asset do tipo Material dentro de uma pasta, ou seja, não informamos ao Unity que ele faz parte do personagem criado na cena. Para associar o Material ao personagem, arraste-o e solte-o em cima da esfera que representa nosso personagem (Figura 21). Repare que enquanto você arrasta o material para o personagem e o mouse passa sobre os outros GameObjects da cena, o Unity aplica temporariamente o Material ao GameObject com a intenção de obter uma visualização de como ele ficaria nele.

Figura 21 - Aplicando o Material ao personagem.



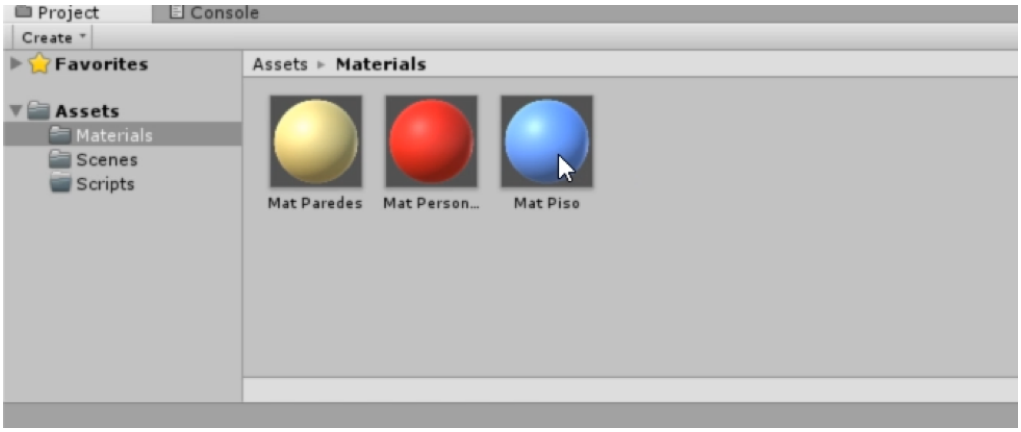
Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Pronto, agora nosso personagem herói é uma linda esfera vermelha! :)

Caso deseje mudar essa cor, entãoclique no Material “Mat Personagem” para realizar a mudança que será automaticamente aplicada ao personagem.

Agora criaremos mais dois Materials, um para o piso e outro para as paredes. Crie esses dois materiais da mesma forma que criou o do personagem (porém escolha outra cor para cada um), renomeando-os para “Mat Piso” e “Mat Paredes”, assim como na Figura 22.

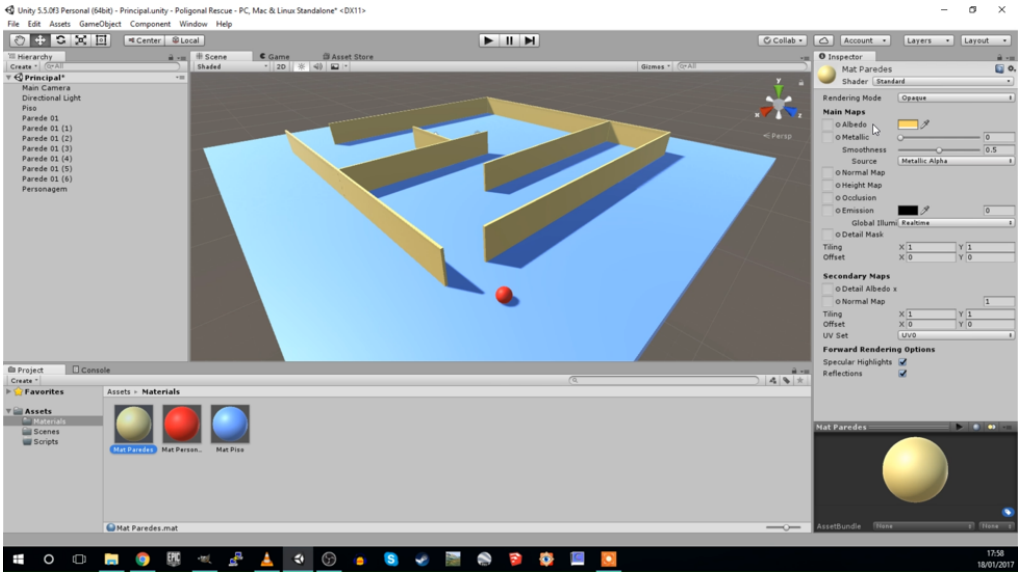
Figura 22 - Materials criados para o personagem, piso e paredes.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Agora, com os Materials criados, arraste o Mat Piso para o piso na sua cena e logo após arraste Mat Paredes para cada parede da cena. Você deve acabar com uma cena similar à vista na Figura 23.

Figura 23 - Materials aplicados em toda a cena.



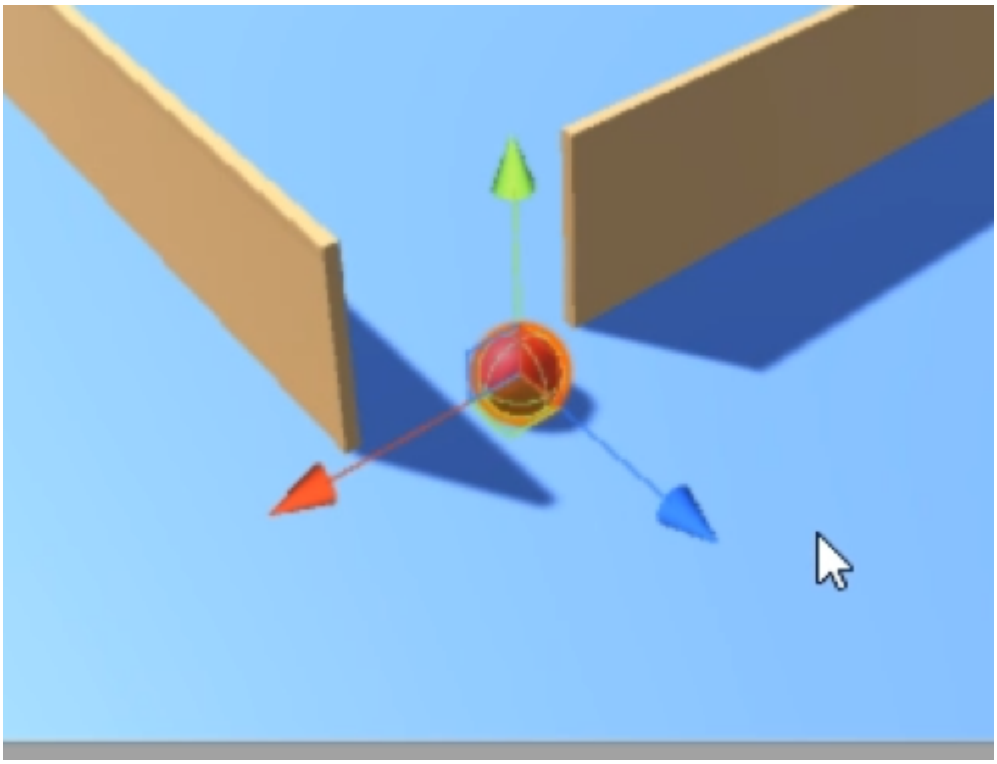
Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Posicionamento da Câmera

Antes de darmos continuidade aos nossos estudos, precisamos ajustar o posicionamento da câmera. Se você executar o jogo agora verá que câmera não está em uma posição adequada. Na primeira parte da criação do jogo polygonal Rescue a câmera será fixa e terá uma visão bem alta e um pouco inclinada para a frente, permitindo a visualização de toda a cena de uma só vez.

Por padrão no Unity, o eixo X representa os movimentos laterais, o eixo Y o movimento para cima e para baixo e o eixo Z para trás e para a frente. Repare que no nosso jogo o personagem está posicionado “de costas” para a cena. Como podemos saber disso já que ele é uma esfera? Bem, na verdade basta olhar para o gizmo de posicionamento e notar que o eixo Z (azul) não está orientado para a entrada do labirinto. Veja Figura 24.

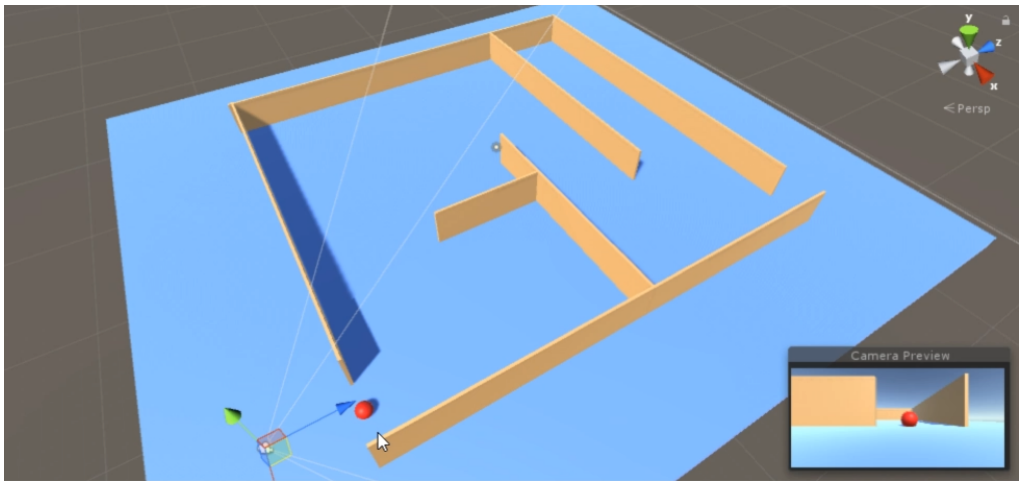
Figura 24 - Personagem com a “Frente” apontando para o sentido oposto da entrada do labirinto.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Você poderia simplesmente resolver esse problema girando o personagem, mas nesse caso inverteremos as entradas e as saídas do labirinto. Mova o personagem para a outra abertura do labirinto e vamos assumir que ela agora é a nova entrada. Isso vai facilitar na programação do nosso script, pois tanto a câmera como o personagem ficarão orientados no mesmo sentido, além disso não ficaria muito interessante o personagem iniciar o jogo na parte mais distante do mapa e vir em direção à câmera para manter essa restrição. A Figura 25 mostra o personagem já posicionado na outra abertura do labirinto (agora chamada de entrada).

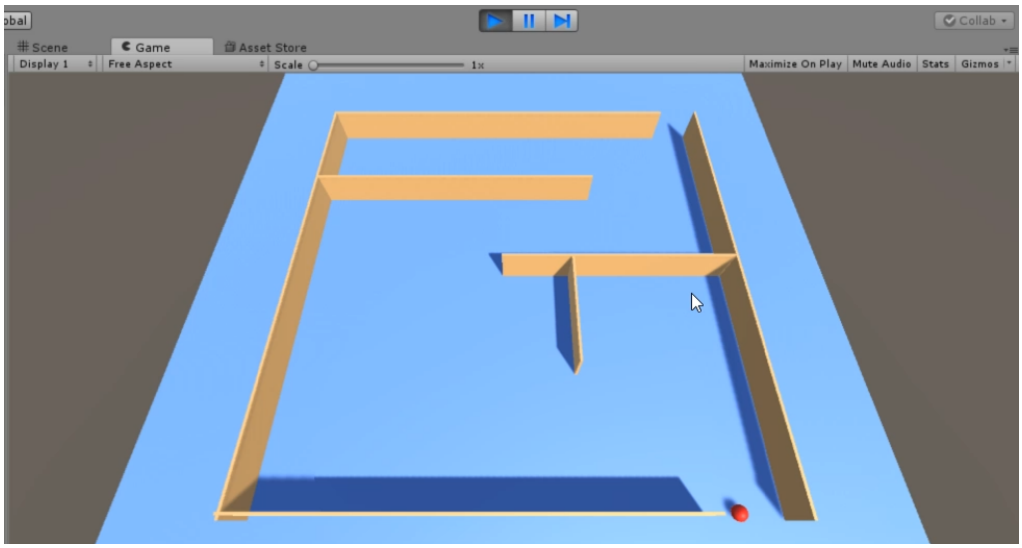
Figura 25 - Reposicionamento do personagem na outra abertura do labirinto para manter a mesma orientação da câmera.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: . Acesso em: 23 de fevereiro de 2017.

Agora ajustaremos a câmera. No Hierarchy, clique na Main Camera e verá uma pequena janela de pré-visualização da câmera no canto inferior direito. Ajuste a posição da câmera de forma que a sua visualização fique como na Figura 26. Você vai precisar mover aproximadamente 30 unidades para cima no eixo Y e rotacionar a câmera aproximadamente 60 graus no eixo X para obter uma visão similar.

Figura 26 - Visualização final da câmera.



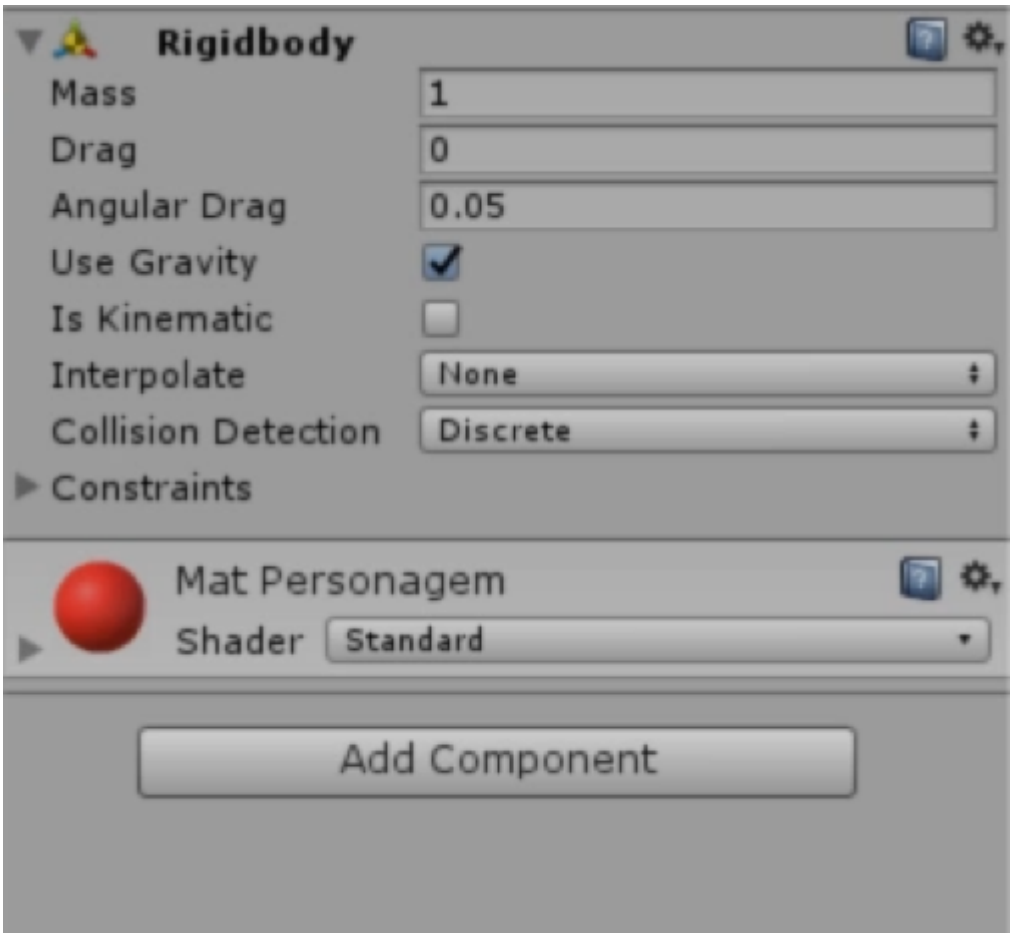
Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Movimento do Personagem com Física

Adicionando o Rigidbody no Personagem

Com nossa cena tomando forma, vamos agora adicionar movimento baseado em física no nosso personagem. Para isso, adicionaremos no nosso personagem um componente chamado Rigidbody. Clique no personagem, vá ao inspector, clique em Add Component->Physics->Rigidbody. Veja na Figura 27 o Rigidbody já adicionado ao personagem.

Figura 27 - Rigidbody adicionado no personagem.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

O componente Ridigbody simula reações físicas no GameObject que ele está associado, como gravidade, atrito, velocidade, aplicação de forças, etc.

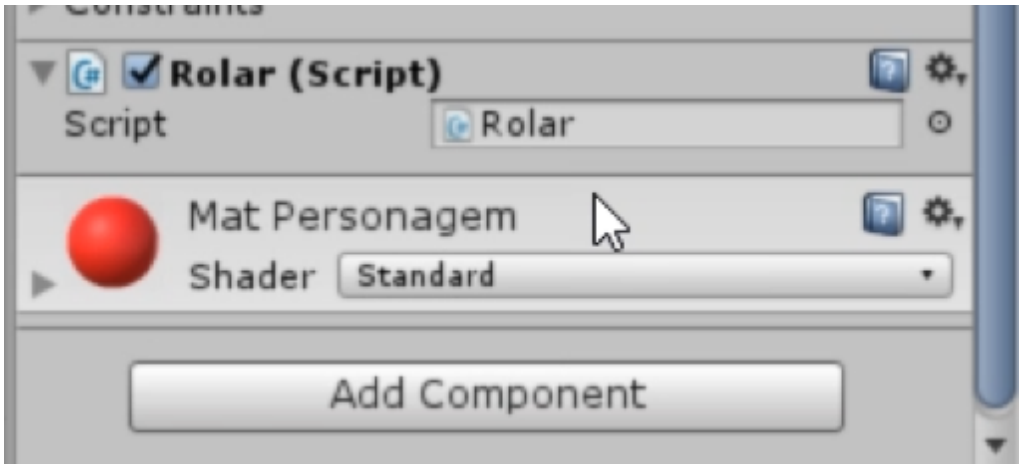
Utilizaremos o Rigidbody, pois nesta aula vamos implementar um tipo de movimento por meio do qual quando o jogador decidir ir para a direita mantendo a tecla pressionada uma força é adiciona ao personagem naquela direção, e o Rigidbody reage adequadamente a essa força fazendo a esfera girar naquela direção. Diferentemente do Translate, a adição de forças dá um aspecto mais realista ao movimento, pelo menos nesse tipo de personagem o qual estamos construindo.

Criação do Script de Movimento

Como nosso personagem é na verdade uma esfera, vamos criar um script chamado “Rolar” para implementar a funcionalidade de movimento baseado em física.

Clique no personagem, vá no inspector e clique em Add Component->New Script. Digite o nome “Rolar” para o novo script o qual será criado e clique em “Create and Add”. Isso criará um script chamado Rolar.cs e já o adicionará ao personagem (Figura 28). Vá no painel “Project” e veja que o script ficou na raíz dos Assets. Mova-o para a pasta “Scripts” criada por você anteriormente para manter o seu projeto mais organizado.

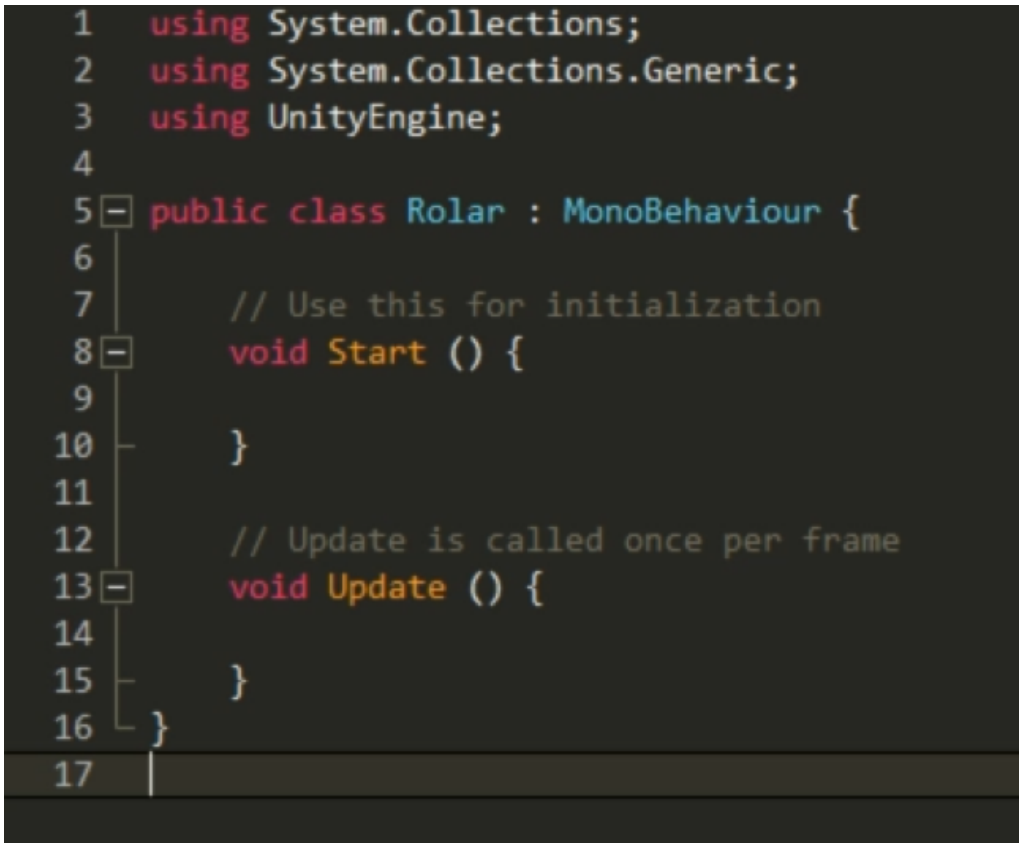
Figura 28 - Script “Rolar.cs” já adicionado no personagem.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Dê um duplo clique no script Rolar.cs para o abrir no MonoDevelop. Ele deve abrir como na Figura 29.

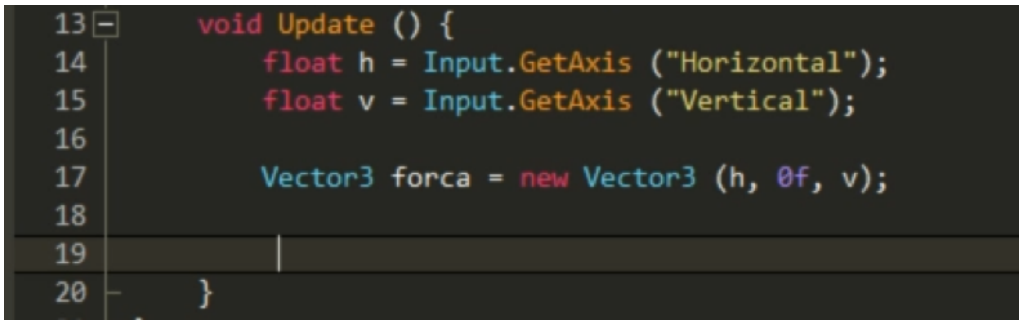
Figura 29 - Script Rolar original.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Similar ao movimento utilizando o Translate visto na aula passada, iniciaremos obtendo os valores dos eixos horizontal e vertical e guardando nas variáveis “h” e “v”. Depois vamos criar uma variável Vector3 chamada “forca”, a qual terá suas componentes criadas utilizando os valores de “h” e “v”, que são os valores dos eixos (a componente Y do vetor de força será zero, pois não queremos movimentos para cima/baixo). Isso fará com que o nosso vetor “forca” represente precisamente para onde o jogador está desejando que o personagem vá, de acordo com os comandos do teclado (teclas W, S, A, D ou setas) ou do joystick. Veja esse código parcial do método Update na Figura 30.

Figura 30 - Criação do vetor de força a ser aplicada ao personagem utilizando os valores dos eixos horizontal e vertical.



Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Não precisaremos indicar que a esfera irá rolar no piso, pois o próprio componente Rigidbody e suas interações físicas com a cena faz esse trabalho da presença de alguma força aplicada.

Então precisamos aplicar essa força ao personagem para que ele se mova. Para isso, devemos executar um método chamado AddForce(), que é na verdade do Rigidbody ao qual o personagem está associado. Lembre-se que tanto o script Rolar como o Rigidbody são ambos componentes associados aos personagens, então precisamos encontrar uma forma do script conseguir acessar outro componente do personagem a ele associado. Por sorte existe um método no Unity chamado GetComponent, ele permite que um script possa obter uma referência a qualquer componente do GameObject associada a ela e salvar essa referência em uma variável para poder usar posteriormente.

Para pegar a referência ao Rigidboby, você precisa digitar somente duas linhas de código. Primeiro crie uma variável na sua classe do tipo Rigidbody com o nome “rigidbody” (em minúsculo). Depois, no método Start() adicione a linha:

rigidbody = GetComponent ();

Pronto. A partir de agora existe uma variável chamada “rigidbody” acessível de qualquer método dessa classe que mantém uma referência ao Rigidbody presente no personagem, assim podemos utilizá-la para aplicar a nossa força agora. Ver Figura 31.

Figura 31 - Criação da referência para o Rigidbody do personagem dentro do script.

Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

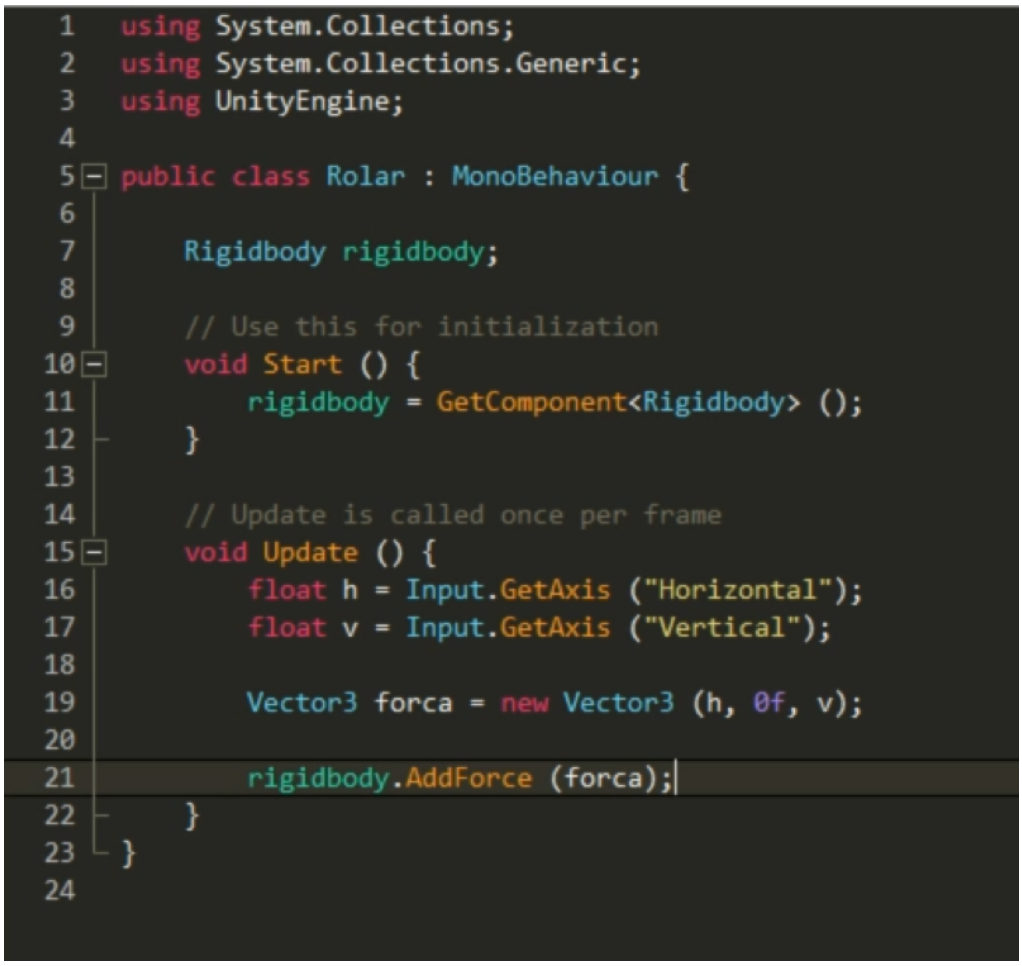
Voltando ao método Update(), agora que temos acesso ao rigidbody do personagem através da nossa variável criada, basta aplicar a força que já calculamos com uma simples linha:

rigidbody.AddForce (forca);

Pronto. Agora pode salvar o script, voltar ao jogo e pressionar Play. Você conseguirá controlar seu personagem com as teclas W, S, A, D do teclado e ele irá se locomover pela cena. O movimento baseado em física é bem mais realista e, portanto, difícil de controlar, pois se a esfera estiver rolando para uma direção com muita velocidade e você der o comando para ela ir para outra direção, a mudança de velocidade não acontecerá bruscamente, e sim, gradualmente, como quem realmente está tentando mover uma esfera bem pesada. Essa jogabilidade realista é um dos desafios de se completar uma fase do jogo em tempo hábil a ainda resgatar os futuros polígonos os quais adicionaremos em outras aulas.

Veja o código atual do script Rolar na Figura 32.

Figura 32 - Código do script Rolar com movimento com Rigidbody.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Rolar : MonoBehaviour {

    Rigidbody rigidbody;

    // Use this for initialization
    void Start () {
        rigidbody = GetComponent<Rigidbody> ();
    }

    // Update is called once per frame
    void Update () {
        float h = Input.GetAxis ("Horizontal");
        float v = Input.GetAxis ("Vertical");

        Vector3 forca = new Vector3 (h, 0f, v);

        rigidbody.AddForce (forca);
    }
}
```

Fonte: Captura de tela do Unity – Game Engine. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de fevereiro de 2017.

Resumo

Nesta aula, definimos e criamos a movimentação básica do jogador, que se refere a uma das principais funcionalidades do nosso jogo Polygonal Rescue. Além disso, aprendemos a criar Materials básicos de cor sólida, bem como aplicá-los em objetos na cena, dando um aspecto único a cada um deles.

Também utilizamos o sistema de Rigidbody do Unity para que o movimento do personagem pudesse realizar as interações físicas com os demais objetos da cena, criando um efeito realista de controle.

Leitura complementar

O sistema de física do Unity é muito completo e só teremos tempo de estudar uma parte dele nas nossas aulas. Como leitura complementar estude a documentação do Unity referente a esse sistema e ao uso de Rigidbody.

- Aprenda com o Unity (oficial). Disponível em: <https://unity3d.com/pt/learn>

- APhysics tutorials. Disponível em: <https://unity3d.com/pt/learn/tutorials/topics/physics>

Autoavaliação

1. Tente modificar o script para que a velocidade da esfera fique maior. Como você acha que isso deve ser feito?

2. Se você entendeu bem a diferença de movimento utilizando o Translate e o AddForce, tente imaginar que na nossa cena existisse uma rampa e a esfera estivesse em direção a essa rampa. O que aconteceria se o movimento fosse implementado utilizando Translate?

3. No mesma caso acima, o que aconteceria se fosse o movimento implementado com AddForce, como fizemos nesta aula?

Referências

UNITY TECHNOLOGIES. 2016 (C). Unity 3D Online Tutorials [online]. Disponível em: <https://unity3d.com/pt/learn/tutorials> [Acessado em 16 de novembro de 2016].

PASSOS, E. B., SILVA, J., NASCIMENTO, G. T., KOZOVITS, L. CLUA, E. W. G. 2008. **Fast and safe prototyping of game objects with dependency injection. Anais do Simpósio Brasileiro de Games e Entretenimento Digital.** São Leopoldo, RS. 2008

STOY, C. 2006. **Game object component system. In Game Programming Gems 6, Charles River Media, M. Dickheiser,** Ed., Páginas 393 a 403.

MARQUES, Paulo; PEDROSO, Hernâni **C# 2.0 . Lisboa: FCA, 2005.**ISBN 978-972-722 208-8

UNITY TECHNOLOGIES. 2016 (C). **Unity 3D Manual [online].** Disponível em: <https://docs.unity3d.com/Manual/index.html> [Acessado em 16 de novembro de 2016].