Fundamentos de la programación

Grado en Desarrollo de Videojuegos

Examen parcial, enero 2019

Indicaciones generales:

- Se entregará un único archivo `Program.cs` con el programa completo. Se proporciona una plantilla con el esquema del programa y un método `LeeInput` ya implementado.
- La línea 1 será un comentario de la forma `// Nombre Apellido1 Apellido2`
- La línea 2 será un comentario de la forma `// Laboratorio, puesto`
- **Lee atentamente el enunciado** e implementa el programa tal como se pide, con los tipos de datos, métodos, parámetros y requisitos que se especifican. No puede modificarse la representación propuesta, ni alterar los parámetros de los métodos especificados. No se especifica el modo de paso de los parámetros en los métodos (`out`, `ref`, ...), que debe determinar el alumno.
- Pueden implementarse todos los métodos adicionales que se consideren oportunos, especificando claramente su cometido, parámetros, etc.
- **El programa debe compilar y funcionar correctamente, y estar bien estructurado y comentado**. Se valorarán la claridad, la concisión y la eficiencia.
- **Entrega:** la entrega se hará a través del servidor FTP de los laboratorios.

Vamos a implementar un solitario de cartas. Utilizaremos una baraja con 4 palos (oros, copas, espadas y bastos), cada uno con N cartas numeradas de 0 a $N-1$ (comenzamos en 0 para facilitar la implementación). En total la baraja tendrá $4*N$ cartas; por ejemplo, la baraja española de 40 cartas puede simularse con $N = 10$. El juego comienza con el mazo de cartas desordenado y consiste en reordenarlas en 4 montones (uno por palo) tal como se explica:

- Inicialmente el **mazo** de cartas contiene la baraja completa con las cartas desordenadas. Los **4 montones** correspondientes a los palos están vacíos y se irán rellenando con las cartas del palo correspondiente en orden creciente a medida que avance el juego.
- El mazo se sostiene en una mano con las cartas boca abajo y se van sacando de 2 en 2, apilándolas boca arriba en una **pila** sobre la mesa, de modo que solo queda accesible la carta de la cima (la de más arriba).
- La carta accesible puede extraerse y colocarse en el montón de su palo siempre que sea la siguiente en el orden de dicho montón, para que finalmente los montones queden ordenados. Así, inicialmente solo pueden sacarse los 0 de cada palo. Cuando se extrae una carta a su montón, queda accesible la que tiene debajo en la pila, que podrá también sacarse a su montón si respeta el orden del mismo. Por otro lado, aunque una carta pueda sacarse de la pila al montón de su palo, es el usuario quien decide sacarla o no.
- Cuando se sacan todas las cartas del mazo se recogen las de la pila de la mesa (sin alterar el orden) y vuelven a utilizarse como mazo. Se repetirá este ciclo de extracción hasta que todos los motones estén completos y ordenados (y el mazo vacío) o que el usuario pare el juego. Cuando en el mazo queda una sola carta (no se pueden sacar 2), se apilará ella sola en la pila de la mesa. Nótese que el solitario puede no tener solución, por ejemplo si queda inicialmente en el mazo el 0 y el 1 de un mismo palo (el 0 nunca será accesible).

Para implementar el juego, cada carta vendrá representada por un único entero entre 0 y $4 * N - 1$ de modo que tenemos la distribución de intervalos:

$$\underbrace{0, ..., N-1}_{oros}, \quad \underbrace{N, ..., 2*N-1}_{copas}, \quad \underbrace{2*N, ..., 3*N-1}_{espadas}, \quad \underbrace{3*N, ..., 4*N-1}_{bastos}$$

Por ejemplo, si $N = 10$ el 3 representa el 3 de oros, el 10 será el 0 de copas y el 36 será el 6 de bastos. En general, dada una carta k, su palo puede obtenerse como k/N y su valor dentro del palo como $k\,\%N$.

El mazo de cartas de la mano y la pila sobre la mesa se representan con un único vector de cartas, `mazo`, y un índice `des` que divide el vector en dos partes: las cartas a la izquierda de `des` representan la pila de descubiertas, `des` apunta a la carta que aparece descubierta sobre la mesa y las cartas a la derecha representan el mazo que aun se tiene en la mano; sacar cartas del mazo equivale a avanzar el índice `des`.

A medida que se van extrayendo cartas a los montones, el vector `mazo` se irá vaciando: las cartas que quedan ocupan las primeras posiciones del vector y las posiciones vacías quedarán al final del vector con valor `-1` (que no representa ninguna carta). Por último, los montones se representan con un vector de 4 enteros (uno por palo), cada uno de los cuales representa la *última carta de dicho montón*. Es decir, si el montón 1 (copas) tiene el valor 3, significa que en ese montón ya están las cartas 0, 1, 2 y 3 de copas.

El estado del juego viene dado por las siguientes variables (además de la constante `N`):

- `mazo`: vector de $4 * N$ enteros correspondientes a las cartas que hay en el mazo y en la pila que hay sobre la mesa;
- `des`: índice al vector `mazo` que indica la primera carta descubierta sobre la mesa. Tal como se ha explicado, las que quedan a la izquierda de `des` son las que hay sobre la mesa y las de la derecha las que quedan en la mano.
- `montones`: vector de 4 enteros (correspondientes a los 4 palos de oros, copas, espadas y bastos, en este orden). Estos enteros representan la última carta que hay el correspondiente montón. Incialmente todos están vacíos, por lo que tendrán el valor -1.

Por ejemplo, con $N = 10$ (baraja con 40 cartas) un estado inicial puede ser:

```
mazo = [4,20,32,0,...,5]  // valores de 0 a 39 desordenados
des = -1                  // toda la baraja en la mano, pila vacia
montones = [-1,-1,-1,-1]  // montones vacíos (oros, copas, espadas, bastos)
```

Al sacar dos cartas del mazo a la mesa tendremos:

```
mazo = [4,20,32,0,...,5]  // igual que antes
des = 1                   // avanza 2 posiciones
montones = [-1,-1,-1,-1]  // vacíos
```

En este momento el índice `des` señala a la carta 20, que corresponde al 0 de espadas, que puede extraerse a su montón, obteniendo:

```
mazo = [4,32,0,...,5,-1]  // se quita el 20 y se compacta el vector (-1 al final)
des = 0                   // al quitar el 20, queda accesible el 4 de oros
montones = [-1,-1,0,-1]   // el 0 de espadas se coloca en su montón
```

Si sacamos otras dos a la pila de la mesa tendremos:

```
mazo = [4,32,0,...,5,-1]  // igual
des = 2                   // avanza 2 posiciones
montones = [-1,-1,0,-1]
```

Ahora `des=2` que corresponde al 0 de oros, que puede sacarse a su montón obteniendo:

```
mazo = [4,32,...,5,-1,-1] // se extrae el 0 y se compacta el vector
des = 1                   // se decrementa
montones = [0,-1,0,-1]    // se coloca el 0 de oros
```

Para implementar el juego desarrollaremos los siguientes métodos:

- [**1 pt**] `void Inicializa(int [] mazo, int des, int [] montones)`: rellena el vector `mazo` con todas las cartas de la baraja ordenadas de 0 a $4 * N - 1$); inicializa `des` de modo que quede todo el mazo en la mano y ninguna carta en sobre la mesa; inicializa los montones para que quden vacíos.
- [**1 pt**] `void Mezcla(int [] mazo)`: desordena las cartas del mazo con el siguiente algoritmo: recorre el vector de cartas de principio a fin intercambiando cada una de ellas con una de las siguientes en el vector elegida de manera aleatoria (utilizando el método `rnd.Next`).
- [**1 pt**] `void MuestraCarta(int valor, int palo)`: escribe en pantalla el `valor` de la carta (entre 0 y $N - 1$) seguido del `palo` de la misma (inicial): `'o'`, `'c'`, `'e'`, `'b'`.
- [**1 pt**] `void MuestraEstado(int [] mazo, int des, int [] montones)`: muestra en pantalla el estado del juego, i.e., el estado de los montones y la pila de cartas de la mesa. Para mostrar cada carta utiliza el método anterior. Por ejemplo, una salida puede ser:

```
   Oros: 0o
  Copas:
Espadas: 0e 1e 2e
 Bastos: 0b

   Mesa: 1b 0c 2c
```

(En este caso, `2c` es la carta que queda accesible en la pila de cartas de la mesa).

Además, si la constante `DEBUG` (véase plantilla proporcionada) es `true` mostrará además la información del estado en un formato simple para facilitar la depuración. Mostrará el vector `mazo` (solo las cartas, no los huecos vacíos -1) con la carta indicada por `des` entre asteriscos, el propio valor de `des` y el valor de los montones. Para el estado anterior, tendremos:

```
mazo: 10 3 *5* 11 2 1 4
des:  2
montones: 0 -1 2 0
```

Nota: para comprobar el funcionamiento del juego será útil depuerarlo con valores pequeños de `N` (por ejemplo, 3) y utilizar esta opción de depuración.

Para comprobar el buen funcionamiento de estos métodos, implementar una primera versión del método `Main` que arranque el juego y muestre el estado inicial. Después continuar con los métodos siguientes:

- [**1 pt**] `int UltimaPos(int [] mazo)`: devuelve la posición de la última carta de `mazo`, i.e., la última posición del vector `mazo` con valor no negativo.

- [**1 pt**] `void SacaCartasMazo(int [] mazo, int des)`: efectúa la operación de sacar 2 cartas del mazo y pasarlas a la pila de la mesa modificando el índice `des` adecuadamente. Para ello será útil el método anterior `UltimaPos` y se tendrá en cuenta la mecánica del juego que hemos explicado:

  - si no quedan cartas en el mazo, se reinicia la secuencia cogiendo las de la pila de la mesa;
  - si queda una única carta en el mazo se saca solo dicha carta;
  - en otro caso se sacan 2 cartas normalmente.

- [**1 pt**] `void QuitaCartaMonton(int [] mazo, int des, int [] montones)`: comprueba si es posible sacar la primera carta descubierta a su montón correspondiente y en ese caso la saca. Además, compacta el vector `mazo` desplazando a la izquierda las siguientes cartas, de modo que todos los huecos vacíos (con -1) queden el final del vector. Si no es posible sacar la carta visible a un montón, este método no hace nada.

- [**2 pt**] `Main()`: integra los métodos anteriores e implementa el juego como tal. Para ello inicializa y mezcla el mazo de cartas, muestra el estado inicial y arranca el bucle principal. En cada vuelta este bucle lee el input de usuario, lo procesa y muestra el estado del juego. El usuario podrá sacar cartas del mazo a la mesa con la tecla `Espacio`, retirar una carta de la pila de la mesa al montón correspondiente con `Enter`, o terminar el juego con `Escape`. El juego termina cuando se han colocado todas las cartas del mazo en los montones, o cando el usuario fuerza la terminación. Para leer el input del usuario se utilizará el método `LeeInput` que ya se da implementado en la plantilla y que devuelve `'s'`, `'r'`, `'q'` para las opciones anteriores.

- [**1 pt**] `LeeCartas(string file, int [] mazo, int des, int [] montones)`: lee la secuencia de cartas cartas del archivo `file` y rellena `mazo` con dichas cartas en el orden dado. Deja todas las cartas en el mazo (sin ninguna sobre la mesa) y los `montones` vacíos. Por ejemplo, un posible archivo de entrada para `N=2` puede ser:

  ```
  0 3 5 1 6 4 7 2
  ```

  Para incorporar este método al programa, al principio del juego dar la opción al usuario de leer las cartas de un archivo o bien generar una mezcla aleatoria como se hacía antes.

  Nota: puede probarse el juego sabiendo que con esta disposición el solitario se termina en dos recorridos del mazo, suponiendo que se sacan a os montones todas las cartas posibles en cada momento del juego.