Desenvolvimento com Motores de Jogos I

Aula 07 - Animação de Sprites – Parte I

Apresentação

E aí, players! Tudo certo? Chegamos à sétima aula da disciplina de Desenvolvimento com Motores de Jogos I! Já aprendemos diversas coisas que nos permitem fazer um jogo bem divertido. No entanto, ainda não estudamos algo comum a quase todos os jogos já lançados até hoje: a animação!

Se voltarmos lá para os primórdios dos jogos digitais, conseguiremos ver alguns exemplos de jogos que funcionavam sem qualquer animação. Depois disso, porém, encontramos facilmente jogos com tal animação presente no personagem, estando, posteriormente, em vários outros elementos do jogo, até atingir o cenário e o background.

Na aula de hoje, iniciaremos os estudos sobre a animação de nosso personagem! Digo iniciar, porque esse é um assunto muito extenso, então utilizaremos duas aulas para abordar mais detalhes sobre ele. E ainda faltará muita coisa, principalmente relacionada à animação 3D! Deixaremos isso para uma próxima disciplina, ok?

Nesta aula, adicionaremos uma animação ao nosso personagem para quando ele não estiver se movendo, começaremos a preparar uma animação para sua caminhada e, por fim, uma animação bem especial para quando ele estiver pulando! Com esses três casos veremos ser possível cobrir a maioria das situações vivenciadas pelo personagem. Começaremos hoje o primeiro caso e concluiremos os outros dois em nossa próxima aula!

Como estamos sempre lidando com 2D em nossa disciplina, focaremos também nas animações 2D e em como elas funcionam e podem ser adicionadas no Unity.

Esta é uma aula bem interessante, mas também cheia de detalhes importantes que poderão passar despercebidos durante uma primeira leitura. Muito cuidado, principalmente porque diversas coisas serão utilizadas na próxima aula! Dedique-se bastante e vamos juntos fazer o nosso amigo robô finalmente começar a ganhar vida! Animados? :P

Objetivos

Ao final desta aula, você deverá ser capaz de:

- Importar e utilizar Sprite Sheets no Unity;
- Adicionar animações aos sprites de seu jogo através de um Animator;
- Criar animações a partir de Sprite Sheets importadas.

1. A Animação em Jogos 2D

A animação é, há muito tempo, um componente importante de qualquer jogo digital desenvolvido, independentemente da plataforma. Mesmo em jogos mais antigos, como Donkey Kong e Dr. Mario para o NES, os objetos envolvidos já possuíam animações, e parte do limitadíssimo poder de processamento dessas máquinas era disponibilizado para a nossa utilização.

Figura 01 - Donkey Kong clássico, com suas diversas animações.

Fonte: Jogo Donkey Kong do NES by Nintendo

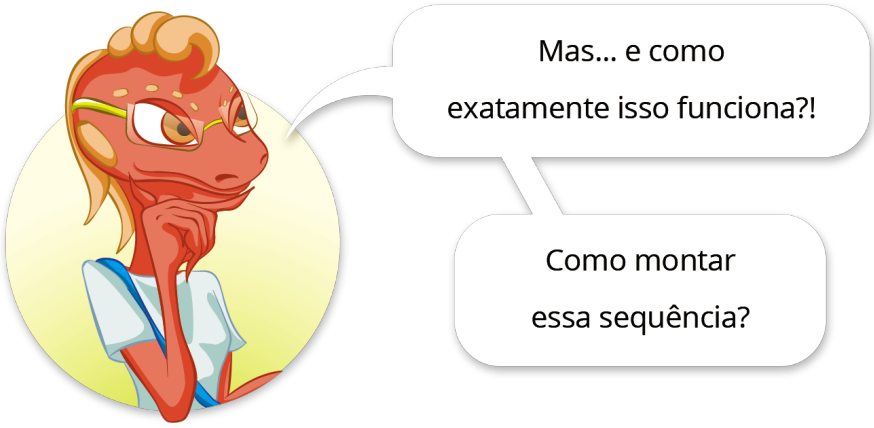
Adicionar animações a um personagem ou objeto qualquer é parte do que faz o jogo ganhar vida. Mesmo as animações sendo as mais simples possíveis, como vemos na **Figura 1**, só o fato de elas existirem já dão uma dimensão diferente ao jogo que estamos jogando.

Se olharmos o nosso jogo, imediatamente percebemos que isso é um dos principais pontos faltando até o momento. O nosso robô se move para um lado e para o outro, e até pula, mas nunca se altera de sua forma inicial. Veremos, ao longo desta aula, como o nosso projeto parecerá diferente após adicionarmos algumas dessas alterações. Depois, conheceremos as demais possibilidades de mudanças na próxima aula.

E você já se questionou como funcionam essas animações tão importantes no desenvolvimento de um jogo? Não? Então, pense um pouco. Como você acha que funcionam as coisas para termos essa ideia de movimento?

A primeira coisa que pensei foi: para a animação funcionar, precisamos de uma sequência de frames. E é justamente esse o ponto de partida de qualquer animação.

Uma animação é, basicamente, um conjunto de frames, os quais, ao serem postos em sequência, dão a ideia de determinado objeto estar realizando certo tipo de ação factível, usualmente baseada em alguma movimentação. E é precisamente isso que trabalharemos dentro do Unity!



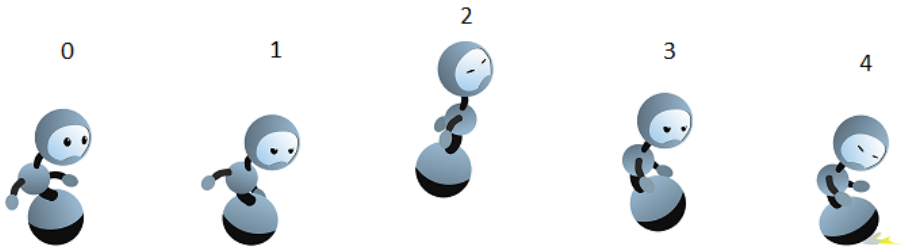
2. Animação Através de *Keyframes*

O Unity utiliza um conceito muito comum para lidar com animações em 2D ou 3D: os *keyframes*. Eles são a definição de pontos específicos, dentro de uma linha do tempo, ou *timeline*, da qual determinado objeto faz parte. Digamos, por exemplo, que queremos animar um personagem para pular. Como faríamos?

Não é nada complicado! Basta pensar como seria esse pulo ao longo do tempo e reproduzir isso na *timeline* através da utilização de *keyframes*. Vamos pensar juntos. Para pular, eu devo começar, no tempo T = 0, um movimento de pulo. No tempo T = 1, logo depois, eu devo me impulsionar para cima. Já no tempo T = 2, eu estarei no ar! Em seguida, no T = 3, estarei me preparando para aterrissar e, por fim, no T = 4, estarei de volta ao chão. Perceba que não definimos aqui a unidade desse tempo T,

a qual pode ser segundos, frames ou qualquer outra coisa. Também não entramos em detalhes de como cada um desses frames é representado de fato. Mas percebam que, somente com essa ideia, imediatamente temos uma noção de como o nosso pulo funcionará. A partir disso, o artista/animador de nossa equipe logo tem um bom entendimento de como fazer o pulo. Poderia ser algo como visto na **Figura 2**, por exemplo!

Figura 02 - Exemplo de pulo partindo do T = 0 até o T = 4.



Fonte: Elaborada pelo autor com os recursos desenvolvidos pelo Setor de Produção Multimídia do IMD.

Se pegarmos os cinco desenhos demonstrados e colocarmos em sequência, já teremos uma animação funcionando, a qual pode ser utilizada para o nosso pulo! Basta que eles sejam separados em intervalos iguais na *timeline* da animação e colocados como *keyframes*! Mas isso ainda geraria uma animação de baixa qualidade, pois a taxa de quadros por segundo não está adequada para o tipo de movimentação desejada. A solução é simples! Desenhar mais poses intermediárias a essa e, então, adicionar mais quadros (*keyframes*) à nossa animação!

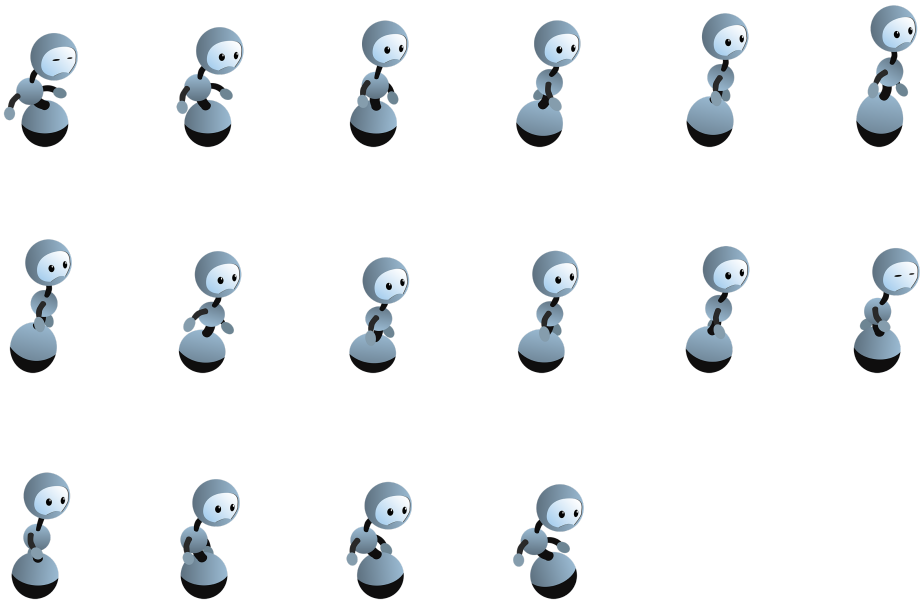
Isso vai gerar bastante desenho, não? E, se cada um desses desenhos for um arquivo, ficará tudo uma bagunça! Como conseguiremos lidar com isso? A resposta vocês já viram! E foi na aula sobre recursos, ou *assets*!

3. A Utilização de Sprite Sheets

As Sprite Sheets, apresentadas em nossa aula sobre recursos de jogos, são arquivos que guardam, dentro deles, uma sequência de sprites, os quais, ao serem colocados em uma *timeline* como *keyframes*, geram uma animação de determinado sprite. Para essa animação ser fluida, é preciso que haja uma quantidade adequada de sprites e que a diferença de movimentação entre cada um deles seja tão pequena

quanto necessária, de acordo com a quantidade de sprites presentes na Sprite Sheet utilizada. Vejamos, na **Figura 3**, um exemplo da Sprite Sheet usada em nosso jogo para a caminhada do robô. Perceba a quantidade de sprites e a diferença de movimentação entre eles.

Figura 03 - Sprite Sheet para movimentação do robô.



Fonte: Elaborada pelo autor a partir de recursos desenvolvidos pelo Setor de Produção Multimídia.

Com essa Sprite Sheet, já podemos fazer o personagem se movimentar enquanto anda. Para tanto, temos, ainda, de separar os sprites contidos nessa Sprite Sheet. Isso é feito, como vimos rapidamente na aula sobre recursos, utilizando o sprite editor do Unity. Antes disso, porém, precisamos que a Sprite Sheet obedeça a alguns detalhes importantes.

A fim de a separação dos sprites individuais ser feita adequadamente, é necessário a Sprite Sheet os posicionar de uma maneira similar, igualmente espaçada e, de preferência, mantendo todos na mesma altura vertical. Tudo isso deve ser feito para, ao cortarmos os sprites, termos um mesmo posicionamento, independentemente do frame no qual estivermos na animação. Pense, por exemplo, se alterarmos a altura do personagem no desenho, como está na **Figura 2**. Você consegue imaginar o que aconteceria durante o pulo do personagem? O sprite se movimentaria, mas o que aconteceria com os colisores, por exemplo? Eles ficariam na mesma posição, não é? E isso geraria uma discrepância visual terrível! O

personagem em si poderia passar por dentro de objetos, pois os colisores não estão mais onde a imagem está, mas ao mesmo tempo ele ficaria enganchado em objetos que nem tocam nele, devido ao mesmo problema. O caos!

Para evitar isso, devemos sempre manter o personagem na mesma posição a cada frame e dividir a Sprite Sheet igualmente, entre todos os frames que lá estão. No nosso caso, por exemplo, recebemos as imagens dos nossos artistas em uma resolução de 600 x 800. Para criar a Sprite Sheet mostrada na **Figura 3**, tivemos de criar uma nova imagem, com a resolução de 6*600 de largura e 3*800 de altura, para posicionar os sprites em 6 colunas com 3 linhas cada. Isso nos resultou na imagem de 3600x2400 que estará em nosso projeto em breve.

Manter essa proporção e conhecê-la, no entanto, é importante não apenas para que não haja essa variação indevida nos sprites, mas também para sabermos como cortar bem a nossa Sprite Sheet a fim de dividi-la em sprites individuais. O Unity oferece, em seu editor de sprites, uma opção de definir automaticamente o corte para estes. Ela, usualmente, é bem útil quando estamos trabalhando com um Atlas e queremos separar os sprites individualmente. Para as Sprite Sheets, porém, utilizar a opção de corte automático é quase uma garantia de o corte ser feito em tamanhos diferentes, criando justamente o efeito que queríamos evitar ao definir bem o grid das Sprite Sheets. Por esse motivo, devemos, quando possível, utilizar a opção de corte por grid, através de um tamanho conhecido.

Há bastante coisa para fazermos e lembrarmos. Como veremos na aula seguinte, também teremos de adicionar scripts que controlem a animação adequadamente. Nessa aula, lidaremos, ainda, com animações que não seguem simplesmente uma sequência de frames, pois obedecem, na verdade, ao comportamento do personagem.

Como temos tudo isso para fazer, que tal irmos ao nosso projeto? Adicionaremos uma primeira animação ao nosso amigo robô e, então, a partir desses passos, discutiremos todos os detalhes de animação diretamente no Unity e nos prepararemos para completá-la na próxima aula. Antes disso, vamos apenas responder uma pequena atividade para fixar o que foi visto até agora?

Atividade 01

1. Defina *keyframes* e o modo como eles se relacionam com a *timeline*.
2. Liste as principais vantagens de se utilizar uma *Sprite Sheet*.

4. Adicionando Animação ao Projeto DMJ I

Começaremos, agora, o processo de animação do nosso joguinho de DMJ I. Passaremos por várias etapas ao longo desse upgrade em nosso projeto e, à medida em que novos conteúdos surgirem, faremos uma pausa e esclareceremos alguns detalhes, combinado?

Como sempre, o projeto, até a etapa desenvolvida ao fim da aula passada, pode ser encontrado neste link.

4.1 Adicionando o Flip ao nosso Personagem

A princípio, cuidaremos de um detalhe simples, ao qual ainda não atentamos devidamente durante o nosso desenvolvimento. Faremos o personagem virar quando estiver se movendo para um lado ou para o outro! E isso pode ser feito de uma maneira bem fácil, servindo, entretanto, como uma perfeita introdução ao que faremos com os próximos passos da animação. Criaremos uma variável em nosso código a qual indica a direção que o nosso robozinho está virado e, a cada novo frame, vamos desenhar o robô voltado a essa direção. Vejamos o código alterado, na **Listagem 1**, objetivando fazer essa modificação!

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

public class PlayerController : MonoBehaviour {

    private bool jumping = false;
    private bool grounded = false;
    private bool doubleJump = false;
    private bool doubleJumping = false;
    private bool movingRight = true;

    private Rigidbody2D rigidBody;
    public Transform groundCheck;
    public LayerMask layerMask;

    public float acceleration = 100f;
    public float maxSpeed = 10f;
    public float jumpSpeed = 500f;

    // Use this for initialization
    void Awake () {
        rigidBody = GetComponent<Rigidbody2D> ();
    }

    // Update is called once per frame
    void Update() {
        grounded = Physics2D.Linecast(transform.position, groundCheck.position,
        layerMask);

        if (Input.GetButtonDown("Jump"))
        {
            if (grounded) {
                jumping = true;
                doubleJump = true;
            } else if (doubleJump) {
                doubleJumping = true;
                doubleJump = false;
            }
        }
    }

    //Called in fixed time intervals, frame rate independent
    void FixedUpdate() {
        float moveH = Input.GetAxis ("Horizontal");

        if (moveH < 0 && movingRight) {
            Flip();
        } else if (moveH > 0 && !movingRight) {
            Flip();
        }
```

```
52
53      if (rigidBody.velocity.x * moveH < maxSpeed) {
54          rigidBody.AddForce (Vector2.right * moveH * acceleration);
55      }
56
57      if (Mathf.Abs (rigidBody.velocity.x) > maxSpeed) {
58          Vector2 vel = new Vector2 (Mathf.Sign (rigidBody.velocity.x) * maxSpeed,
59           rigidBody.velocity.y);
60          rigidBody.velocity = vel;
61      }
62
63      if (jumping) {
64          rigidBody.AddForce(new Vector2(0f, jumpSpeed));
65          jumping = false;
66      }
67      if (doubleJumping) {
68          rigidBody.velocity = new Vector2 (rigidBody.velocity.x, 0);
69          rigidBody.AddForce(new Vector2(0f, jumpSpeed));
70          doubleJumping = false;
71      }
72
73  }
74
75  void Flip() {
76      movingRight = !movingRight;
77      transform.localScale = new Vector3((transform.localScale.x * -1),
78       transform.localScale.y,
79      transform.localScale.z);
80  }
81 }
```

Listagem 1 - Adicionando o método Flip para virar o personagem à medida em que este se mova.
Fonte: Elaborada pelo autor.

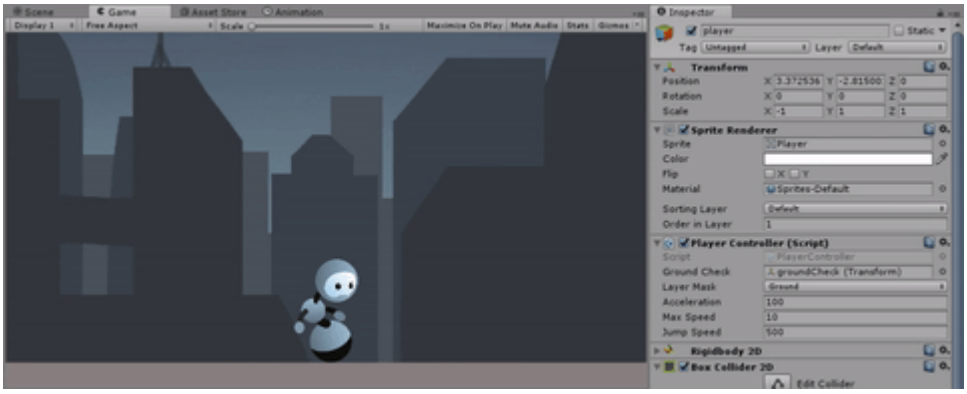
Observe, na **Listagem 1** a criação de uma nova variável booleana chamada de movingRight, indicando esta se o personagem teve sua última movimentação para direita (true) ou não (false). A partir disso, podemos testar, antes de realizar a movimentação do personagem, qual a direção que queremos nos mover e qual a direção que o personagem está orientado. Caso a direção do movimento seja diferente do valor de mover (moveH > 0 && !movingRight), invocamos o novo método criado, chamado Flip().

Esse método, que não possui qualquer parâmetro ou qualquer retorno, inverte o valor da variável de movimentação e, em seguida, altera a escala do personagem. Aí você se pergunta: “Mas a escala não serve para alterar o tamanho?”. Sim. A escala

altera o tamanho. Entretanto, ao alterarmos o valor **do sinal** da escala, espelhamos o objeto no eixo que teve o seu sinal alterado. Se alterarmos, por exemplo, o X, mudamos a direção na qual o objeto está orientado horizontalmente. O mesmo vale para o eixo Y. Se alterarmos o sinal da escala nesse eixo, colocaremos o personagem de cabeça para baixo. Interessante, não?

É importante ressaltar que, para alterarmos a escala em X do objeto, precisamos criar um novo Vector3 (pois a escala é composta por valores em X, Y e Z) e, então, colocar o valor da localScale do objeto como sendo esse novo Vector3. Os valores utilizados para esse nosso vetor são os valores da escala antiga, porém, multiplicando o X por -1, a fim de inverter o seu sinal. É exatamente a mesma coisa que está sendo feita na função Flip(). Legal, não? E, somente com isso, o nosso personagem já muda de direção ao se movimentar!

Figura 04 - Robô se movendo com a alteração do sinal da escala.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

O principal aspecto desse primeiro exemplo realizado é justamente a utilização da variável movingRight. Veremos, na próxima aula, que a animação depende de variáveis, as quais serão indicadoras de onde o personagem está e o que ele está fazendo para que a animação correta seja utilizada. Essa ideia funcionará da mesma maneira como funcionou com o movingRight. Teremos variáveis de controle indicando onde o personagem está e, de acordo com as alterações sofridas por elas, alteraremos as animações. É bem simples! :D

Mas, para criarmos nossas animações, precisamos, antes, que o nosso personagem possua um animador capaz de controlar cada uma delas. A fim de isso ser possível, adicionaremos, agora, um novo animador ao personagem, para que tal animador receba os estados, as animações e faça tudo se mover adequadamente. E isso pode ser programado visualmente! Animador, não? :P

4.2 Adicionando o Animator e Criando o Animator Controller

A fim de adicionar o animador, devemos primeiramente selecionar o nosso personagem, na hierarquia, e, em seguida, clicar no botão Add Component -> Miscellaneous -> Animator. Isso adicionará ao nosso personagem um novo componente chamado Animator. Esse componente possui cinco propriedades, mas não entraremos em detalhes sobre cada uma delas. Deixarei nas leituras complementares um link para o material oficial do Unity sobre animações. Os interessados em olhá-lo, no entanto, verão que a maioria das coisas envolvidas com esse componente, no Unity, estão relacionadas ao 3D. Como estamos lidando aqui apenas com 2D, focaremos somente no que nos interessa!

A propriedade Apply Root Motion deve ser desmarcada, uma vez que queremos controlar a posição e rotação do personagem diretamente, e não através da animação em si. Mantemos o transform sob controle do RigidBody2D!

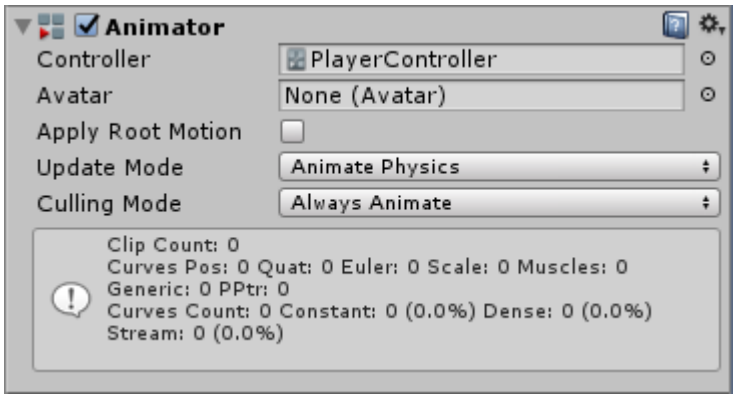
A propriedade Culling Mode deve ser mantida em Always Animate, uma vez que o nosso personagem terá sempre uma animação acontecendo. Já a propriedade Update Mode pode ter seu valor alterado para Animate Physics. Essa propriedade faz as animações serem feitas em conjunto com o motor de física e acontecerem baseadas no FixedUpdate, assim como é o caso das operações desse motor. Com isso, ficamos mais tranquilos do que ao alterar o valor de alguma das variáveis de controle da animação, pois esta responderá adequadamente.

Por falar em variáveis de controle, a primeira dessas propriedades, Controller, indica qual será o controlador que estará associado àquele Animator. Apesar do Animator ser o componente responsável por adicionar todo esse controle de animações ao seu GameObject, quem realmente faz todas as transições, animações e relações entre elas é um asset chamado Animator Controller. Precisamos de um desse também, não é?

Veja que há uma diferença importante aí! O Animator é um **componente** e pode ser adicionado ao seu GameObject como maneira de controlar as animações do personagem. Esse controle, no entanto, é feito por um **asset** do tipo Animator Controller. Então, adicionamos o Animator ao nosso GameObject pelo botão Add Component. Já o Animator Controller, criamos em uma pasta de nosso projeto através do menu Assets -> Create -> Animator Controller. Recomendo que seja

criada uma pasta, nos assets, chamada Animations, para podermos guardar esse objeto, além das animações que o utilizaremos para controlar. Chame o Animation Controller de PlayerController e o adicione como Controller na propriedade. Após todas essas alterações, o componente deve ficar como visto na **Figura 5**.

Figura 05 - Animator configurado para o nosso personagem.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Agora que temos o nosso componente Animator configurado e o nosso Controller definido, podemos fazer as animações de fato ocorrerem. Para isso, precisamos seguir mais quatro etapas: adicionar as Sprite Sheets ao nosso programa, criar as animações a partir destas, configurar o Controller para fazer as transições adequadamente e, por último, codificar a fim de as animações serem exibidas nos momentos adequados! Nesta aula, veremos as duas primeiras etapas e um pouco da terceira. Na próxima aula, veremos as duas últimas etapas com seus devidos detalhes!

4.3 Importando e Tratando as Sprite Sheets

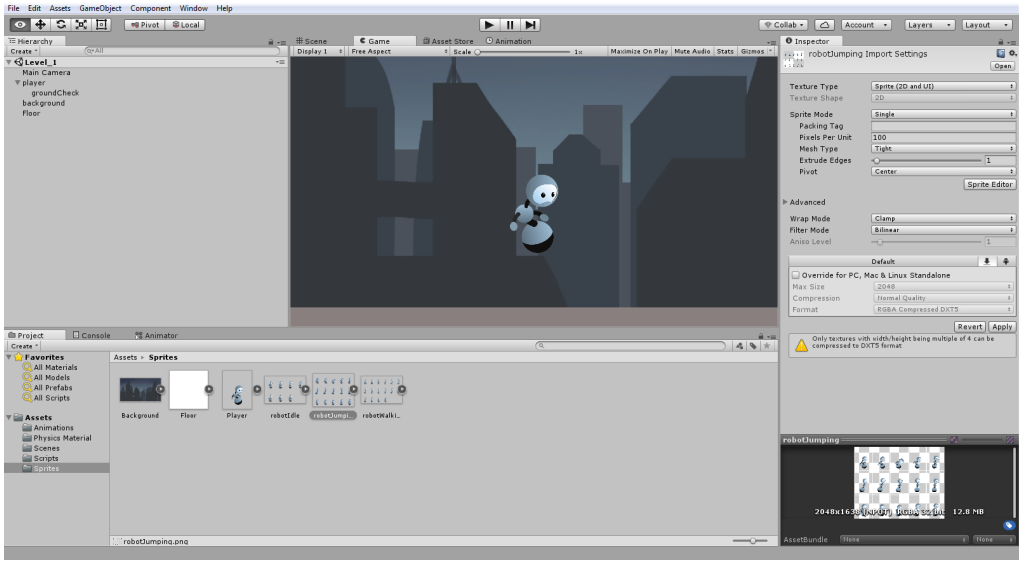
A primeira ação que precisaremos realizar objetivando criar as animações de nosso personagem é a importação das Sprite Sheets para o Unity. As três Sprite Sheets utilizadas nesta aula podem ser baixadas neste link, o qual contém um zip com a Sprite Sheet de Idle, Jumping e Walking, que serão as animações a serem adicionadas ao nosso robô nesse exemplo.

Após baixar e descompactar o arquivo, teremos três imagens PNG contendo as animações que queremos adicionar. O próximo passo é importá-las para o Unity. Já vimos isso na aula sobre recursos, então, recomendo voltar um pouco caso ainda tenham alguma dúvida de como fazer. Só lembrando, basta arrastar até a pasta de

Assets do Unity, e os arquivos aparecerão no local onde forem soltos. Recomendo, por organização, colocar na pasta Sprites! Perceba que, caso haja alguma incompatibilidade, em relação ao Unity, com os sprites utilizados, isso será avisado durante a importação. Se for algo de fácil resolução, como o tamanho da Sprite Sheet não estar adequado, o próprio Unity resolverá para você!

Logo de início, precisaremos alterar o tipo da imagem que foi adicionada ao nosso projeto. Para ter acesso às propriedades da imagem importada, basta clicar nela, na pasta onde foi colocada, dentro do Unity, como vemos na **Figura 6**.

Figura 06 - Sprite Selecionado, com as configurações exibidas no Inspector.



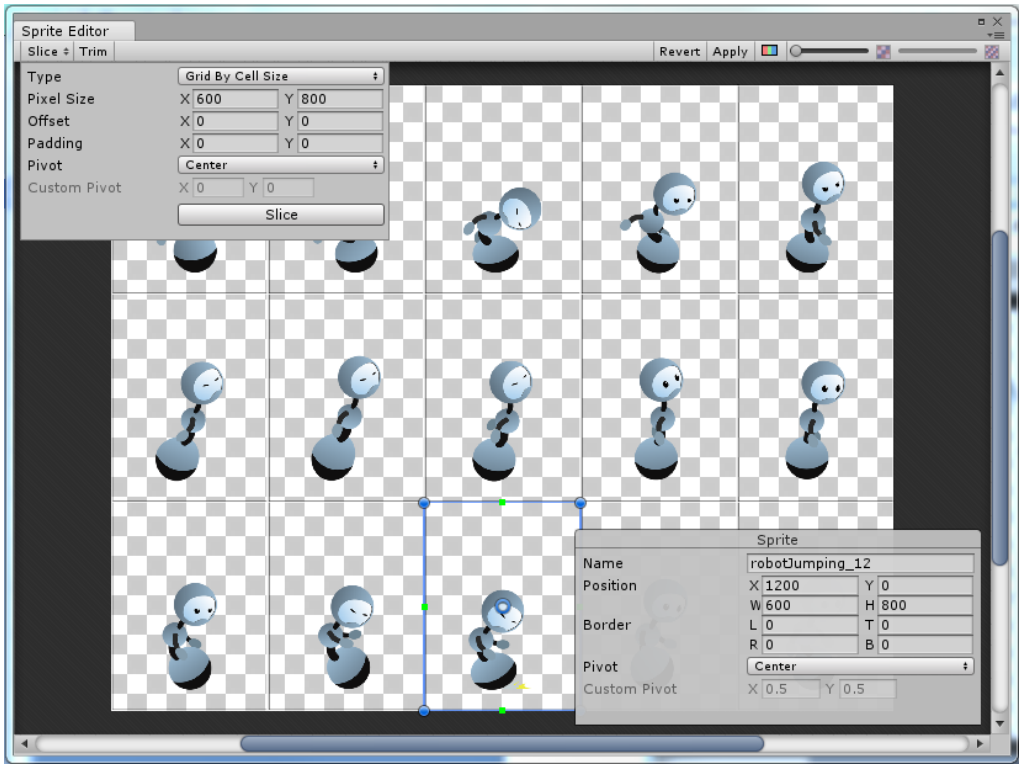
Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Como criamos um projeto 2D, as imagens já serão importadas no formato correto, tendo o Texture Type como Sprite (2D and UI). O problema, no entanto, está no Sprite Mode. Por padrão, as imagens são importadas com Sprite Mode sendo Single. Nesse caso, não será. Precisaremos alterar o valor de Sprite Mode para Multiple, uma vez que o nosso arquivo contém múltiplas imagens. Feito isso, clique no botão Apply, no canto inferior direito do Inspector. Repita esse procedimento para a Sprite Sheet de Idle, Walking e Jumping.

Concluída essa etapa, nossa Sprite Sheet já será considerada múltipla pelo Unity e, como vocês podem ver na **Figura 6**, terá uma setinha ao lado da imagem, permitindo vermos todos os sprites extraídos dela. Inicialmente, no entanto,

teremos apenas um, visto que o corte não foi executado ainda. Precisamos ensinar ao Unity como cortar a nossa Sprite Sheet! E, para isso, utilizaremos o Sprite Editor. Cliquemos no botão Sprite Editor para abrir essa funcionalidade, vista na **Figura 7**.

Figura 07 - Sprite Editor contendo a nossa Sprite Sheet de Jumping.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Após abrir o Sprite Editor, devemos indicar o tipo de corte que faremos em nossa Sprite Sheet. Para isso, utilizaremos justamente a janela mostrada no canto superior esquerdo da **Figura 7**. Para abrir essa janelinha de corte, basta clicar na opção Slice, logo abaixo do nome Sprite Editor.

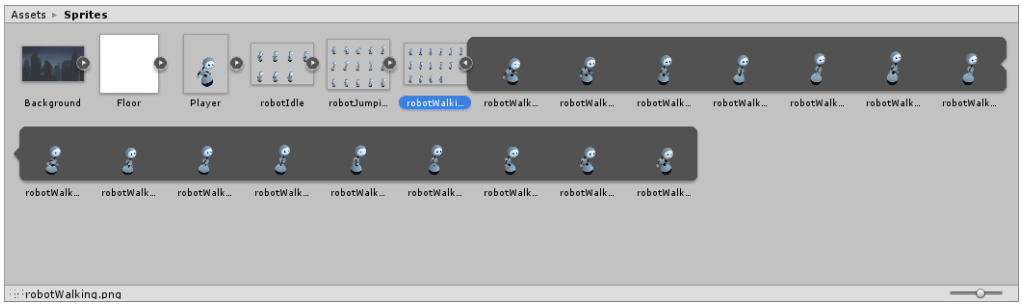
Nela, por padrão, a opção Automatic estará selecionada. Como já discutimos anteriormente, essa não é a melhor opção para o caso de Sprite Sheets. Uma vez que temos nossas imagens divididas especificamente em um Grid, selecionaremos como Type, para o Slice, a opção Grid By Cell Size. Essa opção pede para digitarmos o Pixel Size, o qual diz respeito ao tamanho em pixels de cada célula e, também, à presença de algum Offset ou Padding entre as imagens, além disso, solicita onde será o pivô, ou o centro, de nossa animação, através da opção Pivot. Em conformidade com o que já vimos, o tamanho utilizado para essas imagens foi de

600x800 cada, então, esse é o tamanho a ser colocado para o corte. Não há nenhum Padding ou Offset nesse grid, portanto, basta colocarmos essa opção e clicarmos em Slice.

Feito isso, linhas claras aparecerão ao redor dos sprites, como visto na **Figura 7**, determinando qual a área pertencente a cada um deles. Se você clicar em alguma dessas imagens, verá uma seleção azul ao redor dela, indicando, como também é visto na **Figura 7**, o seu tamanho, juntamente a um quadro no qual é exibido precisamente o nome do sprite, a posição dele na Sprite Sheet, a existência ou não de bordas e o seu pivô, representado pelo círculo azul que está no centro do personagem.

Após esses passos, a nossa Sprite Sheet já está bem dividida e pronta para ser cortada! Basta clicar em Apply, que está no canto superior direito, na mesma barrinha da opção Slice, e pronto! Quando voltarmos ao editor, podemos clicar na setinha ao lado da Sprite Sheet a fim de ver os diversos sprites cortados a partir dela pelo Unity e prontos para o nosso uso. Veja, na **Figura 8**, a exibição desses sprites já cortados para a animação de caminhada.

Figura 08 - Sprite Sheet de robotWalking já cortada, exibindo os 16 sprites que foram extraídos dela.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Depois de cortados, os sprites podem ser tratados como arquivos únicos! Apesar disso, ainda continuam todos organizados como parte de uma mesma Sprite Sheet, portanto, terão o mesmo destino que esta tiver. Se você resolver deletar a Sprite Sheet, todos os sprites também serão deletados. Se ela for atualizada, todos serão. E por aí vai...

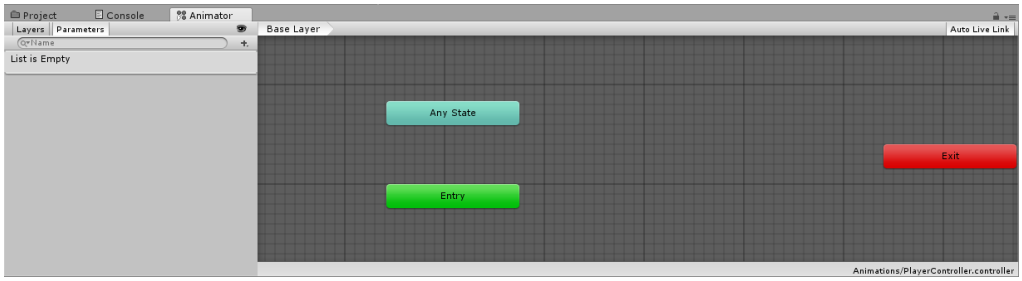
Repita esse procedimento de corte para as três Sprite Sheets e estaremos prontos a seguir adiante! É importante notar: em algumas Sprite Sheets, o grid não estará completo, o que poderia gerar alguns sprites vazios. Para evitar isso, basta clicar nos lugares que estão sem sprites mas estão marcados e, em seguida, apertar Delete. Desse modo, o Unity entenderá aquele bloco como não fazendo parte da Sprite Sheet e dará um fim a ele, evitando maiores problemas!

4.4 Conhecendo o Animator Controller

Agora que temos as nossas Sprite Sheets cortadas e o nosso Animator no personagem, com o Animator Controller, podemos começar a pensar em como colocaremos as animações no Animator Controller para este poder controlá-las. A fim de fazermos isso, o primeiro passo é acessar uma das duas novas abas com as quais trabalharemos até o fim desta aula. A primeira delas é a aba Animator, a segunda, a aba Animations.

Para abrir a aba Animator, navegue até o seu Animator Controller, criado na pasta Animations, dos Assets, e abra-o, dando dois cliques. Ao fazer isso, uma janela, como a vista na **Figura 9**, será aberta. Arraste-a, posicionando-a na mesma barra em que estão Project e Console (esse passo é opcional, mas é assim que eu gosto de trabalhar, pois essa configuração nos permite ver a animação sendo executada enquanto o jogo está em modo Play).

Figura 09 - Animator Window já posicionada na barra de Project e Console.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

A janela do Animator, vista na **Figura 9**, contém, na parte quadriculada, os estados de sua animação. Esses estados são justamente o que o Animator Controller é responsável por gerenciar. Cada um deles representa um conjunto diferente de animações, e somente um pode ser executado a cada frame. Inicialmente, não há qualquer animação carregada no Animator, uma vez que ainda não definimos nenhuma animação em nosso projeto, mas já há alguns estados base possíveis

serem utilizados em qualquer controlador de animação desenvolvido por nós. O primeiro deles, em azul, é o Any State. Esse estado representa qualquer estado e é responsável por armazenar transições que aconteçam independentemente de onde o personagem está. Por exemplo: estamos no meio de uma caminhada e, então, resolvemos pular. Devemos, naquele momento, passar ao estado de pulo! Mas e se tivermos parados e resolvermos pular? A mesma coisa! Vamos ao estado de pulo a partir do Idle ou do Walking, portanto, a partir de qualquer estado, passamos ao pulo. E é para isso que temos o Any State!

Já o estado Entry representada a entrada na máquina de estados da animação. Assim que o Animator Controller for carregado, a primeira ação por ele realizada é partir do Entry para algum lugar. Isso é uma das configurações que deveremos fazer para ele funcionar adequadamente.

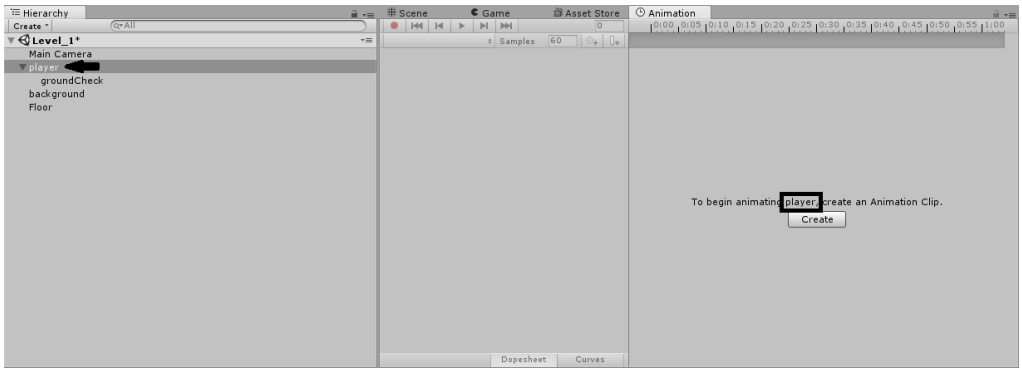
O último estado é o Exit, o qual representa a saída da máquina de estados da animação. Deve ser utilizado caso desejemos fazer o personagem sair daquela máquina de estados e parar de obedecer as animações que ali estavam.

Após entendermos bem a primeira dessas abas, podemos utilizar a segunda para começar a interação entre os elementos visuais e o nosso personagem. Utilize o atalho Ctrl + 6 ou o menu Window -> Animation a fim de abrir a aba de Animation e começar a criar as animações para o nosso personagem! Nesse caso, prefiro posicionar a aba juntamente às abas Scene e Game, uma vez que é interessante conseguir mantê-la aberta juntamente ao Animator, mas não é importante durante a execução do jogo em si. Novamente, o posicionamento é opcional! Fique à vontade para fazer de seu jeito. ;)

4.5 Criando Animações e Adicionando-as ao Controller

Agora que acessamos a janela de Animation, podemos começar a criar, de fato, as animações a serem utilizadas em nosso personagem. Para elas serem criadas corretamente, no entanto, devemos selecionar o **player** em nossa janela de Hierarchy, antes de criar qualquer novo clipe de animação. Em virtude de isso ser muito importante, a própria janela de Animation diz, antes mesmo de criarmos os clipes de animação, qual o objeto que estamos animando, como vemos em destaque na **Figura 10**.

Figura 10 - Criando a animação do Player.



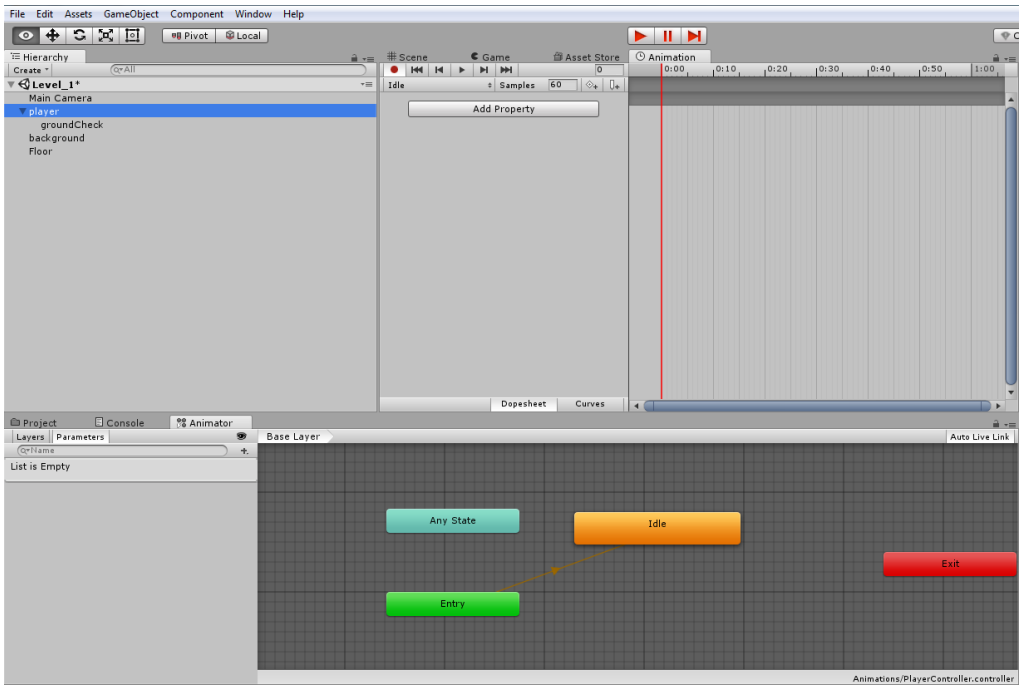
Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Neste momento, com a certeza de estarmos trabalhando em animações para o Player, podemos criar novas animações, as quais serão automaticamente inseridas no Animator Controller daquele elemento, facilitando bastante a nossa vida e permitindo já lidarmos com elas diretamente, sem trabalho adicional!

Para criar a nova animação, basta clicarmos no botão Create, na janela Animation. Isso nos trará uma janela perguntando o nome de nossa nova animação e onde salvá-la. Utilizaremos o nome Idle e a salvaremos juntamente ao seu Controller, na pasta Animations. Ao fazer isso, a animação será criada e a janela de animação mudará para uma *timeline*, na qual posicionaremos os *keyframes* a fim de a nossa animação funcionar!

Além disso, uma outra alteração visível é o Unity ter criado, automaticamente, dentro de nosso Animator Controller, um novo estado para a máquina de estados, chamado de Idle. Esse estado representa justamente a animação criada. Perceba que o Unity também fez, além de criar o estado, uma transição (representada pela linha com uma seta laranja) entre ele e o estado **Entry**, o qual diz respeito ao estado inicial falado anteriormente. Com isso, o Unity está dizendo que, ao iniciar o elemento, a primeira coisa a acontecer é tocar a animação Idle! E é exatamente isso que queremos! Bom! Vejamos as duas views modificadas com a nova animação na **Figura 11**.

Figura 11 - Animação Idle criada. Sua timeline é exibida na aba Animation e seu estado exibido abaixo, na aba Animator.

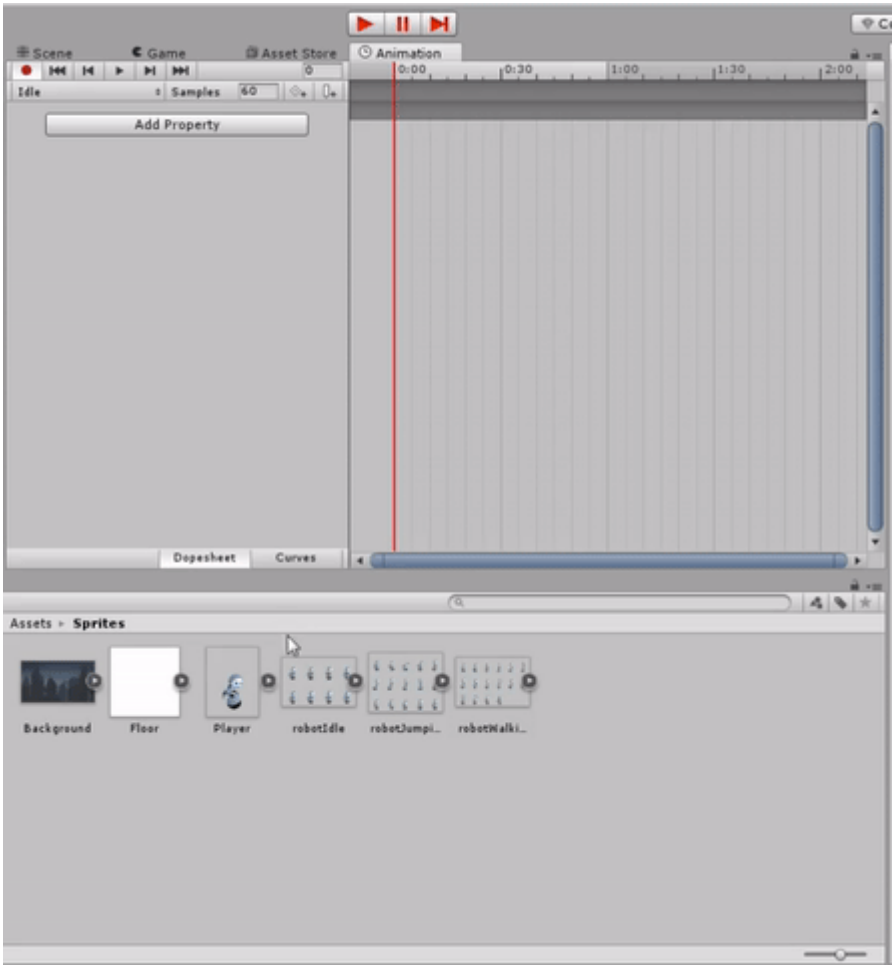


Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Com a animação criada, e seu estado correspondente adicionado no controlador como estado inicial, já teremos uma animação sendo executada quando o personagem for criado, no início do jogo. Podemos ver isso ao apertar Play em nossa cena, sendo a execução do estado representada por uma barrinha enchendo abaixo da animação Idle, na view Animator. Essa barra representa a animação executada e sua porcentagem em relação ao total. Quando a barra enche, a animação está concluída. Como, usualmente, as animações estão em loop, isso permite que a barrinha seque e comece a encher novamente, representando uma nova execução. Apesar disso, o personagem ainda não se moverá, devido ao fato de não termos, ainda, configurado os *keyframes* de nossa animação. Façamos isso!

Cada um dos sprites recortados a partir da Sprite Sheet representará um *keyframe* diferente, então, precisamos adicionar todos os sprites à nossa animação, para que possamos montar, a partir deles, a animação completa. Para fazer isso é também bastante simples. Basta navegar até os sprites na aba Project e ampliar a Sprite Sheet, revelando todos os sprites recortados, como visto lá na **Figura 8**. Feito isso, basta selecionar todos os sprites e arrastá-los até a nossa animação. Clique no primeiro sprite, segure shift e clique no último. Em seguida, arraste-os até o início da animação, na aba Animation. A **Figura 12** demonstra esse procedimento.

Figura 12 - Definindo os *keyframes* da animação a partir das imagens recortadas da Sprite Sheet Idle.



Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Após realizar esse procedimento, perceba que cada uma das imagens fica adicionada à *timeline* da sua animação como um *keyframe*, representado por um quadrado na diagonal (achei ser um losango, mas a produção me informou que não é). Feito isso, podemos clicar em Play e, devido à animação ser a padrão após a inicialização do objeto, veremos o resultado dela, sendo executada automaticamente. Go Play! /o/

Figura 13 - Animação sendo executada em uma velocidade muito alta.

Conteúdo interativo, acesse o Material Didático.

Fonte: Captura de tela do Unity. Disponível em: https://unity3d.com/pt/

Wow! Ficou *um pouco* rápido hein?! Pois é! Mas funciona! :D

A fim de reduzir a velocidade, precisamos voltar à nossa animação, na view Animation, para alterar um parâmetro importante. Perceba existir na parte superior, ao lado do dropdown no qual está o nome da animação, um valor para a variável Samples. Esse valor indica quantos quadros da sua animação serão executados a cada segundo. Por padrão, o valor 60 é utilizado, visando uma taxa de 60 frames por segundo. No caso de animações 2D, no entanto, é comum serem mais lentas e ficarem sem alteração por alguns instantes. Devido a isso, podemos definir que a nossa animação terá uma taxa de sample menor. Podemos utilizar, por exemplo, o valor 4, o qual é a quantidade de quadros que temos em nossa animação dividido pelo tempo total em segundos que ela deve durar.

Podemos, agora, dar Play novamente em nosso jogo e opa! Nosso personagem estará animado para a animação de Idle e pronto para receber novas animações! Entretanto, essa parte, contendo as alterações de script e os detalhes de uma Blend Tree, método muito importante para animações que dependem das ações do personagem e não somente de uma *timeline*, serão assuntos da próxima aula! Já vimos bastante coisa por hoje, e todas elas muito importantes! Vamos exercitar um pouco para fixar bem tudo na cabeça?

Atividade 02

1. Indique qual o objetivo de utilizarmos a view Animator.
2. Defina para que serve a view Animation.

Conclusão

Assim, finalizamos a nossa Aula 07! Foi bastante coisa, hein? Tanta coisa que ainda sobrou uma boa parte para ser discutida na Aula 08, na qual continuaremos falando sobre animação! Aprenderemos, nessa aula, a fazer transições entre as animações e, também, o que é uma Blend Tree e como podemos utilizá-la em nosso jogo. Até lá, pessoal! o/

Leitura Complementar

Manual oficial do Unity sobre animação

<https://docs.unity3d.com/Manual/AnimationSection.html>

Resumo

Na aula de hoje, a primeira sobre animação dentre as duas em que esse assunto será abordado, entendemos melhor a importância da animação nos jogos e como, há bastante tempo, precisamos adicionar esse fator para aumentar a qualidade de nossos jogos. Vimos o que é animação por *keyframes* e como eles funcionam para uma animação 2D.

Ainda nesta aula, começamos a adicionar animação ao nosso projeto, adicionando um Animator ao nosso personagem, com um Animator Controller atrelado e uma animação para seu comportamento Idle. Vimos os detalhes do componente Animator e, também, dos assets necessários para fazer a animação e o controlador.

Para a criação da animação, relembramos as Sprite Sheets, as quais já havíamos conhecido na aula sobre recursos, e importamos algumas delas em nossos projetos, detalhando o modo de como realizar o corte dos sprites e de como utilizá-los depois de cortados.

Foi uma aula bem interessante, mas complexa, que vai ser muito importante para a continuidade do assunto na aula seguinte. Tenha bastante cuidado em relação aos pontos discutidos hoje, para não ter grandes dificuldades na próxima, quando adicionaremos novas animações ao personagem, com transições e controle através de script. Aprenderemos, ainda, o que é e como utilizar uma Blend Tree! Até lá!

Como sempre, o projeto desenvolvido até o momento pode ser encontrado neste link. Esse será o ponto de partida para o desenvolvimento da próxima aula!

Autoavaliação

1. Qual a importância da animação em jogos 2D?
2. Como funciona a animação através de keyframes?
3. É possível utilizar sprites separados no Unity? Qual a desvantagem disso?
4. Qual o principal componente usado para lidar com animações? E quais os assets que precisamos criar para tal atividade?

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.

RABIN, Steve. **Introdução ao Desenvolvimento de Games**, Vol 3. CENGAGE.