créditos imagens flaticon.com

Unity

Pong

Pong

Bem-vindos à primeira aula do curso "Criando PONG na Unity". Nesta aula, vamos estabelecer as bases para o desenvolvimento do jogo Pong, explorando os conceitos fundamentais e preparando o ambiente de trabalho.

Nossa jornada começa com uma compreensão mais profunda do que é o Pong, um dos primeiros jogos eletrônicos e um marco na história dos videogames. Em seguida, vamos mergulhar no ambiente da Unity, conhecendo um pouco mais seus recursos e sua interface.

Além disso, abordaremos a etapa crucial da preparação do ambiente de desenvolvimento. Vamos orientar você na configuração adequada da Unity em seu sistema e mostrar como criar um novo projeto, o ponto de partida para a criação do nosso próprio Pong.

Estamos ansiosos para compartilhar esses conceitos essenciais e dar os primeiros passos na criação do Pong na Unity. Vamos começar!

O que é o Pong?

Pong, lançado pela Atari em 1972, é um dos primeiros e mais icônicos jogos eletrônicos da história. Criado por Nolan Bushnell, o jogo é uma recriação virtual do tênis de mesa, onde dois jogadores controlam paletas virtuais em uma tela dividida.

O objetivo é simples: rebater a bola de um lado para o outro, evitando que ela passe pela paleta e marque um ponto para o adversário. A jogabilidade é despojada e altamente competitiva, proporcionando horas de diversão para jogadores de todas as idades.

O sucesso instantâneo de Pong nos fliperamas inaugurou a era dos videogames comerciais e ajudou a estabelecer a Atari como uma força dominante na indústria. A simplicidade elegante do Pong tornou-o acessível e atraente para um público amplo, desencadeando uma revolução nos jogos eletrônicos.

Embora os gráficos e os sons do Pong possam parecer rudimentares em comparação com os jogos modernos, sua influência é inegável. Ele abriu o caminho para uma infinidade de jogos de esportes eletrônicos e serviu como a base para muitos gêneros que surgiram desde então. Pong é uma lembrança eterna de como um conceito simples pode dar origem a uma indústria inteira e continua a ser uma parte essencial da história dos jogos eletrônicos.

Configuração Inicial

Passos iniciais para a configuração e começo do projeto:

- Abertura da Unity Hub
- Crie de um novo projeto
- Escolha o template 2D
- Salve o nome do projeto como **Pong Demo** ou similar
- Escolha uma pasta para salvar o projeto

Criação da cena

Ao iniciar um novo projeto na Unity, o ambiente é automaticamente configurado para ajudar você a começar a desenvolver seu jogo. Uma cena vazia é criada por padrão, pronta para ser moldada de acordo com sua visão. Também podemos criar uma nova cena, a depender da nossa necessidade.

Game Manager

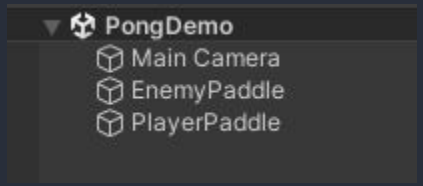
O conceito do script "***Game Manage***r" em jogos desempenha um papel vital na organização e funcionalidade do jogo como um todo. Esse script age como uma central de controle, supervisionando o fluxo do jogo, coordenando diferentes elementos e gerenciando a lógica subjacente. É responsável por tomar decisões importantes, como determinar o estado atual do jogo, controlar transições entre cenas, níveis ou fases, gerenciar recursos como pontuações e vidas, e facilitar a comunicação entre diferentes componentes do jogo. Além disso, o "Game Manager" também pode lidar com eventos-chave, como condições de vitória ou derrota, e pode estar envolvido na persistência de dados, como salvar e carregar o progresso do jogador. Em essência, o script "Game Manager" é o cerne operacional que mantém a coesão e a jogabilidade fluida ao longo de toda a experiência de jogo.

Crie um novo script chamado GameManager, e adicione a um objeto na hierarquia chamado GameManager.

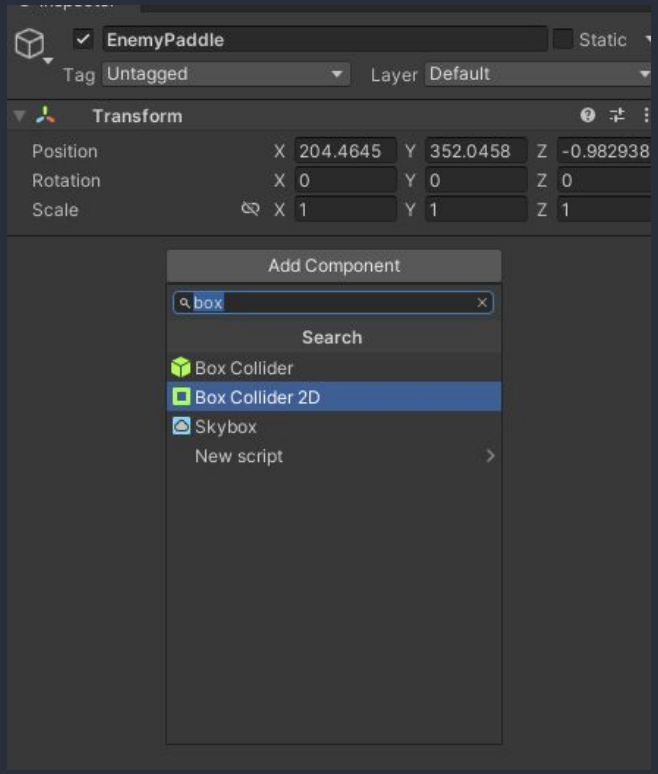
Raquetes

Agora vamos começar criando as raquetes do nosso jogo:

- Clique com o botão direito > Create Empty.
- Repita o processo por 2 vezes
- Renomeie os objetos criados para "**PlayerPaddle**" e "**EnemyPaddle**" para maior clareza.

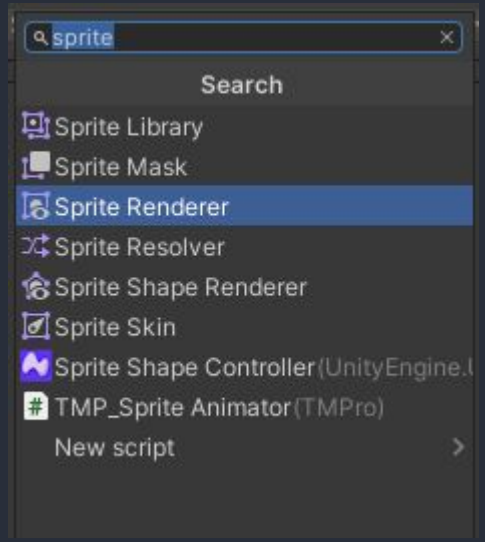


Raquetes



- Selecione **PlayerPaddle** e clique em "Add Component" > Physics2D > Box Collider 2D.
- Selecione **EnemyPaddle** e adicione também um Box Collider 2D.

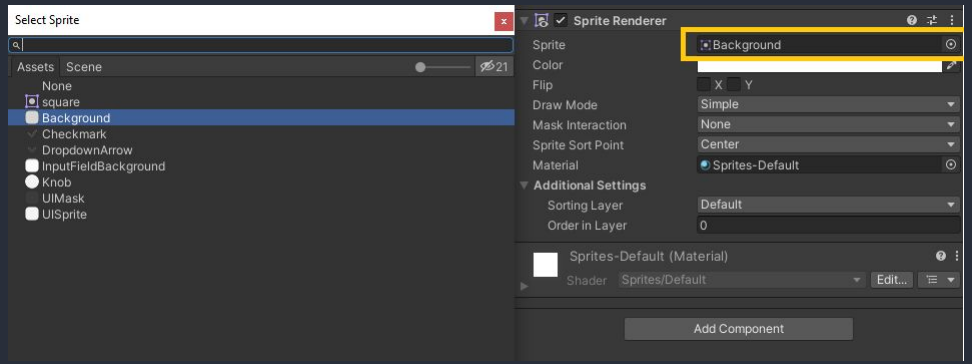
Raquetes



- Selecione **PlayerPaddle** e clique em "Add Component"
- Digite Sprite Renderer e adicione o componente Sprite Renderer

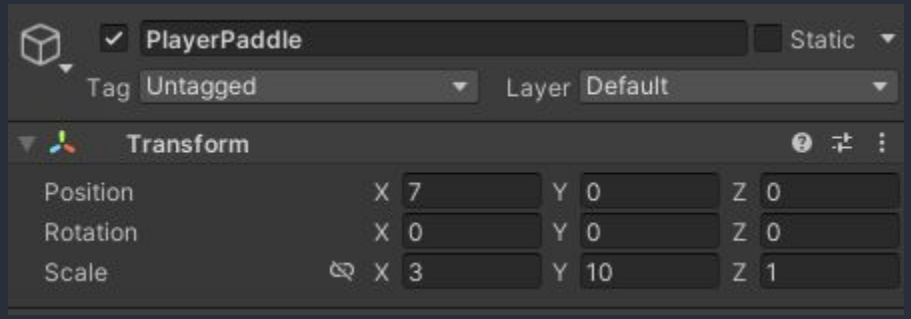
Raquetes

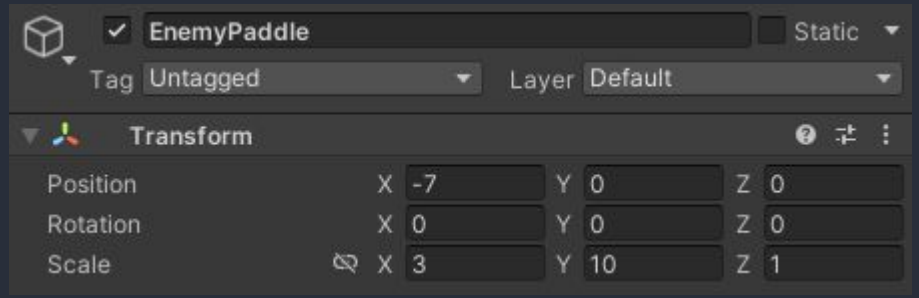
- Nos dois Sprite Renderer (Enemy e Player), selecione a sprite **Background**, em **Sprite.** Essa é uma sprite comum que a unity fornece. Mais a frente podemos trocar por outra imagem.



Raquetes

- Ajuste a posição e tamanho do **EnemyPaddle** e **PlayerPaddle** como na imagem abaixo. Repare que a posição do Enemy é **-7** e a do player **7.**





Raquetes

```
// No script do GameManager
public Transform playerPaddle;
public Transform enemyPaddle;

// ...

void Start()
{
    ResetGame();
}

public void ResetGame()
{
    // Define as posições iniciais das raquetes
    playerPaddle.position = new Vector3(-7f, 0f, 0f);
    enemyPaddle.position = new Vector3(7f, 0f, 0f);
    // ...
}
```

Agora vamos começar adicionar os primeiros scripts ao jogo.

No script GameManager, adicione o código ao lado. Ele fará com que a posição inicial das raquetes sejam sempre resetadas ao início do jogo.

Raquetes

Agora, crie um novo script chamado PlayerPaddleControler. Ele será o responsável por controlar a movimentação do jogador.

Adicione nele o código abaixo:

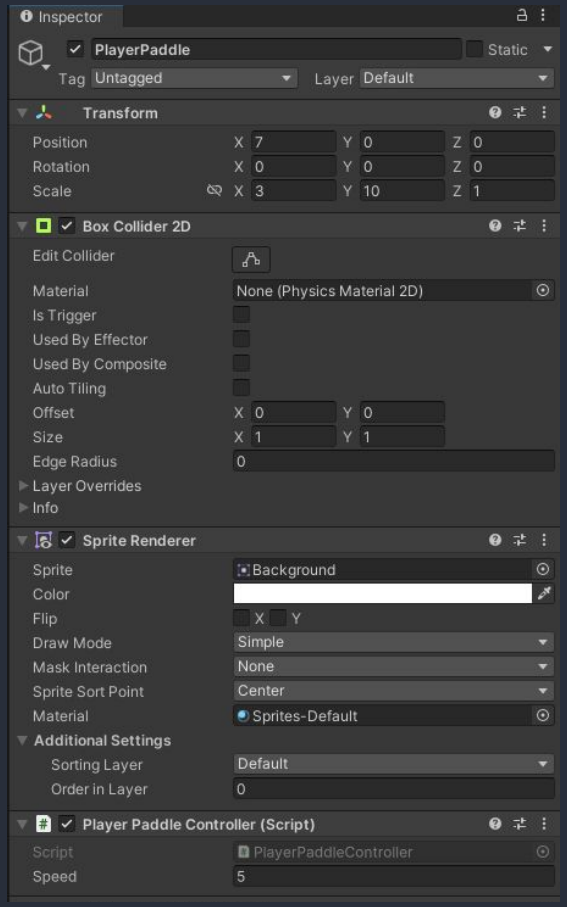
```
public float speed = 5f;

void Update()
  {
      // Captura da entrada vertical (seta para cima, seta para baixo, teclas W e S)
      float moveInput = Input.GetAxis("Vertical");

      // Calcula a nova posição da raquete baseada na entrada e na velocidade
      Vector3 newPosition = transform.position + Vector3.up * moveInput * speed *
Time.deltaTime;

      // Limita a posição vertical da raquete para que ela não saia da tela
      newPosition.y = Mathf.Clamp(newPosition.y, -4.5f, 4.5f);

      // Atualiza a posição da raquete
      transform.position = newPosition;
  }
```



Raquetes

Adicione o PlayerPaddleControler ao PlayerPaddle.

Bola

Adicione um novo objeto na hierarquia, e de o nome de Ball

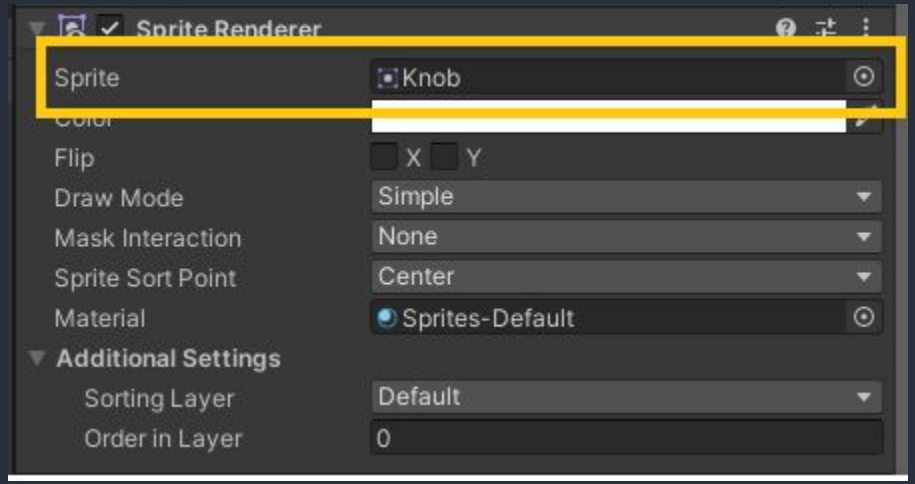
Bola

Agora, adicione ao objeto Ball 2 novos scripts:

- CircleCollider2D
- Rigibody2D

Bola

Também adicione um Sprite Renderer. Desta vez, selecione *Knob* como sprite.



Bola

Crie um script BallController, adicione o código abaixo. Ele será o responsável por fazer a bola iniciar a se movimentar.

```
private Rigidbody2D rb;
private Vector2 startingVelocity = new Vector2(5f, 5f);

public void ResetBall()
{
    transform.position = Vector3.zero;

    if(rb == null) rb = GetComponent<Rigidbody2D>();
    rb.velocity = startingVelocity;
}
```

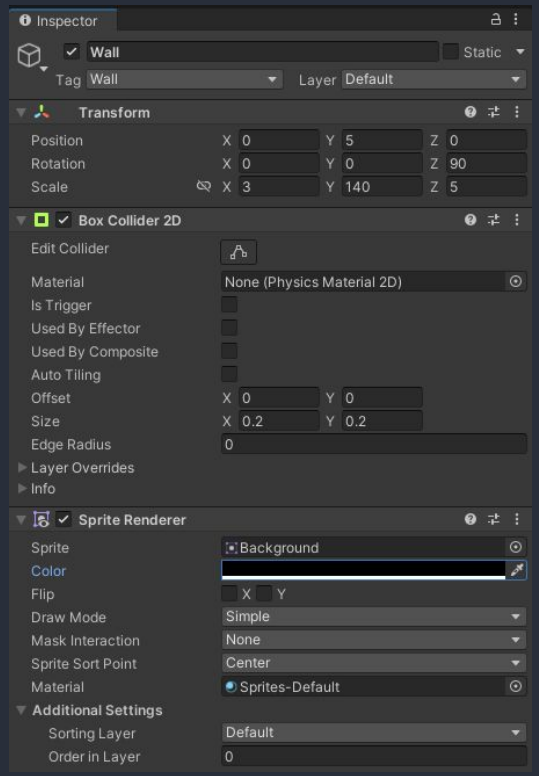
Bola

No GameManager, crie uma referência para BallController. Então, dentro de ResetGame, adicione uma chamada para o método ResetBall, dentro de BallController.

```
public BallController ballController;


//Inserir dentro do ResetGame
ballController.ResetBall();
```

Paredes



Agora vamos adicionar as paredes.

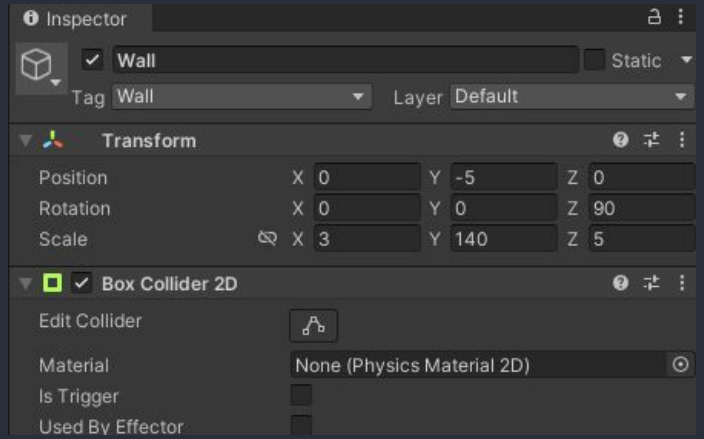
Adicione um GameObject chamado Wall.
Nele, adicione os seguintes scripts:

- BoxCollider2D
- Sprite Renderer (com a imagem Background em sprite)

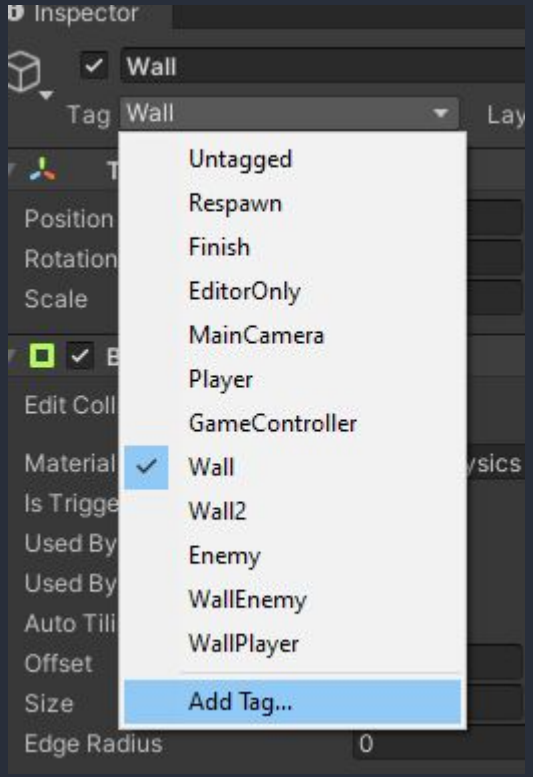
Ajuste o componente Transform para como mostra a imagem ao lado.

Paredes

Duplique o GameObject Wall, e ajuste o transform como ao lado.



Paredes



Crie uma nova Tag chamada **Wall**. Ela será importante para identificarmos quando a bolinha bater neste objeto. Depois disso, selecione **Wall** como a tag dos dois objetos **Wall.**

Paredes

Dentro do Script BallController, adicione o código abaixo. Ele será o responsável por identificar a colisão da bola com a parede, e inverter as velocidades Y.

```
void OnCollisionEnter2D(Collision2D collision)
 {
    if (collision.gameObject.CompareTag("Wall"))
    {
       Vector2 newVelocity = rb.velocity;

       newVelocity.y = -newVelocity.y;
       rb.velocity = newVelocity;

    }
}
```

Paredes

Agora vamos adicionar o seguinte código dentro do método **OnCollisionEnter2D**, no script BallController.

Ele fará com que seja identificada a colisão com o jogador ou inimigo.

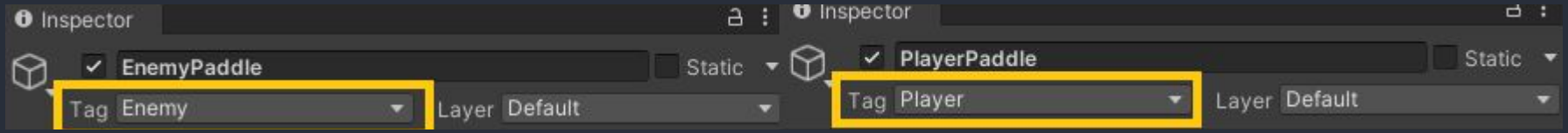
```
if (collision.gameObject.CompareTag("Player") || collision.gameObject.CompareTag("Enemy"))
{

        rb.velocity = new Vector2(-rb.velocity.x, rb.velocity.y);

}
```

Paredes

Precisamos agora adicionar a tag Enemy e Player, ao gameobject EnemyPaddle e PlayerPaddle.



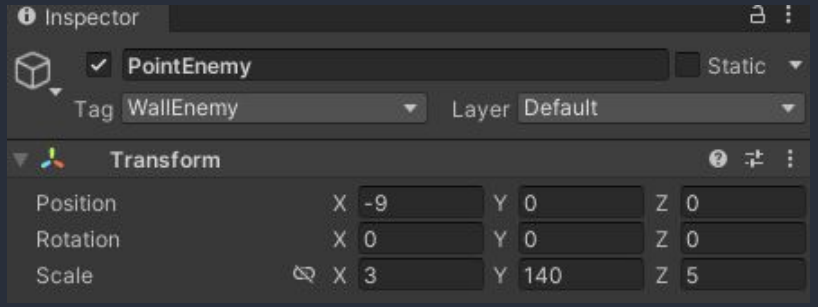
Pontos

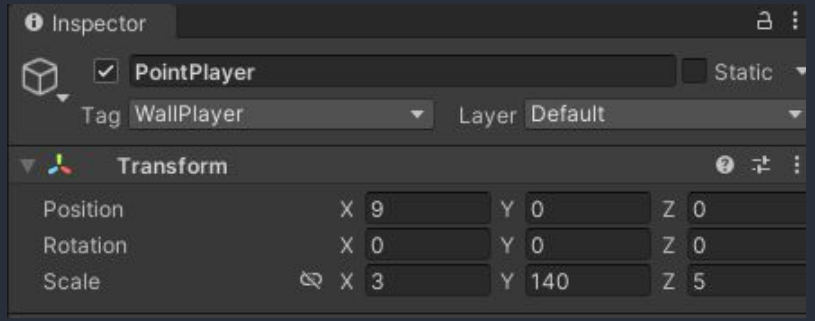
Agora vamos adicionar o sistema de contagem de pontos.

Duplique os gameobjects Wall. Em um, coloque o nome de PointPlayer, e em outro EnemyPlayer.

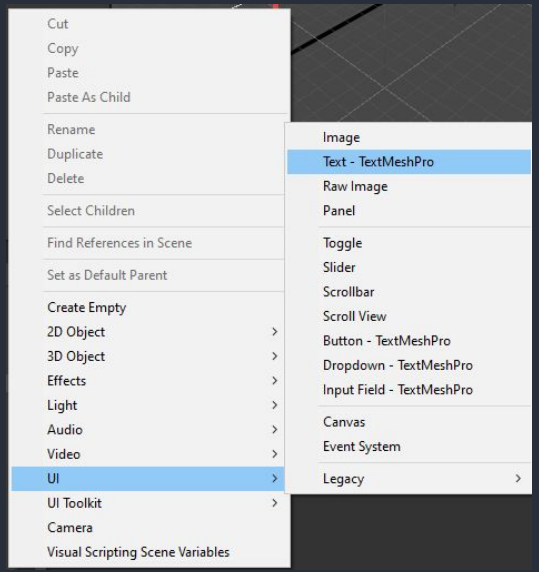
Eles ficarão posicionados atrás de cada jogador, na extremidade da tela. Quando a bola encostar neles, um ponto será marcado: ou do Player, ou do Inimigo.

Também adicione as tags WallEnemy/WallPlayer, como na imagem abaixo.





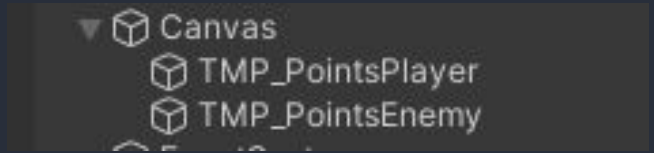
Pontos



Clique com o botão direito na hierarquia -> UI -> Text - TextMeshPro.

Repita o processo duas vezes. Cada objeto criado, de o nome de uma pontuação:

- TMP_PointsPlayer
- TMP_PointsEnemy

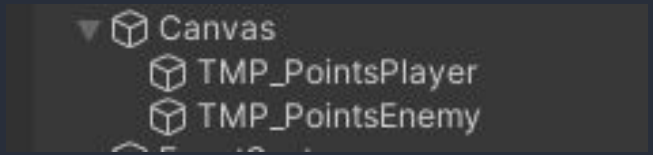


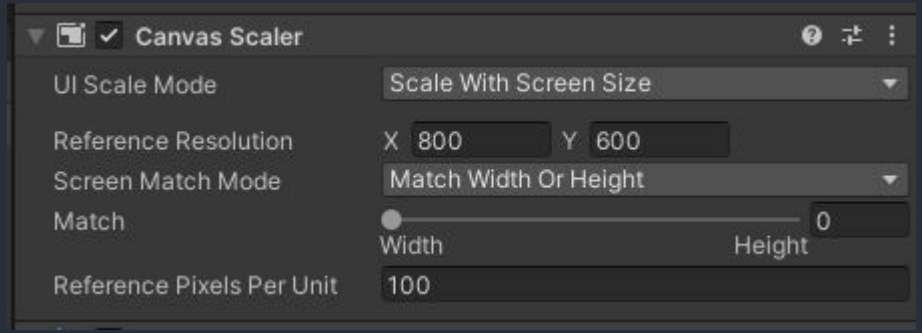
Pontos

Repare que os objetos serão criados dentro de outro objeto chamado **Canvas.**

Selecione o Canvas, procure pelo script **Canvas Scaler.** Mude a configuração para ficar igual a imagem abaixo: **UI Scale Mode: Scale With Screen Size.**

Isso fará com que a UI seja sempre escalada proporcionalmente de acordo com o tamanho de tela.





Pontos

No script GameManager, adicione estas 2 variáveis:

```
public int playerScore = 0;
public int enemyScore = 0;

public TextMeshProUGUI textPointsPlayer;
public TextMeshProUGUI textPointsEnemy;
```

Pontos

Dentro do método ResetGame, no GameManager, vamos resetar a pontuação. Adicione:

```
playerScore = 0;
enemyScore = 0;

textPointsEnemy.text = enemyScore.ToString();
textPointsPlayer.text = playerScore.ToString();
```

Pontos

Agora, vamos adicionar esses dois métodos. Eles serão úteis para controlar a pontuação de cada jogador.

```
public void ScorePlayer()
{
    playerScore++;
    textPointsPlayer.text = playerScore.ToString();
}

public void ScoreEnemy()
{
    enemyScore++;
    textPointsEnemy.text = enemyScore.ToString();
}
```

Pontos

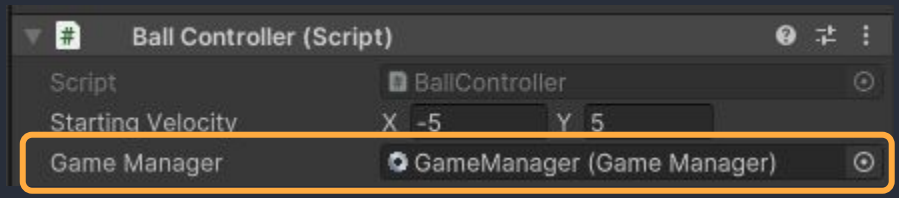
Dentro de BallController, no método OnCollisionEnter2D, adicionar:

```
if (collision.gameObject.CompareTag("WallEnemy"))
{
    gameManager.ScorePlayer();
    ResetBall();
}
else if (collision.gameObject.CompareTag("WallPlayer"))
{
    gameManager.ScoreEnemy();
    ResetBall();
}
```

Pontos

Precisamos também colocar uma referência do gameManager no começo do script. E depois fazer a referência no editor da Unity.

```
public GameManager gameManager;
```



Inimigo Automático

Podemos adicionar agora um movimento ao inimigo. Para que possamos jogar com apenas um jogador.

Para isso, crie um script com o nome de EnemyPaddleController. Adicione o código abaixo nele:

```
private Rigidbody2D rb;
public float speed = 3f;

private GameObject ball;

void Start()
{
    rb = GetComponent<Rigidbody2D>();
    ball = GameObject.Find("Ball"); // Encontra o objeto da bola na cena
}
```

Inimigo Automático

Agora adicione esse trecho ao script criado.

```
  void Update()
   {
       if (ball != null)
       {
           float targetY = Mathf.Clamp(ball.transform.position.y, -4.5f, 4.5f); // Limita a posição Y
           Vector2 targetPosition = new Vector2(transform.position.x, targetY);
           transform.position = Vector2.MoveTowards(transform.position, targetPosition, Time.deltaTime
* speed); // Move gradualmente para a posição Y da bola
       }
   }
```

Fim de jogo

Agora precisamos adicionar a lógica de final de jogo. Ou seja, quando um jogador atingir um certo número de pontos, o jogo será reiniciado.

Fim de jogo

Para isso, vamos adicionar os scripts abaixo no GameManager:

```
public int winPoints;


public void CheckWin()
  {
     if(enemyScore >= winPoints || playerScore >= winPoints)
     {
        ResetGame();
     }
  }
```

Fim de jogo

Então, dentro de cada função de score, adicionamos a chamada do método CheckWin.

```
public void ScorePlayer()
{
    playerScore++;
    textPointsPlayer.text = playerScore.ToString();
    CheckWin();
}

public void ScoreEnemy()
{
    enemyScore++;
    textPointsEnemy.text = enemyScore.ToString();
    CheckWin();
}
```

Fim

Chegamos ao fim de nossa jornada na criação do jogo Pong na Unity. Através de empenho e aprendizado, transformamos conceitos abstratos em um jogo interativo. Parabéns por essa conquista! Seu jogo Pong está pronto para ser jogado e compartilhado. Este é apenas o começo de suas habilidades no desenvolvimento de jogos. Aguardamos ansiosamente suas futuras criações emocionantes. Parabéns mais uma vez e continuem explorando novos horizontes nos jogos eletrônicos.

Exercício

O principal exercício deste módulo é o desenvolvimento do jogo Pong como demonstrado em aula.

Adicione ajustes e features a mais conforme desejar ao jogo.