Informe TP Final Programación Orientada a Objetos

Integrantes:

- *Milagros Cornidez, 61432*
- *Manuel Dizenhaus, 61101*
- *Franco De Simone, 61100*

Para llevar a cabo el Trabajo Práctico, decidimos agregar al juego de Candy Crush los modos **Golden Board** y **Wall Blast**. Para ello, hubo que realizar modificaciones sustanciales al código fuente provisto por la cátedra, así como agregar funcionalidades tanto al backend como al frontend. En principio, agregamos un menú de opciones para que, al ejecutarse el juego, el jugador pueda elegir qué nivel desea jugar. Se tuvo que implementar coloreo de celdas para pintar celdas de dorado o de marrón (para las paredes). También modificamos el ScorePanel, transformándolo en un **ResultsPanel**, para posibilitar que pueda verse tanto el puntaje parcial obtenido por el jugador así como los movimientos y las "celdas especiales" restantes, según el modo de juego en cuestión. Además, se tuvo que modificar la implementación para solucionar un problema que surgía al perder en el juego (permitía seguir jugando tras haberse quedado sin movimientos): para ello, agregamos un cartel con botones para salir del juego (cierra la ventana) o volver a jugar (reinicia la ejecución del programa).

Para implementar los niveles elegidos, tuvimos que modificar y añadir funciones a las clases de backend del código fuente. Primero y principal, para poder presentar las "celdas especiales" restantes se añadió a la clase GameState el int specialCells, con un setter y un getter, y la función **deleteSpecialCells**.

```
//devuelve la cantidad de special cells en el tablero
public Integer getSpecialCells() { return specialCells; }

//decrementa las special cells
public void deleteSpecialCells() { specialCells--; }

//Seteamos la cantidad inicial de special cells
public void setSpecialCells(int specialCells) { this.specialCells = specialCells; }
```

En el caso de Golden Board, la lógica aplicada fue la de pensar a las celdas transparentes como especiales, y a las doradas como normales, de manera que la cantidad inicial de celdas "especiales" sea todo el tablero, y esta cantidad pueda verse decrementada en el contador a medida que se va jugando. Los getters y setters son llamados desde las clases Level2 y Level3, para preparar el juego.

En la clase Grid, se agregó el booleano **isSpecialMove** para poder controlar el nivel Wall Blast, dado que las paredes solo se quitan con explosiones de caramelos especiales. Esto se complementó con la clase abstracta **SpecialMove** que extiende de Move, y que funciona como punto medio entre Move y clases como BombMove, TwoStripedMove, TwoWrappedMove, etc. El booleano isSpecialMove se usa en los métodos clearContent y tryMove para llamar a los métodos setWallBlastFalse y deleteSpecialCells.

```java
public void clearContent(int i, int j) {
    //Si el movimiento que se realizó es especial y la celda a eliminar es wallBlast, se elimina el w
    if (isSpecialMove && g[i][j].isWallBlast()) {
        g[i][j].setWallBlastFalse();
        state.deleteSpecialCells();
    }
    g[i][j].clearContent();
}



public boolean tryMove(int i1, int j1, int i2, int j2) {
    //Si el juego terminó, no se permiten más movimientos
    if (state.gameOver())
        return false;
    //Seteamos el booleano en falso
    isSpecialMove = false;
    Move move = moveMaker.getMove(i1, j1, i2, j2);
    swapContent(i1, j1, i2, j2);
    if (move.isValid()) {
        //Si el movimiento es una instancia de un movimiento especial, seteamos el booleano en true
        if (move instanceof SpecialMove)
            isSpecialMove = true;
        move.removeElements();
```

```java
//Creamos la clase SpecialMove para encapsular a los movimientos especiales
public abstract class SpecialMove extends Move{
    public SpecialMove(Grid grid) { super(grid); }
}
```

Para modelar y manejar las filas doradas de Golden Board y las paredes de Wall Blast, se agregaron a la clase Cell los booleanos **isGolden** e **isWallBlast**, junto con getters y setters, para poder convertir celdas en doradas o paredes y, en el caso de las paredes, revertir el efecto y que vuelvan a ser celdas normales. Además, para WallBlast, se modificaron los métodos expandExplosion y explode, que puedan llamar al método setWallBlastFalse y borrar las paredes.

```java
public void setGolden() { isGolden = true; }

public boolean isGolden() { return isGolden; }

public boolean isWallBlast() { return isWallBlast; }

public void setWallBlast() { isWallBlast = true; }

public void setWallBlastFalse() { isWallBlast = false; }
```

```java
private void expandExplosion(Direction[] explosion) {
    //Si el programa entra a esta función, es que la celda es especial por lo
    //Además, llamamos a deleteSpecialCell para decrementar el contador de Sp
    if (isWallBlast) {
        setWallBlastFalse();
        grid.deleteSpecialCell();
    }
    for(Direction d: explosion) {
        this.around[d.ordinal()].explode(d);
    }
}

private void explode(Direction d) {
    clearContent();
    //Si el programa entra a esta función, es el movimiento fue especial por
    //Además, llamamos a deleteSpecialCell para decrementar el contador de Sp
    if (isWallBlast) {
        setWallBlastFalse();
        grid.deleteSpecialCell();
    }
    if (this.around[d.ordinal()] != null)
        this.around[d.ordinal()].explode(d);
}
```

El código fuente tuvo que modificarse para mejorar su estilo y eficiencia. En muchas clases, como ImageManager, Grid y muchas de las clases que heradaban de Move, había una cantidad innecesaria de ciclos for, que podía reducirse a un solo for que hiciera todo. Esto representaba un problema de eficiencia y un pésimo estilo. Además, el método fillCells fue movido de los niveles a Grid, dado que para todos los niveles habría que preparar un tablero inicial estándar y se repetiría código. Quisimos cambiar la implementación de la clase MoveMaker, por su gran cantidad de líneas que hacían un put en el mapa, pero no encontramos una forma eficiente de manejarlo.

Durante el desarrollo del TP aparecieron algunos obstáculos o problemas a sortear. Hubo que familiarizarse con JavaFX, para poder manejar el frontend de manera satisfactoria. Entender el funcionamiento del programa de base fue un desafío mayor al que se creía; costó entender el diagrama de clases y la naturaleza del programa. Sin embargo, una vez que eso quedó claro, el resto del trabajo se hizo cuesta abajo. Finalmente, hubo que deliberar que nivel queríamos desarrollar; a las distintas opiniones dentro del grupo se le sumaron las numerosas instancias de prueba y error con distintos niveles que, al no creer poder resolverse adecuadamente, fueron descartados. Tuvimos varios frentes abiertos hasta que decidimos quedarnos con Golden Board y Wall Blast.