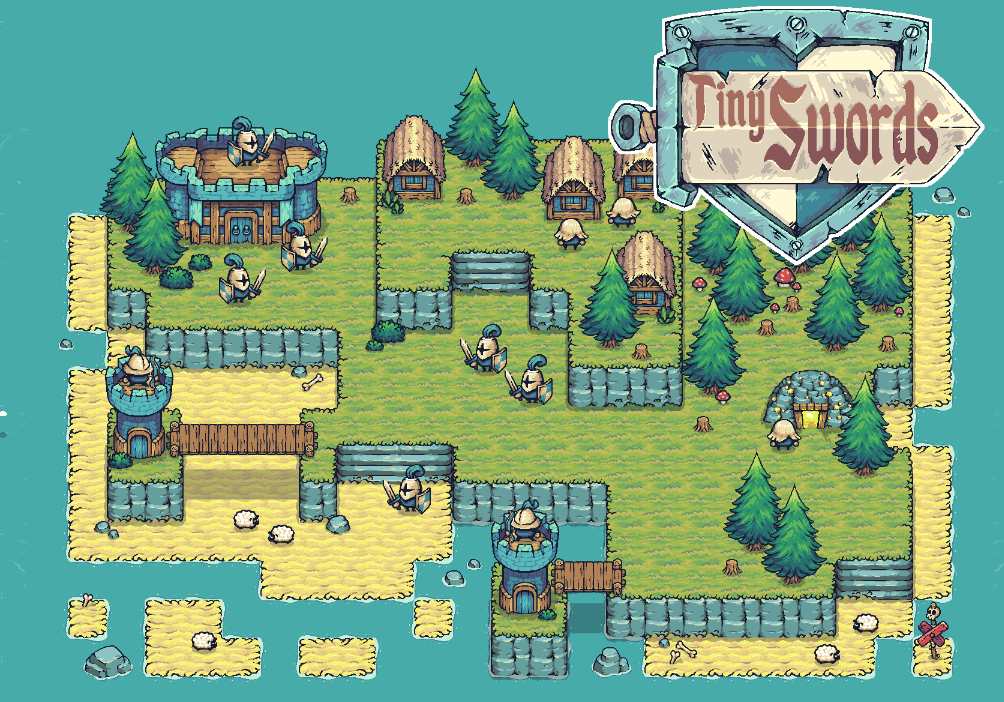
Vamos aprender usando o pacote Tiny Swords ([Espadas minúsculas por Pixel Frog (itch.io)](https://pixelfrog-assets.itch.io/tiny-swords))

O pacote é imenso, existe um mar de possibilidades!



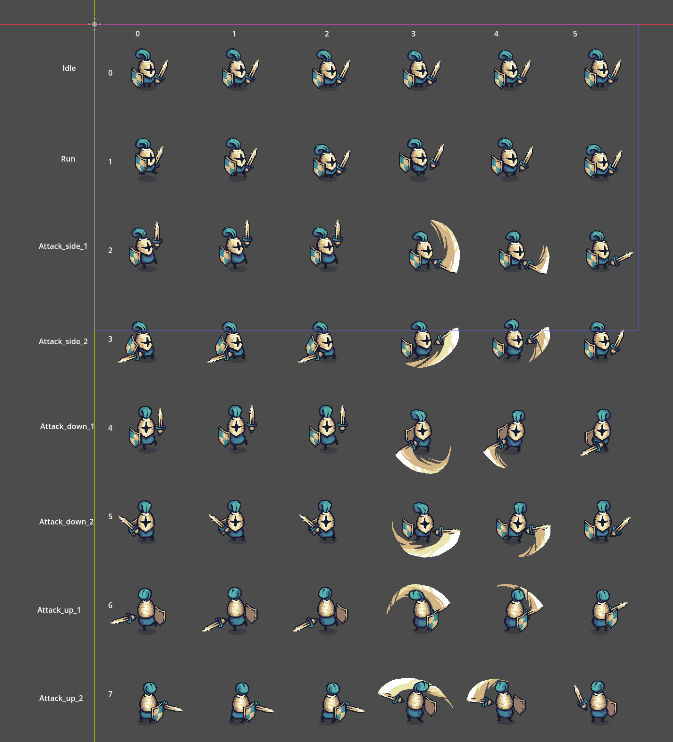
**Iremos primeiro criar o jogador**

Usando a pasta Factions para escolher. Iremos usar o **Warrior** como personagem.



Cada linha representa uma animação e cada animação é uma ação.

Se perceber não tem **nenhuma animação para a esquerda**, mas é só ir na propriedade **offset> e marcar Flip H** que ele gira.



**Agora vamos dar passivamente vida:**

Primeiro iremos criar um novo **sprite2d** e arrastrar a imagem para la. Agora iremos informar para o godot que é “animado”

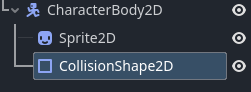
**Iremos em “Animation”:**

Iremos mexer no **H e V frames**. Como vimos anterior H temos 6 e V temos 8. Assim você vai ver que se transformar em 1 só.

E para mexer na animação que você quer você utiliza o **Frame coords.**



Vamos criar o **characterbody2d:**



Queremos colocar a origem do objeto seja os “pês” por conta da visão topdown. (movendo o sprite2d para ficar na origem dos pés do characterbody).

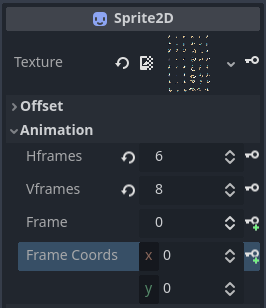
Iremos criar uma cena própria do player. Para personalizar ele sozinho.

**Agora vamos criar animações:**

Criaremos um novo node no player chamado: **AnimationPlayer**.

Nela iremos criar cada Animação separado. Primeiro iremos para a “**idle”**

Quando fazemos isso vai aparecer uma chave em Animation:

Assim vai clicando na chave que ele vai alterando.



Como saber se é para pular 0.1 por cada cena? Ir no assets e ver o recomendado.

Fazendo as cenas, podemos **mexer nela por script**. Criando um novo script.



**extends CharacterBody2D:** Esta linha indica que o script está estendendo a classe CharacterBody2D. Isso significa que este script herda todas as propriedades e métodos de CharacterBody2D, que é uma classe no Godot usada para personagens em 2D.

**@onready var animation\_player: AnimationPlayer = $AnimationPlayer:** Esta linha define uma variável animation\_player do tipo AnimationPlayer. O decorador @onready garante que esta variável seja inicializada apenas depois que o nó ao qual está anexado estiver pronto. O $AnimationPlayer busca um nó filho chamado AnimationPlayer e o atribui à variável.

**var is\_running: bool = false:** Define uma variável chamada is\_running do tipo booleano e a inicializa com false. Esta variável será usada para rastrear se o personagem está correndo ou não.

**func \_process(delta: float) -> void**: Define uma função chamada \_process que **é chamada em cada frame.** O parâmetro delta representa o tempo (em segundos) desde o último frame. O tipo de retorno é void, indicando que a função não retorna nada.

**if Input.is\_action\_just\_pressed("ui\_accept"):** Verifica se a ação "ui\_accept" (geralmente associada à tecla Enter ou Espaço) foi pressionada neste frame. Se a ação foi detectada, o código dentro do bloco if será executado.

**if is\_running:** Dentro do bloco if anterior, esta linha verifica se a variável is\_running é true.

**animation\_player.play("Idle"):** Se is\_running for true, o personagem estava correndo, então a animação Idle é iniciada usando o animation\_player.

**is\_running = false:** Define a variável is\_running para false, indicando que o personagem não está mais correndo.

**else:** Se is\_running for false, este bloco else será executado.

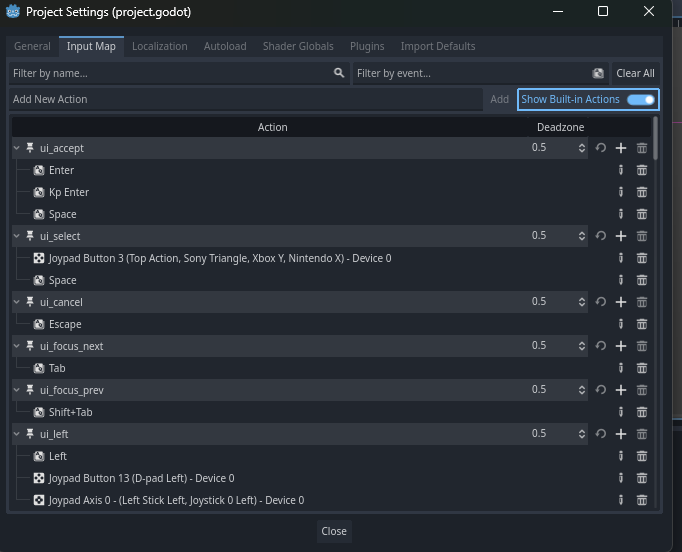
**animation\_player.play("run"):** Inicia a animação run usando o animation\_player, indicando que o personagem começou a correr.

**is\_running = true**: Define a variável is\_running para true, indicando que o personagem agora está correndo.

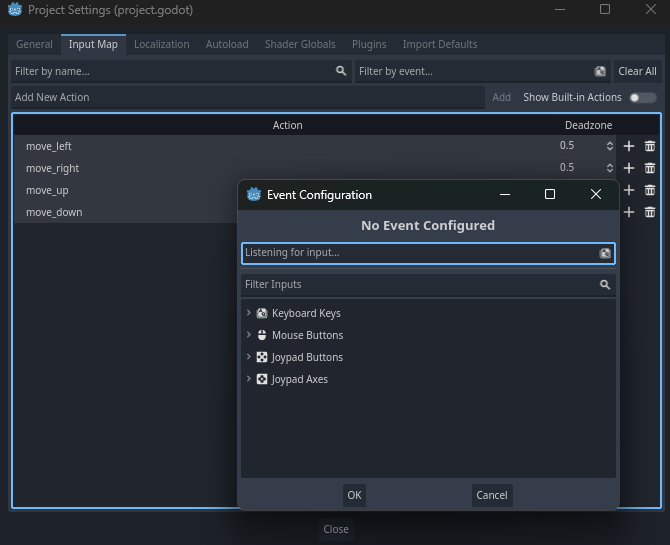
**Criando novas ações:**

Geralmente estamos usando as ações padrões “ui\_left”... Ações já prontas. Então como podemos criar novas ações ou fazer nossas próprias ações?

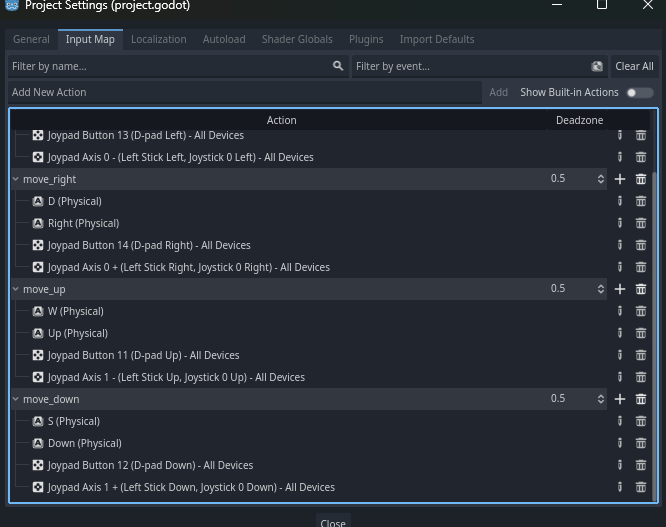
Então vamos em Project> Project Settings> Input Map (vai ter todas as ações do jogo).



Para criar uma nova ação vai em Add New Action e digitar o nome dessa nova ação..



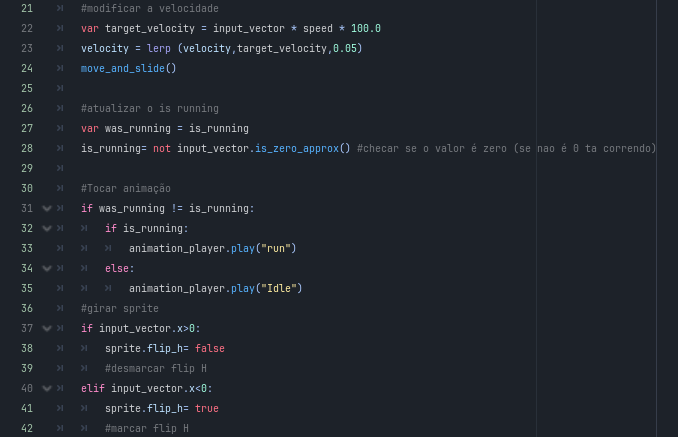
Triscando no + você escolhe o input (apenas triscando na tecla ele já vai direito)



E agora só mudar nosso Script.

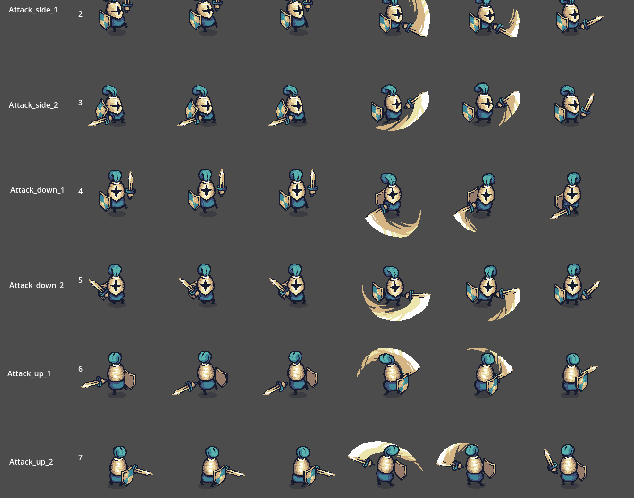
  
**\_phsics\_process** para ser chamada a cada frame da tela. E tudo que mexe com a física.

**Deadzone**: ignorar pequenos movimentos no input (joystick)



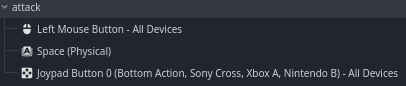
**Move\_and\_slide**= Esta linha aplica a física de movimento ao personagem, fazendo-o deslizar conforme a velocidade definida. É uma função do CharacterBody2D que lida com a movimentação do corpo no espaço 2D, incluindo colisões.

**Programando ataque do personagem:**



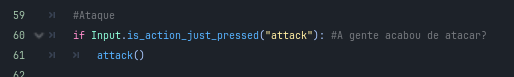
Criar nova animação igual feito anteriormente.

E criar uma ação para o ataque igual mover!



E criar no script o sistema de ataque.

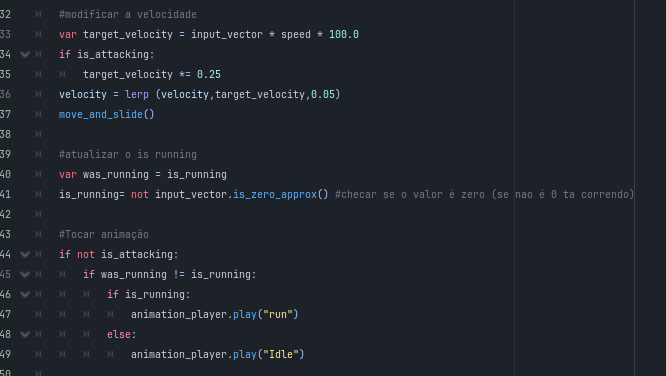
**1ª no \_Phisics\_process criamos o seguinte:**



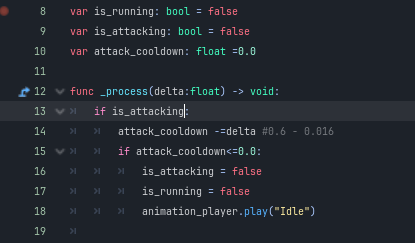
**2º Criamos a função de atacar:**



**3ª. Modificamos a velocidade enquanto tivermos atacando e colocamos para atualizar a animação enquanto não estiver atacando**



**4ª Criamos os padrões das variáveis e fazemos uma func \_process para verificar se esta atacando, colocamos um cooldown para o próximo ataque. Assim diminuindo uma hora vai chegar a 0. Então se o ataque for <=0 não estamos mais atacando.** Colocar que não estamos correndo e sim “idle”

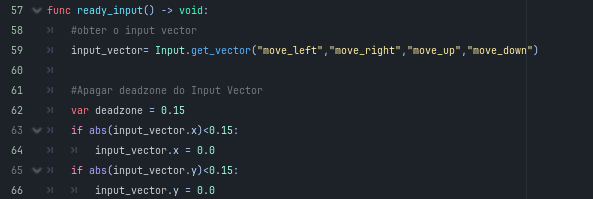


Falta a outra animação, ataque em cima e baixo. (podemos já fazer mas talvez fazemos durante o restante da trilha).

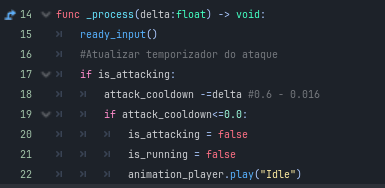
**Organizando o código e simplificando:**

1ª Coisa movemos o **input\_vector** para ser global para poder ser acessado em qualquer lugar. E também movendo para process (que não tem a ver com física).

2ª Movemos o **Apagar o deadzone** para o process e criamos uma função próprias para eles. (esses 2)

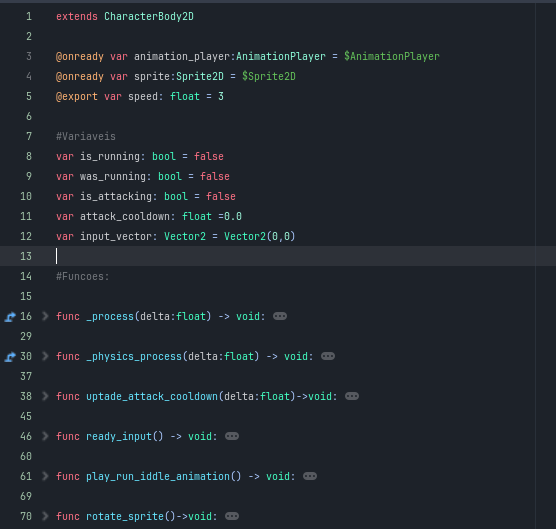


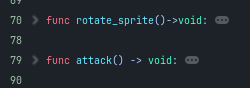
**E no process chamamos ela**



3ª Vamos mexer na \_phsics\_process. Primeiro vamos mexer no “atualizar is running” criando uma variável global para was\_running e movemos ela para a função que criamos ready input.

4ª No \_phsics\_proces também movemos o tocar animação, girar Sprite e ataque para o process. E criamos uma função separada e apenas chamamos no process.





**Assim ficou:**

**extends CharacterBody2D**

**@onready var animation\_player:AnimationPlayer = $AnimationPlayer**

**@onready var sprite:Sprite2D = $Sprite2D**

**@export var speed: float = 3**

**#Variaveis**

**var is\_running: bool = false**

**var was\_running: bool = false**

**var is\_attacking: bool = false**

**var attack\_cooldown: float =0.0**

**var input\_vector: Vector2 = Vector2(0,0)**

**#Funcoes:**

**func \_process(delta:float) -> void:**

**ready\_input()**

**#Processar ataque**

**uptade\_attack\_cooldown(delta)**

**if Input.is\_action\_just\_pressed("attack"): #A gente acabou de atacar?**

**attack()**

**#processar animação e rotação de sprite**

**play\_run\_iddle\_animation()**

**rotate\_sprite()**

**func \_physics\_process(delta:float) -> void:**

**#modificar a velocidade**

**var target\_velocity = input\_vector \* speed \* 100.0**

**if is\_attacking:**

**target\_velocity \*= 0.25**

**velocity = lerp (velocity,target\_velocity,0.05)**

**move\_and\_slide()**

**func uptade\_attack\_cooldown(delta:float)->void:**

**if is\_attacking:**

**attack\_cooldown -=delta #0.6 - 0.016**

**if attack\_cooldown<=0.0:**

**is\_attacking = false**

**is\_running = false**

**animation\_player.play("Idle")**

**func ready\_input() -> void:**

**#obter o input vector**

**input\_vector= Input.get\_vector("move\_left","move\_right","move\_up","move\_down")**

**#Apagar deadzone do Input Vector**

**var deadzone = 0.15**

**if abs(input\_vector.x)<0.15:**

**input\_vector.x = 0.0**

**if abs(input\_vector.y)<0.15:**

**input\_vector.y = 0.0**

**#atualizar o is running**

**was\_running = is\_running**

**is\_running= not input\_vector.is\_zero\_approx() #checar se o valor é zero (se nao é 0 ta correndo)**

**func play\_run\_iddle\_animation() -> void:**

**#Tocar animação**

**if not is\_attacking:**

**if was\_running != is\_running:**

**if is\_running:**

**animation\_player.play("run")**

**else:**

**animation\_player.play("Idle")**

**func rotate\_sprite()->void:**

**#girar sprite**

**if input\_vector.x>0:**

**sprite.flip\_h= false**

**#desmarcar flip H**

**elif input\_vector.x<0:**

**sprite.flip\_h= true**

**#marcar flip H**

**func attack() -> void:**

**#attack\_side\_1 e #attack\_side\_2 (as animações que criamos)**

**if is\_attacking:**

**return**

**#tocar a animação**

**animation\_player.play("attack\_side\_1")**

**attack\_cooldown= 0.6**

**# Marcar ataque**

**is\_attacking = true**

Coisas que eu fiz:

1ª Coloquei para o ataque ser random nas animações, ou é attack\_side\_1 ou attack\_side\_2.



Mas pq não colocar o ataque para cima e ataque para baixo?





