Facultad de Informática – Universidad Complutense de Madrid

Tecnología de la Programación de Videojuegos 1

Hoja de ejercicios del Tema 2

1. Implementa la clase `Circle`, con atributos privados `x`, `y` y `radious` de tipo `int`, donde (`x`, `y`) representa el centro del círculo. Implementa la constructora con argumentos, y dos métodos públicos para calcular el área del círculo y desplazar su centro una cierta distancia. Define la constante `PI` y usa el método `pow` de la biblioteca `cmath` para implementar el cálculo del área. Añade también a la clase un método para mostrar la información del círculo en forma textual (en el flujo `ostream` proporcionado como parámetro). Finalmente implementa una función `main` que haga lo siguiente:

- Pide las coordenadas y el radio del círculo por consola.
- Construye un círculo con dicha información invocando a la constructora.
- Calcula el área del círculo y muestra la información del círculo por pantalla.
- Pide una distancia para trasladar el círculo y a continuación lo traslada.
- Muestra de nuevo la información del círculo.

2. Modifica la representación de la clase `Circle` del ejercicio anterior de manera que el centro del círculo pase a ser un objeto de una nueva clase `Point2D`, con atributos para las coordenadas del punto, la constructora correspondiente, y un método para trasladar el punto una distancia en horizontal. Comprueba que la función `main` del ejercicio anterior sigue funcionando correctamente.

3. Implementa un juego que consiste en encontrar un número en una matriz de enteros. Para ello implementa la clase `Matrix` que, además de una constructora por defecto, contendrá los siguientes métodos públicos:

- `int get(uint row, uint col)`, que devuelve el elemento de la posición (`row`, `col`).
- `void set(uint row, uint col, int v)`, que coloca el valor `v` en la posición (`row`, `col`).
- `bool search(int v)`, que devuelve `true` si y solo si el elemento `v` pertenece a la matriz.

Una vez implementada la clase `Matrix`, implementa la clase `Game`, que será la encargada de ejecutar una partida del juego. Concretamente la clase `Game` contendrá, entre otros, dos atributos privados del tipo `Matrix`, a los que llamaremos `gameMatrix` y `visibleMatrix`. El primero de ellos representa la matriz en la que el jugador debe encontrar el número, mientras que el segundo se utiliza para ir visualizando los números destapados por el jugador.

La constructora por defecto de la clase `Game` inicializa la matriz del juego con números aleatorios comprendidos entre 0 y una constante `MAX_NUM`. La generación de números aleatorios puedes realizarla inicializando el generador con

```
#include <random>
std::mt19937_64 generador; // Generador Mersenne Twister de 64 bits
std::uniform_int_distribution<int> distrib(0, MAX_NUM - 1);  // [0, MAX_NUM)
```

y llamando a `distrib(generador)` cada vez que quieras obtener un número pseudoaleatorio. El atributo `visibleMatrix` inicializa todas sus componentes con un `-1`, que indica que la casilla no es visible.

La clase `Game` tendrá el método `void play()`, que lleva a cabo la simulación del juego. Este método primero genera de forma aleatoria un número a buscar que debe estar en la matriz. A continuación solicita

al jugador que introduzca los valores de una fila y una columna. Si `gameMatrix` contiene en dicha casilla el número buscado, entonces el juego termina y el jugador gana. En otro caso se repite el proceso hasta un máximo de `MAX_ATTEMPTS`, en cuyo caso el jugador pierde. Un ejemplo de ejecución es el siguiente:

```
Busca el número.... 6
X X X X X
X X X X X
X X X X X
X X X X X

Dame las coordenadas: 1 2
X X X X X
X X 8 X X
X X X X X
X X X X X

Dame las coordenadas: 1 1
X X X X X
X 6 8 X X
X X X X X
X X X X X

¡Enhorabuena! Has encontrado el número en solo 2 intentos
```

La función `main` únicamente creará la partida e invocará al método `play`.

4. Implementa una clase para representar un tablero de juego con NxM casillas en las cuales puede haber una pelota (número mayor que 0 y menor que 10) o no haber nada (número 0). La clase debe tener los siguientes métodos:

- `void load(const string& filename)`, que dado un fichero, que empieza con los números N y M separados por espacio y después contiene N*M números enteros del 0 al 9, lo recorra y rellene/cargue el tablero. Si el fichero no existe o hay algún error se lanzará una excepción.
- `void draw()`, que dibuja el tablero en consola, utilizando como separadores de filas y columnas guiones y barras verticales respectivamente. Los ceros se interpretarán y dibujarán como espacios.
- `void moveUp()`, que dado un tablero lo modifique de tal manera que los números del 1 al 9 se muevan hacia arriba en sus columnas respectivas dejando por tanto los espacios debajo. El tablero se deberá repintar tras cada movimiento de todos los números. La función pintar deberá escribir `"\x1b[2J"` (borrar la pantalla) antes de imprimir nada, para así pintar el nuevo tablero justo encima del anterior. Además se deberá llamar a la función `sleep(PAUSE)` tras cada repintado, siendo `PAUSE` un número de milisegundos (por ejemplo 300) definido como una constante entera. De esta manera parecerá que los números se están moviendo hacia arriba.
- `void moveDown()`, análoga a la anterior solo que haciendo caer los números hacia abajo y dejando los espacios arriba.

5. Implementa la clase `Scores` para manejar las mejores puntuaciones obtenidas en un juego. Éstas se almacenarán en un vector (clase `Vector` vista en clase o `vector` de la STL) de estructuras de tipo `ScoreReg` (registros con un `string` para el nombre y un `int` para su puntuación asociada). La clase debe tener los siguientes métodos:

- `void load(const string& filename)`: carga del fichero proporcionado los registros de puntuaciones almacenados en él. El fichero se organiza por parejas de líneas, la primera contiene el nombre del jugador (posiblemente con espacios) y la segunda su puntuación asociada. La lectura acaba cuando ya no queda nada más que leer en el fichero (usa el método `fail()`). Si hay algún error se lanzará una excepción.
- `void printTopNScores(int n)`: imprime por consola los registros con las mejores n puntuaciones, ordenados de mayor a menor puntuación.
- `void addScore(const string& name, int score)`: añade un nuevo registro de puntuación y lo coloca en el vector de manera que éste siga estando en orden.
- `bool save(const string& filename)`: guarda en el fichero proporcionado todos los registros de puntuaciones preservando el formato del fichero indicado anteriormente.

Implementa una función `main` que: primeramente cargue un fichero de puntuaciones, después solicite una nueva puntuación (con su nombre de jugador asociado), pida una n, muestre las mejores n puntuaciones, y repita el proceso hasta que se introduzca un nombre de jugador vacío. Finalmente, se actualiza el fichero.

6. Re-implementa los ejercicios 10 y 11 de la hoja de ejercicios del Tema 1 usando clases.

7. Extiende la implementación de la clase `Vector` del tema 2 de manera que el array, no solo se redimensione cuando es necesario tener más espacio, sino que también lo haga cuando sobre espacio. En concreto, se debe redimensionar a la mitad de su capacidad actual cuando, tras eliminar un elemento, su ocupación sea inferior a un tercio de la capacidad del array. Escribe un programa que mediante un menú principal permita hacer pruebas de la nueva funcionalidad.