Introdução ao Unity

Primeiros passos para a criação de um jogo 2D de plataforma

Professor: Francesco Silva

INSTALAÇÃO

1. Baixar o UnityHub neste link
https://public-cdn.cloud.unity3d.com/hub/prod/UnityHubSetup.exe

2. Instalar o Unity e criar uma conta gratuita

3. No UnityHub, clicar em “New”

4. Escolher a opção 2D, dar um nome para o projeto e selecionar onde deseja salvar

PREPARAÇÃO INICIAL

Vamos precisar de alguns materiais gráficos para o projeto, como por exemplo o personagem, o cenário etc. Existem diversos sites que disponibilizam gratuitamente esse tipo de material.

1. Acessar o site
https://www.gameart2d.com/freebies.html

2. Escolher qual sprite e cenário deseja utilizar (para o personagem, escolha um que possua pelo menos as opções de Parado, Correndo, Pulando e Atacando)

3. Baixar o material

CRIANDO O PERSONAGEM

1. Criar pasta no Assets para > Sprites > Personagem

2. Adicionar os assets do personagem (clicar e arrastar as imagens)

ATENÇÃO
3. Nas propriedades da Sprite, o pivot sempre deve ficar na opção “Bottom”

4. Arrastar a primeira imagem do personagem Parado para a cena e renomear para “Jogador”

Tags
Tags são utilizadas para identificar um ou um grupo de Objetos no Unity.
Mais em: https://docs.unity3d.com/Manual/Tags.html

1. Mudar a Tag do objeto para “Jogador”.

Pronto. Até aqui já temos um personagem na tela. Se clicar no Play, irá mostrar o personagem parado na tela.
Todo objeto no Unity pode receber alguns componentes que servem para coisas específicas. Um personagem, por exemplo, precisa ter um peso, uma movimentação, é preciso controlar a colisão etc.

RigidBody2D
O primeiro componente que precisamos é um RigidBody2D. Esse componente é responsável por controlar a engenharia física do seu objeto. Logo, seu objeto passa a ter um peso e você precisa controlar como ele se comporta. A gravidade vai passar a atuar nesse objeto.

Existem diversas propriedades e não vamos conseguir passar por todas, mas podemos experimentar algumas coisas.

Mais em: https://docs.unity3d.com/ScriptReference/Rigidbody2D.html

1. Com o objeto do personagem selecionado, clicar em “Add component” e adicionar RigidBody2D.

2. Experimentar alguns itens, como “Mass” e “Gravity scale” e disparar o Play.

3. Nas opções do RigidBody, clicar em Constraints e marcar “Freeze Rotation Z”
Considerando um jogo 2D, estamos trabalhando com um plano cartesiano composto por X e Y. Logo, não trabalhamos com a 3ª dimensão de profundidade. Assim, nosso personagem não precisa da movimentação em Z, por isso ela pode ser congelada.

Qual o melhor tipo de rigidBody para cada situação? https://needoneapp.medium.com/unity-should-i-choose-kinematic-static-or-dynamic-for-rigidbody-2d-body-type-191ce65fa35f

CRIANDO UM PISO
Já vimos que a nossa personagem cai quando iniciamos o jogo. Isso porque não temos nenhum tipo de piso na cena. Vamos adicionar esse novo item:

1. Na pasta Sprite, clicar com o botão direito Create > 2D > Sprites > Square

2. Renomear para “Piso”

3. Arrastar para a cena

4. Diminuir/aumentar a escala para o tamanho desejado

Somente isso não é suficiente para controlar o contato com o personagem. Ao clicar no Play, o personagem continua caindo.
Isso porque não definimos os colisores.

BoxCollider2D
https://docs.unity3d.com/Manual/class-BoxCollider2D.html

Os colisores são parte importante dos jogos, já que definem como os objetos se comportam quando existe uma colisão física entre eles. Uma colisão pode determinar o contato de um personagem com o chão (quando o personagem toca o chão), pode determinar a morte ou dano a um personagem (quando o personagem toca um inimigo) etc.
Todos os objetos que podem sofrer colisões necessitam um Collider (no nosso caso, as opções em 2D). Temos vários tipos de colisores, mas para esse caso, vamos utilizar o BoxCollider2D.

1. Clicar no personagem, ir em “Add Component” e selecionar a opção “BoxCollider2D”
(Só isso não é suficiente, já que apenas o personagem possui um Colisor agora)

2. Clicar no piso e repetir o processo.

Agora sim, o personagem se mantém acima do piso.

MOVIMENTAÇÃO DO PERSONAGEM

SCRIPTS
Muitas das ações e atualizações dos jogos são controladas através de scripts, que são códigos-fonte específicos para tratar as movimentações, cliques em teclas específicas para jogos etc. A movimentação do personagem é manipulada através de um Script. Vamos criá-lo nos próximos passos.

1. Criar pasta dentro de Assets e chamá-la de “Scripts”

2. Dentro da pasta, botão direito Create > C# Script

3. Renomear para “Jogador” e duplo clique para editar. O script será aberto em um editor pré-determinado (ou você poderá determinar qual editor usar)

As classes no Unity são filhas de MonoBehavior, que é uma classe-base para manipular todos os scripts derivados do Unity.
Normalmente, os scripts em Unity já se iniciam com dois métodos principais:

- **Start**: executado apenas uma vez, na inicialização da cena.
- **Update**: executa a cada atualização do frame do jogo. Por exemplo, se o jogo que está sendo desenvolvido rodar a 60fps (60 frames por segundo) esse método vai rodar 60 vezes em 1 segundo.

Podemos adicionar mais um: **FixedUpdate**
Esse método é parecido com o Update, mas executa em tempos fixos, diferente do update, que executa a cada frame. Isso quer dizer que, se um frame demorar mais do que outro para ser totalmente renderizado, o jogo poderá perder a sua fluidez.
Quando tratamos de itens referentes a física, que precisam de respostas lineares, como por exemplo a movimentação do personagem, o ideal é que essas atualizações sejam implementadas no FixedUpdate.

Mais em: https://learn.unity.com/tutorial/update-and-fixedupdate#5c8a4242edbc2a001f47cd63

Nesse script precisaremos controlar algumas coisas para permitir a movimentação do personagem. As seguintes variáveis devem ser criadas

- velocidade: float para controlar a velocidade que o personagem se move
- rbd2: do tipo RigidBody2D para controlar a física
- movimento: do tipo Vector2 para controlar como serão os movimentos do personagem em (2D). Vector2 permite a manipulação de 2 valores como em um plano cartesiano x e y.
  Mais sobre Vetor2 em https://docs.unity3d.com/ScriptReference/Vector2.html
- velocidadeX: float para controlar como o personagem se move no plano X

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

public class Personagem : MonoBehaviour
{
    //Controlar o RigidBody do personagem
    private Rigidbody2D rb2d;
    //Controlar o movimento do personagem
    private Vector2 movimento;
    //Controlar o eixo X quando o personagem se move
    private float velocidadeX;

    [Header("Propriedades do personagem")]
    //Controlar a velocidade que o personagem se move
    public float velocidade = 2f;

    // Start is called before the first frame update
    void Start()
    {
        //Ao iniciar, pegamos o RigidBody do personagem
        rb2d = GetComponent<Rigidbody2D>();
    }

    // Update is called once per frame
    void Update()
    {
```

```
        /*Toda vez que os frames são atualizados, queremos saber se o usuário
         * está utilizando as setas direcionais horizontais. Isso quer dizer
         * que ele está movimentando o personagem para frente ou para trás
         * Podemos determinar essas teclas em Unity > Edit > Project Settings > Input Manager
         * Mais sobre GetAxisRaw em:  https://docs.unity3d.com/ScriptReference/Input.GetAxisRaw.html
         */
         float horizontal = Input.GetAxisRaw("Horizontal");

        /*Agora, instanciamos o nosso movimento, passando o valor do horizontal e 0 para y, já que
         * não estamos movimentando o personagem para cima ou para baixo
         * */
        movimento = new Vector2(horizontal, 0);

    }

    private void FixedUpdate()
    {
        //Calculamos a nova velocidade de X que ocorreu com o movimento e levando em consideração a velocidade determinada
        velocidadeX = movimento.normalized.x * velocidade;
        //Atualizamos o RigidBody do personagem
        rb2d.velocity = new Vector2(velocidadeX, rb2d.velocity.y);
    }
}
```

1. Após finalizar todas as alterações, salvar o arquivo e voltar ao Unity.
2. Para adicionar o script ao Personagem: clicar no script e arrastar em cima do personagem

ROTACIONAR PERSONAGEM

Ao inspecionar o personagem, é possível perceber algumas propriedades referentes ao posicionamento do mesmo na Cena.
Essas propriedades são: Position, Rotation e Scale.
Em um jogo 2D, quando modificamos o eixo X da propriedade Scale do personagem, ele irá se movimentar para o outro lado.
Logo, X = 1 mantém o personagem virado para a direita
X = -1 mantém o personagem virado para a esquerda

Logo, queremos que quando o jogador mudar a direção de movimentação do personagem, a propriedade X do Scale seja alterada automaticamente, para que ele se movimente tanto para a esquerda quanto para a direita.
Precisamos fazer isso via Script. Vamos voltar ao script do Personagem

Precisamos de 2 variáveis

- direcao: int que determina qual a direção do personagem
- scaleXOriginal: float que pega o valor da prop Scale do personagem (como vimos no inspector)

Precisamos também de um novo método

- virar: que vai realizar a rotação do personagem quando necessário

As linhas em destaque foram as adicionadas nesse passo.

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

public class Personagem : MonoBehaviour
{
    //Controlar o RigidBody do personagem
    private Rigidbody2D rb2d;
    //Controlar o movimento do personagem
    private Vector2 movimento;
    //Controlar o eixo X quando o personagem se move
    private float velocidadeX;


    private int direcao = 1; //Direção inicial para a direita
    private float scaleXOriginal;



    [Header("Propriedades do personagem")]
    //Controlar a velocidade que o personagem se move
    public float velocidade = 2f;

    // Start is called before the first frame update
    void Start()
```

```
    {
        //Ao iniciar, pegamos o RigidBody do personagem
        rb2d = GetComponent<Rigidbody2D>();

        //Ao iniciar, pegamos o valor inicial de X em scale
        scaleXOriginal = transform.localScale.x;
    }

    // Update is called once per frame
    void Update()
    {
        /*Toda vez que os frames são atualizados, queremos saber se o usuário
         * está utilizando as setas direcionais horizontais. Isso quer dizer
         * que ele está movimentando o personagem para frente ou para trás
         * Podemos determinar essas teclas em Unity > Edit > Project Settings > Input Manager
         * Mais sobre GetAxisRaw em:  https://docs.unity3d.com/ScriptReference/Input.GetAxisRaw.html
         */
         float horizontal = Input.GetAxisRaw("Horizontal");

        /*Agora, instanciamos o nosso movimento, passando o valor do horizontal e 0 para y, já que
         * não estamos movimentando o personagem para cima ou para baixo
         * */
        movimento = new Vector2(horizontal, 0);

        if (velocidadeX * direcao < 0)
        {
            virar();
        }

    }

    private void FixedUpdate()
    {
        //Calculamos a nova velocidade de X que ocorreu com o movimento e levando em consideração a velocidade determinada
        velocidadeX = movimento.normalized.x * velocidade;
        //Atualizamos o RigidBody do personagem
        rb2d.velocity = new Vector2(velocidadeX, rb2d.velocity.y);
    }

    private void virar ()
    {
        //Atualizamos a variável direcao para a direção oposta
        direcao *= -1;

        /*Pegando todos os valores de scale. Como podemos ver no Inspector, essa é uma propriedade
         * composta pelos valores dos 3 eixos (X, Y e Z). Por isso vamos usar um Vector3.
         * Mais sobre Vector3 em: https://docs.unity3d.com/ScriptReference/Vector3.html
         */
        Vector3 scale = transform.localScale;

        //Atualizando o valor de X de scale
        scale.x = scaleXOriginal * direcao;

        //Atualizando a propriedade do personagem
        transform.localScale = scale;
    }
}
```

FAZENDO O PERSONAGEM PULAR

Antes de iniciar o desenvolvimento, é importante entender o mecanismo por trás das interações existentes entre o personagem e o cenário. Enquanto movimentando, é preciso “prever” quais são as possíveis colisões que o personagem irá sofrer. Por exemplo, estando em um plano reto (solo por exemplo), ao realizar a movimentação de pulo, o personagem deixa de tocar esse plano por alguns instantes e volta a tocá-lo. Toda essa movimentação precisa ser considerada, já que o personagem precisa “entender” que ele está de volta ao solo.

Para realizar ações como essa, é necessário entender um pouco sobre Raycast, uma funcionalidade existe no Unity (e em diversas literaturas sobre o desenvolvimento de jogos) que é um “disparador de raios” físicos, invisíveis e para qualquer direção, que “observam” as possíveis colisões que o personagem irá sofrer.

Por exemplo, voltando à ação de pulo, o que nos interessa saber em todo o corpo do personagem é apenas o momento em que os dois pés do personagem tocam o chão. Para isso, é necessário criar dois Raycasts para controlar a física dos pés do personagem.

Mais sobre Raycast em: https://docs.unity3d.com/ScriptReference/Physics.Raycast.html

Vamos criar um método para controlar quando existe uma colisão com o Raycast.

As linhas em destaque foram as adicionadas nesse passo.

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

public class Personagem : MonoBehaviour
{
    //Controlar o RigidBody do personagem
    private Rigidbody2D rb2d;
    //Controlar o movimento do personagem
    private Vector2 movimento;
    //Controlar o eixo X quando o personagem se move
    private float velocidadeX;


    private int direcao = 1; //Direção inicial para a direita
    private float scaleXOriginal;

    RaycastHit2D verificarEsquerda, verificarDireita;

    [Header("Propriedades do personagem")]
    //Controlar a velocidade que o personagem se move
    public float velocidadeCorrida = 2f;
    public float forcaPulo = 2f;


    [Header("Propriedades do chão")]
    public bool isNoChao;
    public LayerMask layerChao;
    public float distanciaChao;
    public Vector3[] pes;

    // Start is called before the first frame update
    void Start()
    {
        //Ao iniciar, pegamos o RigidBody do personagem
        rb2d = GetComponent<Rigidbody2D>();

        //Ao iniciar, pegamos o valor inicial de X em scale
        scaleXOriginal = transform.localScale.x;
    }

    // Update is called once per frame
    void Update()
    {
        /*Toda vez que os frames são atualizados, queremos saber se o usuário
         * está utilizando as setas direcionais horizontais. Isso quer dizer
         * que ele está movimentando o personagem para frente ou para trás
         * Podemos determinar essas teclas em Unity > Edit > Project Settings > Input Manager
         * Mais sobre GetAxisRaw em:  https://docs.unity3d.com/ScriptReference/Input.GetAxisRaw.html
```

```
         */
         float horizontal = Input.GetAxisRaw("Horizontal");

        /*Agora, instanciamos o nosso movimento, passando o valor do horizontal e 0 para y, já que
         * não estamos movimentando o personagem para cima ou para baixo
         * */
        movimento = new Vector2(horizontal, 0);

        if (velocidadeX * direcao < 0)
        {
            virar();
        }

        verificarFisica();

        if (Input.GetButtonDown("Jump") && isNoChao)
        {
            rb2d.velocity = Vector2.zero;
            rb2d.AddForce(Vector2.up * forcaPulo, ForceMode2D.Impulse);
        }

    }

    private void FixedUpdate()
    {
        //Calculamos a nova velocidade de X que ocorreu com o movimento e levando em consideração a velocidade determinada
        velocidadeX = movimento.normalized.x * velocidadeCorrida;
        //Atualizamos o RigidBody do personagem
        rb2d.velocity = new Vector2(velocidadeX, rb2d.velocity.y);
    }

    private void virar ()
    {
        //Atualizamos a variável direcao para a direção oposta
        direcao *= -1;

        /*Pegando todos os valores de scale. Como podemos ver no Inspector, essa é uma propriedade
         * composta pelos valores dos 3 eixos (X, Y e Z). Por isso vamos usar um Vector3.
         * Mais sobre Vector3 em: https://docs.unity3d.com/ScriptReference/Vector3.html
         */
        Vector3 scale = transform.localScale;

        //Atualizando o valor de X de scale
        scale.x = scaleXOriginal * direcao;

        //Atualizando a propriedade do personagem
        transform.localScale = scale;
    }

    private RaycastHit2D verificarRayCast (Vector3 origem, Vector2 direcao, float tamanho, LayerMask objeto)
    {
        Vector3 posicao = transform.position;
        RaycastHit2D hit = Physics2D.Raycast(posicao + origem, direcao, tamanho, objeto);

        Color cor = hit ? Color.red : Color.green;
        Debug.DrawRay(posicao + origem, direcao * tamanho, cor);

        return hit;
    }

    private void verificarFisica()
    {
        isNoChao = false;

        verificarEsquerda = verificarRayCast(new Vector2(pes[0].x, pes[0].y), Vector2.down, distanciaChao, layerChao);
        verificarDireita = verificarRayCast(new Vector2(pes[1].x, pes[1].y), Vector2.down, distanciaChao, layerChao);

        isNoChao = verificarEsquerda.collider != null || verificarDireita.collider != null;
    }
}
```

Antes de testar, precisamos adicionar uma layer no chão para ser possível identificá-la na colisão.

1. Clicar no objeto do chão
2. Inspector > Layer > Add layer
3. Escolhe uma UserLayer e definir para “Chao”

1. Clicar no personagem
2. No script, as variáveis públicas aparecem com opção de setar valores

3. Layer Chao, setar para “Chao”

4. Distancia Chao = 0.5 inicialmente, mas precisa testar

5. Ir para o Play para testar tudo

6. Adicionar dois valores para pés (um para cada pé) e ajustar os raycasts

7. Clicar com o espaço para testar o pulo

8. Ajustar a força do pulo

9. Quando tudo estiver ok, botão direito e Copy componente (tudo que está sendo feito durante o Play é perdido após parar a execução)

10. Parar o play

11. Ir novamente no component e escolher a opção “Paste component values” para copiar os valores usados nos testes

ANIMAÇÕES

1. Na pasta Assets, criar pasta Animations e dentro dela uma nova pasta Personagem

2. Dentro da pasta Personagem: Botão direito > Create > Animator Controller

3. Renomear para PersonagemController

4. Clicar no personagem > Add componente > Animator

5. Arrastar o JogadorController para a opção Controller do componente Animator que acabamos de adicionar

6. Abrir as duas abas de animação: Animation e Animator
Se não estiver aparecendo: menu Window > Animation e selecionar as que não estiverem aparecendo entre as duas citadas.

7. Manter o Personagem selecionado e na aba Animation, escolher a opção Create

8. Verificar se estamos na pasta correta (Animations > Player) e vamos criar a primeira animação “Parado”

9. Selecionar todas as sprites para o personagem parado e arrastar para a janela de animação

10. Clicar no play para ver a animação da sprite e ajustar os Samples
Se não estiver aparecendo, clique nos 3 pontinhos no canto da janela e escolher a opção Show Sample Rate

Na aba Animator, já podemos ver a nossa primeira animação “Parado” implementada.
Repetir o processo para criar a Animação “Correndo”

1. Na aba Animation, onde já está selecionada a animação “Parado”, clicar e escolher a opção “Create New Clip”

Repetir para “Pulando” (precisa ter uma para pulando e outra para “caindo” do pulo), “Morto”, “Atacando” e outras que desejar.

Após criar as animações, iremos manipulá-las na aba Animator, tratando os estados que o nosso personagem pode estar. A intenção aqui é criar as transições entre as animações. Por exemplo, o personagem pode estar parado e começar a andar. Então precisamos criar uma transição para isso.

Todos os passos a seguir deverão ser realizados na aba Animation

1. Clique na animação “Parado” > Make transition > Conectar com a animação “Correndo”

Repetir as transições para:

De: Correndo
Para: Parado

De: Parado
Para: Atacando

De: Atacando
Para: Parado

Para realizar a animação de pulo, precisamos unir quando inicia e quando termina o pulo

1. Botão direito na tela do Animator > Create State > From New Blend Tree

2. Nos parâmetros, mudar o nome de “Blend” para “velocidadeY”

3. Clique duas vezes no componente que foi criado e vamos mudar as propriedades no Inspector:
   Nome: PuloCompleto
   Motion: Adicionar 2 novos itens (Add motion field)
      O primeiro é quando está caindo. Então selecionar a motion “Caindo” e Threshold = -1
      O segundo é quando está pulando. Então selecionar a mottion “Pulando” e Threshold = 1
      Desmarcar a opção Automate Threshold para conseguir alterar os valores

Realizar novas transições:

De: Parado
Para: PuloCompleto

De: Correndo
Para: PuloCompleto

De: PuloCompleto
Para: Exit

PARÂMETROS

Com as animações finalizadas, vamos criar os parâmetros necessários para realizar tais animações.

1. Na janela Animator, aba Parameters, clicar no + e adicionar os seguintes parâmetros:
      Parado (boolean)
      Atacando (trigger)
      isNoChao (boolean)
      velocidadeX (float)
      velocidade (float)
      Morto (trigger)

2. Clicar na transição de Parado para Correndo e vamos mudar algumas propriedades

3. Desmarcar Has Exit Time para determinar que não é necessário terminar a animação para realizar a transição para outra animação

4. Nas “Conditions”, escolher quais parâmetros são necessários para realizar as transições:
      xVelocity tem que ser maior que 0.1 (realizou um movimento na horizontal)
      isGrounded tem que ser igual a true (o personagem está no chão)

Na transição de Correndo para Parado
- Desmarcar Has Exit Time
- Condições:
- velocidadeX tem que ser menor que 0.1 (realizou um movimento na horizontal)
- isNoChao tem que ser igual a true (o personagem está no chão)

De Correndo para PuloCompleto e de Parado para PuloCompleto
- Desmarcar Has Exit Time
- Condições
- isNoChao tem que ser igual a false

De Parado para Atacando
- Desmarcar Has Exit Time
- apenas selecionar o Atacando

De Atacando para Parado
Deixamos como está porque quando ele terminar de atacar ele volta para o Parado

De PuloCompleto para Exit
- Desmarcar Has Exit Time
- isNoChao tem que ser igual a true (o personagem está no chão)

Pronto!

ADICIONANDO AS ANIMAÇÕES NO SCRIPT
Precisamos controlar as animações do personagem via script.

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

public class Personagem : MonoBehaviour
{
    //Controlar o RigidBody do personagem
    private Rigidbody2D rb2d;
    //Controlar o movimento do personagem
    private Vector2 movimento;
    //Controlar o eixo X quando o personagem se move
    private float velocidadeX;


    private int direcao = 1; //Direção inicial para a direita
    private float scaleXOriginal;

    RaycastHit2D verificarEsquerda, verificarDireita;

    [Header("Propriedades do personagem")]
    //Controlar a velocidade que o personagem se move
    public float velocidadeCorrida = 2f;
    public float forcaPulo = 2f;


    [Header("Propriedades do chão")]
    public bool isNoChao;
    public LayerMask layerChao;
    public float distanciaChao;
    public Vector3[] pes;


    private bool isAtacando;
    private Animator animador;

    // Start is called before the first frame update
    void Start()
    {
        //Ao iniciar, pegamos o RigidBody do personagem
        rb2d = GetComponent<Rigidbody2D>();

        //Ao iniciar, pegamos o valor inicial de X em scale
        scaleXOriginal = transform.localScale.x;

        animador = GetComponent<Animator>();
    }

    // Update is called once per frame
    void Update()
    {
```

```
        if (!isAtacando)
        {
            /*Toda vez que os frames são atualizados, queremos saber se o usuário
             * está utilizando as setas direcionais horizontais. Isso quer dizer
             * que ele está movimentando o personagem para frente ou para trás
             * Podemos determinar essas teclas em Unity > Edit > Project Settings > Input Manager
             * Mais sobre GetAxisRaw em:  https://docs.unity3d.com/ScriptReference/Input.GetAxisRaw.html
             */
            float horizontal = Input.GetAxisRaw("Horizontal");

            /*Agora, instanciamos o nosso movimento, passando o valor do horizontal e 0 para y, já que
             * não estamos movimentando o personagem para cima ou para baixo
             * */
            movimento = new Vector2(horizontal, 0);

            if (velocidadeX * direcao < 0)
            {
                virar();
            }
        }

        verificarFisica();

        if (Input.GetButtonDown("Jump") && isNoChao)
        {
            rb2d.velocity = Vector2.zero;
            rb2d.AddForce(Vector2.up * forcaPulo, ForceMode2D.Impulse);
        }

        if (!isAtacando && isNoChao && Input.GetButtonDown("Fire1"))
        {
            movimento = Vector2.zero;
            rb2d.velocity = Vector2.zero;
            animador.SetTrigger("Atacando");
        }

    }

    private void FixedUpdate()
    {
        if (!isAtacando)
        {
            //Calculamos a nova velocidade de X que ocorreu com o movimento e levando em consideração a velocidade determinada
            velocidadeX = movimento.normalized.x * velocidadeCorrida;
            //Atualizamos o RigidBody do personagem
            rb2d.velocity = new Vector2(velocidadeX, rb2d.velocity.y);
        }
    }

    private void virar ()
    {
        //Atualizamos a variável direcao para a direção oposta
        direcao *= -1;

        /*Pegando todos os valores de scale. Como podemos ver no Inspector, essa é uma propriedade
         * composta pelos valores dos 3 eixos (X, Y e Z). Por isso vamos usar um Vector3.
         * Mais sobre Vector3 em: https://docs.unity3d.com/ScriptReference/Vector3.html
         */
        Vector3 scale = transform.localScale;

        //Atualizando o valor de X de scale
        scale.x = scaleXOriginal * direcao;

        //Atualizando a propriedade do personagem
        transform.localScale = scale;
    }

    private RaycastHit2D verificarRayCast (Vector3 origem, Vector2 direcao, float tamanho, LayerMask objeto)
    {
        Vector3 posicao = transform.position;
        RaycastHit2D hit = Physics2D.Raycast(posicao + origem, direcao, tamanho, objeto);

        Color cor = hit ? Color.red : Color.green;
        Debug.DrawRay(posicao + origem, direcao * tamanho, cor);

        return hit;
    }

    private void verificarFisica()
    {
        isNoChao = false;

        verificarEsquerda = verificarRayCast(new Vector2(pes[0].x, pes[0].y), Vector2.down, distanciaChao, layerChao);
        verificarDireita = verificarRayCast(new Vector2(pes[1].x, pes[1].y), Vector2.down, distanciaChao, layerChao);

        isNoChao = verificarEsquerda.collider != null || verificarDireita.collider != null;
    }

    private void LateUpdate()
    {
        animador.SetFloat("velocidadeX", Mathf.Abs(velocidadeX));
        animador.SetBool("isNoChao", isNoChao);
```

```
        animador.SetFloat("velocidadeY", rb2d.velocity.y);

        isAtacando = animador.GetCurrentAnimatorStateInfo(0).IsTag("Atacando");
    }
}
```

Mudar nas Settings das animações Parado e Correndo a prop Transition Duration para 0
Parado para Atacando = 0.1
Atacando para Parado = 0
Parado para Pulando = 0.1
Correndo para Pulando = 0
Entrar no PuloCompleto e remover o loop time dos dois itens

CENÁRIO

Vamos agora criar os cenários e objetos visuais do nosso jogo.

1. Criar dentro da pasta Assets uma nova pasta “Tiles”

2. Criar dentro da pasta Tiles uma pasta “Cenario”

2. Adicionar todas as imagens de cenário nessa pasta

3. Abrir a janela Tile Palette (Menu Window > 2D > Tile Palette)

4. Ir na opção “Create New Palette” e selecionar a pasta “Tiles”

5. Arrastar as imagens da pasta Cenario (dentro do Unity) para a janela

Agora temos a nossa paleta de objetos de cenário. Vamos adicionar o Chão ao cenário.

1. Para isso, clicar com o botão direito na Hierarchy > 2D object > Tile Map > Retangular

2. Renomear para Chao

3. Na Layer, definir Chao
Passo importante para que as propriedades do chão que foram definidas sejam associadas a esse novo item

4. Para controlar as colisões, adicionar um componente Tilemap Collider 2D

5. Como o nosso chão é composto por vários Tiles (cada objeto na tela é um Tile), precisamos adicionar também um Composite Collider 2D

6. Voltar no Tilemap Collider e marcar a opção “Used by composite”

7. No Rigidbody que foi adicionado, selecionar a opção “Static” no Body Type

Para fazer uma plataforma suspensa, repetir os mesmos passos, porém temos que incluir mais um componente:

Platform Effector 2D e no Composite Collider, marcar a opção “Used by effector”