**Criando um projeto**

• Abra o Unity Hub → **New Project**.

• Escolha o template **2D Core**.

• Defina o nome do jogo e a pasta onde ficará salvo.

**Adicionando um chão**

* Na **Barra de menu** (ou Menu Bar) → **GameObject**
  + 2D Object
    - Sprites → **Square**
      * Modele e altere a cor. Ex: **Scale** → **X** = **21** | **Y** = **1** | **Z** = **1**
* No **Hierarchy**
  + Selecione o Square
    - Clique com o botão direito sobre ele
      * Clique em **Rename**
        + Renomei-o. Ex: Chão
* Com o “Chão” selecionado, no **Inspector**
  + Clique em **Add Component** no fim da coluna do inspector
    - Adicione o **Box Collider 2D**
    - **Obs**: O **Box Collider 2D** cria a “hitbox” do objeto 2D.

**Um Player simples**

* Na **Barra de menu** (ou Menu Bar) → **GameObject**
  + 2D Object
    - **Sprites →** escolha uma forma. Ex: **Triangle**
      * Altere o tamanho e a cor. Ex: **Scale** → **X** = **1** | **Y** = **1** | **Z** = **1**
* No **Hierarchy**
  + Selecione o “Triangle”
    - Clique com o botão direito sobre ele
      * Clique em **Rename**
        + Renomei-o. Ex: Player
* Com o “Player” selecionado, no **Inspector**
  + Clique em **Add Component** no fim da coluna do inspector
    - Adicione o **Box Collider 2D**
    - E o **Rigidbody 2D**
    - **Obs**: O **Rigidbody 2D** adiciona massa, gravidade e inércia ao objeto.

**Posicionando a câmera**

* No **Hierarchy**
  + Selecione o “Main Camera”
    - No **Inspector**
      * Em **Transform**
        + Configure o **Position**

**X = 0** (Move Horizontalmente)

**Y = 5** (Move Verticalmente)

**Z = -10** (Define a distância em relação ao plano do jogo, mas não aumenta o alcance de visão)

* + - * Abaixo de Transform, em **Camera**
        + Altere o **Size**
        + Ex: **Size** | **5** | → **Size** | **10** |, para ampliar o alcance da câmera e vice e versa.

**Preparando script para movimentação**

* No canto inferior esquerdo
  + Em **Project** 
    - Na pasta ***Assets*** crie uma pasta. Ex: **Scripts** (Para melhor organização)
      * Selecione **Scripts** e clique com o botão direito sobre ele
        + Selecione **Create** → **Scripting** → **Empty C# Script**

Nomeie o script. Ex: **PlayerMovement.cs**

Clique duas vezes para abrir o código no VSCode

**Estutura Basica do código em C#**

using UnityEngine;

//using → serve para importar bibliotecas, assim como o import do java.

//UnityEngine → é a biblioteca principal do Unity, onde estão as classes como: GameObject, MonoBehaviour, Transform, Rigidbody2D, etc.

public class PlayerMovement

{

//public → é um modificador de acesso. Exemplos: public, private, protected.

    //class → palavra reservada para criar uma classe em C#.

    //PlayerMovement → é o nome da sua classe. Mesmo nome do script.

}

**Adicione o MonoBehaviour à classe**

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

// : Significa que a classe PlayerMovement herda todos os atributos funçoes da classe a seguir, neste caso, o MonoBehaviour.

//MonoBehaviour → permite que um script se torne um componente do Unity, podendo ser adicionado a GameObjects e reagir a eventos da cena.

}

Com o **MonoBehaviour** declarado à classe, podemos adicionar o script ao player na Unity.

* Em **Project**, acesse **Assets/Scripts/**
  + Clique e segure o arquivo **PlayerMovement**
  + E arraste para cima do seu GameObject, “Player”.

**Informações Importantes – MonoBehaviour**

Os GameObjects que possuem scripts com MonoBehaviour posuem um ciclo de vida, sendo eles:

* **Awake()**
* **Start()**
* **Update()**
* **FixedUpdate()**
* **OnDestroy()**

A Unity Procura por métodos com essas assinaturas e os chamam automaticamente em momentos específicos:

**Awake()** → Executado uma vez quando o objeto é instanciado ou carregado na cena. Usado para inicializar variáveis e referências, mesmo que o objeto esteja desativado.

**OnEnable()** → Chamado sempre que o objeto ou componente é ativado (inclusive após ser reativado).  
   
**Start()** → Executado na primeira vez em que o objeto é ativado, logo antes do primeiro **Update()**. Usado para inicializações que dependem de outros objetos já existirem na cena.  
   
**Update()** → Chamado a cada frame. Usado para lógica de jogo em tempo real, como leitura de inputs, movimentação simples e animações que não envolvem física.  
   
**LateUpdate()** → Chamado após todos os **Update()**. Bom para lógica que precisa acontecer depois que outros objetos já se moveram (ex.: câmera seguir o jogador).  
   
**FixedUpdate()** → Chamado em intervalos fixos de tempo, independente do FPS. Usado para manipulação de física e movimentação de objetos com Rigidbody.  
   
**OnDisable()** → Chamado quando o objeto ou componente é desativado ou quando a cena é descarregada.  
   
**OnDestroy()** → Chamado quando o objeto é destruído permanentemente

(ex.: Destroy(gameObject)).

**Capturando entradas do jogador**

adicione o **Update()** do ciclo de vida do monoBehaviour

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

void Update(){

// O Update() é um dos métodos do ciclo de vida do MonoBehaviour.

// Ele faz com que a Unity atualize o comportamento do objeto a cada frame.

}

}

adicione a classe **Input** da UnityEngine, responsável por capturar entradas do jogador

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

void Update(){

float move = Input.GetAxis("Horizontal");

//float → tipo da variável

//move → nome da variável

//Input → classe da UnityEngine que capturar entradas do jogador (teclado, mouse, joystick, touch etc.).

//GetAxis() → Método da classe Input onde declaramos qual eixo queremos capturar, "Horizontal" ou "Vertical".

}

}

**Input.GetAxis()** faz parte do sistema antigo de Input do Unity, chamado **Input Manager**. Isso pode dar erro, pois a Unity por padrão usa o novo sistema, o **Input System Package.** Para evitar erro, siga os passos a seguir:

* **Edit**
  + **Project** **Settings**
    - **Player**
      * **Other Settings**
        + **Active Input Handling**
        + Altere: De **Input System Package(New),** para **Both**

**Adicionando movimentação horizontal**

void Update(){

float move = Input.GetAxis("Horizontal");

transform.Translate(Vector2.right \* move);

//transform → chama diretamente o transform, que pode ser visto no inspector (posição, rotação, escala).

//Translate() → método do transform que move o objeto (soma um vetor à sua posição atual).

//Vector2 → Vector2 é um tipo de variavel do C# no Unity, que representa um vetor de duas Dimensões (Direção e Magnitude da deslocação).

//right → determinha que movimentções a direita serão respresentadas por numeros positivos no vetor. E left, negativos.

//move → a variável com Input.GetAxis() que captura a entrada do jogador.

}

**Aplique o** **deltaTime**

void Update(){

float move = Input.GetAxis("Horizontal");

transform.Translate(Vector2.right \* move \* Time.deltaTime);

//Time → classe do Unity que fornece informações sobre o tempo do jogo.

//deltaTime → calcula o tempo que passou desde o último frame, em segundos.

}

**Obs**: O **Time.deltaTime** Multiplicado pelo **movimento** (“move”), converte o movimento de “por frame” para “por segundo”, garantindo que o objeto se mova **na mesma velocidade**, independente do FPS do computador, deixando a movimetação mais estavél e uniforme.

**Configure a velocidade**

* Crie uma variável global do tipo float e atribua um valor. Ex: **float speed = 15f**
  + Depois multiplique pelas variáveis já existentes no **transform.Translate()**

using UnityEngine;

public class PlayerMovement : MonoBehaviour

{

float speed = 15f;

    //Declaração da variável "speed" do tipo float

//15f → Valor da velocidade de movimento

void Update(){

float move = Input.GetAxis("Horizontal");

transform.Translate(Vector2.right \* move \* Time.deltaTime \* speed);

//speed → Variável que determina a velocidade de movimentação do objeto.

}

}