

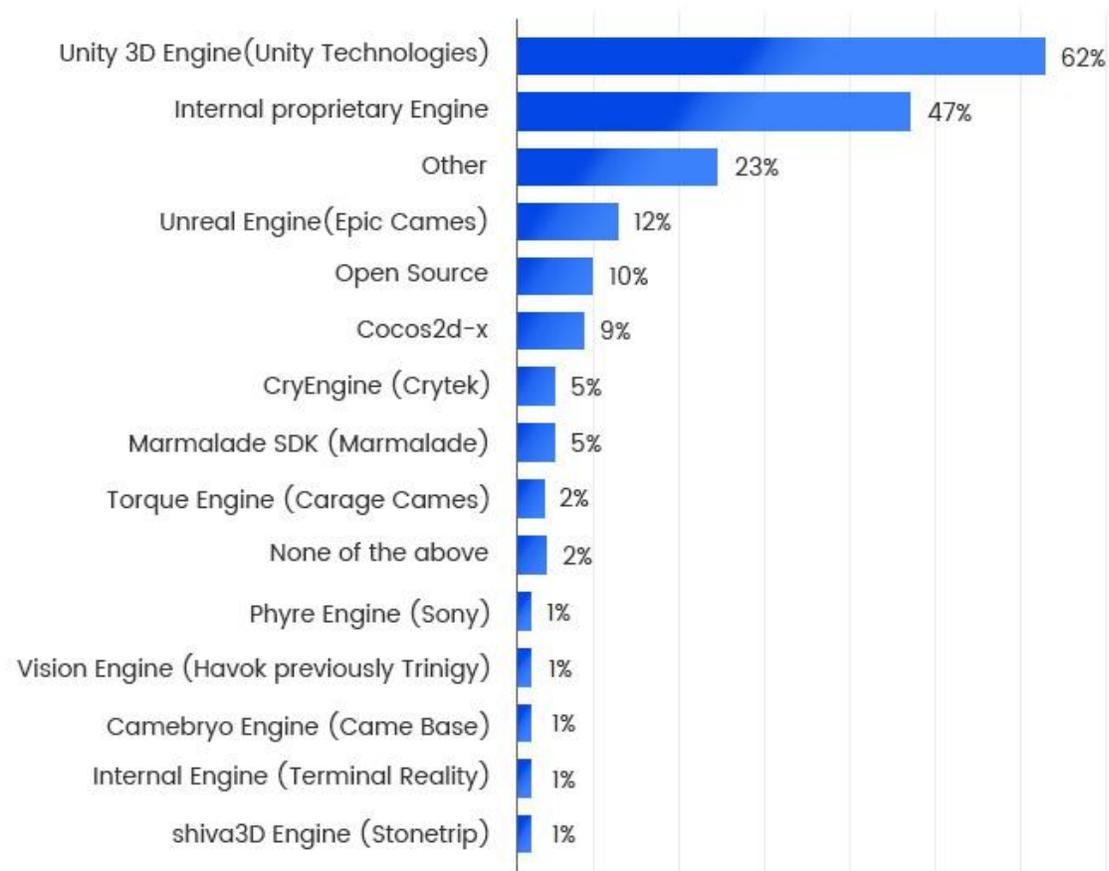
# Introdução ao Unity3d, conceitos fundamentais de física e *Game Engine*

Prof. Me. Hélio Esperidião

# Unity

---

- Fatia de mercado mundial para o desenvolvimento de jogos.
- 71% dos 1.000 jogos mobile mais distribuídos são feitos com Unity.



## Unity

Mobile games made with Unity account for **71%** of the top 1000 titles on the market

The user base (developers + users) amounts to over 2.7 billion people

Asset Store features over 65K assets

Revenue increased by 43% in 2020

Market capitalization is approximately \$27.8 billion.

## Unreal Engine

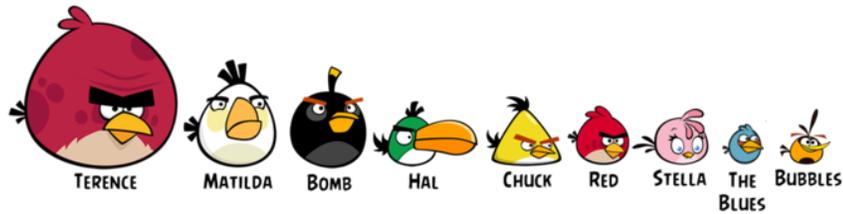
Over 2 million games powered by the engine

Marketplace features over 16K assets

Developer community exceeds **7 million** people

Market share is estimated at 13%

Epic Games' (parent company) market cap is approximately \$17 billion.



# Popularidade do Unity

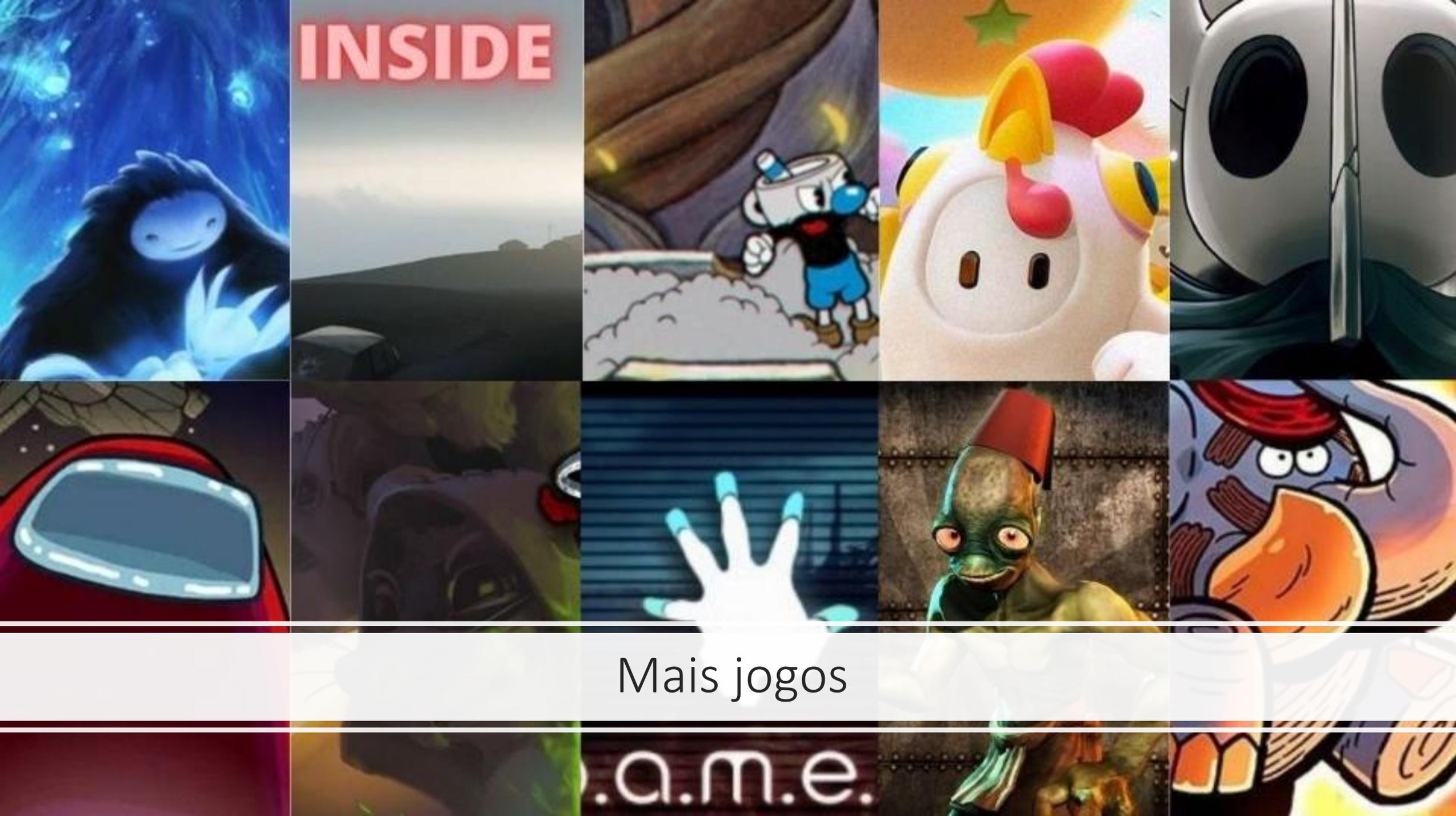
- Essa enorme popularidade faz jus à sua capacidade: a game engine permite criar jogos em 2D ou 3D com os mais diversos estilos de gráficos e mecânicas e para diferentes plataformas.
- Muitos jogos famosos, como Angry Birds 2, Bad Piggies, Roller Coaster Tycoon World e até o Pokemon GO foram criados com ela



# Mais Jogos criados com Unity



- Assassin's Creed: Identity
- Temple Run Trilogy
- Deus Ex: The Fall
- Ballistic Overkill
- Knights of Pen and Paper



INSIDE

Mais jogos

.a.m.e.



# Game Engine

01

Game engine (ou, em português, motor de jogo), consiste em um conjunto de ferramentas capazes de facilitar o desenvolvimento de um jogo.

02

Geralmente possuem recursos para criação de funções gráficas, física aos objetos, trilhas sonoras, entre outras ações.

# Game engine

---



Incorpora um motor gráfico para processar gráficos em 2D ou 3D



Incorpora motor de física para identificar colisões e realizar animações, além de oferecer suporte para áudio, inteligência artificial, gerenciamento de arquivos, programação, entre outras funcionalidades.



Isso implica que a criação de um jogo a partir do zero se torna mais acessível.



As engines de jogos possibilitam que uma pessoa, por conta própria, desenvolva um jogo que, nas décadas de 80, exigiria uma equipe completa.

# Década de 80

- Na década de 80, os desenvolvedores de jogos para consoles populares como o Atari enfrentavam a árdua tarefa de codificar manualmente todos os elementos necessários para criar pixels, movimentos e efeitos físicos.
- Esse processo era extremamente trabalhoso e pouco intuitivo exigindo uma equipe considerável para criar os primeiros jogos em 2D.
- Devido às limitações de hardware, o código precisava ser o mais simples possível, e a manutenção da limpeza do código escrito à mão era um desafio adicional.
- Todo esse extenso trabalho tornava-se obsoleto rapidamente, pois a constante evolução do hardware exigia a reescrita de código para cada novo jogo.

# Por que é o mais usado?

Facilidade de aprendizado

Muita documentação e fórum de discussões na internet

Cria/portar games para várias plataformas

Versão gratuita

Crie jogos do Sistema Operacional que preferir

Jogos em 2D e 3D

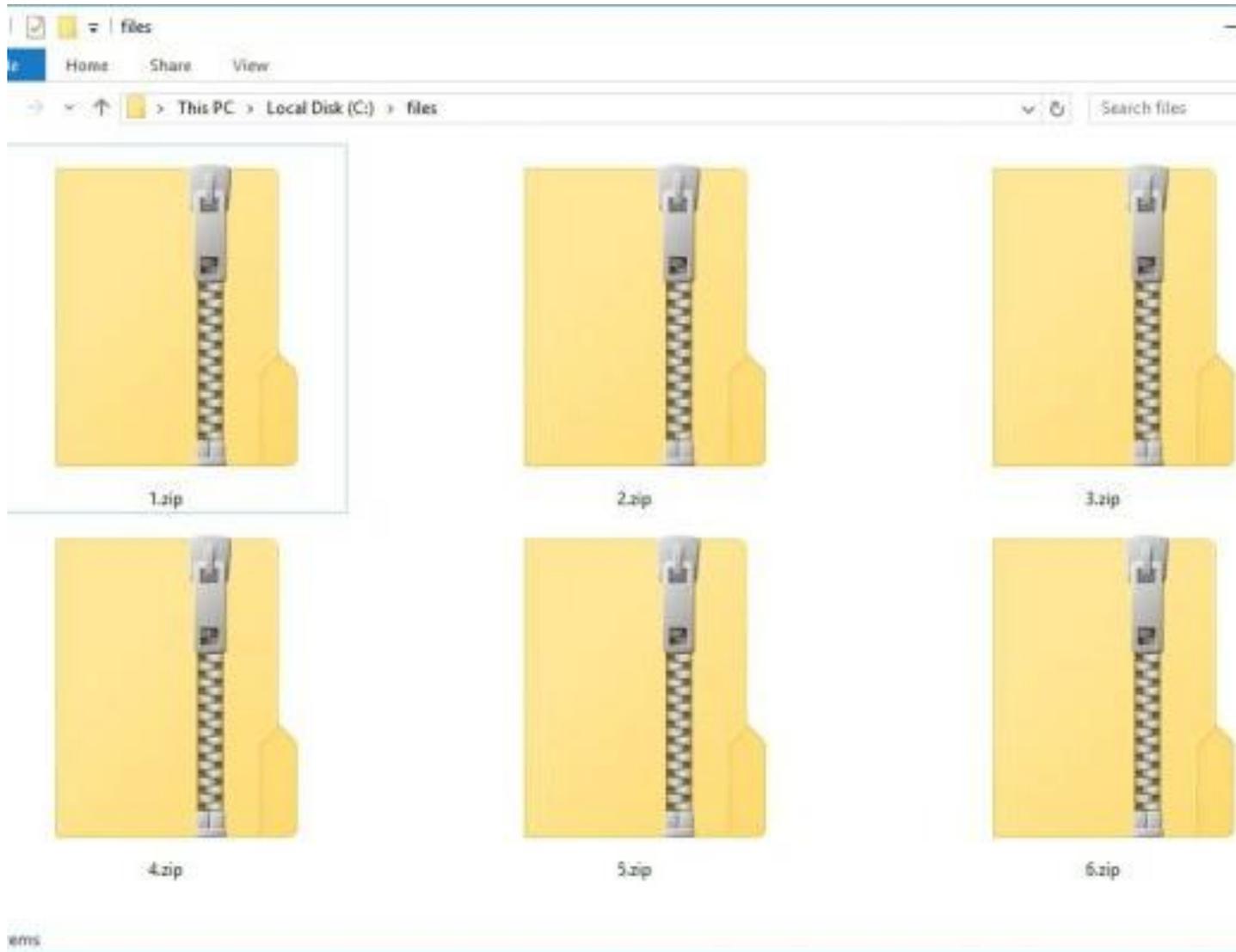
# Assets



- **Asset significa: Ativo.**
- Em tecnologia os "assets" são os recursos do seu projeto. O seu "banco" de bibliotecas.
- No caso de jogos pode ser entendido como o conjunto de imagens, gráficos, sons e recursos que são utilizados para o desenvolvimento do jogo.

# Onde encontrar?

- Os assets podem ser construídos pelo próprio desenvolvedor ou encontrados na web nas seguintes formas de distribuição: Gratuito ou pago.
- Site com conteúdo gratuito:
  - <https://www.gameart2d.com/freebies.html>

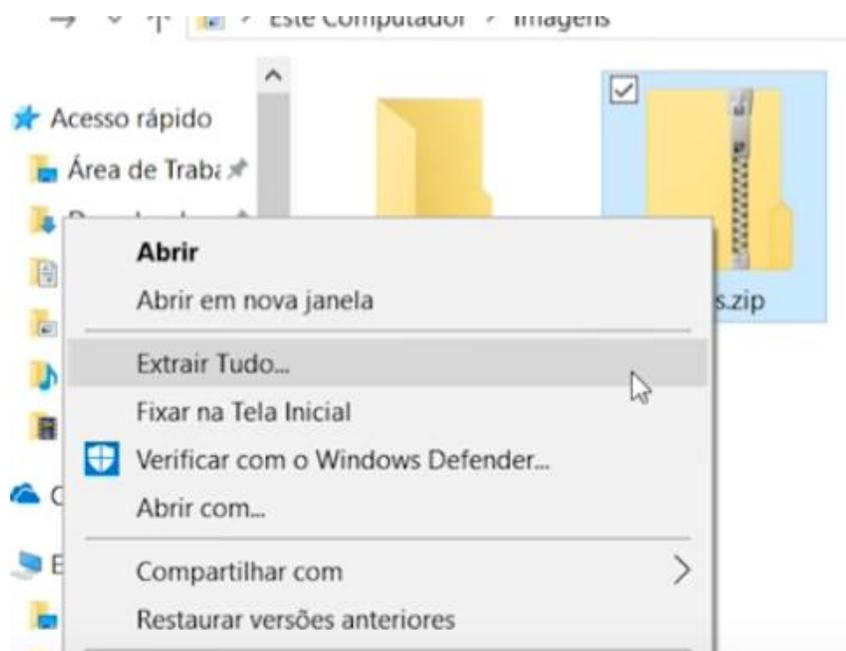


## \* Arquivos compactados (.zip)

- É bem provável que os assets que você baixe estejam compactados ou zipados.
- Arquivos zipados (compactados) ocupam menos espaço de armazenamento e podem ser transferidos para outros computadores mais rapidamente do que arquivos não compactados.

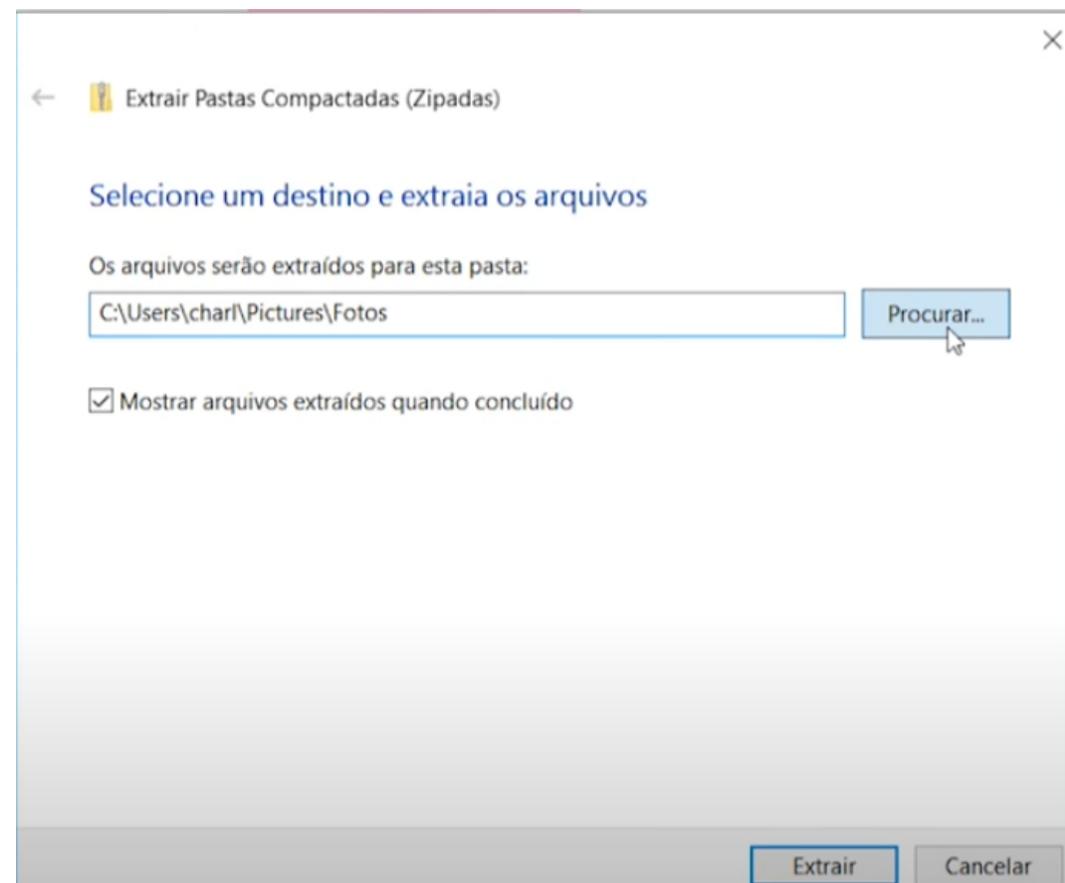
## \* Arquivos compactados (.zip)

- <https://www.youtube.com/watch?v=pg7QWAsUvzA>
- Para descompactar click no arquivo .zip com o botão direito do mouse e escolha a opção “Extrair Tudo”



# \* Arquivos compactados (.zip)

- Selecione uma pasta para salvar os arquivos que serão descompactados
- Depois de escolher uma pasta click em “Extrair”



# Instalação do unity

- Obrigatoriamente instale em sua casa a mesma versão que está instalada na escola:
  - **2017.4.40f1**
- Link para download: <https://unity.com/releases/editor/archive>
- Download direto:  
[https://download.unity3d.com/download\\_unity/6e14067f8a9a/Windows64EditorInstaller/UnitySetup64-2017.4.40f1.exe](https://download.unity3d.com/download_unity/6e14067f8a9a/Windows64EditorInstaller/UnitySetup64-2017.4.40f1.exe)

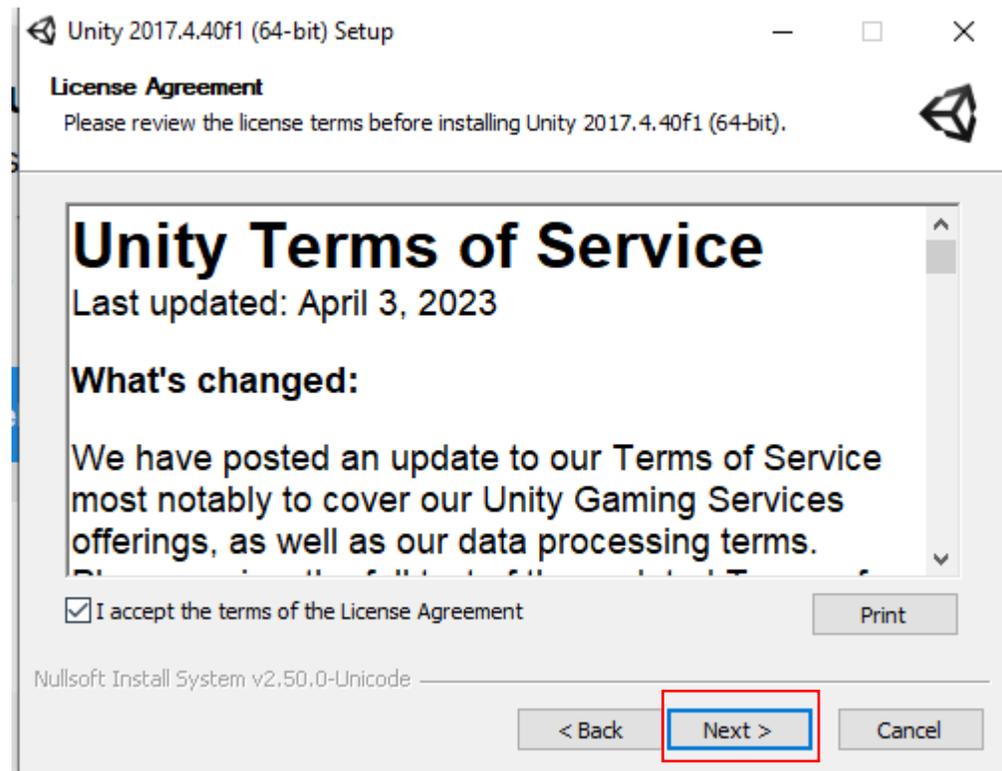
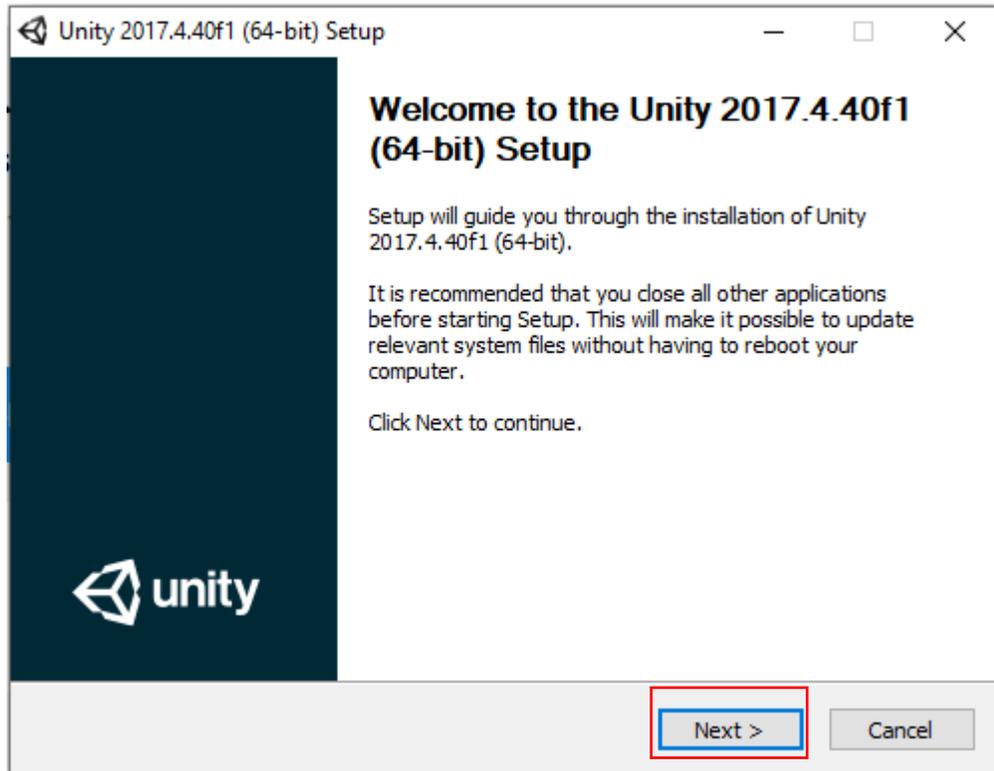
# Versão do Unity: 2017.4.40f1

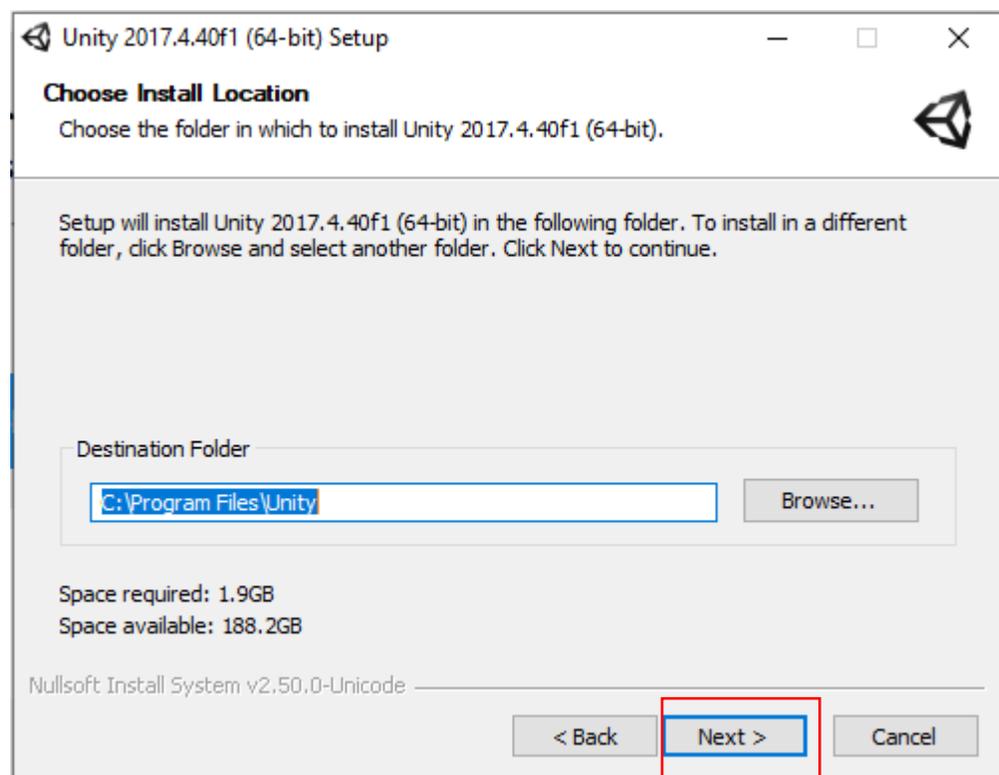
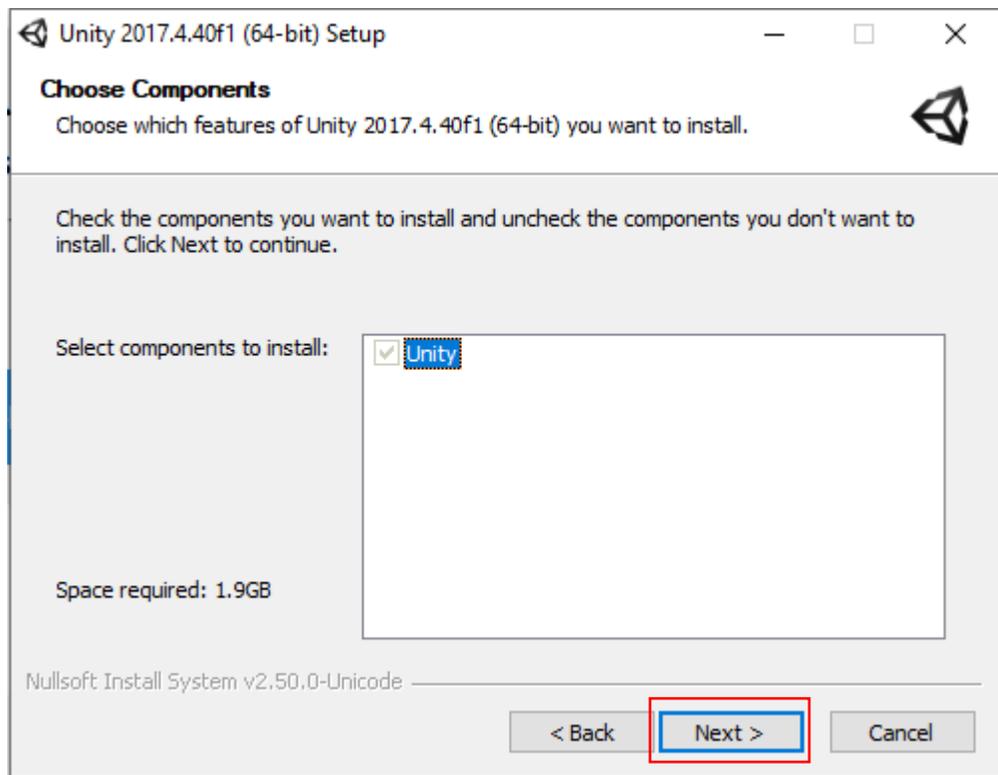
- É possível fazer o download diretamente do site
- Instale o **Unity Editor 64-bit**

The screenshot shows the Unity website's version selection interface. At the top, there are tabs for different Unity versions: Unity 2023.X, Unity 2022.X, Unity 2021.X, Unity 2020.X, Unity 2019.X, Unity 2018.X, **Unity 2017.X** (selected), and Unity 5.X. Below the tabs, there are three rows of version information. The first row is for Unity 2017.4.40 (released May 18, 2020), the second for Unity 2017.4.39 (released April 3, 2020), and the third for Unity 2017.4.38 (released March 20, 2020). Each row includes a 'Unity Hub' button, a dropdown menu for operating systems (Windows, Mac, Linux), and a 'Release Notes' button. The 'Downloads (Win)' dropdown for the 2017.4.39 version is open, and the 'Unity Editor 64-bit' option is highlighted with a red box. Other options in the dropdown include 'Unity Installer', 'Built in shaders', 'Unity Accelerator', and 'Torrent download (Win+Mac)'.

Unity Version	Release Date	Unity Hub	Downloads (Win)	Downloads (Mac)	Downloads (Linux)	Release Notes
Unity 2017.4.40	Maio 18, 2020	Unity Hub	Downloads (Win) ^	Downloads (Mac) v	Downloads (Linux) v	Release Notes
Unity 2017.4.39	Abril 3, 2020	Unity Hub	Unity Editor 64-bit	Downloads (Mac) v	Downloads (Linux) v	Release Notes
Unity 2017.4.38	Março 20, 2020	Unity Hub	Built in shaders	Downloads (Mac) v	Downloads (Linux) v	Release Notes

# Instalação Unity passo a passo

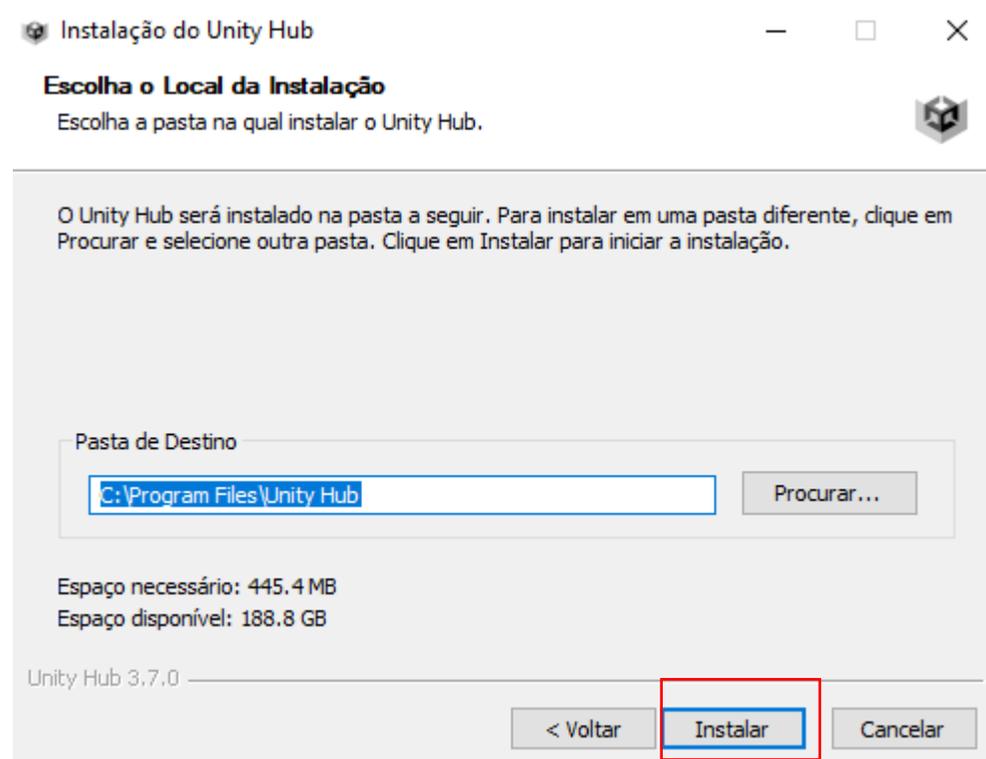
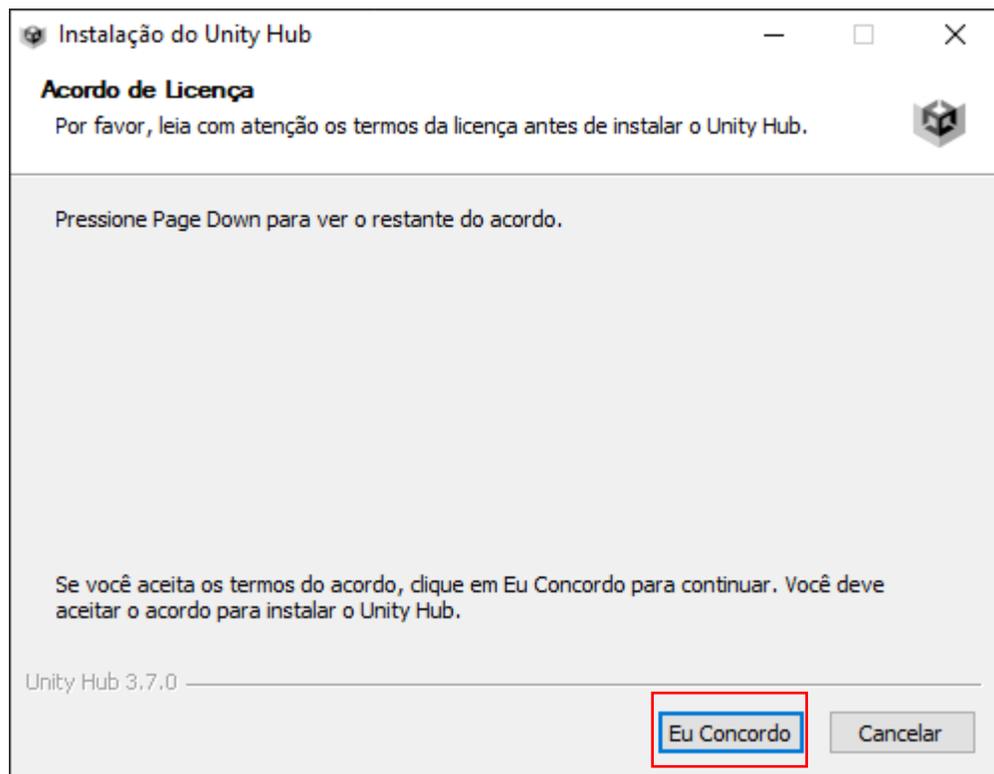




# Instalar o Unity HUB(2024)

- Link para download: <https://public-cdn.cloud.unity3d.com/hub/prod/UnityHubSetup.exe>

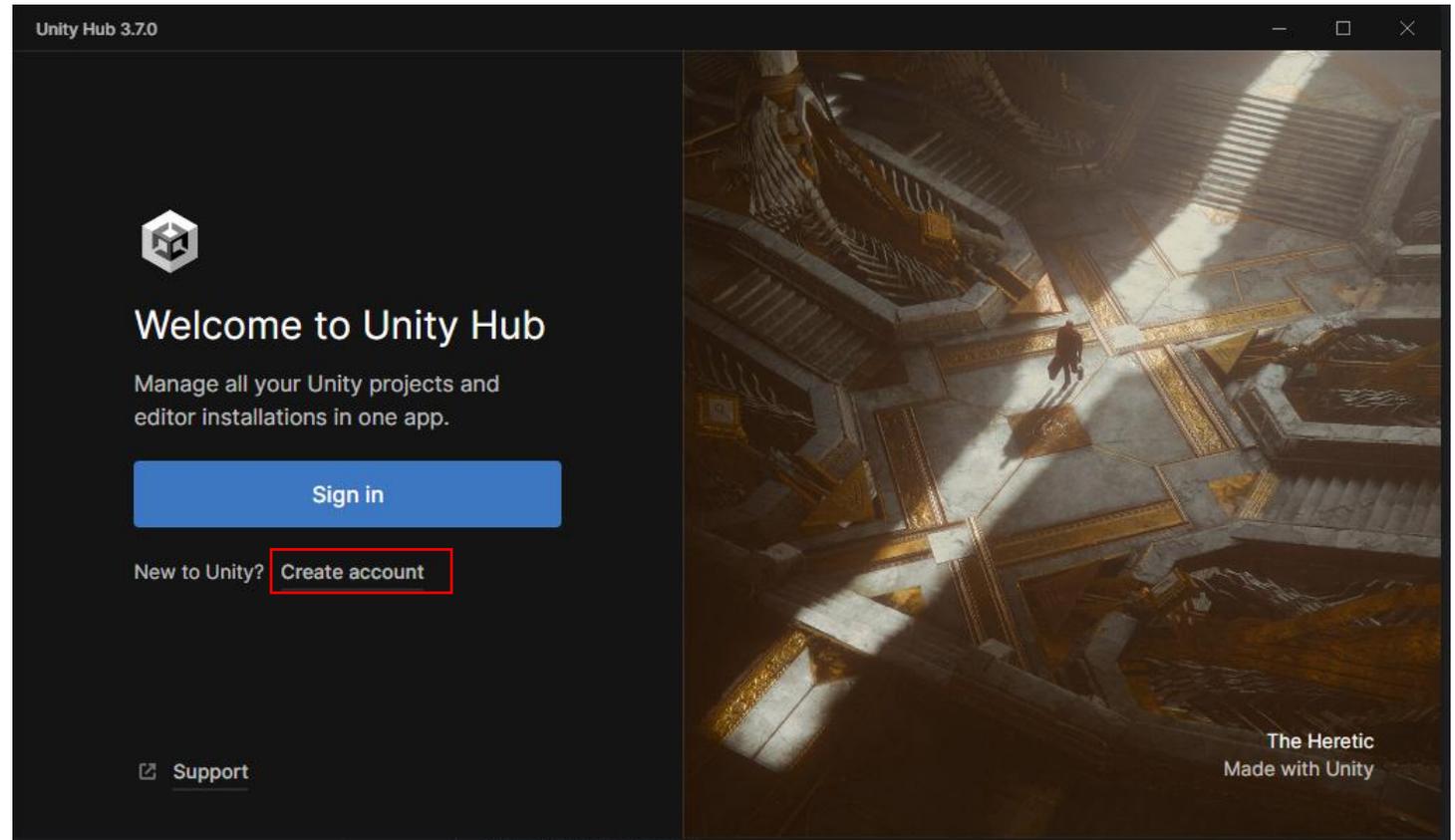
# Instalação do Unity hub



# Faça login

---

- Procedimento necessário para executar o Unity **NO SEU COMPUTADOR**.
  - Faça login ou crie uma conta
- Os laboratórios da escola já estão configurados.



Só é necessário para instalar e utilizar o Unity no seu computador.

Faça login ou crie uma conta

## Sign into your Unity ID

If you don't have a Unity ID, please [create one](#).

Email

Password

Remember me

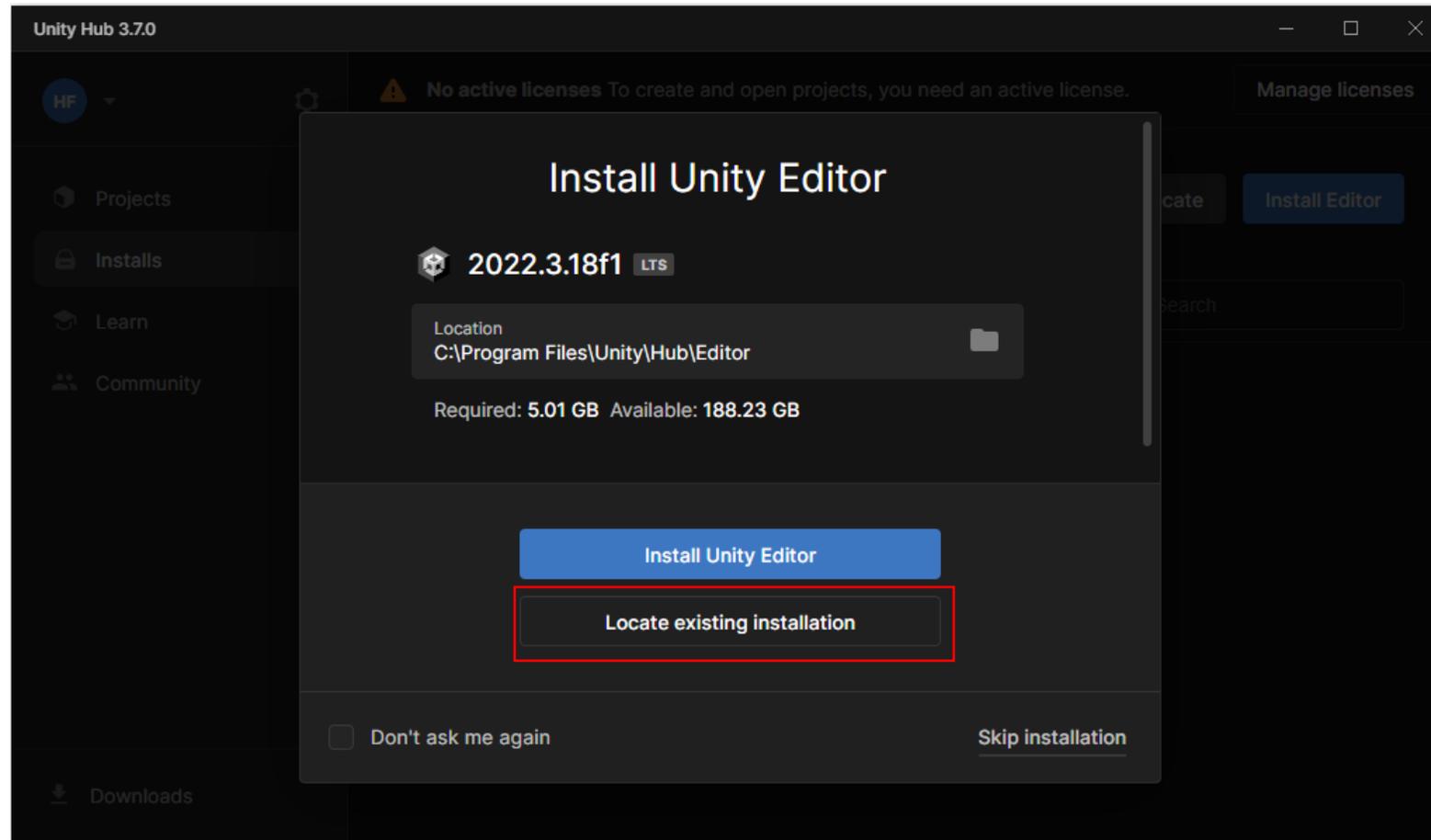
[Forgot your password?](#) [Help](#)

Sign in

OR



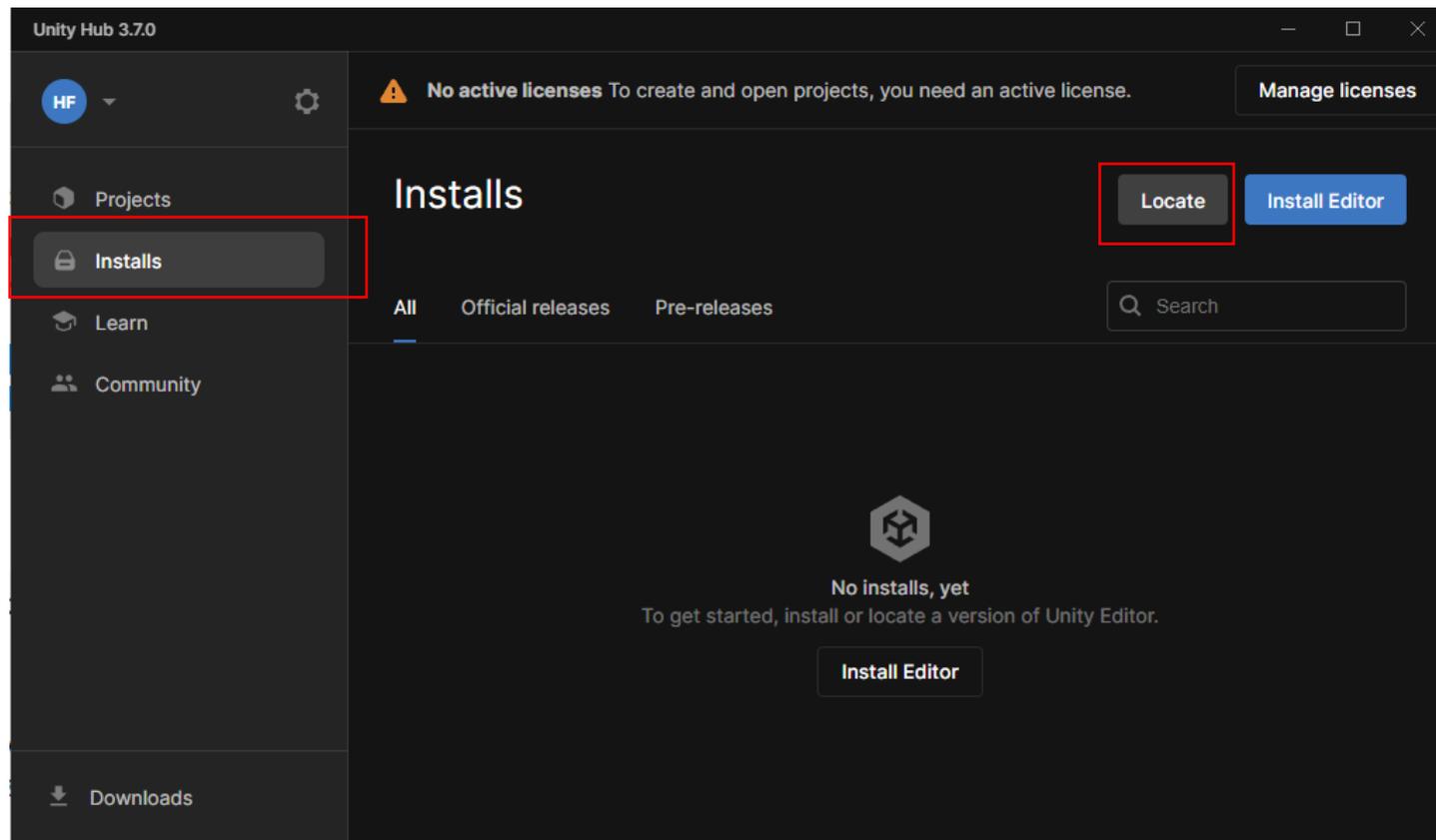
# Locate existing installation

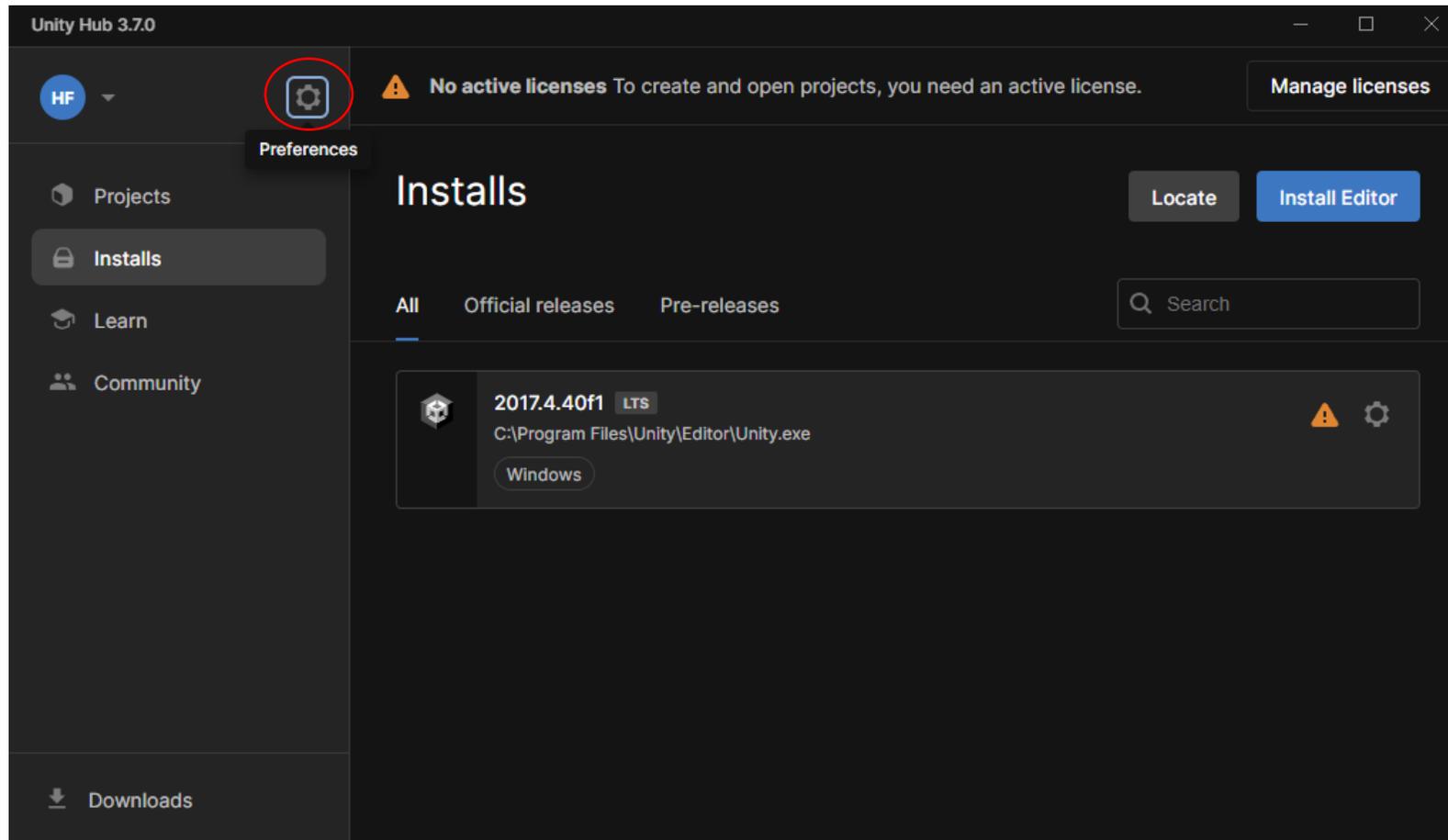


# Versão Unity hub 2024

---

- Vá no menu installs.
- Click no botão **Locate**
  - Selecione o arquivo “C:\Program Files\Unity\Editor\unity.exe”



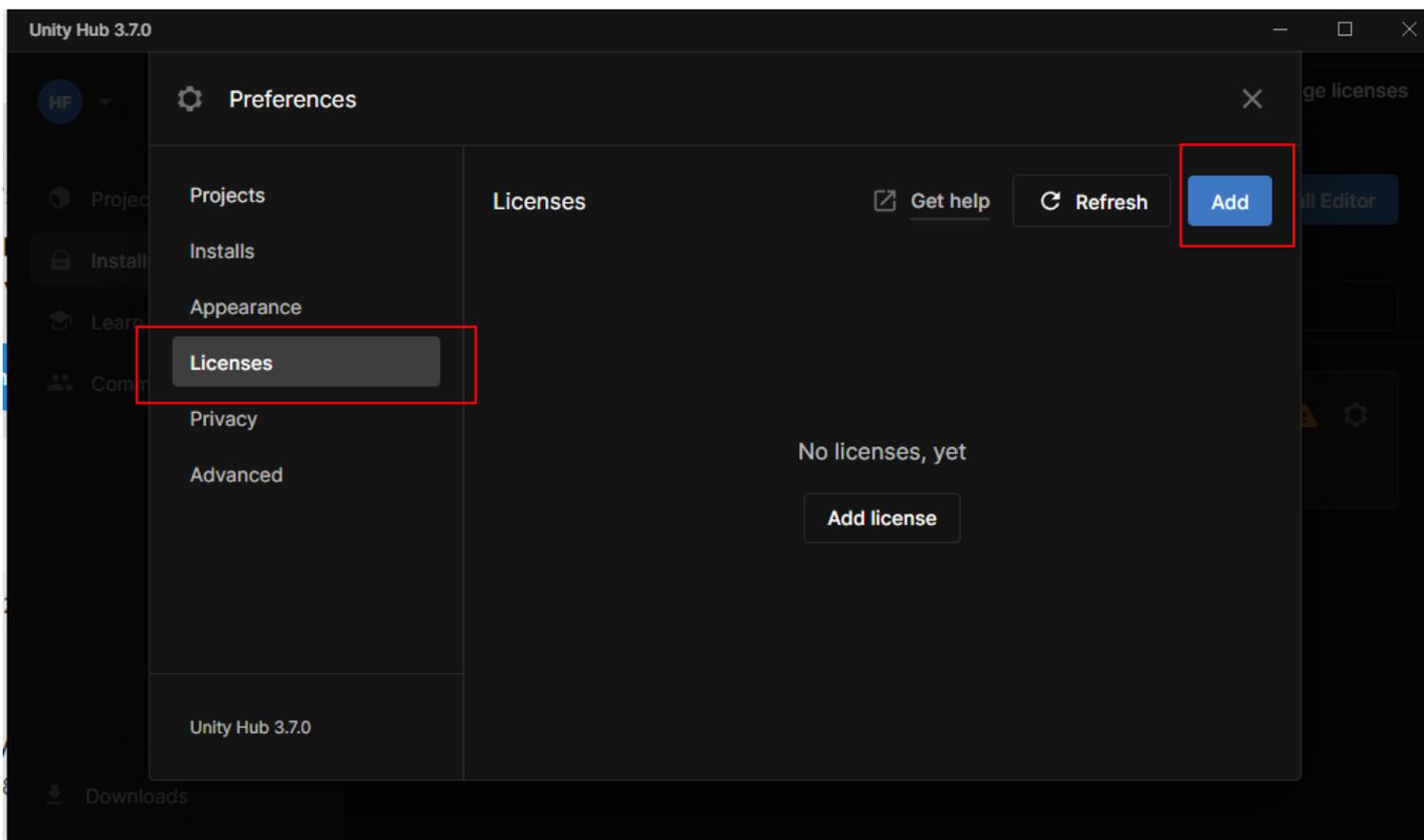


Ative uma  
licença do  
unity

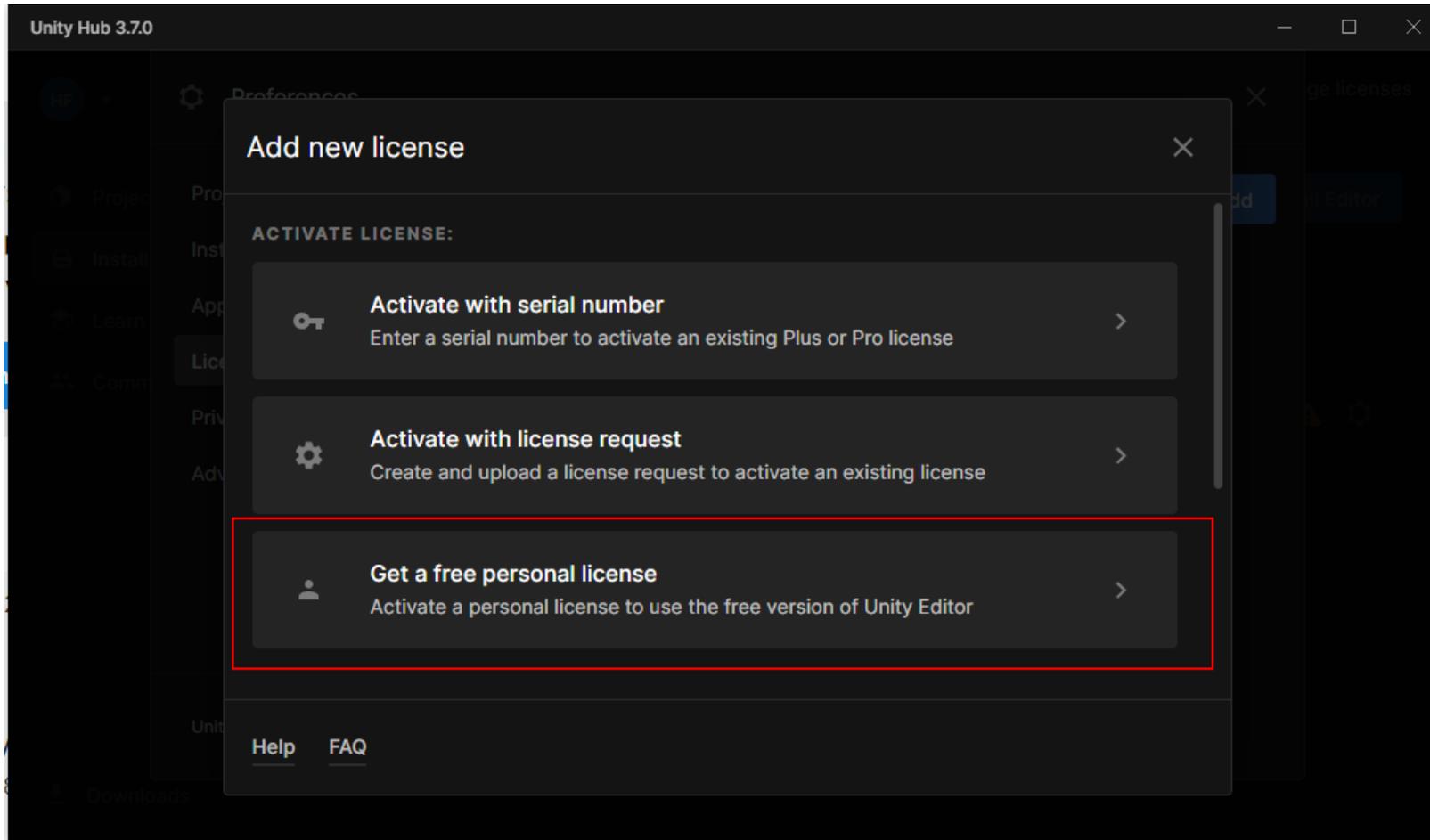
Clique na engrenagem

# Escolha a opção “Licenses”

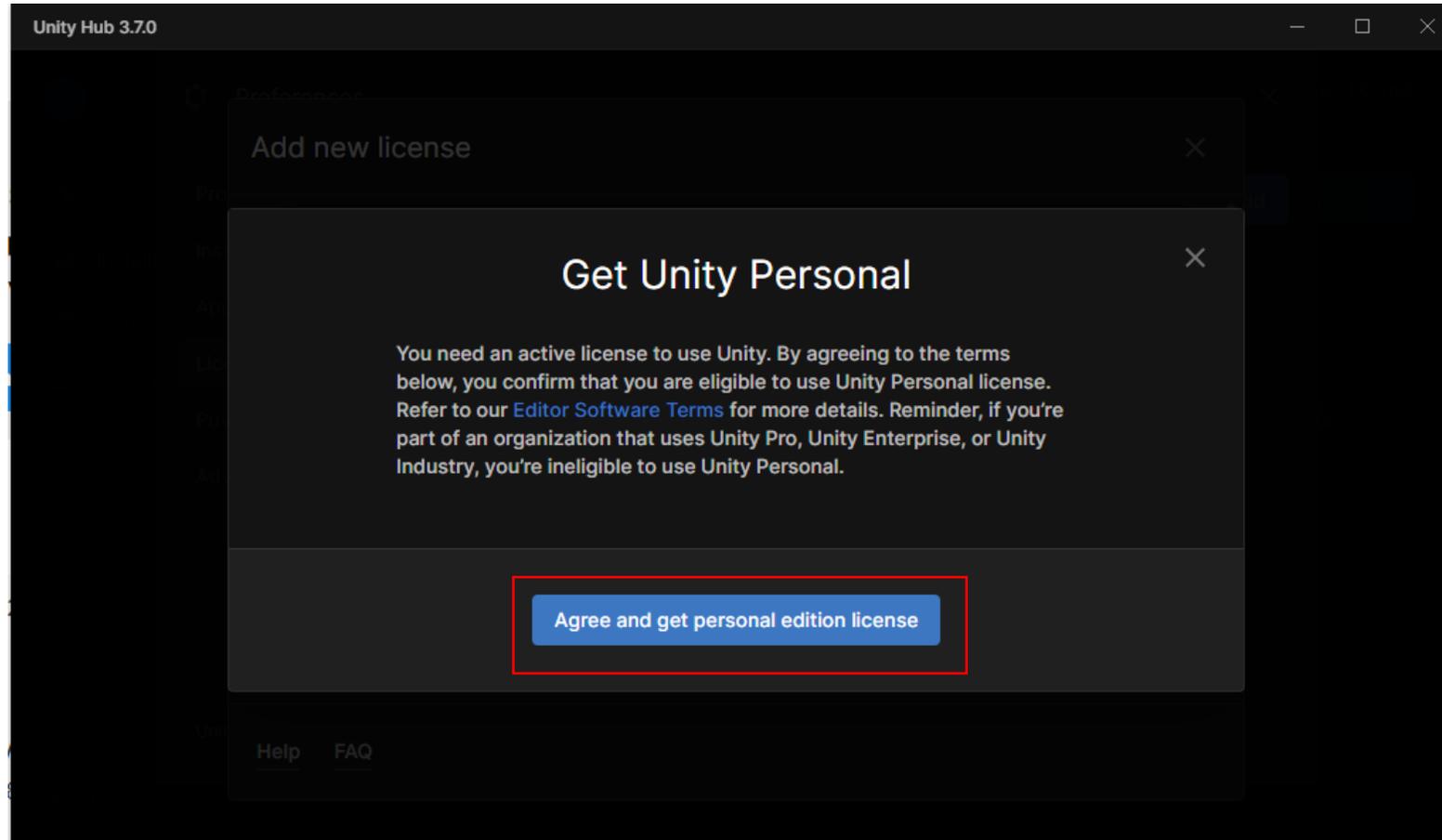
- Clique no botão “add”



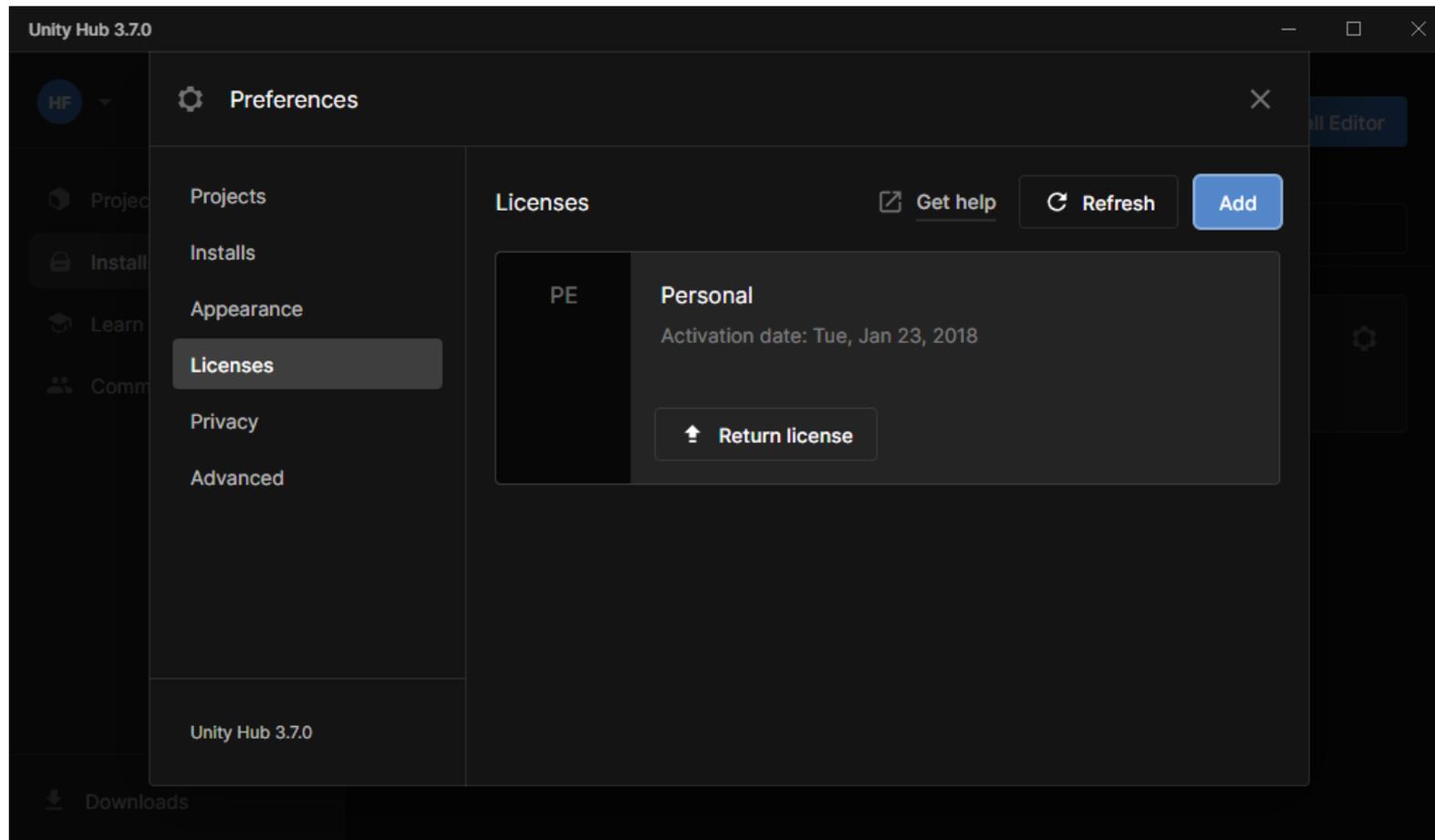
Escolha:  
“Get a free personal license”



# Aceite os termos de uso

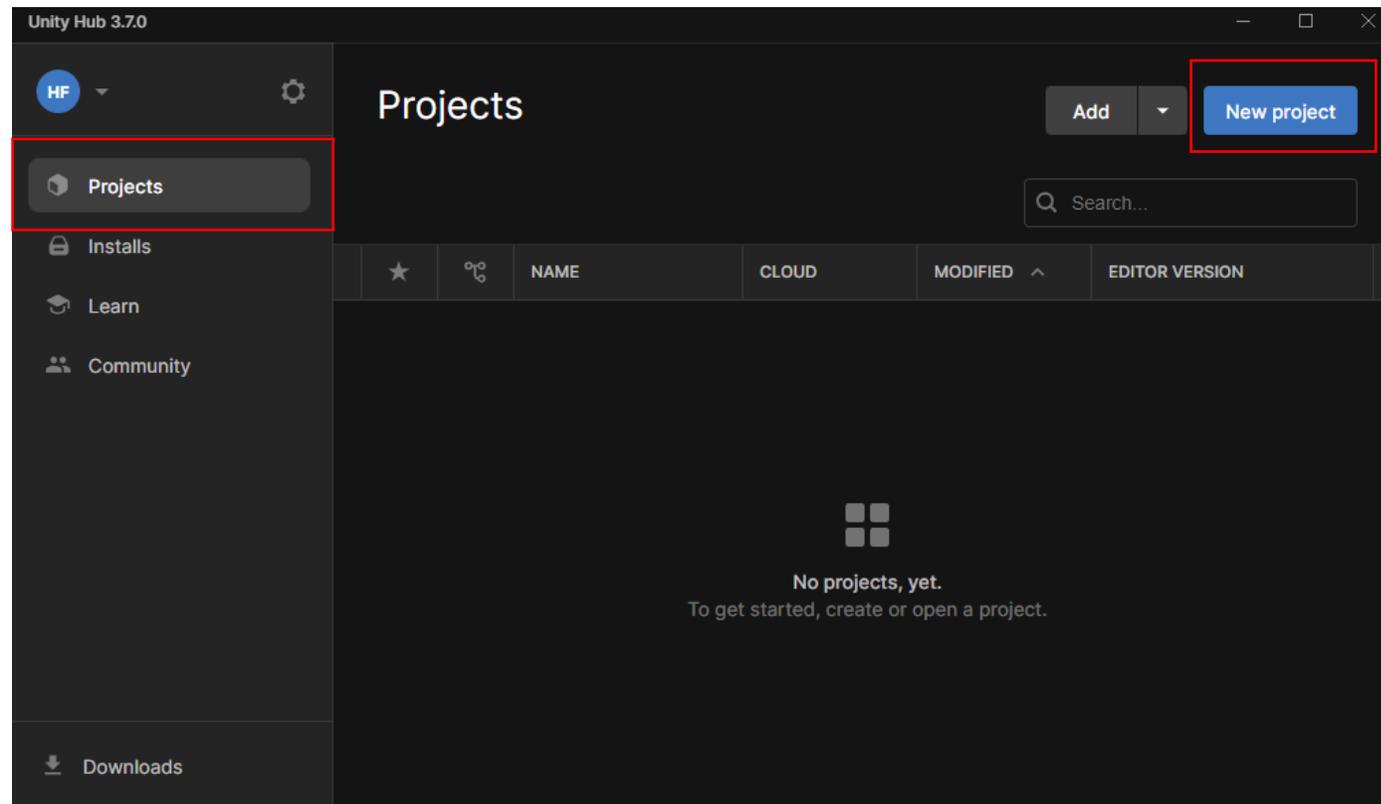


# Licença ativada



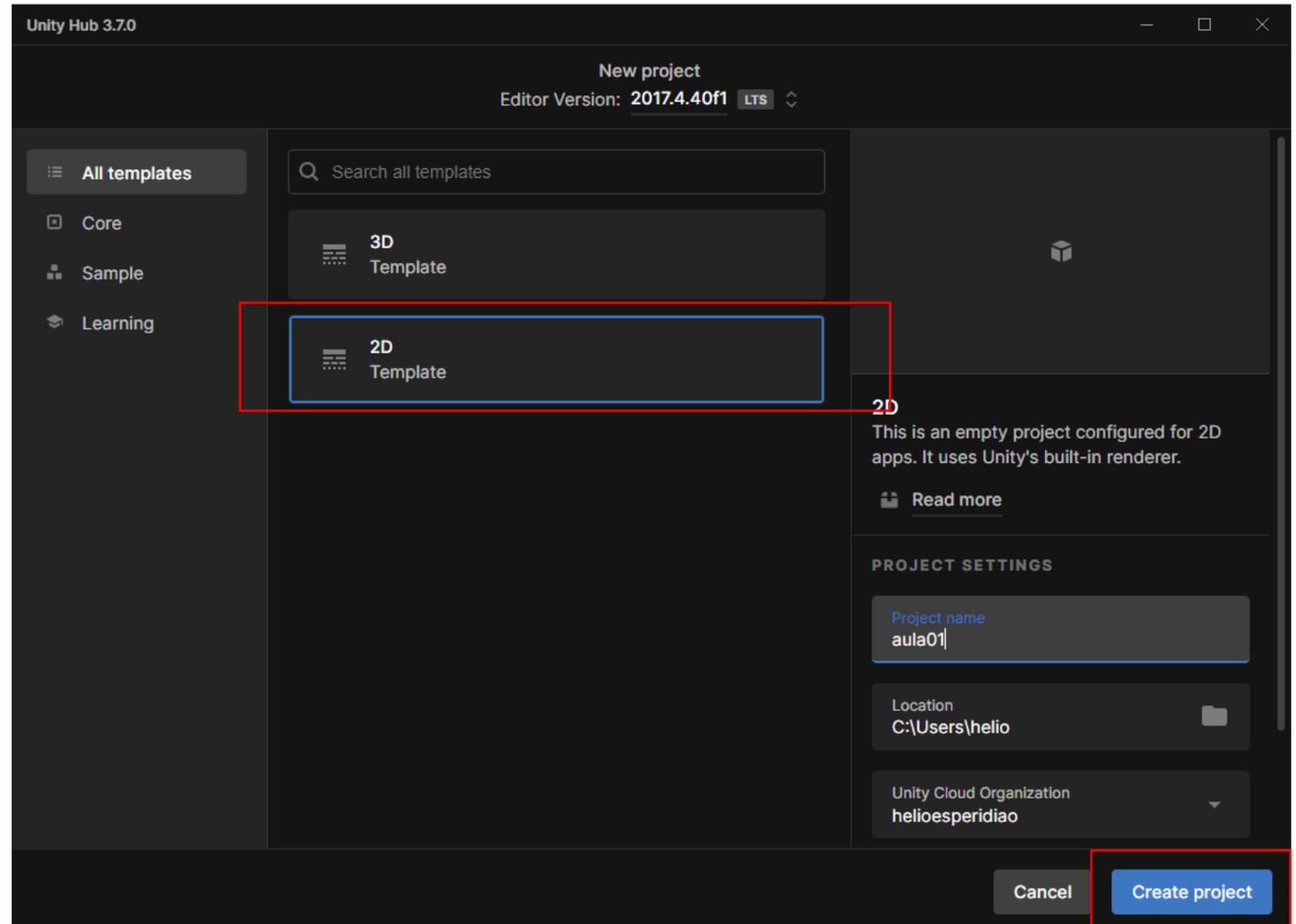
# Criar um novo projeto

- Selecione “Projects”
- Clique no botão “new Project”



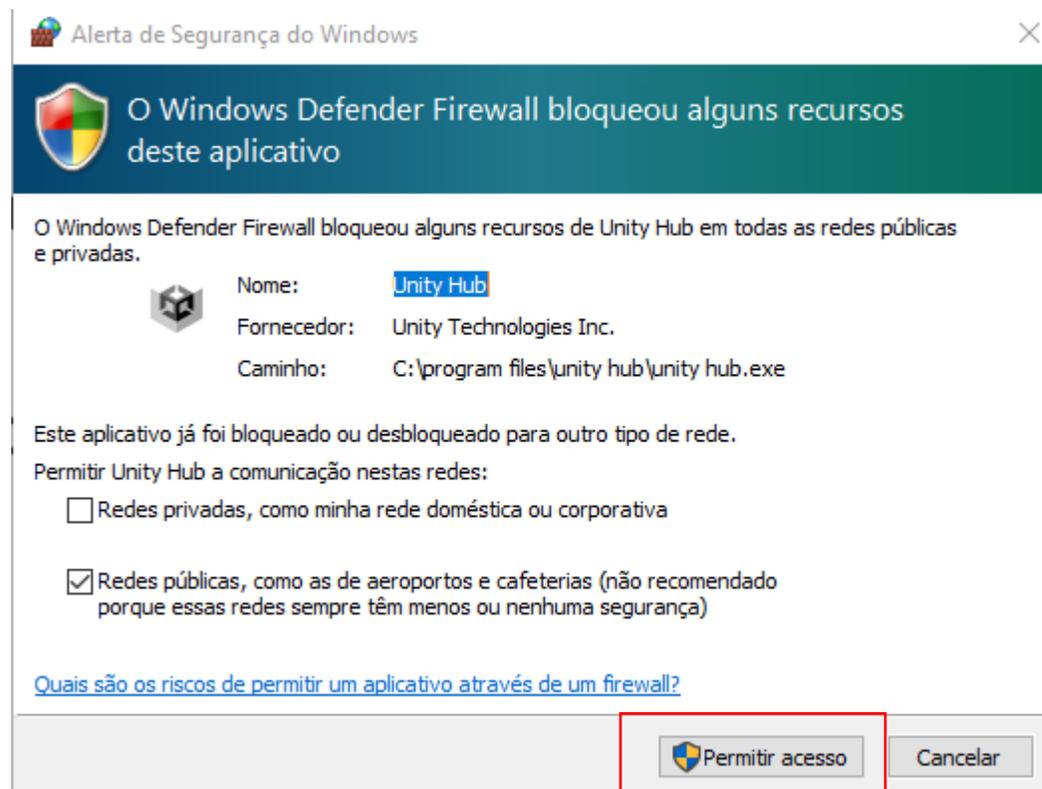
# Defina o template 2d

- Escolha o nome para o projeto
- Escolha o local dos arquivos



# Libere o acesso no firewall.

- Permita se for solicitado.

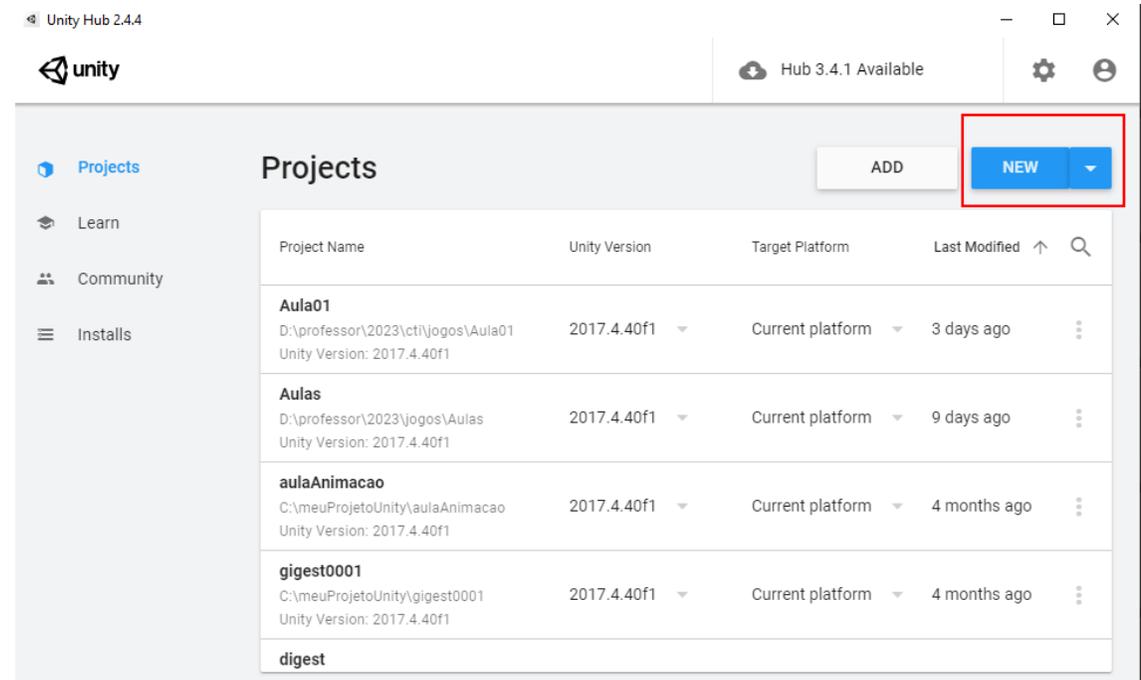


**FAÇA NO LABORATÓRIO:**

# Abra o Unity hub

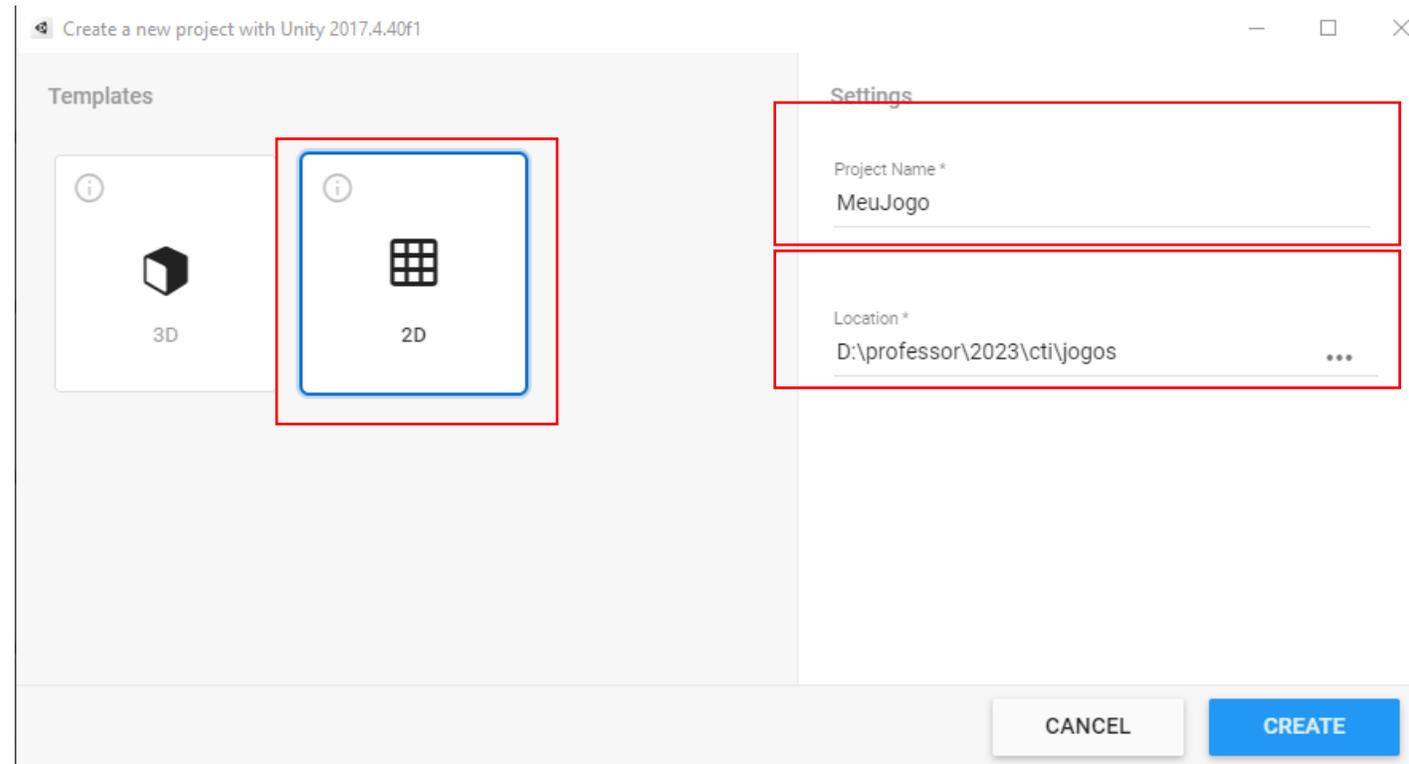
## Criar um novo projeto

- A versão do Unity hub instalada nos laboratórios e ligeiramente diferente
- Click no botão Azul **“NEW”**



# Configurações do projeto

- Escolha a opção 2D
- Defina um nome para o projeto em "Project name"
- **"Location:" Defina onde todos os arquivos do projeto serão salvos.**
- **No exemplo os arquivos serão salvos em:**
- **D:\professor\2023\cti\jogos\MeuJogo**
- **Portanto salve o conteúdo inteiro da pasta "MeuJogo"**
- O projeto é composto de muitos arquivos
- Todos os arquivos e pastas devem ser salvos



# Configurando um editor de código

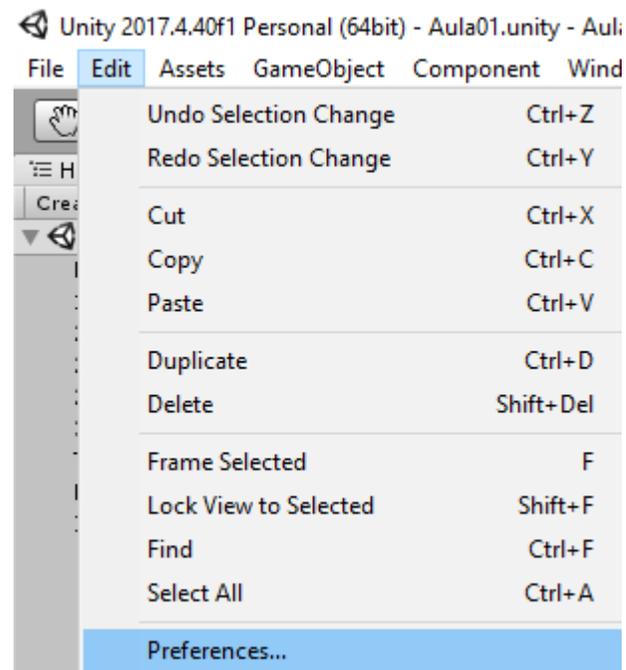
- Não é preciso fazer essa configuração no laboratório
- Você pode escolher seu editor favorito.
  - Editores mais comuns:
    - Mono Develop
    - Visual Studio
    - Visual studio code.

# Configurar o editor de códigos: monoDevelop



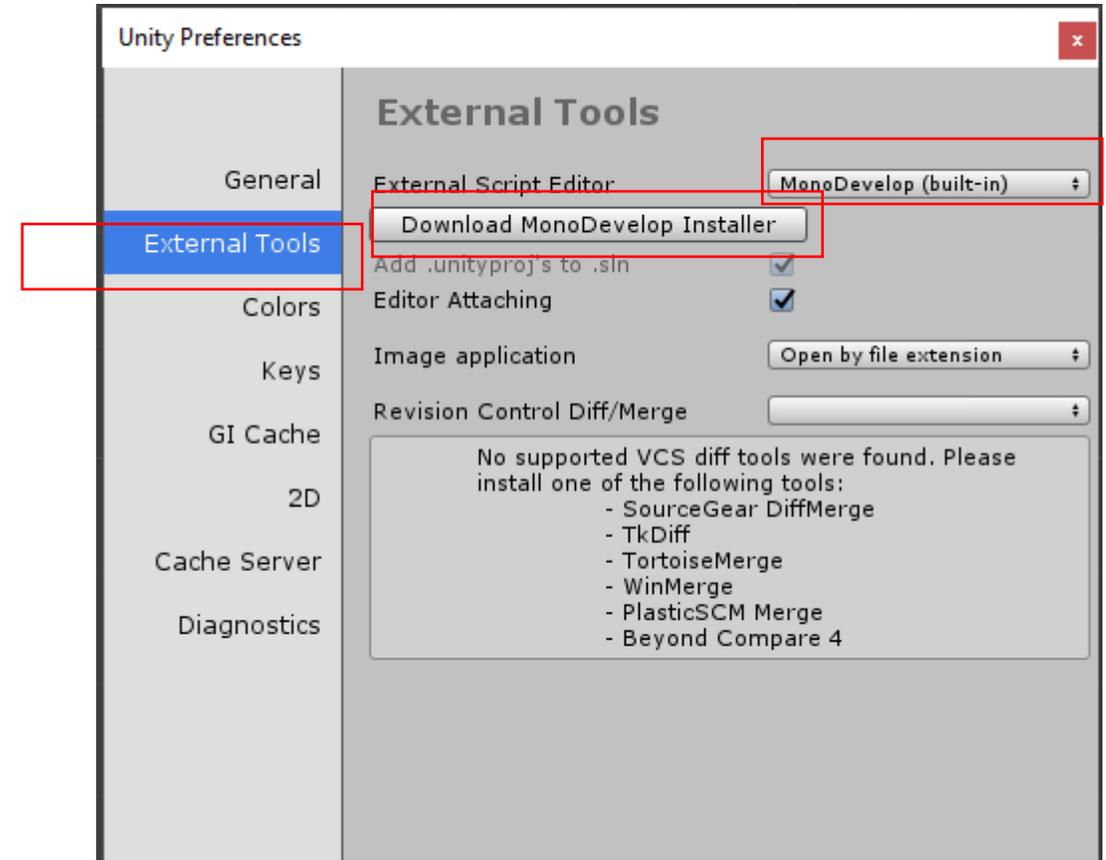
# Configurar o Editor de código monoDevelop

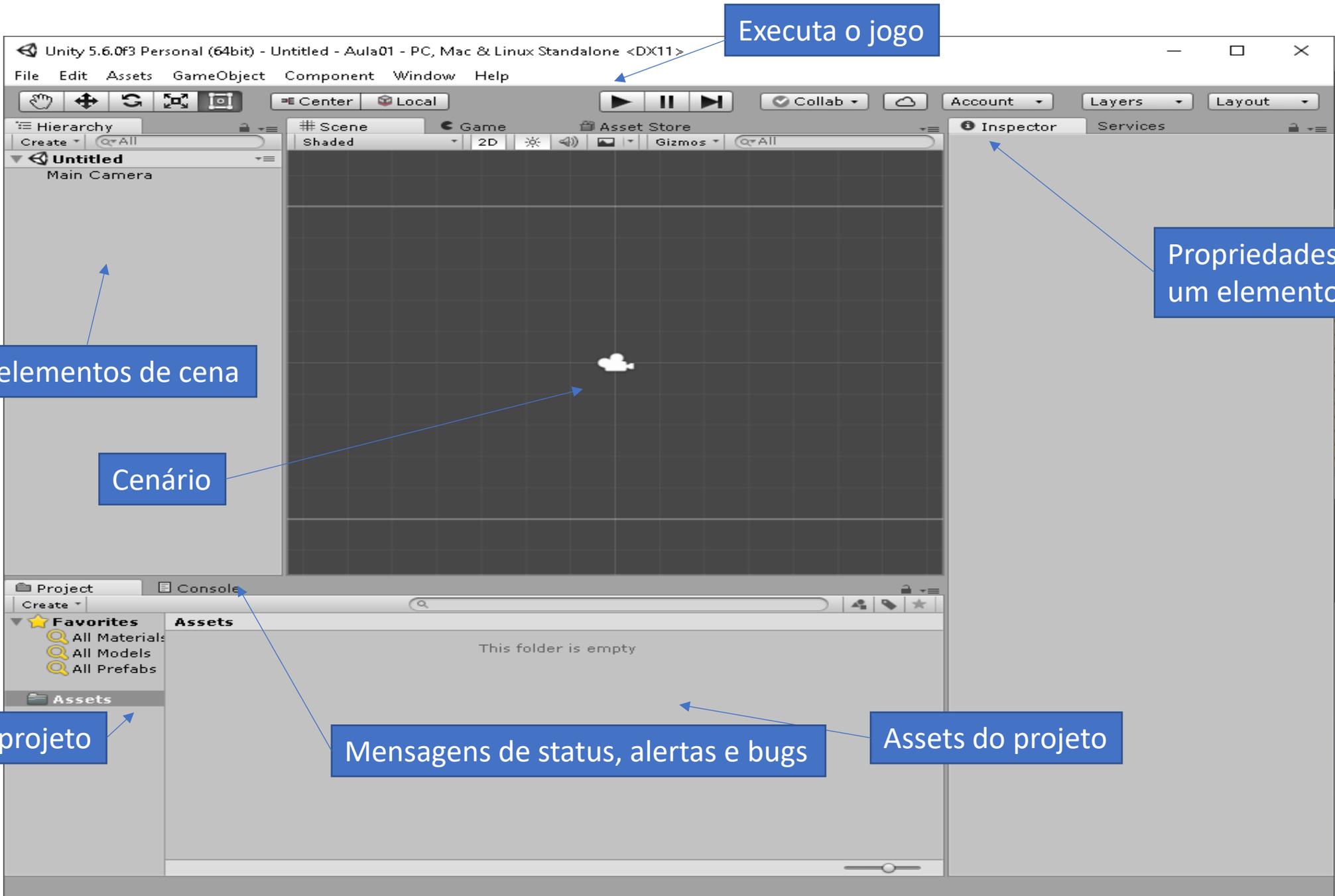
- É necessário realizar esse procedimento apenas uma vez.
- Ao carregar o Unity vá o menu: **“Edit”>>“Preferences...”**



# MonoDevelop

- Click em “**External Tools**”
- Escolha a opção “MonoDevelop (Built-in)”
- Caso seja necessário o unity indicará um link para download.
  - Caso necessário faça o download e instale o **monoDevelop**.
- Feche a janela “**Unity Preferences**”





Executa o jogo

Lista de elementos de cena

Cenário

Propriedades de um elemento de cena

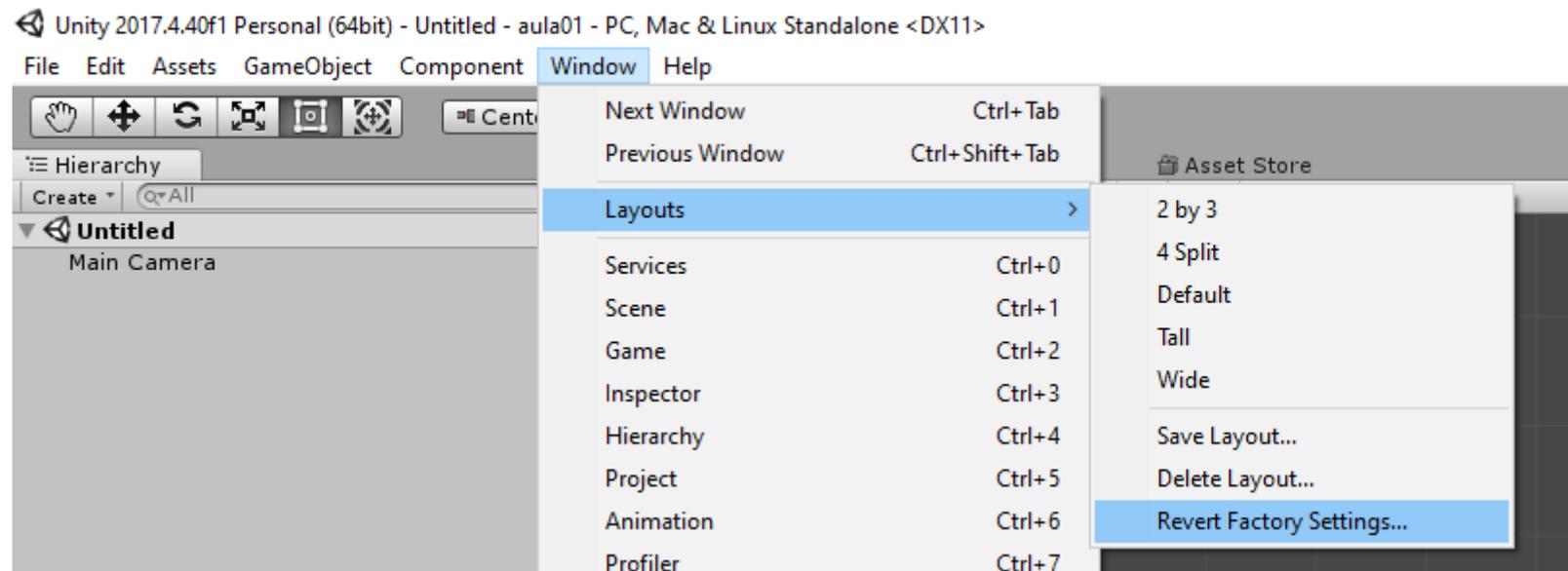
Pastas do projeto

Mensagens de status, alertas e bugs

Assets do projeto

# Restaurando o Layout Original

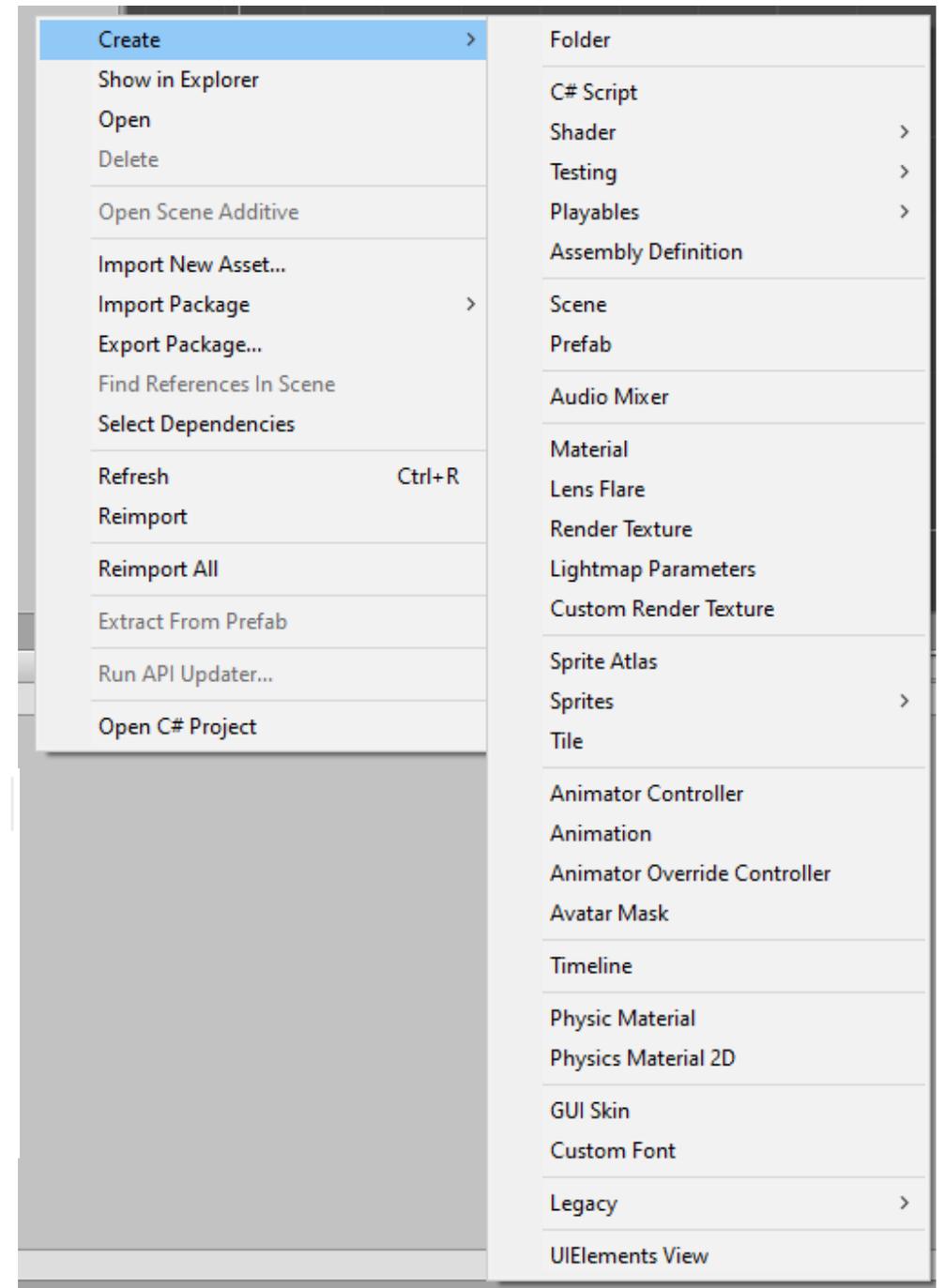
- Se perder ou fechar algum dos painéis e quiser restaurar o layout original do programa, acesse a opção Window > Layouts > Revert Factory Settings



# Estrutura do Projeto

Podemos ignorar todas as pastas e nos atentar apenas à pasta Assets, que é onde ficarão todos os arquivos e recursos do nosso jogo (modelos 3D, sons, gráficos, telas e botões).

Nome	Data de modificação	Tipo	Tamanho
Assets	11/08/2022 09:10	Pasta de arquivos	
Library	11/08/2022 09:10	Pasta de arquivos	
ProjectSettings	11/08/2022 09:10	Pasta de arquivos	
Temp	11/08/2022 09:10	Pasta de arquivos	
UnityPackageManager	11/08/2022 09:10	Pasta de arquivos	
Aula01.sln	11/08/2022 09:10	Visual Studio Solu...	1 KB



# Editor de Cena e Navegação 3D



Atalho de Navegação	Função
Alt + clique Esquerdo	Orbit
Click do Scroll	Pan
Scroll	Zoom
F	Zoom no objeto selecionado
Z	Alinhamento do Pivot (Object/Center)
X	Orientação do Pivot (Local / World)

# Atalho útil



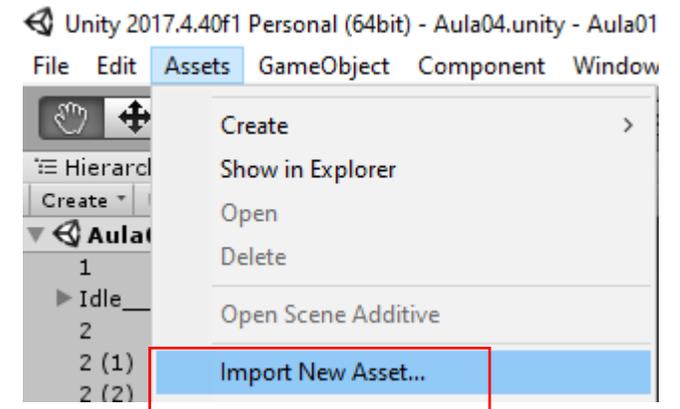
Gizmos de Manipulação	Função
Q	Pan
W	Move
E	Rotate
R	Scale
T	Rect Tool
Y	Move Rotate Scale (segure Shift para View Align)



Outros comandos	Função
Ctrl + P	Executar Aplicação (Play/Stop)
Shift+ Espaço	Tela Cheia na janela ativa (em foco)
Ctrl + D	Duplicar
Ctrl + N	Novo Node – Empty

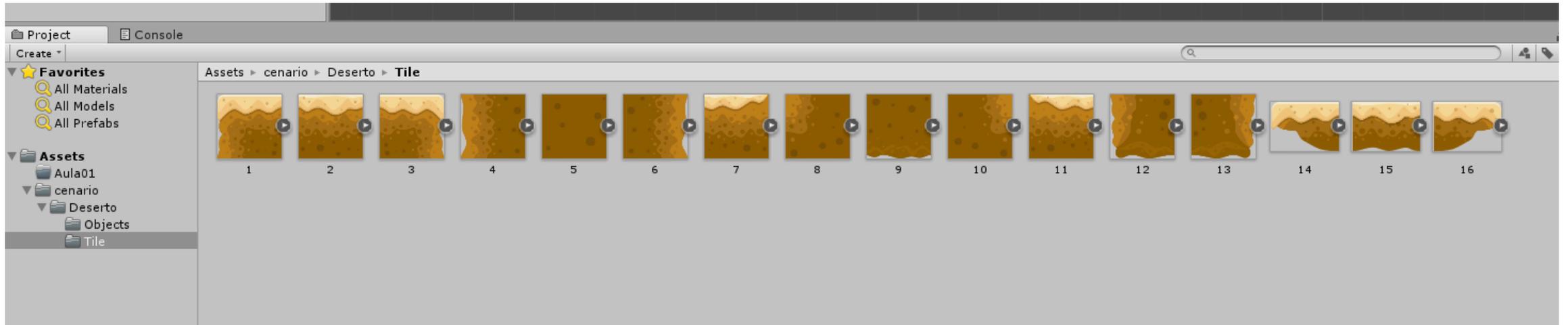
# Acrescente seus Assets

- Click no menu: **Assets** >> **Import new Asset**
- Selecione os arquivos de assets desejados e click em **Import**
- Caso possua muitos assets crie mais pastas dentro da pasta Assets para manter a organização



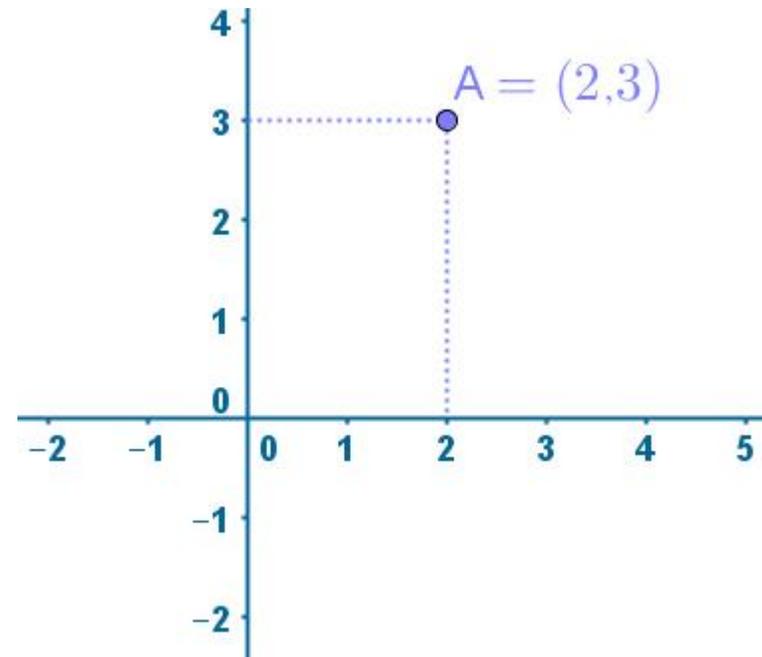
# Após importar

- Na parte inferior da interface padrão do Unity é possível visualizar os assets que foram importados para a plataforma.



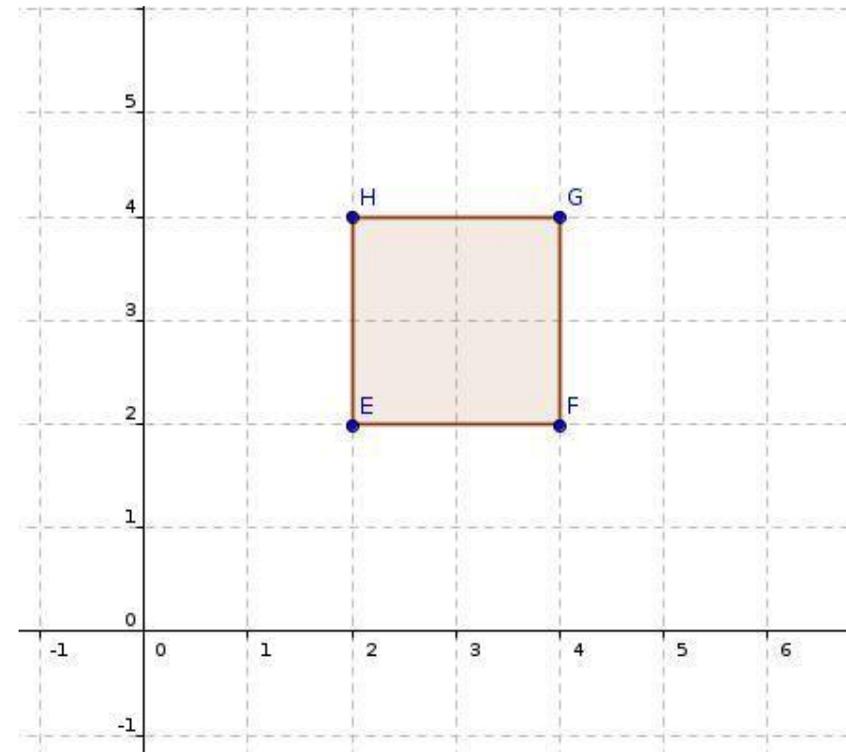
# Posição no plano 2d

- O Unity trabalha com conceitos de posicionamento no plano.
- Observe que o ponto **A** possui  **$x=2$  e  $y=3$**



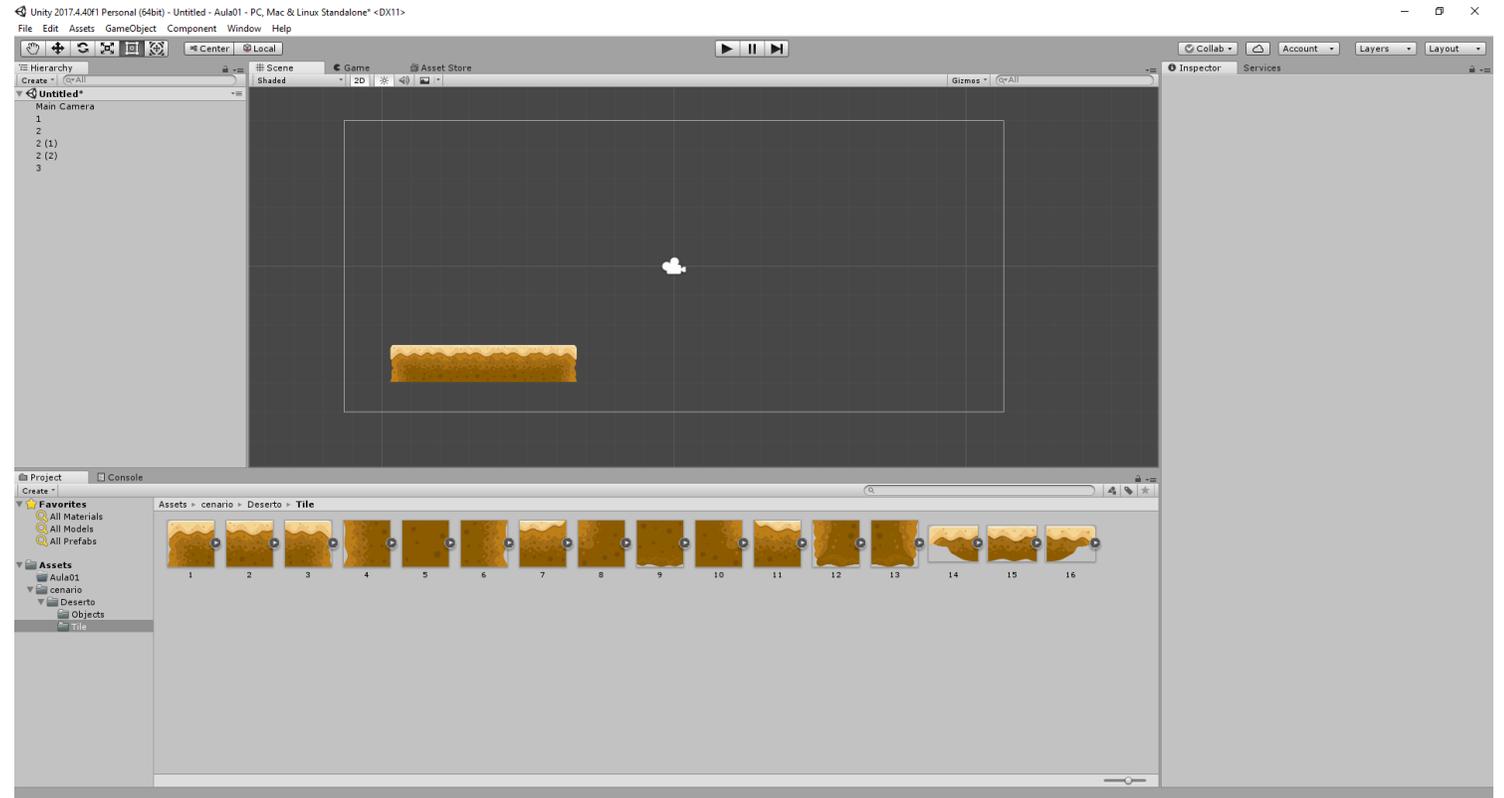
# Posição dos assets no cenário 2d

- Todo asset posicionado no cenário é arranjado conforme as posições  $x$  e  $y$ .
- Observe ao lado um bloco posicionado no plano.

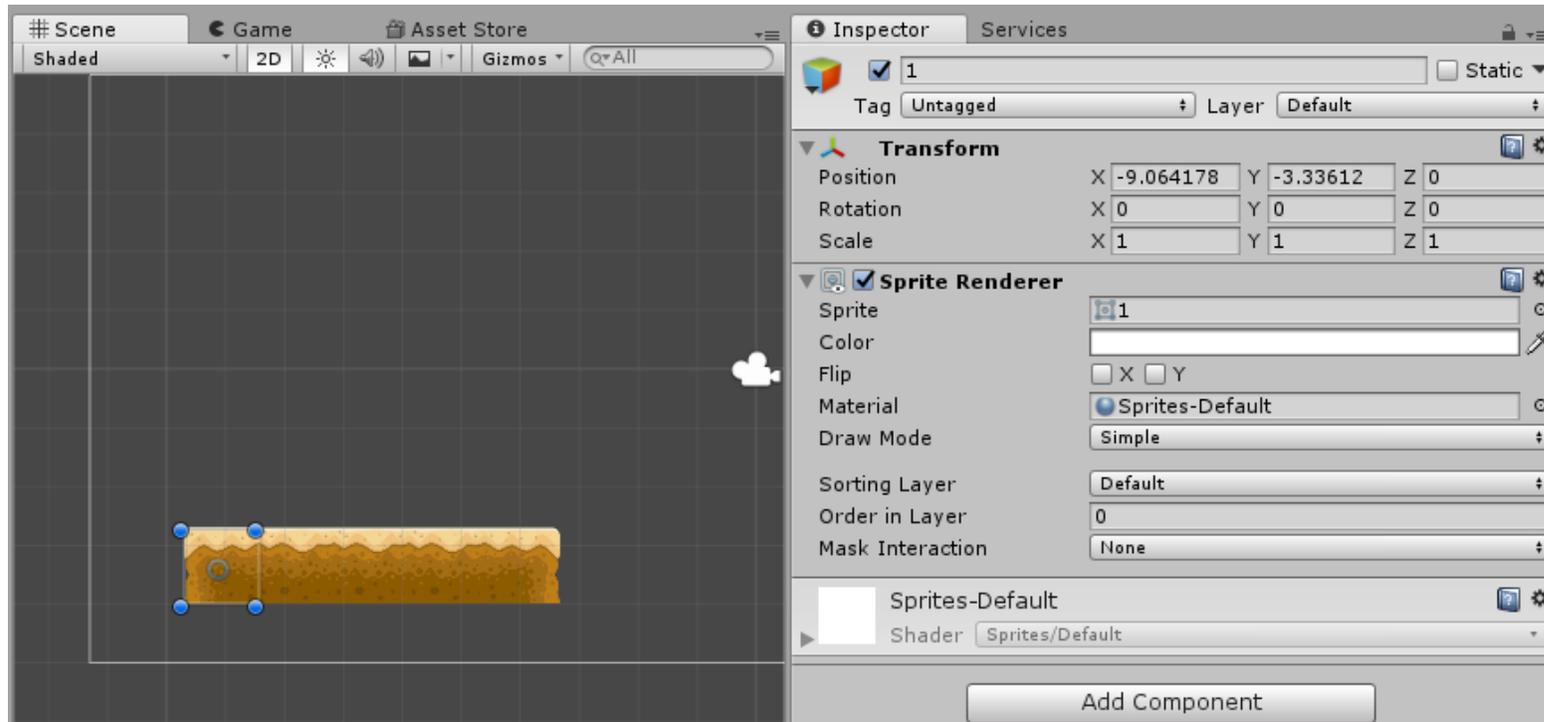


# Posicione os Assets no seu cenário

- Arraste os assets para dentro do cenário e os posicione no plano.

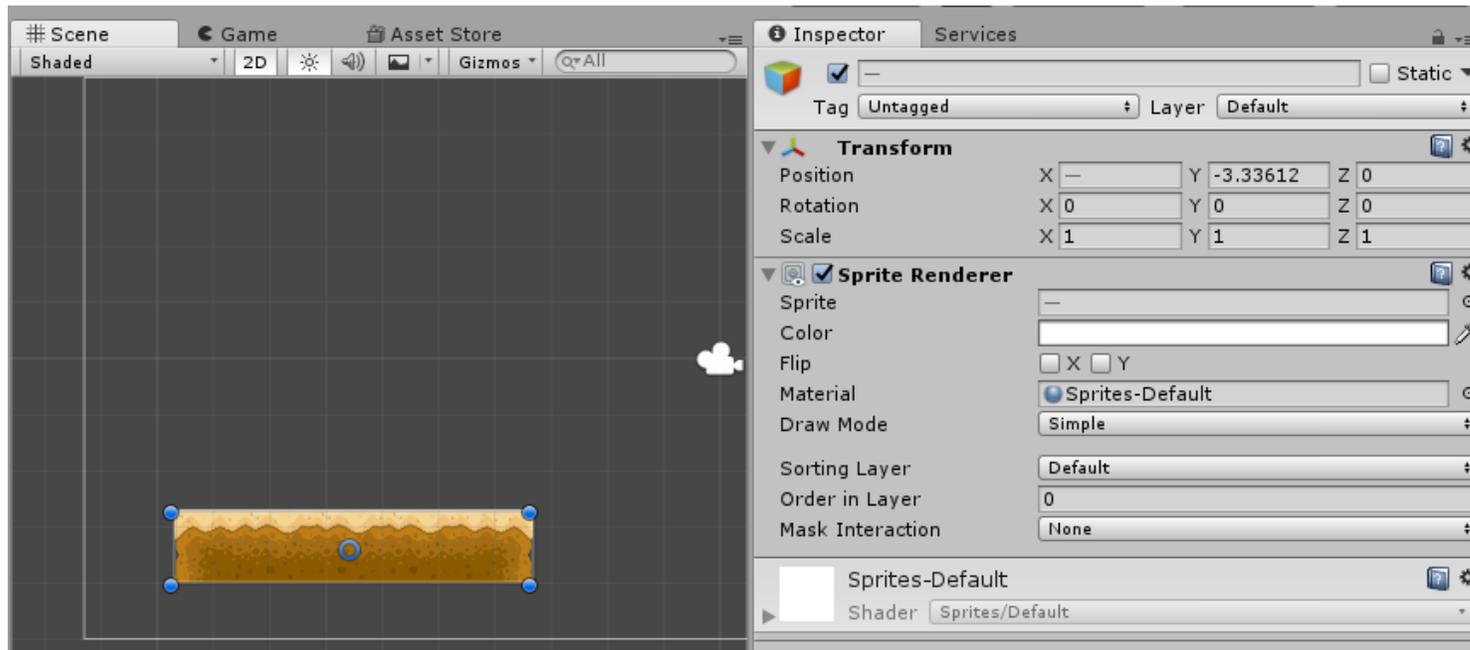


# Propriedade: Position



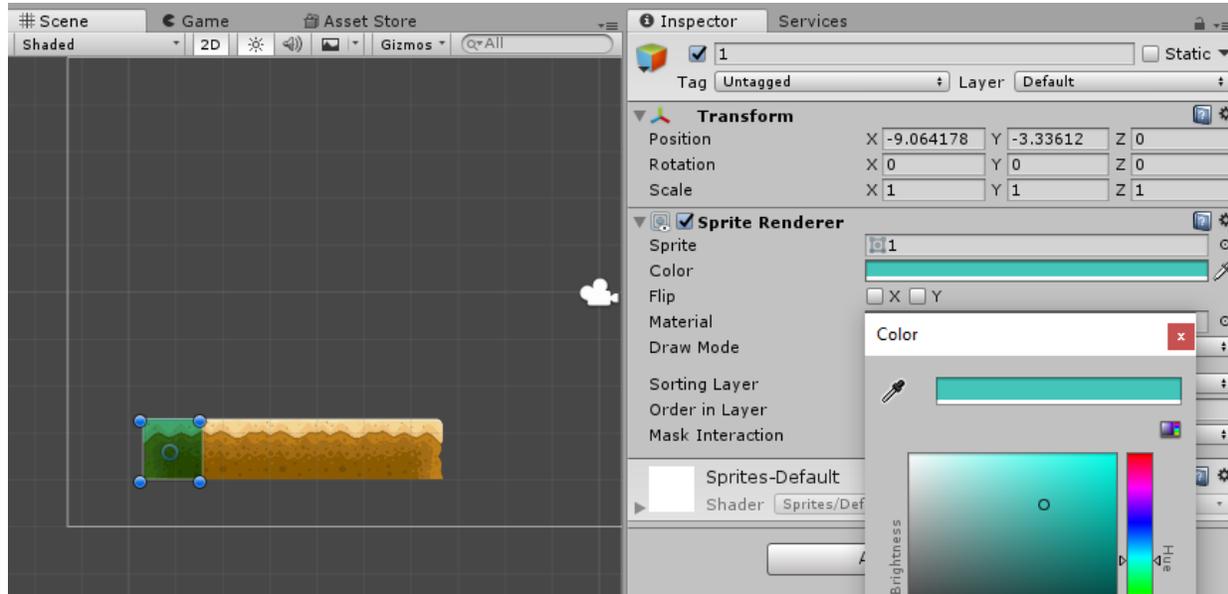
- Cada elemento posicionado no cenário possui uma série de propriedades
- Ao clicar no objeto de cena conforme figura ao lado é possível verificar todas as suas propriedades.
- Exemplo de propriedade: **Position: x,y e z** equivalem a posição de cada elemento dentro do cenário.

# Propriedade: Position



- A imagem ao lado representa a seleção de um conjunto de objetos de jogo.
- É possível verificar que a propriedade Position Y de todos eles é igual.
- Isso permite que aconteça uma alinhamento em **y**, ou seja não há um elemento com alinhamento mais alto que o outros, todos possuem a mesma altura de alinhamento.

# Propriedade: **Color**



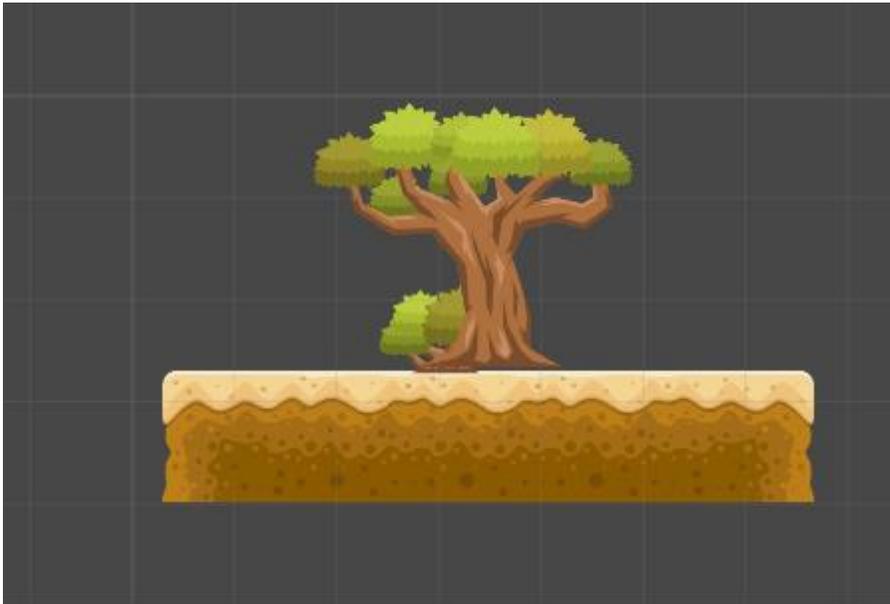
- Existem outras propriedades além das propriedades de posicionamento.
- A propriedade **color** permite aplicar um filtro de cor a um determinado **Asset** conforme figura ao lado.

# Propriedade: **Order Layer**



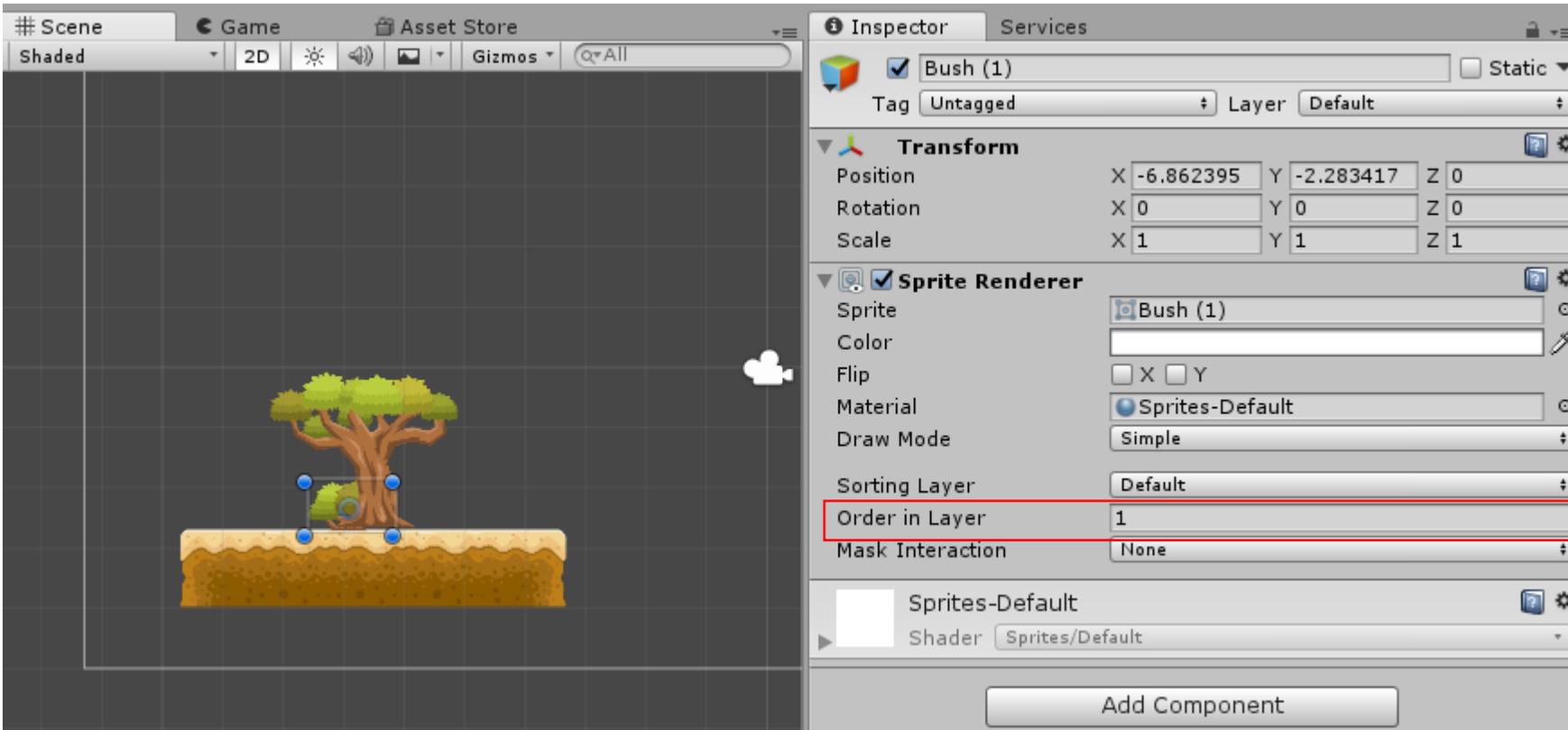
- Essa propriedade permite definir quando um objeto é posicionado a frente ou atrás de outro objeto.
- Observe a figura ao lado, o arbusto deve possuir um **order layer** maior que o **order layer** da árvore, pois o arbusto está a frente da árvore.

# Propriedade: Order Layer



- Observe a figura ao lado, o arbusto deve possuir um **order layer** menor que o **order layer** da árvore, pois o arbusto está atrás da árvore.

# Como mudar o **order layer**?



- Para mudar o **order layer** click sobre o elemento no cenário e procure a propriedade **order layer**

## Conceitos sobre física

- Física em jogos se refere a uma simulação controlada pela própria engine.
- No Unity, existem duas engines de física disponíveis, uma 2D, baseada na Box2D, e outra em 3D, chamada PhysX.

# Collider

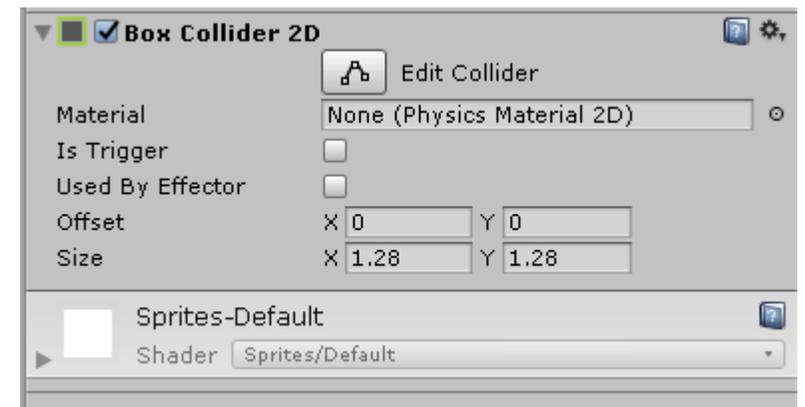
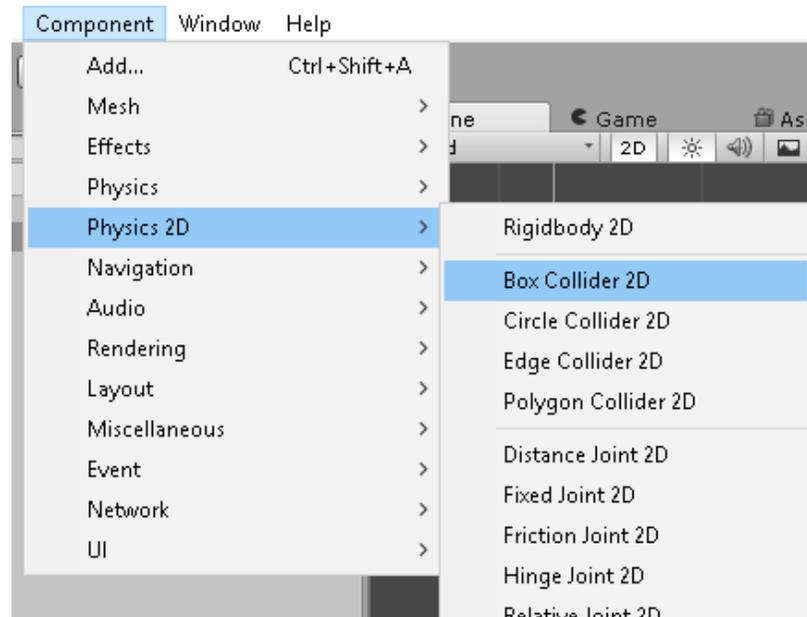
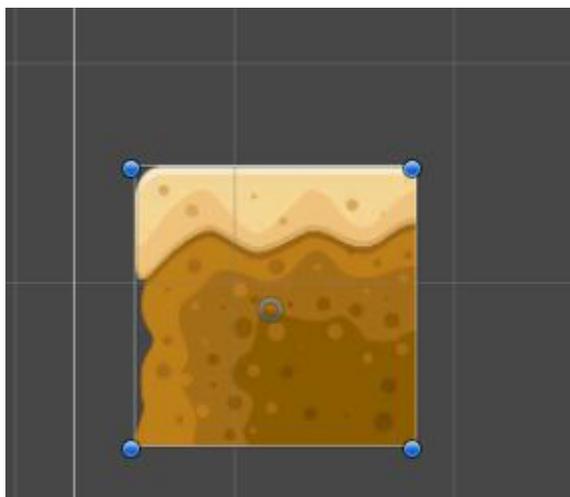
Componentes que marcam o volume físico de um objeto, assim como o material físico (com atrito e elasticidade) que define parte de seu comportamento.

Existem vários tipos de colisores, cada um com formato diferente (por exemplo, BoxCollider, SphereCollider).

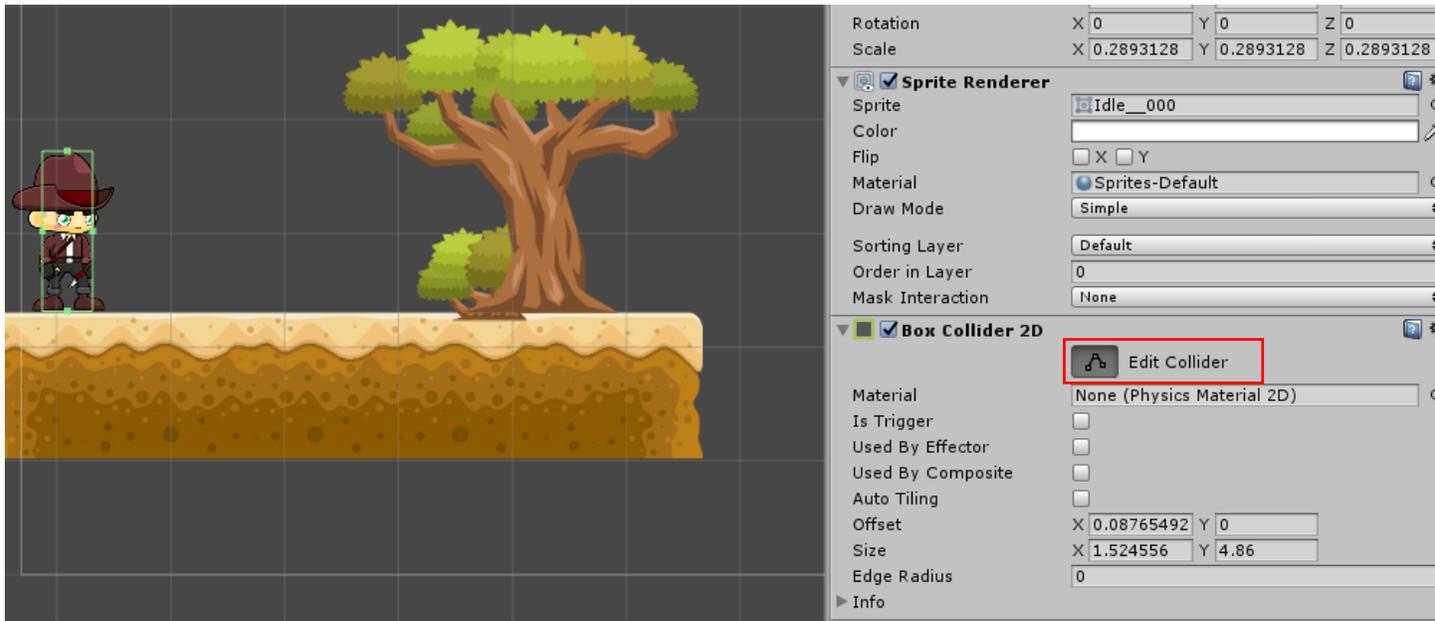
Também mantem informações sobre o tipo de interação que tem com outros objetos (colisão ou sobreposição). Esse componente tem métodos e eventos que utilizamos para criar lógica para interações físicas, como OnCollisionEnter e OnCollisionExit. Para que um script possa utilizar essas funções-evento, o objeto a que está atrelado tem que ter um Collider.

# Colisão de elementos

- Acrescente o chão do cenário.
- Clique sobre o componente.
- Vá ao menu Component >> Physics2D>>Box Collider 2D
- Faça para todos os elementos onde pode ocorrer colisão.



# Colisão de elementos



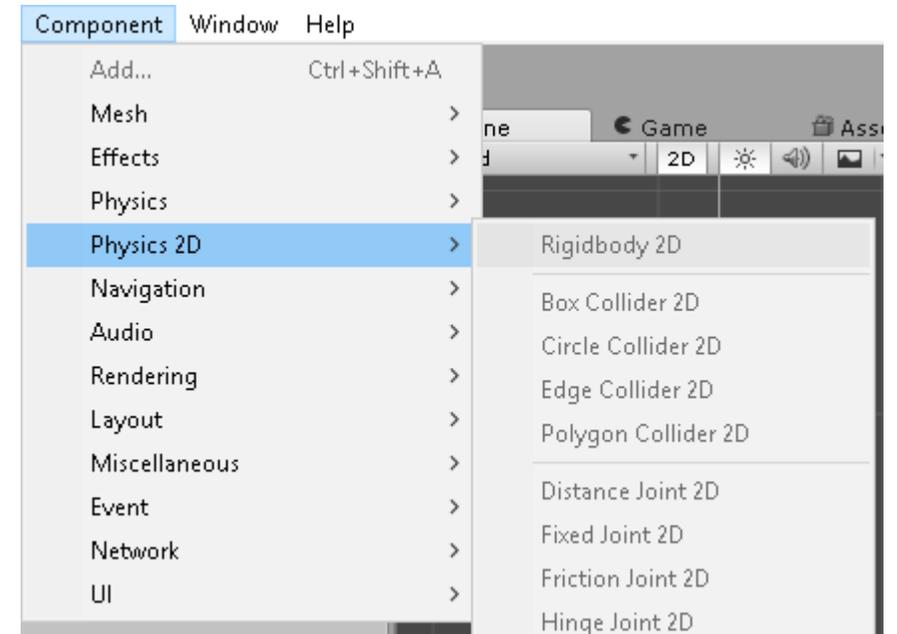
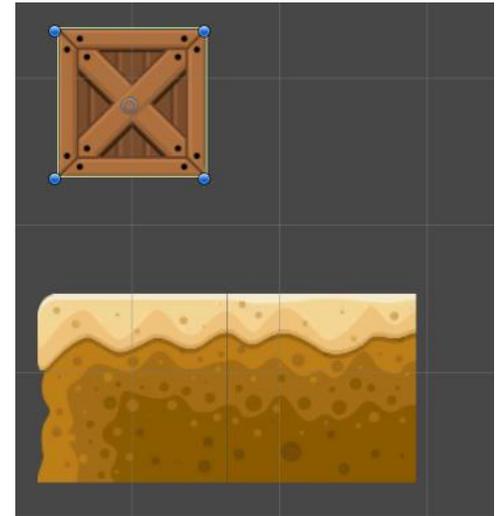
- Quando é adicionada a caixa de colisão a um elemento de cena, a caixa de colisão torna-se uma propriedade do elemento.
- Além das propriedades de **posicionamento** e **cor** agora o elemento possui propriedades de colisão.
- É possível configurar o tamanho da caixa de colisão (**hitbox**) clicando em edit Collider como pode ser visto na figura ao lado.

# Corpos rígidos

- O principal tipo de física usado em jogos é a física de corpos rígidos, que considera que objetos não deformam ou alteram seu volume e material quando afetados por interações como colisões.
- Rigidbody2D
  - Componentes que marcam um objeto como parte da simulação física. Configura propriedades como massa, velocidade e arrasto, e permite o controle de como esse objeto se movimenta e gira em resposta a forças.
  - O componente também guarda métodos para a aplicação de forças sobre o objeto.

# RigidBody 2D

- Acrescento o caixote
- Defina o como caixa de colisão 2D
  - Vá ao menu **Component** >> **Physic2D**>>**Box Collider 2D**
- Acrescente Física ao de corpo Rígido
  - Vá ao menu menu **Component** >> **Physic2D**>>**RigidBody 2D**
- Como a caixa é um corpo rígido ao rodar o jogo ela vai cair se se colidir com o chão que por sinal é uma caixa de colisão.



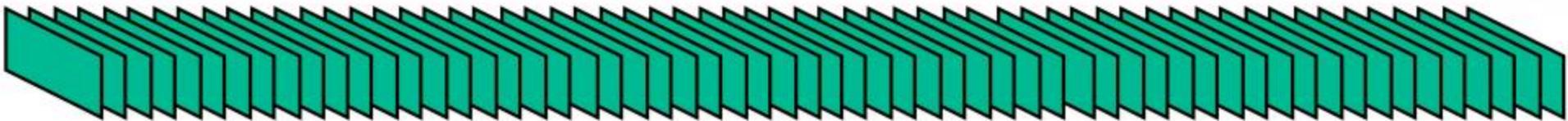
# Faça suposições

- o que acontece se o cowboy e o chão forem configurados com colisor?
- o que acontece se o cowboy for configurado como corpo rígido e o chão for configurado como colisor?
- O que acontece se o chão for configurado como corpoRígido?
- 

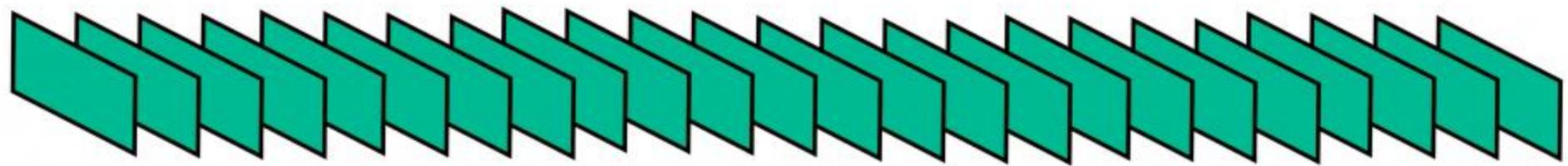


# O que é FPS(frame per second)?

- O FPS é a sigla que determina a quantidade quadros por segundo em um filme, uma animação ou uma cena de um jogo.
- Quanto mais quadros, mais fluída será a animação e mais detalhado será o movimento
- O cérebro humano de transformar imagens em sequência em um “filme” dentro de nossas cabeças.

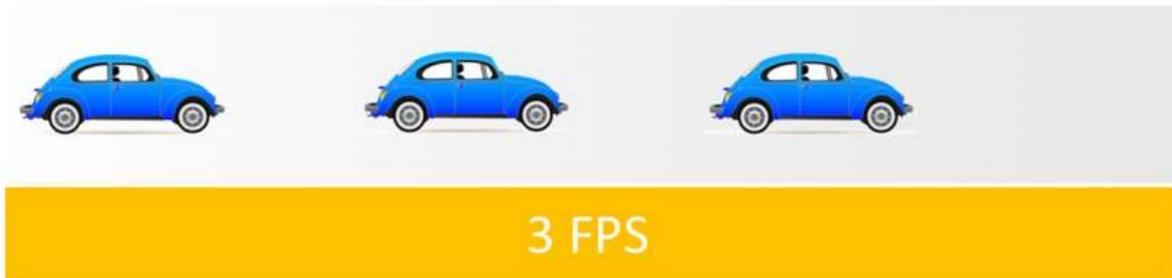


**60 FPS** (FRAMES PER SECOND)



**24 FPS** (FRAMES PER SECOND)

← 1 SECOND →

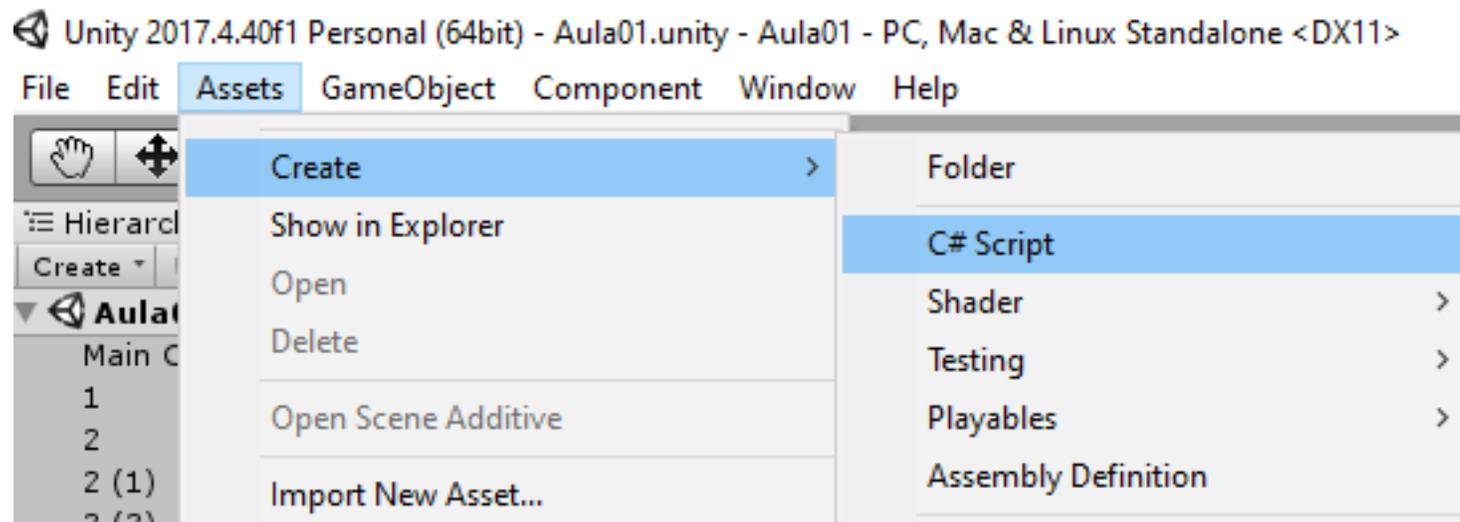


# Script na computação

- Script é um texto com uma série de instruções escritas para serem seguidas por um computador.
- O termo é uma redução da palavra inglesa manuscript, que significa “manuscrito”, “escrito à mão”.

# Criar um Script para o jogo

- Click no meu “Assets”>>”C# Script”
- Mude o nome do arquivo criado para : **Jogador** e efetue dois cliques para editar o script



# Entenda o Script

Nome do Arquivo

Características de comportamento individual.

```
using UnityEngine;
using System.Collections;

public class MovimentoJogador : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

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

    }

}
```

Executa somente quando  
O jogo é iniciado

Executa uma vez a cada frame

# Atributos

- Os atributos são características.
- São utilizados para descrever alguma coisa.
- Quais os atributos da sala?
- Quais os atributos da mesa?
- Quais os atributos de um personagem de jogo?

# Atributos do Personagem

```
using UnityEngine;

// Classe que controla o movimento do personagem
public class PersonagemAula1 : MonoBehaviour {

    // Variáveis para controlar a velocidade do personagem
    float VelocidadeX; // Velocidade no eixo X (horizontal)
    float VelocidadeY; // Velocidade no eixo Y (vertical)
    float VelocidadeHorizontalMaxima; // Velocidade máxima que o personagem pode atingir no eixo X
    float DirecaoHorizontal; // Direção do movimento horizontal (-1 para esquerda, 1 para direita, 0 para parado)
    Vector2 VetorVelocidadePersonagem; // Vetor que armazena a velocidade do personagem em X e Y

    // Referência ao componente Rigidbody2D do personagem
    Rigidbody2D CorpoRigidoPersonagem;

    void Start () {

    }

    void Update () {

    }

}
```

```

using UnityEngine;

// Classe que controla o movimento do personagem
public class PersonagemAula1 : MonoBehaviour {

    // Variáveis para controlar a velocidade do personagem
    float VelocidadeX; // Velocidade no eixo X (horizontal)
    float VelocidadeY; // Velocidade no eixo Y (vertical)
    float VelocidadeHorizontalMaxima; // Velocidade máxima que o personagem pode atingir no eixo X
    float DirecaoHorizontal; // (-1 para esquerda, 1 para direita, 0 para parado)
    Vector2 VetorVelocidadePersonagem; // Vetor que armazena a velocidade do personagem em X e Y
    Rigidbody2D CorpoRigidoPersonagem; // Referência ao componente Rigidbody2D do personagem

    // Método chamado uma vez no início do jogo
    void Start () {

        // Obtém o componente Rigidbody2D do objeto ao qual este script está anexado
        CorpoRigidoPersonagem = GetComponent<Rigidbody2D> ();

        // Define a escala da gravidade para o Rigidbody2D (1 é o valor padrão)
        CorpoRigidoPersonagem.gravityScale = 1f;

        // Congela a rotação do Rigidbody2D para evitar que o personagem gire ao colidir com algo
        CorpoRigidoPersonagem.freezeRotation = true;

        // Inicializa as variáveis de velocidade
        VelocidadeX = 0f; // Começa parado no eixo X
        VelocidadeY = 0f; // Começa parado no eixo Y
        VelocidadeHorizontalMaxima = 2.5f; // Define a velocidade máxima no eixo X
        DirecaoHorizontal = 0f; // Começa sem direção definida

        // Cria um vetor de velocidade inicial para o personagem
        VetorVelocidadePersonagem = new Vector2 (VelocidadeX, VelocidadeY);
    }

    void Update() {

    }

}

```

# Inicialização de Variáveis

- O método **void Start()** deve ser utilizado para iniciar variáveis e propriedades do Personagem.
- Nesse momento o Corpo Rígido que foi adicionado ao personagem (Component>>Physic2D>>Rigid Body2D) é inserido na variável **CorpoRigido**, assim é possível modificar propriedades de física utilizando a variável CorpoRigido.
- Observe que a propriedade freezeRotation é modificada para true, isso impede que o personagem rotacione quanto tocar em quinas.

# Criar movimento horizontal

DirecaoHorizontal Recebe até 1 para direita e até -1 para esquerda.  
Quando parado Recebe 0.

```
void Update () {
```

```
    // Input.GetAxis("Horizontal") retorna um valor entre -1 e 1:  
    // -1 para esquerda, 1 para direita, 0 se nenhuma tecla estiver sendo pressionada  
    DirecaoHorizontal = Input.GetAxis ("Horizontal");
```

```
    // Calcula a velocidade no eixo X multiplicando a direção pela velocidade máxima  
    VelocidadeX = VelocidadeHorizontalMaxima * DirecaoHorizontal;
```

```
    // Mantém a velocidade atual no eixo Y (para não interferir na gravidade ou pulo)  
    VelocidadeY = CorpoRigidoPersonagem.velocity.y;
```

```
    // Cria um novo vetor de velocidade com os valores calculados  
    VetorVelocidadePersonagem = new Vector2 (VelocidadeX, VelocidadeY);
```

```
    // Aplica o vetor de velocidade ao Rigidbody2D para mover o personagem  
    CorpoRigidoPersonagem.velocity = VetorVelocidadePersonagem;
```

```
}
```

= 5 \* DirecaoHorizontal.

Determina o módulo, sentido e direção do movimento

# Código Completo

```
using UnityEngine;
public class PersonagemAula1 : MonoBehaviour {
    // Variáveis para controlar a velocidade do personagem
    float VelocidadeX; // Velocidade no eixo X (horizontal)
    float VelocidadeY; // Velocidade no eixo Y (vertical)
    float VelocidadeHorizontalMaxima; // Velocidade máxima que o personagem pode atingir no eixo X
    float DirecaoHorizontal; // Direção do movimento horizontal (-1 para esquerda, 1 para direita, 0 para parado)
    Vector2 VetorVelocidadePersonagem; // Vetor que armazena a velocidade do personagem em X e Y
    Rigidbody2D CorpoRigidoPersonagem; // Referência ao componente Rigidbody2D do personagem
    void Start () {
        CorpoRigidoPersonagem = GetComponent<Rigidbody2D> (); // Obtém o Rigidbody2D do objeto ao qual este script está anexado
        // Define a escala da gravidade para o Rigidbody2D (1 é o valor padrão)
        CorpoRigidoPersonagem.gravityScale = 1f;
        // Congela a rotação do Rigidbody2D para evitar que o personagem gire ao colidir com algo
        CorpoRigidoPersonagem.freezeRotation = true;
        VelocidadeX = 0f; // Começa parado no eixo X
        VelocidadeY = 0f; // Começa parado no eixo Y
        VelocidadeHorizontalMaxima = 2.5f; // Define a velocidade máxima no eixo X
        DirecaoHorizontal = 0f; // Começa sem direção definida
        VetorVelocidadePersonagem = new Vector2 (VelocidadeX, VelocidadeY);
        CorpoRigidoPersonagem.velocity = VetorVelocidadePersonagem;
    }
    void Update () { // Método chamado uma vez por frame (quadro)
        // -1 para esquerda, 1 para direita, 0 se nenhuma tecla estiver sendo pressionada
        DirecaoHorizontal = Input.GetAxis ("Horizontal");
        // Calcula a velocidade no eixo X multiplicando a direção pela velocidade máxima
        VelocidadeX = VelocidadeHorizontalMaxima * DirecaoHorizontal;
        // Mantém a velocidade atual no eixo Y (para não interferir na gravidade ou pulo)
        VelocidadeY = CorpoRigidoPersonagem.velocity.y;
        // Cria um novo vetor de velocidade com os valores calculados
        VetorVelocidadePersonagem = new Vector2 (VelocidadeX, VelocidadeY);
        // Aplica o vetor de velocidade ao Rigidbody2D para mover o personagem
        CorpoRigidoPersonagem.velocity = VetorVelocidadePersonagem;
    }
}
```

# Melhorando o código - Organização

- Conforme o andar do desenvolvimento de um jogo é possível verificar que o método Update ganha mais linhas de código conforme o Personagem ganha funcionalidade.
- É possível que ao final do desenvolvimento o jogo possua centenas de linhas de código ou mais.
- Para melhor organizar o que é programado em **void Update()** você pode criar blocos de código separados, onde cada bloco possui uma funcionalidade diferente.

# Organização

```
// Classe que controla o movimento do personagem
public class PersonagemAula1 : MonoBehaviour {
```

```
    // Variáveis para controlar a velocidade do personagem
    float VelocidadeX; // Velocidade no eixo X (horizontal)
    float VelocidadeY; // Velocidade no eixo Y (vertical)
    float VelocidadeHorizontalMaxima; // Velocidade máxima que o personagem pode atingir no eixo X
    float DirecaoHorizontal; // Direção do movimento horizontal (-1 para esquerda, 1 para direita, 0 para parado)
    Vector2 VetorVelocidadePersonagem; // Vetor que armazena a velocidade do personagem em X e Y
    Rigidbody2D CorpoRigidoPersonagem; // Referência ao componente Rigidbody2D do personagem
    void Start () { // Método chamado uma vez no início do jogo
        CorpoRigidoPersonagem = GetComponent<Rigidbody2D> (); // Obtém o componente Rigidbody2D do objeto ao qual este script está anexado
        CorpoRigidoPersonagem.gravityScale = 1f; // Define a escala da gravidade para o Rigidbody2D (1 é o valor padrão)
        CorpoRigidoPersonagem.freezeRotation = true; // Congela a rotação do Rigidbody2D para evitar que o personagem gire ao colidir com algo
        // Inicializa as variáveis de velocidade
        VelocidadeX = 0f; // Começa parado no eixo X
        VelocidadeY = 0f; // Começa parado no eixo Y
        VelocidadeHorizontalMaxima = 2.5f; // Define a velocidade máxima no eixo X
        DirecaoHorizontal = 0f; // Começa sem direção definida
        VetorVelocidadePersonagem = new Vector2 (VelocidadeX, VelocidadeY); // Cria um vetor de velocidade inicial para o personagem
        CorpoRigidoPersonagem.velocity = VetorVelocidadePersonagem; // Aplica o vetor de velocidade ao Rigidbody2D para mover o personagem
    }
    void Update () { // Método chamado uma vez por frame (quadro)
        MovimentoHorizontal (); // Chama o método para controlar o movimento horizontal do personagem
    }
    void MovimentoHorizontal(){ // Método para controlar o movimento horizontal do personagem
        // Obtém a direção horizontal a partir do input do jogador (teclado ou controle)
        // Input.GetAxis("Horizontal") retorna um valor entre -1 e 1:
        // -1 para esquerda, 1 para direita, 0 se nenhuma tecla estiver sendo pressionada
        DirecaoHorizontal = Input.GetAxis ("Horizontal");

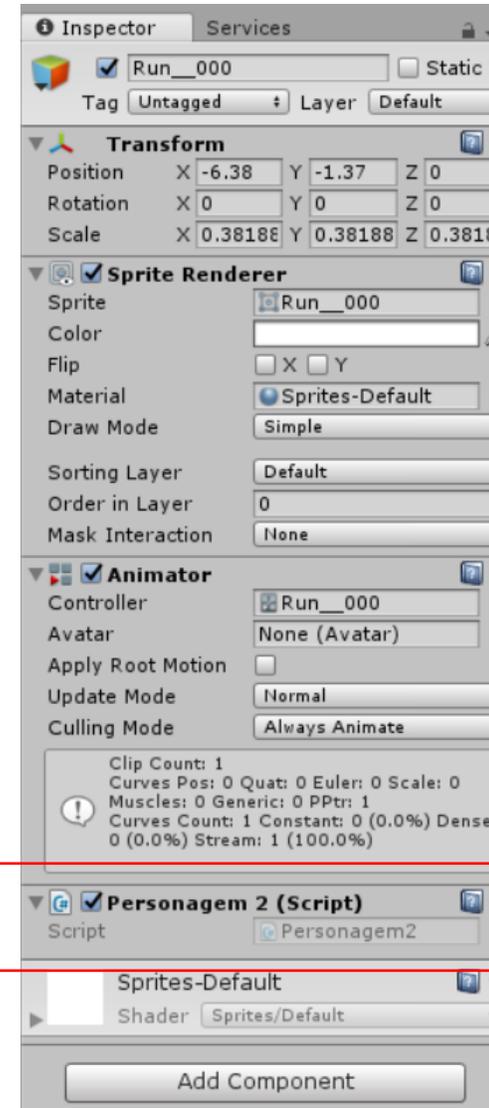
        VelocidadeX = VelocidadeHorizontalMaxima * DirecaoHorizontal; // Calcula a velocidade no eixo X multiplicando a direção pela velocidade máxima
        VelocidadeY = CorpoRigidoPersonagem.velocity.y; // Mantém a velocidade atual no eixo Y (para não interferir na gravidade ou pulo)
        VetorVelocidadePersonagem = new Vector2 (VelocidadeX, VelocidadeY); // Cria um novo vetor de velocidade com os valores calculados
        CorpoRigidoPersonagem.velocity = VetorVelocidadePersonagem; // Aplica o vetor de velocidade ao Rigidbody2D para mover o personagem
    }
}
```

Nesse momento do curso  
trataremos como um bloco de  
código que pode ser chamado  
quando acontecer um update

# Adicione o script no personagem

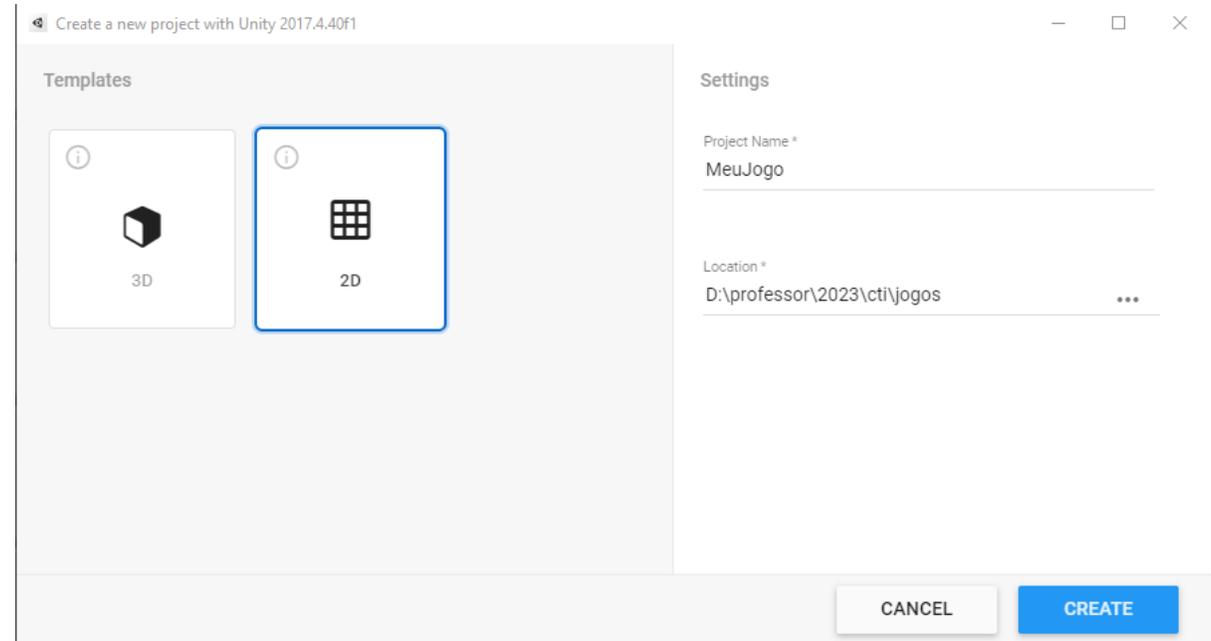
- Para que o script comece a funcionar é necessário que ele seja inserido dentro do personagem.
  - Click no arquivo do script e arraste para dentro do **asset** do personagem localizado na cena.
    - Se não existir nenhum problema no script o mesmo aparecerá na lista de propriedades do personagem no inspector

Script inserido no asset do personagem



# Salvar o projeto

- Lembre-se da pasta que você escolheu para salvar os arquivos do projeto 2d.
- No exemplo ao lado os arquivos serão salvos na pasta:
  - **d:\professor\2023\cti\jogos\meujogo**
- Antes de fechar o jogo salve a cena.
- Menu:
  - File>> Save Scene>> de um nome para o cenário
- Salve todo o conteúdo da pasta:
  - **d:\professor\2023\cti\jogos\meujogo**
- **Veja:** <https://youtu.be/mT122XNuAPU>



# Salvar e Abrir Projeto de jogo no Unity

<https://youtu.be/mT122XNuAPU>

# Git Hub da Disciplina

<https://github.com/helioesperidiao/Desenvolvimento-de-jogos>

# Tutoriais sobre github

- <https://www.youtube.com/watch?v=EGmzAs1G0z0>
- <https://www.youtube.com/watch?v=nAHVEzDBVeo>
- <https://www.youtube.com/watch?v=P9xXbEhqhqA>