Alumnos: Calderón Sergio, Ercoli Juan Martín

TRABAJO PRÁCTICO N°1 – JUEGO DEL PACMAN

En este documento, explicamos la funcionalidad de las clases que no forman parte del paquete "personajes", para una mejor comprensión del diagrama UML y del diagrama de secuencia.

Clase Ventana de Juego

Representa la interfaz gráfica de la aplicación, por ello tiene atributos como alto y ancho.
Al crearse, invoca a **mostrarMenu()** para visualizar 3 opciones: Iniciar Juego, Ver Récords y Salir.

Esta clase tiene acceso a **Historial**, del cual puede recuperar el puntaje más alto *(highscore).*
Para iniciar el juego, se invoca a **lanzarPartida()**, el cual crea una instancia de la clase Partida y procede a mandarle un mensaje con el *highscore* al único método público **transcurrir(int).**

```
int highscore = mRecords.getHighscore();
Jugador jug = mPartida.transcurrir(highscore);      // se retorna cuando termina
this.accionarFinJuego(jug);                         // actualiza los récords
```

Clase Partida – Inicialización y Retorno

Se encarga de llevar a cabo el desarrollo de la partida, por lo que cuenta con una instancia de la clase **Tablero**, **Jugador, Pacman,** un vector de 4 **Fantasmas** y una utilidad de **Temporizador.**

Al crearse la instancia *mPartida*, se realizarán las siguientes acciones de inicialización:

```
tablero = new Tablero();                                    // atributo privado
mPacman = new Pacman(tablero, 3);                           // atributo privado
fantasmas = new Fantasma[4];                                // atributo privado
fantasmas[0] = new Blinky(tablero, mPacman);
fantasmas[1] = new Pinky(tablero, mPacman);
fantasmas[2] = new Inky(tablero, mPacman, fantasmas[0]);
fantasmas[3] = new Clyde(tablero);

tablero.crearPorDefecto();          // se cargan todas las celdas en una matriz
this.reubicarPersonajes();          // resetearPosicion() a Pacman y fantasmas
```

El método **transcurrir(int)** se ejecuta durante toda la partida. Dentro del mismo se llamará, cuando corresponda, a los métodos **iniciar(), moverPacman(), autoMoverFantasmas()…** y recién al terminar (se gane o pierda), se retorna la instancia de Jugador (nombre y puntaje).

De la manera planteada, la clase VentanaDeJuego se independiza de todas estas cuestiones, incluso de escuchar al teclado (para realizar movimientos de Pacman o pausar el juego).

Clase Partida – Acciones automáticas en transcurrir()

Lo primero que se realiza es **iniciar()**, que habilita a PacMan y los fantasmas a moverse. Debido a que los fantasmas son *manejados por la CPU* (atributo definido en Entidad), la clase Partida invoca cada un segundo al método **autoMoverFantasmas()**, que básicamente consiste en enviar un mensaje **moverSegunModo()** a los fantasmas. Ellos se desplazarán una casilla hacia el objetivo o hacia una esquina asignada según estén en modo persecución, dispersión o asustado.

Otro detalle importante a aclarar es que, periódicamente, además del *autoMoverFantasmas(),* también se invoca *verifPosibleColision()*. Este método revisa la posición de PacMan junto con la de los fantasmas; si detecta que PacMan chocó con un fantasma, devuelve el mismo, sino null:

```
Fantasma fant = this.verifPosibleColision();
if (fant != null) {
        boolean comido = fant.intentarComer();          // true si está asustado
        if (comido) jugActual.sumarPuntaje(100);        // puntos por comer
        else {
                boolean sinVidas = mPacman.restarVida();
                …
        }
}
```

Clase Partida – Acciones manuales en transcurrir()

Con acciones manuales nos referimos a cuando el agente (usuario) pulsa una tecla para mover a PacMan. Al detectarse esta solicitud, se invoca a **moverPacman(Direccion)**, donde Direccion es un enumerativo (UP, DOWN, LEFT, RIGHT). Dicho método ejecuta estas dos sentencias:

```
boolean pudoAvanzar = mPacman.intentarMov(dirección);
if (pudoAvanzar) this.accionarSegunNuevaPos();
```

intentarMov() de Pacman utiliza la utilidad *newPositionsHelper* para conocer si puede avanzar a la siguiente celda. En caso afirmativo, **efectúa el movimiento** y devuelve true a la Partida para que sepa si es necesario chequear si hay comida en la nueva posición que se encuentra.

Entonces, si pudo moverse, Partida consulta a Tablero si la nueva celda **tieneComida(x,y)**, **tieneFruta(x,y)**, o bien, **esTunel(x,y)**. Si es comida, se realiza *jugActual.sumarPuntaje(10)* y *tablero.quitarPunto(x,y)*; si es fruta, se suman 30 puntos en vez de 10 y *this.asustarFantasmas().*

En caso de ser un túnel, se enviará el mensaje: *mPacman.pasarPorTunel(Direccion);*

IMPORTANTE: si PacMan pudo moverse, se envía el mensaje *notificarPosPacman()* a todos los fantasmas por igual, aprovechando el polimorfismo, ya que cada uno de ellos (los fantasmas) lo tiene sobreescrito y sabe cómo debe actualizar su esquina objetivo según su estrategia.

Por ejemplo, Inky usa *NewPositionsHelper* para calcular la celda 2 posiciones delante de PacMan, traza un vector usando dicha posición y la de Blinky, lo duplica y luego *cambia su objetivo*.