Guía de exposición – CodeRunner (enfoque acotado)

Esta guía resume solo las clases y funciones indicadas por ti, con puntos clave para explicar sin mostrar código.

/game/interfaz.py

class MenuPrincipal

- Menú con 4 opciones; devuelve 1–4 según botón presionado.
- Se redibuja y maneja eventos a 60 FPS; es usado por `juego.iniciar()`.

class PantallaIniciarJuego

- Captura el nombre del jugador con un input; valida que no esté vacío.
- Devuelve el nombre o None si se cancela; flujo simple de "Continuar / Volver".

/game/juego.py

def init() (juego)

- Prepara reloj, estado y servicios (sonido, salón de la fama).
- Mantiene referencias a jugador, computadora y laberinto (asignadas al jugar).

def iniciar()

- Inicializa Pygame y la ventana; muestra `MenuPrincipal` en loop.
- Si se elige "Jugar", instancia `PantallaJuego(nombre)` y ejecuta.
- Si se elige "Salir", rompe el loop y cierra ordenadamente.

def terminar()

- Cierra Pygame y realiza limpieza básica (fin de aplicación).

/game/pantalla_juego.py

def init() (pantalla_juego)

- Calcula `tam_celda` óptimo y offsets (`offset_x`, `offset_y`) para centrar.
- Construye muros de colisión y crea `Jugador` y `Computadora` en celdas de spawn.
- Configura: movimiento por celdas con cooldown (8 frames), timers de obsequios (10 s) y dificultad progresiva (velocidad del enemigo).

def ejecutar()

- Game loop a 60 FPS: manejar eventos → `_actualizar()` → `_renderizar()`.
- Controles: WASD/Flechas (mover), P (pausa), ESC (salir menú).

def __actualizar()

- Orden: 1) mover jugador, 2) perseguir con BFS, 3) timers de obsequios, 4) recolección, 5) captura, 6) aumentar dificultad, 7) tiempo.
- Si `game_over`, espera 5 s antes de permitir salir.

def __mover__jugador__por__celdas(direccion)

- Mueve exactamente una celda; usa `temp_rect` para prevalidar colisiones y límites.
- Evita atravesar paredes y respetar el área centrada del laberinto.

def __verificar__captura()

- Calcula distancia centro a centro; si es menor al umbral, el jugador pierde vida.
- Si no quedan vidas: activa `game_over`; si quedan, respawn en posiciones iniciales y limpia el camino BFS.

def __renderizar()

- Orden de dibujo: fondo → laberinto → obsequios → personajes → HUD → overlays (pausa/game over) → `display.flip()`.

/models/jugador.py

def init() (jugador)

- 3 vidas, 0 puntos, velocidad 4; hitbox rectangular ajustada a círculo (radio $\times$ 1.8).

def mover()

- Modo legacy; el movimiento actual vive en `pantalla_juego.py` (por celdas o píxeles).

def sumar__puntos()

- Incrementa el puntaje al recolectar un obsequio.

def perder__vida()

- Resta una vida al ser atrapado.

def esta_vivo()

- Retorna True/False según vidas restantes.

def dibujar_jugador_principal()

- Dibuja el jugador como círculo rojo centrado en su hitbox.

/models/computadora.py

def init() (computadora)

- Define hitbox ajustada, color y estado de IA: `_bfs_camino`, `_bfs_target_cell`, `_bfs_recalc_cooldown`.

def perseguir_bfs()

- 1) Convertir posiciones a celdas; 2) decidir si recalcular camino; 3) mover hacia la siguiente celda; 4) avanzar cuando llega al centro.

def __calcular_camino_bfs()

- BFS sobre la grilla (4 vecinos). Devuelve lista de celdas del camino más corto o None.

def __cell_from_pos

- Convierte píxeles a celda (fila, columna), considerando offsets y `tam_celda`.

def __pos_center_of_cell

- Convierte (fila, columna) a coordenadas de píxeles del centro de esa celda (con offsets).

def dibujar_computadora_principal()

- Dibuja con efectos de pulsación, color variable y detalles (borde y "ojos").

BFS (Breadth-First Search)

- Ventaja: encuentra siempre el camino más corto.
- Cómo funciona: explora capa por capa desde el inicio.
- Complejidad: O(N) donde N es el número de celdas.
- Resultado: lista de celdas desde el enemigo hasta el jugador.

Cooldown de recálculo

- Recalcula cada 6 frames (~0.1 s a 60 FPS) para optimizar.
- Si el jugador cambia de celda, recalcula inmediatamente.

Movimiento suave

- No “teletransporta”: avanza píxel a píxel hacia el centro de la siguiente celda.
- Al llegar al centro, consume la celda actual y continúa.

/models/laberinto.py

def init() (laberinto)

- Inicializa estructuras: `_muros`, `_pasillos`, `_obsequios`, `laberinto`; carga metadatos y posiciones iniciales.

def cargar_desde_archivo()

- Pasos: 1) resolver ruta en `/data`, 2) leer JSON, 3) validar, 4) cargar datos, 5) procesar mapa, 6) cargar obsequios.

def procesar_laberinto()

- Clasifica cada celda como muro o pasillo para validaciones rápidas (en código: `_procesar_laberinto`).

def validar_estructura()

- Chequea que exista “mapa”, que sea lista no vacía y que todas las filas tengan el mismo ancho.

def recolectar_obsequio()

- Si hay obsequio en la celda, lo elimina y retorna su valor; si no, retorna 0.

def dibujar_obsequio()

- Renderiza los obsequios con efecto de pulsación y varias capas (en código: `dibujar_obsequios`).