PROGRAMAÇÃO PARA JOGOS - CAMELOT

1 A HISTÓRIA

O jogo se passa em um mundo inspirado na lenda de Camelot, onde o personagem principal é mantido como prisioneiro, após ser capturado por Mordred, que busca pelo paradeiro de Arthur. A história se desenrola após Arthur ter sido gravemente ferido por Mordred e levado para Avalon para se curar, um fato desconhecido pelo antagonista.

Em meio à batalha desencadeada pela traição de Mordred, Arthur, gravemente ferido, reúne suas últimas forças. Antes de sucumbir à escuridão, Arthur faz com que seus cavaleiros façam um último juramento, de ir contra qualquer instinto de vingança e por fim perdoar Mordred, proibindo que qualquer um busque vingança em seu nome. Em seguida, Arthur é levado a Avalon, onde receberá cuidados para curar seus ferimentos, mas o tempo que levará é incerto. E assim, a Bretanha perdeu seu maior guerreiro, seu mais nobre rei e o melhor dos homens.

A compaixão de Arthur foi vista como fraqueza por Mordred, que então, começa a perseguir os seguidores de Arthur, interrogando-os e torturando-os em busca do paradeiro de seu Rei. Assim encontra-se o protagonista, em uma das masmorras de Mordred, suportando a interrogatórios e torturas, enquanto testemunha todos os seus irmãos sendo mortos, um após o outro.

O protagonista tem um sonho intenso com três figuras, antigos cavaleiros que lutaram ao seu lado, que lhe fazem questionamentos sobre suas motivações, falando sobre dor, honra e o peso das escolhas acerca de vingança e compaixão. Ao acordar, ele descobre que sua cela foi destruída, simbolizando sua luta interna pela liberdade e pela vingança.

1.1 O PROTAGONISTA, O JOGADOR E A VIOLÊNCIA

O protagonista está imerso na busca por vingança, especialmente contra Mordred que representa o mal absoluto. No entanto, o protagonista é constantemente atormentado por sentimentos de compaixão e perdão. Ele não apenas enfrenta a dor da perda de seus irmãos, mas também a luta interna entre honrar o juramento de proteger os inocentes e sucumbir à sedução da vingança. Essa dualidade se torna central na narrativa, desafiando-o a confrontar não apenas seu desejo de justiça, mas também os ecos da misericórdia que Arthur, seu rei, sempre defendeu. Compelindo o Protagonista a questionar se a vingança realmente trará paz ou se a verdadeira vitória reside em perdoar, mesmo aqueles que causaram tanto sofrimento. Essa luta interna entre vingança e compaixão oferece uma profundidade emocional, refletindo uma luta complexa e humana.

A luta interna entre vingança e compaixão proporciona uma profundidade emocional à jornada. À medida que o protagonista enfrenta as consequências da traição de Mordred, a sedução da vingança se torna uma presença constante, sussurrando promessas de justiça e alívio para a dor causada pela perda de seus irmãos. A raiva que o consome é uma resposta natural ao sofrimento, mas essa mesma raiva também se torna um veneno que ameaça destruir sua humanidade.

Essa dualidade se reflete na violência do jogo, onde cada ato de combate e cada derramamento de sangue carregam um peso significativo. A busca por vingança confere sentido e propósito a ações que, em outros contextos, poderiam parecer desnecessárias. Essa exploração da violência se torna um reflexo do conflito interno do protagonista, questionando se ele realmente está lutando por justiça ou apenas se deixando levar por um impulso destrutivo.

Por outro lado, a percepção do jogador em relação à violência pode ser complexa. Ao acompanhar a jornada do protagonista, o jogador é convidado a contemplar o custo emocional da vingança. A intensidade das cenas de combate pode evocar uma mistura de adrenalina e desconforto, forçando-o a refletir sobre a moralidade das ações do protagonista. Enquanto o heroi busca redenção, o jogador pode se sentir dividido entre a empatia por sua dor e a repulsa pela brutalidade do que testemunha.

Assim, a narrativa oferece uma experiência intensa e envolvente, onde a jornada do protagonista se desdobra em um ambiente de dor e redenção. A busca por vingança é apresentada não como um caminho simples, mas como uma jornada repleta de dilemas morais, revelando a profundidade da luta interna que define a essência humana.

2 O JOGO

O jogo se inspira em "Hotline Miami", apresentando uma jogabilidade rápida e tática, onde os jogadores enfrentam desafios intensos e devem realizar movimentos ágeis em combates frenéticos. A experiência é ainda mais intensificada por uma trilha sonora pulsante, que eleva a adrenalina das batalhas e cria uma atmosfera de urgência e fluidez nas ações. Cada confronto é uma dança entre estratégia e reflexos rápidos.

No entanto, as semelhanças param por aí, “Hotline Miami” inicia sua história com a provocativa pergunta "Do you like hurting other people?", estabelecendo um tom sombrio e brutal, usando da violência como um fim para si mesmo. O protagonista se vê imerso em um ciclo de violência, enquanto a narrativa gira em torno da natureza da moralidade em um mundo desolador. A busca por justiça é frequentemente ofuscada pela violência gratuita, sem uma reflexão profunda sobre o impacto emocional dessas ações.

Em contraste, Camelot se aprofunda na luta interna do protagonista entre vingança e compaixão. Ao buscar retribuição contra Mordred pela traição e pela dor infligida a seus irmãos, a narrativa oferece espaço para reflexões sobre honra e perdão. A violência, embora presente, é apresentada como um dilema moral, forçando o protagonista a confrontar as consequências emocionais de suas ações. A busca por justiça não se reduz a um impulso cego, mas se torna uma jornada repleta de complexidade e humanidade, utilizando da violência para questionar a verdadeira natureza da justiça e da compaixão.

3 AS MECÂNICAS

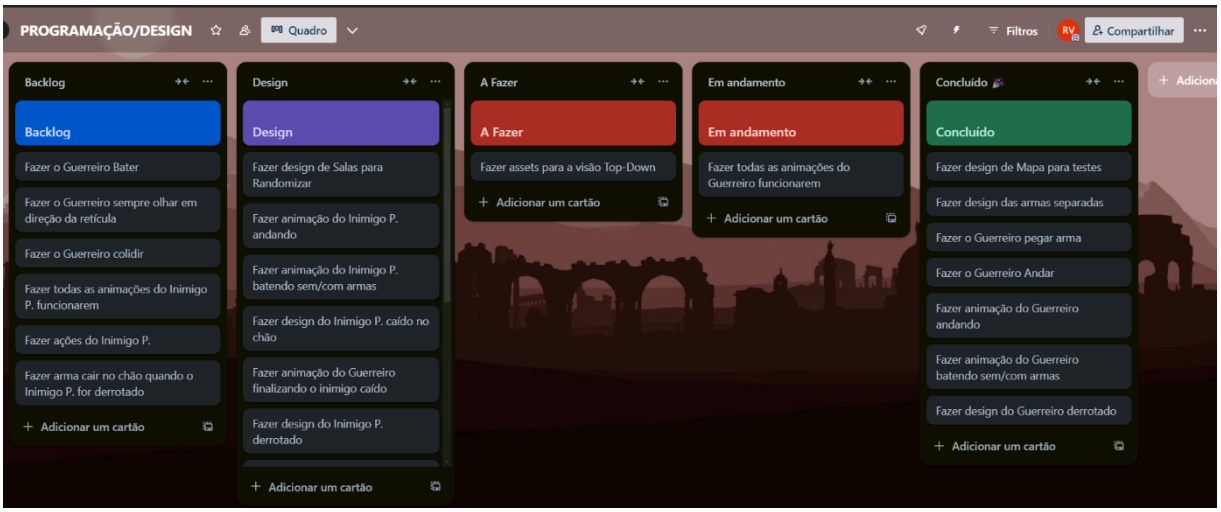
Os jogadores controlam o protagonista em uma série de níveis desafiadores, utilizando uma combinação de armas corpo a corpo e de fogo para eliminar inimigos em ambientes variados. A jogabilidade é caracterizada por um sistema de controle intuitivo e simples, o personagem pode andar, bater, arremessar e pegar armas.

O jogo apresenta uma dificuldade que influencia diretamente a experiência do jogador. Uma das mecânicas centrais é o tamanho do mapa: quanto maior o mapa, mais inimigos aparecem, elevando a complexidade e o desafio das confrontações. Além disso, a mecânica de morte instantânea significa que um único golpe pode resultar em falha, incentivando uma abordagem cautelosa. No entanto, a trilha sonora pulsante e o sistema de pontuação estimulam a rapidez nas ações, criando um equilíbrio entre estratégia e agilidade. O design dos níveis é meticulosamente elaborado para promover a experimentação, permitindo que os jogadores explorem uma variedade de estilos de combate e estratégias, enriquecendo ainda mais a experiência de jogo.

4 O ANDAMENTO

Estamos estruturando nosso desenvolvimento em duas frentes: uma focada em design e outra em programação. Os responsáveis por cada área colaboram ativamente, oferecendo suporte mútuo, mas sempre priorizando suas respectivas funções. Para gerenciar nosso progresso, adotamos o Kanban, combinado com uma adaptação do Scrum voltada para equipes pequenas. Estamos utilizando o GitHub para compartilhar o código e facilitar a colaboração entre as equipes.

Inicialmente, realizamos uma análise detalhada dos requisitos, tanto funcionais quanto não funcionais. Isso nos ajudou a entender as necessidades do jogo e a definir claramente o que deveria ser priorizado para o andamento eficiente do projeto. Utilizamos o Trello para criar um quadro visual que nos permite acompanhar o progresso das tarefas. Isso proporciona uma visão clara do que está sendo realizado e do que ainda precisa ser feito.



Mantemos uma comunicação constante entre as equipes, realizando reuniões semanais para discutir o progresso e resolver quaisquer impedimentos. Essa colaboração é essencial para garantir que o design e a programação estejam alinhados.

Estamos adotando um ciclo de iteração rápida, onde testamos protótipos e coletamos feedback constantemente. Isso nos permite fazer ajustes antes de avançar para etapas mais complexas, garantindo que a experiência do jogador esteja sempre no centro das nossas decisões.

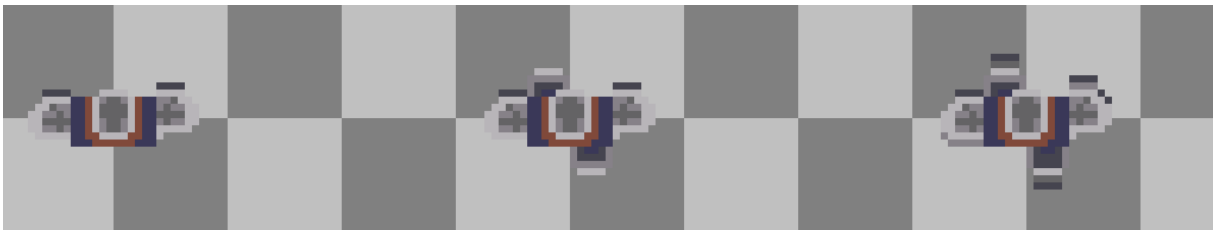
4.3 O DESIGN

Inicialmente, priorizamos a criação do personagem principal, focando em seu design geral e nas animações de movimento e morte. Em seguida, desenvolvemos as animações das armas, tanto para quando o personagem as empunha enquanto anda quanto ao realizar ataques. O design do personagem foi inspirado pela armadura “Elite Knight Set” da série *Dark Souls*.

4.3.1 O GUERREIRO

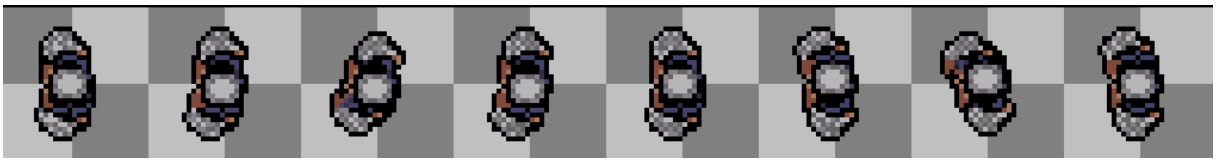
VERSÃO 1.0 – CAVALEIRO

A primeira versão serviu como um teste, permitindo que escolhemos o design geral do corpo do personagem, assim como a paleta de cores. No entanto, essa versão ficou muito pálida, o que foi corrigido em iterações posteriores, buscando uma melhor utilização das cores.



VERSÃO 2.0 – CAVALEIRO

Na segunda versão, abordamos os problemas identificados na versão anterior. O personagem ganhou uma forma menos quadrada e começou a parecer mais humano. As cores foram melhoradas e técnicas como dithering foram aplicadas para enriquecer a pixel art.



Em seguida, com esse modelo em mão foram feitas as animações de soco, e do personagem usando armas.

Dithering foi aplicado em todos os frames da animação para dar vida à armadura de cota de malha, realçando os detalhes da vestimenta do guerreiro. Além disso, foram adicionados efeitos em branco tanto para os socos quanto para a utilização do machado, proporcionando uma sensação de força e movimento, enriquecendo ainda mais a experiência visual durante as animações.

Nesta versão, identificamos alguns problemas, embora menores do que os da primeira. Um dos principais pontos a ser ajustado é o capuz de couro na parte de trás do personagem, que não se movimenta de maneira fluida e carece de detalhes. Planejamos corrigir essas questões em iterações futuras para aprimorar a animação e o design do personagem.

4.3.2 O MAPA

Com o Guerreiro em mãos foi possível entregá-lo à equipe de programação, em seguida foi feito um mapa simples para o teste do personagem e posteriormente para a IA inimiga.

VERSÃO 1.0 – MAPA DE TESTES

Este mapa tem como importância para o encaminhamento da IA a se locomover, e um mapa para testar as físicas e a colisão das ações do Guerreiro.

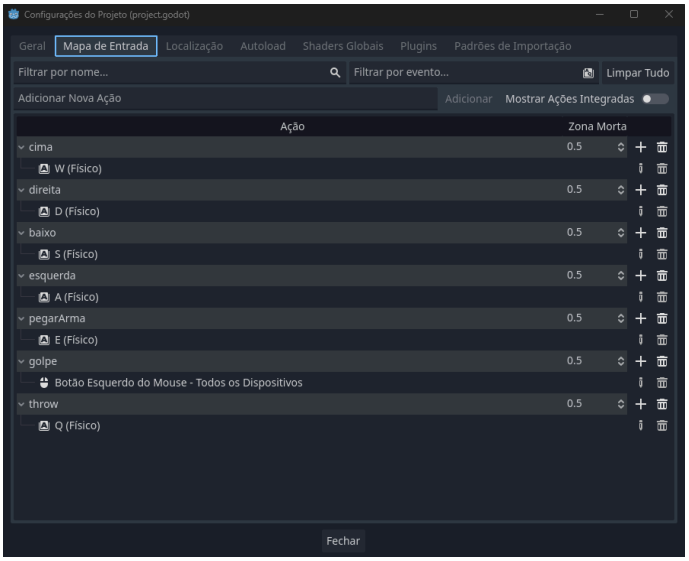
Após a sua criação, utilizando os assets do artista Anokolisa, percebemos que a perspectiva do mapa não estava alinhada com a do personagem. Enquanto o personagem é visto de cima, cem por cento de cima, o mapa possui detalhes angulados. Ao posicionar o personagem no mapa, a discrepância se tornou evidente, o que nos levará a ajustar a perspectiva para garantir uma melhor integração entre os elementos.

Diante dessa discrepância, decidimos descartar os assets pré-feitos. A equipe de design agora está responsável por criar todos os assets que utilizaremos na construção dos mapas. Essa abordagem nos permitirá garantir uma coesão estética e uma melhor integração entre o personagem e o ambiente.

VERSÃO 2.0 – MAPA DE TESTES

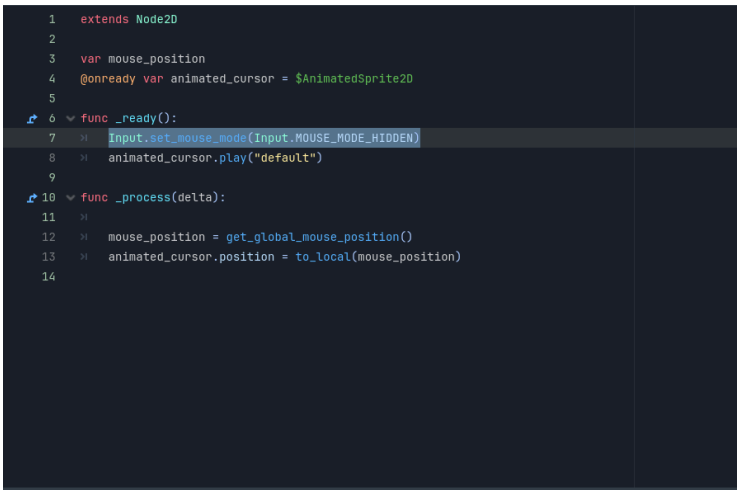
4.4 A PROGRAMAÇÃO

Com os assets e animações do guerreiro concluídas foi possível começar a aplicar a lógica das animações no jogo, a primeira parte da programação do jogo foi fazer o guerreiro se movimentar de uma forma que seja confortável para os jogadores, assim foi optado por usar a programação clássica de W,S,A,D no jogo, com W movimentando para cima, S para baixo, D para direita e A para a esquerda.



Após a conclusão da movimentação básica para o guerreiro foi feito o primeiro aprimoramento que é o guerreiro sempre olhar na direção do ponteiro ou seja nós podemos andar com o boneco utilizando as teclas e controlar a direção para onde o mesmo está olhando através do mouse, esse aprimoramento foi feito utilizando uma função pronta da godot chamada look_at, ela faz um objeto se direcionar para o parâmetro que é colocado nela e para fazer o parâmetro ser o mouse utilizamos a função get_global_mouse_position.

O próximo desafio foi criar um cursor personalizado e animado para o jogo e para isso foi criada a cena cursor que é a responsável por pegar as coordenadas do mouse em tempo real e passar para o cursor animado, a cena primeiramente esconde o cursor utilizando a função Input.set_mouse_mode(Input.MOUSE_MODE_HIDDEN) essa função com esse parâmetro é o responsável pelo cursor padrão do SO sumir. logo após isso o código inicia a animação do ponteiro personalizado e fica sempre checando a localização do mouse e atualizando a posição do cursor personalizado.



```
1   extends Node2D
2
3   var mouse_position
4   @onready var animated_cursor = $AnimatedSprite2D
5
6   func _ready():
7       Input.set_mouse_mode(Input.MOUSE_MODE_HIDDEN)
8       animated_cursor.play("default")
9
10  func _process(delta):
11
12      mouse_position = get_global_mouse_position()
13      animated_cursor.position = to_local(mouse_position)
14
```

Após a conclusão de toda a movimentação do guerreiro, foi feito o próximo passo que foi o sistema de coleta de armas, para realizar isso de forma organizada foi criado um nó chamado game_mananger, esse nó consiste em realizar uma comunicação do nó das armas que estão no cenário com o jogador. Essa comunicação é feita através do botão de coletar armas, o game_mananger se comunica com as armas verificando se o jogador está na área de alguma arma e se estiver ele poderá coletar a arma.

```
58      if Input.is_action_just_pressed("pegarArma"):
59          if(GameMananger.collectWeapon()):
60              var nomeArma        = GameMananger.verNome()
61              animationMovment    = dictArma[nomeArma][0]
62              animationStop       = dictArma[nomeArma][1]
63
```

```
1   extends Node2D
2    # Sinal para coletar o machado
3   signal coletarArma
4   signal danoInimigo
5
6   var player_is_armed: bool = false
7
8   var nameArma: String
9   var idArma
10  var armaColetada = "semArma"
11  # Called when the node enters the scene tree for the first time.
12  var shapeWeapon: bool = false
13  func _ready():
14      pass
15  func desarma():
16      armaColetada = "semArma"
17
18  func collectWeapon():
19      if (!shapeWeapon): return false
20      player_is_armed = true
21      armaColetada = nameArma
22      emit_signal("coletarArma" , idArma)
23      return true
24
25  func verNome():
26      return armaColetada
27
28  func areaArma(id , nome):
29      nameArma = nome
30      idArma = id
31      shapeWeapon = true
32      print(nameArma)
33
34  func foraArma():
35      shapeWeapon = false
36      nameArma = "semArma"
37
```

```gdscript
class_name Arma
extends Area2D

var id : int
var nome: String
var arma = false

func _init(id : int , nome: String ):
	self.id = id
	self.nome = nome

# Called when the node enters the scene tree for the first time.
func _ready():
	GameMananger.coletarArma.connect(coletaArma)


# Called every frame. 'delta' is the elapsed time since the previous frame.
func _process(delta):
	pass


func _on_body_entered(body):
	if(body.is_in_group("jogador")):
		GameMananger.areaArma(id, self.nome)
		arma = true

		body.z_index = 1

func coletaArma(armaId):
	if(armaId == id and arma):
		queue_free()


func _on_body_exited(body):
	if(body.is_in_group("jogador")):
		GameMananger.foraArma()
		arma = false
```

Com as mecânicas de coleta de armas concluídas foi feito as mecânicas de golpes, elas consistem em conferir a arma que o jogador está e com base nisso executar as animações corretas, ou seja se o jogador está desarmado ele tem que fazer as animações do personagem desarmado e se ele está armado o personagem tem que executar as animações daquela determinada arma.

```gdscript
var dictArma = {
	"machado": ["andandoMach","paradoMach",  "golpeMach"],
	"martelo": ["andandoMat", "paradoMat",   "golpeMat"],
	"espada":  ["andandoEsp", "paradoEsp",   "golpeEsp"],
	"semArma": ["andando",    "parado",      "golpe"]
}
```

Como podemos ver na imagem acima o dicionário é o responsável por armazenar todas as animações e com base nele é feito os golpes em diferentes armas.

```
func attack():
    if Input.is_action_just_pressed("golpe"):
        var nome = GameMananger.verNome()



        if(!atack):
            atack = true
            animatedSprite.play(dictArma[nome][2])

            $deal_attack.start()

```

A função attack vista na imagem acima confere a arma que o jogador está equipado no momento e com base nisso ela acessa a chave equivalente a dessa arma no dicionário e a partir disso a função executa a animação correta.

4.4.1 COMO A HABILIDADE DE ARREMESSAR ARMAS FOI IMPLEMENTADA

```
extends Node2D

var weapon_name = GameMananger.armaColetada
var weapon_scene: PackedScene = load("res://" + weapon_name + ".tscn")

var speed:    float = 600.0
var spin_speed: float = randi_range(30, 50)
var direction:  Vector2

var timer: Timer
```

Primeiro, um *script* chamado *player_projectile* foi criado para definir o que a arma que foi arremessada pelo jogador faz ao ser criada.

No bloco de código acima, a arma equipada atualmente pelo jogador é copiada do script *GameManager* através da variável *armaColetada*. Em seguida, a *scene* da arma é criada dinamicamente usando uma *string* dinâmica. Ou seja, se o jogador tiver um machado, o *sprite* do projétil será o do machado. Isso segue para as outras armas.

Em seguida, a velocidade do projétil, velocidade de giro, direção e um cronômetro, ou *timer*, são definidos. O cronômetro é usado para chamar a função *projectile_stopped()*, que define o que acontece quando o projétil é destruído.

```
func _ready():
    var sprite_name: String

    if    weapon_name == "machado":
        sprite_name = "Axe"
    elif    weapon_name == "espada":
        sprite_name = "Sword"
    else:
        sprite_name = "Hammer"

    $Sprite2D.texture = load("res://assets/weapons/spr" + sprite_name + ".png")
    timer = Timer.new()
    add_child(timer)

    timer.wait_time = 0.4 # distância
    timer.one_shot  = true

    timer.connect("timeout", Callable(self, "projectile_stopped"))
    timer.start()

    set_process(true)

func _process(delta: float) -> void:
    position += direction  * speed * delta
    rotation += spin_speed * delta
```

Na função *_ready()*, o nome do sprite do projétil é definido e determinado usando instruções *if* e *else*, e seu *sprite* correspondente, ou *texture*, é carregado usando uma *string* dinâmica. Em seguida, o timer é criado e configurado para ativar após 400 milissegundos, onde a função *projectile_stopped()* é chamada. A função *_process()* atualiza a direção e a velocidade do projétil em tempo real.

```
func projectile_stopped() -> void:
    var dropped_weapon     = weapon_scene.instantiate() as Node2D
    dropped_weapon.position = position
    dropped_weapon.rotation = rotation
    get_tree().current_scene.add_child(dropped_weapon)

    queue_free()

func _on_body_entered(body: Node2D) -> void:
    projectile_stopped()
```

Finalmente, na função *projectile_stopped()*, uma variável chamada *dropped_weapon* é criada. Como o nome sugere, essa é a arma que será criada na mesma posição e ângulo em que o projétil for destruído. Isso é feito para que o jogador possa jogar sua arma e equipá-la novamente quando ela cair no chão. Em seguida, o projétil é destruído.

_on_body_entered() é usado para chamar a função *projectile_stopped()* em uma colisão.

```
if Input.is_action_just_pressed("throw"): # Lançar arma com Q
    if !GameMananger.player_is_armed: return

    animationMovment    = "andando"
    animationStop       = "parado"
    animatedSprite.play("golpe")
    GameMananger.player_is_armed = false

    var player_projectile = projectile_scene.instantiate() as Node2D
    get_tree().current_scene.add_child(player_projectile)
    player_projectile.position  = position
    player_projectile.direction = Vector2.RIGHT.rotated(rotation).normalized()
    print('lançar arma')
```

No script *player.gd*, dentro da função *_physics_process(),* uma ação "throw" aguarda *input* do jogador, que foi definida como *Q* nas configurações do projeto. Uma vez acionada, ela primeiro verifica se o jogador está armado usando o booleano *player_is_armed* do *GameManager*. Se for falso, isso nos diz que o jogador não tem arma para arremessar, e então a função é retornada imediatamente.

Agora que sabemos que o jogador tem uma arma, primeiro definimos suas animações de *movimento*, *parado* e *golpe* para seus estados desarmados e o booleano *player_is_armed* se torna falso. Em seguida, o projétil da arma é criado na posição do jogador e herda a sua direção.