Introdução ao Unity

Jay Clei Garcia dos Santos

Unity Technologies

Resumo

Este tutorial tem como objetivo demonstrar a facilidade criação de um nível de um jogo de primeira pessoa (First Person Shooter) desde o zero.

Introdução

A Unity é uma ferramenta de desenvolvimento de jogos 2D / 3D multiplataforma que tem como principais características a facilidade de uso, rápida prototipagem e integração com ferramentas externas como Maya, 3D Studio, Photoshop, Blender, entre outras.

O objetivo da empresa é democratizer o desenvolvimento de jogos. Sua facilidade de uso e preço acessível fizeram com que a Unity atingisse mais de 50% de penetração no mercado de desenvolvimento mobile mundial.

Hoje a Unity conta com 1,5 milhões de usuários registrados no mundo, desde desenvolvedores amadores fazendo um jogo em seu tempo livre, até grandes empresas como Electronic Arts, BigPoint e Nintendo.

Conceitos básicos

OBS.: Todos os materiais e conteúdos utilizados neste tutorial podem ser baixado no site.

OBS. 2: Na parte de animação do robô ("8. Criando um inimigo") é utilizado o sistema de animação Mecanim, presente na nova versão 4.0 do Unity. Todo o resto do tutorial pode ser feito na versão 3.x

1. Criando o Projeto

Caso você ainda não tenha o Unity instalado em sua máquina, você pode baixá-lo em http://unity3d.com/unity/download/

Ao iniciar o Unity clique em File -> New Project defina onde seu projeto vai ser salvo e na janela "Import the following packages" selecione Character

Controller, Skyboxes e Water (Pro Only) (Fig. 1) e clique em "Create Project".

OBS.: O package Water (Pro Only) está disponível apenas na versão Pro do Unity. Ao baixar do site você tem 30 dias de trial da versão Pro. Caso seu trial tenha expirado, utilize a Water (Basic).

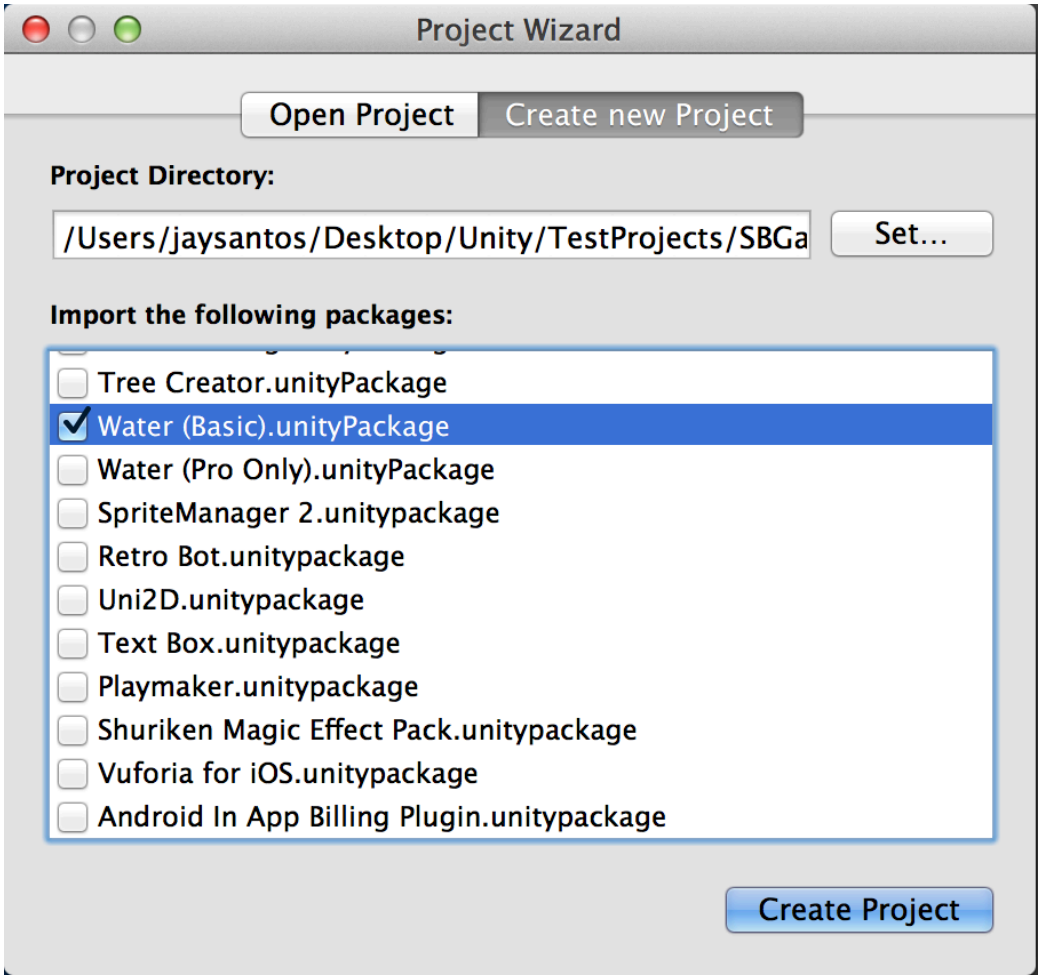

Fig. 1: Janela "Project Wizard"

2. Criando o personagem de primeira pessoa

Ao terminar a criação do projeto, a tela abaixo é exibida (Fig. 2)

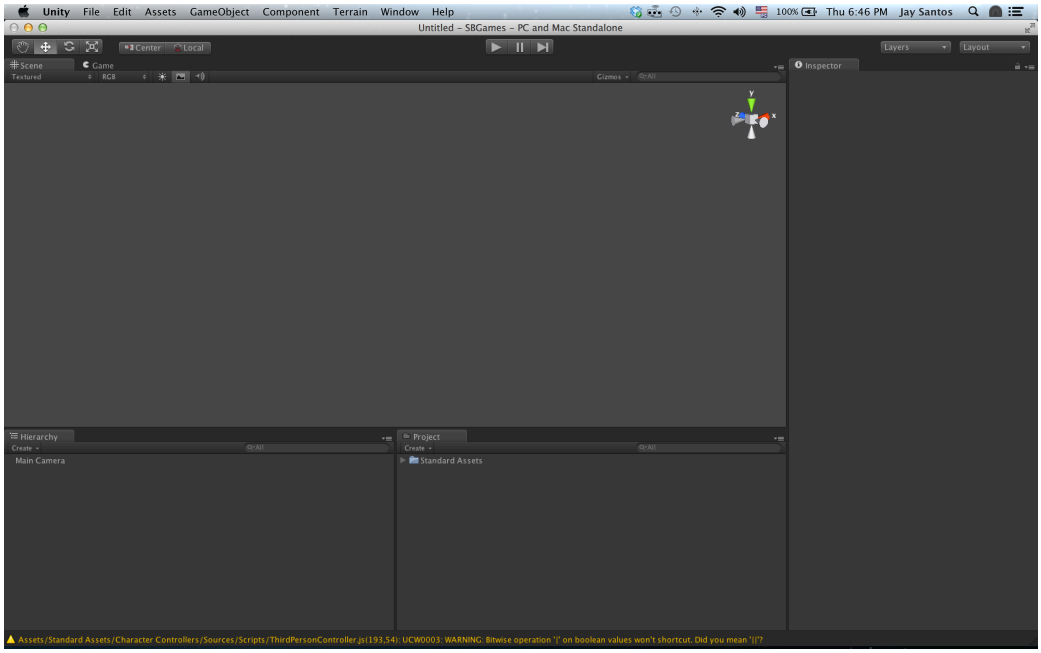

Fig. 2: Tela inicial do projeto

Clique na aba "Game", arraste ela um pouco para baixo e para a direita para separarmos ela da janela "Scene" (Fig. 3)

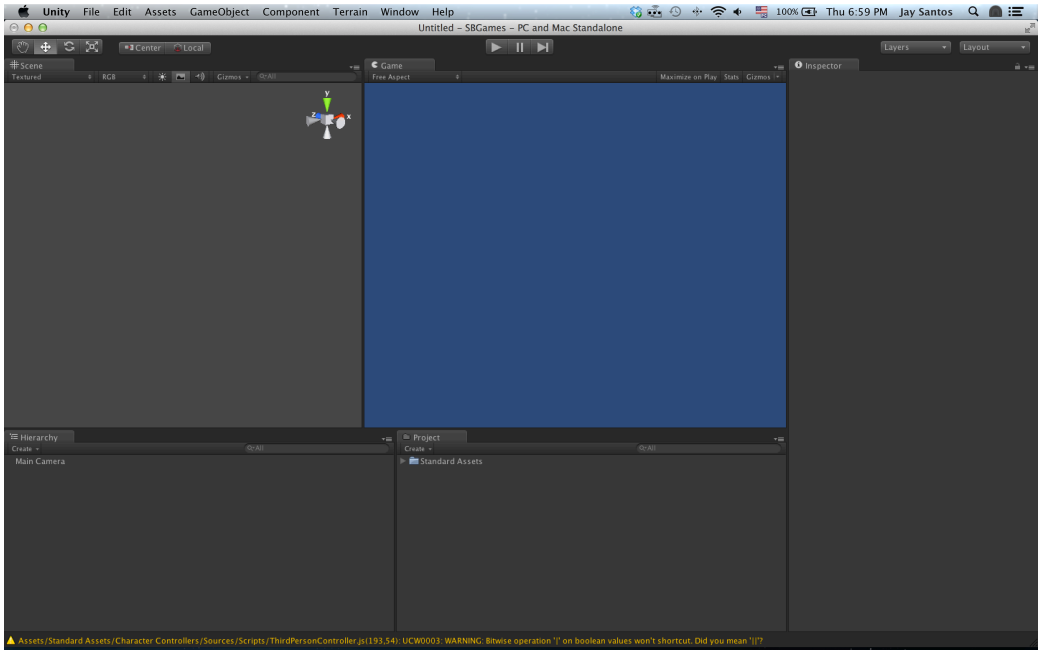

Fig. 3: Tela inicial com a janela "Game" separada

Agora vamos criar o controle de primeira pessoa. Vamos iniciar criando um plano, que vai servir de chão. Clique em "Game Object -> Create Other -> Plane". Apos acrescentar o plano, clique na tela scene e aumente o zoon (usando o mouse scroll) para deixar o plano mais próximo para edicar. Expanda as pastas "Standard Assets" e "Character Controllers", clique e arraste o "First Person Controller" para a janela "Scene" e coloque-o sobre o plano (Fig. 4).

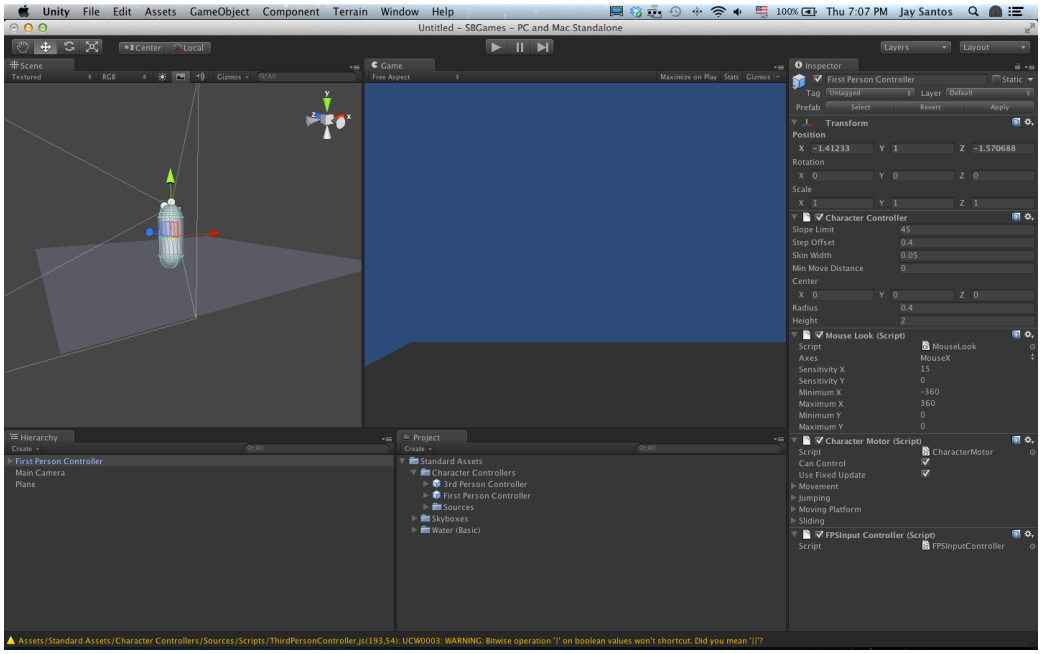

Fig. 4: Plano e controle de Primeira Pessoa acrescentados

Clique o botão "Play" na parte central no topo da tela e clique na janela "Game". Você agora pode controlar o personagem usando W, S, A e D e espaço para pular. Você pode notar também que a gravidade já está atuando na cena: Caminhe com o personagem até além do plano criado e você vai notar que o personagem vai cair. Na janela "Inspector" se você expandir a opção "Movement" dentro de "Character Motor (Script)" você tem todos os parâmetros de movimento do personagem, que podem ser configurados diretamente no editor, sem a necessidade de alterar código-fonte (Fig. 5).

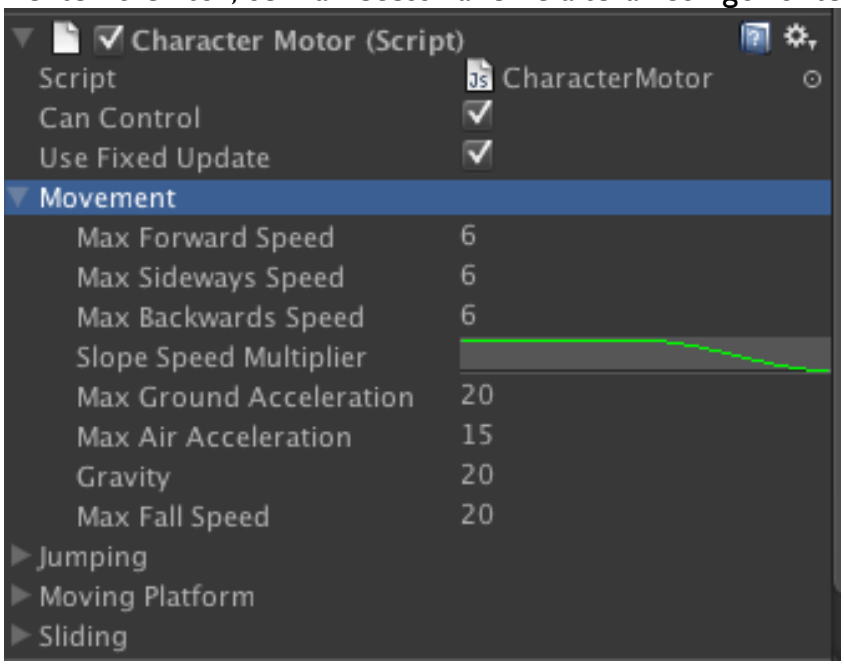

Fig. 5: Parâmetros de configuração de movimento

3. Acrescentando elementos a cena: Luz, objetos, Skybox e texturas

A cena que criamos está escura. Vamos agora adicionar iluminação. Clique em "Game Object -> Create Other -> Directional Light". Você vai notar que o chão da cena já irá ficar muito mais iluminado. Por enquanto vamos manter a luz assim (Fig. 6).

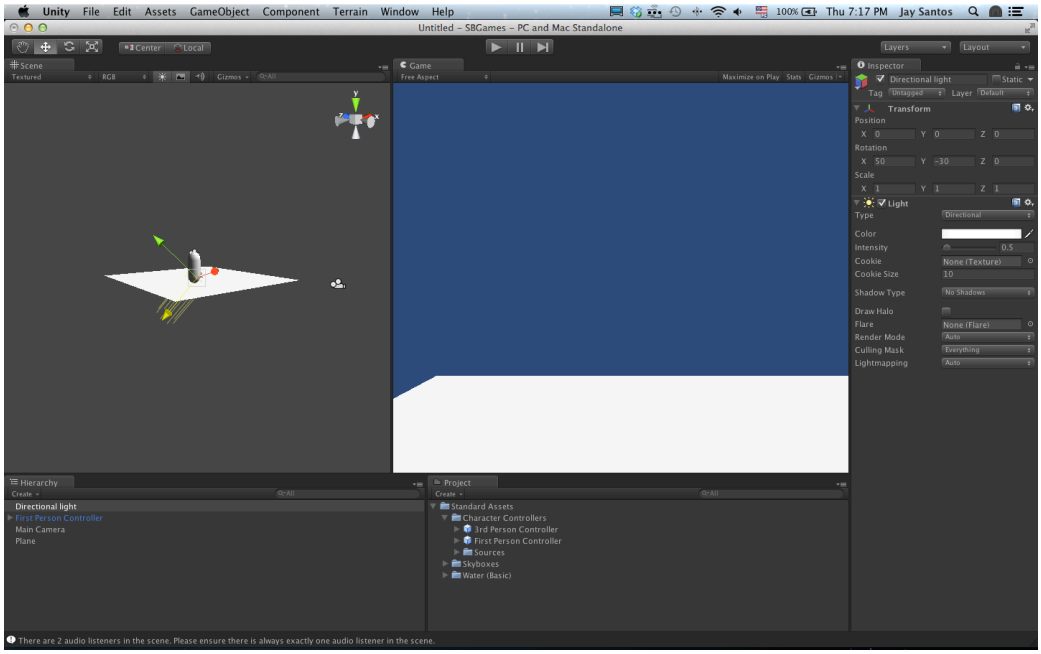

Fig. 6: Mesma cena, agora iluminada

Vamos agora criar duas plataformas. Clique em "Game Object -> Create Other -> Cube". O Cubo vai ser criado aproximadamente na mesma posição que o personagem. Na tela "Scene" clique na seta vermelha e arraste o cubo para o lado. Quando a tela "Scene" está com esta funcionalidade ativada você pode mover objetos através da cena. Clique também na seta verde e arraste um pouco o cubo para cima.

No canto superior esquerdo da tela existem quatro botões com as funcionalidades da tela "Scene", da esquerda para a direita:

- **Primeiro Botão (mão):** Arrasta a cena inteira para facilitar a visualização.
- **Segundo Botão (tooltip: move the selected objects):** Move o objeto dentro da cena.
- **Terceiro Botão (tooltip: rotate the selected objects):** Rotaciona o objeto na cena.
- **Quarto Botão (tooltip: scale the selected objects):** Altera o tamanho do objeto na cena.

Os atalhos para este botões são Q, W, E e R, respectivamente. Após arrastar o cubo vamos alterar suas dimensões, transformando-o em uma plataforma e colocando-o um pouco acima do chão. Uma vez feito isso digite CTRL+D (ou command+D no MacOS) para duplicar a plataforma e coloque a nova plataforma um pouco mais a frente e um pouco mais alto da primeira plataforma (Fig. 7).

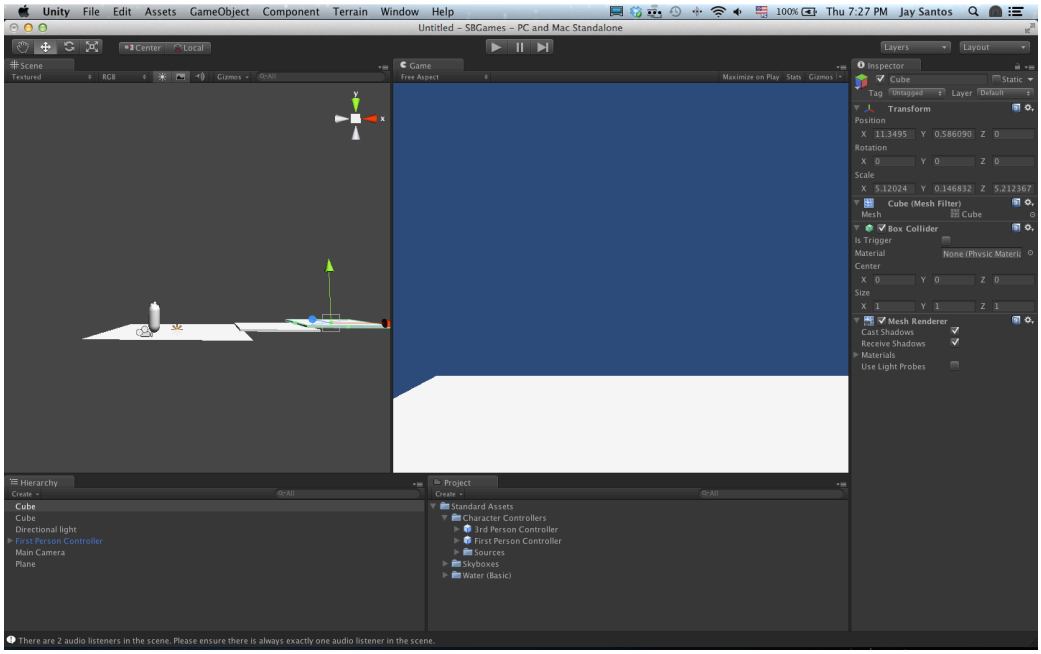

Fig. 7: Plataformas criadas

Clique em "Play" e mova seu personagem para cima das plataformas. Se as plataformas estiverem muito altas e você não conseguir alcança-las pulando, diminua um pouco a distância entre elas. Agora vamos adicionar um Skybox (uma imagem que envelopa a cena e da sensação de profundidade). Clique em "Edit -> Render Settings", uma série de parâmetros serão exibidos na janela "Inspector", clique no botão a direita da opção "Skybox Material" e selecione qualquer uma das opções apresentadas no pop-up. Uma vez selecionado o material feche o pop-up (Fig. 8).

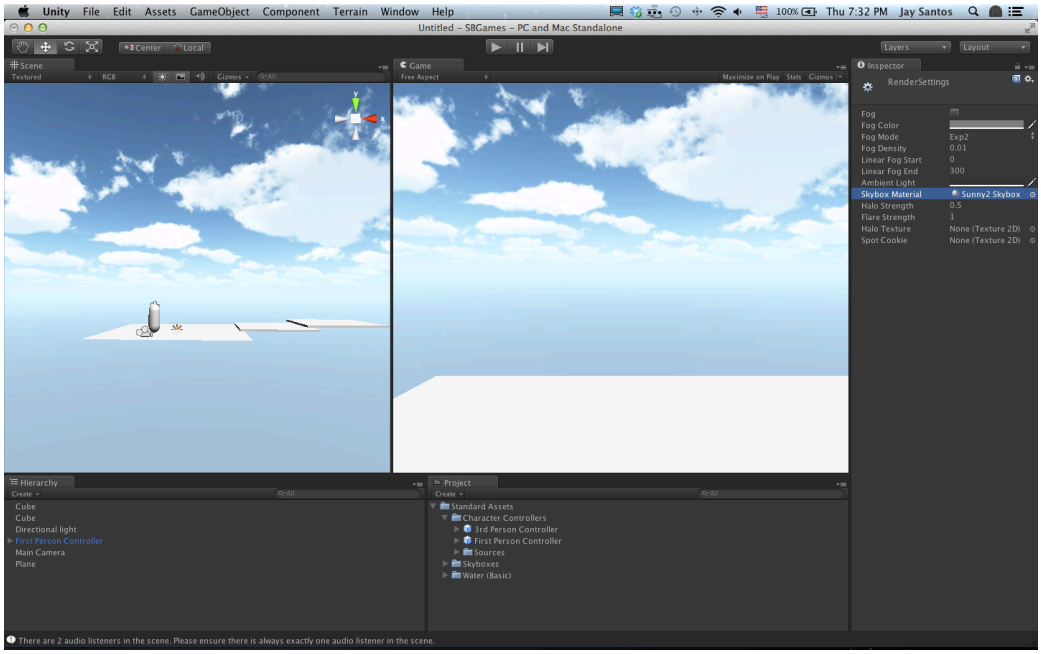

Fig. 8: Cena com o Skybox "Sunny 2 Skybox" aplicado.

Porém os objetos continuam apenas brancos, o próximo passo é adicionar uma textura aos nossos pisos. Clique com o botão direito na janela "Project", selecione "Create -> Folder" e crie uma pasta "Textures". A criação de pastas não é necessária mas é recomendada para a organização do projeto.

Arraste o arquivo "Sand.psd" para dentro da pasta "Textures", uma vez feito isso arraste o "Sand.psd" de dentro da pasta "Textures" para cima do plano (Fig. 9).

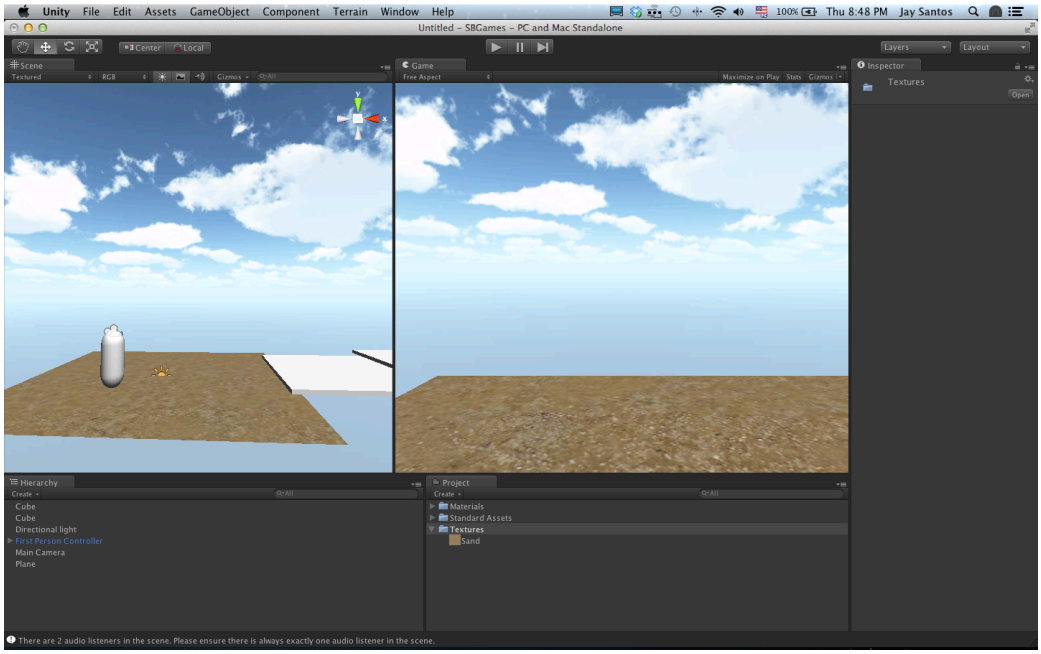

Fig. 9: Plano com a textura de areia aplicada. Repare que as plataformas estão sem textura.

No Unity você também pode trabalhar com texturas procedurais. Primeiro crie um novo cubo (como já feito anteriormente) e altere as dimensões e posição para que ele fique como um muro saindo do chão de areia. Arraste o arquivo "bricks_008.sbsar" para dentro da pasta "Textures" na janela "Project", expanda o object "bricks_008" criado e arraste o "bricks_008" para cima do muro na janela "Scene" (Fig. 10).

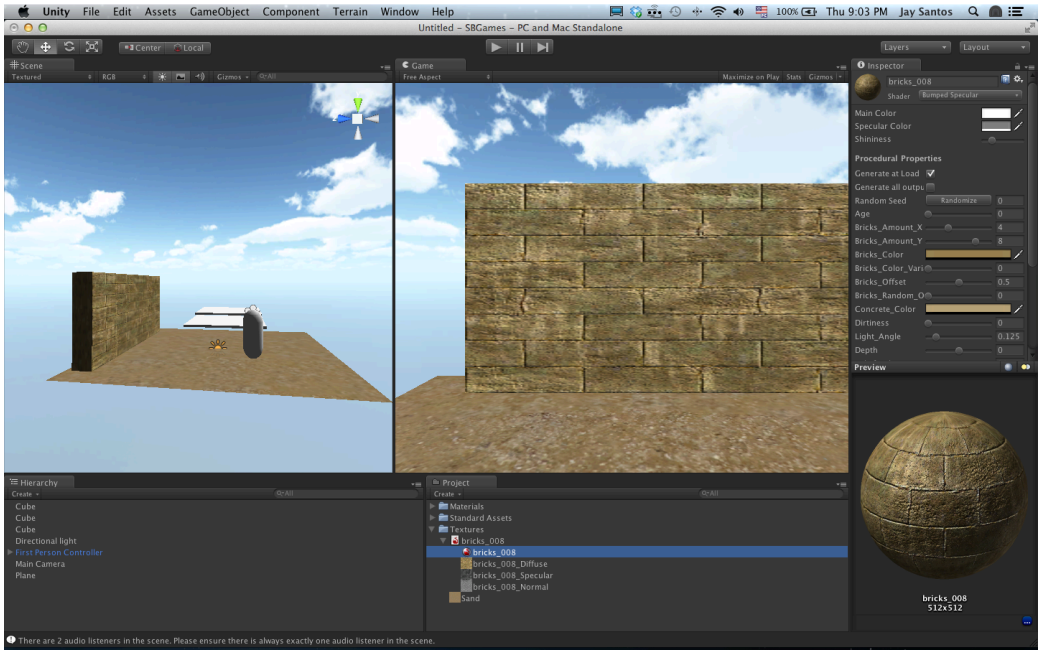

Fig. 10: Muro com a textura procedural aplicada. Você pode ver qual o "bricks_008" que deve ser arrastado indicado na janela "Project".

Com o "bricks_008" selecionado no "Project" você pode ver na janela "Inspector" diversos parâmetros que podem ser alterados para modificar a textura (Fig. 11).

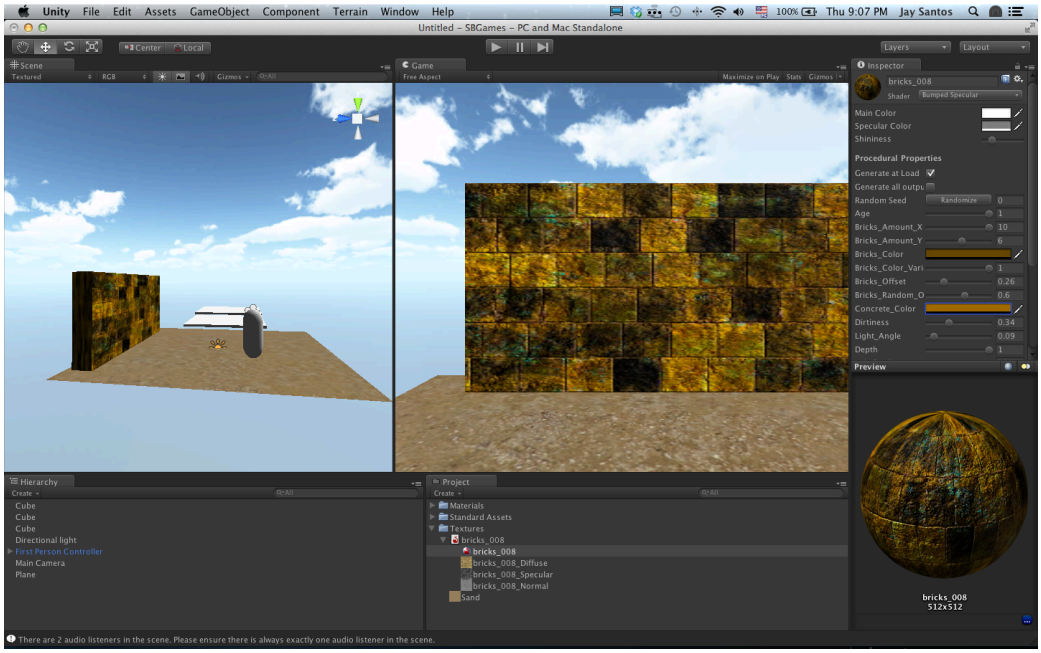

Fig. 11: Muro após alterar alguns dos parâmetros da textura. Compare com o muro da Fig. 10

4. Acrescentando elementos a cena: Modelos 3D e texturas

Vamos substituir nossas plataformas brancas e sem forma por modelos 3D. Na janela "Projects" crie uma nova pasta chamada "3D Models". Arraste os arquivos "platform_LOD0.fbx" e "platform_DIF.tga" para dentro da pasta "3D Models". O arquivo .fbx é o modelo 3D e o arquivo .tga é a textura a ser aplicada a esse modelo. Como o nome dos dois é igual a textura é imediatamente aplicada ao modelo. Se você clicar no objeto "platform_LOD0" dentro da pasta "3D Models" você já vai ver na janela "Inspector" o modelo com a textura, altere o "Scale Factor" dentro da janela Inspector para 0.5 (Fig. 12).

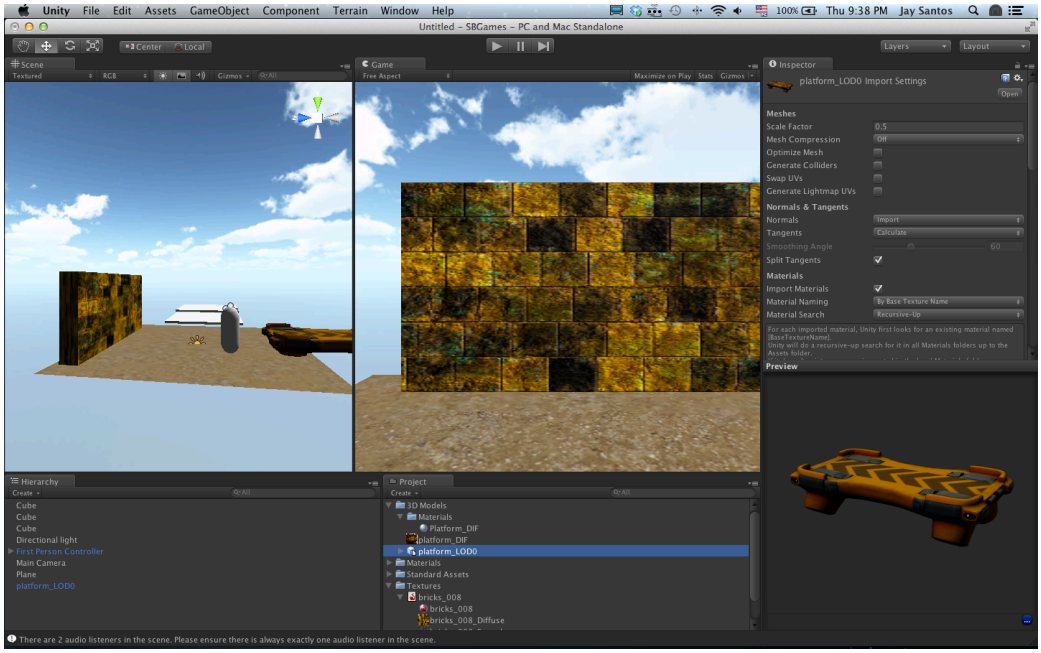

Fig. 12: Modelo 3D da plataforma com a textura já aplicada. Note que o Scale Factor já foi alterado para 0.5

Clique no objeto "platform_LOD0" e arraste-o até a janela "Scene" para acrescentar uma plataforma a cena. Clique em Play e tente pular em cima da plataforma. Você caiu direto por ela! O que aconteceu?

Porque apesar de você já ter o modelo 3D da plataforma ela ainda não tem uma caixa de colisão para impedir sua queda. Vamos criar uma agora.

Selectione o objeto "platform_LOD0" na janela "Hierarchy" e clique em "Components -> Physics -> Box Collider". Você pode ver na janela "Scene" que uma grade verde aparece em volta da plataforma. Clique em Play e agora você pode subir na plataforma (Fig. 13).

Uma explicação rápida sobre a diferença entre as janelas "Project" e "Hierarchy": A janela "Project" mostra todos os elementos disponíveis no seu

projeto, e "Hierarchy" mostra todos os elementos que estão sendo utilizados neste exato momento na cena em que você está trabalhando.

Fig. 13: Personagem em cima da plataforma após adição do Box Collider. Repare na caixa verde em volta da plataforma.

Seria possível também criar um Mesh Collider na plataforma. A diferença é que o Mesh Collider se ajusta exatamente ao modelo 3D da plataforma. A vantagem de usar um Mesh Collider é que você não terá colisões inexistentes, principalmente nas bordas dos objetos. A desvantagem é que o custo de processamento de um Mesh Collider é muito mais alto do que de um Box Collider. Normalmente é recomendado utilizar Box Collider (ou um conjunto de Box Colliders distribuídos pelo objeto) ao invés de Mesh Collider, o resultado é satisfatório e é muito mais barato em termos de processamento.

Vamos agora criar um muro e entender como duplicar e conectar elementos. Arraste os arquivos "EV_Wall.FBX" e "EV_Wall_DIF.tga" para a janela "Project" e clique no objeto EV_Wall criado dentro da pasta 3D Models.

Você vai reparar que o nosso modelo está sem textura. Para aplicar uma textura a um objeto de scroll na janela "Inspector" ate o final. Uma pequena janela quadrada com o texto "None (Texture)" vai aparecer. Arraste o objeto "EV_Wall_DIF.tga" de dentro da pasta 3D Models para esta janela e a textura vai ser aplicada (Fig. 14).

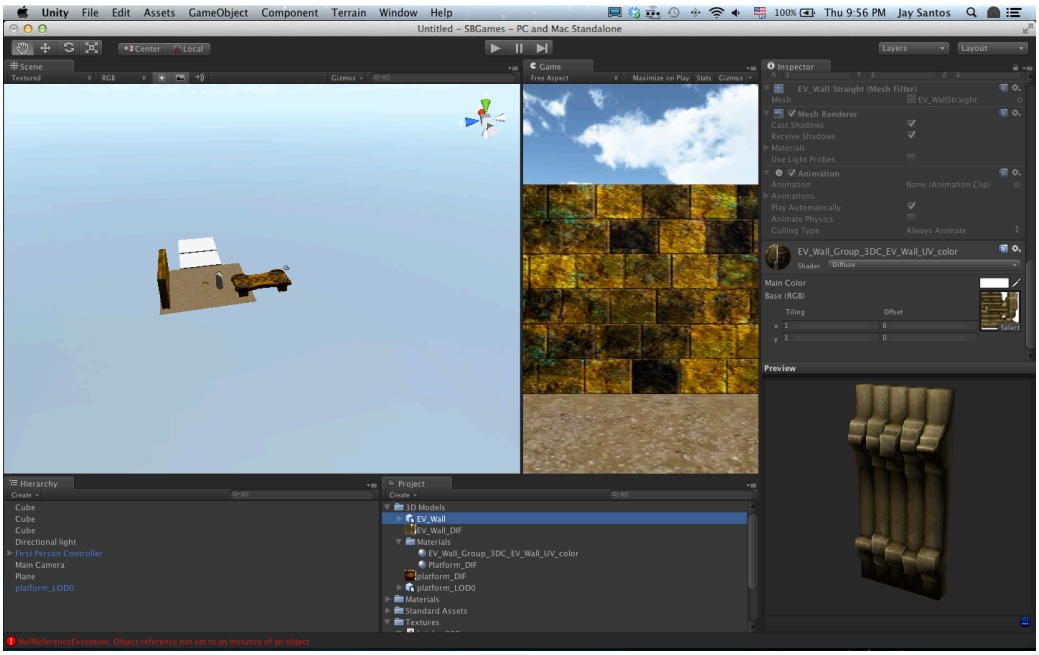

Fig. 14: Modelo 3D EV_Wall com a textura já aplicada.

Para trabalhar com o EV_Wall vamos limpar nossa cena um pouco. Na janela "Hierarchy" apague todos os objetos (selecione o objeto, clique com o botão direito e selecione "Delete") **com exceção de:**

- First Person Controller
- Plane
- Directional Light

Clique em EV_Wall na janela Project e arraste para a janela Scene para adicionarmos uma parede a cena. Agora queremos fazer com que essa parede comece exatamente em cima do nosso chão, para fazer isso vamos usar o comando de "Vertex Lock".

Para fazer isso, clique na parede na janela Scene e clique no botão "Move the selected objects" no canto superior esquerdo da janela (ou use o atalho "W"). Feito isso ponha o cursor em cima do muro e aperte a tecla "V", você vai notal que o cubo para movimentação do muro vai se mover para cima do cursor, e se você mover o cursor o cubo vai acompanhar. Posicione o cursor em um dos cantos inferiores do muro, depois clique no muro e arraste. Você vai notar que a movimentação do muro agora se dá em pulos. Isso porque com o "V" pressionado, o vértice selecionado do muro está se conectando diretamente ao vértice do chão mais próximo. Dessa maneira você garante que o muro está começando assim que o chão termina (Fig. 15).

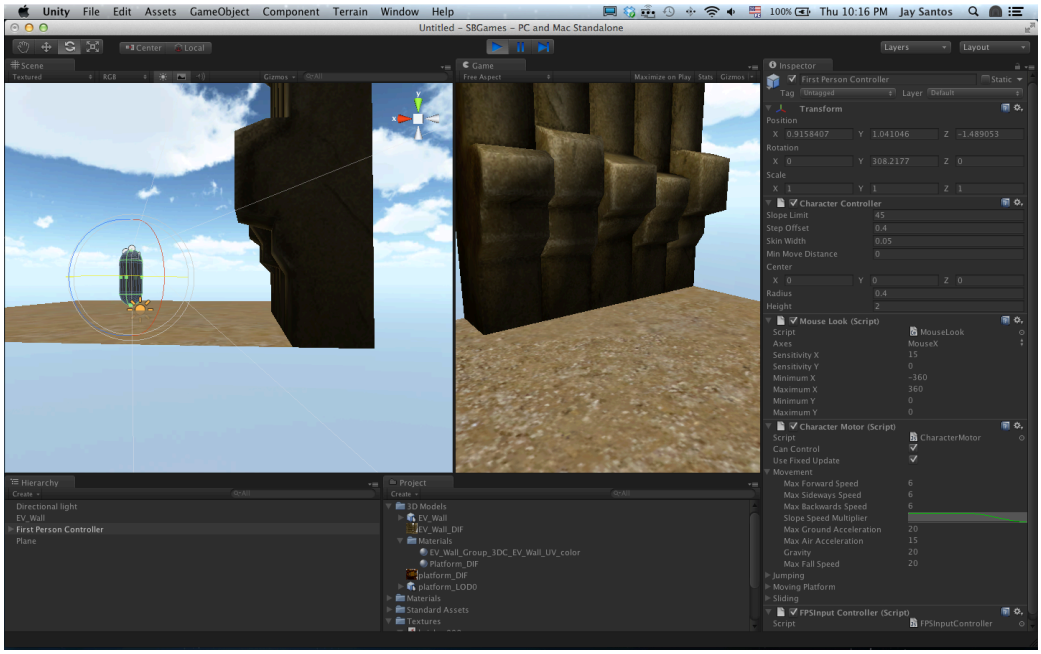

Fig. 15: Muro "preso" ao chão após o uso do comando Vertex Lock

Podemos usar o Vertex Lock para facilmente expandir o muro. Selecione o muro na janela "Scene" e pressione CTRL+D (ou command+D) para duplicar o objeto, feito isso arraste o muro duplicado para o lado usando a seta verde para arrastá-lo e afaste-o do muro original. Depois disso pressione o "V", selecione um vértice do canto inferior e prenda-o a um vértice do canto inferior do muro original. Repita o procedimento algumas vezes até você se sentir confortável com esse procedimento (Fig. 16).

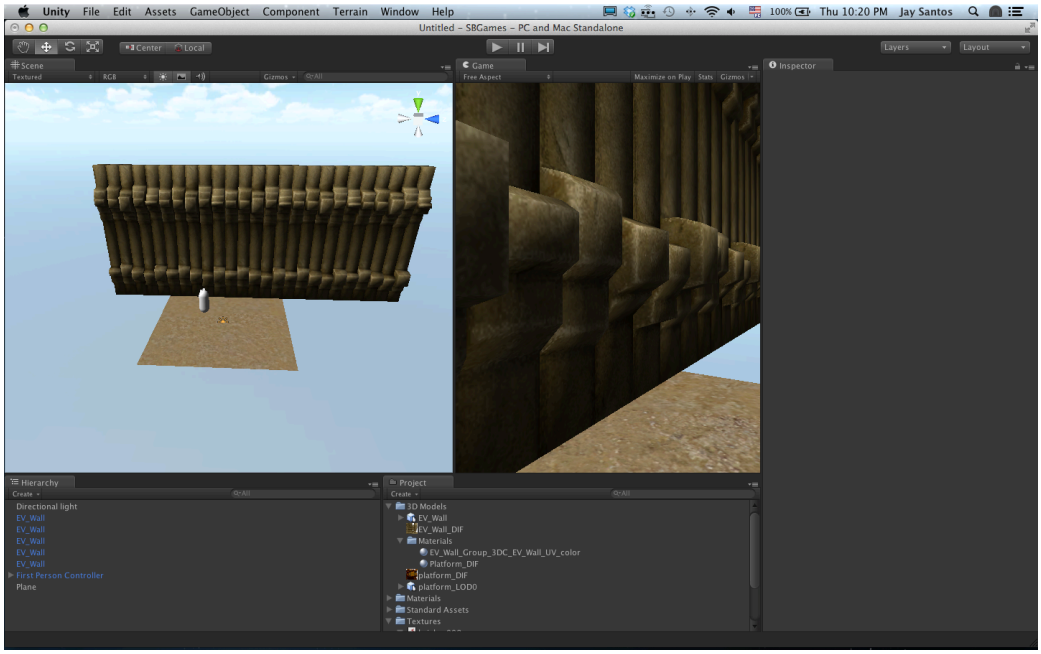

Fig. 16: Muro copiado cinco vezes (Note cinco objetos EV_Wall na janela "Hierarchy")

Se você pressionar Shift e clicar em cada um dos muros criados na janela "Scene", você pode selecionar todos os muros e trabalhar com todos ao mesmo tempo.

5. Prefabs e Packages

Prefabs (de prefabricated, pré-fabricado), são objetos que você pode replicar e utilizar diversas vezes durante seu projeto. O uso de prefabs facilita muito a criação de items iguais ou mesmo com comportamento similares. Vamos criar um prefab contendo o muro com vários segmentos que criamos.

Colapse todas as pastas da janela "Project" e clique em "GameObject -> Create Empty". Você vai notar um novo objeto na janela "Hierarchy" chamado GameObject.

Todo objeto de um projeto Unity é um "GameObject". É importante saber isso para a programação de scripts pois praticamente todas as classes são herdeiras da classe GameObject.

Selecione todos os EV_Wall da janela "Hierarchy" utilizando Shift e clicando nos EV_Wall e arraste todos para cima do GameObject recém-criado. Você vai notar que todos os EV_Wall agora estão abaixo do GameObject, isso significa que agora todos eles são filhos do GameObject (vamos falar mais a respeito no futuro). Agora arraste o GameObject para a janela "Project". Foi criado um objeto novo com um cubo como ícone. Esse cubo significa que esse objeto é um prefab (Fig. 17).

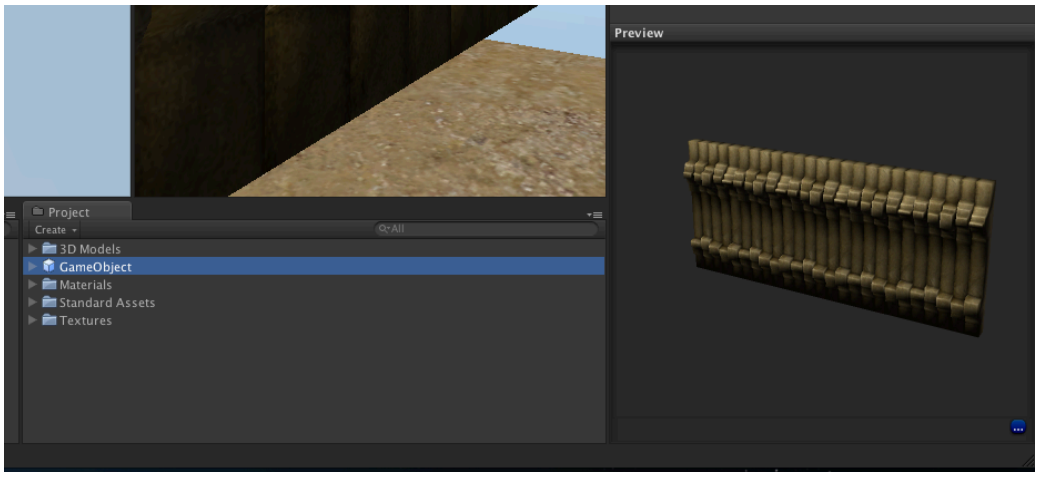

Fig. 17: Prefab recém criado na janela "Project". Note no preview o muro mais extenso que criamos.

Para entender como funcionam os prefabs, selecione o objeto chamado "GameObject" na janela "Hierarchy" e apague-o (clique com o botão direito e selecione "Delete"). Todos os muros da cena irão sumir. Agora clique no Prefab criado na janela "Project" e arraste para dentro da janela "Scene ". Você vai notar que um muro extenso foi criado. Como esse prefab está disponível no meu Projeto ele pode ser usado em qualquer cena do meu jogo.

Mas você também pode usar o prefab em outros projetos, criando um Package. Para criar um package, clique com o botão direito no prefab GameObject na janela "Project" e selecione "Export Package". Um pop-up mostrando os elementos que serão inclusos no package é exibido. Note no pop-up que além do prefab também são inclusos todos os modelos 3D, materiais e scripts que estão associados a esses objetos (no nosso caso não temos nenhum script associado ainda), clique em export, dê um nome ao package, defina onde você vai salvá-lo e clique em "Save". Será criado um arquivo .unitypackage contendo o prefab que foi criado.

Agora esse package pode ser importado para outros projetos.

6. Importando um package

Já temos tudo que precisamos para criar nossa fase do FPS, mas para acelerar o processo vamos importar um arquivo .unitypackage com nossa fase já pronta.

Apague todos os elementos da cena e deixe apenas:

- First Person Controller
- Plane

Agora clique em "Assets -> Import Package -> Custom Package..." e selecione o arquivo game_level_prefab.unitypackage e clique "Import" no pop-

up. Na janela "Project" dentro de "Game_level_and_Props -> Prefab" há um prefab chamado "Level", arraste-o para a janela "Scene". Ajuste a posição do First Person Controller e do Plane dentro da janela Scene para que o First Person Controller fique em cima de uma das plataformas e o Plane cubra toda a parte debaixo da fase (Fig. 18).

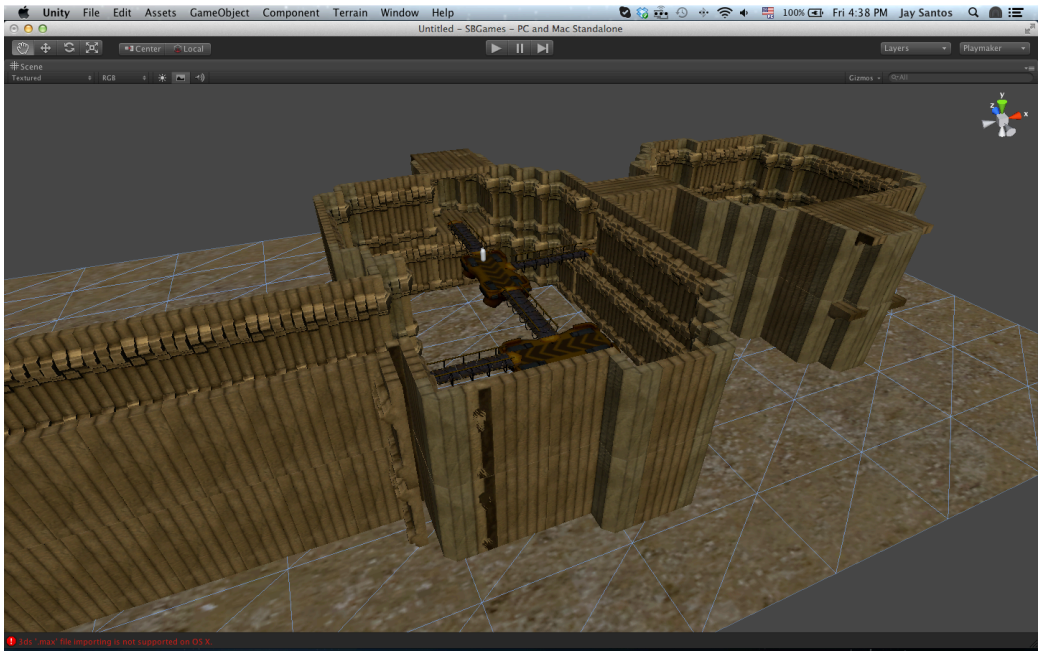

Fig. 18: Janela Scene mostrando o First Person Controller em cima de uma plataforma e o Plane cobrindo a fase todas

Já temos tudo que precisamos para criar nossa fase do FPS, mas para acelerar o processo vamos importar um arquivo .unitypackage com nossa fase já pronta.

A cena está bastante escura, vamos ajustar a iluminação, mudando a iluminação ambiente e acrescentando um Directional Light.

Para mudar a iluminação ambiente clique em "Edit -> Render Settings" e clique em "Ambient Light", e ajuste-a para aproximadamente o exibido na Fig. 19, você vai perceber que a cena já ficará um pouco mais clara.

Fig. 19: Configuração do Ambient Light

Agora clique em "Game Object -> Create Other -> Directional Light", na janela "Inspector" na sessão "Light", mude a opção "Shadow Type" para "Soft Shadows", e você vai perceber que a cena ficará mais iluminada e aparecerão sombras. Após acrescentar a luz, na janela "Inspector" altere a propriedade "Rotation" para:

- **X: 50**
- **Y: 260**
- **Z: 0**

Dessa maneira teremos algumas áreas de sombra que utilizaremos para a parte de Lightprobing (Fig. 20).

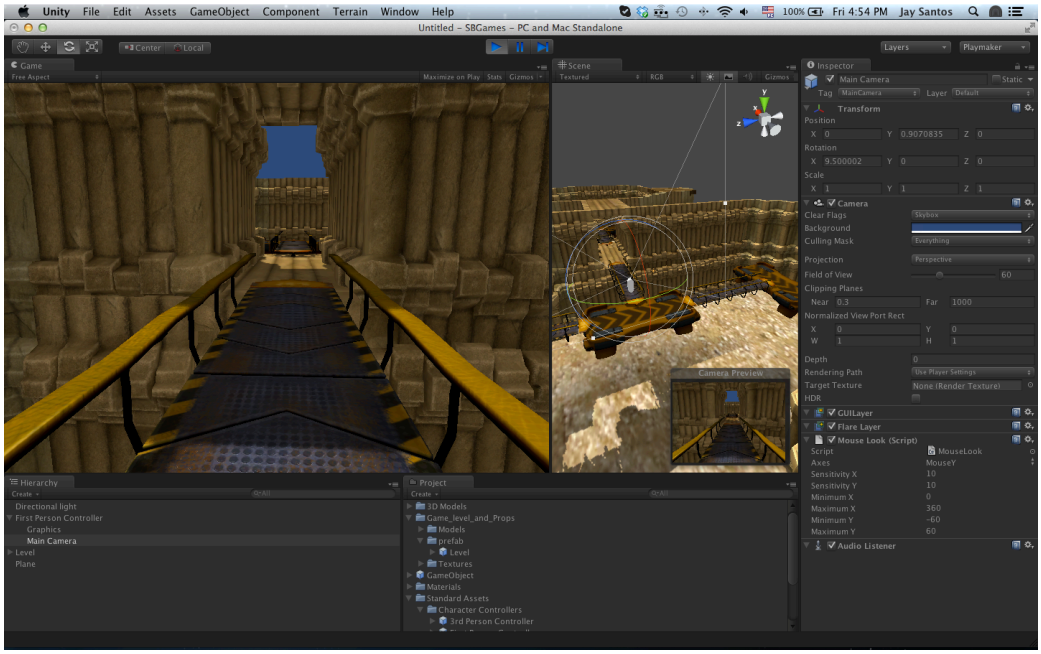

Fig. 20: Fase com a Directional Light acrescentada, note a configuração de rotação e a área de sombra abaixo do túnel.

6. Acrescentando água, arma e animação da arma

Para acrescentar a água, na janela "Project", na pasta "Standard Assets -> Water (Pro Only) ou Water (Basic)", clique em "Daylight Simple Water" e arraste-a para a tela "Scene". Ajuste o tamanho para cobrir toda a fase e coloqueá um pouco acima do plano original (Fig. 21).

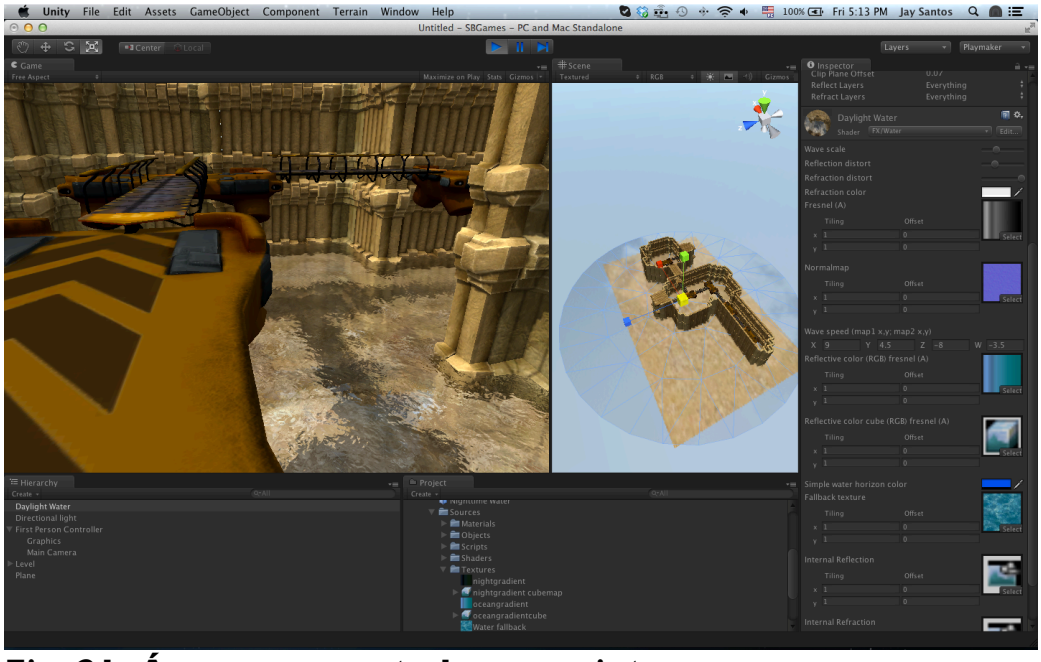

Fig. 21: Água acrescentada ao projeto

Após colocar a água, vamos acrescentar uma arma ao nosso jogador. Arraste para dentro da pasta "3D Models" na janela "Projects" os arquivos:

- **bazooka.fbx**
- **bazooka_colormal.tga**
- **bazooka_normalmap.tga**

Se um pop-up aparecer, pode clicar em "Fix now". Como os arquivos já estão com os nomes corretos, se você clicar no prefab bazooka dentro da pasta "3D Models", na janela inspector já será exibido o modelo 3D com a textura aplicada corretamente.

Agora clique no prefab bazooka dentro de "3D Models" e arraste-o para cima do objeto Main Camera dentro de First Person Controller na janela "Hierarchy". Dessa maneira o objeto bazooka ficará dentro do Main Camera, isso significa que bazooka agora é filho de Main Camera, e sempre que Main Camera se mover, bazooka se moverá também (fig. 22).

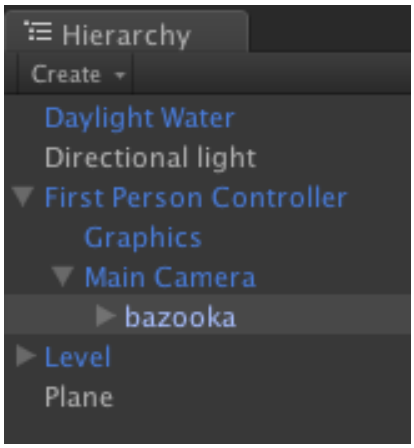

Fig. 22: bazooka como filho de Main Camera na janela Hierarchy

Precisamos ajustar a posição da bazooka para que ela fique no canto inferior direito da tela. Você pode ajustar manualmente através da janela "Scene", ou então selecione bazooka na janela "Hierarchy", e configure o parâmetro "Position" da janela "Hierarchy" para:

- **X: 0.7**
- **Y: -0.1**
- **Z: 0.7**

Sua janela "Game" ficará similar a da Fig. 23.

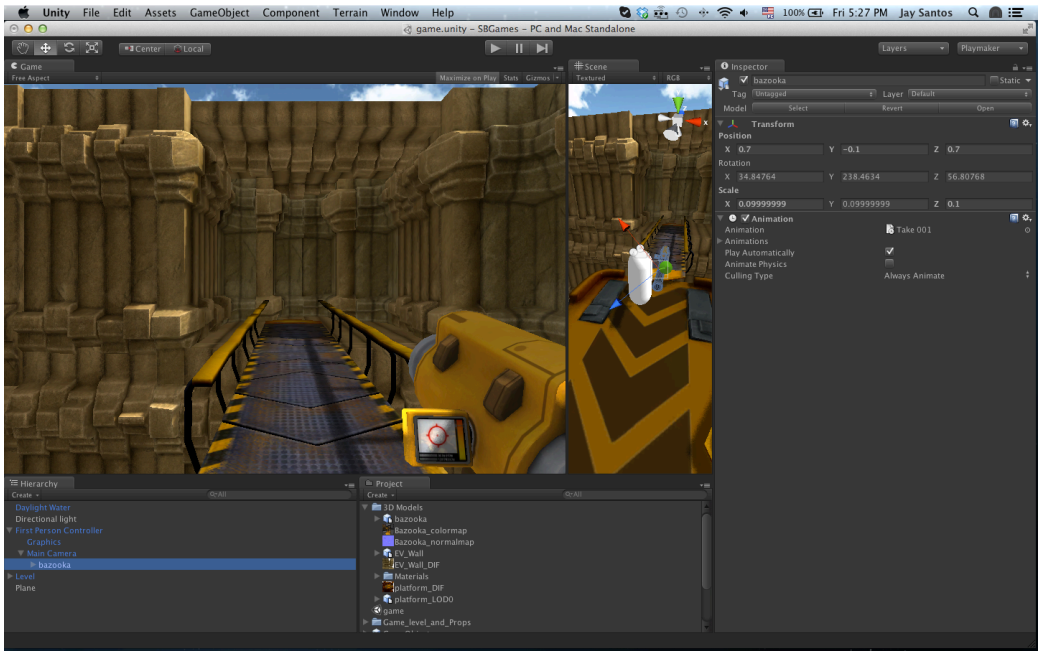

Fig. 23: Posição da arma após ajuste da posição

Clique em Play, você pode perceber que a animação da arma é executada apenas uma vez e a arma para. Isso acontece pois as duas animações associadas a esse modelo (a animação quando a arma está "Idle" e quando a arma é disparada) estão salvas como uma animação só. Agora vamos dividir estas animações e aplicar a animação de "Idle". Na janela "Project" embaixo da pasta "3D Models" clique no prefab Bazooka. Na janela "Inspector", procure a seção "Animations", logo abaixo de "Split Animations" há uma lista que por enquanto está vazia (Fig. 24).

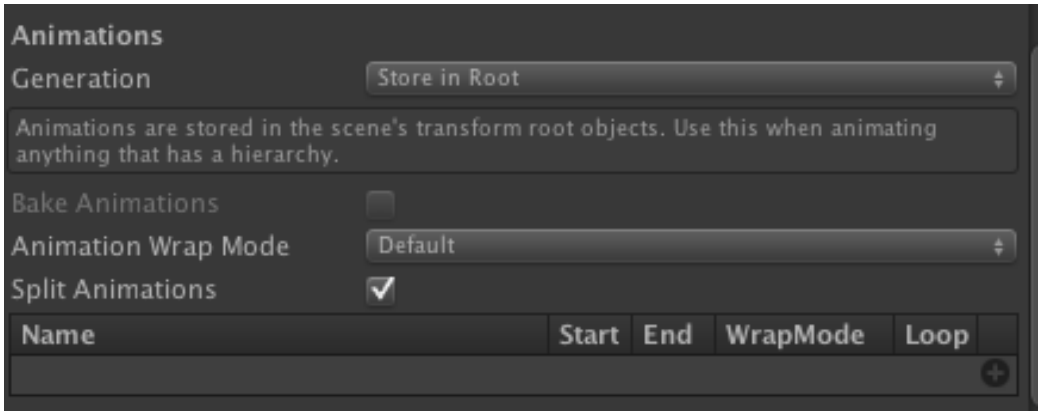

Fig. 24: Seção "Animations" do prefab. A tabela está na parte debaixo da imagem.

Clique no "+" para acrescentar uma animação a lista, os parâmetros da animação são:

- **Name:** O nome da animação, pode manter "idle"
- **Start:** Quadro de início da animação, pode manter 1
- **End:** Quadro de término da animação, configure para 60
- **WrapMode:** Modo de Wrap da animação, para esta animação configure como "Loop"

Clique novamente no "+" para acrescentar mais uma animação, a configuração da nova animação deve ser:

- **Name:** "shoot"
- **Start:** 61
- **End:** 75
- **WrapMode:** Once

Desça um pouco mais na janela "Inspector" e clique no botão "Apply". Clicando em "Play" você pode notar que a animação de idle (a arma subindo e descendo) fica ativada em loop.

7. Criando a munição e o processo de disparo

O próximo passo é criar a munição da arma. Vamos usar um cubo como munição. Clica em "GameObject -> Create Other -> Cube", diminua o tamanho do cubo (na janela "Inspector", configure os parâmetros X, Y e Z de "Scale" para 0.15) e aperte Play. Você pode notar que o cubo fica flutuando no espaço. Para que a gravidade (e que a gente possa aplicar outras forças no cubo), você deve adicionar um Rigidbody ao cubo. Na janela "Hierarchy" selecione o Cubo, depois clique em "Components -> Physics -> Rigidbody", após fazer isso se você apertar Play você vai notar que o cubo vai cair.

Agora vamos definir a Tag associada a nosso cubo. A Tag vai ser utilizada mais para a frente quando analisarmos a colisão da bala com o inimigo. Na janela "Inspector", clique no pull-down ao lado de Tag no canto superior esquerdo e selecione a opção "Add Tag...", expanda a opção "Tags" (primeira linha) e na linha "Element 0" escreva "Bullet"

Vamos definir um tempo para o cubo desaparecer após ser disparado. Na janela "Project" clique em "Create -> C# Script", procure o objeto NewBehaviourScript e renomeie-o para "BulletController" e clique duas vezes no objeto para abrir o editor. Uma vez aberto, altere o nome da classe para BulletController. A classe que foi criada já tem duas funções:

- **Start():** Essa função é chamada sempre que o GameObject ao qual o script está associado é criado.
- **Update():** Essa função é chamada toda vez que um frame é desenhado.

O que queremos é que, depois de um determinado tempo, o cubo desapareça. Na função Start() acrescente a seguinte linha:

```
Destroy(gameObject, 5.0f);
```

O que essa linha faz é destruir o gameObject ao qual o script está associado em cinco segundos. Salve o script, volte ao Unity e na janela "Project", clique no BulletController e arraste-o ao objeto Cube na janela "Hierarchy", se depois disso você clicar no Cube em "Hierarchy", voce vai notar no "Inspector" que surgiu uma nova sessão chamada "Bullet Controller (Script)" (Fig. 25).

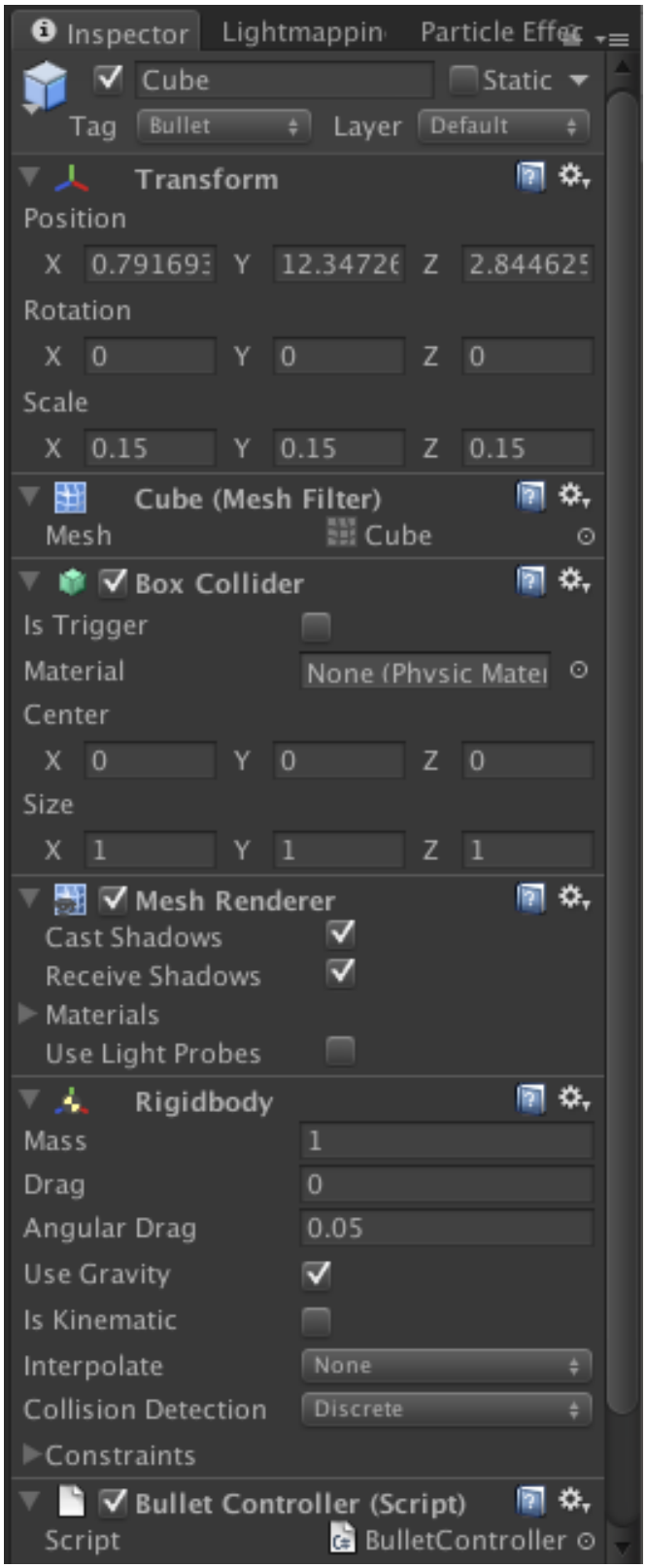

Fig. 25: "Inspector" do Cube com o Bullet Controller na parte de baixo. Note também a Tag "Bullet" já associada ao Cube

Clique em Play, depois de cinco segundos o cubo desaparece do jogo e desaparece também do "Hierarchy" (lembre-se, o Hierarchy mostra apenas os objetos que existem na cena, como destruímos o objeto ele deixa de existir).

Mas da maneira como o tempo para destruição do cubo está configurado, a única maneira de alterá-lo é entrando no código fonte e alterando o valor, vamos facilitar esse processo, volte ao editor do código, e crie uma variável na classe:

```
public float timeToLive = 5.0f;
```

Depois disso, altere a chamada de `Destroy` para:

```
Destroy(gameObject, timeToLive);
```

Depois disso, volte ao Unity, clique no Cubo na janela "Hierarchy" e repare no "Inspector" que dentro da seção "Bullet Controller" você agora pode configurar a variável do tempo de destruição do cubo sem ter que alterar o código fonte (Fig. 26).

Fig. 26: Inspector agora com o parâmetro Time To Live

Arraste o Cube da janela "Hierarchy" para a janela "Project". Dessa maneira vamos criar um prefab do Cube que usaremos no futuro para criar as balas que são disparadas pela nossa arma. Agora você já pode apagar o Cube da Scene.

Agora vamos criar o processo de disparo da arma, o processo é basicamente o seguinte:

- Quando o botão de disparo for pressionado:
  - Uma bala será criada na frente da arma
  - A arma vai tocar a animação de "Shoot" e depois voltar para "Idle"
  - Um som será tocado

Primeiramente vamos definir a posição onde serão geradas as balas que são disparadas. Clique em "GameObject -> Create Empty". Esse GameObject será "vazio", apenas com um script associado, portanto para facilitar sua visualização vamos acrescentar um label a esse GameObject. Selecione o GameObject criado na janela "Hierarchy" e clique no cubo verde, vermelho e azul no canto superior esquerdo da janela "Inspector" e selecione um dos

ícones apresentados. Agora um label com o nome do GameObject estará presente na janela Scene sempre (Fig. 27).

Fig. 27: Label GameObject presente na janela Scene

Antes de posicionar o GameObject vamos renomeá-lo e torna-lo filho da bazooka (para que ele acompanhe o objeto bazooka), clique com o botão direito no GameObject na janela "Hierarchy" e selecione "Rename" e altere o nome para "BulletSpawn". Agora posicione o GameObject vazio em frente a arma arrastando o objeto na janela "Scene" ou selecione o objeto

"BulletSpawn" na janela "Hierarchy", e na janela "Inspector" configure os parâmetros de position para:

- **X: 0**
- **Y: -7.7**
- **Z: -1.2**

Feito isso, selecione o arquivo BulletShooter.cs e arraste para dentro do diretório "Project" e depois arraste-o da janela "Project" para o objeto BulletSpawn na janela "Hierarchy". Feito isso, clique no objeto BulletSpawn na janela "Hierarchy" e verifique se agora há uma seção chamada "Bullet Shooter (Script)" (Fig. 28).

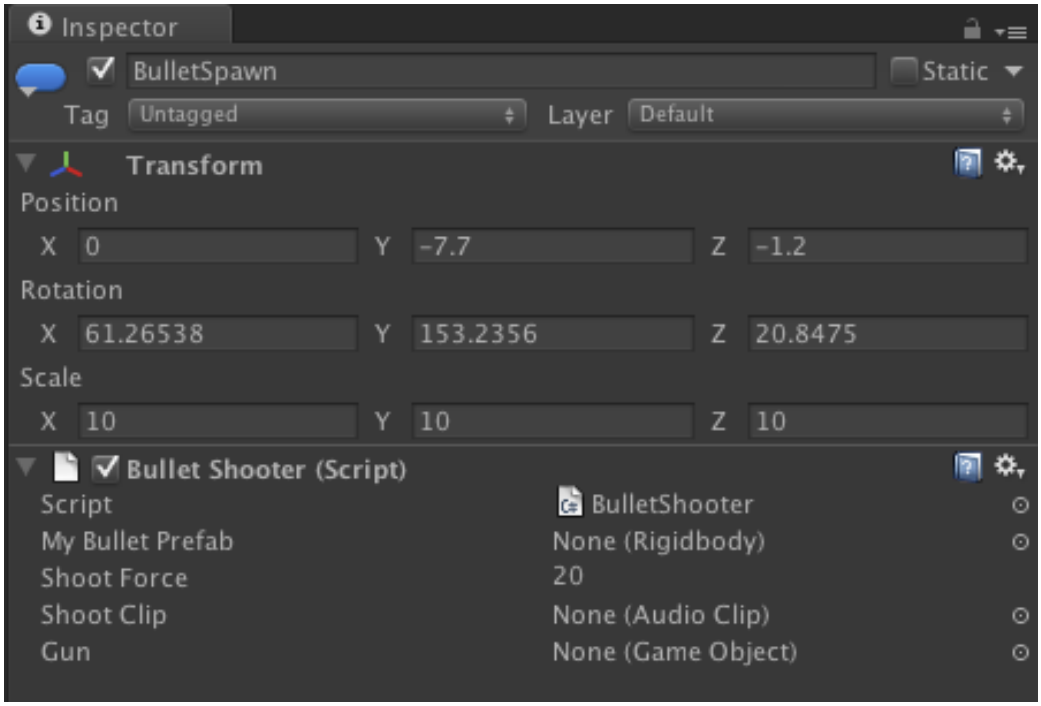

Fig. 28: Janela "Inspector" do BulletSpawn com o script

Vamos analisar o que está sendo feito no script BulletShooter.cs, a função Start() não faz nada, ou seja, quando o cubo é criado nada acontece, na função Update():

```
void Update ()
{
     if(Input.GetButtonDown("Fire1"))
     {
          Rigidbody bullet = Instantiate(myBulletPrefab,
transform.position, transform.rotation) as  Rigidbody;
          bullet.velocity =
transform.TransformDirection(new Vector3
(0,0,shootForce));
          audio.PlayOneShot(shootClip);
          gun.animation.Play ("shoot");
          gun.animation.PlayQueued ("idle");
          }
     }
```

Toda vez que a tela é redesenhada, eu verifico se o botão foi pressionado (if(Input.GetButtonDown("Fire1"))):

- `Rigidbody bullet = Instantiate(myBulletPrefab, transform.position, transform.rotation) as Rigidbody;` – Eu crio uma cópia do prefab `myBulletPrefab` na posição e rotação do objeto ao qual está associado o script (no caso, o BulletSpawn).
- `bullet.velocity = transform.TransformDirection(new Vector3 (0,0,shootForce));` – Defino a velocidade do prefab instanciado para X = 0, Y = 0 e Z = `shootForce`.
- `audio.PlayOneShot(shootClip);` – Começo a tocar o audio
- `gun.animation.Play ("shoot");` – Começo a tocar a animação "shoot"
- `gun.animation.PlayQueued ("idle");` – Assim que a animação atual terminar, começo a tocar a animação "idle"

Agora é necessário definer as variáveis utilizadas pela classe, que são:

```
public Rigidbody myBulletPrefab;
public int shootForce = 20;
public AudioClip shootClip;
public GameObject gun;
```

- `Rigidbody myBulletPrefab:` Esse é o prefab da bala que criamos
- `int shootForce:` Velocidade inicial da bala
- `AudioClip shootClip:` O som a ser tocado quando a arma é

disparada
- `GameObject gun:` A arma, precisamos saber qual é a arma para poder tocar as animações shoot e idle

Como demonstramos quando criamos a bala, como essas variáveis são publicas elas podem ser alteradas dentro do editor:

- **Variável myBulletPrefab:** Arraste o prefab Cube da janela "Project" para o campo My Bullet Prefab da seção "Bullet Shooter (Script)"
- **Variável shootForce:** Mantenha o valor 20
- **Variável shootClip:** Arraste o arquivo bang.wav para a janela "Project" e depois arraste o objeto bang da janela "Project" para o campo Shoot Clip da seção "Bullet Shooter (Script)"
- **Variável Gun:** Arraste o objeto bazooka da janela "Hierarchy" para o campo Gun Clip da seção "Bullet Shooter (Script)"

Sua configuração deve estar igual a Fig. 29.

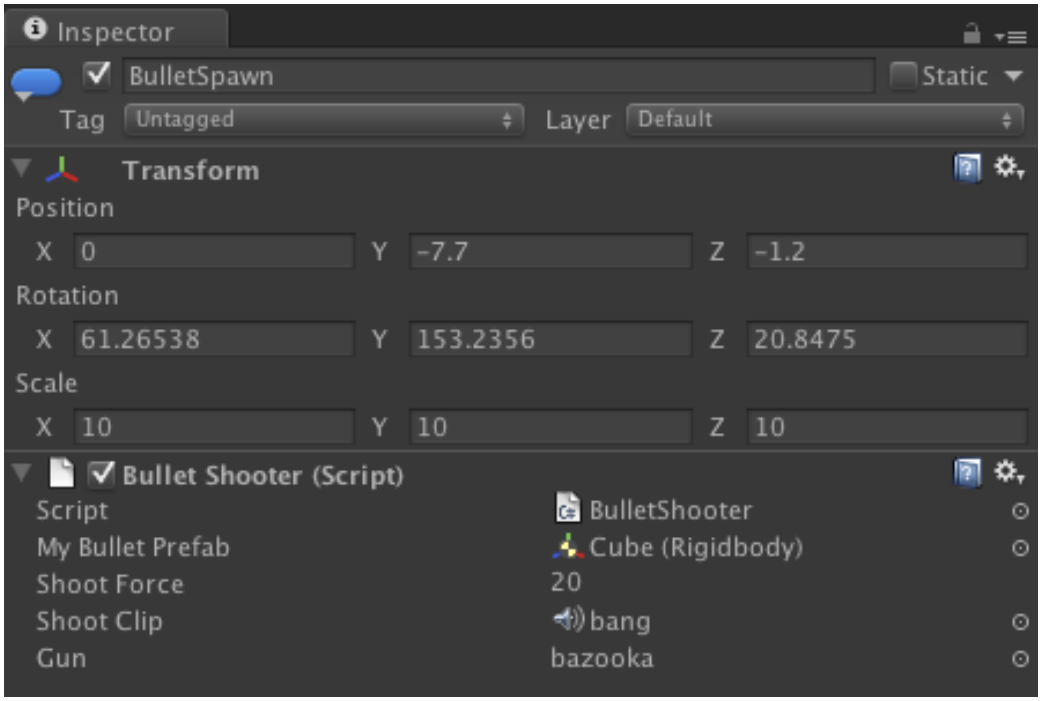

Fig. 29: Janela "Inspector" do objeto BulletSpawn após configuração das variáveis

Clique Play e teste o disparo clicando o botão esquerdo do mouse. Tudo funciona exceto o som. Isso porque é necessário também acrescentar um componente chamado **AudioSource** ao objeto BulletSpawn, para que o som possa ser executado. Clique no objeto BulletSpawn na janela "Hierarchy",

depois clique em “Component -> Audio -> Audio Source”. Clique Play e agora o áudio deve estar funcionando corretamente.

8. Criando um inimigo

Agora vamos criar um robô que será o nosso alvo. Clique em “Assets -> Import Package -> Custom Package” e selecione o arquivo robot.unitypackage, clique em “Import” no pop-up. Selecione o prefab Robot_Animated e arraste-o para a janela “Hierarchy”, depois posicione o robô em frente a câmera (Fig. 30).

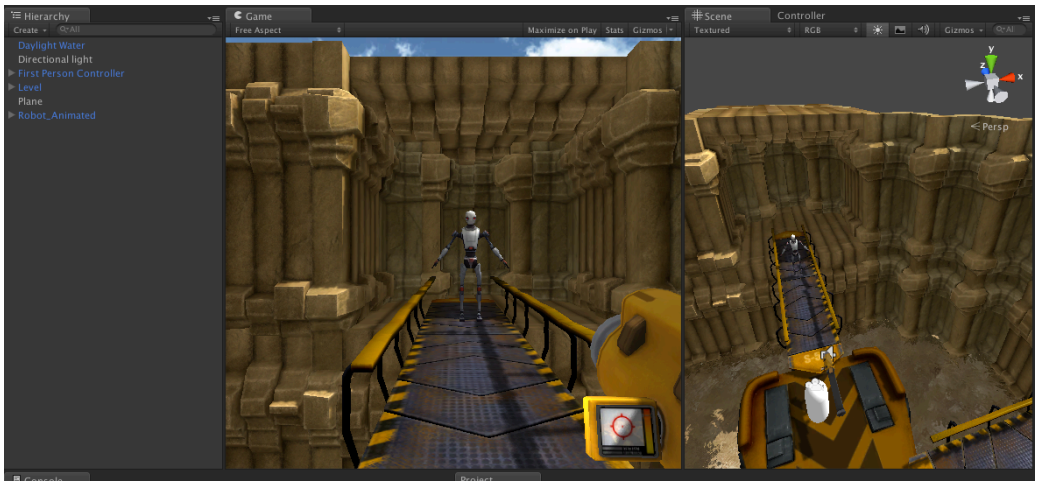

Fig. 30: Robô criado e em frente a câmera

Aperte Play. Você vai notar que o Robô já tem uma animação de Idle (ele fica balançando), tente disparar no robô, você vai notar que a bala passa direto, esse é o mesmo problema da plataforma que tivemos no início do tutorial, o robô não tem um box collider.

Na janela “Hierarchy”, clique no objeto Robot_Animated, depois clique em “Component -> Physics -> Box Collider. Depois você deve ajustar o tamanho e posição do Box Collider de acordo com a Fig. 31.

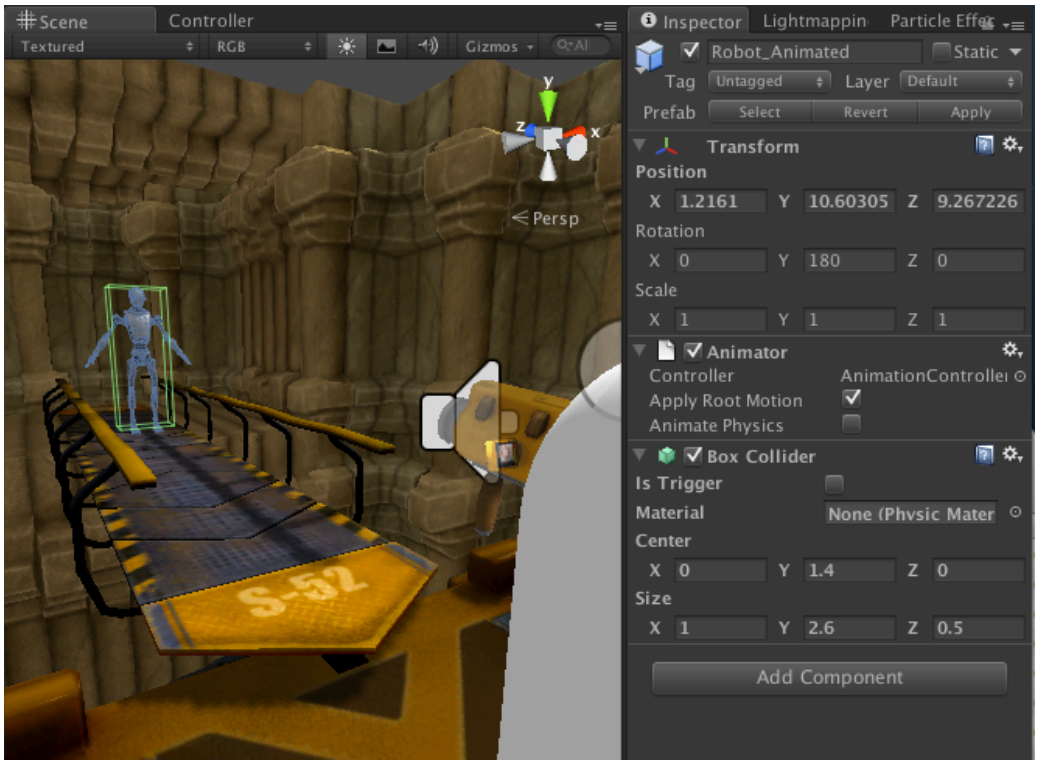

Fig. 31: Configuração do Box Collider para o robô, note o Box Collider (caixa verde) na janela "Scene"

Clique Play, agora as balas "param" no Robô. O próximo passo e fazer o tratamento da colisão da bala com o robô.

Arraste o arquivo BulletHit.cs para a janela "Project", depois arraste o objeto BulletHit para o objeto Robot_Animated na janela "Hierarchy".

Vamos analisar a classe BulletHit:

```
public class BulletHit : MonoBehaviour
{
     void OnCollisionEnter (Collision other)
     {
          if (other.gameObject.tag == "Bullet")
          {
               Destroy (gameObject);
               Destroy (other.gameObject);
          }
     }
}
```

O primeiro detalhe é a falta das funções Start() e Update(), na verdade estas funções não são necessárias, você pode ter uma classe sem ambas sem problemas.

A função `OnCollisionEnter()` é chamada no momento em que um objeto entra numa área de colisão pertencente ao objeto ao qual este script está associado. Neste caso, é o Box Collider do robô que acabamos de criar, e o objeto que entrou na área de colisão é o parâmetro `other` da função. O que está acontecendo dentro desta função:

- `if (other.gameObject.tag == "Bullet")`: Verifica se o objeto que colidiu com a caixa de colisão (other) tem a Tag "Bullet"
- `Destroy (gameObject)`: O objeto ao qual o script está associado é destruído.
- `Destroy (other.gameObject)`: O objeto que colidiu com a caixa de colisão é destruído

Após arrastar o script para o objeto Robot_Animated, clique em Play e tente atirar no robô, agora tanto o robô quanto a bala desaparecem, conforme explicado acima.

9. Acrescentando um Navigation Mesh

Vamos fazer agora com que o inimigo nos siga pelo nível. Para isso vamos criar um Navigation Mesh (mapa indicando os caminhos que podem ser percorridos) e adicionar o NavMesh Agent ao nosso robô.

Para fazer parte de um Navigation Mesh, um objeto deve ser "Navigation Static", para verificar o status de um objeto, clique nele na janela "Scene", ou "Hierarchy", e no canto superior direito da janela "Inspector" você tem um pull-down com todas as configurações de estática do objeto (Fig. 32).

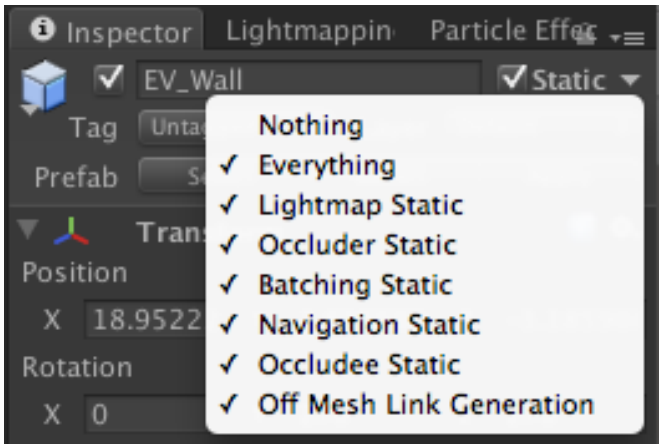

Fig. 32: Configurações de static para um objeto. Como ele é "Navigation Static" ele pode ser utilizado num Navigation Mesh

Para começar é necessário selecionar todos os pisos que farão parte do nosso Navigation Mesh, faça isso clicando em cada uma das partes do chão da nossa fase na janela "Scene" (para selecionar mais de um objeto utilize CTRL+click no Windows ou command+click no MacOS) (Fig. 33).

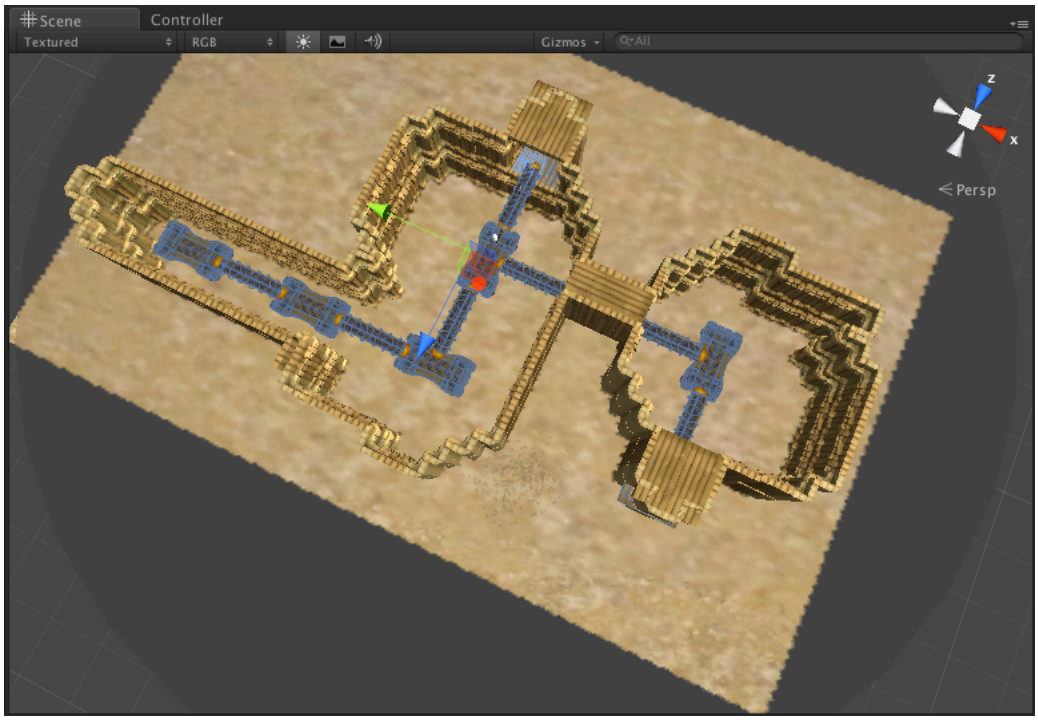

Fig. 33: Todas as partes de chão selecionadas que farão parte do Navigation Mesh (o chão dentro do túnel também foi selecionado).

Uma vez selecionados os segmentos de chão que farão parte do Navigation Mesh, clique em "Window -> Navigation" e clique no botão "Bake" no canto inferior direito (Fig. 34).

Fig. 34: Janela Navigation

Após o termino do Bake, adicione um Nav Mesh Agent ao objeto Robot_Animated: Clique em Robot_Animated na janela "Hierarchy" e depois clique em "Components -> Navigation -> Nav Mesh Agent". Depois arraste os arquivos agentLocomotion.js e robotController.js para a janela "Project" e depois arraste os objetos agentLocomotion e robotController para o objeto Robot_Animated na janela "Hierarchy", depois disso clique em Clique em Robot_Animated na janela "Hierarchy" e na seção "Robot Controller (Script)" altere a variável Nav Distance para 3 e arraste o objeto First Person Controller da janela "Hierarchy" para a variável Nav Target (Fig. 35).

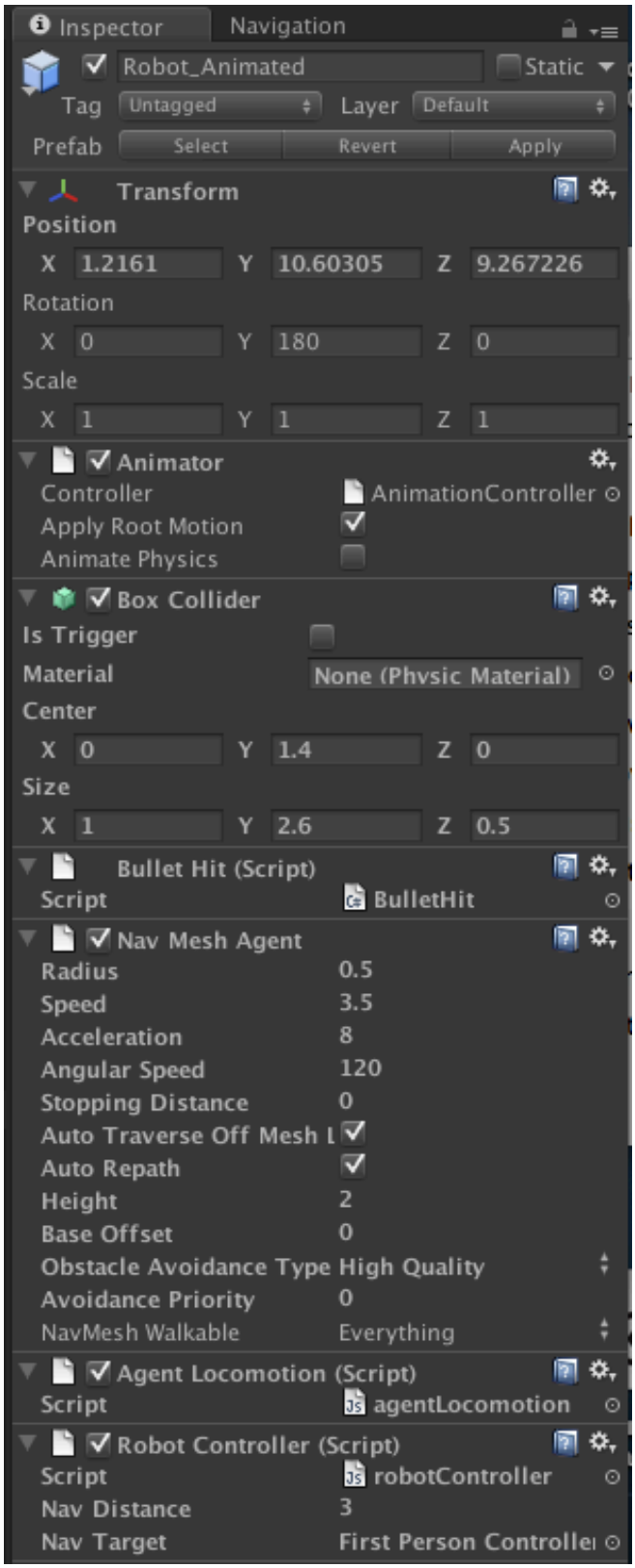

Fig. 35: Configuração do objeto Robot_Animated. Note os componentes adicionados e as configurações do Robot Controller