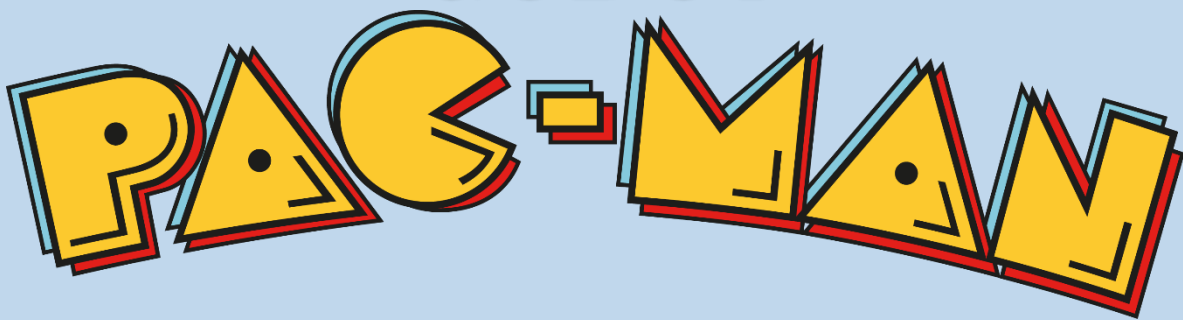


Documentación de PROYECTO JUEGO

GODOT



Introducción a la Inteligencia Artificial

Dr. Luis Felipe Marín Urias

Joel Jácome Pioquinto

Josué Cristofer Téllez Huerta

Leissa Mariana Cervantes Sánchez

Para la realización de este proyecto se tomaron en cuenta las consideraciones de evaluación donde de ellas se escogió realizar un juego en Godot con los algoritmos de búsqueda:

- BPA
- BPP
- Greedy
- A*

Se escogió como proyecto un juego de PACMAN, donde el mapa de juego es un laberinto con obstáculos donde los personajes dentro de él son 4 fantasmas y un Pacman.

El protagonista y el personaje manipulado por el usuario es el Pacman, que es un personaje circular de color amarillo, el cual su propósito es comer todas las pacdots que se encuentran dentro del mapa y no ser alcanzado por los villanos, que, en este caso son los fantasmas.

El propósito de los fantasmas es alcanzar a Pacman y “matarlo” antes de que se coma todos los pacdots.

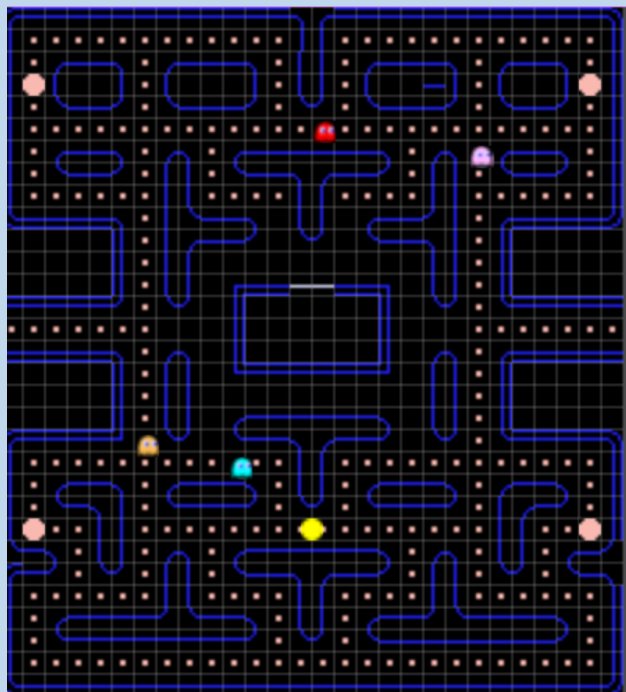
Donde cada fantasma será regido por métodos de búsqueda IA

Fantasma Azul : BPA

Fantasma Naranja: BPP

Fantasma Rojo : Greedy

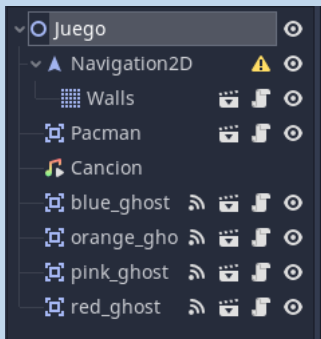
Fantasma Rosa: A*



ESTRUCTURA DEL JUEGO

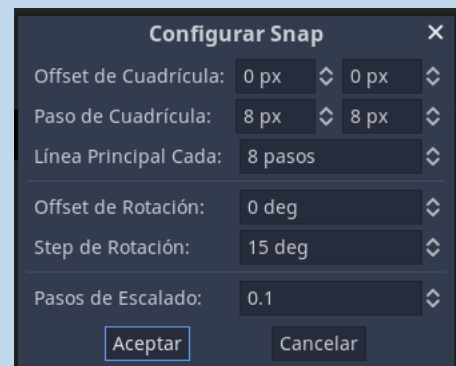
El proyecto está estructurado con un nodo principal llamado juego por donde por medio de “Instanciar escena hija” se le agregarán las escenas creadas posteriormente, para las paredes del juego, la navegación, los fantasmas, el pacman, las escenas que se proyectan al perder o ganar, los tiles, el sonido, etc.

EXPLICACIÓN DE CADA NODO:

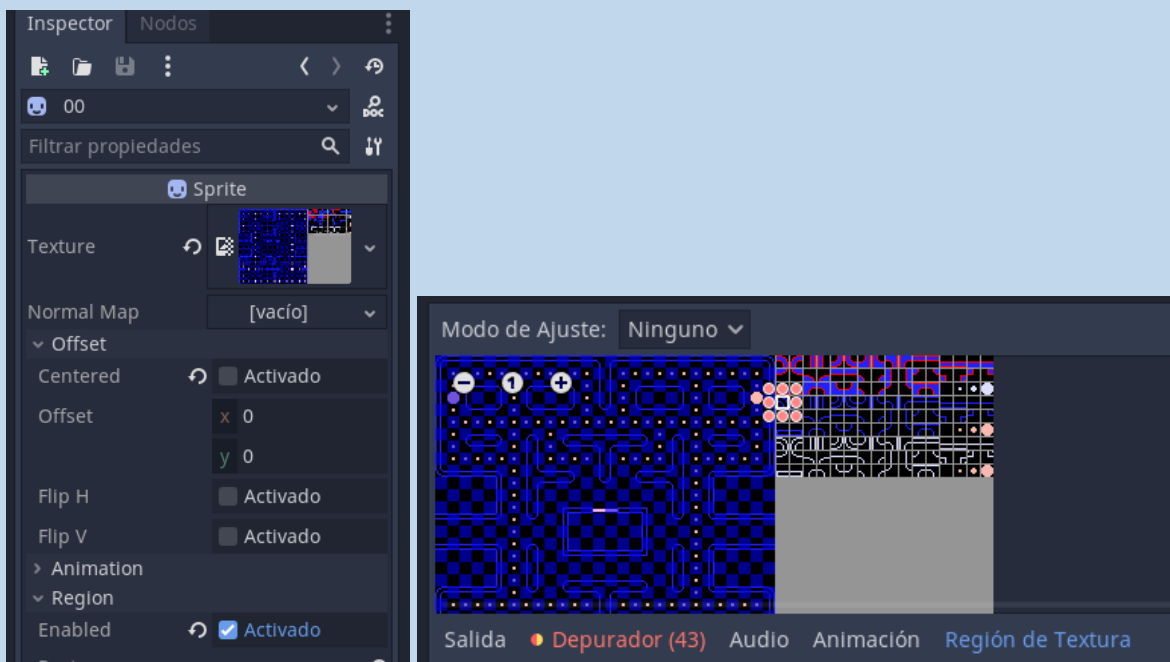


- **Navigation2D:** Este nodo se realizó para que en él se le agregó un nodo hijo llamado “Walls” que son las paredes del juego, para la creación de este walls se creó un documento tiles.tres dentro de la escena “Walls” que nos sirvió para posiciones tile por tile y formar así el laberinto.

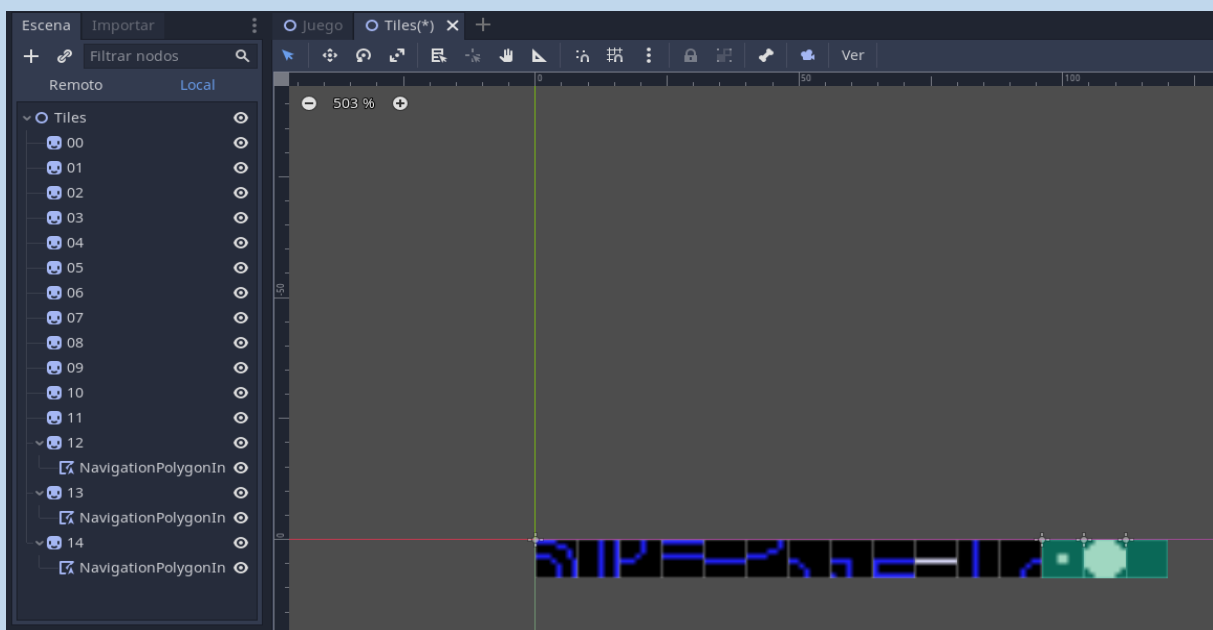
Para hacer estos tiles lo que se hace es que se creó una escena llamada tiles donde se agregaron 14 sprites que serán los ocupados dentro del laberinto, para esto se realizó un pixel snap, dándoles una dimensión de 8 x 8



Donde se le agrega una imagen de referencia la cual será recortada por medio de “Región”

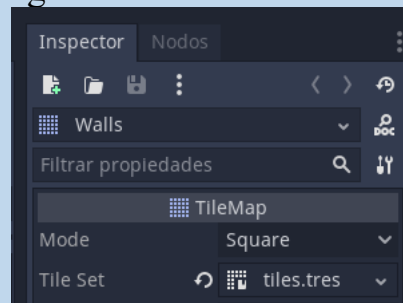


Se selecciona la fracción de imagen deseada para cada sprite y se forma lo siguiente:

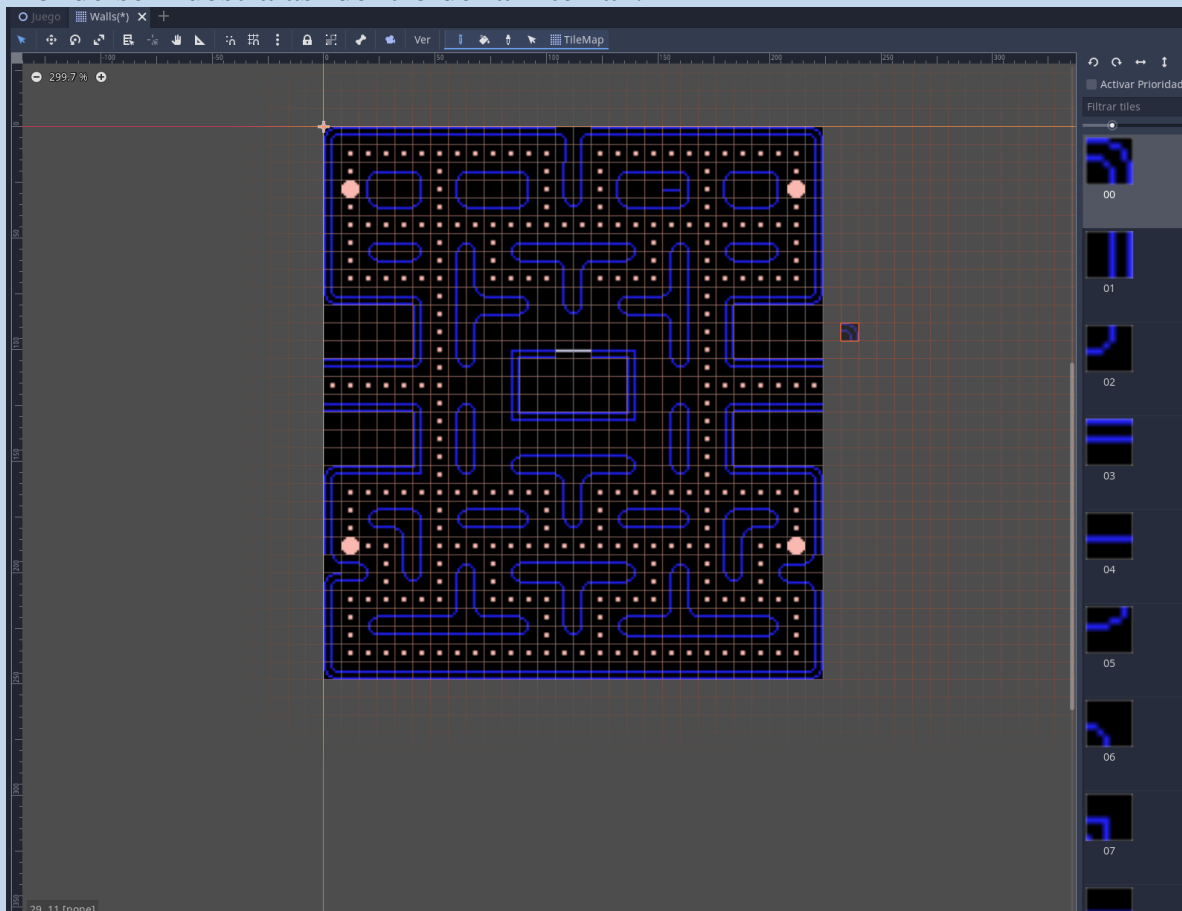


Donde los sprites 12, 13, 14 se les agrega un nodo hijo de Polígono de Navegación que son donde se van a poder trasladar nuestros personajes.

Después de realizar eso, se guardó el documento como tiles.tres, ya que se usó para poder realizar el nodo hijo tipo TILEMAP, llamado Walls, donde se pone una imagen de referencia con baja opacidad para sobre ella irnos guiando para poner los tiles. Para cargar los tiles previamente realizados dentro de la ventana de Inspector se cargó la escena de "tiles.tres":



Donde se muestra así dentro de la interfaz:



Para posteriormente ir armando el mapa tile por tile hasta que quede de la manera deseada.

SCRIP DE WALLS:

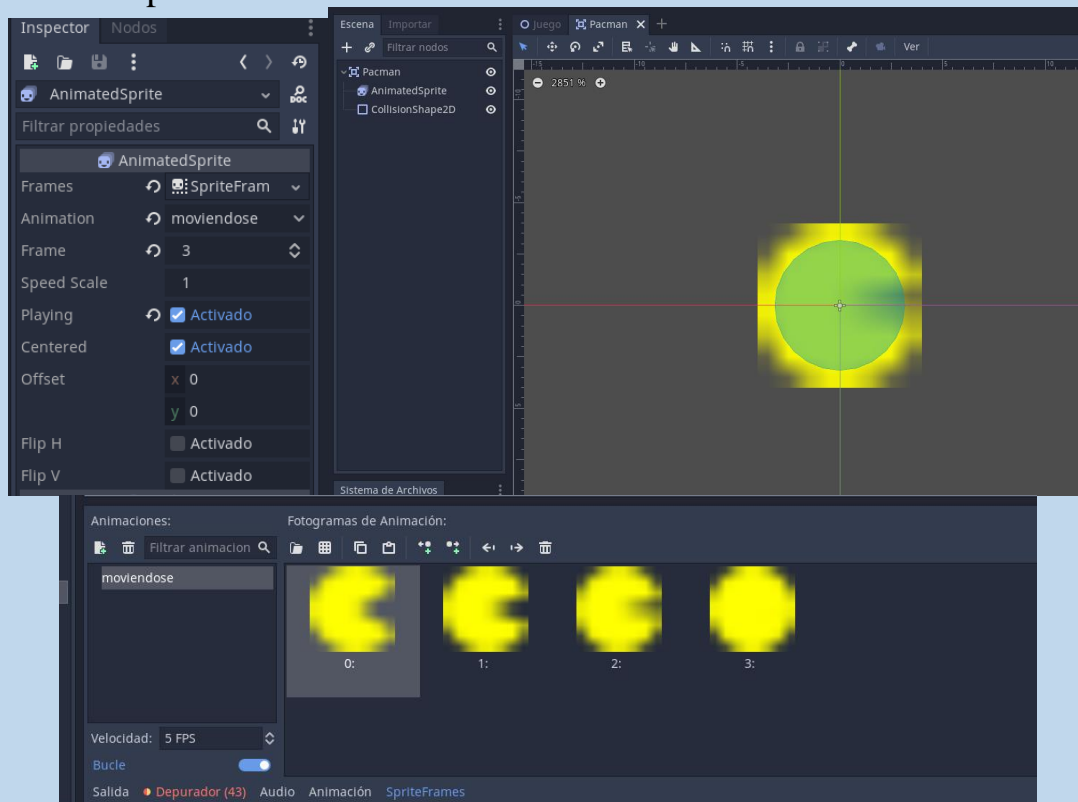
```
1 extends TileMap
2
3 onready var half_cell_size = get_cell_size()/2 #Variable para posición enmedio de la celda
4 onready var player = get_parent().get_parent().get_node("Pacman") #Obteniendo nodos de cada personaje
5 onready var FRosa = get_parent().get_parent().get_node("pink_ghost")
6 onready var FRojo = get_parent().get_parent().get_node("red_ghost")
7 onready var FNaranja = get_parent().get_parent().get_node("orange_ghost")
8 onready var FAzul = get_parent().get_parent().get_node("blue_ghost")
9
10 func get_player_init_pos(): #La posición inicial del jugador
11     var pos = map_to_world(Vector2(14,23)) #Convierte tiles a pixeles
12     pos.y += half_cell_size.y #Posiciona el player en medio de la celda
13     return pos
14
15 func is_tile_vacant(pos, direction): #Evalúa y devuelve los tiles de navegación
16     var curr_tile = world_to_map(pos)
17     var next_tile = get_cellv(curr_tile + direction)
18     var next_tile_pos = Vector2()
19     if (next_tile == 12 or next_tile == 13 or next_tile == 14):
20         next_tile_pos = map_to_world(curr_tile + direction)+half_cell_size
21     else:
22         next_tile_pos = relocate(pos)
23     return next_tile_pos
24
25 func relocate(pos):
26     var tile = world_to_map(pos) #vuelve la posición pixeles a tiles
27     return map_to_world(tile) + half_cell_size #Los tiles regresa a pixeles pero ahora centrados
28
29 func comer(pos): #Funcion del pacman comiendo
30     var curr_tile = world_to_map(pos)
31     var tile = get_cellv(curr_tile)
32     if (tile == 12 or tile == 13): #Si el pacman pasa por los tile 12 o 13, que son las bolitas, estaría "comiendo"
33         set_cellv(curr_tile, 14) #Los tiles pasan a ser tipo tile 14 que es el recuadro negro sin pac dots
34
35 func _process(delta):
36     var contador=0
37     for i in range(get_used_rect().size.x): #Contador de pacdots, si no hay bolitas dentro del mapa, ganas
38         for j in range(get_used_rect().size.y):
39             var tile = get_cell(i,j)
40             if (tile == 12):
41                 contador += 1
42     if (contador == 0):
43         print("*****FELICIDADES HAS GANADO*****")
44         set_process(false)
45         get_tree().change_scene("res://YouWin.tscn") #llama script para proyectar escena ganadora
46
47 func get_fantasma_pos(): #Posicion inicial de los fantasmas
48     var pos = map_to_world(Vector2(14,11))
49     pos.y += half_cell_size.y #Posiciona el fantasma enmedio de la celda
50     return pos #Regresa la posición
51
```

Descripción de funciones:

- `get_player init_pos()`: se creó para darle una posición inicial al pacman.
- `is_tile vacant()`: función para evaluar y devolver los tiles de navegación.
- `relocate()`: es para posicionar el personaje en medio de las celdas.
- `comer()`: es una función creada para que cuando el pacman pase por los tiles que contienen pacdots cambien de tipo de tile y se vuelvan un tile vacío (sin pacdots) lo cual da la impresión que se los comió.
- `_process()`: es el que indica si ganaste, lo que hace es que hace la cuenta de cuántos tiles con pacdots se encuentran dentro del juego, si no hay ninguno, te arroja una línea que dice “FELICIDADES HAS GANADO” y llama a una escena creada aparte que proyecta una imagen ganadora.
- `get_fantasma_pos()`: Es para crear la posición inicial de los fantasmas, al igual que los posiciones en medio de la celda.

- **NODO ESCENA PACMAN**

Se creó una nueva escena con un nodo tipo Area 2D el cual se le agregó un nodo hijo tipo AnimatedSprite donde se cargaron las imágenes secuenciales para hacer la ilusión de movimiento del pacman donde su animación se guardó como “moviéndose”, también al nodo padre se le agregó un nodo tipo collision shape con forma circular.



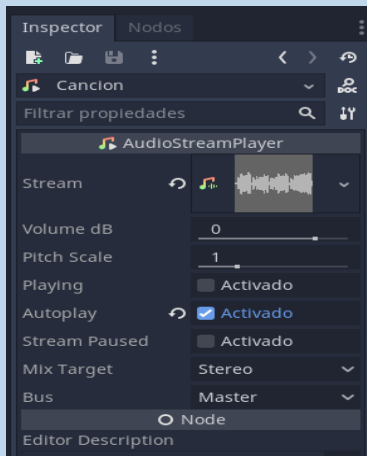
SCRIPT DE PACMAN:

```
1 extends Area2D
2
3 var direction = Vector2(0,0)
4 var speed= 15
5 onready var walls = get_parent().get_node("Navigation2D/Walls") #Obtiene nodo walls
6
7 func _ready():
8     $AnimatedSprite.play("moviendose") #Reproduce la animación llamada "moviendose"
9     position = walls.get_player_init_pos() #Ubica al pacman en su posición inicial
10
11 func _process(delta):
12     #Función para controlar le pacman por medio de
13     # las teclas de dirección
14     if Input.is_action_pressed("ui_up"):
15         direction = Vector2(0,-1)
16         rotation = deg2rad(-90) #Rota el sprite en dirección al movimiento
17     elif Input.is_action_pressed("ui_down"):
18         direction = Vector2(0,1)
19         rotation = deg2rad(90) #Rota el sprite en dirección al movimiento
20     elif Input.is_action_pressed("ui_left"):
21         direction = Vector2(-1,0)
22         rotation = deg2rad(180) #Rota el sprite en dirección al movimiento
23     elif Input.is_action_pressed("ui_right"):
24         direction = Vector2(1,0)
25         rotation = deg2rad(0) #Rota el sprite en dirección al movimiento
26
27     var pos_to_move = walls.is_tile_vacant(position, direction) #Llama a función dentro de walls para ubicar el pacman dentro del m
28     if (direction != Vector2(0,0)): #Si la dirección no está en el vector (0,0)
29         position = position.linear_interpolate(pos_to_move, speed * delta) #Mueve de un punto a otro el pacman respecto a su veloci
30         walls.comeer(position) #Mientras avanza va comiendo
```

Descripción de funciones:

- `_ready()`: Se crea para que se reproduzca la animación del Sprite animado y conecta a la escena walls donde obtiene su posición inicial que se encuentra del código walls.
- `_process()`: La función es para controlar el movimiento del pacman así como su dirección y orientación, también llama a “walls” para ubicarlo dentro del mapa y ubicar la posición en donde se encuentre durante su movimiento.

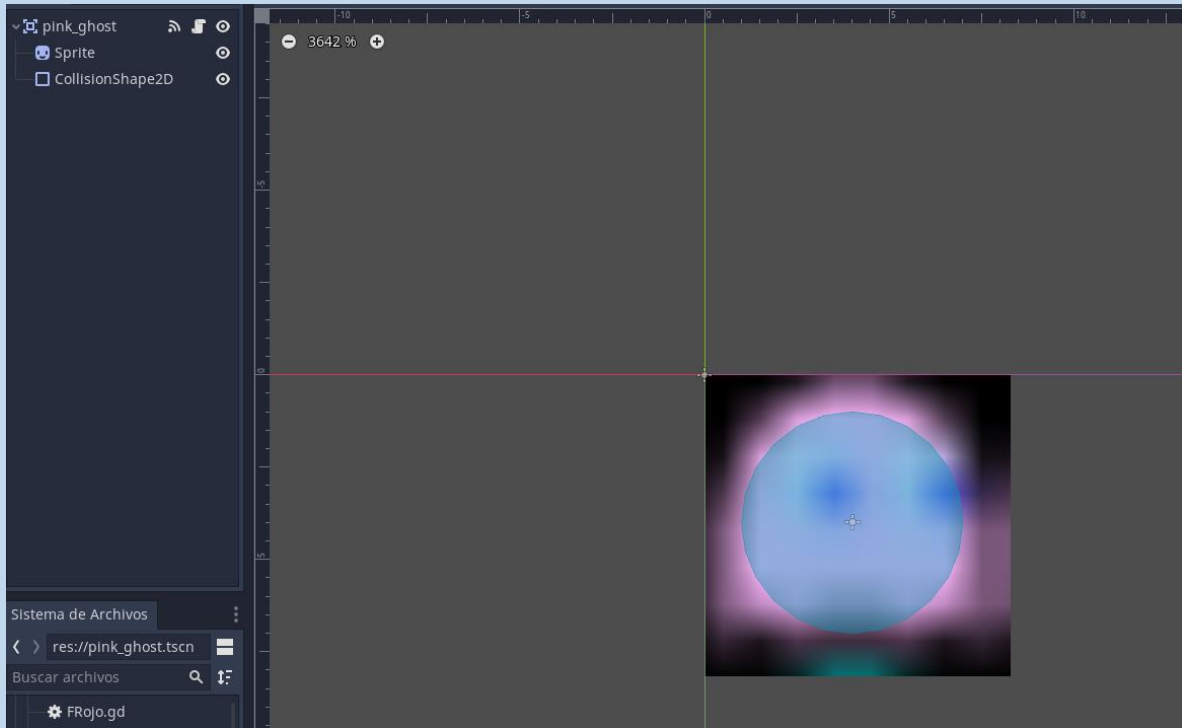
• NODO CANCIÓN



Es un nodo tipo AudioStreamPlayer donde se le carga un documento de audio tipo .WAV y para que se reproduzca al momento de inicial el juego se activa la opción de autoplay que se encuentra dentro de la ventana “Inspector”

- **NODOS FANTASMA**

Se creó una nueva escena donde se creó un nodo padre tipo Area2D y se agregó dos nodos hijos, un tipo sprite y otro tipo collision shape de forma circular.



Donde se realizó igual que con los tiles, un recorte de imagen por medio de región de textura.

- **NODO FANTASMA AZUL (blue_ghost) BPA**

Para los fantasmas, la siguiente estructura es similar en todos, solo cambia el ordenamiento y recorrido de los nodos a visitar (en adelante franja).

```
1  #BUSQUEDA PRIMERO EN ANCHURA (BPA - BFS)
2  extends Area2D
3
4  onready var walls = get_parent().get_node("Navigation2D/Walls")
5  var pacman #goal
6
7  onready var bfs_visits : Array = []
8  onready var bfs_fringe : Array = []
9
10 var path
11 var direction = Vector2(0,0)
12 var speed = 30
13
14 var bfs_root
15 var tile_pos
16
17 func _ready():
18     »
19     » position = walls.get_fantasma_pos()
20     »
21     » pacman = get_parent().get_node("Pacman")
22     »
23     » tile_pos = walls.world_to_map(pacman.global_position)
24     » print("PAC-MAN: ", tile_pos.x, " ", tile_pos.y)
25     » var goal = Vector2(int(round(tile_pos.x)), int(round(tile_pos.y)))
26     »
27     » tile_pos = walls.world_to_map(global_position)
28     » print("BLUE GHOST: ", tile_pos.x, " ", tile_pos.y)
29     » bfs_root = BFSNode.new(tile_pos.x, tile_pos.y)
30     » path = bfs_root.search(goal, bfs_visits, bfs_fringe, walls)
31     » »
32     » #path = walls.get_path_to_player("blue_ghost")
```

Se agregó la variable de las paredes y una variable llamada pacman que en este caso es nuestra meta, también se encuentran las listas de las visitas y la franja, donde se van a almacenar los nodos a visitar. Variables de path (camino) la de dirección, velocidad, la raíz del método de búsqueda y el tile_pos.

La función ready se encuentra dentro de todos los fantasmas que es la que obtiene la posición del fantasma en el walls, y obtiene el nodo del pacman, es para la búsqueda.

```

31  >>>
32  >> #path = walls.get_path_to_player("blue_ghost")
33  >>
34  < func _process(delta):
35  < >> if (path.size() > 1):
36  < >>     var pos_to_move = path[0]
37  < >>     direction = (pos_to_move - position).normalized()
38  < >>     var distance = position.distance_to(path[0])
39  < >>     if (distance > 1):
40  < >>         position += speed * delta * direction
41  < >>     else:
42  < >>         path.remove(0)
43  < >>     else:
44  < >>         position = walls.get_fantasma_pos()
45  < >>
46  < >>     pacman = get_parent().get_node("Pacman")
47  < >>
48  < >>     tile_pos = walls.world_to_map(pacman.global_position)
49  < >>     print("PAC-MAN: ", tile_pos.x, " ", tile_pos.y)
50  < >>     var goal = Vector2(int(round(tile_pos.x)), int(round(tile_pos.y)))
51  < >>
52  < >>     tile_pos = walls.world_to_map(global_position)
53  < >>     print("BLUE GHOST: ", tile_pos.x, " ", tile_pos.y)
54  < >>     var bfs_root = BFSNode.new(tile_pos.x, tile_pos.y)
55  < >>     bfs_visits.clear()
56  < >>     bfs_fringe.clear()
57  < >>     path = bfs_root.search(goal, bfs_visits, bfs_fringe, walls)
58  < >>
59  < >>     #path = walls.get_path_to_player("blue_ghost")
60
61

```

El `_process` es para que mientras la lista no esté vacía va a ir recorriendo, si el camino se encuentra vacío va a volver a hacer la búsqueda.

La función `_on_area_entered()`: es para que cuando el fantasma entre dentro del área del pacman se termine el juego, ya que, como antes mencionado si esto ocurre se pierde el juego, al igual que llama una escena creada llamada "GAME OVER" que se explicará posteriormente.

```

61
62  < func _on_blue_ghost_area_entered(area):
63  < >> if (area.name == "Pacman"):
64  < >>     print("GAME OVER")
65  < >>     get_tree().change_scene("res://GameOver.tscn")
66

```

La clase del nodo contiene los atributos necesarios para poder evaluarse en la búsqueda y su respectivo método constructor, también un método de impresión del nodo como se puede observar.

```
67 class BFSNode:
68     var posx
69     var posy
70     var branches
71     var parent
72
73     func _init(posx, posy, parent = null):
74         self.posx = posx
75         self.posy = posy
76         self.branches = []
77         self.parent = parent
78
79     func print_position():
80         print("x: ", self.posx, ", y: ", self.posy)
81
```

La expansión del nodo consiste en evaluar la posición del nodo a donde nos quereos mover (arriba, abajo, izquierda o derecha) y si no es pared, se crea el nodo de movimiento y se agrega a los hijos del nodo actual, finalmente se agrega al final de la lista de franja.

```
79     func print_position():
80         print("x: ", self.posx, ", y: ", self.posy)
81
82     func expand(goal, fringe, tile_map):
83         var move
84         var cell
85
86         cell = tile_map.get_cell(self.posx, self.posy - 1)
87         if cell == 12 or cell == 13 or cell == 14:
88             move = BFSNode.new(self.posx, self.posy - 1, self)
89             self.branches.append(move)
90
91         cell = tile_map.get_cell(self.posx + 1, self.posy)
92         if cell == 12 or cell == 13 or cell == 14:
93             move = BFSNode.new(self.posx + 1, self.posy, self)
94             self.branches.append(move)
95
96         cell = tile_map.get_cell(self.posx, self.posy + 1)
97         if cell == 12 or cell == 13 or cell == 14:
98             move = BFSNode.new(self.posx, self.posy + 1, self)
99             self.branches.append(move)
100
101         cell = tile_map.get_cell(self.posx - 1, self.posy)
102         if cell == 12 or cell == 13 or cell == 14:
103             move = BFSNode.new(self.posx - 1, self.posy, self)
104             self.branches.append(move)
105
106         for branch in self.branches:      #no ordena los hijos, solo los mete a la franja
107             fringe.append(branch)
108
109
110     func search(goal, visits, fringe, tile_map):
```

En la busqueda, que es recursiva, se evalúa primero si el nodo es la meta, si lo es, se empieza a agregar al camino los nodos padres hasta llegar al estado inicial, y este regresarlo, pues será en esas coordenadas, donde el fantasma se moverá para buscar al Pac-Man.

Si no es meta, evalúa si ya fue visitado, si no lo ha sido, lo agrega a la lista de visitados y expande a sus hijos (función explicada previamente).

Finalmente, y siempre y cuando la franja tenga nodos, entrará a la recursión de la función de búsqueda con el elemento del frente que le saque a la lista.

```
109
110 ▾ » » func search(goal, visits, fringe, tile_map):»
111 » » » var way = []»
112 » » » var world_pos
113 #» » » print("* Entra a la cola *")
114 #» » » self.print_position()
115 ▾ » » » if self.posx == goal.x and self.posy == goal.y:
116 » » » » print("* Encuentra la meta *")
117 » » » »
118 » » » »
119 #» » » » way.append(Vector2(self.posx, self.posy))
120 » » » » world_pos = tile_map.map_to_world(Vector2(self.posx, self.posy))
121 » » » » world_pos.x = world_pos.x + 2
122 » » » » world_pos.y = world_pos.y + 2
123 » » » » way.append(world_pos)
124 » » » »
125 » » » » var parent = self.parent
126 ▾ » » » » while parent:
127 » » » » » world_pos = tile_map.map_to_world(Vector2(parent.posx, parent.posy))
128 » » » » » world_pos.x = world_pos.x + 2
129 » » » » » world_pos.y = world_pos.y + 2
130 » » » » » way.append(world_pos)
131 » » » » » parent = parent.parent
132 » » » » » way.invert()
133 » » » » » return PoolVector2Array(way)
134 » » » » #return way
135 » » »
136 » » » var is_visited = false
137 ▾ » » » if not visits == []:
138 ▾ » » » » for visited in visits:
139 ▾ » » » » » if self.posx == visited.posx and self.posy == visited.posy:
140 » » » » » » is_visited = true
```

```
135 » » »
136 » » » var is_visited = false
137 ▾ » » » if not visits == []:
138 ▾ » » » » for visited in visits:
139 ▾ » » » » » if self.posx == visited.posx and self.posy == visited.posy:
140 » » » » » » is_visited = true
141 » » » » » » break
142
143 ▾ » » » if not is_visited:
144 » » » » visits.append(self)
145 » » » » self.expand(goal, fringe, tile_map)
146
147 ▾ » » » if not fringe == []:
148 » » » » return fringe.pop_front().search(goal, visits, fringe, tile_map)
149 » » » print("No hay solucion AZUL")
150 » » » return []
151
```

- **NODO FANTASMA NARANJA (orange_ghost) BPP**

El código de este algoritmo es el mismo que el BPA solo que en esta ocasión los hijos se agregan al principio de la franja.

```
100  >| >|
101  >| >|   for branch in self.branches:
102  >| >| >|   var pos = 0
103  >| >| >|   fringe.insert(pos, branch)      #Hace inserts a una posición en específico
104  >| >| >|   pos += 1
105  >| >|
106
```

- **NODO FANTASMA ROJO (red_ghost) BÚSQUEDA GREEDY**

Para la búsqueda greedy, se agrega un elemento especial, la heurística, la cual consiste en obtener la distancia de Manhattan absoluta que hay entre nodo en evaluación y el nodo meta, y en la franja son ordenados por este atributo.

Dicha heurística se obtiene despues de crear el nodo de movimiento (move).

```
62
63 class GreedyNode:
64     var posx
65     var posy
66     var h # es el valor de la heurística del nodo
67     var branches #hijos
68     var parent #el jefe
69
70     func _init(posx, posy, parent = null): #Metodo constructor, si no tiene padre es NULO
71         self.posx = posx #Si no tiene padre quiere decir que llegó al final del cami
72         self.posy = posy
73         self.h = 0
74         self.branches = []
75         self.parent = parent
76
77     func print_position(): #Imprime la posición del fantasma
78         print("x: ", self.posx, ", y: ", self.posy)
79
80     func heuristic(goal):
81         self.h = abs(goal.x - self.posx) + abs(goal.y - self.posy) #Se agrega el valor de la
82
83     func expand(goal, fringe, tile_map): #Expande, crea nuevos hijos
84         var move #siguiente movimiento
85         var cell #donde se almacena el valor de la celda
86
87         #movimiento hacia arriba
88         cell = tile_map.get_cell(self.posx, self.posy - 1)
89         if cell == 12 or cell == 13 or cell == 14:
90             move = GreedyNode.new(self.posx, self.posy - 1, self)
91             move.heuristic(goal)
92             self.branches.append(move)
93
94         #movimiento hacia abajo
95         cell = tile_map.get_cell(self.posx, self.posy + 1)
96         if cell == 12 or cell == 13 or cell == 14:
97             move = GreedyNode.new(self.posx, self.posy + 1, self)
98             move.heuristic(goal)
99             self.branches.append(move)
100
101         #movimiento hacia la izquierda
102         cell = tile_map.get_cell(self.posx - 1, self.posy)
103         if cell == 12 or cell == 13 or cell == 14:
104             move = GreedyNode.new(self.posx - 1, self.posy, self)
105             move.heuristic(goal)
106             self.branches.append(move)
107
108         #Agrega hijos a la franja odenados por heurística
109         for branch in self.branches:
110             if fringe == []:
111                 fringe.append(branch) #si está vacía agrega hijo
112             else:
113                 var pos = 0
114                 for node in fringe:
115                     if node.h > branch.h: #ordena por heurística de menor a mayor
116                         fringe.insert(pos, branch)
117                         break
118                 pos += 1
119         pos = 1
120
121         self.branches.append(move)
```

- **NODO FANTASMA ROSA (pink_ghost) BÚSQUEDA A***

En este caso

```
56 class GreedyNode:
57     var posx
58     var posy
59     var h # es el valor de la heurística del nodo
60     var branches
61     var parent
62     var peso
63     var fn
64
65     func _init(posx, posy, parent = null):
66         self.posx = posx
67         self.posy = posy
68         self.h = 0
69         self.fn
70         self.branches = []
71         self.parent = parent
72
```

Se agrega la clase greedynode que es la que realiza el fantasma rojo, para que el A* funcione se debe primero hacer el greedy y utilizar su heurística, la diferencia es que ahora se le debe agregar peso como se muestra a continuación:

```
72
73     if (parent):
74         self.peso = parent.peso+1
75     else:
76         self.peso=0
77
```

Para que el A* funcione se debe hacer el cálculo de la función fn que es el peso + heurística

```
81     func f_nAS():
82         self.fn = self.peso + self.h
83
```

Siguientes líneas de código:


```

v >| func heuristic(goal):
>| >| self.h = abs(goal.x - self.posx) + abs(goal.y - self.posy)
>|
v >| func expand(goal, fringe, tile_map):
>| >| var move
>| >| var cell
>| >|
>| >| cell = tile_map.get_cell(self.posx, self.posy - 1)
v >| >| if cell == 12 or cell == 13 or cell == 14:
>| >| >| move = GreedyNode.new(self.posx, self.posy - 1, self)
>| >| >| move.heuristic(goal)
>| >| >| move.f_nAS() #se saca el estimado
>| >| >| self.branches.append(move)
>| >|
>| >| cell = tile_map.get_cell(self.posx + 1, self.posy)
v >| >| if cell == 12 or cell == 13 or cell == 14:
>| >| >| move = GreedyNode.new(self.posx + 1, self.posy, self)
>| >| >| move.heuristic(goal)
>| >| >| move.f_nAS()
>| >| >| self.branches.append(move)
>| >|
>| >| cell = tile_map.get_cell(self.posx, self.posy + 1)
v >| >| if cell == 12 or cell == 13 or cell == 14:
>| >| >| move = GreedyNode.new(self.posx, self.posy + 1, self)
>| >| >| move.heuristic(goal)
>| >| >| move.f_nAS()
>| >| >| self.branches.append(move)
>| >|
>| >| cell = tile_map.get_cell(self.posx - 1, self.posy)
v >| >| if cell == 12 or cell == 13 or cell == 14:
>| >| >| move = GreedyNode.new(self.posx - 1, self.posy, self)
>| >| >| move.heuristic(goal)
>| >| >| move.f_nAS()
>| >| >| self.branches.append(move)

```

Donde el valor de la heurística se almacena dentro de la variable h

Y se realiza la acción move.f_AS() que es para...

Posteriormente se agregan los nodos a la franja por medio de AS:

```

>| >| for branch in self.branches: #Ordenamiento por fn
>| >| >| if fringe == []:
>| >| >| fringe.append(branch)
>| >| >| else:
>| >| >| >| var pos = 0
>| >| >| >| for node in fringe:
>| >| >| >| >| if node.fn > branch.fn:
>| >| >| >| >| fringe.insert(pos, branch)
>| >| >| >| >| break
>| >| >| >| >| pos += 1

```

Siguientes líneas de código, que es lo mismo que se realizó en el fantasma rojo, previamente explicado:

```
131 > func search(goal, visits, fringe, tile_map):# la busqueda greedy es totalmente recursiva
132 > > if self.posx == goal.x and self.posy == goal.y:
133 > > > print("Fantasma llegó a PACMAN X_x")
134 > > > var way = []
135 > > > var world_pos
136 > > > world_pos = tile_map.map_to_world(Vector2(self.posx, self.posy))
137 > > > world_pos.x = world_pos.x + 1
138 > > > world_pos.y = world_pos.y + 1
139 > > > way.append(world_pos)
140 > > > var parent = self.parent
141 > > > while parent:
142 > > > > world_pos = tile_map.map_to_world(Vector2(parent.posx, parent.posy))
143 > > > > world_pos.x = world_pos.x + 1
144 > > > > world_pos.y = world_pos.y + 1
145 > > > > way.append(world_pos)
146 > > > > parent = parent.parent
147 > > > way.invert()
148 > > > return PoolVector2Array(way)
149 > >
150 > > var is_visited = false
151 > > if not visits == []:
152 > > > for visited in visits:
153 > > > > if self.posx == visited.posx and self.posy == visited.posy:
154 > > > > > is_visited = true
155 > > > > > break
156 > > > >
157 > > if not is_visited:
158 > > > visits.append(self)
159 > > > self.expand(goal, fringe, tile_map)
160 > >
161 > > if not fringe == []:
162 > > > return fringe.pop_front().search(goal, visits, fringe, tile_map)
163 > > print("No hay solucion de rosa")
164 > > return []
165 > >
```