Grupo Weboo
Universidad de La Habana

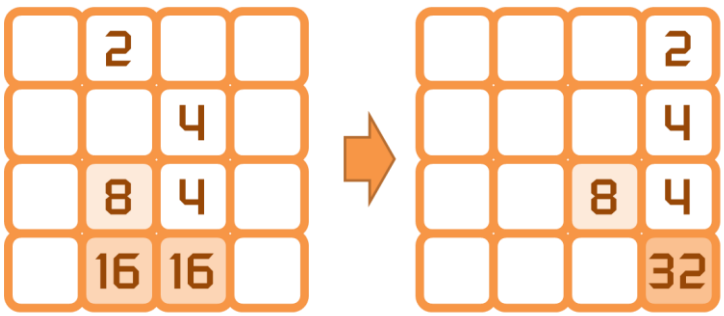
2048eXtendido

Primer Proyecto de Programación
Curso 2014-2015

RESUMEN

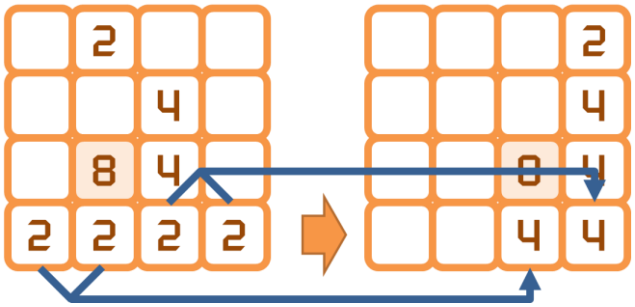
El 2048 es un juego que consiste en un tablero de 4x4 casillas en el que se ubican piezas con números. Los números siempre son una potencia de dos mayor o igual que dos. En cada turno el usuario debe decidir una dirección en la que mover todas las piezas del tablero (arriba, abajo, derecha o izquierda). Si en la dirección en que se mueven las piezas existen dos piezas de igual valor consecutivas, estas se combinan para formar una única pieza con valor igual a la suma de las dos mezcladas.

La siguiente figura muestra un tablero y el resultado de haber “movido” las piezas hacia la derecha.



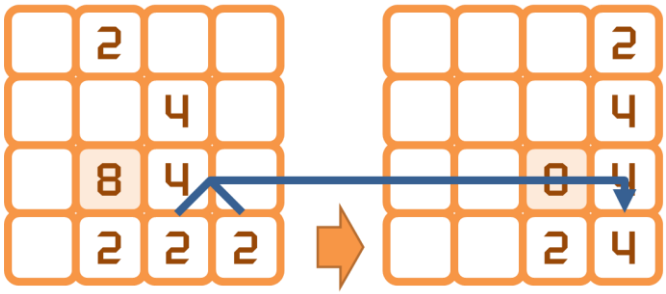
Note que las piezas se mueven hasta que llegan a la última casilla posible o chocan con una pieza que ya no se puede mover más. En un único movimiento una pieza sólo se mezcla una vez y si existen varias posibilidades se mezcla solamente la de la pieza más avanzada en la dirección seleccionada. Las figuras a continuación representan estos escenarios.

4 casillas iguales consecutivas, se mezclan únicamente 2 pares.



Aunque quedan dos 4 luego de mezclar los 2s estos no se mezclan en el mismo movimiento.

3 casillas iguales consecutivas, se mezcla únicamente el par más avanzado en la dirección.



En el movimiento se podrían mezclar el 2 más a la izquierda con el 2do, sin embargo se debe mezclar el más a la derecha con el anterior.

Al principio el juego empieza con sólo dos piezas en posiciones aleatorias y que pueden ser de valor 2 ó 4. Después de cada movimiento válido aparece en una casilla aleatoria un 2 ó un 4. Un movimiento es válido si con él se puede mover (o mezclar) alguna pieza. Si no existen movimientos posibles en un tablero el juego termina. En cada jugada se acumula en una puntuación del jugador el valor de las piezas que fueron mezcladas (por ejemplo, si se mezclan dos 4s se acumularía un 8). Si se alcanza una casilla con valor 2048 se dice que el jugador ganó (aunque puede continuar acumulando puntos).

El jugador tiene un conjunto de posibilidades de deshacer la última jugada (ambos, el movimiento y la aparición de la nueva pieza). En el juego original solo se puede realizar esta operación en dos ocasiones durante la partida.

EXTENSIONES

Usted deberá desarrollar una aplicación visual de Windows (ya sea Windows Forms o Windows Presentation Foundation) que permita a un usuario jugar 2048 extendido. Este juego será la misma lógica del original pero con algunos cambios.

Primero. El tamaño del tablero puede ser especificado antes de comenzar la partida y podrá ser desde 4x4 hasta 16x16 (siempre cuadrado). La cantidad de valores iniciales puede ser también variable desde 1 hasta 3.

Segundo. La cantidad de veces que se puede deshacer la jugada no tiene límites (tenga en cuenta que cada vez que se deshace una jugada, además de restaurar el tablero se quita los puntos ganados en la jugada deshecha).

Tercero. El jugador puede salvar un juego para continuarlo más adelante.

TUTORIALES RELACIONADOS CON EL PROYECTO

A continuación presentaremos un conjunto de tutoriales para el trabajo con generación de valores aleatorios y el manejo de ficheros.

GENERACIÓN DE NÚMEROS ALEATORIOS

Para la generación de una secuencia de números aleatorios en .NET se cuenta con el tipo `Random`. Este tipo se inicializa con un constructor vacío o con un valor como "semilla". La secuencia de valores está determinada por esta semilla, por lo tanto dos objetos de tipo `Random` creados con la misma semilla generarán los mismos números "aleatorios".

Se debe tener un único objeto `Random` en su aplicación y pedir todos los valores aleatorios a éste. Si crea constantemente objetos de tipo Random para pedir un único número puede ser que se repitan y por tanto que no sean "tan aleatorios".

El siguiente código imprime una secuencia de números aleatorios entre 0 y 99.

```
Random rnd = new Random();
while (true)
    Console.WriteLine(rnd.Next(0, 100));
```

TRABAJANDO CON FICHEROS

Para manejar ficheros y leer o escribir de ellos similar a como se lee o escribe en la consola usted puede utilizar los tipos `System.IO.StreamReader` y `System.IO.StreamWriter` para la lectura de un fichero y la escritura en un fichero. En el constructor de estos tipos se especifica el nombre del fichero que se desea abrir. Luego se puede invocar sucesivos llamados de `ReadLine` o `WriteLine` (similar al trabajo con el tipo `Console`). Al finalizar se debe invocar el método `Close` para que cierre el fichero y en caso que se esté escribiendo se termine de salvar el contenido.

El siguiente código crea un fichero y salva en éste un conjunto de valores, luego lo abre y lee estos valores para imprimirlos en la consola.

```
// Se crea el StreamWriter para escribir en un fichero de texto test.txt.
// Si no existe el fichero lo crea.
StreamWriter sw = new StreamWriter("test.txt");
// Se escribe una linea con el texto "4"
sw.WriteLine(4);
// Se escribe una linea con el texto "6"
sw.WriteLine(6);
// Se escribe una linea con el texto "7"
sw.WriteLine(7);
// Se cierra el fichero salvando la información
sw.Close();

// Se crea el StreamReader para leer de un fichero de texto
StreamReader sr = new StreamReader("test.txt");
// Se lee la primera linea y se interpreta como un entero
int valor1 = int.Parse(sr.ReadLine()); // Lee el "4" y lo convierte a entero
int valor2 = int.Parse(sr.ReadLine()); // Lee el "6" y lo convierte a entero
int valor3 = int.Parse(sr.ReadLine()); // Lee el "7" y lo convierte a entero
sr.Close();

// Imprimir los valores en la consola para chequear que se escribieron y leyeron
correctamente.
Console.WriteLine("Se escribieron y leyeron los valores {0}, {1} y {2}", valor1, valor2,
valor3);
```

SOBRE LEGIBILIDAD DEL CÓDIGO.

IDENTIFICADORES

Todos los identificadores (nombres de variables, métodos, clases, etc) deben ser establecidos cuidadosamente, con el objetivo de que una persona distinta del programador original pueda comprender fácilmente para qué se emplea cada uno.

Los nombres de las variables deben indicar con la mayor exactitud posible la información que se almacena en ellas. Por ejemplo, si en una variable se almacena “la cantidad de obstáculos que se ha encontrado hasta el momento”, su nombre debería ser `cantidadObstaculosEncontrados` o `cantObstaculos` si el primero le parece demasiado largo, pero nunca ~~`cOb`~~, ~~`aux`~~, ~~`temp`~~, ~~`miVariable`~~, ~~`juan`~~, ~~`contador`~~, ~~`contando`~~ o ~~`paraQueNoChoque`~~. Note que el último identificador incorrecto es perfectamente legible, pero no indica “qué información se guarda en la variable”, sino quizás “para qué utilizo la información que almaceno ahí”, lo cual tampoco es lo deseado.

- Entre un nombre de variable un poco largo y descriptivo y uno que no pueda ser fácilmente comprensible por cualquiera, es preferible el largo.
- Como regla general, los nombres de variables **no** deben ser palabras o frases que indiquen acciones, como ~~eliminando~~, ~~saltar~~ o ~~parar.~~

Existen algunos (muy pocos casos) en que se pueden emplear identificadores no tan descriptivos para las variables. Se trata generalmente de pequeños fragmentos de código muy comunes que “todo el mundo sabe para qué son”. Por ejemplo:

```
int temp = a;
a = b;
b = temp;
```

"Todo el mundo" sabe que el código anterior constituye un intercambio o *swap* entre los valores de las variables a y b, así como que la variable `temp` se emplea para almacenar por un instante uno de los dos valores. En casi cualquier otro contexto, utilizar `temp` como nombre de variable resulta incorrecto, ya que solo indica que se empleará para almacenar "temporalmente" un valor y en definitiva todas las variables se utilizan para eso.

Como segundo ejemplo, si se quiere ejecutar algo diez veces, se puede hacer

```
for (int i = 0; i < 10; i++)
  ...
```

en lugar de

```
for (int iteracionActual = 0; iteracionActual < 10; iteracionActual++)
  ...
```

Para los nombres de las propiedades (*properties* en inglés) se aplica el mismo principio que para las variables, o sea, expresar "qué devuelven" o "qué representan", solo que los identificadores deben comenzar por mayúsculas. No deben ser frases que denoten acciones, abreviaturas incomprensibles, etc.

Los nombres de los métodos deben reflejar "qué hace el método" y generalmente es una buena idea utilizar para ello un verbo en infinitivo o imperativo: Agregar, Eliminar, ConcatenarArrays, ContarPalabras, Arranca, Para, etc.

En el caso de las clases, obviamente, también se espera que sus identificadores dejen claro qué representa la clase: Robot, Obstaculo, Ambiente, Programa.

COMENTARIOS

Los comentarios también son un elemento esencial en la comprensión del código por una persona que lo necesite adaptar o arreglar y que no necesariamente fue quien lo programó o no lo hizo recientemente. Al incluir comentarios en su código, tome en cuenta que no van dirigidos solo a Ud., sino a cualquier programador. Por ejemplo, a lo mejor a Ud. le basta con el siguiente comentario para entender qué hace determinado fragmento de código o para qué se emplea una variable:

// lo que se me ocurrió aquel día

Evidentemente, a otra persona no le resultarán muy útiles esos comentarios.

Algunas recomendaciones sobre dónde incluir comentarios

- Al declarar una variable, si incluso empleando un buen nombre para ella pueden quedar dudas sobre la información que almacena o la forma en que se utiliza
- Prácticamente en la definición de todos los métodos para indicar qué hacen, las características de los parámetros que reciben y el resultado que devuelven
- En el interior de los métodos que no sean demasiado breves, para indicar qué hace cada parte del método

Es cierto que siempre resulta difícil determinar dentro del código qué es lo obvio y qué es lo que requiere ser comentado, especialmente para Ud. que probablemente no tiene mucha experiencia programando y trabajando con código hecho por otras personas. Es preferible entonces que “por si acaso” comente su código lo más posible.

Otro aspecto a tener en cuenta es que los comentarios son fragmentos de texto en lenguaje natural, en los cuales deberá expresarse lo más claramente posible, cuidando la ortografía, gramática, coherencia, y demás elementos indispensables para escribir correctamente.

Todos estos elementos son importantes para la calidad de todo el código que produzca a lo largo de su carrera, pero además **tendrán un peso considerable en la evaluación de su proyecto de programación**