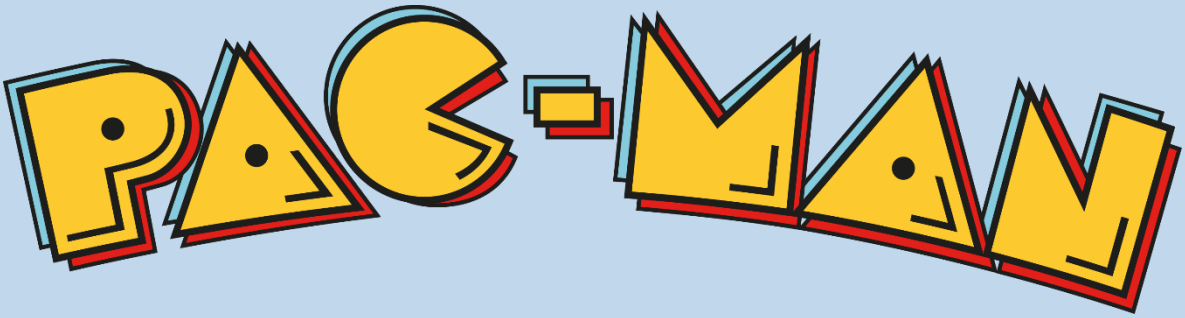
Documentación de PROYECTO JUEGO GODOT



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.

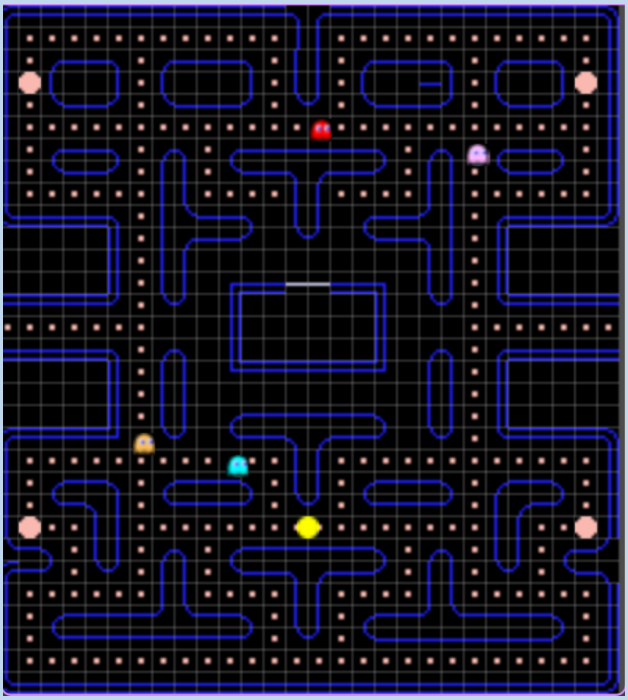
Donde cada fantasma será regido por métodos de búsqueda IA

Fantasma Azul : BPA

Fantasma Naranja: BPP

Fantasma Rojo : Greedy

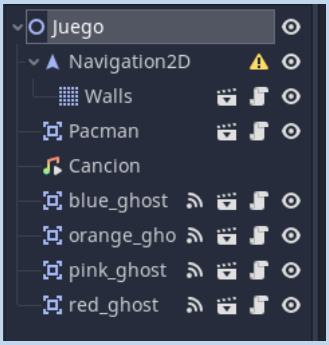
Fantasma Rosa: A*



ESTRUCTURA DEL JUEGO

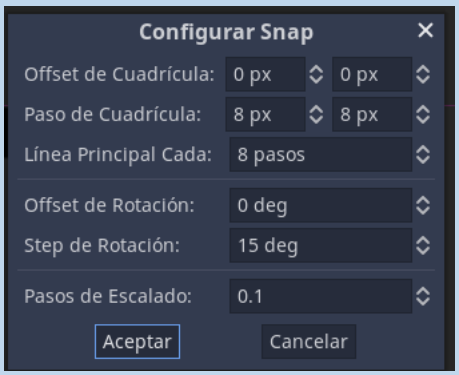
El proyecto está estructurado con un nodo principal llamado juego por donde por medio de “Instanciar escena hija”se le agregarán las escenas creadas posteriormente, para las paredes del juego, la navegación, los fantasmas, el pacman, las escenas que se proyectan al perder o ganar, los tiles, el sonido,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

EXPLICACIÓN DE CADA NODO:

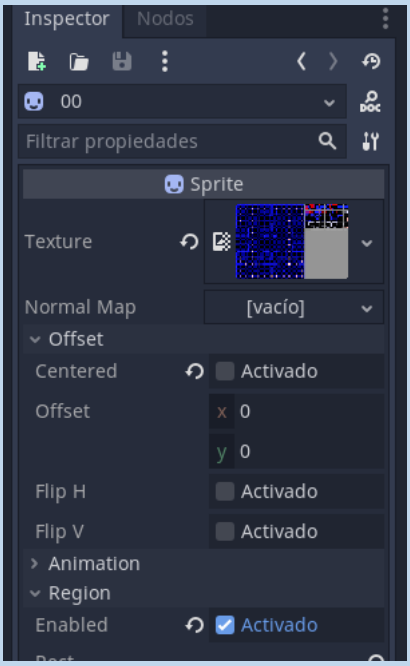


- **Navigation2D:** Este nodo se realizó para que en él se le agregó un nodo hijo llamado “Walls” que son las paredes del juego, para la creación de este walls se creó un documento tiles.tres dentro de la escena “Walls” que nos sirvió para posiciones tile por tile y formar así el laberinto.

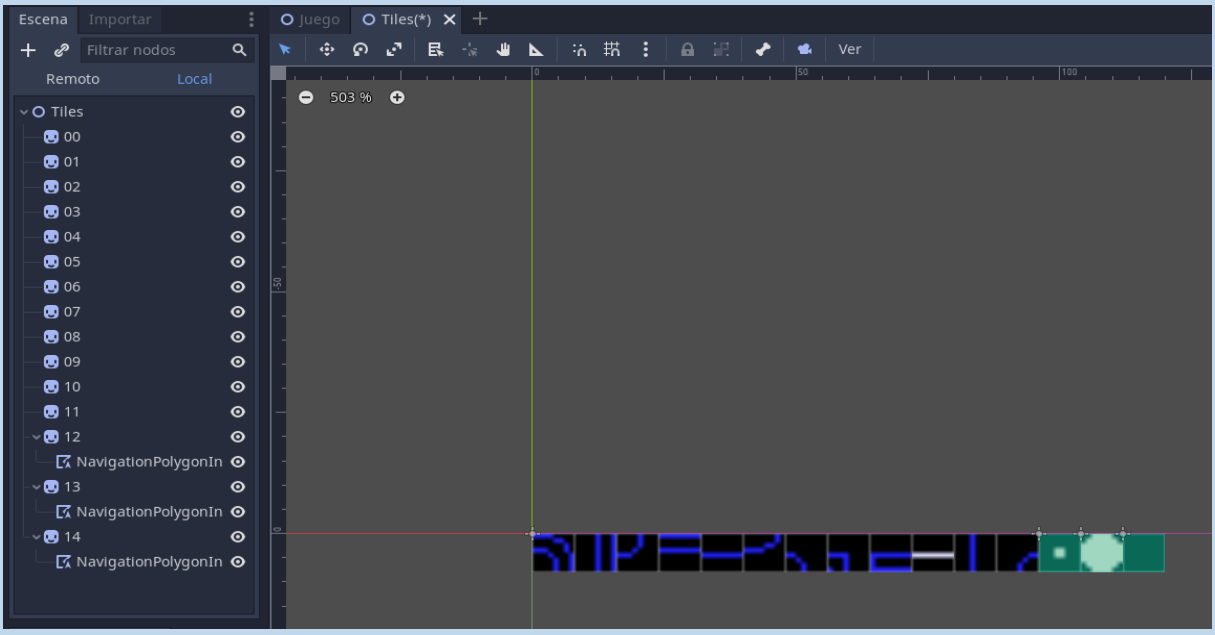
Para hacer estos tiles lo que se hace es que se creó una escena llamada tiles donde se agregaron 14 sprites que serán los ocupados dentro del laberinto, para esto se realizó un pixel snap, dándoles una dimensión de 8 x 8



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

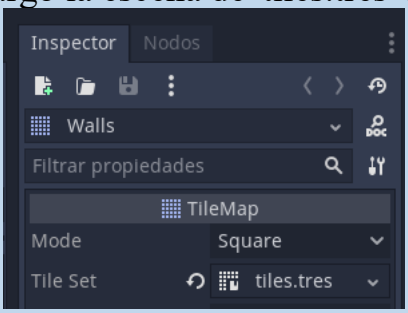


Se selecciona la fracción de imagen deseada para cada sprite y se forma lo siguiente:

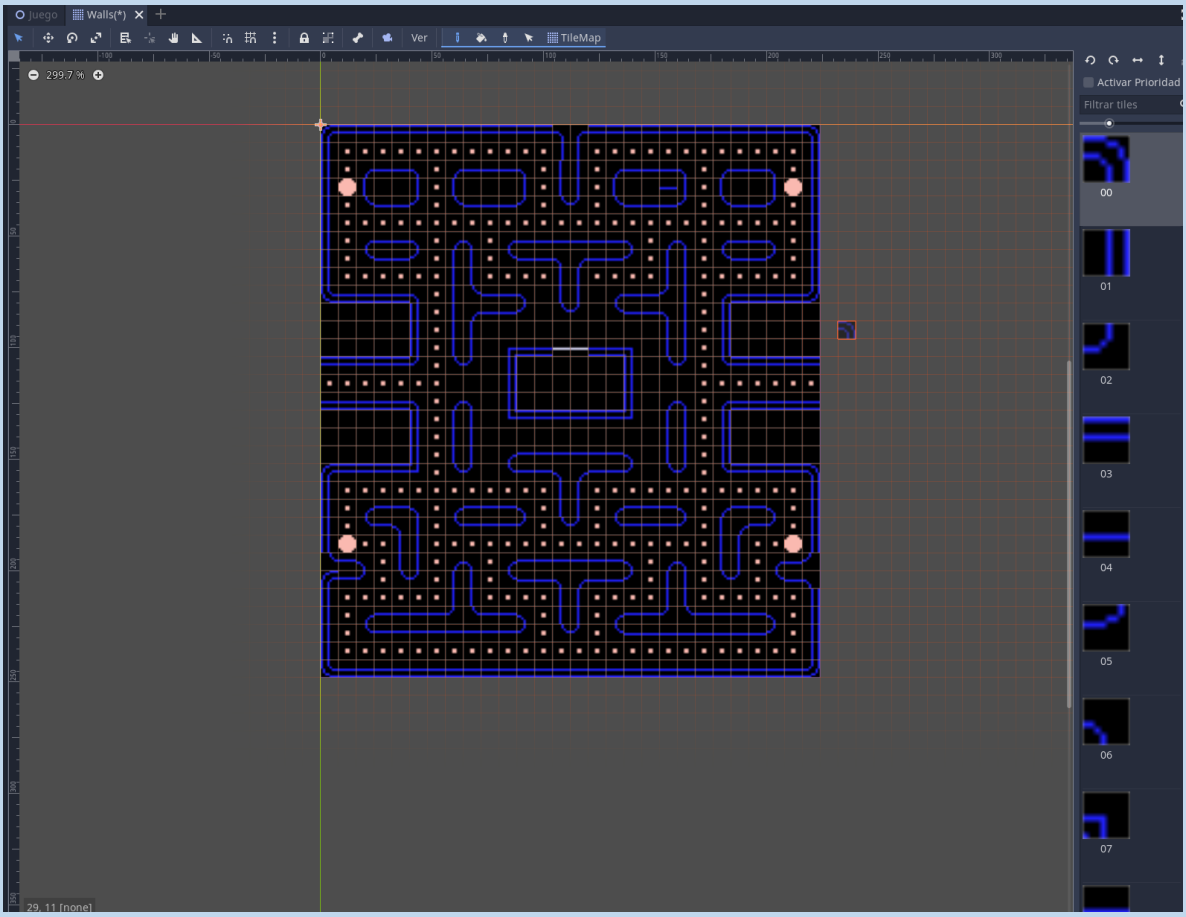


Donde los sprites 12, 13, 14 se les agrega un nodo hijo de Polígono de Navegación que son donde se van a poder trasladar nuestros personajes.

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:

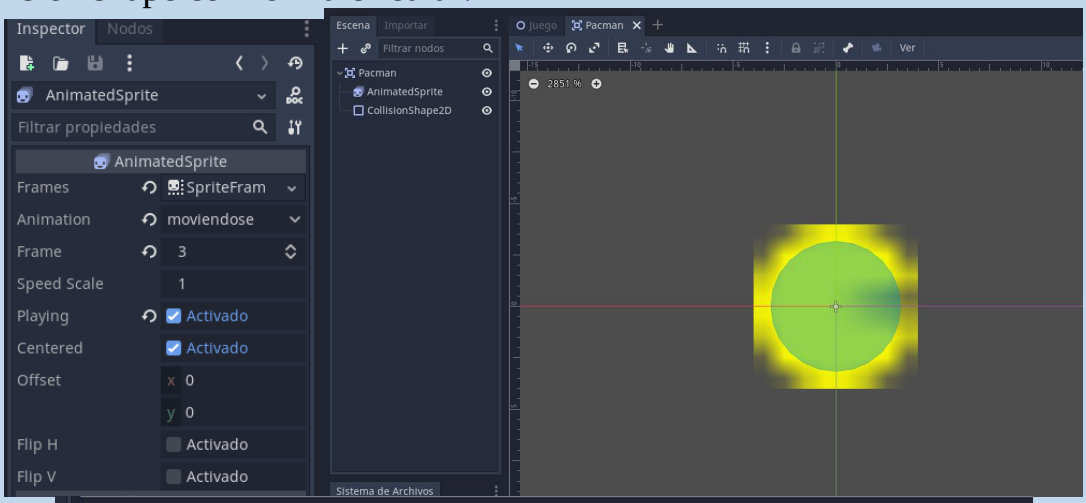
```gdscript
extends TileMap

onready var half_cell_size = get_cell_size()/2                          #Variable para posición enmedio de la celda
onready var player = get_parent().get_parent().get_node("Pacman")       #Obteniendo nodos de cada personaje
onready var FRosa = get_parent().get_parent().get_node("pink_ghost")
onready var FRojo = get_parent().get_parent().get_node("red_ghost")
onready var FNaranja = get_parent().get_parent().get_node("orange_ghost")
onready var FAzul = get_parent().get_parent().get_node("blue_ghost")

func get_player_init_pos():                 #La posición inicial del jugador
    var pos = map_to_world(Vector2(14,23))  #Convierte tiles a pixeles
    pos.y += half_cell_size.y               #Posiciona el player en medio de la celda
    return pos

func is_tile_vacant(pos, direction):                                    #Evalúa y devuelve los tiles de navegación
    var curr_tile = world_to_map(pos)
    var next_tile = get_cellv(curr_tile + direction)
    var next_tile_pos = Vector2()
    if (next_tile == 12 or next_tile == 13 or next_tile == 14):
        next_tile_pos = map_to_world(curr_tile + direction)+half_cell_size
    else:
        next_tile_pos = relocate(pos)
    return next_tile_pos

func relocate(pos):
    var tile = world_to_map(pos)                    #vuelve la posicón pixeles a tiles
    return map_to_world(tile) + half_cell_size      #los tiles regresa a pixeles pero ahora centrados

func comer(pos):                            #Funcion del pacman comiendo
    var curr_tile= world_to_map(pos)
    var tile = get_cellv(curr_tile)
    if (tile == 12 or tile == 13):          #Si el pacman pasa por los tile 12 o 13, que son las bolitas, estaría "comiendo"
        set_cellv(curr_tile, 14)            #Los tiles pasan a ser tipo tile 14 que es el recuadro negro sin pac dots

func _process(delta):
    var contador=0
    for i in range(get_used_rect().size.x):         #Contador de pacdots, si no hay bolitas dentro del mapa, ganas
        for j in range(get_used_rect().size.y):
            var tile = get_cell(i,j)
            if (tile == 12):
                contador += 1
    if (contador == 0):
        print("*****FELICIDADES HAS GANADO*****")
        set_process(false)
        get_tree().change_scene("res://YouWin.tscn")    #llama script para proyectar escena ganadora

func get_fantasma_pos():                    #Posicion incial de los fantasmas
    var pos = map_to_world(Vector2(14,11))
    pos.y += half_cell_size.y               #Posiciona el fantasma enmedio de la celda
    return pos                              #Regresa la posición

```

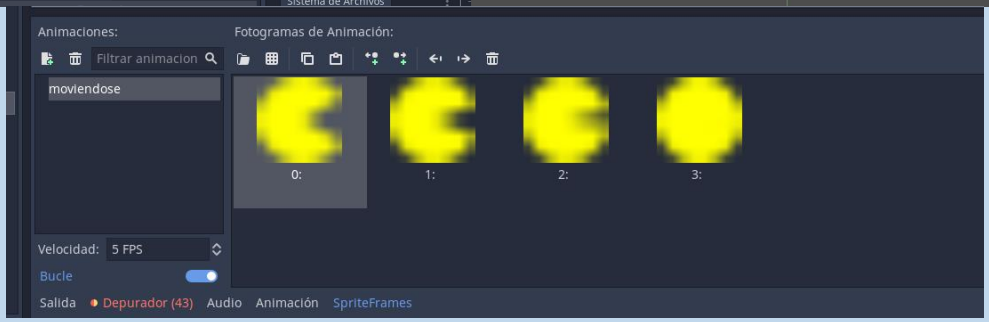
Descripción de funciones:

- 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.





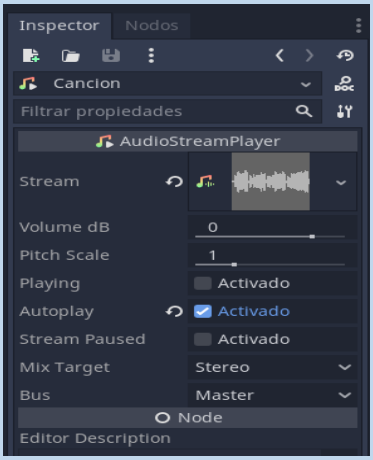
SCRIPT DE PACMAN:

```gdscript
extends Area2D

var direction = Vector2(0,0)
var speed= 15
onready var walls = get_parent().get_node("Navigation2D/Walls")   #Obtiene nodo walls

func _ready():
    $AnimatedSprite.play("moviendose")                          #Reproduce la animación llamada "moviendose"
    position = walls.get_player_init_pos()                      #Ubica al pacman en su posición inicial

func _process(delta):                                           #Función para controlar le pacman por medio de
    if Input.is_action_pressed("ui_up"):                        # las teclas de dirección
        direction = Vector2(0,-1)
        rotation = deg2rad(-90)                                 #Rota el sprite en dirección al movimiento
    elif Input.is_action_pressed("ui_down"):
        direction = Vector2(0,1)
        rotation = deg2rad(90)                                  #Rota el sprite en dirección al movimiento
    elif Input.is_action_pressed("ui_left"):
        direction = Vector2(-1,0)
        rotation = deg2rad(180)                                 #Rota el sprite en dirección al movimiento
    elif Input.is_action_pressed("ui_right"):
        direction = Vector2(1,0)
        rotation = deg2rad(0)                                   #Rota el sprite en dirección al movimiento

    var pos_to_move = walls.is_tile_vacant(position, direction)  #Llama a función dentro de walls para ubicar el pacman dentro del m
    if (direction != Vector2(0,0)):                             #Si la dirección no está en el vector (0,0)
        position = position.linear_interpolate(pos_to_move, speed * delta)  #Mueve de un punto a otro el pacman respecto a su veloci
        walls.comer(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.

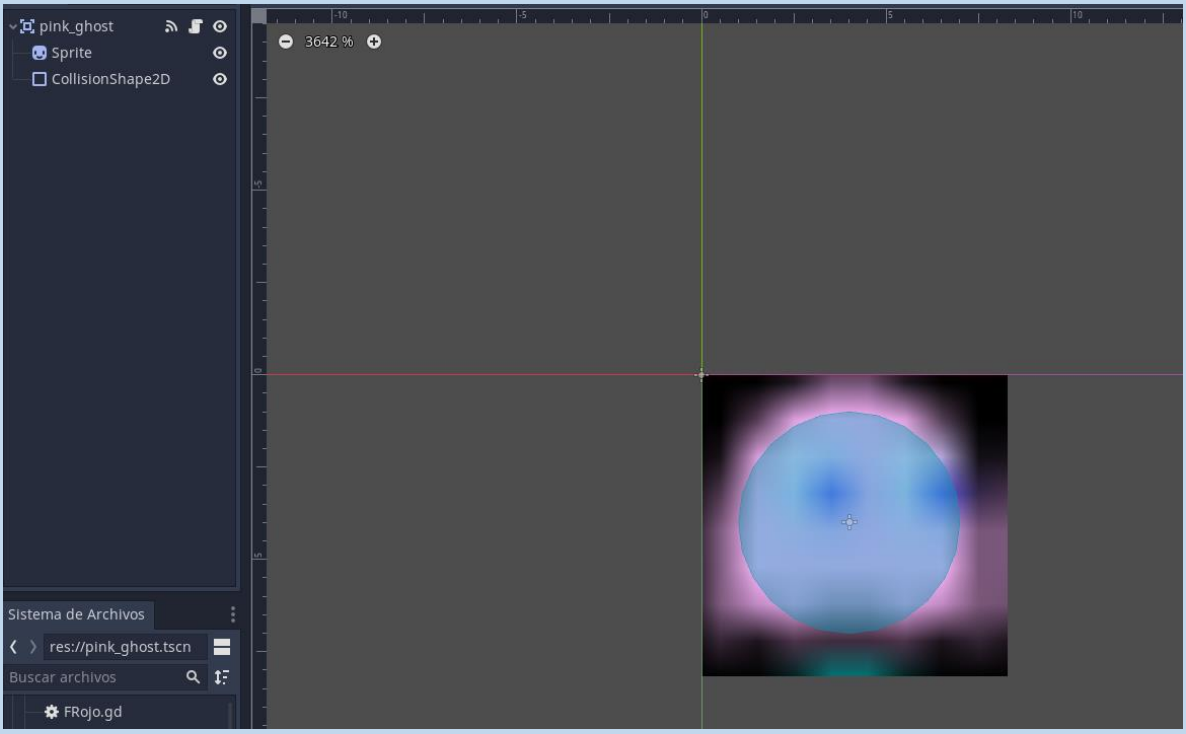
- # NODO CANCIÓN



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**

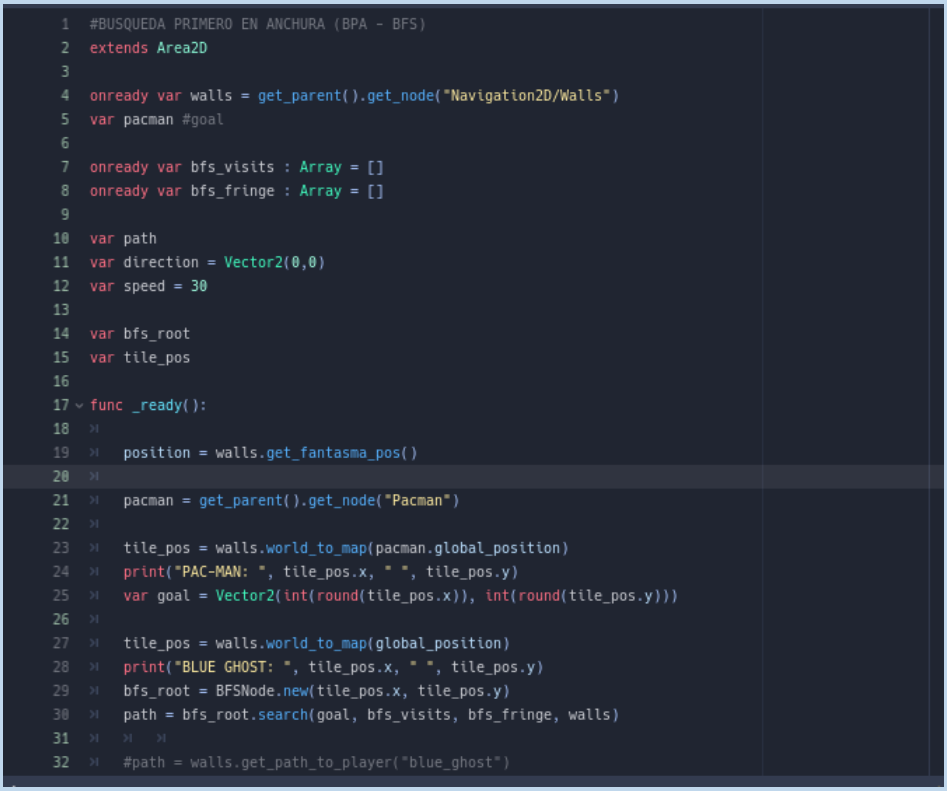
Se creó una nueva escena donde se creó un nodo padre tipo Area2D y se agregó dos nodos hijos, un tipo sprite y otro tipo collision shape de forma circular.



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).



```gdscript
#BUSQUEDA PRIMERO EN ANCHURA (BPA - BFS)
extends Area2D

onready var walls = get_parent().get_node("Navigation2D/Walls")
var pacman #goal

onready var bfs_visits : Array = []
onready var bfs_fringe : Array = []

var path
var direction = Vector2(0,0)
var speed = 30

var bfs_root
var tile_pos

func _ready():

    position = walls.get_fantasma_pos()

    pacman = get_parent().get_node("Pacman")

    tile_pos = walls.world_to_map(pacman.global_position)
    print("PAC-MAN: ", tile_pos.x, " ", tile_pos.y)
    var goal = Vector2(int(round(tile_pos.x)), int(round(tile_pos.y)))

    tile_pos = walls.world_to_map(global_position)
    print("BLUE GHOST: ", tile_pos.x, " ", tile_pos.y)
    bfs_root = BFSNode.new(tile_pos.x, tile_pos.y)
    path = bfs_root.search(goal, bfs_visits, bfs_fringe, walls)
        
    #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 busqueda 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.

```
#path = walls.get_path_to_player("blue_ghost")

func _process(delta):
    if (path.size() > 1):
        var pos_to_move = path[0]
        direction = (pos_to_move - position).normalized()
        var distance = position.distance_to(path[0])
        if (distance > 1):
            position += speed * delta * direction
        else:
            path.remove(0)
    else:
        position = walls.get_fantasma_pos()

        pacman = get_parent().get_node("Pacman")

        tile_pos = walls.world_to_map(pacman.global_position)
        print("PAC-MAN: ", tile_pos.x, " ", tile_pos.y)
        var goal = Vector2(int(round(tile_pos.x)), int(round(tile_pos.y)))

        tile_pos = walls.world_to_map(global_position)
        print("BLUE GHOST: ", tile_pos.x, " ", tile_pos.y)
        var bfs_root = BFSNode.new(tile_pos.x, tile_pos.y)
        bfs_visits.clear()
        bfs_fringe.clear()
        path = bfs_root.search(goal, bfs_visits, bfs_fringe, walls)

        #path = walls.get_path_to_player("blue_ghost")
```

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.

```
func _on_blue_ghost_area_entered(area):
    if (area.name == "Pacman"):
        print("GAME OVER")
        get_tree().change_scene("res://GameOver.tscn")
```

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.

```
class BFSNode:
    var posx
    var posy
    var branches
    var parent

    func _init(posx, posy, parent = null):
        self.posx = posx
        self.posy = posy
        self.branches = []
        self.parent = parent

    func print_position():
        print("x: ", self.posx, ", y: ", self.posy)

```

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.

```
    func print_position():
        print("x: ", self.posx, ", y: ", self.posy)

    func expand(goal, fringe, tile_map):
        var move
        var cell

        cell = tile_map.get_cell(self.posx, self.posy - 1)
        if cell == 12 or cell == 13 or cell == 14:
            move = BFSNode.new(self.posx, self.posy - 1, self)
            self.branches.append(move)

        cell = tile_map.get_cell(self.posx + 1, self.posy)
        if cell == 12 or cell == 13 or cell == 14:
            move = BFSNode.new(self.posx + 1, self.posy, self)
            self.branches.append(move)

        cell = tile_map.get_cell(self.posx, self.posy + 1)
        if cell == 12 or cell == 13 or cell == 14:
            move = BFSNode.new(self.posx, self.posy + 1, self)
            self.branches.append(move)

        cell = tile_map.get_cell(self.posx - 1, self.posy)
        if cell == 12 or cell == 13 or cell == 14:
            move = BFSNode.new(self.posx - 1, self.posy, self)
            self.branches.append(move)

        for branch in self.branches:          #no ordena los hijos, solo los mete a la franja
            fringe.append(branch)


    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

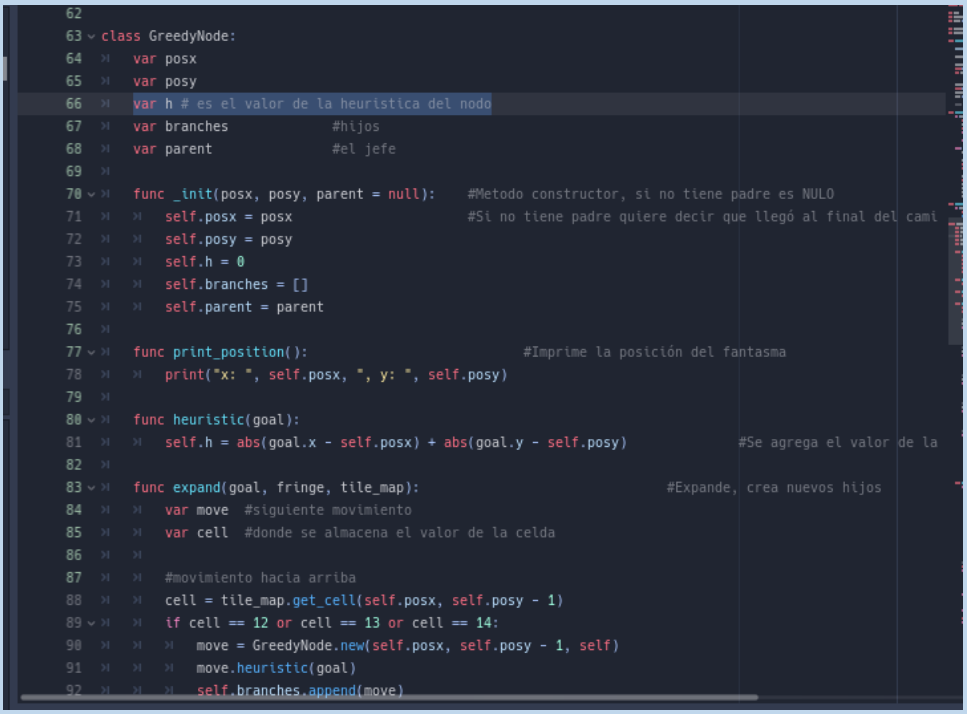
El código de este algoritmo es el mismo que el BPA solo que en esta ocasión los hijos se agregan al principio de la franja.

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

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

Para la busqueda 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 heuristica 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)
```

```
99              self.branches.append(move)
100
101         #movimiento hacia abajo
102         cell = tile_map.get_cell(self.posx, self.posy + 1)
103         if cell == 12 or cell == 13 or cell == 14:
104             move = GreedyNode.new(self.posx, self.posy + 1, self)
105             move.heuristic(goal)
106             self.branches.append(move)
107
108         #movimiento hacia la izquierda
109         cell = tile_map.get_cell(self.posx - 1, self.posy)
110         if cell == 12 or cell == 13 or cell == 14:
111             move = GreedyNode.new(self.posx - 1, self.posy, self)
112             move.heuristic(goal)
113             self.branches.append(move)
114
115         #Agrega hijos a la franja odenados por heurística
116         for branch in self.branches:
117             if fringe == []:
118                 fringe.append(branch)        #si está vacía agrega hijo
119             else:
120                 var pos = 0
121                 for node in fringe:
122                     if node.h > branch.h:              #ordena por heurística de menor a mayor
123                         fringe.insert(pos, branch)
124                         break
125                     pos += 1
126
```

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

En este caso

```
class GreedyNode:
    var posx
    var posy
    var h # es el valor de la heuristica del nodo
    var branches
    var parent
    var peso
    var fn

    func _init(posx, posy, parent = null):
        self.posx = posx
        self.posy = posy
        self.h = 0
        self.fn
        self.branches = []
        self.parent = parent
```

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 acontinuación:

```
    if (parent):                            #AGREGAMOS PESO
        self.peso = parent.peso+1           #Le agrega peso a cada nodo
    else:
        self.peso=0                         #Si no hay padre el peso es igual a 0
```

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

```
    func f_nAS():                           #A*
        self.fn = self.peso + self.h        #Calculo fn = peso + heuristica
```

Siguientes líneas de código:

```
func heuristic(goal):
    self.h = abs(goal.x - self.posx) + abs(goal.y - self.posy)

func expand(goal, fringe, tile_map):
    var move
    var cell

    cell = tile_map.get_cell(self.posx, self.posy - 1)
    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)
    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)
    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)
    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 obtener el valor fn del nodo

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:

```
func search(goal, visits, fringe, tile_map):# la busqueda greedy es totalmente recursiva
    if self.posx == goal.x and self.posy == goal.y:
        print("Fantasma llegó a PACMAN X_x")
        var way = []
        var world_pos
        world_pos = tile_map.map_to_world(Vector2(self.posx, self.posy))
        world_pos.x = world_pos.x + 1
        world_pos.y = world_pos.y + 1
        way.append(world_pos)
        var parent = self.parent
        while parent:
            world_pos = tile_map.map_to_world(Vector2(parent.posx, parent.posy))
            world_pos.x = world_pos.x + 1
            world_pos.y = world_pos.y + 1
            way.append(world_pos)
            parent = parent.parent
        way.invert()
        return PoolVector2Array(way)

    var is_visited = false
    if not visits == []:
        for visited in visits:
            if self.posx == visited.posx and self.posy == visited.posy:
                is_visited = true
                break
            
    if not is_visited:
        visits.append(self)
        self.expand(goal, fringe, tile_map)

    if not fringe == []:
        return fringe.pop_front().search(goal, visits, fringe, tile_map)
    print("No hay solucion de rosa")
    return []

```