Fundamentos de la programación

Práctica 2. Bombardero

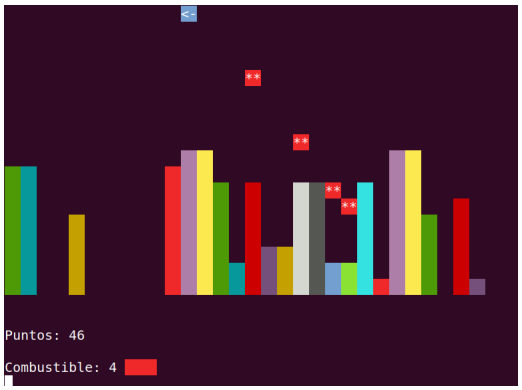
Indicaciones generales:

- La línea 1 del programa (y siguientes) deben contener los nombres de los alumnos de la forma:

  `// Nombre Apellido1 Apellido2`

- **Lee atentamente** el enunciado e implementa el programa tal como se pide, con la representación y esquema propuestos, implementando los métodos que se especifican, **respetando los parámetros y tipos de los mismos**. Pueden implementarse los métodos auxiliares que se consideren oportunos comentando su cometido, parámetros, etc.
- El programa, además de correcto, debe estar bien estructurado y comentado. Se valorarán la claridad, la concisión y la eficiencia.
- La entrega se realizará a través del campus virtual, subiendo únicamente el archivo `Program.cs`, con el programa completo.
- El **plazo de entrega** finaliza el lunes, 18 de enero. No obstante, se recomienda entregar la semana anterior para poder disponer de la corrección antes del examen.

En esta práctica vamos a implementar el juego *Bomber* (ZX Spectrum, 1984). Consiste en un avión que va recorriendo el cielo de una ciudad, bombardeando los edificios de la misma. El avión se desplaza con velocidad constante de derecha a izquierda en el área de juego, de manera que cuando llega al extremo izquierdo reaparece por el derecho, y además pierde una unidad de altura en cada vuelta. Este es el movimiento *de planeo del avión*, que no consume combustible, pero el jugador puede utilizar los controles "aswd" para acelerar/decelerar y ganar/perder altura consumiendo una unidad de combustible por cada una de estas acciones. El objetivo destruir todos los edificios con las bombas y aterrizar en el suelo antes de colisionar con alguno de ellos. A continuación se muestra el aspecto del juego:



La flecha azul de la línea superior representa el avión, que se mueve hacia la izquierda. Los dobles asteriscos rojos son bombas en caída y las columnas de color son los edificios. Cuando el usuario pulsa *'b"* el avión suelta una bomba que cae verticalmente. Cuando un edificio es alcanzado por una bomba pierde una unidad de altura, y además la explosión se propaga a todos los edificios contiguos que tengan la misma altura, perdiendo todos ellos también una unidad de altura. Cada bloque de altura incrementará un contador de puntos. Por ejemplo, en el dibujo de arriba, cuando la bomba de la derecha alcance el edificio verde, tanto ese edificio, como el azul de su izquierda perderán una unidad de altura y sumará 2 puntos. El número máximo de bombas simultáneas en el juego está limitado por una constante, como veremos.

En la clase principal declararemos los siguientes elementos:

```
class MainClass {
  static Random rnd = new Random(); // generador de números aleatorios
  const int ANCHO = 30, ALTO = 18, MAX_BOMBAS = 10, COMBUSTIBLE = 15;
  const int RETARDO = 150;
  const bool DEBUG = true;
```

El generador de aleatorios se utilizará para inicializar la altura de los edificios al principio del juego. La constante `ANCHO` determina número de edificios; `ALTO` determina la altura del área de juego (número de filas); `MAX_BOMBAS` determina el número máximo de bombas en el juego simultáneamente; `COMBUSTIBLE` son las unidades de combustible que tiene el avión; `RETARDO` determina los milisegundos de retardo entre frames. Por último, `DEBUG` es un flag para indicar si estamos en modo *depuración*, en cuyo caso se inicializará el juego con una configuración de edificios simple y fija (no aleatoria) para probar el funcionamiento; se sacarán también en pantalla las coordenadas del avión y cualquier otra información que nos sea de ayuda para depurar el programa.

Además de la constantes anteriores, el **estado del juego** viene dado por las siguientes variables:

- `int [] edificios`: array de tamaño `ANCHO` que contiene las alturas de los edificios en cada una de las columnas.
- `int [] bombas`: array de tamaño `ANCHO` que contiene la altura de las bombas en cada una de las columnas correspondientes (a lo sumo hay una bomba por columna). Tal como se ha dicho, el número máximo de bombas está acotado por `MAX_BOMBAS`. Para las columnas que no tienen bomba se guardará el valor `-1`.
- `avionX, avionY`: columna y fila donde se encuentra el avión, *teniendo en cuenta que* `(0,0)` *corresponde a la esquina inferior izquierda del área de juego.*
- `puntos`: contador de puntos que se inicia a 0 y se incrementa en 1 por cada altura destruida de los edificios.
- `combustible`: inicialmente toma el valor `COMBUSTIBLE` y se decrementa cuando alteramos el planeo del avión con las teclas "asdw".

Para desarrollar el juego se implementarán los métodos que se detallan a continuación. Para cada uno de ellos se especifican los parámetros, pero no la forma de paso de los mismos (entrada, salida o entrada/salida), que debe decidir el alumno. Se recomienda implementar estos métodos en el orden establecido e ir probando el funcionamiento de los mismos.

- `Inicializa(edificios, bombas, avionX, avionY, puntos, combustible)`: inicializa el vector `edificios` con alturas aleatorias en el intevalo `[1,ALTURA-3]`. Inicializa el vector `bombas` de modo que inicialmente no haya ninguna bomba en el aire. Así mismo, inicializa las coordenadas `(avionX,avionY)` posicionando el avión en la esquina superior derecha del área de juego, inicializa el contador de puntos a 0 y `combustible` a `COMBUSTIBLE`.
- `Renderiza(edificios, bombas, avionX, avionY, puntos, combustible)`: muestra en pantalla el estado del juego, con los edificios, las bombas y el avión, así como los puntos y el combustible. Nótese que, de acuerdo con lo explicado, el origen de coordenadas que manejamos en la *lógica interna* del juego está en la esquina inferior izquierda, mientras que el origen de coordenadas en pantalla está la esquina superior izquierda, por lo que habrá que **hacer la conversión de coordenadas correspondiente**.

  Por otro lado, para mejorar el aspecto del juego en pantalla, **cada ítem (edificios, bombas, avión) ocupa dos caracteres en horizontal**, tal como se aprecia en el dibujo. Nótese que el avión ocupa dos caracteres (`"<-"`), así como las bombas (`"**"`) y cada planta de los edificios. De este modo, el área juego ocupará $2 * ANCHO$ en horizontal.

Para dibujar los edificios de distintos colores, podemos obtener la paleta disponible en un vector `colors` con la sentencia:

```
ConsoleColor[] colors =
        (ConsoleColor[]) ConsoleColor.GetValues(typeof(ConsoleColor));
```

De dicha paleta podemos excluir los colores del avión, las bombas y otros, y utilizar el resto para los edificios.

Tal como se aprecia en el dibujo, se muestra también una barra de combustible que será verde cuando supere el 66 %, después amarilla hasta el 33 % y luego roja.

En este punto deben probarse estos dos métodos de inicialización y renderizado, antes de seguir adelante con los siguientes.

Para leer el input del usuario se utilizará el método `leeInput` que se proporciona implementado. Este método reconoce como entrada los caracteres "asdw" para direcciones y "b" para bombas (tanto en mayúcula como en minúscula). El bucle principal tendrá siguiente aspecto:

```
public static void Main (){
  // inicialización
  // renderizado
  ...
  // bucle ppal
  while ( ... ) { // mientras no haya colision y quedan edificios
    // input de usuario
    char c = LeeInput();

    // movimiento del avión, lazamiento de nueva bomba, movimiento (caída) de bombas...
    ...
    // colision de bombas con edificios y de avion con edificio
    ...
    // renderizado
    ...
    // retardo
    System.Threading.Thread.Sleep (RETARDO);
  } // fin while
  // informe resultados: estrellado o aterrizado
  ...
}

static char leeInput(){
  char d = ' ';
  if (Console.KeyAvailable) {
    string tecla = Console.ReadKey (true).Key.ToString ();
    tecla = tecla.ToUpper(); // conversion a mayúsculas
    switch (tecla) {
      case "A": d = 'a'; break;     case "S": d = 's'; break;
      case "W": d = 'w'; break;     case "D": d = 'd'; break;
      case "B": d = 'b'; break;     case "P": d = 'p'; break;
      case "Q": d = 'q'; break;
      }
  }
  while (Console.KeyAvailable) Console.ReadKey().Key.ToString();
  return d;
}
```

Los métodos a implementar para completar este código son los siguientes:

- `bool FinPartida(edificios)`: determina si todos los edificios han sido destruidos.
- `int CuentaBombas(bombas)`: devuelve el número de bombas actualmente en caída.

- `LanzamientoBomba(bombas, avionX, avionY)`: utiliza el método anterior para determinar si es posible lanzar una nueva bomba (si no se supera el máximo de bombas permitido); en ese caso, se lanza una nueva bomba en la posición donde se encuentra el avión. Este método **no** se ocupa de detectar posibles colisiones.

- `MueveBombas(bombas)`: mueve una posición hacia abajo cada una de las bombas en juego. Este método **no** se ocupa de detectar posibles colisiones.

- `MueveAvion(c, avionX, avionY, combustible)`: mueve el avión de acuerdo con el input `c` (leído en el bucle principal). El movimiento por defecto de planeo (si el jugador no pulsa ninguna tecla) es desplazarse una posición a la izquierda. Si se sale por la izquierda reaparece por la derecha perdiendo una unidad de altura.

  El jugador puede utilizar el combustible disponible del siguiente modo. Si pulsa `'a'` avanzará dos posiciones en vez de una; si pulsa `'d'` anula el movimiento por defecto y se queda parado en ese frame; con `'w'` asciende 1 posición además de avanzar a la izquierda (por defecto); si pulsa `'s'` desciende una posición además de avanzar a la izquierda. En todos estos movimientos consume 1 unidad de combustible. Cuando agota el combustible no puede realizar estos movimientos y el avión planea.

  Este método **no** se ocupa de detectar posibles colisiones.

- `ColisionBombas(edificios, bombas, puntos)`: para cada una de las bombas en el aire, comprueba si hay colisión con el edificio de abajo. En ese caso, destruye la cima de los edificios correspondientes e incrementa los puntos, utilizando el siguiente método:

  - `Impacto(edificios, i, puntos)`: resta una unidad de altura al i-ésimo edificio, así como a todos los contiguos que tengan la misma altura que éste. Incrementa en 1 el contador `puntos` por cada altura destruida en cada uno de esos edificios. Por ejemplo, si tenemos una secuencia de 8 edificios de altura 4 y cae una bomba en cualquiera de ellos, todos quedarán con altura 3 y se sumarán 8 puntos.

- `bool ColisionAvion(c, edificios, avionX, avionY)`: determina si el avión ha colisionado con algún edificio. Nótese que si el jugador ha avanzado 2 posiciones (pulsando `"a"`) y solo comprobamos la posición actual del avión, se podrían *traspasar* edificios. Para evitarlo, este método también recibe el parámetro `c` con la pulsación de teclado. En caso de avance habrá que comprobar también la colisión con la posición anterior.

  El resultado de este método se utilizará para terminar el bucle de juego.