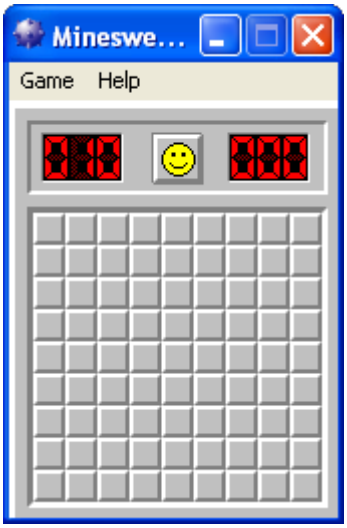
El Buscaminas

Estructuras de Datos
Facultad de Informática - UCM

Este ejercicio es opcional y puede entregarse a través del problema `F03` de *DOMjudge*. Corregiré las entregas que se hagan para proporcionar feedback, pero este ejercicio **no puntúa para la evaluación continua**.

El buscaminas es un videojuego inicialmente creado por Curt Johnson para el sistema operativo OS/2 de IBM, aunque alcanzó su popularidad gracias a su versión para MS Windows, que incluía este juego preinstalado en muchas de las versiones de este sistema operativo, concretamente desde la versión 3.1 hasta la versión 7.



El juego del buscaminas transcurre en un tablero donde se encuentran ocultas una serie de minas. El jugador puede hacer clic en una de las casillas del tablero para explorar su contenido. Si la casilla seleccionada contiene una mina, el jugador pierde. En caso contrario, se muestra un número indicando cuántas minas hay en las casillas colindantes a la seleccionada (en horizontal, vertical y diagonal). Por otro lado, si un jugador sospecha de la existencia de una mina en una casilla determinada, puede colocar una bandera en dicha casilla para marcarla y así no hacer clic luego sobre ella de manera accidental.

En este ejercicio vamos a considerar una variante multijugador de este juego. Tenemos a varios jugadores dentro de un mismo tablero, y cada uno puede explorar las casillas del mismo o colocar banderas. Los jugadores ganan puntos cuando colocan banderas en posiciones en las que se haya una mina, y pierden puntos cuando las colocan en posiciones vacías.

En este ejercicio tendrás que implementar un TAD que implemente la lógica de este juego. Para ello defina una clase `Buscaminas` con las operaciones que se describen a continuación. Antes de ello, define un tipo de datos `jugador` como sinónimo de `std::string`.

Las operaciones a implementar son las siguientes:

- `anyadir_mina(x, y)`

  Añade una mina en las coordenadas indicadas del tablero. Si ya existe una mina en esa posición, no hace nada.

- `anyadir_jugador(jugador)`

  Añade a la partida un jugador con el nombre pasado como parámetro. Si ya existe un jugador con ese nombre, lanza una excepción `std::invalid_argument` con el mensaje `Jugador existente`.

- `explorar(jugador, x, y)`

  Hace que `jugador` compruebe el contenido de la casilla situada en la posición (`x`, `y`). Si esa casilla contiene una mina en ese momento, devuelve -1 y se elimina al jugador de la partida. En caso contrario, se devuelve un número entre 0 y 8 indicando cuántas casillas colindantes a (`x`, `y`) contienen una mina.

  En el caso en el que el jugador haya explorado una casilla con mina, la mina sigue en el tablero, y puede seguir eliminando a otros jugadores que exploren esa misma casilla posteriormente.

Si el `jugador` indicado no se encuentra dentro de la partida, el método lanza una excepción `std::invalid_argument` con el mensaje `Jugador no existente`.

- `bandera(jugador, x, y)`

  Indica que el `jugador` quiere colocar una bandera en la posición (`x`, `y`) del tablero. Si existe una mina en esta posición, se le sumará un punto al jugador y se devolverá `true`. En caso contrario, se le restaran cinco puntos al jugador, y se devolverá `false`. Un jugador puede colocar una bandera en una misma casilla tantas veces como quiera.

  Si el `jugador` indicado no se encuentra dentro de la partida, el método lanza una excepción `std::invalid_argument` con el mensaje `Jugador no existente`.

- `puntuacion(jugador)`

  Devuelve la puntuación del `jugador` dado. Si no se encuentra dentro de la partida, lanza una excepción `std::invalid_argument` con el mensaje `Jugador no existente`.

- `num_minas()`

  Devuelve el número de minas que existen en el tablero.

- `bordes_tablero(&min_x, &min_y, &max_x, &max_y)`

  Devuelve cuatro números enteros indicando los bordes del área del tablero en la que se encuentran las minas. En otras palabras, devuelve las coordenadas del rectángulo más pequeño que contiene todas las minas del tablero. Para ello devuelve cuatro números enteros, indicando que dicho rectángulo se extiende desde la posición (`min_x`, `min_y`) hasta la posición (`max_x`, `max_y`).

  Si el tablero no tiene minas se debe lanzar la excepción `std::domain_error` con el mensaje `Tablero vacio` (sin tilde en la palabra *vacío*).

No olvides indicar el coste de cada una de las operaciones. Ten en cuenta que todas las operaciones deben implementarse de la manera más eficiente posible, desde el punto de vista del coste asintótico en tiempo.

Ninguno de los métodos de la clase `Buscaminas` debe realizar operaciones de E/S. El manejo de E/S debe hacerse de manera externa al TAD.

Entrada

La entrada consta de una serie de casos de prueba. Cada caso de prueba consta de una serie de líneas, cada una describiendo una operación en el TAD:

- `mina x y`, donde `x` e `y` son números enteros. Sirve para añadir una mina al tablero.
- `jugador jug`, donde `jug` es un nombre de jugador. Sirve para añadir dicho jugador al juego.
- `explorar jug x y`, donde `jug` es un nombre de jugador, `x` e `y` son números enteros. Con esta operación se indica que el jugador con nombre `jug` explora la casilla (`x`, `y`).
- `bandera jug x y`, donde `jug` es un nombre de jugador, `x` e `y` son números enteros. Con esta operación se indica que el jugador con nombre `jug` coloca una bandera en la casilla (`x`, `y`).
- `bordes`. Con esta operación se indica que se quiere acceder a los bordes del área del tablero.

Suponemos que los nombres de jugadores están formados por una única palabra, y que las posiciones de las casillas son números enteros contenidos en el rango $[-10^6, 10^6]$.

Cada caso de prueba finaliza con una cadena `FIN` que no realiza ninguna acción.

Salida

Para cada una de las operaciones se mostrará por pantalla una línea, cuyo contenido depende de la acción realizada:

- `mina`. Tras llamar al método correspondiente imprime por pantalla el número de minas que hay en el tablero.
- `jugador`. No imprime nada por pantalla.
- `explorar`. Si la casilla indicada contiene una mina, imprime la cadena `Has perdido`. En caso contrario, imprime el número de minas que existen en las casillas adyacentes a la indicada.
- `bandera`. Imprime la cadena `Si` (sin tilde) o `No`, dependiendo de si hay una mina en la casilla indicada o no. Esta cadena va seguida de un espacio en blanco, seguido de la puntuación del jugador correspondiente tras colocar la bandera.
- `bordes`. Imprime cuatro números enteros con los límites resultantes `min_x`, `min_y`, `max_x`, `max_y`, separados por un espacio.

En cualquier caso, si alguna de las operaciones produce una excepción, debe imprimirse una línea con el mensaje de error correspondiente.

Al final de cada caso de prueba debe imprimirse una línea con tres guiones (`---`).

Entrada de ejemplo

```
mina 1 1
jugador Guillermo
jugador Guillermo
explorar Guillermo 1 0
mina 2 1
explorar Guillermo 1 0
mina 3 2
bandera Guillermo 1 1
bandera Noexiste 1 1
bordes
FIN
mina -1 0
jugador Clara
explorar Clara -1 0
explorar Clara 0 0
FIN
```

Salida de ejemplo

```
1
Jugador existente
1
2
2
3
Si 1
Jugador no existente
1 1 3 2
---
1
Has perdido
Jugador no existente
---
```