Paradigmas Avanzados de Programación

Candy Crush CUDA

Grado en Ingeniería Informática
Universidad de Alcalá

Alexander Brun Guajardo
Pablo García García

31 de marzo de 2023

Índice general

Capítulo 1

Implementación básica

Durante este primer capítulo trataremos de explicar el funcionamiento de nuestro juego. El objetivo será imitar al famoso juego *Candy Crush*, pero implementando el paradigma paralelo con las herramientas que *nvidia* nos ofrece.

1.1. Funcionamiento del juego

Primero veamos como es el juego. Para lanzarlo abriremos una consola en la carpeta del proyecto, y escribimos `CandyCrush.exe -m/-a 1/2 N M`, donde podemos elegir jugar nosotros en modo manual con el parámetro `-m`, o que la máquina juegue sola de forma automática (`-a`). Con el primer valor numérico, podemos escribir `1` o `2` para elegir la dificultad del juego, teniendo en el primer caso cuatro tipos de caramelos diferentes, y seis en el segundo. Por último, mediante los parámetros `-n` y `-m`, elegiremos las dimensiones de nuestro tablero de juego. Veamos un ejemplo para `CandyCrush.exe -m 1 10 15`.

```
TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  1  1  3  2  1  4  2  3  4  1  1  3  2  2
1|  2  1  2  1  2  4  2  2  4  1  1  3  4  3  1
2|  1  4  1  2  2  4  1  3  1  4  1  3  1  1  3
3|  3  2  3  1  2  4  1  4  4  2  2  4  4  2  1
4|  1  4  1  3  4  2  1  3  1  4  1  1  4  3  1
5|  3  1  1  3  1  4  4  1  3  1  2  4  4  1  2
6|  4  1  1  3  1  3  2  4  2  1  2  4  3  1  4
7|  3  4  1  2  4  4  1  3  4  4  3  3  2  1  2
8|  3  1  3  3  4  4  2  4  4  4  4  2  2  3  1
9|  2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Selecciona fila y columna de la casilla a eliminar:
```

Figura 1.1: Ejecución de ejemplo

En esta situación podemos escribir dos números separados por un espacio, que harían referencia a la posición del caramelo que queremos "tocar". Hagamos antes un resumen del funcionamiento del juego.

1. Al comenzar una partida, disponemos de **cinco vidas**.
2. Introduciremos la fila y columna del caramelo a eliminar, pero este solo se eliminará con sus contiguos en caso de que los haya.
3. Si no hay contiguos del mismo tipo, no se borrará nada y perderemos una vida.
4. Romper cinco caramelos nos otorgará un objeto especial, **bomba**. Al romper esta, se romperá toda la fila o columna (de forma aleatoria).
5. Romper seis caramelos nos otorgará un objeto especial, **TNT**. Al romper este, se romperán todos los caramelos a distancia 4 de este.
6. Romper más de seis caramelos nos otorgará un objeto especial, **rompecabezas**. Al romper este, se romperán todos los caramelos de un cierto tipo (de forma aleatoria).

Siguiendo con el ejemplo anterior, veamos qué sucede al romper el caramelo en la posición (8, 5). Como tiene más caramelos del mismo tipo alrededor, se romperán este y sus contiguos. Deberán "caer" todos los caramelos, y en las posiciones donde quedarían vacías, se meten nuevos caramelos aleatorios.

```
Selecciona fila y columna de la casilla a eliminar: 8 5

Caramelos eliminados: 4

TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  1  1  3  2  1  4  2  3  4  1  1  3  2  2
1|  2  1  2  1  2  4  2  2  4  1  1  3  4  3  1
2|  1  4  1  2  2  4  1  3  1  4  1  3  1  1  3
3|  3  2  3  1  2  4  1  4  4  2  2  4  4  2  1
4|  1  4  1  3  4  2  1  3  1  4  1  1  4  3  1
5|  3  1  1  3  1  4  4  1  3  1  2  4  4  1  2
6|  4  1  1  3  1  3  2  4  2  1  2  4  3  1  4
7|  3  4  1  2  X  X  1  3  4  4  3  3  2  1  2
8|  3  1  3  3  X  X  2  4  4  4  4  2  2  3  1
9|  2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Vidas: 5

TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  1  1  3  3  1  4  2  3  4  1  1  3  2  2
1|  2  1  2  1  2  4  2  2  4  1  1  3  4  3  1
2|  1  4  1  2  2  1  1  3  1  4  1  3  1  1  3
3|  3  2  3  1  2  4  1  4  4  2  2  4  4  2  1
4|  1  4  1  3  2  4  1  3  1  4  1  1  4  3  1
5|  3  1  1  3  2  4  4  1  3  1  2  4  4  1  2
6|  4  1  1  3  4  2  2  4  2  1  2  4  3  1  4
7|  3  4  1  2  1  4  1  3  4  4  3  3  2  1  2
8|  3  1  3  3  1  3  2  4  4  4  4  2  2  3  1
9|  2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Selecciona fila y columna de la casilla a eliminar:
```

Figura 1.2: Romper caramelos

De la misma manera, si seleccionamos una casilla sin similares contiguos, perderemos una vida:

```
Selecciona fila y columna de la casilla a eliminar: 1 13

Caramelos eliminados: 0

No hay suficientes caramelos juntos, pierdes una vida!

TABLERO:
     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
     -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|   2  1  1  3  3  1  4  2  3  4  1  1  3  2  2
1|   2  1  2  1  2  4  2  2  4  1  1  3  4  3  1
2|   1  4  1  2  2  1  1  3  1  4  1  3  1  1  3
3|   3  2  3  1  2  4  1  4  4  2  2  4  4  2  1
4|   1  4  1  3  2  4  1  3  1  4  1  1  4  3  1
5|   3  1  1  3  2  4  4  1  3  1  2  4  4  1  2
6|   4  1  1  3  4  2  2  4  2  1  2  4  3  1  4
7|   3  4  1  2  1  4  1  3  4  4  3  3  2  1  2
8|   3  1  3  3  1  3  2  4  4  4  4  2  2  3  1
9|   2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Vidas: 4

TABLERO:
     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
     -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|   2  1  1  3  3  1  4  2  3  4  1  1  3  2  2
1|   2  1  2  1  2  4  2  2  4  1  1  3  4  3  1
2|   1  4  1  2  2  1  1  3  1  4  1  3  1  1  3
3|   3  2  3  1  2  4  1  4  4  2  2  4  4  2  1
4|   1  4  1  3  2  4  1  3  1  4  1  1  4  3  1
5|   3  1  1  3  2  4  4  1  3  1  2  4  4  1  2
6|   4  1  1  3  4  2  2  4  2  1  2  4  3  1  4
7|   3  4  1  2  1  4  1  3  4  4  3  3  2  1  2
8|   3  1  3  3  1  3  2  4  4  4  4  2  2  3  1
9|   2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Selecciona fila y columna de la casilla a eliminar:
```

Figura 1.3: Perder vida

Veamos ahora ejemplos de ejecución de los diferentes objetos especiales. Comenzamos por la bomba con el ejemplo de la Figura 1.4. En la posición (8, 10) de este ejemplo vemos 5 cuatros juntos, por lo que podremos borrarlos y generar una bomba. Al explotarla, esta ha elegido borrar toda la fila en la que se encuentra.

Siguiendo con el TNT, vemos en el ejemplo de la Figura 1.5, que en la posición (4, 4) vemos seis 2 contiguos, por lo que borramos todos y ponemos en esta posición el TNT, y dejamos caer los caramelos. Al seleccionar el TNT se borran todos los elementos en un radio de 4 elementos a este.

Como último ejemplo de objeto especial, veamos el rompecabezas. En el ejemplo de la Figura 1.6, vemos que en la posición (8, 11) tenemos más de seis 2, por lo que se genera un rompecabezas. Al romper este, elige de forma aleatoria un tipo de caramelo y borra todos estos del tablero, en este caso ha sido el tipo 4.

```
TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  2  4  3  2  3  1  3  3  4  2  1  3  2  2
1|  2  4  3  2  3  1  2  1  4  1  3  3  4  3  1
2|  1  2  2  2  4  2  1  3  1  4  1  3  1  1  3
3|  3  4  1  3  1  1  4  2  4  2  2  4  4  2  1
4|  1  2  3  1  1  2  3  3  1  4  1  1  4  3  1
5|  3  1  3  3  2  1  2  4  3  1  2  4  4  1  2
6|  4  4  4  3  2  4  3  1  2  1  2  4  3  1  4
7|  3  1  1  3  2  1  4  2  4  4  3  3  2  1  2
8|  3  1  2  3  4  4  2  2  4  4  4  2  2  3  1
9|  2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Selecciona fila y columna de la casilla a eliminar: 8 10

Caramelos eliminados: 5

TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  2  4  3  2  3  1  3  3  4  2  1  3  2  2
1|  2  4  3  2  3  1  2  1  4  1  3  3  4  3  1
2|  1  2  2  2  4  2  1  3  1  4  1  3  1  1  3
3|  3  4  1  3  1  1  4  2  4  2  2  4  4  2  1
4|  1  2  3  1  1  2  3  3  1  4  1  1  4  3  1
5|  3  1  3  3  2  1  2  4  3  1  2  4  4  1  2
6|  4  4  4  3  2  4  3  1  2  1  2  4  3  1  4
7|  3  1  1  3  2  1  4  2  X  X  3  3  2  1  2
8|  3  1  2  3  4  4  2  2  X  X  B  2  2  3  1
9|  2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Vidas: 4

TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  2  4  3  2  3  1  3  4  4  2  1  3  2  2
1|  2  4  3  2  3  1  2  1  4  4  3  3  4  3  1
2|  1  2  2  2  4  2  1  3  3  4  1  3  1  1  3
3|  3  4  1  3  1  1  4  2  4  1  2  4  4  2  1
4|  1  2  3  1  1  2  3  3  1  4  1  1  4  3  1
5|  3  1  3  3  2  1  2  4  4  2  2  4  4  1  2
6|  4  4  4  3  2  4  3  1  1  4  2  4  3  1  4
7|  3  1  1  3  2  1  4  2  3  1  3  3  2  1  2
8|  3  1  2  3  4  4  2  2  2  1  B  2  2  3  1
9|  2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Selecciona fila y columna de la casilla a eliminar: 8 10

                OBJETO especial, se aplica el efecto de la bomba borrar FILA

Caramelos eliminados: 14

TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  2  4  3  2  3  1  3  4  4  2  1  3  2  2
1|  2  4  3  2  3  1  2  1  4  4  3  3  4  3  1
2|  1  2  2  2  4  2  1  3  3  4  1  3  1  1  3
3|  3  4  1  3  1  1  4  2  4  1  2  4  4  2  1
4|  1  2  3  1  1  2  3  3  1  4  1  1  4  3  1
5|  3  1  3  3  2  1  2  4  4  2  2  4  4  1  2
6|  4  4  4  3  2  4  3  1  1  4  2  4  3  1  4
7|  3  1  1  3  2  1  4  2  3  1  3  3  2  1  2
8|  X  X  X  X  X  X  X  X  X  X  X  X  X  X  X
9|  2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Vidas: 4

TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  4  2  1  4  2  4  3  2  1  2  1  1  2  1
1|  2  2  4  3  2  3  1  3  4  4  2  1  3  2  2
2|  2  4  3  2  3  1  2  1  4  4  3  3  4  3  1
3|  1  2  2  2  4  2  1  3  3  4  1  3  1  1  3
4|  3  4  1  3  1  1  4  2  4  1  2  4  4  2  1
5|  1  2  3  1  1  2  3  3  1  4  1  1  4  3  1
6|  3  1  3  3  2  1  2  4  4  2  2  4  4  1  2
7|  4  4  4  3  2  4  3  1  1  4  2  4  3  1  4
8|  3  1  1  3  2  1  4  2  3  1  3  3  2  1  2
9|  2  1  2  4  3  1  4  3  2  1  2  4  2  4  2
```

Figura 1.4: Ejemplo explosión bomba

```
TABLERO:
     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
     -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|   2  1  1  3  3  1  4  2  3  4  1  1  3  2  2
1|   2  1  2  1  2  4  2  2  4  1  1  3  4  3  1
2|   1  4  1  2  2  1  1  3  1  4  1  3  1  1  3
3|   3  2  3  1  2  4  1  4  4  2  2  4  4  2  1
4|   1  4  1  3  2  4  1  3  1  4  1  1  4  3  1
5|   3  1  1  3  2  4  4  1  3  1  2  4  4  1  2
6|   4  1  1  3  4  2  2  4  2  1  2  4  3  1  4
7|   3  4  1  2  1  4  1  3  4  4  3  3  2  1  2
8|   3  1  3  3  1  3  2  4  4  4  4  2  2  3  1
9|   2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Selecciona fila y columna de la casilla a eliminar: 4 4

Caramelos eliminados: 6

TABLERO:
     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
     -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|   2  1  1  3  3  1  4  2  3  4  1  1  3  2  2
1|   2  1  2  1  X  4  2  2  4  1  1  3  4  3  1
2|   1  4  1  X  X  1  1  3  1  4  1  3  1  1  3
3|   3  2  3  1  X  4  1  4  4  2  2  4  4  2  1
4|   1  4  1  3  T  4  1  3  1  4  1  1  4  3  1
5|   3  1  1  3  X  4  4  1  3  1  2  4  4  1  2
6|   4  1  1  3  4  2  2  4  2  1  2  4  3  1  4
7|   3  4  1  2  1  4  1  3  4  4  3  3  2  1  2
8|   3  1  3  3  1  3  2  4  4  4  4  2  2  3  1
9|   2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Vidas: 4

TABLERO:
     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
     -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|   2  1  1  3  2  1  4  2  3  4  1  1  3  2  2
1|   2  1  2  3  4  4  2  2  4  1  1  3  4  3  1
2|   1  4  1  1  1  1  1  3  1  4  1  3  1  1  3
3|   3  2  3  1  3  4  1  4  4  2  2  4  4  2  1
4|   1  4  1  3  3  4  1  3  1  4  1  1  4  3  1
5|   3  1  1  3  T  4  4  1  3  1  2  4  4  1  2
6|   4  1  1  3  4  2  2  4  2  1  2  4  3  1  4
7|   3  4  1  2  1  4  1  3  4  4  3  3  2  1  2
8|   3  1  3  3  1  3  2  4  4  4  4  2  2  3  1
9|   2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Selecciona fila y columna de la casilla a eliminar: 5 4

                OBJETO especial, se aplica el efecto del TNT

Caramelos eliminados: 49

TABLERO:
     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
     -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|   2  1  1  3  2  1  4  2  3  4  1  1  3  2  2
1|   2  1  2  3  4  4  2  2  4  1  1  3  4  3  1
2|   1  X  X  X  X  X  X  X  1  4  1  3  1  1  3
3|   3  X  X  X  X  X  X  X  4  2  2  4  4  2  1
4|   1  X  X  X  X  X  X  X  1  4  1  1  4  3  1
5|   3  X  X  X  X  X  X  X  3  1  2  4  4  1  2
6|   4  X  X  X  X  X  X  X  2  1  2  4  3  1  4
7|   3  X  X  X  X  X  X  X  4  4  3  3  2  1  2
8|   3  X  X  X  X  X  X  X  4  4  4  2  2  3  1
9|   2  1  2  4  3  1  4  3  2  1  2  4  2  4  2

Vidas: 4

TABLERO:
     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
     -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|   2  2  4  3  2  3  1  3  3  4  1  1  3  2  2
1|   2  4  3  2  3  1  2  1  4  1  1  3  4  3  1
2|   1  2  2  2  4  2  1  3  1  4  1  3  1  1  3
3|   3  4  1  3  1  1  4  2  4  2  2  4  4  2  1
4|   1  2  3  1  1  2  3  3  1  4  1  1  4  3  1
5|   3  1  3  3  2  1  2  4  3  1  2  4  4  1  2
6|   4  4  4  3  2  4  3  1  2  1  2  4  3  1  4
7|   3  1  1  3  2  1  4  2  4  4  3  3  2  1  2
8|   3  1  2  3  4  4  2  2  4  4  4  2  2  3  1
9|   2  1  2  4  3  1  4  3  2  1  2  4  2  4  2
```

Figura 1.5: Ejemplo explosión TNT

```
TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  4  2  1  4  2  4  3  2  2  1  4  1  1  4
1|  2  2  4  3  2  3  1  3  4  4  2  4  2  4  3
2|  2  4  3  2  3  1  2  1  4  4  1  3  1  1  2
3|  1  2  2  2  4  2  1  3  3  3  2  3  1  3  1
4|  3  4  1  3  1  1  4  2  4  3  2  3  4  3  1
5|  1  2  3  1  1  2  3  3  1  2  1  2  2  3  4
6|  3  1  3  3  2  1  2  4  4  2  4  3  3  3  3
7|  4  4  4  3  2  4  3  1  1  1  2  1  2  2  1
8|  3  1  1  3  2  1  4  2  3  4  2  2  2  2  2
9|  2  1  2  4  3  1  4  3  2  4  3  1  1  3  1

Selecciona fila y columna de la casilla a eliminar: 8 11

Caramelos eliminados: 8

TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  4  2  1  4  2  4  3  2  2  1  4  1  1  4
1|  2  2  4  3  2  3  1  3  4  4  2  4  2  4  3
2|  2  4  3  2  3  1  2  1  4  4  1  3  1  1  2
3|  1  2  2  2  4  2  1  3  3  3  2  3  1  3  1
4|  3  4  1  3  1  1  4  2  4  3  2  3  4  3  1
5|  1  2  3  1  1  2  3  3  1  2  1  2  2  3  4
6|  3  1  3  3  2  1  2  4  4  2  4  3  3  3  3
7|  4  4  4  3  2  4  3  1  1  1  X  1  X  X  1
8|  3  1  1  3  2  1  4  2  3  4  X  R  X  X  X
9|  2  1  2  4  3  1  4  3  2  4  3  1  1  3  1

Vidas: 4

TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  4  2  1  4  2  4  3  2  2  3  4  2  3  3
1|  2  2  4  3  2  3  1  3  4  4  4  4  1  1  4
2|  2  4  3  2  3  1  2  1  4  4  1  3  1  1  3
3|  1  2  2  2  4  2  1  3  3  3  2  3  2  4  2
4|  3  4  1  3  1  1  4  2  4  3  1  3  1  1  1
5|  1  2  3  1  1  2  3  3  1  2  2  2  1  3  1
6|  3  1  3  3  2  1  2  4  4  2  2  3  4  3  4
7|  4  4  4  3  2  4  3  1  1  1  1  1  2  3  3
8|  3  1  1  3  2  1  4  2  3  4  4  R  3  3  1
9|  2  1  2  4  3  1  4  3  2  4  3  1  1  3  1

Selecciona fila y columna de la casilla a eliminar: 8 11

               OBJETO especial, se aplica el efecto del rompecabezas, borrar caramelos tipo 4

Caramelos eliminados: 33

TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  X  2  1  X  2  X  3  2  2  3  X  2  3  3
1|  2  2  X  3  2  3  1  3  X  X  X  X  1  1  X
2|  2  X  3  2  3  1  2  1  X  X  1  3  1  1  3
3|  1  2  2  2  X  2  1  3  3  3  2  3  2  X  2
4|  3  X  1  3  1  1  X  2  X  3  1  3  1  1  1
5|  1  2  3  1  1  2  3  3  1  2  2  2  1  3  1
6|  3  1  3  3  2  1  2  X  X  2  2  3  X  3  X
7|  X  X  X  3  2  X  3  1  1  1  1  1  2  3  3
8|  3  1  1  3  2  1  X  2  3  X  X  X  3  3  1
9|  2  1  2  X  3  1  X  3  2  X  3  1  1  3  1

Vidas: 4

TABLERO:
    0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
    -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|  2  3  2  3  2  2  1  2  1  1  4  4  1  3  4
1|  2  3  3  1  3  2  1  3  2  1  2  2  2  3  2
2|  2  3  2  3  2  3  4  3  1  4  3  2  1  1  3
3|  2  1  3  2  3  1  2  1  3  1  1  3  1  1  3
4|  1  2  2  2  1  2  1  3  2  2  2  3  2  1  2
5|  3  2  1  3  1  1  2  2  3  3  1  3  1  3  1
6|  1  2  3  1  2  2  1  3  1  3  2  2  1  3  1
7|  3  1  3  3  2  1  3  1  1  2  2  3  2  3  3
8|  3  1  1  3  2  1  2  2  3  2  1  1  3  3  1
9|  2  1  2  3  3  1  3  3  2  1  3  1  1  3  1
```

Figura 1.6: Ejemplo explosión rompecabezas

Como último ejemplo de ejecución, veamos el final de cómo la máquina juega sola, que

para cuando pierde, que suele ser rápido, pues al elegir las posiciones de forma aleatoria, es fácil elegir aquellas que no tienen contiguos iguales.

```
Seleccionada fila 5 y columna 3

Caramelos eliminados: 0

No hay suficientes caramelos juntos, pierdes una vida!

TABLERO:
     0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
     -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
0|   4  4  2  3  4  1  3  1  4  3  3  1  4  3  3
1|   2  1  3  1  3  2  2  2  3  4  2  2  3  4  2
2|   3  4  3  2  3  2  1  3  1  3  1  4  3  1  4
3|   2  T  3  2  3  4  4  3  1  2  2  3  1  3  2
4|   1  2  1  4  1  1  2  3  4  4  1  2  4  1  3
5|   4  3  4  2  3  1  1  3  2  2  3  4  3  4  3
6|   4  3  1  1  1  2  3  1  2  4  1  1  3  3  1
7|   3  2  1  2  2  3  4  4  1  3  1  2  4  3  2
8|   4  2  2  4  1  2  1  3  2  4  4  2  3  1  3
9|   2  1  1  4  2  1  4  1  2  4  3  4  1  4  1

Vidas: 0

FIN DEL JUEGO, TE HAS QUEDADO SIN VIDAS
```

Figura 1.7: Ejemplo fin de partida

1.2. Implementación

1.2.1. Funciones y métodos

Una vez demostrado el correcto funcionamiento del juego, abordaremos cómo funciona el código de la implementación realizada. Primero comentaremos una serie de funciones auxiliares que hemos usado:

- `aleatorio(int a, int b)`: esta función nos devuelve un número aleatorio $\mathcal{R} \in [a, b]$. Se ejecuta en la GPU.
- `pertenece(int* x, int n, int y)`: siendo `x` un vector de longitud `n`, comprueba si $y \in x$. Se puede ejecutar tanto en CPU como en GPU.
- `buscar_camino(char* tablero, int inicio, int fin, int* visitados, int* x, int* camino, int* y)`: implementa un algoritmo de *backtracking* para desde una posición del tablero, encontrar el camino que lleva a otra. Será ejecutado por cada hilo para intentar llegar al elemento tocado. Se devolverá en el puntero el mayor camino encontrado, pero ya veremos que necesitaremos tratar esto antes de dar la solución. Se

ejecuta en la GPU.

La idea principal es mirar si el tipo de caramelo en el que empieza es del mismo tipo del que se quiere acabar (si son distintos no hace falta buscar camino). Después, se consulta si las casillas de encima, debajo, izquierda, y derecha existen. Las comprobaciones siguen así:

- **Encima**: si estamos en la posición i, y tenemos m columnas, entonces debe cumplirse que $i - m > 0$.
- **Debajo**: si estamos en la posición i, y tenemos n filas y m columnas, entonces $i + m < nm$.
- **Izquierda**: si estamos en la posición i, entonces i no puede ser múltiplo de m, por lo que debe cumplirse que $i \pmod m \not\equiv 0$.
- **Derecha**: si estamos en la posición i, entonces si nos desplazamos una posición, no puede ser múltiplo del número de columnas, por lo que debe cumplirse que $i + 1 \pmod m \not\equiv 0$.

En caso de ser del mismo tipo, se vuelve a aplicar el algoritmo de forma recursiva a cada una de estas, marcando todas como visitadas, y aquellas que sí eran del mismo tipo como parte del camino. En un principio se pensó en implementar listas usando `realloc`, pero tuvimos problemas de memoria, por lo que optamos por usar vectores de tamaño $N \times M$, pues si al principio tenemos el conjunto de casillas C, el nuevo conjunto de casillas C' cumplirá que $C' \subseteq C$.

- `cargar_argumentos(int argc, char* argv[])`: configura los parámetros del juego en función de los argumentos pasados por terminal, comprobando además que el tablero solicitado no supere dimensiones con las que el SM de la tarjeta gráfica del usuario puede trabajar. Se ejecuta en la CPU.
- `generar_elemento()`: nos devuelve un número $n \in [1, 6]$ en función de la dificultad del juego, y lo convierte en carácter para rellenar el tablero. Se ejecuta en la CPU.
- `mostrar_tablero(char* tablero, int n, int m)`: muestra el tablero por pantalla. Se ejecuta en la CPU.

1.2.2. Kernels

Aquí comentaremos los diferentes kernels de ejecución que hemos diseñado para poder llevar a cabo el correcto funcionamiento del juego de manera paralela.

1.2.2.1. `encontrar_caminos`

En este kernel se lleva a cabo la búsqueda de caminos para poder eliminar caramelos similares contiguos. Recibe el tablero, el índice de la casilla seleccionada, y un puntero a un

entero donde se contabilizarán las casillas borradas.

En primer lugar se inicializan un par de vectores, `camino` y `visitados`, que servirán para la ejecución de la función `buscar_caminos`, explicada ya en la Sección 1.2.1. Una vez obtenido el camino, en caso de que el elemento seleccionado pertenezca a este, cada hilo cambiará todos los elementos de su camino por una `X`, para indicar que esa casilla se ha de borrar.

A continuación, con la ayuda de `__syncthreads()` todos los hilos se esperan a que hayan escrito, y es cuando mediante la ayuda de `atomicAdd`, comunican a la CPU si su casilla se ha eliminado.

1.2.2.2. `recolocar_tablero`

El objetivo de este kernel es, dado un tablero con `X` listas para ser borradas, quitarlas, bajar los caramelos existentes, y meter nuevos por encima. Para lograr esto, se guarda en una variable el valor que corresponde a la casilla del hilo, y acto seguido, se cuentan cuántas `X` hay debajo del hilo, y cuántos caramelos quedan por encima de este, y se espera a que todos los hilos terminen.

Una vez hecho esto, con la ayuda del valor de huecos que quedarán por debajo, cada hilo escribe el caramelo en su nueva posición (en caso de que el valor anterior no fuera una posición a borrar, pues en dicho caso la añadiríamos como hueco). Ahora solo queda saber si en nuestro hilo deberá insertarse un nuevo caramelo. Esto lo indica la diferencia entre los que había que borrar debajo y los que nos quedaban encima.

1.2.2.3. `bloques_especiales`

Este kernel es el encargado de realizar las acciones de los objetos especiales, pues se ejecuta en caso de que el elemento seleccionado no sea un caramelo. Con motivo de usarlos más adelante, se guarda en variables el objeto que tiene el hilo, y el que tenía el elemento seleccionado, y se espera a que todos terminen. Ahora distinguimos los siguientes objetos:

- **Bomba**: si el hilo es el seleccionado, pone su casilla lista para ser borrada y decide si borra una fila o columna. Mientras esto sucede, el resto de hilos esperan. Ahora, el resto comprueban la decisión del hilo bomba, y en caso de pertenecer a la fila o columna de la decisión, se marcan como borrados.
- **TNT**: en este caso se calcula si el hilo está a una distancia menor de cuatro filas o columnas del TNT, y en dicho caso se marca para borrar.
- **Rompecabezas**: de manera similar a la bomba, el hilo rompecabezas elige su tipo, y se marca borrado, mientras el resto espera. Ahora, el resto, en caso de coincidir con el tipo elegido por el hilo rompecabezas, se marcar para ser borrados.

En todos los casos, se usa `atomicAdd` a la hora de marcar borrados para decidir si el usuario debe perder una vida o no.

Capítulo 2

Implementación optimizada

2.1. Memoria global

Para este punto se nos ha pedido modificar nuestro proyecto de forma que se ejecute en distintos bloques, utilizando memoria global. Para llevar esto a cabo se han modificado los 3 kernels originales así como la implementación de 2 métodos extra, y una modificación en el *main*.

2.1.1. `encontrar_caminos`

El método de `encontrar_caminos` va a pasar a recibir un nuevo tablero auxiliar, pues al trabajar con diversos bloques no podemos hacer uso de `__syncthreads()`, por lo que deberemos hacer que el trabajo de los kernels pueda realizarse "por partes", para lo que escribimos en el nuevo tablero, y solo leemos del antiguo. Además de la fila, la columna, `n` y `m`, lo que hace que deje de recibir los borrados ya que esto se va a desarrollar en un nuevo método (por el mismo motivo).

Se modifican los valores de fila y columna además del nuevo cálculo de índices. También se añade una condición de forma que se verifique nos encontremos dentro del tablero. Se han eliminado los `__syncthreads()` de esta funcion así como el condicional encargado de comprobar si el valor era una `X` para así aumentar la cantidad de nodos borrados.

2.1.2. `recolocar_tablero`

Este método ha pasado a recibir también el tablero auxiliar (por el motivo ya explicado), así como `N` y `M`.

Aquí nuevamente se modifican tanto la fila, como la columna, y el id. También se añade la comprobación de estar dentro del tablero. Se eliminan los `__syncthreads()` y la condición del `for` se ve modificada (pues cambia la forma de calcular el identificador) para que se siga recorriendo la matriz completa.

2.1.3. `bloques_especiales`

Pasa a recibir ambos tableros, la dificultad, fila y columna, los borrados, la elección, el seleccionado y, `N` y `M`.

Hacemos el cambio de fila, columna, e id y metemos la comprobación de que esté dentro del tablero, así como eliminación de `__syncthreads()`. La primera diferencia está en que copiamos el tablero original en el auxiliar. Además, deben realizarse una serie de ajustes en cada uno de los objetos:

- **Bomba**: ahora es la CPU quien elige cómo se hará el borrado, en vez de la propia bomba, pues así acabamos con los problemas de sincronización a la hora de dividir el problema.
- **TNT**: en este caso se mantiene igual, pero modificando el cálculo de índices para obtener las distancias.
- **Rompecabezas**: para el rompecabezas lo que se ha hecho ha sido algo similar a la bomba. Es ahora el main quien decide qué tipo de elemento se rompe, marcando estos con una `X` para posteriormente borrarlos e incrementar el respecto contador de forma atómica.

2.1.4. `contar_borrados`

Aparece el método contar borrados, el cual recibe como parámetros, el tablero, una variable para los borrados que se incrementará de manera atómica si el valor al que accede el hilo es `X`. Este método nace también por el requisito de usar varios bloques en el kernel.

2.1.5. `calcular_dimensiones_optimas`

Aparece un nuevo método que devolverá un par de `dim3` para el grid y los bloques. Lo que se hace es obtener el máximo de hilos en un SM y el máximo de bloques en un SM, y mediante el cociente de estas dos magnitudes podemos hallar el número de hilos por bloque:

$$\mathcal{H} = \left\lfloor \frac{\texttt{maxThreadsPerMultiProcessor}}{\texttt{maxBlocksPerMultiProcessor}} \right\rfloor$$

Sin embargo, $\mathcal{H}$ puede no ser una potencia de 2, y en CUDA se acostumbra a que este valor lo sea, por lo que podemos aproximar a la siguiente potencia haciendo lo siguiente:

$$\mathcal{H}' = 2^{\lceil \log_2(\mathcal{H}) \rceil}$$

Ahora que ya sabemos el número de hilos por bloque, falta decidir la dimensión de este mismo. Para este caso haremos que la anchura (y altura) sea cuadrada, mediante

$$\mathcal{A} = \left\lfloor \sqrt{\mathcal{H}'} \right\rfloor$$

Finalmente, para decidir cuántos bloques debemos lanzar, dividimos $\mathcal{A}$ entre la anchura y altura de nuestro tablero, de forma que:

$$\texttt{dim\_grid} = \left(\left\lceil \frac{\texttt{M}}{\mathcal{H}'} \right\rceil, \left\lceil \frac{\texttt{N}}{\mathcal{H}'} \right\rceil \right) \quad \text{y} \quad \texttt{dim\_bloque} = (\mathcal{H}', \mathcal{H}')$$

En resumen, esta parte se basa en introducir bloques a la llamada del kernel, retocar ciertas partes del código para que los problemas sean divisibles entre los bloques de forma que trabajen a la par, y optimizar estos mediante el cálculo de la anchura del bloque, obtenida como:

$$\mathcal{A} = \left\lfloor \sqrt{2^{\left\lceil \log_2 \left\lfloor \frac{\texttt{maxThreadsPerMultiProcessor}}{\texttt{maxBlocksPerMultiProcessor}} \right\rfloor \right\rceil}} \right\rfloor$$

2.2. Memoria compartida

Para esta sección se pide introducir el uso de memoria compartida en los algoritmos previos. Durante el intento de realización de esta parte, nos dimos cuenta de que ciertos algoritmos que estábamos planteando no funcionaban adecuadamente, y tras probar los mismos códigos en otros equipos con diferentes versiones y características, sí funcionaban. Al no disponer de estos, no hemos podido desarrollar esta parte al completo, así como hacer pruebas para verificar que no existan fallos. Lo que sí podemos hacer aquí es decir qué es lo que haríamos, y en qué casos convendría usar esta memoria compartida.

- `buscar_caminos`: en el kernel que se buscan los caminos sería altamente recomendable introducir memoria compartida, pues la finalidad de esta es reducir el tiempo que se tarda en hacer lecturas en memoria global, y como en este kernel se accede muchas veces a diferentes posiciones del tablero, conseguiríamos rebajar este tiempo enormemente. Al ser una función más compleja y no disponer del equipo, no hemos podido implementarla, pero sí podemos dar un ejemplo de qué haríamos.

  Fijándonos en la Figura 2.1, suponiendo que el hilo X tiene que comprobar las posiciones que se ven, y que pertenece al bloque rojo, lo que haría el algoritmo es lo siguiente:

  1. Al llamar al kernel, pasar en el tercer argumento entre mayores y menores, `sizeof(char) * s_sim * s_dim`, para indicar la memoria compartida dinámica que se va a usar.
  2. Al entrar en el kernel, se deberá crear un nuevo tablero en memoria compartida dinámica como `extern __shared__ char s_tablero[]`.
  3. Cada hilo deberá cargar en la memoria compartida el valor de su id global, pero el id global no coincidirá con el de compartida. Esta vez será como si fuese el primer bloque del grid, es decir, obviamos los `blockIdx`.

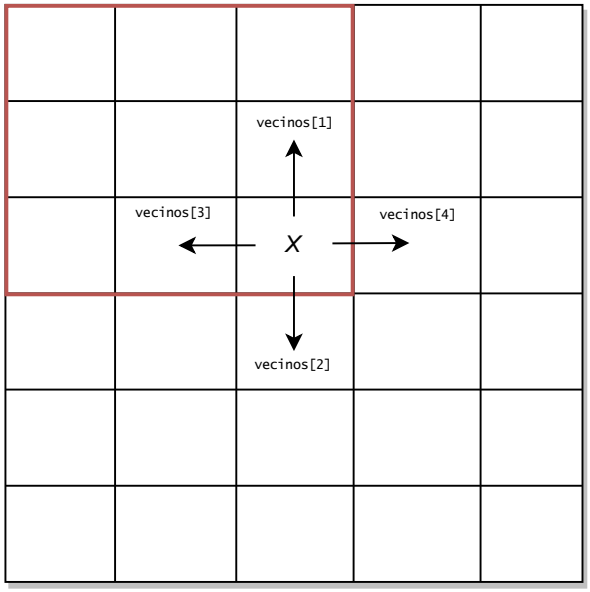

Figura 2.1: Situación de ejemplo en `buscar_caminos`

4. Una vez cargados, los hilos que cada bloque se esperarán mediante un `__syncthreads()`, de forma que otros hilos no intenten leer algo que aún no ha sido cargado.
5. Finalmente, en cada acceso que antiguamente fuese `tablero[id]`, se pondrá una sentencia `if` de forma que en función de la fila y columna del elemento a visitar, pueda determinarse si es del bloque o no. Entonces si es del bloque, como en los casos de la izquierda de la imagen, comparamos utilizando un acceso a `s_tablero[s_id]`. En caso contrario se pone lo que ya teníamos en las partes 1 y 2, pues querrá decir que ese valor no lo tiene el bloque del hilo en memoria compartida, teniendo que acceder a global, lo que implicará un mayor tiempo.

- `recolocar_tablero`: siendo este el kernel que trata de eliminar las `X` generadas por los kernels que borran caramelos, también sería recomendable hacer uso de la memoria compartida, pues como vemos en la Figura 2.2 cada hilo recorrerá toda la columna a la que pertenece, produciéndose una gran cantidad de accesos a memoria global, que podemos aligerar usando compartida. La idea es la siguiente:

  1. Seguimos los mismos primeros pasos que antes para preparar la memoria compartida dinámica.
  2. Una vez la memoria compartida esté lista, cada hilo va a ir visitando una posición de una fila a la que pertenece, por lo que si dicho elemento pertenece al bloque, accederá a la memoria compartida, en caso contrario a la memoria global, pues esa posición estaría en la memoria compartida del bloque de esa casilla. Esto se implementaría con *boundary checks* similares a los explicados para el caso previo.

  En este caso, sí se intentó implementar este algoritmo, y al poder hacer una breve prueba en otro equipo, pareció funcionar correctamente, aunque no se garantiza debido

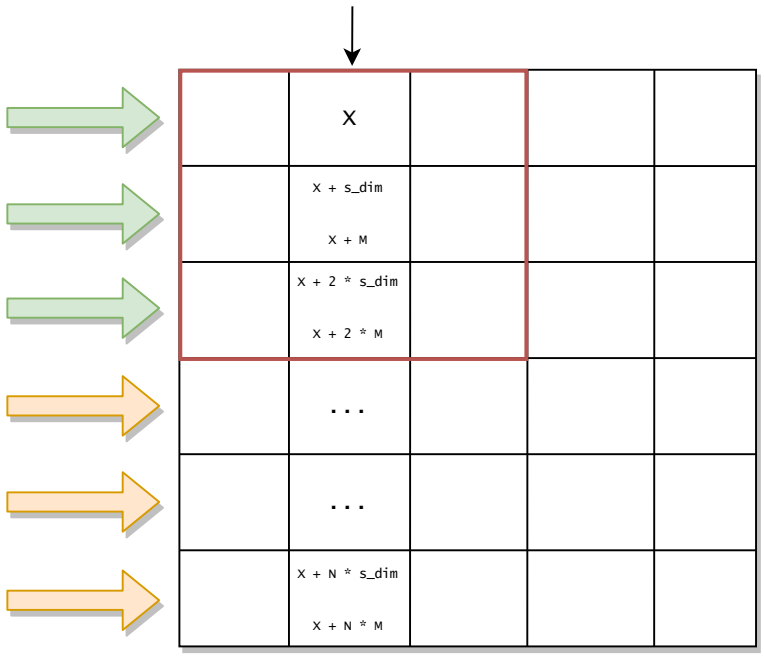

Figura 2.2: Situación de ejemplo en `recolocar_tablero`

a la ausencia de pruebas con más casos. Lo que se observaba en nuestros equipos, era que a la hora de acceder a la memoria global, los datos habían desaparecido, quedando solo los que se habían copiado a compartida, y no poder acceder a los datos que apuntan las flechas naranjas. Un estracto de la implementación se observa en el Código 2.1, el resto del proyecto de la parte 3 es igual que el de la parte 2 exceptuando este kernel.

```
191 int filas_recorridas = 0;
192 int shared_index = threadIdx.x;
193 for (int i = threadIdx.x; i < N * M; i += M) {
194     if (filas_recorridas <= s_dim) {
195         if (i < id && s_tablero[shared_index] != 'X') {
196             noX_encima++;
197         }
198         if (i > id && s_tablero[shared_index] == 'X') {
199             X_debajo++;
200         }
201     }
202     else {
203         if (i < id && tablero[i] != 'X') {
204             noX_encima++;
205         }
206         if (i > id && tablero[i] == 'X') {
207             X_debajo++;
208         }
209     }
210     shared_index += s_dim;
211 }
```

Código 2.1: Intento de implementación de mem. compartida

Los dos kernels que aún no hemos mencionado son `bloques_especiales` y `contar_borrados`, y esto se debe a que la cantidad de lecturas que hacen de memoria global ni siquiera se acerca a la cantidad que hacían los dos anteriores. Podría resultar que se tardase más tiempo en hacer la copia a memoria compartida y acceder a esta, que hacer ciertos accesos a global, por lo que independientemente de no disponer de los recursos para poder hacer correctas implementaciones, lo consideramos innecesario.

Como observación, los `malloc` y `free`, han sido reemplazados por `new` y `delete`, pues al utilizar los anteriores se obtenían errores en tiempo de ejecución, que estas nuevas sentencias parecen solucionar.