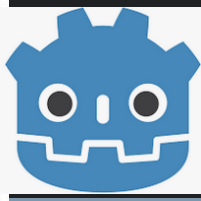
Detectando Colisões: RayCast2D

Prof. Thiago Felski Pereira, MSc.

Visão Geral

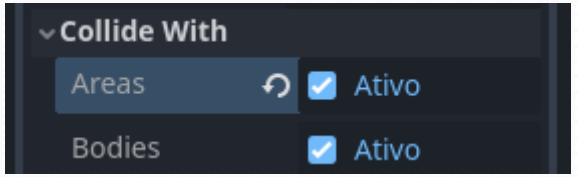
- O jogo está começando a tomar forma.
  - Temos um objeto **Jogador** que anda em todas as direções em grade.
    - Como o movimento de um personagem em um tabuleiro.
  - Também temos objetos **Parede** que irá permitir desenhar nosso cenário.
  - Agora precisamos fazer esses dois objetos interagirem, pois o jogador está ignorando a parede e entrando nela.
    - Para isso iremos incluir um `RayCast2D` no **Jogador** e programa-lo para verificar se tem algo no caminho do seu movimento.

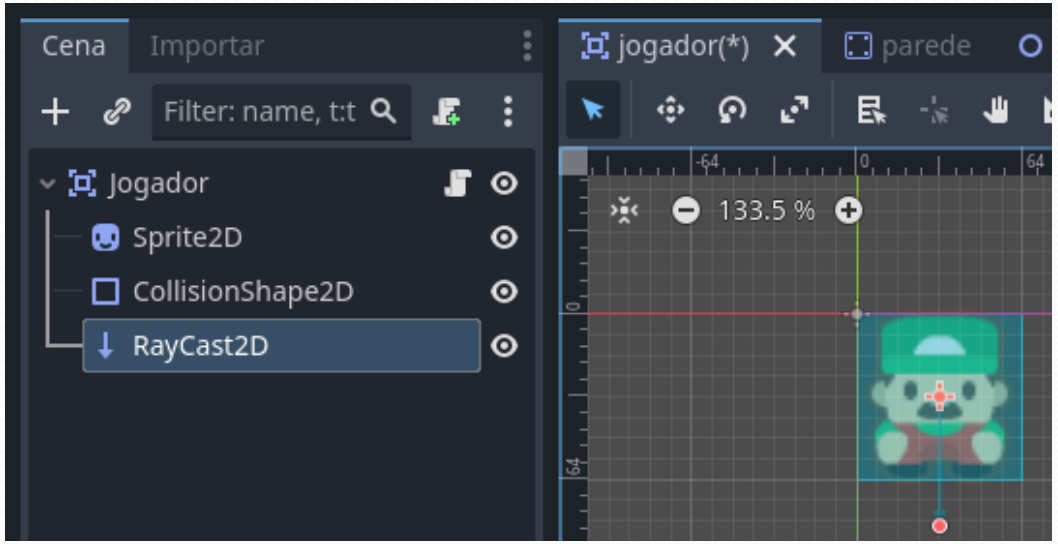


UNIVALI

Cena Jogador (detectar colisão)

- Usaremos um `Raycast2D` para resolver o problema encontrado no teste anterior.
  - Não há nada que impeça o **Jogador** de entrar na área da **Parede**.
  - O `Raycast2D` vai permitir verificar o que tem no caminho do **Jogador**.
- Selecione o Jogador na cena **Jogador**.
  - Adicione um nó `Raycast2D` a cena **Jogador**.
  - Posicione o `Raycast2D` para o centro do **Jogador**.
  - No Inspetor do Raycast2D ative a colisão com áreas, usaremos essa funcionalidade no futuro.





UNIVALI

Cena Jogador (detectar colisão)

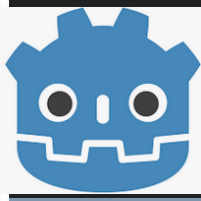
- No `Script` do **Jogador**.
  - Crie uma variável privada do tipo `RayCast2D` e chame-a de *ray*.

```
private RayCast2D ray;
```

  - Na função `_Ready()` carregue a variável *ray*.

```
public override void _Ready() {
    ray = GetNode<RayCast2D>("RayCast2D");
}
```

    - Usamos a função `GetNode` sempre que queremos controlar as ações de um nodo em nosso código.



UNIVALI

Cena Jogador (detectar colisão)

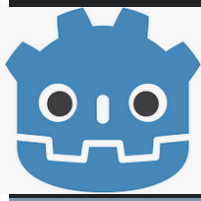
- No `Script` do **Jogador**.
  - A função `IsColliding()` do `RayCast2D` retorna verdadeiro existe algum objeto colidindo no trajeto.

  - *Mas como o código sabe o trajeto que o **Jogador** está indo?*
  - Os comandos, a seguir, são utilizados para verificar a direção desejada no momento do movimento.

```
ray.TargetPosition = direcao * area_quadrado;
ray.ForceRaycastUpdate();
```

  - Dessa forma, permitiremos que o objeto se mova sempre que não estiver colidindo.
  - O comando, a seguir, é utilizado para executar o movimento. Então precisamos testar se o objeto não está colidindo antes de permitir que ele execute esse código.

```
Position += direcao * area_quadrado;
```



UNIVALI

Atividade

- No `Script` do **Jogador**.
  - Só permita a execução do código, a seguir, se o Jogador não estiver colidindo.

```
Position += direcao * area_quadrado;
```

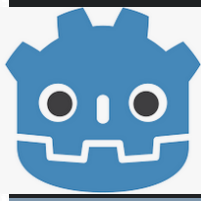
  - **Dica1**: Atualize a trajetória do `RayCast2D` com os seguintes comandos.

```
ray.TargetPosition = direcao * area_quadrado;
ray.ForceRaycastUpdate();
```

  - **Dica2**: O comando que verifica se o objeto está colidindo é:

```
ray.IsColliding()
```

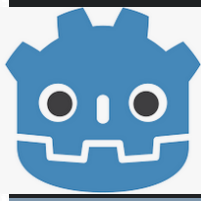
    - Resulta em verdadeiro se está colidindo.
    - Resulta em falso se não está colidindo.
    - Dessa forma, `!ray.IsColliding()` resulta em verdadeiro se não está colidindo.
    - Teste seu código: se tudo estiver correto, o **Jogador** não conseguirá mais entrar nas **Paredes**.



UNIVALI

Cena Caixa

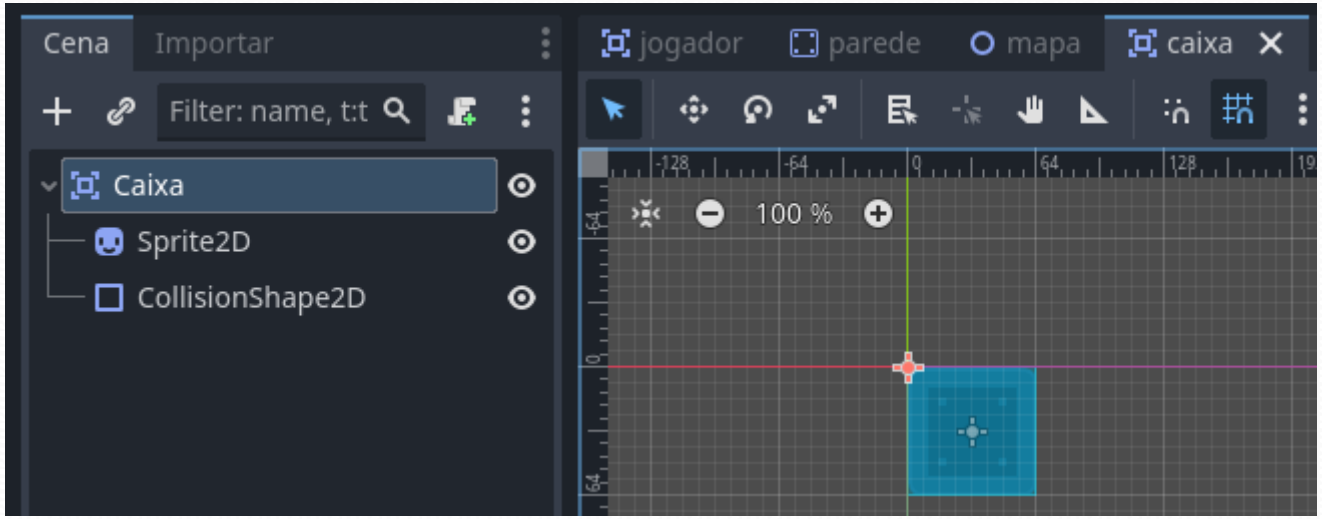
- Iremos criar caixas para o nosso jogo.
  - O **Jogador** poderá empurrá-las para qualquer direção livre.
    - Uma **Caixa** não poderá ser empurrada para dentro da parede.
    - Uma **Caixa** não poderá ser empurrada para dentro de outra caixa.
    - O **Jogador** não poderá entrar na **Caixa**.
    - *Futuramente as caixas poderão ser empurradas para posições livres e para dentro de posições finais.*
- A Caixa será composta de:
  - `Area2D`: por isso ativamos o `RayCast2D` para detectar áreas.
  - `Sprite2D`: para colocarmos a imagem da **Caixa**.
  - `CollisionShape2D`: Para definir a área de colisão.
  - Como em outros objetos iremos definir o tamanho desse objeto como `64x64`.



UNIVALI

Cena Caixa

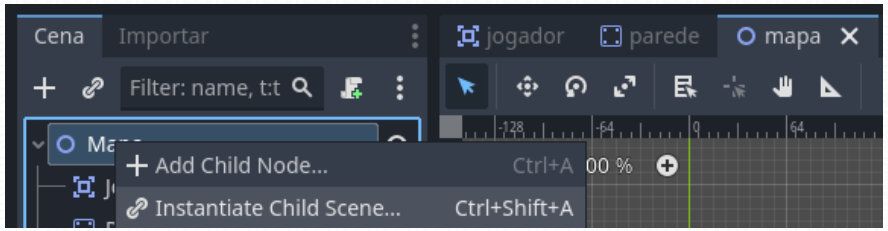
- Desenhando e definindo as áreas da **Caixa**.
  - Lembre-se de criar a **Caixa** como uma nova Cena.



UNIVALI

Cena Mapa

- Vá até o **Mapa**:
  - Inclua uma **Caixa**.
    - Lembre-se de criar a **Caixa** como uma nova Cena.



  - Teste o **Mapa**.
    - Reparou que a **Caixa** funciona como uma **Parede**?
    - O **Jogador** ainda não consegue empurrar a **Caixa**.

UNIVALI

Atividade

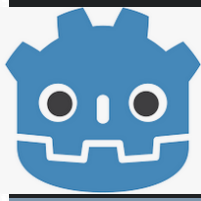
- Edite o Script do **Jogador**:
  - O primeiro passo para o **Jogador** conseguir empurrar a **Caixa** é saber que colidiu com ela.
  - PASSO 1: Faça o jogador imprimir um texto no console quando detectar que está colidindo com uma Area2D.
  - **Dica 1**: Exemplo de código de impressão.

```
GD.Print("Area2D bloqueando o caminho.");
```

  - **Dica 2**: Exemplo de verificação de colisão com a `Area2D`.
    - O código pode ficar ainda mais simples se você colocá-lo como complemento do teste que você utilizou para verificar se o **Jogador** não está colidindo.

```
ray.IsColliding() && ray.GetCollider() is Area2D
```

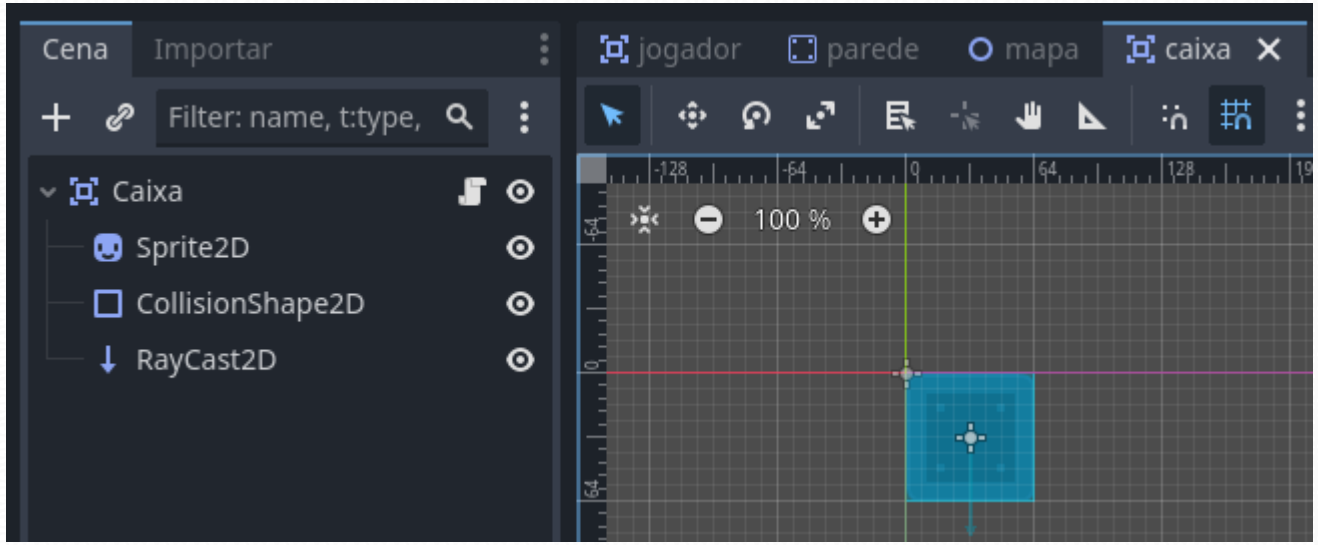
  - **OBS**: Note que será o **Jogador** quem perceberá a **Caixa** e não o contrário, precisaremos fazer a **Caixa** perceber que o **Jogador** está tentando empurrá-la.
  - Teste seu código: você deverá ver os textos aparecendo quando o jogador tenta empurrar.



UNIVALI

Cena Caixa

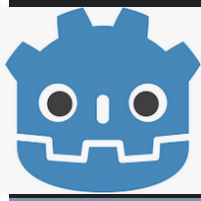
- Volte até a **Caixa**:
  - Inclua um `RayCast2D` na **Caixa**, do mesmo modo que fez com o **Jogador**.
    - Usaremos esse **ray** para identificar se a caixa estará entrando em colisão quando for empurrada.
    - Lembre-se te ativar: Inspetor ->Collide With -> Areas.
      - Pois iremos verificar se a caixa não está sendo empurrada para dentro de outra caixa.



Cena Caixa

- Insira e edite o Script da **Caixa**:
  - O Script da **Caixa** será uma simplificação do Script do **Jogador**, pois a caixa não poderá se mover por conta própria.
  - Inclua um Script na **Caixa** e inicie carregue o **ray** no Script.

```
private RayCast2D ray;
1 reference
public override void _Ready() {
    ray = GetNode<RayCast2D>("RayCast2D");
}
```

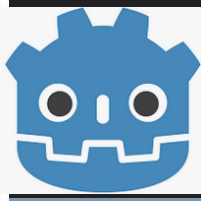


UNIVALI

Cena Caixa

- Edite o Script da **Caixa**:
  - Inclua uma função de Movimento que será chamada pelo **Jogador** quando ele colidir com a **Caixa**.
  - PASSO 2: Nosso desafio será fazer o **Jogador** executar esse código no script da **Caixa** que ele está tentando empurrar.

```
public void Empurrar () {
    GD.Print("Detectada: tentativa de empurrar");
}
```



UNIVALI

Atividade

- Edite o Script do **Jogador**:
  - Na atividade anterior nós verificamos se o **Jogador** estava colidindo com uma `Area2D`, agora queremos identificar com qual área ele está realmente colidindo.
  - PASSO 3: Chame a função empurrar da Caixa que o jogador está colidindo.
    - Comece armazenando o objeto o objeto que colidiu com o jogador em uma variável. Precisaremos garantir que o objeto foi guardado como uma Area2D, pois é lá que colocamos o script da Caixa.

```
Area2D colisor =  (Area2D)ray.GetCollider();
```

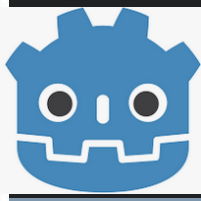
ou

```
Area2D colisor =  ray.GetCollider() as Area2D;
```

    - Chame a função da Caixa que você guardou na variável colisor.

```
colisor.Call("Empurrar");
```

  - Teste seu código!

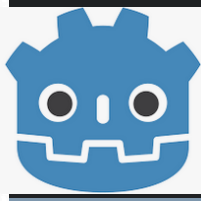


UNIVALI

Cena Caixa

- Melhorando o Script da **Caixa**:
  - Vamos fazer a caixa se mover quando o jogador chamar a função `Empurrar`.
  - PASSO 4: Vamos melhorar a função `Empurrar` recebendo dois parâmetros importantes do **Jogador**.
    - `direcao`: a direção que o **Jogador** está empurrando.
    - `area_quadrado`: a distância que a **Caixa** deve ser empurrada.

```
public void Empurrar (Vector2 direcao, int area_quadrado) {
    GD.Print("Detectada: tentativa de empurrar");
}
```



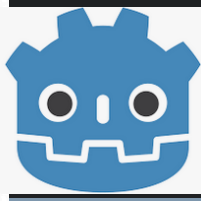
UNIVALI

Cena Caixa

- Melhorando o Script da **Caixa**:
  - Vamos fazer a caixa se mover quando o jogador chamar a função `Empurrar`.
  - PASSO 5: Vamos mover a **Caixa** utilizando os parâmetros recebidos no passo anterior.

```
public void Empurrar (Vector2 direcao, int area_quadrado) {
    GD.Print("Detectada: tentativa de empurrar");
    Position += direcao * area_quadrado;
}
```

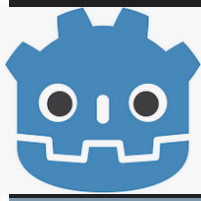
  - Teste seu código!



UNIVALI

Atividade

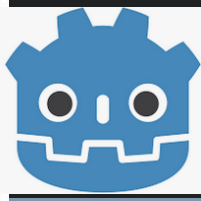
- Corrija os problemas encontrados no teste anterior:
  - Quando o **Jogador** chama a função `Empurrar` ele deveria se mover também, mas realmente for empurrada.
    - Precisamos retornar ao Jogador uma resposta a tentativa de empur
  - A **Caixa** pode ser empurrada para dentro da **Parede** ou para dentro de outra **Caixa**.
    - Inclua mais uma **Caixa** no seu **Mapa** para fins de teste.
    - A **Caixa** tem um `RayCast2D` que podemos utilizar para verificar se o caminho está livre.



UNIVALI

Desafio

- Após corrigir os problemas anteriores:
  - Crie uma `Area2D` para ser o ponto final das **Caixas**.
  - Crie todas as regras para as **Caixas** interagirem com as posições finais.
  - Detecte se a quantidade de **Caixas** que está na posição final é igual ao número de **Caixas** no seu **mapa**.
    - Se sim, imprima vitória.



UNIVALI

Obrigado pela atenção!