Comecocos – Imanol Mosquera , Juan Esteban Izquierdo

**Introducción**

El presente documento tiene como propósito presentar de forma detallada el desarrollo y análisis del proyecto de programación titulado "Comecocos".

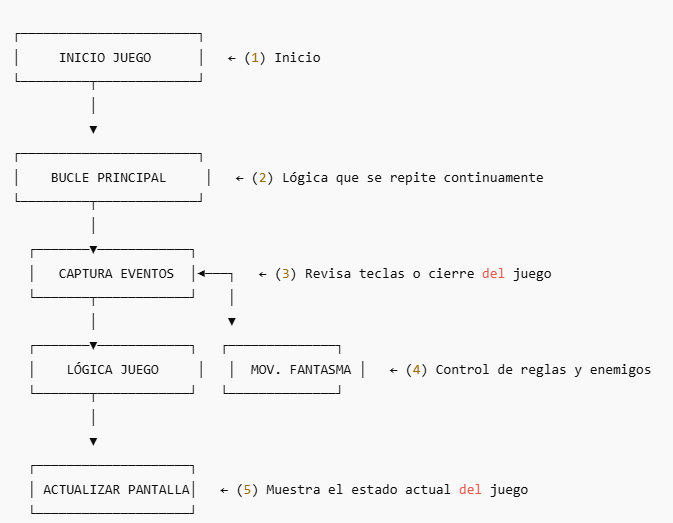
Este trabajo consistió en la implementación de una versión básica del clásico videojuego Pac-Man, utilizando el lenguaje de programación Python junto con las librerías Pygame y Tkinter para la creación de interfaces gráficas y manejo de eventos.

Durante el desarrollo del juego, se aplicaron conceptos fundamentales como estructuras de control, uso de funciones, manejo de matrices, interacción con el teclado, condiciones de finalización y visualización de resultados. Asimismo, se integraron mejoras al código base original, entre ellas la incorporación de una condición de victoria, la personalización de mensajes de final de juego y el control estructurado de errores.

Este informe está dividido en secciones que describen el código fuente, diagramas explicativos, documentación técnica, pruebas funcionales, errores corregidos, reflexiones personales y propuestas de mejora. Todo ello con el fin de evidenciar no solo el producto final, sino también el proceso lógico y técnico detrás de su construcción.

**Diagramas**

**Diagrama de Bloques**

****

**Explicación por partes:**

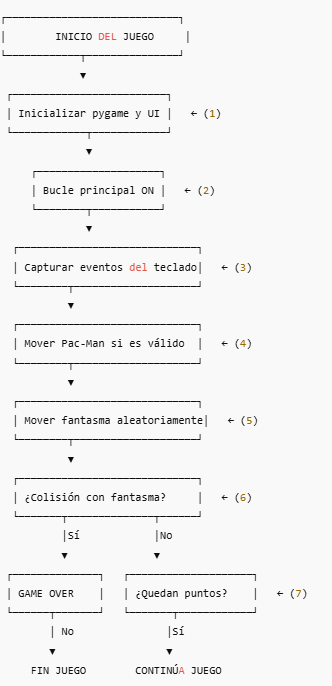
**Inicio del juego:** Se prepara el mapa, se cargan variables, personajes, y se inicializa Pygame.

**Bucle principal:** Mientras el juego esté corriendo, se repiten las siguientes acciones.

**Captura eventos:** Detecta si el usuario presiona teclas para mover a Pac-Man o si cierra la ventana.

**Lógica del juego:** Aquí se decide si se puede mover, si come un punto, si choca con el fantasma, etc.

**Actualizar pantalla:** Redibuja el juego para mostrar los cambios (posiciones, puntos, etc.).

**Diagrama de flujo**

**Explicación paso a paso:**

**Inicialización:** Se prepara la pantalla, el mapa, las variables, y los personajes.

**Bucle principal:** Inicia la repetición infinita del juego hasta que ocurra un final.

**Captura de eventos:** Detecta las teclas para mover a Pac-Man.

**Movimiento del jugador:** Se mueve solo si no hay una pared.

**Movimiento del fantasma:** De forma aleatoria, sin atravesar paredes.

**Verifica colisión con fantasma**:

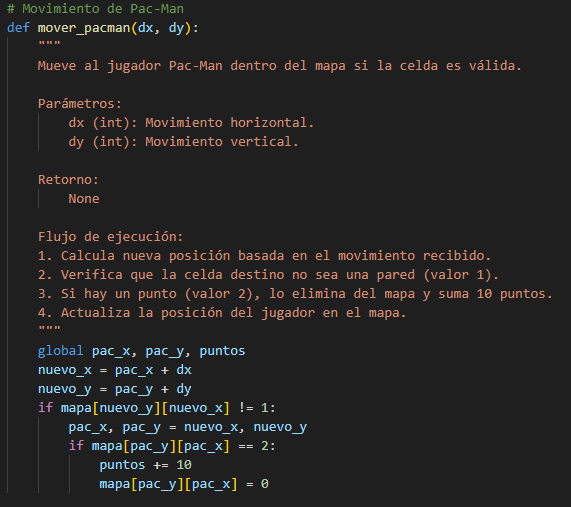
* Si colisionan: termina el juego con mensaje “Game Over”.
* Si no: sigue al paso 7.

**¿Quedan puntos en el mapa?**:

- Si no hay más puntos (2), muestra “Ganador”.

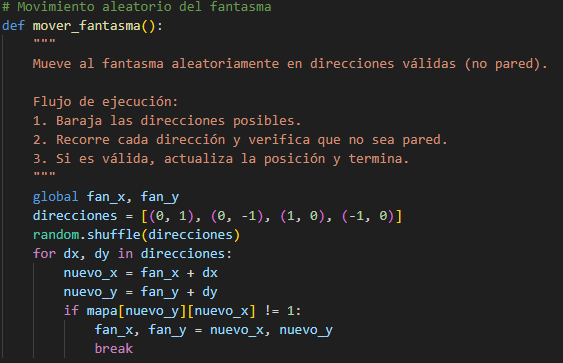
- Si todavía quedan: el juego continúa.

**Explicación detallada del código**

****

**Explicación:** La función mover\_pacman(dx, dy) permite mover al personaje principal del juego (Pac-Man) por el mapa, siempre que no se trate de una pared.

Primero calcula la nueva posición sumando dx y dy a la posición actual. Luego revisa si esa nueva celda es distinta de una pared. Si la celda contiene un punto (valor 2), lo convierte en 0 y suma 10 puntos al marcador. Finalmente, actualiza la posición del personaje.



**Explicación:** La función mover\_fantasma() define cómo se mueve el fantasma dentro del juego. Utiliza una lista con las 4 direcciones posibles (arriba, abajo, izquierda y derecha) y las ordena aleatoriamente usando random.shuffle().

Luego, verifica cada dirección una por una. Si encuentra una celda que no sea pared, el fantasma se mueve allí. Si todas las celdas son paredes, no se mueve.

|  |
| --- |
|  |

**Explicación:** La función preguntar\_volver\_a\_jugar() aparece justo después de que el juego termine. Utiliza la librería tkinter para crear una ventana emergente con una pregunta personalizada (como “¡Ganador! ¿Quieres jugar otra vez?”).

Si el jugador elige “Sí”, la función llama a reiniciar\_juego() y vuelve a empezar. Si elige “No”, cierra el juego usando pygame.quit() y sys.exit().

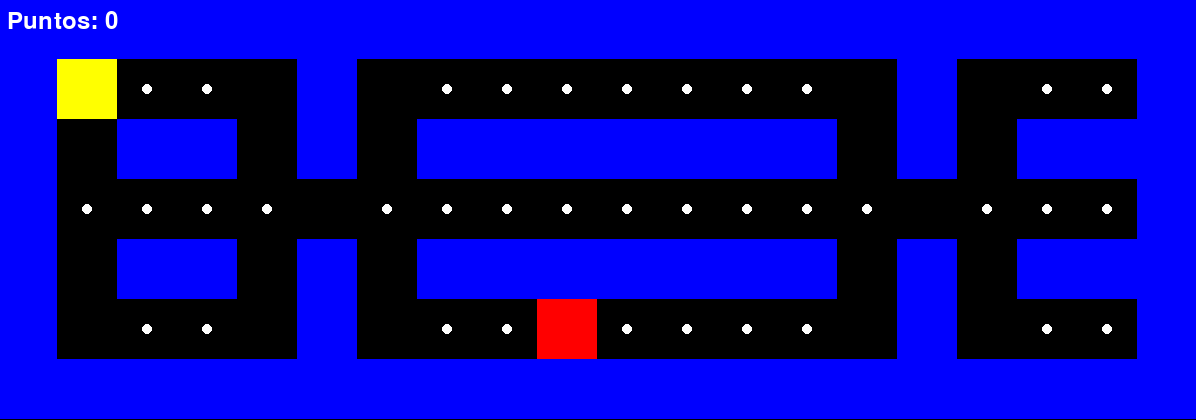
**Casos de Prueba**

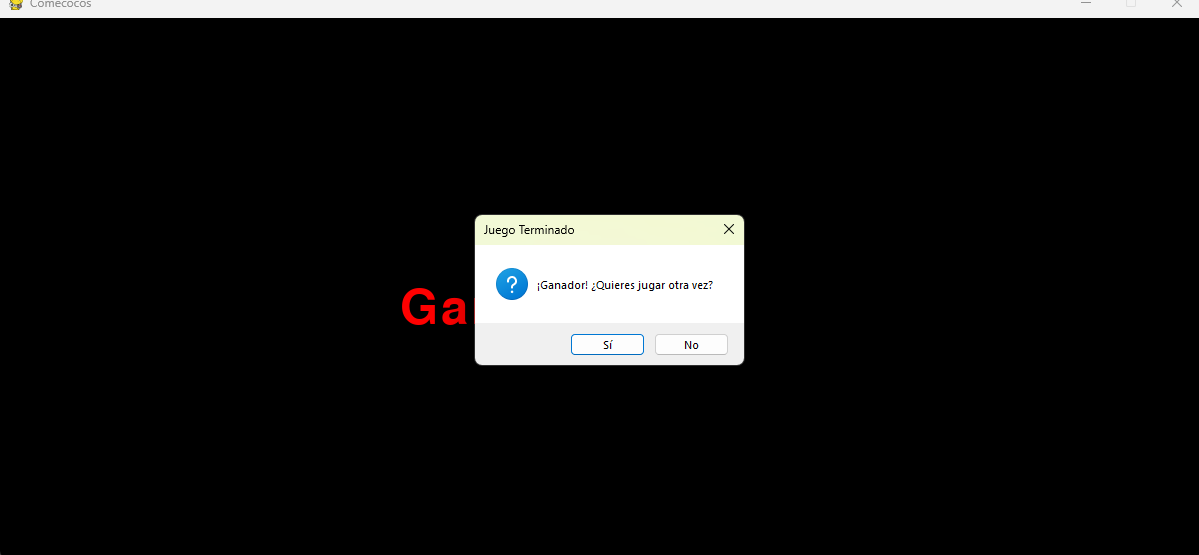
|  |  |  |  |
| --- | --- | --- | --- |
| N.º | **Caso de prueba** | **Entrada / Condición** | **Resultado esperado** |
| 1 | Movimiento válido | Presionar flecha derecha estando frente a una celda libre | Pac-Man se mueve una celda a la derecha |
| 2 | Movimiento contra pared | Presionar flecha hacia una pared (valor 1 en el mapa) | Pac-Man no se mueve, su posición no cambia |
| 3 | Comer punto | Mover a celda con valor 2 | Se suma 10 puntos, el punto desaparece (se convierte en 0) |
| 4 | Colisión con fantasma | Pac-Man y fantasma ocupan la misma celda | Se muestra “Game Over” y se pregunta si quiere volver a jugar |
| 5 | Victoria por recoger todos los puntos | El mapa ya no tiene valores 2 | Se muestra “Ganador” y se pregunta si quiere volver a jugar |
| 6 | Cerrar ventana | El jugador cierra la ventana del juego | El juego se termina y la ventana se cierra |
| 7 | Reiniciar juego | Se presiona “Sí” en el cuadro de mensaje final | El juego se reinicia con puntos restaurados |
| 8 | Rechazar reinicio | Se presiona “No” en el cuadro de mensaje final | El juego se cierra completamente |

**Registro de errores detectados y soluciones aplicadas**

|  |  |  |
| --- | --- | --- |
| **Error detectado** | **Causa / Comportamiento observado** | **Solución aplicada** |
| El fantasma no se movía o quedaba atascado | Solo intentaba moverse en una dirección fija que estaba bloqueada por una pared | Se usó random.shuffle() para barajar las direcciones y probar aleatoriamente |
| Pac-Man atravesaba paredes | No se verificaba correctamente si la nueva celda era una pared | Se agregó una condición if mapa[nuevo\_y][nuevo\_x] != 1: antes de moverlo |
| El juego no terminaba al recolectar todos los puntos | No había ninguna verificación de si quedaban puntos en el mapa | Se añadió if not any(2 in fila for fila in mapa): mostrar\_game\_over("Ganador") |
| El mensaje final siempre decía “Game Over” | mostrar\_game\_over() no aceptaba mensajes personalizados | Se modificó la función para aceptar un argumento mensaje\_final |
| La ventana emergente de “¿Deseas volver a jugar?” bloqueaba o generaba errores | El uso de tkinter junto con pygame podía causar conflictos de ventana o interacción | Se ocultó la ventana principal con root.withdraw() para evitar interferencias |
| Puntuación no se reiniciaba correctamente al volver a jugar | Las variables no se reiniciaban al presionar “Sí” | Se creó la función reiniciar\_juego() que restablece posiciones, puntaje y puntos |

**Evidencia visual**

****

****

****

**Reflexión y Mejoras Propuestas**

**Dificultades encontradas**

**Coordinación del movimiento del fantasma:** Hacer que se mueva sin quedarse atascado ni atravesar paredes fue un reto que implicó lógica condicional y aleatoriedad.

**Manejo de la matriz del mapa:** Evitar errores de índice y actualizar correctamente las celdas al recoger puntos requería atención constante.

**Integración de tkinter con pygame:** Mostrar ventanas emergentes sin que interfieran con la ventana principal fue complejo.

**Lógica de fin de juego por victoria:** Inicialmente, solo existía el Game Over al perder, pero se necesitó desarrollar una verificación para detectar cuando se ganaba.

**Aspectos que se mejorarían con más tiempo o experiencia**

**Diseño de niveles múltiples:** Poder cargar diferentes mapas con obstáculos únicos.

**Efectos de sonido y música:** Para ambientar mejor el juego y hacerlo más divertido.

**Animaciones visuales:** Agregar movimiento al personaje o efectos al recoger puntos.

**Interfaz gráfica de inicio y menú:** Con botones para comenzar, ver instrucciones y salir del juego.

**Mejor organización del código:** Separar aún más las funciones y crear módulos reutilizables.

**Propuestas de mejora futura**

**Más inteligencia para los fantasmas:** Que persigan a Pac-Man o tengan trayectorias específicas.

**Sistema de vidas:** Que Pac-Man tenga 3 intentos antes de perder definitivamente.

**Power-ups:** Elementos especiales que den velocidad, invulnerabilidad o detengan al fantasma.

**Modo multijugador local:** Para jugar por turnos o competir por el mejor puntaje.

**Modo contrarreloj:** Superar el nivel antes de que se acabe el tiempo.