Trabajo Final POO

# Integrantes

Ignacio Méndez Legajo: 59058

Franco Navarro Legajo: 59055

# Funcionalidades agregadas y modificaciones realizadas

Nivel 2: Fruit Level

Para empezar, agregamos las imágenes de la avellana (Hazelnut) y de la cereza (Cherry) a la clase ImageManager. Luego, creamos los dos elementos como subclases de un nuevo elemento abstracto llamado Fruit. Este último extiende de Element y contiene todas las características que las frutas deben tener. Si más adelante se agregaran otros elementos cuya finalidad también es llevarlos hasta el fondo del tablero, pero no son frutas, simplemente habría que cambiar el nombre de la clase Fruit por otro.

Decidimos crear una clase abstracta llamada Level que extienda a Grid. El objetivo de esta decisión fue para que así sea más clara la herencia que involucra a todos los niveles.

Luego, creamos la clase Level2, que extiende de Level. De esta forma, hereda principalmente el método fillCells. Dentro del nivel 2 nosotros como programadores decidimos en donde van a estar las frutas desde el comienzo del nivel. Se consideró que la posición de las frutas sea aleatoria cada vez que se inicia el nivel, pero luego se descartó la idea ya que de esta manera el nivel podría generarse o muy difícil o muy fácil.

Se agregó el método setSpecialElements, que es justamente el que inserta los elementos especiales (es decir, los que no son caramelos) en sus respectivas posiciones al iniciarse el nivel. Luego, en la inner class Level2State se definió la manera en la que el nivel se da por ganado y por perdido. También agregamos el método getCurrentState, que es el que devuelve el estado en el que se encuentra el nivel en un momento determinado. Es decir, la cantidad de movimientos restantes, o en el caso del nivel 1, el puntaje actual. Cabe mencionar también nuestra decisión de cambiar el nombre de la clase del front end ScorePanel por “LevelDataPanel”. Esto se debe a que el panel inferior del juego ya no muestra solamente un puntaje, ahora puede mostrar otro tipo de información del nivel.

Luego, se tuvieron que agregar todos los tipos de movimientos nuevos a la clase MoveMaker.

Por último, se tuvo que modificar ligeramente la clase Cell. Nos dimos cuenta que esta era la encargada de propagar las explosiones por el tablero, pero una explosión, aunque se haya provocado lejos de nuestra fruta, no debería eliminarla. Por lo tanto, tuvimos que crear algún tipo de excepción a la regla, por lo que aclaramos que, si una explosión se encuentra con nuestras frutas, no la elimina; es decir, no reemplaza a la fruta por Nothing. Esto lo hicimos agregando un método booleano llamado isExplodable a todos los elementos, que es el que determina si un elemento es susceptible a las explosiones. En el caso de las frutas, retorna falso.

Nivel 3: Timer Candy Level

Primero creamos el elemento TimerCandy que extiende de Candy. Funciona parecido a como funciona Candy, pero con el agregado de un contador interno de movimientos llamado “timer” y un método llamado visualRepresentation que se explicará más adelante. Además, tuvimos que agregar la imagen correspondiente de este nuevo elemento, pero fue tan simple como usar las imágenes de los Candy que ya había.

Luego, creamos la clase Level3 que extiende de Level para que obtenga su método fillCells. Al igual que el nivel 2, nosotros decidimos en donde van a estar los TimerCandy al iniciarse el nivel. Se agregó el método obtainMinTimer, que como su nombre lo indica, contempla todos los casos posibles y devuelve el menor timer que se encuentre en el tablero en un determinado momento. Sobrescribimos también el método tryMove, que es el que se encarga de decrementar en 1 todos los contadores del tablero si y solo si el movimiento fue exitoso.

Se agregó el método removeSpecialElements, que se usa cuando un elemento especial (en este caso los TimerCandy) debe ser removido. En Level3 lo que hace es dejar el timer en -1 para visualizar que ese elemento especial ya no tiene uso (es decir, fue eliminado).

La inner class Level3State se encarga de definir la manera en la que el nivel se da por ganado o por perdido. El método getCurrentState devuelve la cantidad de movimientos restantes aprovechando el comportamiento del método obtainMinTimer.

Luego, se agregaron todos los tipos de movimientos nuevos a la clase MoveMaker. Incluso se agregaron los movimientos entre los TimerCandy y la avellana y la cereza. Aunque estos elementos no se crucen en ninguno de los 3 niveles, decidimos agregarlos para futuros niveles en donde podrían cruzarse.

En este nivel también se tuvo que modificar la clase Cell. Lo que sucedía era que si un TimerCandy era alcanzado por una explosión lejana (por ejemplo, una explosión vertical lejana), el nivel no se daba cuenta de esto y no actualizaba que en realidad el caramelo había sido eliminado. Por lo tanto, tuvimos que crear un camino desde Cell hasta el Level3. Primero, tuvimos que avisarle al State que un elemento había sido eliminado. Esto lo hicimos agregándole un método llamado addSpecialElementsRemoved y una variable a la clase GameState llamada removeSpecialElements. Anteriormente habíamos puesto esta variable en el Level3State, pero al encontrarnos con este problema, tuvimos que “subirla” hasta GameState. Luego, en la clase Cell llamamos al método removeSpecialElements, que como explicamos anteriormente, es al que se llama cuando un elemento especial debe ser eliminado. Todas estas funcionalidades se usan en la clase Cell si y solo si se cumple con el método booleano isSpecial que determina si el elemento de la celda en cuestión es especial o no. También debe cumplir que sea explotable, ya que en caso contrario estas funcionalidades se ejecutarían también para las frutas del nivel 2, y eso no es lo deseado.

Cabe aclarar que toda la corrección de este último error serviría también en el nivel de los caramelos que agregan tiempo.

Por último, tuvimos que trabajar con la manera de superponer la imagen de los TimerCandy con sus contadores. Primero, modificamos la clase BoardPanel para que en vez de usar una matriz de instancias de ImageView, use una de instancias de StackPane. Luego, modificamos su método setImage. Si el elemento en cuestión debe tener una representación visual en el tablero, se le superpone aquella representación a la imagen del elemento. Si no, el método hace lo mismo que hacía anteriormente. El método visualRepresentation se tuvo que agregar a la clase Element y se hizo que retorne null por defecto. Luego, los elementos que tienen alguna información para mostrar en el tablero lo reescribirán, como es el caso de los TimerCandy. Otra vez, estas modificaciones hubieran servido también para el nivel de los caramelos que agregan tiempo.

Otras modificaciones y soluciones de errores

* Se agregó al panel inferior del nivel 1 más información que permitirá al jugador entender mejor el estado actual del nivel.
* Se modularizó el método constructor de la clase CandyFrame para que no sea tan extenso y sea más fácil de seguir.
* La clase CandyFrame, en la parte en la que se agrega el MouseEvent, se reemplazó el getScore por el getCurrentState, ya que debe mostrar otro tipo de información además del puntaje.
* El método getCurrentState de la clase GameState se definió para que por defecto retorne un string vacío. Esto se hizo para que cuando el juego se gane o se pierda, solo se muestre el mensaje de “ganaste” o “perdiste” en la pantalla, ya que no nos pareció importante indicar en el momento de la derrota o la victoria el estado actual del nivel.
* Se modificó el método tryMove de la clase Grid para que no se puedan hacer más movimientos una vez que el juego haya terminado (independientemente de si se ganó o se perdió). La razón principal de esto es porque se podía llegar a ganar un nivel luego de haberlo perdido (por ejemplo, en el nivel 1 si se perdía porque se terminaron los movimientos, era posible ganar si luego de esto se superaba el puntaje máximo).
* Debido a que agregamos las funcionalidades de addSpecialElementsRemoved y getSpecialElementsRemoved al hacer el nivel 3, decidimos también implementarlas para el nivel 2. Es decir, nos deshicimos de una variable local a Level2State que se llamaba “fruitsRemoved”.
* En la clase BombStrippedMove agregamos que, justo antes de reemplazar los Candy seleccionados con instancias de caramelos rayados usando el método setContent, se eliminasen los elementos. Esto lo tuvimos que hacer ya que, si solamente se pisaban los elementos anteriores, y si justo uno de ellos era un TimerCandy, no se llamaba a clearContent de Cell, y por lo tanto no se llamaba a los métodos addSpecialElementsRemoved y removeSpecialElements. Esto provocaba que el nivel no se daba cuenta que un TimerCandy había sido eliminado.
* Analizando la corrección descrita anteriormente, nos dimos cuenta que debíamos tener cuidado cada vez que en alguna parte del juego se hace un setContent, ya que no queremos que se aplique este método a un TimerCandy antes de aplicar los métodos addSpecialElementsRemoved y removeSpecialElements. Por lo tanto, al método removeFigure de la clase Grid le agregamos una validación para evitar que esto suceda.
* En el nivel 3 nos encontramos con un error importante: luego de llenar el tablero y setear los TimerCandy en sus posiciones, a veces ocurría que, por ejemplo, un TimerCandy rojo se seteaba en el medio de otros dos Candy rojos. Visualmente se veía como al iniciar el nivel había 3 caramelos del mismo color uno al lado del otro. Por detrás, lo que sucedía era que uno de esos 3 caramelos se había convertido en Nothing en algún momento del arranque del nivel. Cuando intentábamos mover alguno de estos 3 caramelos, nos aparecía una excepción que nos llevaba a la clase FigureDetector, a su método removeFigure. Nos decía que no se puede castear una instancia de Nothing a Candy. Para arreglarlo, decidimos añadir una verificación a este método para que así no haya errores de casteo.

Sin embargo, esto solamente solucionaba la excepción, pero visualmente los 3 caramelos seguían estando uno al lado del otro. En este momento fue cuando encontramos dos posibles soluciones al problema:

1. Podíamos setear, además de los TimerCandy, los 4 caramelos adyacentes a estos. De esta forma, si había un TimerCandy rojo, lo rodearíamos de 4 Candy de distinto color. Esto reduce dramáticamente las posibilidades de que aparezcan dos caramelos del mismo color del TimerCandy junto a él.
2. Sin embargo, la solución anterior nos pareció de muy mala programación, ya que estamos modificando la aleatoriedad a nuestro favor y eso arruina la esencia del juego Candy Crush. Por lo tanto, decidimos que está bien que este caso particular suceda. Es decir, aceptamos la posibilidad que, al iniciar el nivel, que debería tener 3 TimerCandy, se inicie con menos.

Al elegir la segunda opción, tuvimos que agregar al método setSpecialElements de la clase Level3 código que se encarga de eliminar el TimerCandy en caso de que forme una figura al iniciarse el nivel.

Cabe mencionar que no deshicimos el cambio que le habíamos hecho a la clase de FigureDetector ya que creemos que en futuros niveles puede llegar a aparecer la misma excepción que arreglamos.

* Debido a que decidimos dejar la posibilidad de que el nivel 3 se inicie con menos caramelos, queremos aclarar lo que sucede en el caso en que todos los TimerCandy sean eliminados al principio. Debido a como está hecho el juego, particularmente la clase CandyFrame, el juego “no se empieza a preguntar si el nivel esta ganado o perdido” hasta que se hagan 2 clicks sobre el tablero. Por lo tanto, aclaramos que, si los 3 TimerCandy son eliminados al comienzo del nivel, no se verá de inmediato que el nivel está ganado; solo basta con hacer 2 clicks en cualquier lugar del tablero y ya el nivel se dará cuenta que está ganado.

# Menú principal: selector de niveles

Lo primero que hicimos fue crear el método startLevel en el cual pusimos el código que originalmente se encontraba en el método start. Esto lo hicimos porque en el código original el método start estaba hecho para que se ejecute el nivel 1 únicamente. Entonces nosotros creamos este método para generalizar el código y que ejecute el nivel que se le pasa como parámetro.

Luego, en el método start diseñamos un menú con botones para seleccionar que nivel desea jugar el usuario. Para eso creamos tres botones cuyas acciones son las de ejecutar un determinado nivel usando el startLevel. Tras setear los tres botones creamos el layout usando un modelo Vbox, al que le agregamos todos los botones. Por último, creamos la scene usando el layout que diseñamos.