TEC | Tecnológico de Costa Rica

Tecnológico de Costa Rica

Centro académico de Limón

Randall Corella Castillo

Aarón Piñar Mora

Johan Vargas Quesada

Proyecto: Busca Minas

Estructuras de datos

Grupo: 22-IC3101

Profesor: David Mora

2022, I semestre

Contenido

Objetivo Principal

Desarrollar un programa en consola que permita jugar el famoso juego Buscaminas por medio de programación a bajo nivel en una arquitectura RISC.

Implementación

1. El programa se debe ejecutar en consola de comandos, no debe tener interfaz gráfica ni se debe interactuar con el usuario más que con el teclado y la impresión en pantalla.
2. En línea con las reglas, el juego debe permitir modificar el ancho, alto y cantidad de minas en el campo antes de iniciar, pudiendo escoger entre los tres modos predeterminados o una configuración personalizada con las siguientes restricciones: un tamaño máximo de 100 en cualquier dirección y una cantidad de minas que no sobrepase el 50% de los campos disponibles.
3. En todo momento debe desplegarse la siguiente información:

    - Configuración del campo de juego actual con los campos descubiertos, campos con número y marcas de bandera.
    - Estado actual del juego (minas restantes por marcar y otras estadísticas sobre el campo de juego).

Tecnología

Para el desarrollo de este programa se debe utilizar lenguaje ensamblador para arquitecturas RISC por medio de Raspbian OS. Esto se puede llevar a cabo por medio de dos alternativas: directamente en un dispositivo Raspberry Pi o por medio del emulador QEMU; ambas alternativas serán aceptadas. Este compilador es el que se debe utilizar a pesar de existir otras alternativas. No se pueden utilizar funciones de C o C++ directamente en el código del programa, para ejecutar funciones elevadas, únicamente se puede hacer por medio de llamadas a servicios del sistema opera

Desarrollo

Abstracción de las reglas

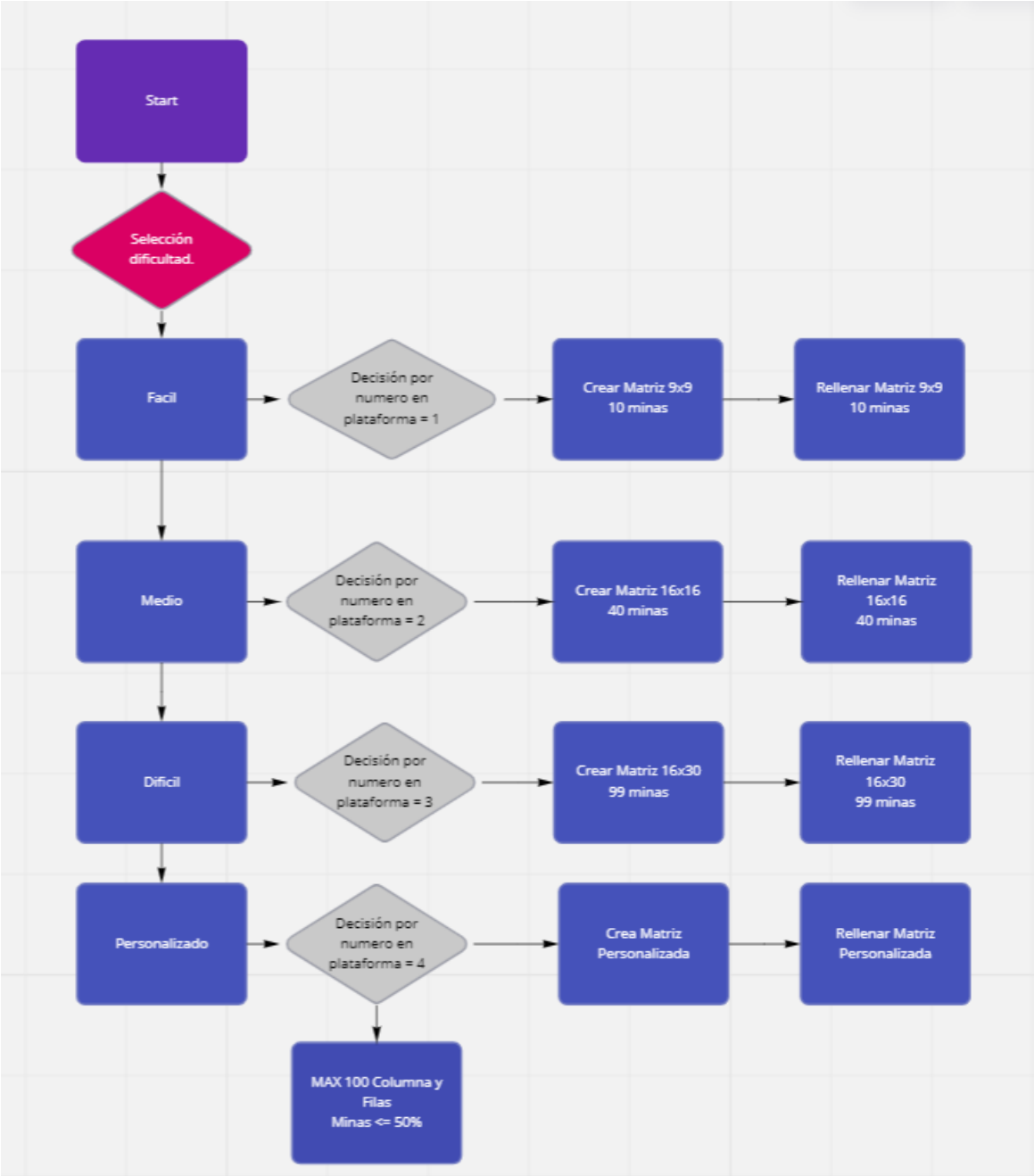
Para jugar al buscaminas hay que tener en cuenta las siguientes reglas:

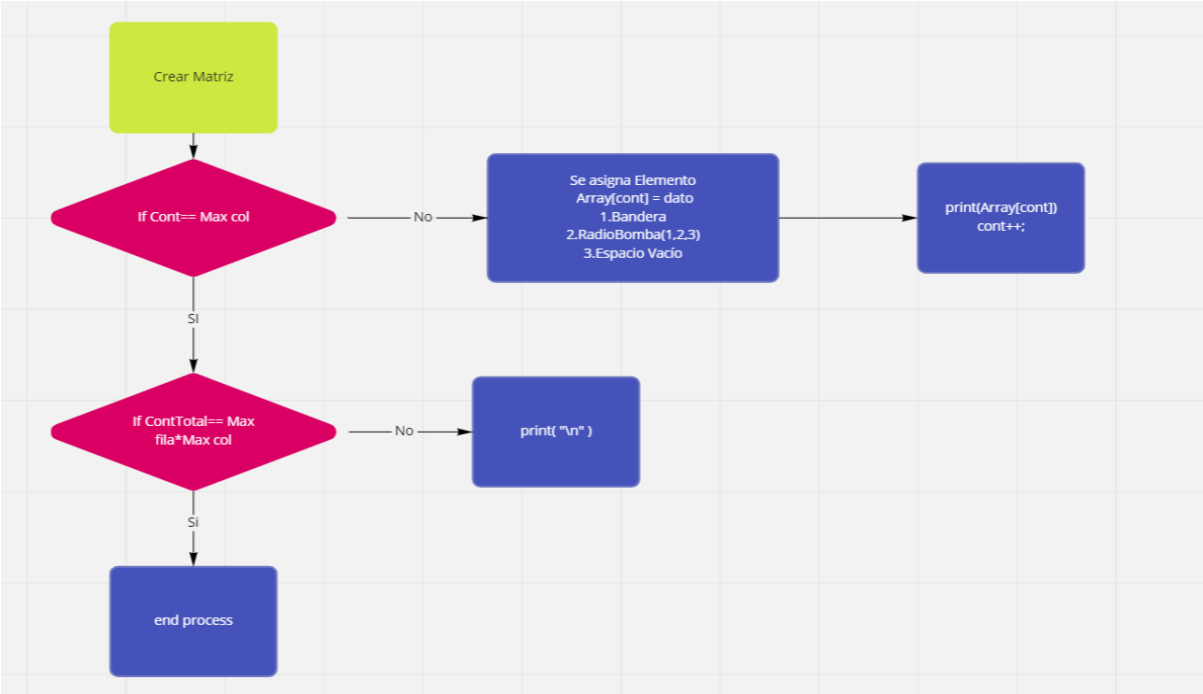
- Al dar click se abren todas las casillas sin Bombas (Para efectos del proyecto, será al indicar el espació en la matriz)
- Existen 8 tipos de representaciones diferentes de casillas: Bomba, Casilla vacía, Casilla sin descubrir, Bandera, Interrogante, números de cercanía de una bomba 1,2 y 3.

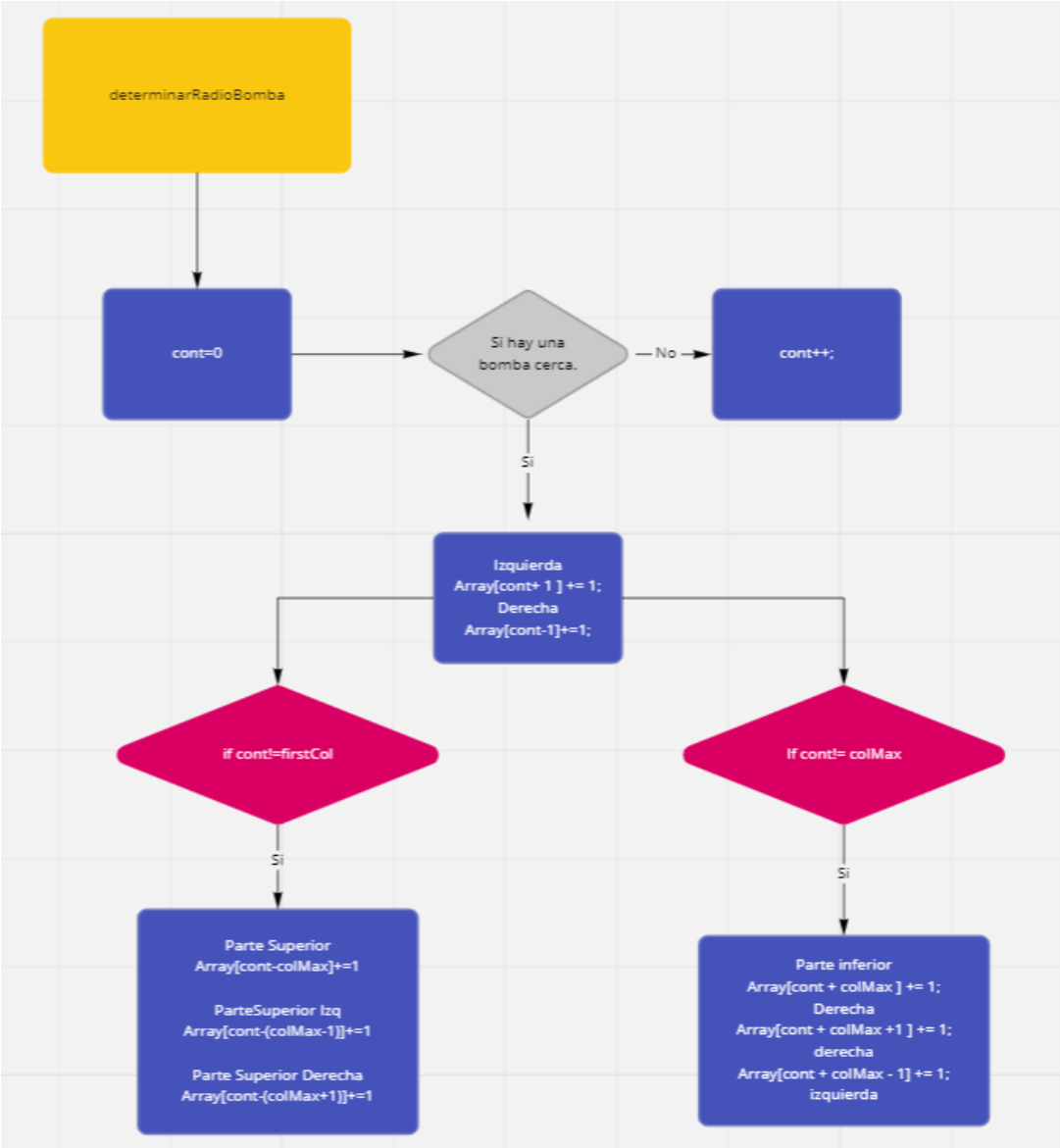
Dimensiones en base al nivel de dificultad:

- Nivel principiante: 9 × 9 casillas y 10 minas.
- Nivel intermedio: 16 × 16 casillas y 40 minas.
- Nivel experto: 16 × 30 casillas y 99 minas.
- Nivel personalizado: en este caso el usuario personaliza su juego eligiendo el número de minas y el tamaño de la cuadricula

Planteamiento en UML




Crear Matriz
If Cont== Max col
No
Se asigna Elemento
Array[cont] = dato
1.Bandera
2.RadioBomba(1,2,3)
3.Espacio Vacio
print(Array[cont])
cont++;
SI
If ContTotal== Max
fila*Max col
No
print("\n")
Si
end process


determinarRadioBomba
cont=0
Si hay una bomba cerca.
No
cont++;
Si
Izquierda
Array[cont+ 1] += 1;
Derecha
Array[cont-1]+=1;
if cont!=firstCol
If cont!= colMax
Si
Si
Parte Superior
Array[cont-colMax]+=1
ParteSuperior Izq
Array[cont-(colMax-1)]+=1
Parte Superior Derecha
Array[cont-(colMax+1)]+=1
Parte inferior
Array[cont + colMax] += 1;
Derecha
Array[cont + colMax +1] += 1;
derecha
Array[cont + colMax - 1] += 1;
izquierda

Algoritmos

Largo de un input (R4 – R5)

Se carga en R4 el byte de R1 en la posición de R3

Si R4 = = \n

Se finaliza

Indica el Largo del input

Si no

Se añade un byte en R1

Se llama Largo de un Input

```
11  largo:
12      PUSH    { R4 - R5, lr }
13
14  largoInput:
15
16      LDRB    R4,     [ R1, R3 ]  @ Carga en R4 el byte de R1 en la posicion que de R3
17      CMP     R4,     #0xa        @ Compara cuando R4 sea igual a \n
18      BEQ     largoInputEnd       @ Si se cumple lo dirigue a validarLen
19      ADD     R3,     #1          @ Cada vez que se cargue un byte de R1 se va a sumar 1 al contador de bytes donde se guarda la posicion
20      B       largoInput          @ Repite el proceso
21
22  largoInputEnd:
23      MOV     R0,     r5          @Se mueve el valor obtenido a R0 para luego acceder a el
24      POP     { R4 - R5, lr }     @->R5:Indica el largo del input
25      MOV     pc,     lr
26
```

Crear Modo de juego (En base a la dificulta)

Se crea:

Matriz: X = columna Y= fila (en base a la dificultad varia y si es perso se ingresan)

Se coloca bombas (Con valores segun la dificultada)

Se coloca Números (Con valores segun la dificultada)

Se crea la matriz a printear

Contar caracteres (recibe un puntero con una cadena de bytes)

R4 = contador inicializado en 0

Mientras que la posición sea diferente a nulo

Se agrega 1 al contador

Hace la llamada recursiva a la función

Si no

Se finaliza la función

Se almacena la cantidad de caracteres en r4 y se retorna

```
fLU01_contar_caracteres:
    PUSH {r4, r5, lr}
    MOV r4, #0 // r4: contador.
//
fLU01_while:
    LDRB r5, [r0,r4] //Posicion actual de la cadena.
    CMP r5, #0 //Comparamos con caracter nulo.
    BEQ fLU01_while_end
    ADD r4, #1 //Agregamos al contador.
    BAL fLU01_while

fLU01_while_end:
    MOV r0, r4
    POP {r4, r5, pc} //lr a pc para volver a donde se llamo esta funcion.
```

Dividir

Contador inicializado en 0

Acumulador inicializado en 0

Mientras el dividiendo sea mayor al divisor

Se resta con el acumulador

Añade 1 al contador

Se agrega el acumulador el divisor

Retorna el resultado de la división

```
fLU04_dividir:
    PUSH {r4 - r6, lr}
    MOV r4, #0 // Contador.
    MOV r5, #0 //Acumulador.

fLU04_while: //ciclo division.
    MOV r6, r0 //Guardar el dividendo.
    SUB r6, r5 // Resta para comparar.
    CMP r6, r1 //Comparamos si es menor al divisor.
    BLT fLU04_end //Finaliza el proceso.
    ADD r4, #1 //Añade 1 mas al contador.
    ADD r5, r1 //Agregamos al acumulador el divisor.
    BAL fLU04_while

fLU04_end: //Finaliza la division.
    MOV r0, r4
    POP {r4 - r6, pc}
```

Conseguir el residuo de la división de R0 Y R1

Contador = 0

Accumulator = 0

Mientras el dividendo sea mayor al divisor

se resta con el acumulador

añade 1 al contador

Se agrega el divisor al acumulador

Se llama así misma

Finaliza y retorna cociente

```
//->r0: dividendo.
//->r1: divisor
//<-r0: cociente resultado de la division o el residuo entre r0/r1.
fLU06_modulo:
    PUSH {r4 - r6, lr}
    MOV r4, #0 // Contador.
    MOV r5, #0 //Acumulador.

fLU06_while: //ciclo division.
    MOV r6, r0 //Guardar el dividendo.
    SUB r6, r5 // Resta para comparar.
    CMP r6, r1 //Comparamos si es menor al divisor.
    BLT fLU06_end //Finaliza el proceso.
    ADD r4, #1 //Añade 1 mas al contador.
    ADD r5, r1 //Agregamos al acumulador el divisor.
    BAL fLU06_while

fLU06_end: //Finaliza la division.
    MOV r0, r6
    POP {r4 - r6, pc}
```

Crear Matriz (Recibe un valor entero y cantidad de datos a guardar)

Contador = 0

Mientras el contador sea diferente a la cantidad de datos a ingresar

- Se ingresa el valor para obtener dirección de memoria
- Se agrega 1 a la dirección de memoria
- Se agrega 1 al contador
- Se llama así misma

Retorna el puntero con la matriz

```
//recibe un valor entero y lo almacena en la matriz.
//->r1: el dato a ingresar.
//->r2: la cantidad de datos que deseamos guardar.
//<-r4: retorna el puntero que señala al primer elemento de la secuencia se bytes.
fM01_crear_matriz:
  PUSH {r5 - r7, lr}
  //Obtenemos la posicion actual del final del segmento de datos.
  MOV   r0, #0 //Aqui se ingresa el valor 0 para obtener la direccion del final de la memoria.
  MOV   r7, #45 //System call BRK.
  SVC   #0      //Interrupcion del sistema.
  //Guardamos la posicion de la secuencia de bytes.
  MOV   r4, r0 //Movemos a r4 el puntero que señala la posicion inicial de la matriz.
  MOV   r3, #0 //Inicializamos el contador.

//Funcion de iteracion para crear la matriz.
fM01_while01:
  CMP r2, r3 //comparamos el contador con la cantidad de datos a ingresar.
  BEQ fM01_while01_end //Saltamos al final.

  MOV   r0, #0 //Aqui se ingresa el valor 0 para obtener la direccion del final de la memoria.
  MOV   r7, #45 //System call BRK.
  SVC #0 //Interrupcion.

  ADD r0, #1 //Agregamos 1 a la direccion de memoria actual para pedir mas espacios en memoria.
  MOV r7, #45 // Systema call BRK.
  SVC #0 //Interrupcion y incrementa la memoria del segmento de datos.

  //
  STRB  r1, [r4, r3] //Insertamos en r4 posicion r5, el dato ingresado.
  ADD   r3, #1 //Agregamos 1 al contador de datos.
  B fM01_while01 //Llamada iterativa.
```

Printar Matriz (cantidad de datos a printar, tamaño de fila, puntero)

Mientras la cantidad de datos a printar sea diferente a la cantidad evaluada

Mientras contador sea diferente al tamaño de fila

Se imprime el carácter

Imprime el salto de línea

Imprime el carácter

Añade 1 al contador

Y al contador de evaluados

Se llama así misma

Se finaliza y printa la matriz

```
//Se encarga de printar la matriz
//->r1: Recibe la cantidad de datos que debemos printar
//->r2: Recibe el tamaño de la fila.
//->r4: recibe el puntero que vamos a recorrer.
fM02_printar_matriz:
  PUSH {r5 - r7, lr}
  MOV r3, #0  //contador para saber cuantos han sido evaluados
  MOV r5, #0
//Iteracion para printar la matriz.
fM02_while02:
  CMP r1, r3
  BEQ fM02_while_end
  CMP r5, r2
  BEQ fM02_imprimir_salto

  LDRB r0,  [r4,r3]
  ADD r0, #48
  BL fLU03_imprimir_caracter

  ADD r3, #1
  ADD r5, #1
  B fM02_while02
//Finaliza el programa.
fM02_while_end:

  POP {r5 - r7, pc}
```

Revela casilla por la que se envió (matriz, matriz de printeo, posiciónABuscar, cantidad maxima)

Se verifica en la matriz en base a la posición para saber si ya está revelado

Si no se revela por los bloque en base a la posición de la matriz, medio derecha, medio izquierda, etc..

```
fM03_revelar_casillas:
  PUSH {R6, R8 ,lr}
  //============================Good
  LDRB R1, [R4, R6]
  //Comparamos la casilla que eligio.
  CMP R1, #6
  BEQ fM03_end
  //verificamos lo que hay en la matriz en esa posicion
  LDRB R0, [R5, R6]
  CMP R0, #0 //CMP para saber si ya esta revelada.
  BGT fM03_end
  MOV R0, R1 //guardamos el valor de la casilla.
  CMP R1, #0
  MOVEQ R0, #9
  STRB R0, [R5, R6]
  CMP R1, #0 //CMP con 0 para saber si hacer la recursion.
  BEQ fM03_while_revelar
  B fM03_end
  //============================================================
  //Si no es un 0, solo la revelamos a ella.
fM03_revelar_casilla:

  STRB R1, [R5, R6]
  BGE fM03_end
```

Colocar Bombas (Cantidad Bombas, fila, columna, puntero, contenedor)

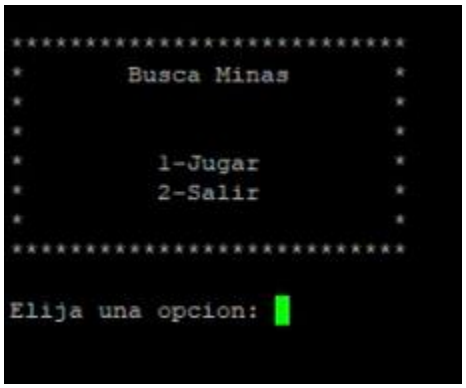
Se cuentan las bombas actuales con las totales (No se puede exceder)

Se genera una fila y columna al azar para multiplicarlas

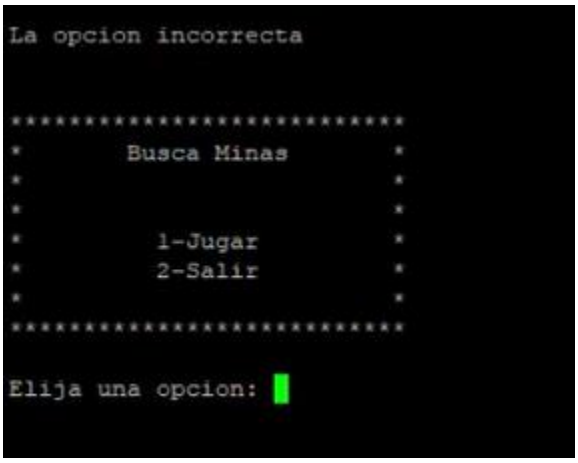
Se coloca la bomba en la ubicación dada por la multiplicación

Manual de Usuario

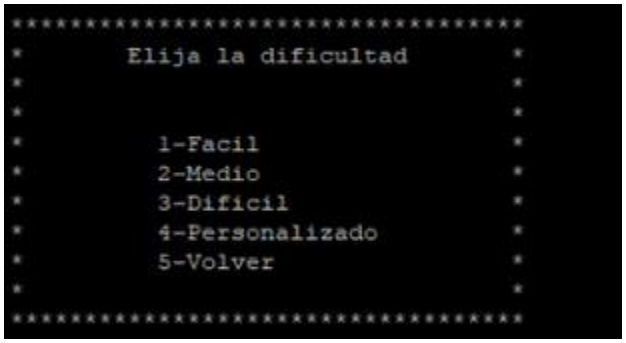
Al inicializar el usuario se encontrará con un menú, donde debe de indicar una de las opciones mostradas en consola.



Si el usuario no indica una opción mostrada se le brindara un mensaje a consola y se refrescara.

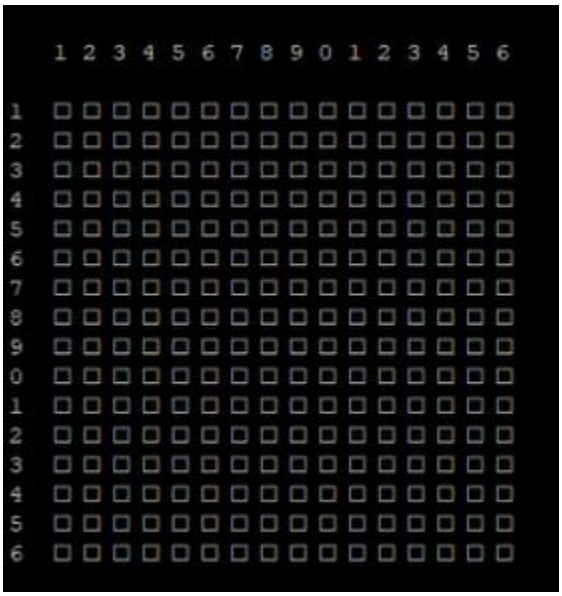


Si el usuario selecciona jugar se mostrará en pantalla los niveles de dificultad para generar el juego.



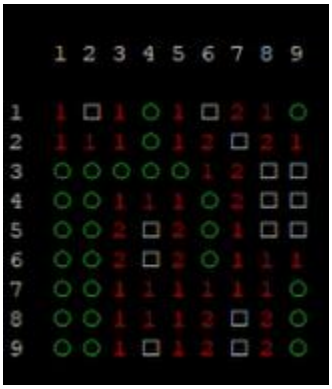
Si el usuario selecciona volver lo devuelve a la sección anterior. Si el usuario elije una modalidad de juego, genera un tablero con las medidas del nivel. De igual manera si el usuario escribe un numero no mostrado en pantalla se le indicara con un mensaje que debe seleccionar uno de los mostrados y se refrescara.

El tablero se verán los espacios posibles representados de cuadrados y a su alrededor tendrá de guía una línea de números para identificar con mayor facilidad la ubicación. El usuario deberá de escribir primero la fila y luego la columna. El programa siempre pedirá esos dos valores.



Tablero Generado de juego medio

En el momento que el usuario indique fila y columna, se le mostraran los espacios vacíos al usuario con un símbolo redonde verde. También, se indicará la aproximación de las bombas con números en rojo.

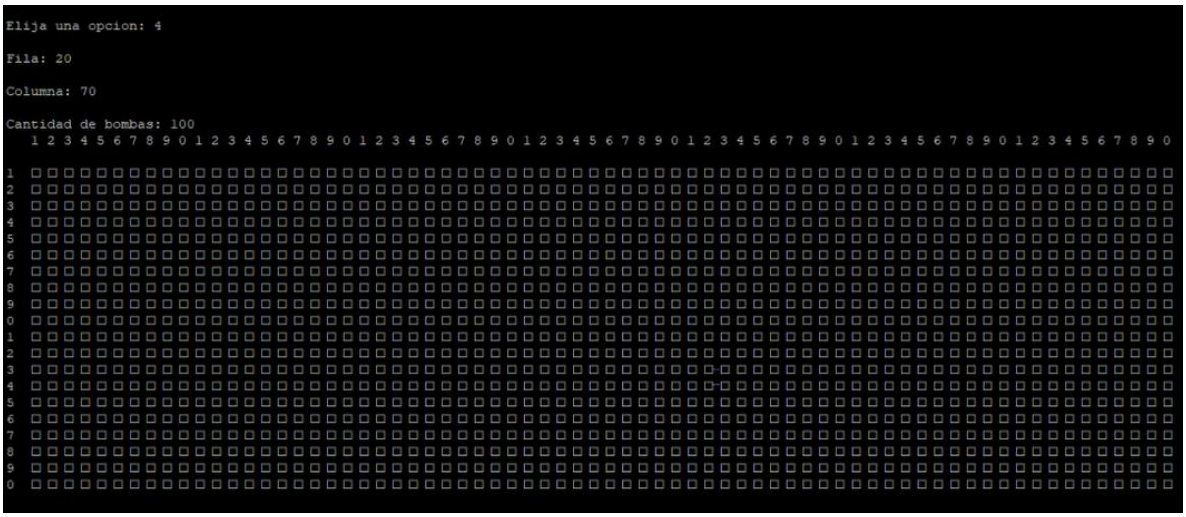


Si el usuario logra completar todas las casillas sin tocar ninguna bomba se le mostrara felicidades, en representación de que ha ganado

Pero si el usuario coloca la ubicación de una bomba se le avisara que perdió

Si el usuario selecciona 4 Se le pedirá los siguientes datos para crear el juego

Dependiendo de los valores ingresados se creará el tablero de juego



Su funcionalidad será la misma dicha con anterioridad