UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Reto 2: Abstracción

J. Fdez-Valdivia
Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Estructuras de Datos

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

El objetivo de este reto es resolver (a nivel de diseño) un problema en el que es necesario el uso de TDA de forma que el diseño de la solución deberá estar basado en nuevos tipos de datos creados mediante clases.

Básicamente, el reto que se propone consiste en desarrollar la infraestructura de TDA's necesarios para construir un programa para jugar al TETRIS.

El juego se representa en una ventana (ver figura 1) que dividiremos en 3 partes:

- Acumulador o tablero: donde se van apilando las piezas. En la imagen aparece en la parte izquierda con fondo de color blanco, con una pieza amarilla de forma cuadrada que está cayendo hacia la parte inferior, donde se encuentran acumuladas otras piezas.

- Cola: donde aparecen, de forma ordenada, las piezas que se van a introducir en el acumulador. En la imagen aparece una cola con 4 piezas en la parte central de la ventana.

- Marcadores: mensajes que informan de las puntuaciones y evolución del juego.

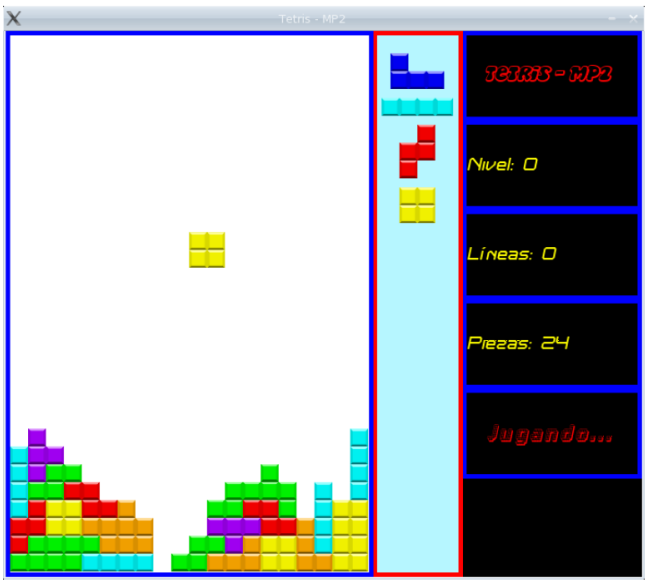

Figura 1.- Ejemplo gráfico del juego del Tetris

La dinámica del juego consiste en:

Mientras que el acumulador no esté "lleno":

1.- Obtener la siguiente ficha para jugar:

1.1.- Se saca la primera ficha de la cola y se introduce en la columna central del acumulador.

1.2.- A continuación se genera una nueva ficha de manera aleatoria y se añade al final de la cola.

2.- Bajar o colocar la ficha:

2.1.- Mover la ficha que está descendiendo según el control del jugador.
2.2.- Si la ficha se ha depositado sobre las ya acumuladas:

2.2.1.- Borrar las líneas completas, en caso de que existan, de forma que todas las líneas de encima bajan una posición para ocupar la línea eliminada.
2.2.2.- Actualizar marcadores.

El objetivo del juego consiste en maximizar el número de líneas que se han completado. En la figura 1 se puede observar que en la fila más baja sólo se necesita rellenar un hueco para ser completada. Por ejemplo, si la ficha que está cayendo se situara en el hueco doble de la penúltima fila, provocaría un borrado de ésta y un descenso, de una posición, de todas las de encima.

El acumulador y las piezas

El acumulador (figura 2) es la zona donde el jugador tiene que controlar la caída de las piezas de forma que se depositen formando líneas completas.

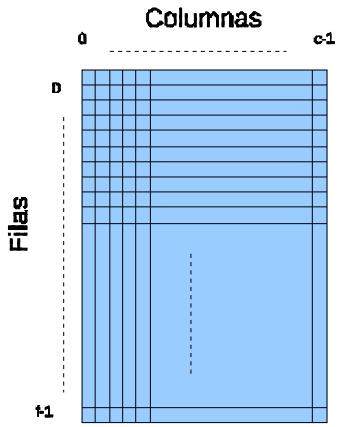

Figura 2.- Acumulador

Podemos considerar que está formado por una matriz de casillas con f filas y c columnas. Estas casillas pueden estar libres u ocupadas, de forma que una pieza podrá descender por el acumulador mientras las posiciones en su camino de descenso estén vacías, y se quedará depositada cuando, debajo de ella, haya alguna casilla que le impida descender.

Por otro lado, las piezas se pueden considerar, de forma similar, como una matriz de casillas donde algunas posiciones están ocupadas y otras vacías.
En el juego, el acumulador, el número de piezas y las respectivas dimensiones serán variables. Por tanto, el programa debería ser capaz de aceptar distintas configuraciones En la figura 3, se presentan las piezas clásicas del juego. Podemos ver que hay siete piezas: una de dimensiones 1x4 (con todas las casillas ocupadas), una de dimensiones 2x3 (con las posiciones 0,1 y 0,2 libres), etc.

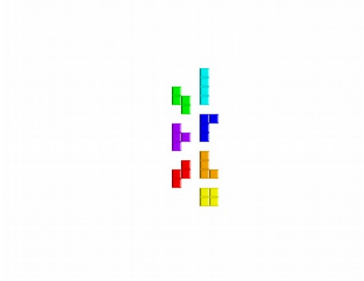

Figura 3.- Piezas cásicas del juego

Cada una de las casillas se representará como un hueco vacío o con la correspondiente imagen. En el caso de las piezas clásicas, podemos ver que cada osición se define con un pequeño bloque de un determinado color. Así, el acumulador o las piezas para este caso tendrá en cuenta 7 posibles imágenes con colores celeste, azul, naranja, amarillo, verde, violeta y rojo.
Durante una partida, el jugador debe tener la opción de controlar la posición donde caerá cada una de las piezas mediante controles. P.ej:

- Tecla flecha a la izquierda. Si es posible, mueve la pieza que está cayendo una posición a la izquierda.
- Tecla flecha a la derecha. Si es posible, mueve la pieza una posición a la derecha.
- Tecla flecha hacia abajo. Si es posible, mueve la pieza una posición hacia abajo.
- Tecla q. Si es posible, rota la pieza hacia la izquierda.
- Tecla w. Si es posible, rota la pieza hacia la derecha.

La cola de piezas

El juego dispone de un visualizador de las piezas que están por salir. Así, en la figura 1 se presenta un ejemplo donde podemos observar una cola que contiene 4 piezas. Cuando la pieza que está cayendo se deposite, la pieza azul se insertará en la posición central de la parte superior del acumulador, las otras 3 subirán una posición, y se seleccionará una nueva pieza aleatoria para ocupar la cuarta posición. El programa debería admitir distintas longitudes de cola (este será un parámetro dado en la configuración del juego), aunque para una partida concreta, el número de piezas que se incluyen en la cola es constante.
Mensajes y marcadores: La última parte de la ventana del juego contiene los mensajes y marcadores. Se puede observar en la figura 1, en la parte de la derecha. Aparecen los siguientes mensajes de texto:

- Mensaje del título: "Tetris". Este mensaje será variable, ya que se podrá modificar según la configuración de la partida.
- Nivel. Indica el nivel de juego en cada instante. El nivel inicial de una partida es el cero, pero se incrementará conforme aumente el número de líneas completadas.
- Líneas. Indica el número de líneas completadas hasta el momento.
- Piezas. Indica el número de piezas insertadas en el acumulador hasta el momento.
- Estado. Indica si se está jugando, o si la partida ya ha terminado.

Márgenes, colores y tipo de letra

La ventana de juego es el resultado de presentar distintos elementos: un acumulador, una cola, y 5 visualizadores de texto.

Modularización del problema

Aunque sois libres de definir los módulos que considereis necesarios, en este apartado se sugieren algunas ideas sobre posibles clases que se pueden diseñar. Independientemente de que sigais o no estas sugerencias, la solución final deberá estar basada en una correcta modularización del problema en base al diseño de TDA adecuados.

A título orientativo, éstas podrían ser algunas de las clases que se pueden definir:

- Clase Pieza. Permite gestionar las piezas individuales del juego. Almacena la configuración de las piezas y permitirá realizar acciones sobre ellas tales como construirlas, rotarlas, consultar sus dimensiones, etc.
- Clase Tablero. Permite gestionar el tablero de juego o acumulador. Sobre ella se realizarán operaciones como por ejemplo, consultar dimensiones, ver si una pieza encaja o no, añadirle una línea aleatoria, borrar una línea completa, comprobar si hay líneas completas y cuántas hay, etc.
- Clase Imagen. Para almacenar los bitmaps. Permitirá cargarlos desde un fichero en disco, dibujarlos en pantalla, etc.
- Clase Cola de piezas. Para disponer de la cola con las piezas que irán colocándose en el acumulador. Dispondrá de operaciones tales como obtener la siguiente pieza de la cola, consultar cual es la pieza n-ésima, añadir una nueva pieza al final de la cola, etc.

Aunque estas son algunas de las clases que se nos pueden ocurrir de forma más o menos directa a partir del enunciado del problema, debemos tener en cuenta que un buen diseño podría incluir otras clases. Por ejemplo, desde el punto de vista de su representación interna, tanto un tablero como una pieza contienen una estructura de datos de tipo matricial, por lo que se podría pensar en tener un módulo genérico que permita trabajar con matrices y que sea utilizado por ambas clases. Obviamente, aunque su representación interna pueda coincidir en mayor o menor medida, lo que las diferenciará será su interfaz. De esta forma, dos clases pueden ser muy distintas desde un punto de la abstracción, aunque su representación interna sea muy similar o incluso igual.

Se pide:

Especificar los TDA's (clases) necesarios indicando en cada caso el tipo de datos y con qué propiedades funcionales lo dotaríais. La idea no es que contruyais un programa que juegue al tetris. La idea es que especifiqueis las clases que considereis necesarias para el juego. Todas las funciones que veais interesantes en cada caso para formar parte del correspondiente TDA las habreis de especificar correctamente (preferiblemente usando Doxygen, aunque no es obligatorio) en el formato que querais (precondiciones/postcondiciones, o con clausulas de argumentos, valores de retorno, etc..).

Consideraciones:

1.- Podrán hacerse equipos de un máximo de 2 personas para debatir la solución y esa solución que se envíe se valorará con una puntuación igual para cada uno de los 2 miembros del equipo.

2.- Los 2 miembros del equipo deberán introducir el mismo fichero pdf con la solución en el sistema. Se sugiere como nombre del fichero reto2.pdf. Al principio del fichero deberá constar el nombre de los miembros del equipo.

3.- Todas las soluciones viables se puntuarán con 0.2 para la evaluación continua.

4.- Me interesa analizar vuestra capacidad de abstracción, de forma habréis de prestar atención al diseño de las clases, no a su implementación. De hecho no quiero que implementéis ni una sola línea de código. Todo habrá de hacerse a nivel de especificación. No pretendo que le dediquéis mucho tiempo, solo el necesario para pensar a alto nivel en la solución.

5.- El plazo para entregar vuestras soluciones (que deberéis dar obligatoriamente en un fichero pdf) se extenderá hasta el 31 de Octubre a las 23.55h.