

Documentación ejercicio propuesto (Buscaminas)

Índice

Objetivo.....	2
Práctica C.....	3
Estructuras de datos.....	4
Memoria dinámica.....	5
Algoritmos y funciones importantes.....	6
Compilación y ejecución.....	8
Mejoras futuras.....	10

Objetivo

El objeto del ejercicio es programar el juego **Buscaminas** o **Minefinder**.

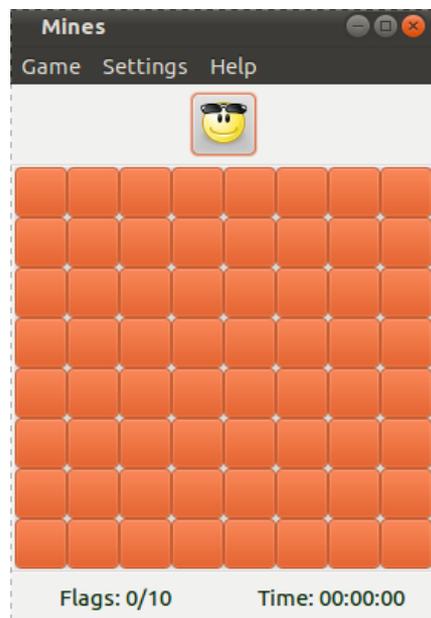
El juego consiste de un tablero cuadrado de ciertas proporciones compuesto por celdas ocultas, debajo de las mismas se encuentra distribuido un número determinado de bombas.

Existen celdas que contienen números, que representan cuantas celdas contiguas a estas contienen bombas.

Si se descubre (detona) una bomba se pierde el juego.

Cuando se considere que en una celda se encuentra una bomba deberá marcarse con una bandera.

La meta es marcar con banderas cada una de las celdas que contienen bombas, o bien, descubrir las restantes celdas libres de bombas.



Inicio del juego. Tablero con todas las casillas cubiertas

Elementos de C

En este ejercicio se emplean los siguientes conceptos o elementos de programación C:

- Directivas de preprocesador: macros
- Memoria dinámica
- Funciones recursivas
- Array multi-dimensionales
- Archivos de cabecera propios
- Funciones para la generación de números aleatorios

El programa consiste en sólo dos archivos

- **params.h**
con constantes simbólicas, typedefs y estructuras de datos
- **busca.h**
archivo de cabecera con la declaración de todas las funciones empleadas
- **busca.c**
Incluye a params.h y a busca.h
Utiliza las variables definidas en params.h como variables globales.
Define todas las funciones que implementan el juego

Total de líneas del programa: 373

Estructuras de datos

Se emplean dos arrays de caracteres multi-dimensionales, ambos con las dimensiones según la elección del usuario al inicio del juego.

Una opción sería reservar memoria para el tablero más grande posible y, en caso de que el usuario seleccione un tablero de menores dimensiones, no emplear parte de la memoria reservada. Esto no es lo adecuado.

En cambio, se opta por solicitar memoria dinámicamente, una vez que el usuario haya seleccionado el tipo de tablero.

Los dos array multi-dimensionales que se emplean son los siguientes y tiene el siguiente objeto:

- **sys_board:** es el tablero donde se establecen las bombas y los valores numéricos y es el que se emplea para verificar que se va descubriendo en el tablero del usuario.
Los valores posibles para las celdas de este tablero son:
 - . un caracter representando un número, del 0 al 8
 - . una bomba, representada con el caracter 'B'Al inicio se carga con todos valores '0'.

- **user_board:** es el tablero del usuario, donde se van descubriendo celdas o bien se establecen banderas.
Admite los siguientes valores:
 - . un caracter representando una celda descubierta con un número, del 1 al 8
 - . un caracter '#' indicando celda cubierta
 - . un caracter '_' indicando celda descubierta vacía
 - . un caracter 'F' señalando una bandera establecidaAl inicio se carga totalmente con valores '#'

Memoria dinámica

Dado que el usuario puede seleccionar jugar con uno de los siguientes tipos de tablero:

Tipo tablero	Filas	Columnas	Bombas
SMALL	8	8	8
MEDIUM	16	16	32
LARGE	24	24	64

Observar en el código fuente que estos valores están definidos en el archivo **params.h**.

En particular, debajo del comentario “Especificaciones de cada tipo de tablero”, y la tabla anterior en la matriz **settings**.

En `busca.c` la siguiente función es la que realiza la solicitud de memoria dinámica.

```
void create_board(BOARD *b) {
    int i;
    //solicita memoria dinamica para un array de N punteros a TYPE
    *b=(char**)malloc(sizeof(TYPE*)*N);
    for (i=0; i<N; i++) {
        //para cada posicion del array solicita memoria para N array s
        //de N TYPE's
        (*b)[i]=(char*)malloc(sizeof(TYPE)*N);
    }
}
```

En primer lugar observar el pasaje por referencia de la variable `b`, como puntero a `BOARD`.

La función solicita memoria para almacenar un array multi-dimensional con igual número de filas que de columnas, para ello en primer lugar solicita un array donde en cada celda se almacenará un puntero a un tipo determinado, luego, mediante el bucle `for`, se almacena en cada uno de estos punteros la dirección de memoria donde comienza un array de `N` elementos del mismo tipo.

Algoritmos y funciones importantes

Los pasos generales del programa son los siguientes:

- Recibir la preferencia del usuario en relación al tamaño del tablero
- Crear el tablero del sistema, donde se almacenan las bombas y números, servirá de referencia para consultas sobre posiciones determinadas
- Cargar en posiciones aleatorias la cantidad de bombas que corresponda para el tablero seleccionado
- En cada carga de bomba calcular los valores numéricos de las celdas limítrofes
- Crear el tablero del usuario con el cual jugará
- Consultar por posición que el usuario quiere descubrir o bien establecer una bandera
- Bucle principal
 - descubrir celdas, si la celda esta vacía intentar descubrir más celdas
 - si se detona bomba terminar, mostrar mensaje y solución
 - si se establece el máximo numero de banderas permitido verificar si cubren todas las bombas, si es así ganó, en caso contrario mostrar mensaje y solución
 - si se descubren todas las celdas sin bombas, ganó
- Liberar recursos de memoria solicitados, finalizar

Se muestran las funciones de carga de bombas en el tablero del sistema y la de descubrimiento de celdas.

Carga de bombas

```
/*
Carga las bombas en un tablero. La seleccion de las celdas es aleatoria.
Termina al finalizar la carga del numero de bombas corresp al tablero elegido
*/
void load_bombs(BOARD b) {
    int bombs=0; //contador de bombas
    int row, col;

    while (bombs < settings[PREF_SIZE][BOMB]) { //mientras falten bombas por establecer
        row = rand() % settings[PREF_SIZE][ROW]; //se elige fila aleatoriamente
        col = rand() % settings[PREF_SIZE][COL]; //se elige columna aleatoriamente

        if (is_a_bomb(b,row,col)==TRUE) //si hay una bomba alli ignorar y
            continue; //buscar nueva posicion

        insert_bomb(b,row, col); //insertar la bomba
        compute_numbers(b,row,col); //(re)calcular los valores cercanos a las bombas
        bombs++; //sumar una nueva bomba
    }
}

/*
Calcula los valores numericos cercanos a las bombas en un tablero dado
Recorre todas las celdas validas que rodean a la posicion indicada
*/
void compute_numbers(BOARD b, int r, int c) {
    int i,j;
    for (i=r-1; i<=r+1; i++) {
        if (i<0 || i>settings[PREF_SIZE][ROW]-1) continue;
        for (j=c-1; j<=c+1; j++) {
            if (j<0 || j>settings[PREF_SIZE][COL]-1) continue;
            if (is_a_bomb(b,i,j) == FALSE) //si no es una bomba, en este tablero es un nro
                b[i][j]++; //modifica el valor numerico sumandole uno
        }
    }
}
```

Analista Universitario en Sistemas Taller de Programación II

Lic. Diego A. Bottallo

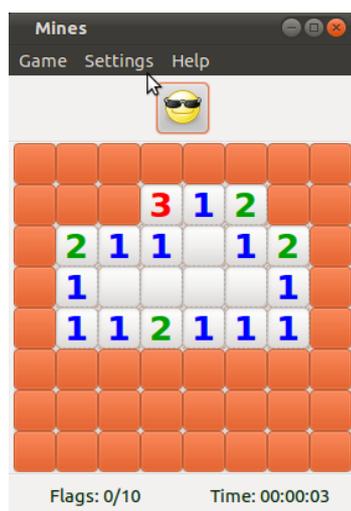
Observar que una vez seleccionada la posición donde se ubicará una bomba se recorren todas las celdas limítrofes para modificarlas, por supuesto considerando sólo las celdas que pertenecen al tablero y donde no hay una bomba establecida previamente.

Descubrimiento de celdas

```
/*
Descubre la celda elegida y realiza un recorrido por las celdas vecinas
Recursion, se auto-invoca para las celdas vacias vecinas
*/
USINT discover(int r, int c) {
    int i,j;
    if (!is_hidden(user_board,r,c)) return 0; //es celda descubierta en tablero usuario, volver
    if (is_a_bomb(sys_board,r,c)) {          //chequear si hay una bomba alli
        return BOOM;                          //se retorna la condicion correspondiente
    }
    if (is_a_number(sys_board,r,c)) {        //si es un numero en el tablero del sistema
        user_board[r][c] = sys_board[r][c];   //se establece ese valor el tablero del usuario
        free_cells++;                          //se incrementa el numero de celdas descubiertas
        return 0;
    }

    if (is_a_flag(user_board,r,c)) return 0; //si es una bandera no hacer nada

    //el paso recursivo, aplica cuando la celda esta vacia
    user_board[r][c] = EMPTY;                 //descubrir la celda
    free_cells++;                             //se incrementa el numero de celdas descubiertas
    for (i=r-1; i<=r+1; i++) {
        if (i<0 || i>=settings[PREF_SIZE][ROW]) continue; //si se va de los bordes, ignorar
        for (j=c-1; j<=c+1; j++) {
            if (j<0 || j>=settings[PREF_SIZE][COL]) continue; //si se va de los bordes, ignorar
            if (i==r && j==c) continue; //si es la misma celda, ignorar
            if (is_hidden(user_board,i,j)) {
                //chequear solo si es una celda cubierta en el tablero de usuario
                discover(i,j); //RECURSION: invocacion recursiva a discover
            }
        }
    }
    return 0;
}
```



En la imagen anterior se puede observar un área de celdas vacías descubiertas, que se abre en caso, por ejemplo, si se elige la posición en fila 4 y columna 5. El descubrimiento de las celdas continua mientras se sigan detectando celdas vacías contiguas y el contorno del área descubierta lo marcarán números y/o banderas previamente establecidas.

Compilación y ejecución

Para compilar el programa realice las siguientes acciones:

- 1) Descomprima en un directorio el archivo **buscaminas.tgz**
- 2) Acceda al directorio generado
cd BUSCAMINAS
- 3) Compile
gcc -Wall busca.c -o buscaminas
- 4) Ejecute
./buscaminas

Ejemplo de ejecución:

Ingrese el tablero de su preferencia:

- 0) SMALL
- 1) MEDIUM
- 2) LARGE
- 9) QUIT

Preferencia: 0

```
  1  2  3  4  5  6  7  8
1 # # # # # # # #
2 # # # # # # # #
3 # # # # # # # #
4 # # # # # # # #
5 # # # # # # # #
6 # # # # # # # #
7 # # # # # # # #
8 # # # # # # # #
```

Flags 0 de 8 - Celdas descubiertas 0 de 56

Ingrese fila: 8

Ingrese columna: 7

Ingresar una bandera? (s/n): n

```
  1  2  3  4  5  6  7  8
1 # # # # # # # #
2 # # # # # # # #
3 # # # # # # # #
4 # # # # # # # #
5 # # # # 2 2 3 #
6 # # # # 1 - 1 #
7 # # # 3 1 - 1 1
8 # # # 1 - - -
```

Flags 0 de 8 - Celdas descubiertas 47 de 56

Analista Universitario en Sistemas
Taller de Programación II

Lic. Diego A. Bottallo

La imagen siguiente muestra un tablero resuelto, donde todas las bombas fueron marcadas con banderas.

En nuestro sistema se vería del siguiente modo:

```
Ingrese fila: 1
Ingrese columna: 2
Ingresa una bandera? (s/n): s
  1  2  3  4  5  6  7  8
1 #  F  1  -  -  -  -  -
2 #  3  2  2  1  1  -  -
3 1  2  F  2  F  1  -  -
4 1  3  2  3  1  1  -  -
5 F  2  F  1  -  -  -  -
6 1  2  1  1  -  1  1  1
7 -  -  -  1  1  2  F  1
8 -  -  -  1  F  2  1  1
Flags 7 de 8 - Celdas descubiertas 55 de 56
```



```
Ingrese fila: 1
Ingrese columna: 1
Ingresa una bandera? (s/n): n
Gano ;) !!
  1  2  3  4  5  6  7  8
1 2  B  1  0  0  0  0  0
2  B  3  2  2  1  1  0  0
3 1  2  B  2  B  1  0  0
4 1  3  2  3  1  1  0  0
5  B  2  B  1  0  0  0  0
6 1  2  1  1  0  1  1  1
7 0  0  0  1  1  2  B  1
8 0  0  0  1  B  2  1  1
```

En este ejemplo el usuario descubrió todas las celdas que no tenían bomba, logrando el objetivo.

Mejoras futuras

Se podrían agregar funcionalidades con el fin de obtener un mejor sistema y mejorar la experiencia de juego, por ejemplo:

- una interfaz gráfica para que el usuario pueda emplear un mouse
- tiempo de juego a cada partida, con el fin de confeccionar un ranking (por ejemplo, top ten de mejores tiempos) para cada tipo de tablero