Programação para jogos 1

apresentação do jogo

Plataforma

Godot 3.0

O Godot é um motor de jogo de código aberto publicado no âmbito da licença MIT desenvolvido por sua própria comunidade.

Process VS Physics_process

Conceitos

- Game Loop
- Observer
- Update
- Componentes
- Máquina de estados
- Grupos

Entidades

Player

Máquina de estados

- O jogador funciona a base de uma máquina de estados que roda dentro do physics_process, ela chama métodos baseado no estado atual dele que é influenciado por inputs que estão no process e por colisões feitas com outras entidades.

```
30 func _process(delta):
31     _get_input()
32     _set_animation()
```

```
62
63 func _physics_process(delta):
64     match current_state:
65         WALK:
66             _walk_state(delta)
67         KICK_UP:
68             _kick_up_state(delta)
69         IDLE:
70             _idle_state()
71         HIT:
72             _hit_state()
```

Entidades

Player

Sinais

- A mecânica principal do jogador é feita com um auxílio de um “signal” que é um peculiaridade do godot, e corresponde ao padrão Observer.

- E esse sinal por sua vez é conectado com o bola. E ao receber essa notificação, ela trata a colisão de forma diferente ao que trataria sem o sinal.

```
func _kick_up_state(delta):
    if enter_state:
        enter_state = false

    emit_signal("up_kick_signal")
```

```
func _on_Player_up_kick_signal():
    upKick = true
```

```
if collision.collider.name == "Player" and upKick:
    combo += 1 #modificador de combo
    emit_signal("play_sound", "kick") #som de chute

    #Scrip que faz a bola não dar dano e so voltar
    mov.y = - 1
    mov.x = -mov.x
    if mov.y > 0:
        mov.y = -mov.y
```

Entidades

Player

Sinais

- Além da mecânica principal, ainda se faz uso de um outro sinal com o auxílio da função de grupos do godot para detectar a colisão com os projéteis que saem do inimigo.

- Vale lembrar que exceto os sinais que são criados automaticamente como métodos por fora do loop, todo o cálculo de colisão e movimentação é feito dentro do loop fixo, por isso que normalmente os sinais no player sempre acabam definindo um estado..

```
func _on_Laser_body_entered(body):
    if body.name == "Player":
        get_tree().call_group("player", "set_hit")
        queue_free()
```

```
func set_hit():
    set_state(HIT)
```

Entidades

Bola

Como a bola é uma entidade que não se controla, tudo é feito apenas no Physics_process.

Lá é onde a maior parte das colisões do jogo são testadas, e a partir da objeto que a bola colide ela tomar uma ação, algumas dessas que também são acompanhadas de sinais ou o uso de grupos para auxiliar na comunicação com as outras entidades.

Como por exemplo, a notificação do contador de combo, que é feita usando um sinal para outra entidade onde é calculada a pontuação.

```
get_tree().call_group("HUD", "get_combo", combo)
```

Entidades

Inimigos

O inimigo do jogo é feito de maneira simples, já que faz uso de um sinal do próprio godot que detecta colisões.

Ao detectar a colisão e se testar que é a bola, se executar seu script de disparo e explosão.

A exploração é feito usando o sistemas de partículas do godot.

```
func _on_Inimigo_body_entered(body):
    #Verifica colisão com a bola
    if body.name == "Ball":
```

Observer

Observer de áudio

Apesar dos sinais já serem um implementação do padrão observer e ajudarem muito na notificação.

Foi feito um observer para servir de controle para os sons no geral.

E quando um som é necessário é emitido um sinal com o som que deve ser tocado.

Há ainda, alguns sons que não estão no observer, pois tocam raríssimas vezes e somente 1 vez.

```
extends Node2D

func _on_Menu_play_sound(som):
    match som:
        'menu':
            $Menu/theme.play()
        'accept':
            $Menu/accept.play()
        'move':
            $Menu/move.play()


func _on_Ball_play_sound(som):
    match som:
        "bounce":
            $Bola/quique.play()
        "kick":
            $Bola/chute.play()

func _on_World_play_sound():
    $Stages/Stage1.play()

func _on_Player_play_sound(som):
    match som:
        "hit":
            $Bola/hit.play()
```

Telas

Menu e HUD

Os menus em si basicamente são feito baseado em um estado que muda baseado nos inputs que o usuário vai colocando. Sendo assim, como eles apenas inputs e draws, ficam 100% no process.

```
4 func inimigo_destruido(score):
5     score_final = score_final + (score * combo)
6     $Score.text = str(score_final)
7
```

Para a HUD, não é diferente.

Mas é nela onde se é feito o cálculo de pontos com a ajuda dos sinais, baseado nas notificação do combo vinda da bola e da pontuação ganha ao destruir um inimigo, vinda do inimigo.

Telas extras

No mais as outras tela do jogo, animações, transições são feitas basicamente com visuais e animações provenientes do godot.

E se é usado em casos onde se pede, o uso de Inputs no process.

Obrigado!