Programación II. Curso 2021-2022
Trabajo Obligatorio
betris

Descripción

En este trabajo se propone diseñar e implementar una versión particular del juego de `tetris` (`https://en.wikipedia.org/wiki/Tetris`) a la que denominaremos `betris`. El algoritmo principal estará basado en la estrategia de búsqueda con retroceso (backtracking).

En `betris` se dispone de un tablero rectangular cuyas dimensiones las determinan los parámetros de entrada dados al juego. En dicho tablero se deberán colocar las piezas según su orden de llegada. Cada familia de piezas válidas tiene una forma, un color y se identificará por un número. La forma de la pieza viene dada por la ubicación de los 4 bloques que la componen. El número que identifica a una familia de piezas, determinará su posición en el vector `vPiezas` que se proporciona en el fichero `betris.hpp`. Las familias de piezas que se definen en dicho vector serán las únicas que se podrán usar en nuestro juego `betris`, y sus características son:

Número	Color	Forma
0	AZUL	
1	VERDE	
2	ROJO	
3	CIAN	
4	AMARILLO	

Durante el juego `betris`, se asignará un número de columna en el tablero a cada una de las piezas que llegan. Una vez asignada una columna, la esquina superior izquierda de la pieza se colocará en dicha columna y lo más abajo posible, de forma que la pieza quede sobre la base del tablero o sobre otra pieza si la hubiera, pero sin solaparse con otra pieza ni salirse del tablero. En `betris` las piezas NO se podrán rotar.

Se describen a continuación la entrada, el objetivo y la salida esperada de `betris`.

Entrada: El programa leerá de la *entrada estándar* los siguientes parámetros:

```
nfils ncols objetivo retardo N1 N2 N3 N4 N5 N6 N7 N8.... -1
```

donde:

- `nfils` y `ncols` son el número de filas y columnas del tablero.
- `objetivo` es el número de filas que se deben completar.
- `retardo` es el número de milisegundos que se debe esperar tras mostrar el estado del tablero, antes de mostrarlo en su estado siguiente.
- `N1 N2 N3 N4 N5 N6 N7 N8... -1` son la secuencia de piezas que van llegando al juego y que se deben colocar. Cada número identificará la forma de la pieza que llega. La secuencia acabará con el número `-1` indicando el final de la entrada. Si el primer número `N1` es ya un número negativo, entonces el programa deberá generar una secuencia aleatoria de piezas de entrada, con un total de `-N1` piezas.

Objetivo: El objetivo es asignar columnas a las piezas que entran, por orden de llegada, y de tal manera que una vez colocadas en el tablero se completen `objetivo` filas del tablero, es decir haya `objetivo` filas sin huecos. Si `retardo`> 0 entonces se deberá mostrar por la pantalla el estado del tablero tras poner o retirar una pieza. Después de mostrar el estado del tablero, el programa deberá hacer una pausa de `retardo` milisegundos. Si el retardo es negativo, solo deberá mostrarse el tablero en su estado final.

El programa acabará cuando se rellenen el número `objetivo` de filas, o cuando se determine que es imposible asignar columnas a las piezas de tal manera que se rellenen `objetivo` filas.

Por ejemplo, para los parámetros:

```
6 5 4 100 0 0 4 2 3 2 1 -1
```

se considerará un tablero de 6 filas y 5 columnas, y el objetivo será rellenar 4 filas. Tras colocar o retirar una pieza se mostrará el estado del tablero y se realizará una pausa de 100 milisegundos. Los identificadores de las piezas que entran son `0 0 4 2 3 2 1`. Un posible tablero final solución (4 filas rellenadas) para estos parámetros de entrada es:

Sin embargo, si los parámetros de entrada son:

```
6 5 4 100 2 0 4 2 3 2 1 -1
```

entonces no es posible encontrar solución, y el programa deberá acabar indicándolo en la salida tal como se describe a continuación.

Salida: Además de mostrar el tablero en cada estado intermedio si `retardo`> 0, al finalizar se mostrará el tablero en su estado final y se deberán mostrar los identificadores de las fichas en la entrada, el número de fichas colocadas y los números de columna asignados respectivamente a cada ficha colocada.

Por ejemplo, para los parámetros:

```
6 5 4 100 0 0 4 2 3 2 1 -1
```

la salida final será:

```
Entrada (números de piezas): 0 0 4 2 3 2 1
Numero de piezas colocadas: 5
Salida (números de columnas): 1 1 3 0 0
```

Como para los parámetros:

```
6 5 4 100 2 0 4 2 3 2 1 -1
```

no hay solución, al finalizar se mostrará el estado del tablero como vacío, y se indicará que no se han podido colocar piezas con éxito, con una salida final como la siguiente:

```
Entrada (números de piezas): 2 0 4 2 3 2 1
Numero de piezas colocadas: -1
Salida (números de columnas):
```

Material de apoyo

Como material de apoyo se proporciona el fichero `betris.hpp`. Dicho fichero define constantes y tipos de datos que serán necesarios para realizar el trabajo. El fichero también contiene las cabeceras y la descripción informal de las funciones que necesariamente habrán de implementarse. Lee con atención el contenido de este fichero antes de empezar a diseñar el resto de tu código.

Al final del fichero, aparecen comentados los códigos de los colores que se deben utilizar como atributos para mostrar el tablero y las fichas con sus colores. Ten en cuenta que

para *"dibujar"* un bloque de una pieza o hueco del tablero, bastará representarlo escrito en pantalla como un caracter espacio en blanco, siempre y cuando el color de fondo con el que se escriba dicho espacio sea el adecuado en cada caso.

Tareas

Se deben realizar las siguientes tareas:

- Implementar en el fichero `betris.cpp` las funciones descritas en `betris.hpp`.
- Implementar en el fichero `mainB.cpp` el programa principal que lea los parámetros de entrada y llame a las funciones descritas en `betris.hpp` para resolver el juego de `betris` propuesto en este trabajo. Es decir, dados unos parámetros leídos de la *entrada estándar*, deberá mostrar la evolución del estado del tablero en pantalla y la salida final esperada, de acuerdo a los datos recibidos y a lo descrito en la sección anterior.
- Escribir un fichero `Makefile` que genere el ejecutable `mainB` que resuelva el juego de `betris`.

Notas:

- Se puede hacer el uso que se desee del fichero `betris.hpp`, pero obligatoriamente deberán implementarse las funciones que se describen en el fichero suministrado.
- Si es necesario (seguramente lo será), el fichero `betris.cpp` puede contener más funciones que las que se dan descritas en `betris.hpp`.

Forma y fecha de entrega

- El trabajo se realizará por parejas.
- Se deberá entregar un fichero con nombre `AAAAAA_BBBBBB.zip` en la tarea de *Moodle* habilitada para dicha entrega, donde `AAAAAA` y `BBBBBB` son los NIAs de los miembros de la pareja y `AAAAAA` $<$ `BBBBBB`. El fichero contendrá exclusivamente lo siguiente:
  - Todos los ficheros de código fuente, `.hpp` y `.cpp`, que se hayan utilizado.
  - Un fichero `Makefile` que compile dichos ficheros y genere el ejecutable `mainB` que resuelva el juego de `betris`.
- Las cabeceras de todos los ficheros de código fuente contendrán: los nombres, apellidos y NIAs de los dos miembros de la pareja.
- La entrega la realizará solamente el miembro de la pareja con NIA `AAAAAA`.
- El último día para entregar el trabajo será el 26 de mayo de 2022.