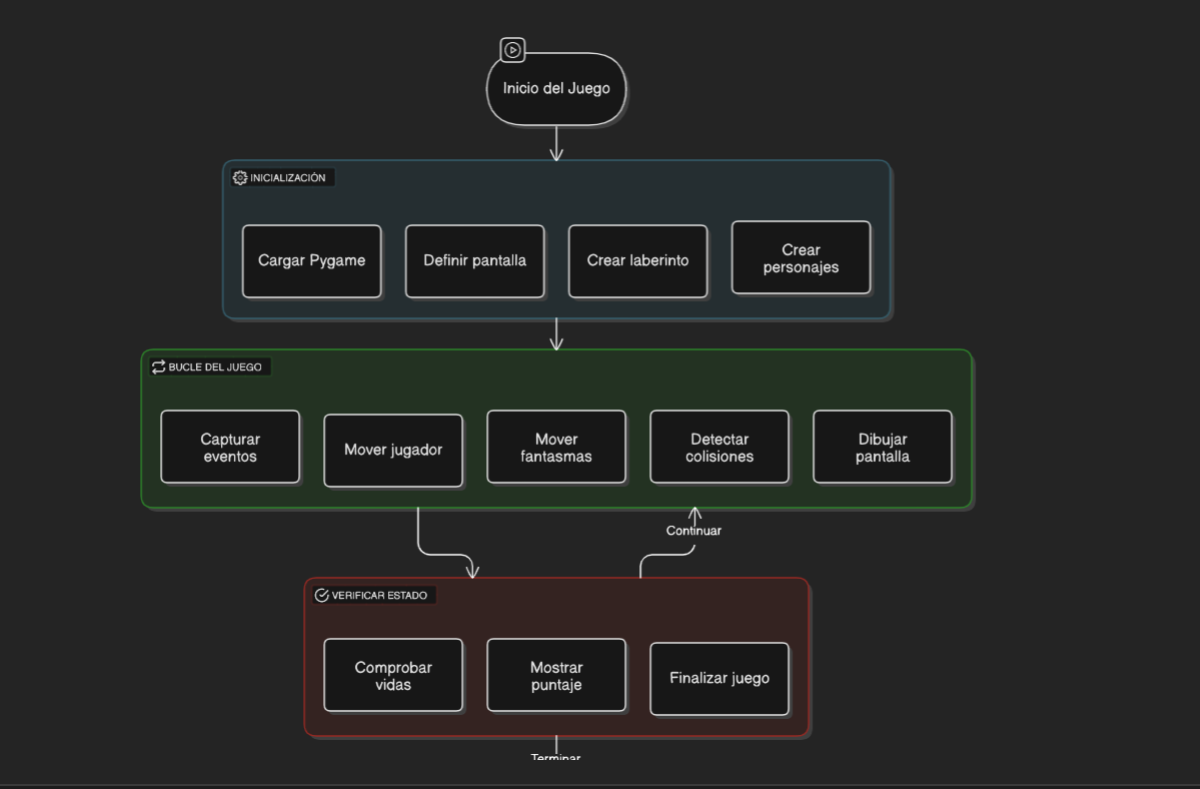
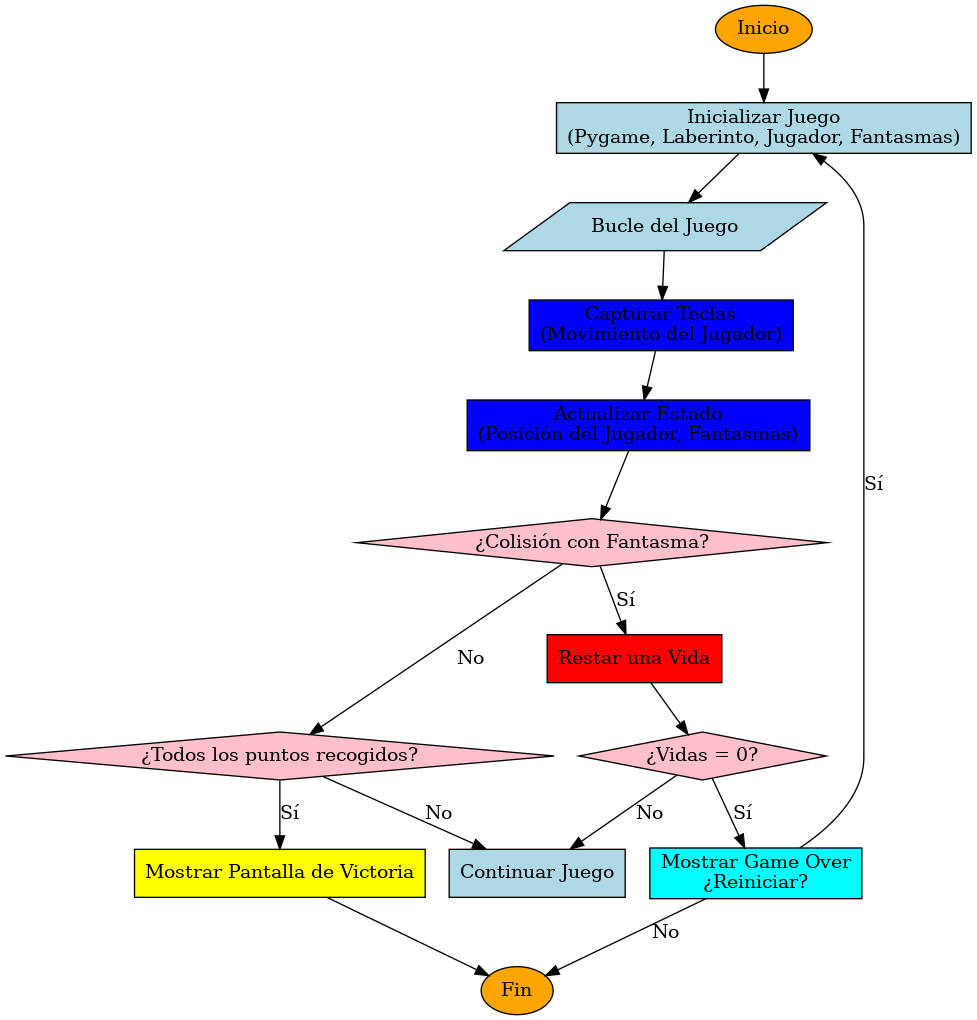
# Documentación del Juego Pac-Man (come cocos )

## . Diagrama de Bloques

El siguiente diagrama de bloques nos representa la estructura general del juego y la interacción entre sus componentes:



## . Diagrama de Flujo

El siguiente diagrama de flujo describe la lógica principal del juego:

### ****1. Importaciones****

python

CopiarEditar

import pygame

import sys

import random

from tkinter import messagebox, Tk

#### Descripción:

* **pygame**: Se utiliza para desarrollar juegos en 2D y manejar gráficos, eventos, y sonidos.
* **sys**: Proporciona acceso a variables y funciones relacionadas con el sistema.
* **random**: Se utiliza para generar movimientos aleatorios para los fantasmas.
* **tkinter.messagebox** y **Tk**: Se utilizan para mostrar mensajes emergentes cuando el jugador pierde todas sus vidas.

### ****2. Inicialización de**** pygame

python

CopiarEditar

pygame.init()

#### Descripción:

Inicializa todos los módulos necesarios de pygame para empezar a usar sus funcionalidades, como la creación de ventanas, manejo de eventos, etc.

### ****3. Dimensiones y colores****

python

CopiarEditar

CELL\_SIZE = 30

ROWS = 11

COLS = 21

WIDTH, HEIGHT = COLS \* CELL\_SIZE, ROWS \* CELL\_SIZE + 50

#### Descripción:

Define el tamaño de cada celda del laberinto, el número de filas y columnas, y el tamaño total de la pantalla.

python

CopiarEditar

BLACK = (0, 0, 0)

WHITE = (255, 255, 255)

BLUE = (0, 0, 255)

YELLOW = (255, 255, 0)

RED = (255, 0, 0)

#### Descripción:

Define algunos colores en formato RGB que se usarán para dibujar el laberinto, al jugador y los fantasmas.

### ****4. Definición del laberinto****

python

CopiarEditar

MAZE = [

# (contenido del laberinto)

]

#### Descripción:

Define el diseño del laberinto usando una lista bidimensional. Cada número representa un tipo de celda:

* **1**: Representa una pared.
* **0**: Representa un camino libre.
* **2**: Representa un punto que el jugador puede recoger.
* **3**: Representa el área de aparición de los fantasmas.

### ****5. Inicialización de la pantalla y el reloj****

python

CopiarEditar

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Pac-Man")

clock = pygame.time.Clock()

#### Descripción:

* **screen**: Crea la ventana del juego con las dimensiones establecidas previamente.
* **pygame.display.set\_caption**: Establece el título de la ventana como "Pac-Man".
* **clock**: Crea un objeto de reloj para controlar la tasa de actualización del juego (los FPS).

### ****6. Clase**** Player

python

CopiarEditar

class Player:

def \_\_init\_\_(self):

self.x, self.y = 1, 1

self.lives = 3

self.score = 0

#### Descripción:

Esta clase define al jugador. El constructor inicializa su posición en el laberinto, sus vidas y su puntaje.

python

CopiarEditar

def move(self, dx, dy):

new\_x = self.x + dx

new\_y = self.y + dy

if 0 <= new\_y < ROWS and 0 <= new\_x < COLS and MAZE[new\_y][new\_x] != 1:

self.x = new\_x

self.y = new\_y

if MAZE[self.y][self.x] == 2:

self.score += 10

MAZE[self.y][self.x] = 0

#### Descripción:

Esta función mueve al jugador dentro del laberinto en la dirección indicada por dx y dy, si el movimiento es válido (es decir, no choca con una pared). Si el jugador recoge un punto (valor 2), el puntaje aumenta y el punto desaparece.

### ****7. Clase**** Ghost

python

CopiarEditar

class Ghost:

def \_\_init\_\_(self, x, y):

self.x, self.y = x, y

#### Descripción:

Esta clase define a los fantasmas. El constructor recibe las coordenadas iniciales del fantasma en el laberinto.

python

CopiarEditar

def move(self):

directions = [(0,1), (1,0), (0,-1), (-1,0)]

random.shuffle(directions)

for dx, dy in directions:

new\_x = self.x + dx

new\_y = self.y + dy

if 0 <= new\_y < ROWS and 0 <= new\_x < COLS and MAZE[new\_y][new\_x] != 1:

self.x = new\_x

self.y = new\_y

break

#### Descripción:

Esta función permite que los fantasmas se muevan aleatoriamente dentro del laberinto. Se barajan las direcciones posibles y el fantasma se mueve en la primera dirección válida que encuentre (no puede moverse a través de paredes).

### ****8. Inicialización del jugador y los fantasmas****

player = Player()

ghosts = [Ghost(9, 4), Ghost(10, 4), Ghost(5, 4), Ghost(14, 4)]

#### Descripción:

Se crea una instancia del jugador y se inicializan cuatro fantasmas en posiciones específicas del laberinto.

### ****9. Bucle principal del juego****

while not game\_over:

screen.fill(BLACK)

#### Descripción:

El bucle principal del juego se ejecuta hasta que game\_over sea True. Aquí es donde se actualiza y renderiza cada fotograma del juego.

### ****10. Captura de eventos****

for event in pygame.event.get():

if event.type == pygame.QUIT:

pygame.quit()

sys.exit()

elif event.type == pygame.KEYDOWN:

if event.key == pygame.K\_LEFT:

player.move(-1, 0)

elif event.key == pygame.K\_RIGHT:

player.move(1, 0)

elif event.key == pygame.K\_UP:

player.move(0, -1)

elif event.key == pygame.K\_DOWN:

player.move(0, 1)

#### Descripción:

Este bloque captura los eventos que ocurren en el juego, como el cierre de la ventana o las pulsaciones de teclas. Dependiendo de la tecla que se presione, el jugador se moverá en una dirección específica.

### ****11. Renderización del laberinto****

for row in range(ROWS):

for col in range(COLS):

if MAZE[row][col] == 1:

pygame.draw.rect(screen, BLUE, (col \* CELL\_SIZE, row \* CELL\_SIZE, CELL\_SIZE, CELL\_SIZE))

elif MAZE[row][col] == 2:

pygame.draw.circle(screen, WHITE, (col \* CELL\_SIZE + CELL\_SIZE // 2, row \* CELL\_SIZE + CELL\_SIZE // 2), 5)

#### Descripción:

Dibuja el laberinto en la pantalla. Las paredes se dibujan como rectángulos azules y los puntos se dibujan como círculos blancos.

### ****12. Dibujar al jugador****

pygame.draw.circle(screen, YELLOW, (player.x \* CELL\_SIZE + CELL\_SIZE // 2, player.y \* CELL\_SIZE + CELL\_SIZE // 2), 10)

#### Descripción:

Dibuja al jugador como un círculo amarillo en la pantalla, en su posición actual dentro del laberinto.

### ****13. Movimiento y renderización de los fantasmas****

for ghost in ghosts:

ghost.move()

pygame.draw.circle(screen, RED, (ghost.x \* CELL\_SIZE + CELL\_SIZE // 2, ghost.y \* CELL\_SIZE + CELL\_SIZE // 2), 10)

if player.x == ghost.x and player.y == ghost.y:

player.lives -= 1

if player.lives == 0:

root = Tk()

root.withdraw()

if messagebox.askyesno("Game Over", "¿Quieres jugar de nuevo?"):

player = Player()

else:

game\_over = True

root.destroy()

#### Descripción:

Cada fantasma se mueve de manera aleatoria en cada iteración del bucle. Luego se dibujan como círculos rojos. Si un fantasma colisiona con el jugador (misma posición), el jugador pierde una vida. Si las vidas del jugador llegan a 0, se muestra un mensaje para reiniciar o terminar el juego.

### ****14. Mostrar vidas del jugador****

font = pygame.font.Font(None, 36)

lives\_text = font.render(f"Vidas: {player.lives}", True, WHITE)

screen.blit(lives\_text, (10, HEIGHT - 40))

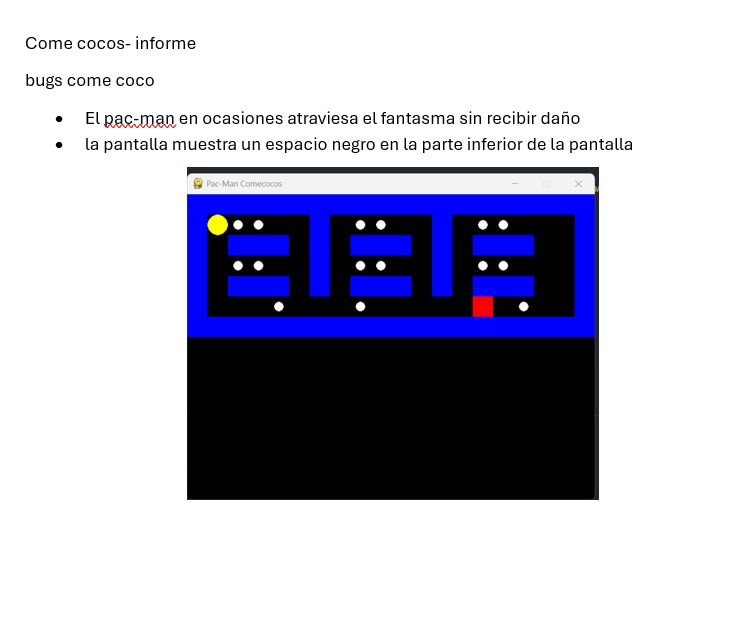
#### Descripción:

Se muestra el número de vidas restantes del jugador en la parte inferior de la pantalla.

### ****15. Actualización de la pantalla****

pygame.display.flip()

clock.tick(10)

****

**Bug 1: Pac-Man atraviesa el fantasma sin recibir daño**

**Explicación**: Este problema ocurre porque el juego no detecta correctamente cuando Pac-Man y el fantasma colisionan. Es posible que la detección de colisiones, que es la parte del código encargada de comprobar si Pac-Man y el fantasma están en la misma posición, no esté funcionando bien. Como resultado, Pac-Man puede atravesar al fantasma sin recibir daño, cuando lo normal sería perder una vida.

**Solución**:

* Primero, necesitamos revisar el código que maneja la **detección de colisiones**. Tenemos que asegurarnos de que las coordenadas de Pac-Man y del fantasma se están comparando de manera correcta. Podría ser útil implementar o mejorar el sistema de "hitboxes" (que define el área en la que se produce la colisión), para que la detección sea más precisa.
* Además, habría que verificar si el juego está comprobando las colisiones lo suficientemente rápido. Si la velocidad del juego es alta y no se verifica la colisión en cada frame o con la frecuencia necesaria, puede que el choque no se detecte a tiempo. Ajustar la **frecuencia de actualización** puede resolver este problema.

**Bug 2: La pantalla muestra un espacio negro en la parte inferior**

**Explicación**: En este caso, hay un área negra que aparece en la parte inferior de la pantalla del juego y no debería estar ahí. Este problema generalmente se debe a que el juego no está utilizando correctamente el espacio disponible en la pantalla o ventana. Podría ser que el área del juego no esté ajustada a la resolución de la pantalla o que se esté mostrando un área más grande de lo que realmente necesita el juego.

**Solución**:

* Lo primero es revisar el código de **renderizado o ajuste de pantalla** para asegurarnos de que el área del juego ocupe todo el espacio de la ventana. Podemos configurar el juego para que ajuste automáticamente su tamaño al de la ventana o asegurarnos de que el área visible del juego coincida con el laberinto que se muestra.
* También sería bueno revisar si hay algún **contenedor** o elemento que esté causando ese espacio adicional y ajustar sus propiedades para eliminar ese espacio vacío.
* Si el juego tiene diferentes **modos de resolución o escalado**, es importante comprobar que está configurado correctamente para evitar estos errores de visualización.

Informe de Errores y Soluciones en el Desarrollo del Juego Pac-Man

1. Error: El juego no reiniciaba correctamente tras perder todas las vidas

Descripción:

Cuando Pac-Man perdía todas las vidas, el programa mostraba una ventana emergente preguntando si el jugador quería reiniciar. Sin embargo, si el jugador elegía "Sí", el juego no se reiniciaba correctamente, ya que los fantasmas no volvían a sus posiciones iniciales.

Solución:

Se modificó la lógica para reiniciar correctamente tanto el jugador como los fantasmas. Se actualizó el código de reinicio dentro de la verificación de vidas:

python

Copiar

Editar

if player.lives == 0:

root = Tk()

root.withdraw()

if messagebox.askyesno("Game Over", "¿Quieres jugar de nuevo?"):

player = Player()

ghosts = [Ghost(9, 4), Ghost(10, 4), Ghost(5, 4), Ghost(14, 4)] # Reiniciar fantasmas

else:

game\_over = True

root.destroy()

2. Error: El juego no detectaba la victoria correctamente

Descripción:

El juego no tenía una condición clara para determinar cuándo el jugador había comido todos los puntos y ganado la partida.

Solución:

Se agregó una función check\_victory() que verifica si quedan puntos en el laberinto. Si no quedan más puntos, el juego muestra un mensaje de victoria y permite reiniciar o salir.

python

Copiar

Editar

def check\_victory():

for row in MAZE:

if 2 in row:

return False

return True

if check\_victory():

root = Tk()

root.withdraw()

if messagebox.askyesno("¡Felicidades!", "¡Has ganado! ¿Quieres jugar de nuevo?"):

player = Player()

ghosts = [Ghost(9, 4), Ghost(10, 4), Ghost(5, 4), Ghost(14, 4)] # Reiniciar fantasmas

else:

game\_over = True

root.destroy()

3. Error: Movimiento de los fantasmas era poco dinámico

Descripción:

Los fantasmas se movían aleatoriamente sin restricciones, lo que podía hacer que se quedaran atascados en ciertos puntos.

Solución:

Se mejoró el movimiento de los fantasmas para que intenten moverse en la dirección que los acerque más al jugador:

python

Copiar

Editar

def move\_towards\_player(self, player\_x, player\_y):

directions = [(0,1), (1,0), (0,-1), (-1,0)]

random.shuffle(directions)

best\_move = None

min\_distance = float('inf')

for dx, dy in directions:

new\_x = self.x + dx

new\_y = self.y + dy

if 0 <= new\_y < ROWS and 0 <= new\_x < COLS and MAZE[new\_y][new\_x] != 1:

distance = abs(new\_x - player\_x) + abs(new\_y - player\_y)

if distance < min\_distance:

min\_distance = distance

best\_move = (new\_x, new\_y)

if best\_move:

self.x, self.y = best\_move

En el bucle del juego, se llamó a esta función en lugar de move():

python

Copiar

Editar

for ghost in ghosts:

ghost.move\_towards\_player(player.x, player.y)

4. Error: Mensaje de "Game Over" bloqueaba el juego

Descripción:

Cuando el jugador perdía todas las vidas y aparecía el cuadro de diálogo, si el usuario no respondía, el juego quedaba congelado.

Solución:

Se añadió un pygame.time.delay(500) antes de abrir el cuadro de diálogo para evitar bloqueos:

python

Copiar

Editar

if player.lives == 0:

pygame.time.delay(500) # Espera antes de mostrar el mensaje

root = Tk()

root.withdraw()

if messagebox. askyesno("Game Over", "¿Quieres jugar de nuevo?"):

player = Player()

ghosts = [Ghost(9, 4), Ghost(10, 4), Ghost(5, 4), Ghost(14, 4)]

else:

game\_over = True

root.destroy()

Conclusión

Tras la implementación de estas correcciones, el juego ahora:

✅ Reinicia correctamente cuando el jugador pierde todas las vidas.

✅ Detecta cuándo el jugador ha ganado y permite reiniciar o salir.

✅ Los fantasmas tienen un movimiento más estratégico.

✅ El cuadro de "Game Over" ya no bloquea el juego.