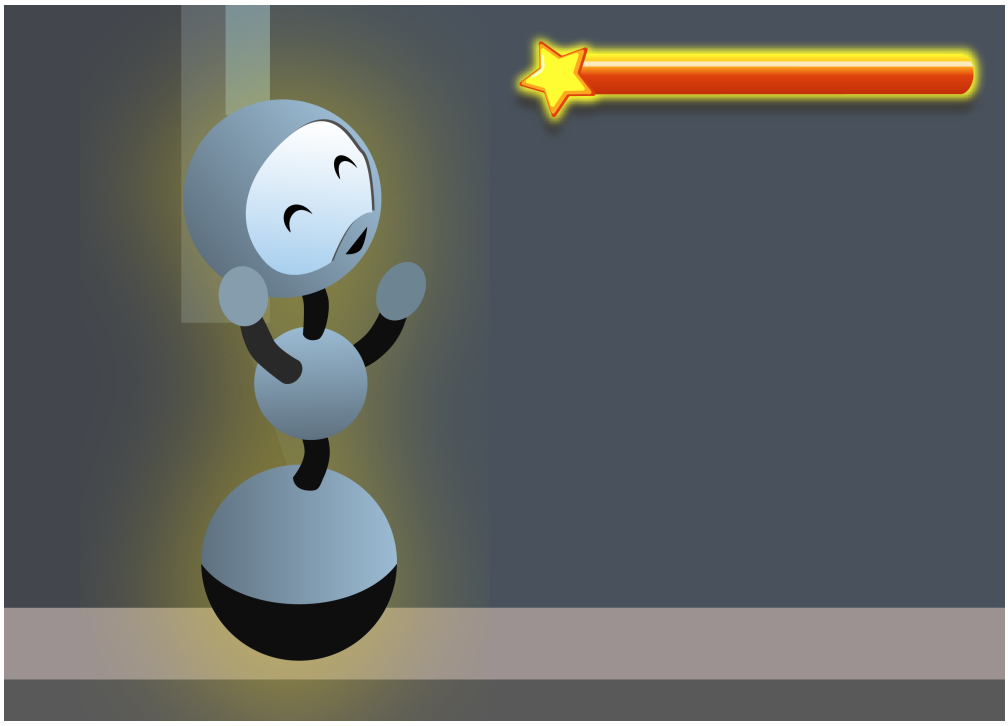
Desenvolvimento com Motores de Jogos I

Aula 15 - Áudio e Compilação do Jogo

Apresentação

Olá, pessoal! Chegamos à nossa última aula da disciplina de Desenvolvimento com Motores de Jogos I! Faremos hoje as nossas últimas modificações no projeto, para, então, gerarmos uma versão final e a compilarmos a fim de que possamos distribuí-la.



Vimos, ao longo das aulas, muitos conteúdos importantes. Não conseguimos nos aprofundar em alguns assuntos do modo como gostaríamos, mas estejam certos de suas capacidades para desenvolverem jogos de estilos diversos, com funcionalidades bem complexas, desde menus até jogabilidade, incluindo elementos do cenário e várias outras coisas que conhecemos ao longo da disciplina.

O projeto exemplo foi muito divertido de desenvolver. Ter um exemplo prático sempre facilita. Não achem, no entanto, que vocês conseguem fazer somente um jogo 2D de plataforma! Buscamos explanar o conteúdo da melhor maneira, justamente para vocês poderem desenvolver o que desejarem!

Na aula de hoje, veremos o último aspecto restante dentre as coisas que um jogo usualmente tem. Conheceremos o sistema de som do Unity, adicionando, através dele, uma trilha sonora e alguns efeitos sonoros ao nosso projeto.

Com isso, concluiremos uma versão, ainda que bem básica, de nosso jogo. E, então, veremos como é simples gerar uma versão standalone, ou seja, uma versão capaz de ser executada sozinha, sem a necessidade de instalação de qualquer outra ferramenta, ou mesmo da utilização do Unity.

Sabemos que ainda há bastante coisa para aprendermos no mundo do desenvolvimento de jogos, mas sabemos, também, que toda a base construída aqui será de grande valia para vocês seguirem adiante. E a disciplina de Desenvolvimento com Motores II serve para isso!

Aproveitem bem essa última aula, revisem as outras a fim de ver se todos os conceitos foram bem absorvidos e, como sempre digo, não fiquem tímidos para utilizar o fórum! Sempre alguém estará por lá para ajudar vocês e garantir que a experiência será a melhor possível!

Divirtam-se! o/

Objetivos

Ao final desta aula, você deverá ser capaz de:

- Utilizar os principais elementos de som do Unity;
- Adicionar sons às suas cenas;
- Compilar um jogo para distribuição, utilizando as configurações básicas de compilação.

1. O Áudio no Unity

Toque isso uma vez.

Isso te lembra alguma coisa? Para muitos, sim! Mario! E basta esse pedacinho da música para você logo lembrar do jogo e de tudo ligado a ele. Talvez uma fase que você não conseguia passar, ou até a pessoa com quem você jogava naquela época. Talvez uma fita alugada na locadora, para a galera mais... experiente! Ou um ROM pirata que poderia te colocar na cadeia, para os mais novos e aventureiros.

De qualquer maneira, o som ajuda a criar uma identidade para o jogo, seja através dos efeitos sonoros ou da trilha sonora em si. Esses dois aspectos são igualmente importantes e fazem parte de um jogo bem desenvolvido de qualquer estilo. Em jogos de Puzzle, a música acelerando é capaz de criar uma atmosfera de urgência, à medida que o tempo avança. Em jogos de mundo aberto, uma música condizente com o ambiente ajuda muito na imersão do jogador.

Sendo assim, sempre teremos o som como parte importante na concepção, no desenvolvimento e no produto final de um jogo. E nós ainda não temos nada disso no nosso! Começaremos a nossa última aula justamente entendendo melhor o funcionamento dessa parte importante do Unity e, em seguida, veremos como podemos adicioná-la ao nosso projeto.

O Unity divide o áudio em três principais elementos: temos os componentes Audio Listener, Audio Source e o asset Audio Clip. Esse asset representa um som que o os outros dois componentes são capazes de utilizar. Conheceremos em detalhes cada um desses elementos.

1.1 O Audio Listener

O componente Audio Listener, no Unity, equivale aos ouvidos de sua cena! É através desse componente que todo o áudio será captado para ser reproduzido no sistema de som do dispositivo no qual o jogo está sendo executado. O Unity, por padrão, é capaz de simular todo o áudio de uma cena detalhadamente, incluindo espacialidade e, também, alguns efeitos sonoros simples, como zonas de reverberação, as quais adicionam eco à sua cena.

O Unity trabalha com o posicionamento 3D do áudio, tocando-o adequadamente de acordo com os canais de áudio disponíveis (como o lado esquerdo e o direito de um fone de ouvido, por exemplo). Todo esse áudio é captado justamente no componente Audio Listener.

Por padrão, a Main Camera de uma cena já vem com um componente desse tipo atrelado a ela. Diferentemente do que estamos acostumados, o componente Audio Listener não possui qualquer propriedade e não necessita de qualquer configuração. Ele é bem simples! Nós o adicionamos a um componente, o som é simulado pelo Unity e, então, o que chegar até aquele componente é repassado ao sistema de som do usuário.

> Para isso não gerar qualquer tipo de problema ou conflito, é necessário haver apenas um Audio Listener por cena, a fim de evitar que o áudio seja capturado múltiplas vezes e conflitos sejam gerados ao utilizá-los.

Além disso, dependendo do tipo de jogo que está sendo desenvolvido, é interessante mover o Audio Listener para o personagem, removendo-o da câmera principal. Assim, o áudio será mais precisamente renderizado em relação ao personagem e não à câmera em si. Isso melhora a precisão do áudio, de modo a facilitar a detecção de sua origem, por exemplo. Em jogos mais simples, como o nosso, não é necessário fazer essa troca uma vez que a câmera estará sempre no personagem e estamos trabalhando apenas em 2D.

Mas, e de onde vêm esses áudios que são “ouvidos” pelo nosso Audio Listener? Eles vêm de Audio Clips, sendo estes criados a partir de Audio Files, utilizados pelos Audio Sources e colocados em objetos regulares! Quanto "Audio", não? Pois é! Vamos conhecer esses componentes.

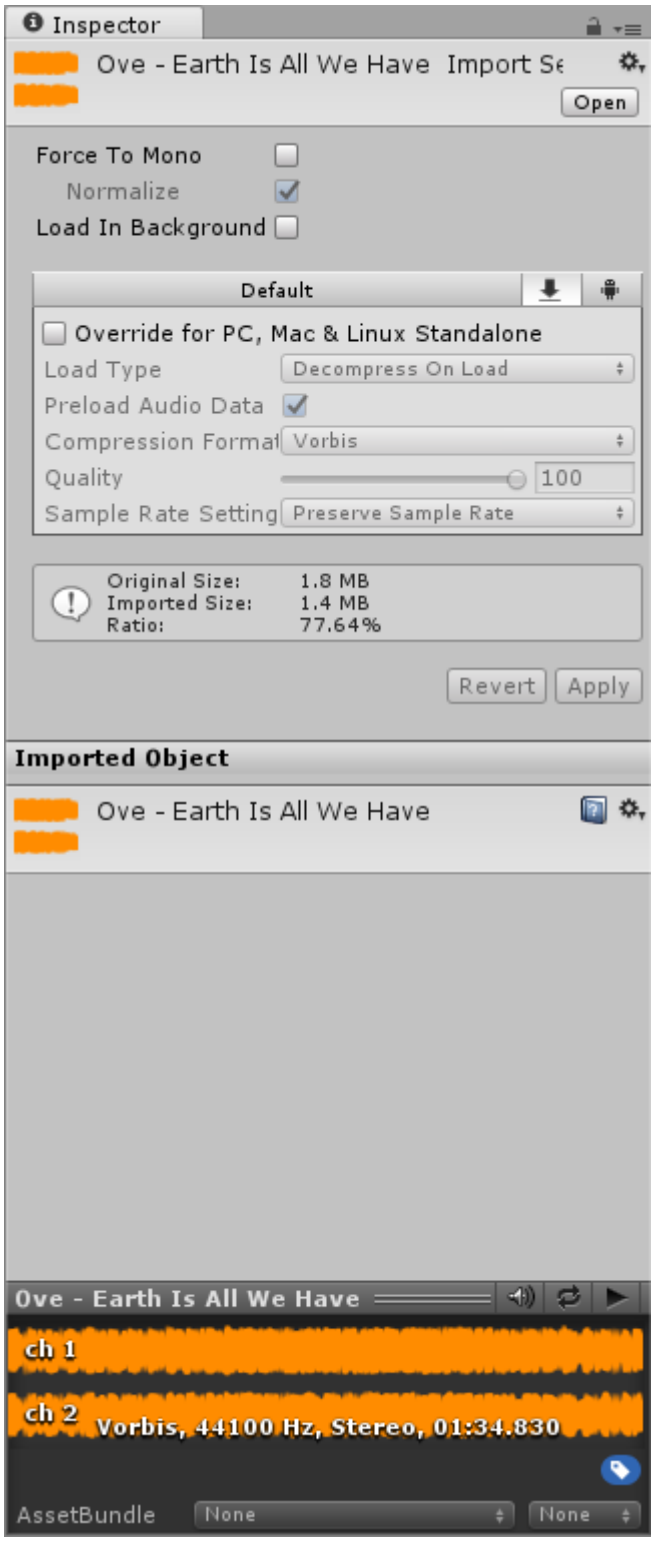
1.2 Os Audio Clips

O segundo elemento de áudio que conheceremos em nossa aula são os Audio Clips. Eles contêm os arquivos de áudio a serem utilizados em nossa cena, pelos Audio Sources, para reproduzir áudio no Audio Listener. O Unity se esforça bastante

para facilitar a nossa experiência e é possível perceber isso facilmente ao importar um novo Audio Clip. O caminho é o mesmo de sprites ou texturas! Basta arrastar para a pasta do Unity e ele carregará, já otimizando, o arquivo para o seu projeto.

O Unity também facilita bastante em relação aos tipos e formatos de áudio suportados. Pode-se utilizar sons mono, estéreo ou multifaixas de até oito canais! E esses arquivos podem ser importados em extensões diversas. Aceita-se .mp3, .ogg, .wav, .aiff / .aif, .mod, .it. s3m e .xm! Àqueles não muito curiosos sobre o tema "áudio", é importante saber que o Unity aceita mp3! Já ao pessoal mais envolvido na música, percebam serem aceitos formatos de faixas como .xm ou .it, permitindo, assim, trabalharmos diretamente no Unity. Vejamos, na **Figura 1**, a aba Inspector da faixa de trilha sonora, o qual importaremos ao nosso jogo!

Figura 01 - Inspector do áudio importado para trilha sonora do jogo.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de mar. de 2017

Como os outros recursos que já importamos, ao importar um arquivo de áudio no Unity, este permite, quando selecionamos o arquivo e observamos a aba Inspector, editar algumas de suas propriedades, bem como ter algumas pré-visualizações.

A primeira propriedade que vemos é a **Force To Mono**. Essa propriedade, se for habilitada, comprimirá os dois (ou mais) canais de áudio de um clipe qualquer e, então, irá transformá-los em um canal só, transformando o arquivo de áudio em mono.

Em seguida, temos a propriedade **Normalize**, a qual é habilitada somente se selecionarmos a opção de **Force To Mono**. Caso transformemos o áudio para mono, ela normalizará o sinal a partir de seus picos, para evitar que ele perca intensidade (volume).

Por fim, temos a propriedade **Load in Background**. Essa propriedade, se habilitada, fará o arquivo de áudio ser carregado em uma segunda thread de seu jogo, não parando, assim, a execução da thread principal. Qualquer áudio que for solicitado a tocar enquanto ainda estiver carregando terá essa chamada atrasada até ser carregado com sucesso!

A seguir, temos algumas propriedades que podem ser editadas para plataformas específicas. Não detalharemos essas propriedades, mas saiba onde encontrá-las, caso seja necessário quando você for compilar o seu jogo para uma outra plataforma!

Abaixo dessas plataformas, temos uma pequena caixinha indicando o tamanho do nosso áudio e o quanto ele foi comprimido pela importação padrão do Unity. Não alteraremos qualquer propriedade envolvendo a compressão do Unity, mas, caso seja de seu interesse, deixaremos o link, nas Leituras Complementares, para o material oficial que trata disso!

Por fim, temos um pequeno pré-visualizador para o nosso arquivo de áudio. Ele é capaz de reproduzir o arquivo uma vez ou em loop, além de trazer mais algumas informações sobre o áudio. Para ouvi-lo, basta clicar no botão de play!

1.3 Os Audio Sources

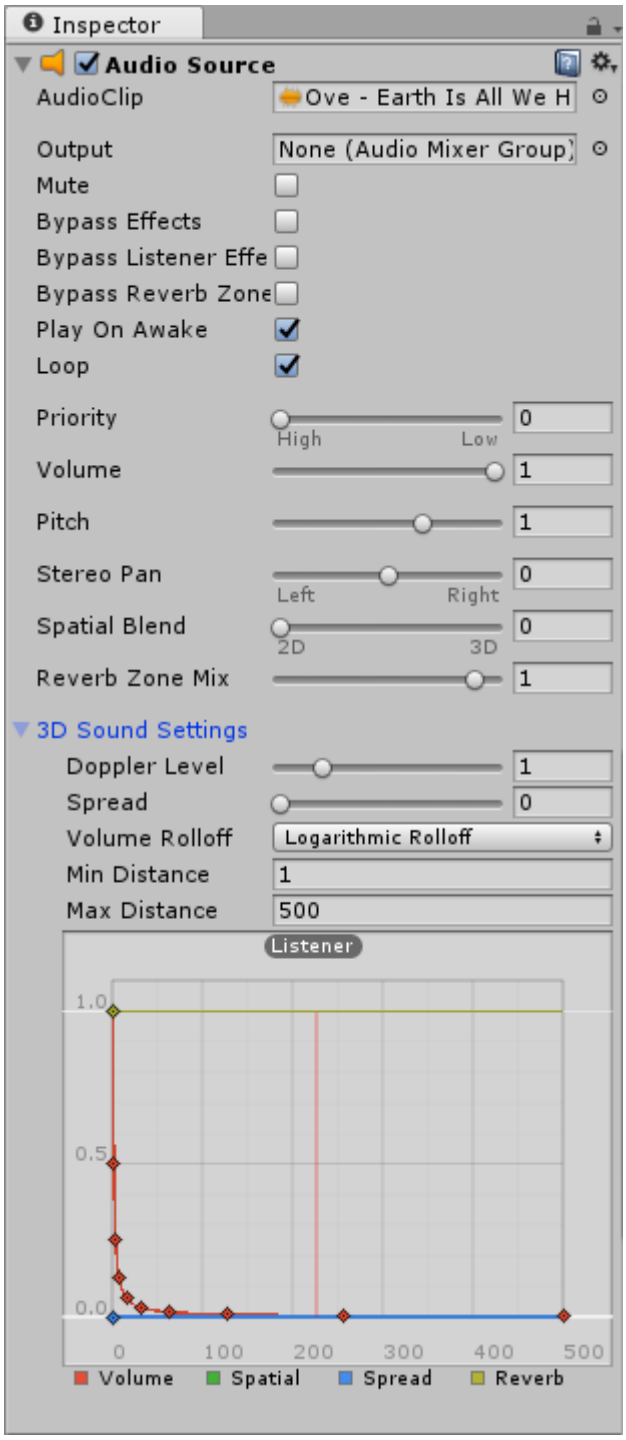
Por último, temos os Audio Sources. Esses componentes são responsáveis por, de fato, adicionar sons ao nosso jogo. Eles deverão ser adicionados a um GameObject qualquer e utilizarão um **Audio Clip** já importado anteriormente para, usando as propriedades desse componente, tocar um som na cena. Esse som, caso esteja na área do Audio Listener, será captado e, então, redirecionado à saída de áudio do jogador.

Mas como assim, “caso esteja na área do Audio Listener”? Como dissemos anteriormente, o Unity trabalha com o áudio em 3D! Desse modo, ele consegue posicionar o áudio na cena de maneira bem precisa, fazendo o usuário só escutar o que deve escutar. Pensemos em um exemplo.

Adicionamos à nossa cena, em aulas anteriores, uma bazuca que atira mísseis. Já imaginou se toda a cena fosse capaz de ouvir os mísseis sendo lançados? Seria um tanto chato, não? E também seria informação desnecessária! Então, na aula de hoje, ao adicionarmos som à nossa bazuca, podemos configurá-la para que ele só seja tocado em uma certa área, sumindo quando a distância aumentar após esse limite. Assim, o som serve até como pista da aproximação da bazuca. Legal, não?

E o que mais esse componente Audio Source pode fazer? Bem... Bastante coisa! Vejamos, na **Figura 2**, as suas propriedades, utilizando, mais uma vez, nossa trilha sonora como exemplo.

Figura 02 - Propriedades do componente Audio Source responsável pela trilha sonora do jogo.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de mar. de 2017

Como vimos na **Figura 2**, a primeira propriedade do componente Audio Source é o **Audio Clip** que esse Audio Source será responsável por executar. Por isso falamos anteriormente sobre Audio Clips! Em seguida, temos a propriedade **Output**, a qual indica se haverá algum grupo de mixagem a ser utilizado a fim de mixar a

faixa para esse Audio Source específico. Não detalharemos os grupos de mixagem, pois precisaríamos entender mais sobre música para falar a respeito desse assunto. Mas fica a referência!

Logo após, temos a propriedade **Mute**. Essa propriedade, quando selecionada, funciona como se o volume tivesse sido alterado para 0, fazendo a faixa continuar sendo executada, mas sem emitir qualquer som. Em seguida, há três propriedades de **Bypass**.

Bypass significa que a Audio Source ignorará componentes de um determinado tipo e continuará a executar normalmente, embora seja indicado o contrário. Se adicionarmos, por exemplo, uma zona de reverberação onde esse áudio deverá ter eco adicionado a ele e marcarmos a opção de **Bypass Reverb Zones**, o áudio passará pela zona da mesma maneira como se ela não existisse.

A próxima propriedade é a **Play on Awake**. Se marcada, ela indica que o áudio deverá ser tocado no mesmo momento no qual o objeto que o contém for criado. É equivalente a criarmos um script com a função Play() para esse áudio no método Awake(). A propriedade **Loop**, a seguir, indica se o áudio deve ser tocado em **Loop** ou não.

A seguir, temos algumas propriedades que obedecem a uma faixa de valor e podem ser ajustadas por sliders. A primeira delas é a propriedade **Priority**. Ela indica a prioridade do áudio em relação aos outros, sendo o menor áudio o que tem maior prioridade. Como na **Figura 2** utilizamos um print de nossa trilha sonora, essa propriedade está configurada para 0, por exemplo, indicando ser provavelmente esse o áudio de maior prioridade.

A propriedade **Volume** indica o volume em que a faixa deve ser tocada, obedecendo a uma escala de decibéis, que é uma escala logarítmica (lembra das aulas de matemática?). Isso é interessante de lembrar ao lidarmos com esse valor, para sabermos, por exemplo, que o valor 0.5 não é bem a metade, em intensidade, do valor 1.0.

A propriedade **Pitch** indica a velocidade na qual a faixa será executada, sendo o valor 1 o padrão. Valores menores que esse fazem a faixa ficar mais lenta; valores maiores, por sua vez, a fazem ficar mais rápida. Caso utilize um valor negativo, a faixa será executada ao contrário.

Stereo Pan indica o balanço no qual o áudio será executado, entre o canal da esquerda e o da direita. Valores positivos indicam que será tocado na saída direita, valores negativos, na esquerda. O zero indica o balanço idêntico para os dois lados.

Por fim, a propriedade **Spatial Blend** indica quanto utilizaremos da espacialização 3D em nosso áudio. Caso deixemos a propriedade **Spatial Blend** apenas para 2D, o áudio ignorará qualquer conceito de posição e será executado em toda a cena. Caso utilizemos totalmente 3D, o áudio será de acordo com o posicionamento do Listener em relação ao Source. Valores intermediários são uma mixagem desses casos.

A aba de 3D Settings contém as propriedades do som 3D. Detalharemos mais esse menu na seção seguinte, quando precisarmos modificar essas propriedades para posicionar melhor o som de nossos mísseis.

Ufa! Muitas propriedades para o Audio Source, não? Por isso que o Audio Listener pode ficar tranquilo, sem nenhuma configuração. Cada áudio cuida de si! E, então, quando começarem a ser emitidos, sobra para o Listener apenas a função de captar tudo e passar para os falantes. Bom, não é?

> Se tiver ficado alguma dúvida em relação à essas propriedades, você pode verificar o manual oficial que contém uma descrição detalhada disso tudo e está lá na seção de Leitura Complementar. Utilize, também, os fóruns para resolver qualquer problema encontrado!

2. Adicionando Áudio ao Projeto

Agora que estudamos bem o sistema de áudio do Unity, podemos utilizar nossos novos conhecimentos para adicionar áudios ao nosso projeto! Infelizmente, não temos como entrar em detalhes e adicionar todos os elementos de áudio que gostaríamos ainda nesta aula, em razão de nosso tempo ser limitado, mas aproveitaremos esse momento muito bem! Adicionaremos uma trilha sonora ao

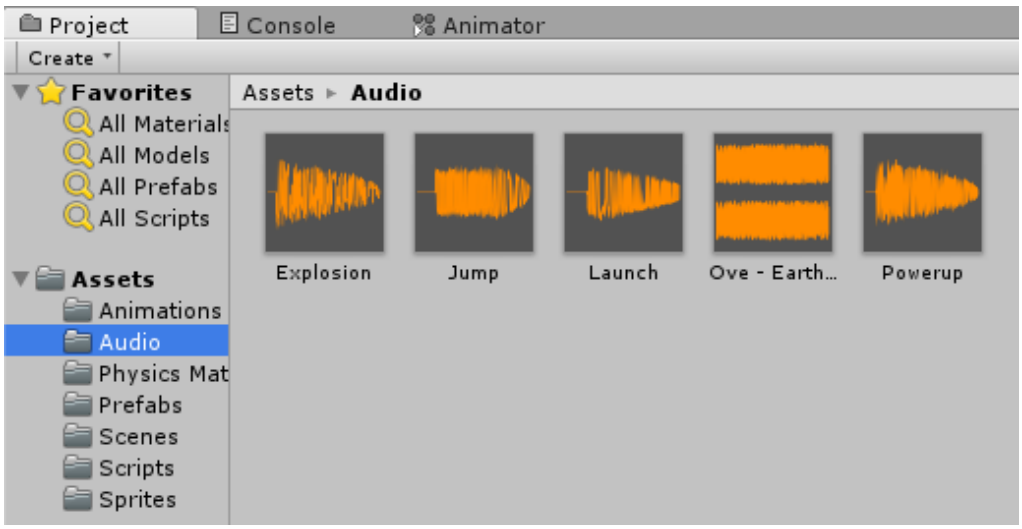
nosso jogo, além de efeitos sonoros ao pulo, ao tiro da bazuca e à explosão do míssil! Disponibilizaremos, também, um som de Power Up, para quem quiser se aventurar um pouco mais. O pacote completo de sons pode ser obtido aqui.

E de onde vem todos esses sons que estamos disponibilizando? Alguma pista? Lembram da aula de assets? De nosso background? Isso! O site opengameart.org, que conhecemos em aulas anteriores, também tem músicas e efeitos sonoros disponíveis para podermos baixar e utilizar sem qualquer problema! Na verdade, os sons a serem utilizados são todos de lá! A música foi escrita, produzida e é uma cortesia de Ove Melaa, enquanto os efeitos sonoros são todos desenvolvidos e disponibilizados no OpenGameArt.org pelo Mark McCorkle! Ambos são disponibilizados através da licença Creative Commons 3.0, discutida em aulas anteriores e possível de ser encontrada na referência. Muito bacana! Podemos usar tranquilamente! :)

Então, agora que já temos os sons, podemos começar a implementação deles em nossa cena! Faremos algumas modificações para isso, no entanto. Para quem ainda não tem o projeto, ele pode ser obtido aqui.

Após baixar os assets e abrir o projeto, basta criar uma nova pasta chamada Audio e importar para ela todos os arquivos contidos no pacote disponibilizado. Não será necessário realizar qualquer configuração nos arquivos, pois eles já serão importados do modo como os utilizaremos! Nossa pasta Audio ficará como vista na **Figura 3**.

Figura 03 - Pasta Audio, contendo todos os assets de áudio importados.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de mar. de 2017

2.1 Adicionando a Trilha Sonora ao Projeto

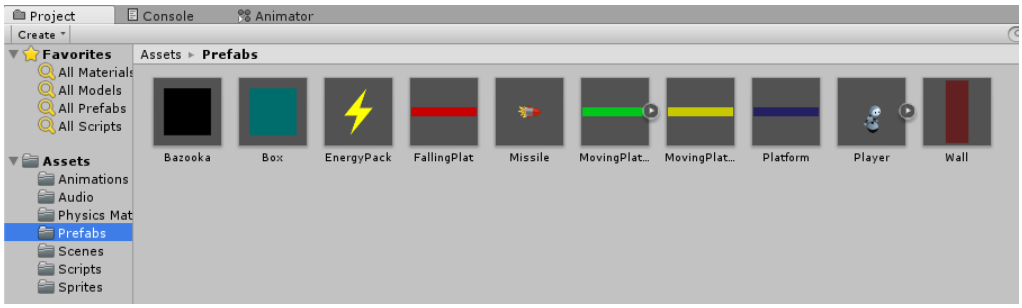
A primeira nova adição ao nosso projeto será a trilha sonora. Para que ela fique sempre tocando, a adicionaremos como um componente a um objeto novo, o qual será adicionado com o propósito exclusivo de prover a música à cena. Faremos isso para cada cena poder ter a sua música.

No entanto, antes de adicionarmos esse novo componente, faremos uma modificação em nosso projeto. Como o Player estará viajando por várias fases e será sempre o mesmo, podemos transformá-lo em um Prefab! Assim, ao adicionarmos um novo nível ao nosso cenário, podemos simplesmente importar o Player que já estamos utilizando, pois quem carrega as informações entre as cenas é o Game Controller e não o próprio Player. Para isso, precisamos de dois passos. Abra a cena Level_1 e arraste o Player à nossa pasta Prefabs, criando o Prefab do Player. Em seguida, salve a cena atual e abra a cena Level_2. Delete o Player de nosso segundo nível e importe o Prefab. Isso fará com que possamos alterar uma vez só o Player para as duas cenas! Não esqueça de reposicioná-lo em seu local inicial, o (-225, 10, 1).

E, como estamos lidando com Prefabs, que tal transformar também a nossa bazuca e o nosso pacote de energia em Prefabs, em razão de eles poderem ser componentes interessantes para replicarmos em outros momentos? Façam isso, arrastando-os da cena para a pasta Prefabs, como já vimos diversas vezes.

Com isso, nossa pasta Prefab deve estar como vista na **Figura 4**.

Figura 04 - Pasta Audio, contendo todos os assets de áudio importados.

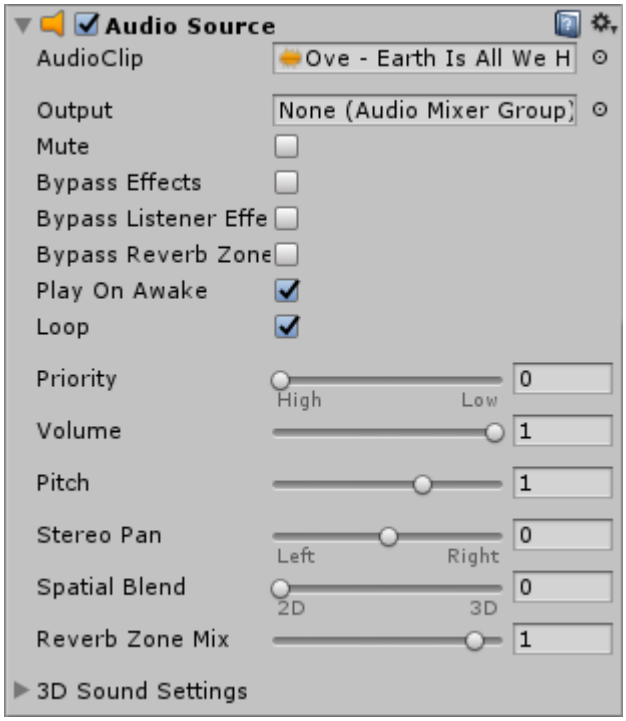


Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de mar. de 2017

Agora, com todos os Prefabs definidos, adicionaremos o nosso elemento de música a cada uma de nossas cenas. Na cena atual, adicione um novo GameObject vazio, o renomeie para BGMusic e em seguida, com o objeto selecionado, selecione, em nosso Inspector, o botão Add Component -> Audio -> Audio Source. Ao fazer isso, um Audio Source vazio será criado para o objeto.

A primeira configuração necessária corresponde a alterar o Audio Clip, selecionando a nossa trilha sonora Ove - Earth Is All We Have. Fique à vontade para escolher uma outra trilha, caso queira! Em seguida, precisamos alterar mais três propriedades do Audio Source. Primeiro, ativaremos o Play on Awake e o Loop, para a música iniciar juntamente à fase e tocar enquanto esse objeto existir. Por fim, alteraremos a prioridade para 0, a fim de permitir que a trilha sonora tenha prioridade sobre os outros efeitos. O Audio Source configurado deve ficar como visto na **Figura 5**.

Figura 05 - Audio Source de nossa trilha sonora, adicionado a um GameObject



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de mar. de 2017

Repita esse processo para cada uma de nossas cenas! Com isso, já podemos dar o Play em nosso jogo e vermos que a música já está tocando, em Loop, desde o momento no qual o jogo é iniciado até ele se encerrar, seja devido à quebra do personagem ou à chegada ao fim do nível. Em ambos os casos, a música é reiniciada

quando o novo nível é iniciado, seja ele o mesmo ou uma nova fase. Perceba: se desejarmos, podemos alterar a música do segundo nível para uma diferente, a da tela de Game Over para outra, e assim por diante. Basta alterarmos o objeto novo que estamos adicionando às nossas cenas!

2.2 Adicionando os Efeitos Sonoros

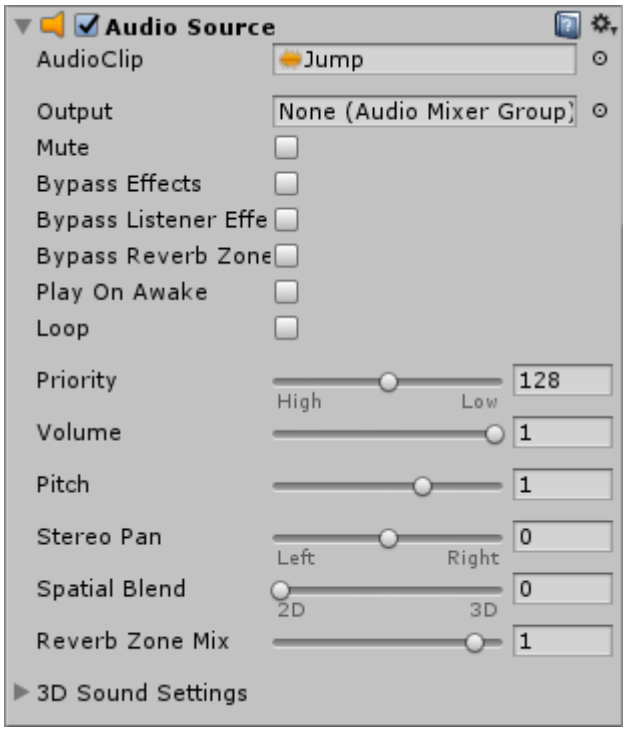
Adicionaremos três efeitos sonoros ao nosso jogo – um ao pulo do personagem, um à bazuca e um ao míssil. Como já dissemos anteriormente, os dois últimos utilizarão os efeitos de som 3D, então, começaremos pelo pulo do personagem por ser o mais simples.

2.2.1 Adicionando Som ao Pulo

Para adicionarmos um som ao personagem, precisamos, antes de tudo, adicionar a ele um Audio Source e configurar o Audio Clip para ser o áudio de Jump que temos em nossa pasta Audios. Para fazer isso, basta, como já vimos, selecionar o nosso Player Prefab criado anteriormente e, então, clicar no botão Add Component -> Audio -> Audio Source. Isso criará um Audio Source vazio. Para a propriedade Audio Clip, selecionaremos o valor Jump, o qual é o nome do nosso arquivo de áudio que representará o pulo.

Em seguida, ao contrário do realizado para a nossa música de fundo, desmarcaremos a propriedade Play on Awake e a propriedade Loop, indicando que esse som deve ser tocado apenas uma vez, somente quando for solicitado. O resultado final pode ser visto na **Figura 6**.

Figura 06 - Audio Source de pulo, adicionado ao personagem.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de mar. de 2017

Ok. Adicionamos o áudio ao personagem. Mas como faremos esse áudio tocar? É simples! Basta alterar o script para ter uma referência ao nosso Audio Source e, logo após, tocar o Audio Source toda vez que o personagem for executar um pulo. Criaremos uma variável, do tipo AudioSource, de nome jumpAudio para isso e, então, no método FixedUpdate, por meio do qual executamos o pulo, adicionaremos uma ordem para executar também o áudio. O código alterado para o método ficará como visto na **Listagem 1**.

```
1  public AudioSource jumpAudio;
2
3  //Called in fixed time intervals, frame rate independent
4  void FixedUpdate() {
5     if (active) {
6        float moveH = Input.GetAxis ("Horizontal");
7
8        ani.SetFloat("speed", Mathf.Abs(moveH));
9
10       grounded = Physics2D.OverlapBox (groundCheck.position, (new Vector2 (1.3f, 0.2f)), 0f, layerMa
11
12       ani.SetBool("grounded", grounded);
13       ani.SetFloat("vertSpeed", rigidBody.velocity.y);
14
15       if (moveH < 0 && movingRight) {
16          Flip();
17       } else if (moveH > 0 && !movingRight) {
18          Flip();
19       }
20
21       rigidBody.velocity = new Vector3 (maxSpeed * moveH, rigidBody.velocity.y, 0);
22
23       if (jumping) {
24          rigidBody.AddForce(new Vector2(0f, jumpSpeed));
25          jumping = false;
26          jumpAudio.Play();
27       }
28       if (doubleJumping) {
29          rigidBody.velocity = new Vector2 (rigidBody.velocity.x, 0);
30          rigidBody.AddForce(new Vector2(0f, jumpSpeed));
31          doubleJumping = false;
32          jumpAudio.Play();
33       }
34    }
35 }
```

Listagem 1 - Código atualizado do método FixedUpdate contendo, em negrito, as linhas de execução de áudio.
Fonte: Elaborada pelo autor

Perceba que, para esse código funcionar, é necessário declarar a variável no início de nosso script, como exemplificado na linha 1. Além disso, é necessário, no Editor, definir que o Audio Source ao qual estamos nos referindo para a nossa variável pública é o Audio Source do próprio Player. Podemos fazer isso clicando e arrastando um componente até o outro!

Após feitas as alterações, o nosso personagem já estará tocando um som ao pular, em conjunto com a música de fundo. O vídeo a seguir demonstra o resultado.

Video 01 - Resultado

2.2.2 Adicionando Som à Bazuca

Adicionado o efeito sonoro ao pulo, o próximo efeito será o da nossa bazuca. Tocaremos o som de tiro toda vez que criarmos um novo míssil. Precisaremos, no entanto, editar o nosso componente de som, para esse som não se propagar por todo o nível.

O primeiro passo continua o mesmo! Selecionamos o nosso Prefab da bazuca recém-criado e, então, adicionamos a ele, pelo botão Add Component -> Audio -> Audio Source, um novo Audio Source. Selecionamos o clip de áudio Launch para esse componente. Da mesma maneira, retiramos o Play on Awake e o Loop, pois o áudio será executado uma vez e sob demanda. A diferença, no entanto, estará na propriedade Spatial Blend e, consequentemente, no menu 3D Sound Settings.

Como estaremos utilizando um som 3D, embora apenas em duas dimensões, precisamos alterar a propriedade Spatial Blend para 1, fazendo o slider mover-se totalmente para o lado 3D. Isso indica que nós trabalharemos com os efeitos definidos no menu exibido abaixo dessa propriedade, o 3D Sound Settings.

Esse menu possui cinco propriedades, as quais geram um gráfico que está logo abaixo delas. A primeira propriedade indica o nível do efeito Doppler, sendo esse efeito adicionado ao nosso componente em caso de movimentos rápidos. Para quem não lembra bem do efeito Doppler, pode dar uma olhada neste link.

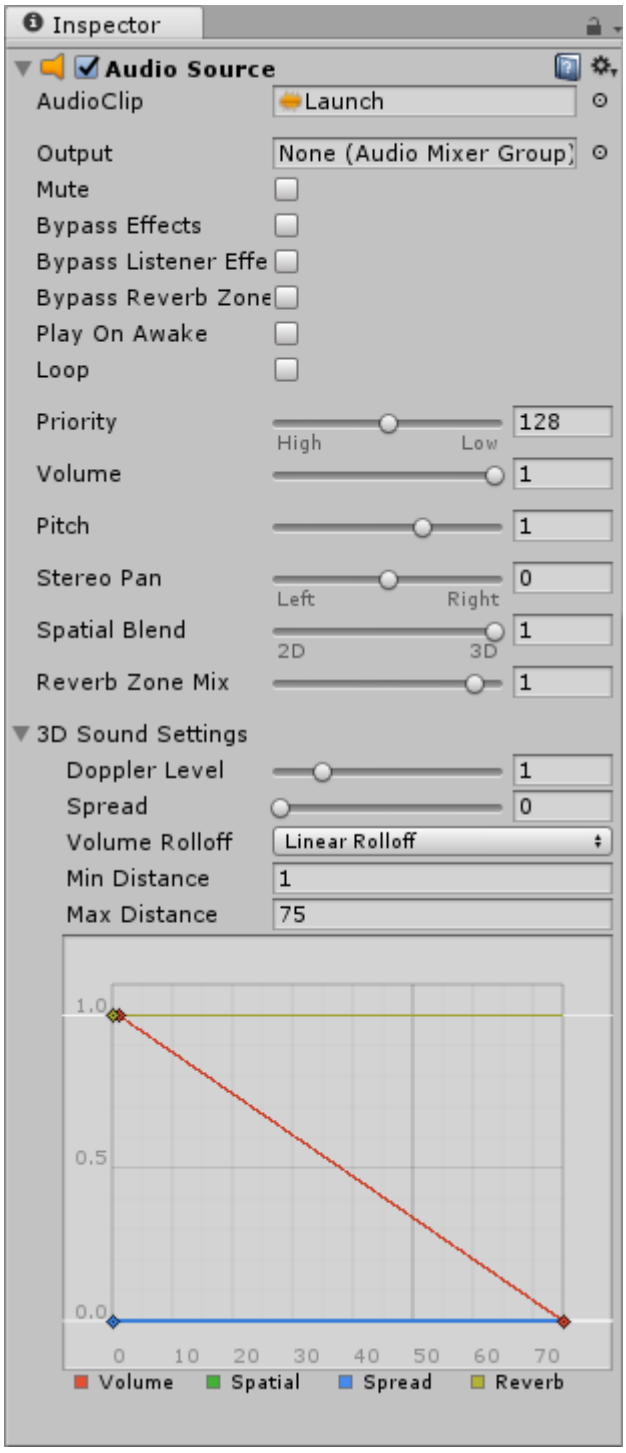
A segunda propriedade diz respeito ao espalhamento do som ao seu redor. É a propriedade Spread. Não utilizaremos qualquer espalhamento, nem entraremos nesses detalhes. Basta colocar o valor para 0.

A terceira propriedade, a Volume Roloff, indica qual modelo de gráfico será utilizado para o volume se alterar conforme a distância. Se mantivermos o Logarithmic Rolloff, que é o padrão, o som se alterará, em relação ao volume, com

um comportamento logarítmico, como é apresentado no gráfico abaixo da propriedade 3D Sound Settings. Utilizaremos, no entanto, para simplificar as coisas, o decaimento linear, ou Linear Rolloff. Ao selecionarmos esse novo valor para a propriedade, o gráfico se alterará para uma reta, indicando o nosso novo comportamento do volume. Para finalizar, precisamos apenas reduzir a área de efeito desse som. Faremos isso alterando a Min Distance para 1 e a Max Distance para 75, indicando que o efeito sonoro começará a ser efetivo a partir da distância de 75 metros.

Ao fim das configurações, o Audio Source adicionado ao Prefab Bazooka deverá estar como o da **Figura 7**.

Figura 07 - Audio Source adicionado ao Prefab Bazooka.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de mar. de 2017

Com o componente configurado, basta modificarmos o script para tocar o áudio sempre que um novo míssil for gerado. Isso pode ser feito adicionando as linhas em negrito, da **Listagem 2**. Perceba utilizarmos uma outra maneira para detectar o Audio Source, diferente da variável pública criada para o Player.

```
using UnityEngine;
using System.Collections;

public class ShooterScript : MonoBehaviour {

    public GameObject missilePrefab;

    private float speed = 12f;
    private Vector3 spawningPos;
    private bool active = true;
    private AudioSource shootingAudio;

    // Use this for initialization
    void Start () {
        shootingAudio = GetComponent<AudioSource>();
        spawningPos = new Vector3(transform.position.x+transform.localScale.x, -4.5f, transform.posit
        Spawn ();
    }

    void Spawn() {
        if (active) {
            GameObject missile = GameObject.Instantiate (missilePrefab) as GameObject;

            missile.transform.position = spawningPos;
            missile.transform.parent = transform;

            missile.gameObject.GetComponent<MissileController> ().speed = speed;
            shootingAudio.Play();

            Invoke ("Spawn", Random.Range (1f, 1.5f));
        }
    }

    public void Stop() {
        BroadcastMessage("StopMissile");
        active = false;
    }
}
```

Listagem 2 - Código alterado do ShooterScript para adicionar áudio à bazuca.
Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em mar. 2017

Com isso, a bazuca já estará executando o áudio, em uma região delimitada, todas as vezes que gerar um novo míssil. O resultado disso pode ser visto no vídeo a seguir. Note como o gráfico de volume (adicionado no canto inferior esquerdo do vídeo) indica a posição do Audio Listener em relação à fonte de áudio, através de uma linha com o nome Listener, quando estamos com o jogo sendo executado. Perceba que, quando a linha sai do limite de áudio, o som desaparece e, quando ela se aproxima, o volume aumenta!

Video 02 - Resultado

2.2.3 Adicionando Som aos Mísseis

O último elemento ao qual adicionaremos som na aula de hoje, marcando a última alteração a ser feita, até o momento, em nosso projeto, é o Prefab Missile. Os mísseis terão o mesmo comportamento sonoro da bazuca, ativando-se apenas em uma área e evitando que as explosões sejam ouvidas de toda a fase.

Por essa razão, o procedimento de adição de uma nova Audio Source realizado nesse componente será exatamente igual ao anterior. A única diferença é que o Audio Clip selecionado será o Explosion, em vez do Launch. A configuração final ficará igual à vista na **Figura 7**, apenas com essa alteração, incluindo as mudanças no som 3D.

Adicionado o componente, precisamos apenas alterar o script para garantir que o áudio será tocado adequadamente. O novo script pode ser visto na **Listagem 3**, a seguir, com as linhas alteradas em negrito.

```
using UnityEngine;
using System.Collections;

public class MissileController : MonoBehaviour {

    public float speed;
    public float damage;

    private Animator ani;
    private AudioSource explosionAudio;

    // Use this for initialization
    void Start () {
        ani = gameObject.GetComponent<Animator>();
        explosionAudio = gameObject.GetComponent<AudioSource>();
    }

    // Update is called once per frame
    void Update () {
        transform.GetComponent<Rigidbody2D>().velocity = Vector2.right * speed;
    }

    void OnTriggerEnter2D (Collider2D col) {
        ani.Play("Exploding");
        explosionAudio.Play();
        transform.GetComponent<Rigidbody2D>().simulated = false;

        if (col.CompareTag ("Player")) {
            if (col.gameObject.GetComponent<PlayerController>().HealthChange(-damage) <= 0)
                transform.parent.GetComponent<ShooterScript>().Stop();
        }

        Invoke("Remove", 1f);
    }

    void StopMissile () {
        transform.GetComponent<Rigidbody2D>().simulated = false;
    }

    void Remove () {
        Destroy(gameObject);
    }

}
```

Listagem 3 - Script MissileController alterado para guardar as mudanças de áudio.
Fonte: Elaborada pelo autor

Com isso, teremos a nossa explosão também com um som, funcionando de maneira bastante similar à nossa bazuca. O resultado de todas as adições pode ser visto no vídeo a seguir.

Video 03 - Resultado de Todas as Adições

E, assim, concluímos as alterações do nosso projeto, adicionando a ele, por último, áudio! Como combinamos em nossa primeira aula, chegamos ao final de nossa disciplina com um jogo completo!

Ainda faltam alguns elementos, claro, mas podemos construir todos eles com base no que vimos ao longo dessa nossa jornada de quinze aulas, a qual está quase chegando ao fim... :~

Mas eu disse quase! Ainda temos um último ponto para ver, antes de encerrar tudo e partir para Desenvolvimento com Motores de Jogos II, disciplina na qual adicionaremos uma nova dimensão à maneira como encaramos os jogos (literalmente :P). Como será que distribuiremos esse joguinho criado?

Hey! Lembrando da importância de você compartilhar nos fóruns ou durante os encontros presenciais as suas dúvidas. Quem sabe se aquele recurso que você está tentando inserir e não está conseguindo já foi feito pelo seu colega de sala? Vai lá, aproveita e esclarece suas dúvidas!

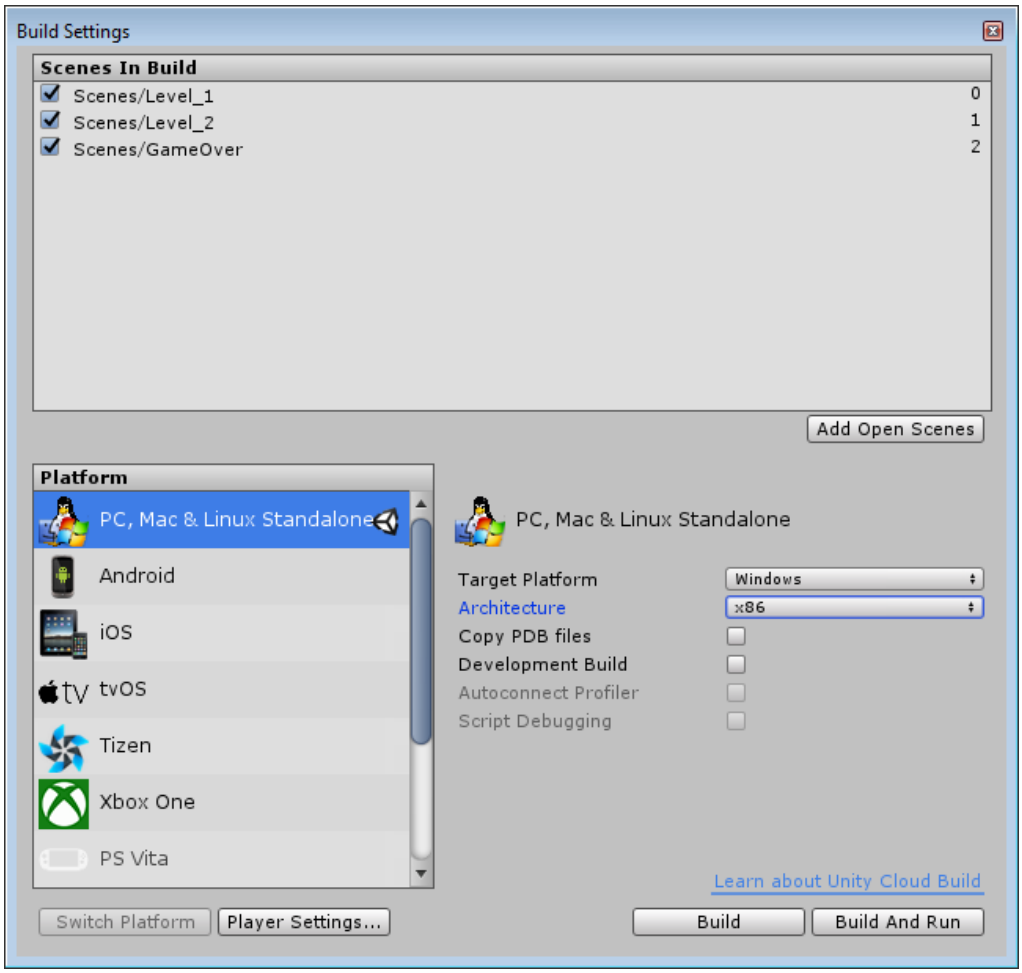
3. Compilando o Projeto para Distribuição

Para finalizar a nossa disciplina, veremos como é simples compilar um jogo a fim de ele poder ser distribuído como um standalone, ou seja, uma versão que roda independente do Unity, em qualquer máquina na qual haja as configurações necessárias para tal.

E o melhor de tudo! É muito simples!

Lembra de, algumas aulas atrás, quando falamos sobre a mudança de cenas, dizermos que precisaríamos adicionar as cenas trabalhadas ao Build Settings (Ctrl+Shift+B ou File->Build Settings)? Pois é! Precisamos fazer apenas isso para compilar o nosso jogo para Windows e o distribuir a qualquer outra pessoa que também possua uma máquina rodando Windows. Vejamos a **Figura 8**.

Figura 08 - Build Settings de nosso projeto com as cenas adicionadas, como fizemos anteriormente.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/. Acesso em: 23 de mar. de 2017

A **Figura 8** nos mostra o Build Settings já configurado com as cenas de nosso projeto. Com isso, podemos adicionar mais algumas configurações, caso desejemos, através do botão Player Settings, ou até mesmo modificar algo dentre as opções mostradas.

Target Platform indica para qual das plataformas nós compilaremos. O Unity suporta Mac, Windows e Linux! Utilizaremos o Windows, no entanto, uma vez que estamos trabalhando no Windows. Em Architecture, é possível escolher se trabalharemos com 32 ou 64 bits. Utilizaremos 32 bits para facilitar a compatibilidade.

As outras opções servem para ativarmos o Debug de nosso jogo, se quisermos rodar a versão final de nosso projeto em modo de desenvolvimento. Como não é esse o nosso objetivo, uma vez que já testamos tudo em nosso Editor, deixaremos essas opções desmarcadas.

Feito isso, basta clicarmos no botão Build, no canto inferior direito. Isso abrirá uma janela para escolhermos onde queremos salvar o nosso jogo. Vá até a pasta de Documentos, crie uma nova pasta lá, com o nome do seu jogo, e salve o projeto dentro, como ProjetoDMJI.exe. Ao selecionar essa opção, o Unity fará um Build de seu jogo e colocará, nessa pasta, juntamente ao executável gerado do jogo, todos os arquivos necessários para executá-lo.

Ao concluir o processo de build (pode demorar um pouco), o Unity abrirá a nova pasta, contendo, além do jogo em si, uma pasta chamada ProjetoDMJI_Data, a qual contém todos os dados necessários para que o jogo seja executado.

Pronto! Basta dar dois cliques em ProjetoDMJI.exe e o seu jogo estará executando, ao vivo, independentemente do Unity! E assim chegamos ao fim de nossa jornada! Uhuuuul \o/

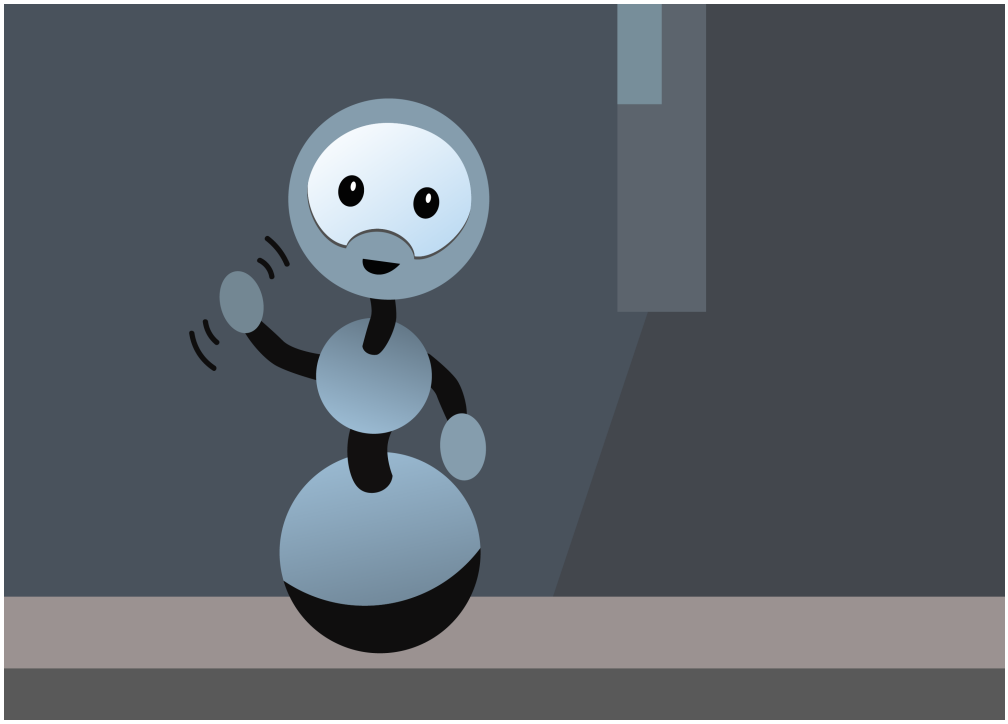
Conseguimos partir de uma tela vazia, no Unity, e terminar com um jogo completinho, bem divertido e capaz de ser executado em qualquer computador Windows, sem a necessidade de ter sequer o Unity instalado.

Muito massa, não?

Espero que vocês tenham gostado dessa jornada tanto quanto eu e que possam sempre utilizar diversos dos conteúdos aqui apresentados.

Foi uma experiência única! Agora, vão todos para o fórum deixar aquela mensagem de despedida, com as últimas dúvidas que surgirem.

Foi um prazer! Até a próxima!!! \o



Leitura Complementar

Documentação oficial do Unity acerca do áudio - https://docs.unity3d.com/Manual/Audio.html

Documentação oficial do Unity acerca da publicação de jogos - https://docs.unity3d.com/Manual/PublishingBuilds.html

Resumo

Na aula de hoje, a última da disciplina, aprendemos a adicionar som ao nosso jogo, concluindo o último aspecto o qual faltava nele. Conhecemos todo o sistema de som do Unity, por meio da utilização do Audio Listener, que capta o áudio gerado e o reproduz ao jogador, do Audio Clip, que representa os arquivos de áudio em nosso projeto, e do Audio Source, que emite de fato o áudio em nossa cena!

Com esses componentes, adicionamos som à nossa cena na forma de música de background, além de efeitos sonoros de pulo, tiro e explosão. Conseguimos esses efeitos através do OpenGameArt.org e os adicionamos utilizando os conhecimentos adquiridos do sistema de som do Unity.

Para encerrar a aula e a disciplina, vimos o modo de compilar o nosso jogo e distribuí-lo como um standalone, o qual funciona independente do Unity instalado e em qualquer máquina que tenha o Windows instalado. Assim, além de desenvolvermos um jogo bem completo, pudemos passá-lo adiante, aos nossos colegas! Para quem quiser baixar a versão final, ela está disponível aqui!

Foi um prazer estar com vocês ao longo desta disciplina. Desejo que Desenvolvimento com Motores II seja uma experiência interessante para todos! Boa sorte e até mais!

Autoavaliação

1. Para que serve um Audio Listener?
2. Para que serve um Audio Source?
3. Como um Audio Clip interage com um Audio Source?
4. O que precisamos adicionar ao Build Settings para compilar uma versão simples, standalone, de nosso jogo?
5. Qual o botão do Build Settings que nos permite realizar configurações adicionais na versão final do jogo?

Referências

Documentação oficial do Unity - Disponível em: https://docs.unity3d.com/Manual/index.html.

Tutoriais oficiais do Unity - Disponível em: https://unity3d.com/pt/learn/tutorials.

RABIN, Steve. **Introdução ao Desenvolvimento de Games**, Vol 2. CENGAGE.