Documentación del Juego "Comecocos"

1. Introducción

"Comecocos" es un juego inspirado en el clásico Pac-Man, desarrollado en Java utilizando las bibliotecas Swing y AWT para la interfaz gráfica y la gestión de eventos.

2. Arquitectura General

El juego se estructura en cinco clases principales, cada una con responsabilidades específicas:

- **Main.java**: Configura la ventana principal del juego y lanza la aplicación.
- **Mapa.java**: Define la estructura del mapa, que incluye paredes y cocos, y proporciona métodos para la interacción con dichos elementos.
- **Game.java**: Controla la lógica central del juego, el renderizado, la gestión de eventos (teclado y temporización) y la verificación de colisiones y condiciones de victoria.
- **Comecocos.java**: Representa al personaje principal, gestionando su posición, movimiento, puntaje y representación gráfica.
- **Fantasma.java**: Representa a los enemigos del juego, implementando movimiento aleatorio y dibujo en pantalla.

3. Descripción de Clases

3.1. Clase Comecocos

Responsabilidad:
Gestionar la lógica del personaje principal, incluyendo:

- La posición en el mapa.
- El movimiento basado en entradas del usuario.
- La interacción con el mapa (evitar paredes y consumir cocos).
- El manejo del puntaje.

Atributos:

- private int x, y;
  Coordenadas actuales del personaje en el mapa.
- private int puntos;
  Puntaje acumulado al consumir cocos.
- private Mapa mapa;
  Referencia al objeto Mapa para interactuar con los elementos del juego.

Métodos Principales:

- **Constructor Comecocos(int startX, int startY, Mapa mapa)**
  Inicializa la posición inicial, asigna el mapa y establece el puntaje en cero.
- **mover(int dx, int dy)**
  Calcula la nueva posición según el desplazamiento indicado; si la posición no es una pared, actualiza la posición y, en caso de haber un coco, lo consume y suma 10 puntos.
- **dibujar(Graphics g)**
  Dibuja el personaje en forma de óvalo amarillo, centrado en la celda correspondiente del mapa.
- **Métodos Getters (getX(), getY(), getPuntos())**
  Permiten obtener la posición actual y el puntaje del personaje.

3.2. Clase Fantasma

Responsabilidad:
Representar a un enemigo del juego, gestionando:

- Su posición en el mapa.
- El movimiento aleatorio sin atravesar paredes.
- La representación gráfica en la pantalla.

Atributos:

- private int x, y;
  Posición actual del fantasma.
- private Mapa mapa;
  Referencia para validar movimientos.
- private Random rand;
  Objeto para generar desplazamientos aleatorios.

Métodos Principales:

- **Constructor Fantasma(int startX, int startY, Mapa mapa)**
  Establece la posición inicial y guarda la referencia al mapa.
- **mover()**
  Selecciona aleatoriamente un desplazamiento en X e Y (puede ser 0, 1 o -1) y actualiza la posición si la celda destino no es una pared.
- **dibujar(Graphics g)**
  Dibuja al fantasma como un óvalo rojo, centrado en la celda correspondiente.
- **Métodos Getters (getX(), getY())**
  Retornan la posición actual del fantasma.

3.3. Clase Mapa

Responsabilidad:
Definir y gestionar la estructura del mapa del juego, que incluye paredes y cocos, y proveer métodos para:

- Verificar colisiones (p.ej., si una celda es una pared).
- Consumir cocos.
- Dibujar el mapa completo.

Constantes:

- MAP_WIDTH y MAP_HEIGHT
  Definen el número de columnas y filas del mapa.
- WINDOW_WIDTH y WINDOW_HEIGHT
  Dimensiones deseadas de la ventana del juego.
- CELL_SIZE
  Tamaño calculado de cada celda, basado en las dimensiones de la ventana y el mapa.
- MAPA_INICIAL
  Matriz que representa el diseño inicial del mapa:
  - 0 representa un espacio vacío.
  - 1 representa una pared.
  - 2 representa un coco.

Atributos:

- private int[][] mapa;
  Matriz que almacena el estado actual del mapa.

Métodos Principales:

- **Constructor Mapa()**
  Inicializa el mapa copiando el diseño de MAPA_INICIAL.
- **esMuro(int x, int y)**
  Verifica si la celda en la posición (x, y) contiene una pared.
- **comerCoco(int x, int y)**
  Si la celda contiene un coco, lo elimina (cambiándolo a 0) y retorna true; en caso contrario, retorna false.

- **dibujar(Graphics g)**
  Dibuja las paredes en azul y los cocos en gris claro, utilizando la celda y sus márgenes para centrar los dibujos.
- **hayCocosRestantes()**
  Recorre el mapa para determinar si aún existen cocos, condición necesaria para la victoria del juego.

3.4. Clase Game

Responsabilidad:
Controlar la lógica central del juego, que abarca:

- El renderizado de todos los elementos gráficos.
- La gestión de eventos de teclado para mover a Comecocos.
- La actualización de la lógica del juego mediante un timer.
- La verificación de colisiones y condiciones de victoria o derrota.
- La reinicialización del juego en caso de terminar una partida.

Atributos:

- private Comecocos comecocos;
  Instancia del personaje principal.
- private Fantasma fantasma; y private Fantasma fantasma2;
  Instancias de los enemigos.
- private Mapa mapa;
  Instancia del mapa.
- private Timer timer;
  Temporizador que actualiza la lógica del juego cada 100 ms.

Métodos Principales:

- **Constructor Game()**
    - Inicializa el mapa, al personaje y a los fantasmas en posiciones predefinidas.
    - Configura el fondo de la pantalla y el timer.
    - Inicia el timer para comenzar la actualización del juego.
- **paintComponent(Graphics g)**
  Método sobrescrito de JPanel que se encarga de:
    - Limpiar el área de dibujo.
    - Dibujar el mapa, los personajes y los fantasmas.
    - Mostrar el puntaje actual.
- **dibujarPuntos(Graphics g)**
  Dibuja el puntaje obtenido por el jugador en la parte superior del panel.
- **verificarColision()**
  Comprueba si Comecocos ha colisionado con alguno de los fantasmas:
    - Si ocurre la colisión, se detiene el juego, se muestra un mensaje de "Game Over" y se ofrece reiniciar la partida.
- **verificarVictoria()**
  Determina si se han consumido todos los cocos del mapa, lo que indica la victoria del jugador, mostrando un mensaje y ofreciendo reiniciar el juego.
- **reiniciarJuego()**
  Reinicializa el mapa, al personaje y a los enemigos, y reanuda el timer para continuar la partida.
- **actionPerformed(ActionEvent e)**
  Método invocado periódicamente por el timer para:
    - Mover a los fantasmas.
    - Verificar colisiones y la condición de victoria.

    - Solicitar el repintado del panel.

- **Métodos de KeyListener (keyPressed, keyTyped, keyReleased)**
  Gestionan las entradas del usuario:
    - keyPressed(KeyEvent e): Detecta las pulsaciones de las flechas para mover a Comecocos en la dirección correspondiente.

3.5. Clase Main

Responsabilidad:
Servir como punto de entrada del juego, encargándose de configurar la ventana principal y lanzar la aplicación.

Atributos:

- private Game gamePanel;
  Panel que contiene la lógica y el renderizado del juego.

Métodos Principales:

- **Constructor Main()**
    - Configura la ventana (título, tamaño, comportamiento al cerrar, etc.).
    - Agrega el panel de juego (Game) a la ventana.
    - Registra el KeyListener para captar las pulsaciones del usuario.
    - Hace visible la ventana.
- **main(String[] args)**
  Punto de entrada que utiliza SwingUtilities.invokeLater para asegurar que la creación de la interfaz gráfica se realice en el hilo de eventos de Swing.

---

4. Flujo del Juego

1. **Inicio:**
   La aplicación se lanza desde la clase Main, configurando la ventana y agregando el panel del juego (Game).
2. **Ejecución:**
   - El Timer en Game actualiza la lógica del juego cada 100 ms, moviendo a los fantasmas y verificando las condiciones de colisión y victoria.
   - El método paintComponent se encarga de redibujar el mapa y los personajes en cada actualización.
3. **Interacción del Jugador:**
   El jugador controla a Comecocos usando las teclas de flechas, permitiendo desplazarse por el mapa para consumir cocos y evitar a los fantasmas.
4. **Finalización:**
   - Si Comecocos colisiona con un fantasma, se detiene el juego y se muestra un mensaje de "Game Over" con la opción de reiniciar.
   - Si se consumen todos los cocos, se declara la victoria y se ofrece la posibilidad de reiniciar la partida.