

Movendo 2D em Grade

Prof. Thiago Felski Pereira, MSc.

Visão Geral

- O movimento em grade é um recurso muito utilizado em jogos 2D, pois:
 - Simplifica os movimentos.
 - Simplifica as animações.
 - Facilita a construção de regras.
- Essas vantagens são bastante úteis para que o jogo possa funcionar em plataformas com limitações sem apresentar latência (lag) e reduzindo o tamanho do programa.



Preparando o ambiente

- Crie uma pasta para seu jogo.



Este Computador > DATA (D:) > Workspace > GODOT > CAU_Grade2D

- Eu chamei a minha de **CAU_Grade2D**, mas vocês podem chamar como quiserem.

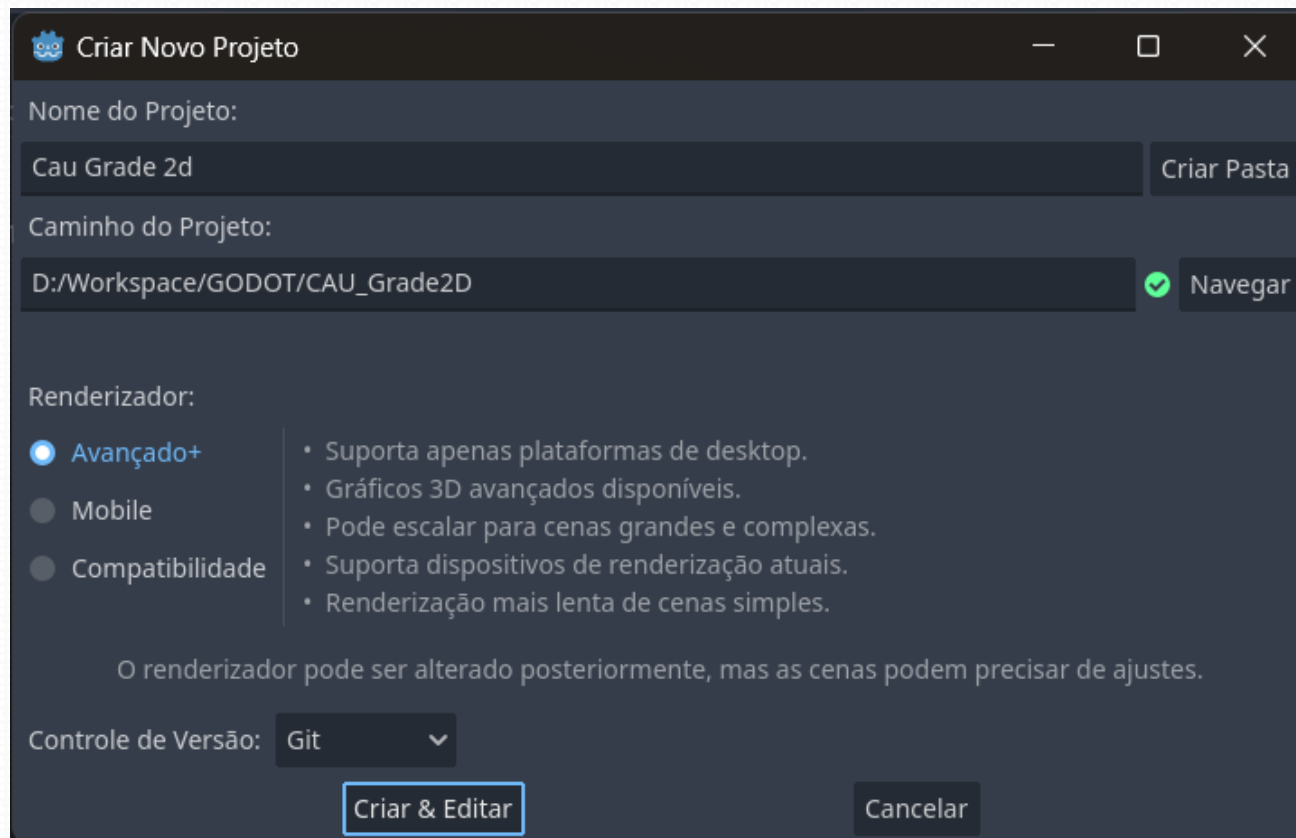


[3]



Preparando o ambiente

- Abra o aplicativo do Godot
 - Lembre-se de extraí-lo do .zip
- Crie um novo projeto, escolhendo um nome e a pasta que você criou para ele



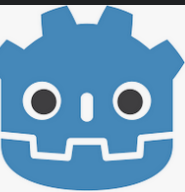
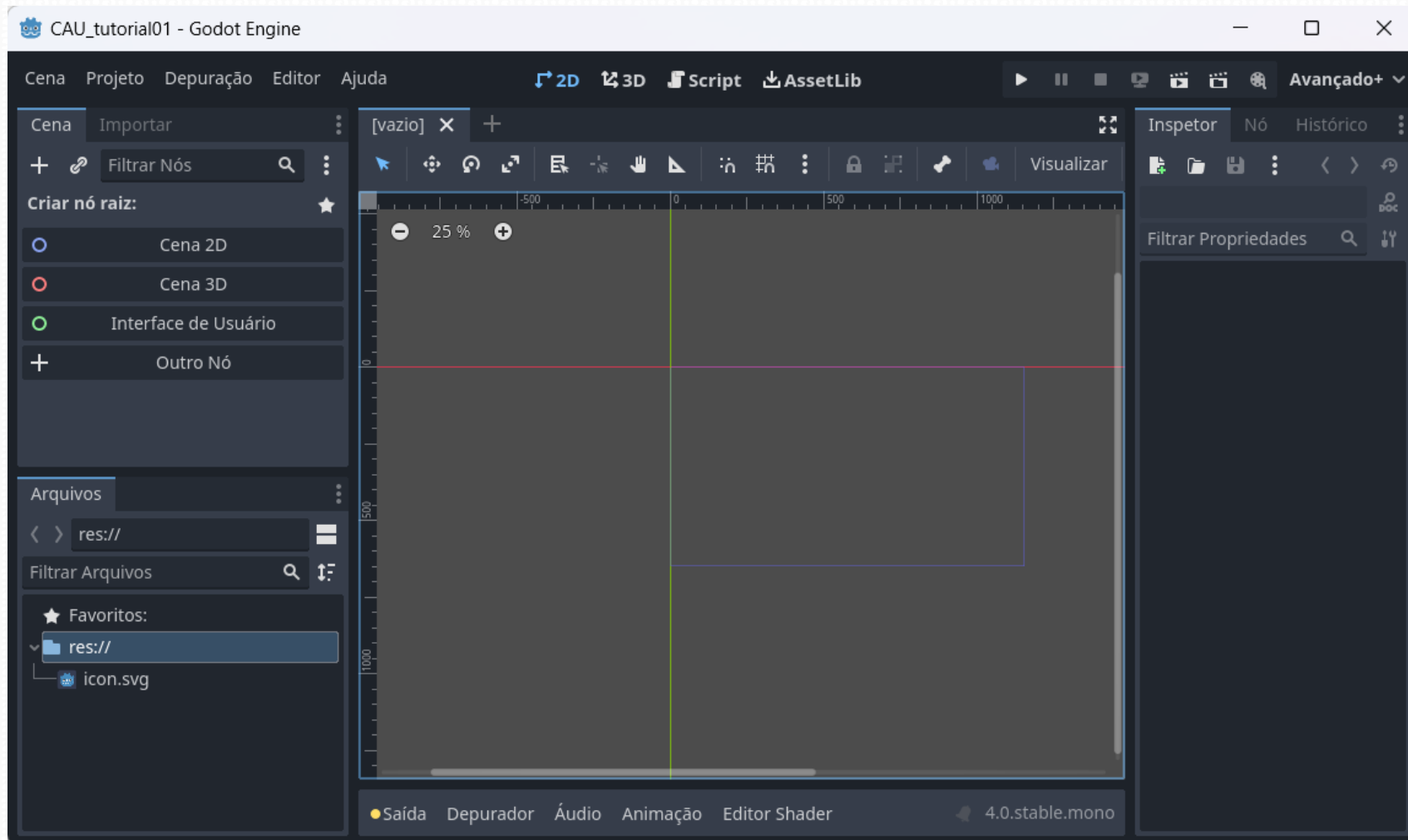
The screenshot shows the 'Criar Novo Projeto' (Create New Project) window in Godot. The window has a title bar with the Godot logo and the text 'Criar Novo Projeto'. It contains the following fields and options:

- Nome do Projeto:** A text field containing 'Cau Grade 2d' and a 'Criar Pasta' button.
- Caminho do Projeto:** A text field containing 'D:/Workspace/GODOT/CAU_Grade2D' and a 'Navegar' button with a green checkmark icon.
- Renderizador:** Three radio buttons: 'Avançado+' (selected), 'Mobile', and 'Compatibilidade'. To the right of 'Avançado+' is a list of features:
 - Suporta apenas plataformas de desktop.
 - Gráficos 3D avançados disponíveis.
 - Pode escalar para cenas grandes e complexas.
 - Suporta dispositivos de renderização atuais.
 - Renderização mais lenta de cenas simples.
- A note below the renderizer options: 'O renderizador pode ser alterado posteriormente, mas as cenas podem precisar de ajustes.'
- Controle de Versão:** A dropdown menu showing 'Git'.
- At the bottom are two buttons: 'Criar & Editar' (highlighted with a blue border) and 'Cancelar'.



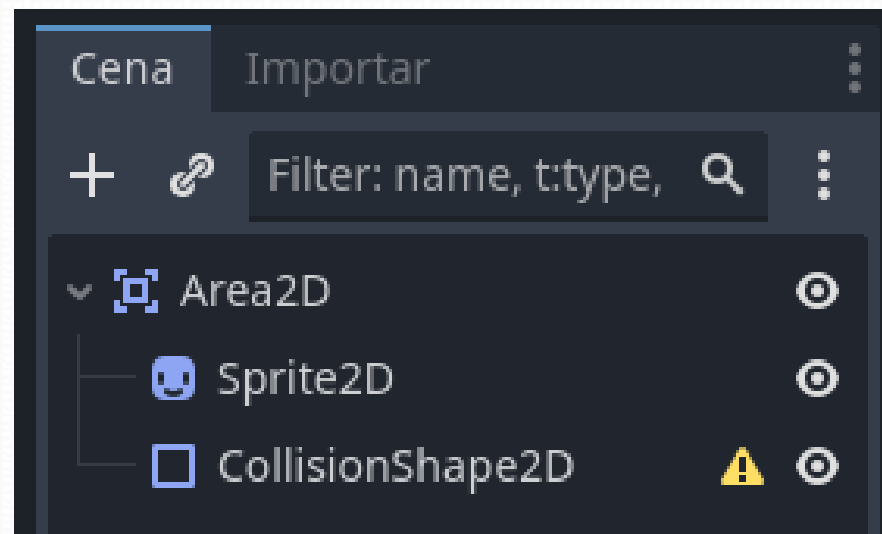
Preparando o ambiente

- Configure o ambiente para 2D, pois assim será nosso projeto



Desenvolvendo o jogo

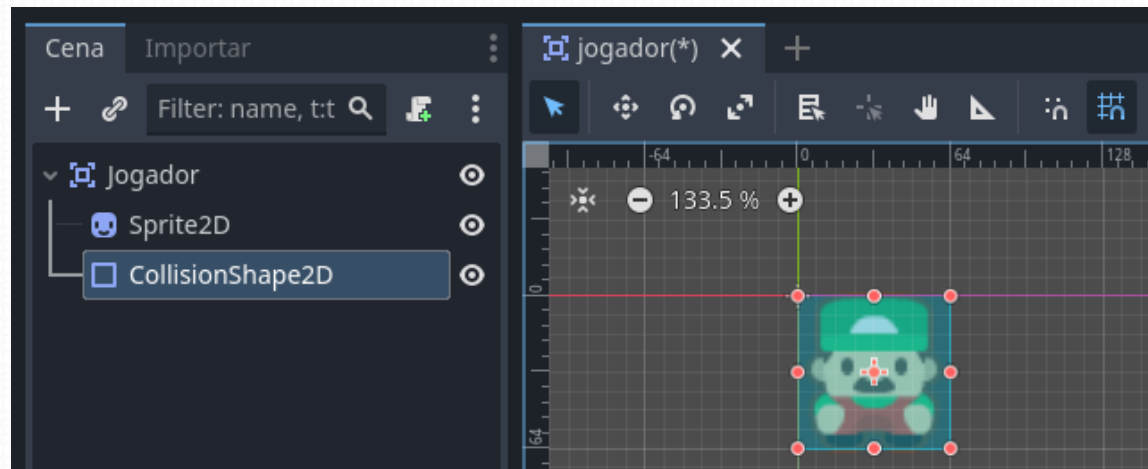
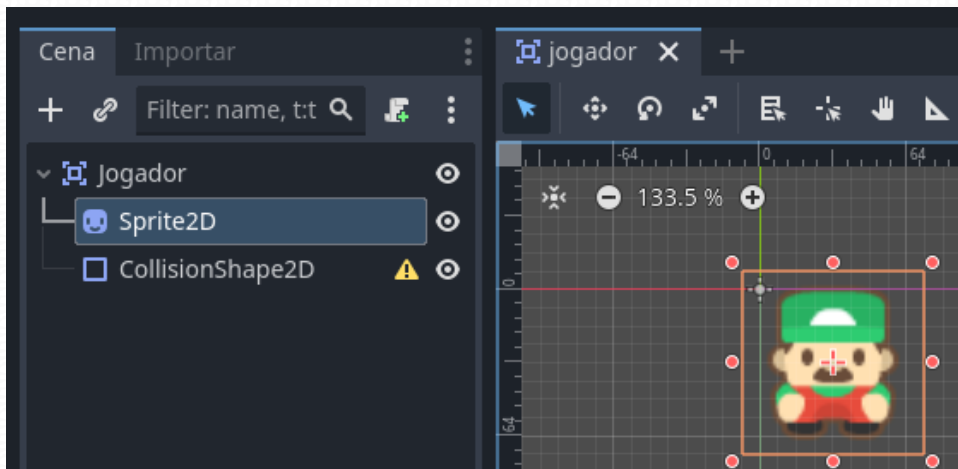
- Vamos criar nosso **Jogador**.
 - Ele será composto de uma `Area2D` com `Sprite2D` e `CollisionShape2D`
 - `Area2D`: nos permitirá tratar as colisões que acontecerem.
 - Passar pela parede ou empurrar uma caixa.
 - `Sprite2D`: nos permitirá colocar um desenho para o personagem.
 - `CollisionShape2D`: desenhar a área de colisão que não precisa ser igual ao desenho.
 - Note que ao inserir esse nodo aparecerá um sinal de alerta, nos avisando que ainda não definimos a forma de colisão.



[6]

Desenvolvendo o jogo

- Vamos editar nosso **Jogador**.
 - Altere o nome `Area2D` para `Jogador` e salve o projeto.
 - Copie a pasta “imagens sokoban” para o diretório do projeto.
 - Imagem do jogador e arraste para o nodo `Sprite2D`.
 - Ligue a guia de grade, pois todos os nossos objetos terão o mesmo tamanho.
 - Ajuste o jogador para ficar no meio e do tamanho de um quadrado (64x64).
 - No Inspetor do `CollisionShape2D`, mude o Shape de vazio para novo `RectangleShape2D` e ajuste para o tamanho do quadrado do jogador.



Desenvolvendo o jogo

- Vamos criar e editar o Script do nosso **Jogador**.

- Adicione um Script ao nó **Jogador**.
- Delete a função `_Process`, pois não iremos usá-la.
- Adicione uma variável inteira ao projeto.
 - `private int area_quadrado = 64;`
- Adicione uma variável Vector2 ao projeto.
 - `private Vector2 direcao;`

```
public partial class jogador : Area2D {  
    4 references  
    private int area_quadrado = 64;  
    4 references  
    private Vector2 direcao;  
    1 reference  
    public override void _Ready() {  
  
    }  
}
```



{ 8 }



Desenvolvendo o jogo

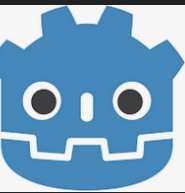
- Vamos criar e editar o `Script` do nosso **Jogador**.
 - Crie uma função `Movimento()`
 - Funções são uma maneira útil de separar as funcionalidades no nosso código em partes mais fáceis de entender e editar.
 - As funções poderão ser chamadas pelo seu nome, irão executar e retornarão para o código que a chamou.
 - Podemos chamar qualquer função dentro da nossa classe. Além disso aprenderemos a chamar funções de outros locais.

```
public void Movimento () {  
    //detecta a direção desejada  
    if (Input.IsActionJustPressed("cima")) {  
        |     direcao = new Vector2(0,-1);  
    }  
}
```



Atividade

- Implemente os comandos para o jogador ir nas 4 direções básicas: cima, baixo, esquerda e direita.



(10)



UNIVALI

Desenvolvendo o jogo

- Vamos criar e editar o `Script` do nosso **Jogador**.
 - Após criar os movimentos para todas as direções, vamos realizar o movimento na direção desejada.
 - Como nosso movimento é em grade, queremos que o movimento seja do tamanho do quadrado. Para isso utilizaremos o seguinte comando dentro da função de movimento:
 - `Position += direcao * area_quadrado;`
 - Coloque o código após testar todas as direções.



(11)



UNIVALI

Desenvolvendo o jogo

- Vamos criar e editar o `Script` do nosso **Jogador**.
 - Note que o movimento ainda não é realizado, isso ocorre porque nossa função de movimento nunca é chamada.
 - Ao invés de ficarmos testando o movimento a cada frame, utilizaremos uma função que irá detectar se uma tecla foi pressionada para chamar a função de movimento.
 - Coloque a função, a seguir, no `Script` **Jogador**.
 - Ela deve ficar fora da função de movimento, mas dentro da classe **Jogador**.

```
public override void _UnhandledInput(InputEvent @event)
{
    if (@event is InputEventKey eventKey)
    {
        if (eventKey.Pressed)
        {
            Movimento();
        }
    }
}
```



