**Laboratorio Organización de Computadores**

**“Busca Minas”**

|  |  |
| --- | --- |
| Integrante : | Diego Miranda Gipoulou |
| Profesor : |  |
| Ayudante |  |
| Fecha de entrega : |  |

Tabla de Contenidos

[CAPÍTULO 1. Introducción 3](#_Toc439721254)

[1.1 Antecedentes y motivación 3](#_Toc439721255)

[1.2 Descripción del problema 3](#_Toc439721256)

[1.3 Solución propuesta 3](#_Toc439721257)

[1.4 objetivos y alcances del proyecto 3](#_Toc439721258)

[1.4.1 Objetivo general 3](#_Toc439721259)

[1.4.2 Objetivos específicos 3](#_Toc439721260)

[1.4.3 Alcances 3](#_Toc439721261)

[1.5 Metodologías y herramientas utilizadas 3](#_Toc439721262)

[1.6 Organización del documento 4](#_Toc439721263)

[1.7 El juego 4](#_Toc439721264)

[1.7.1 ¿Cómo se juega en esta versión? 4](#_Toc439721265)

[1.7.2 ¿Cuándo se gana? ¿Cuándo se pierde? 4](#_Toc439721266)

[CAPÍTULO 2. Desarrollo 4](#_Toc439721267)

[2.1 Fundamentos teóricos y estado del arte del dominio del problema 4](#_Toc439721268)

[2.1.1 Syscall 4](#_Toc439721269)

[2.1.2 Memoria 5](#_Toc439721270)

[2.1.3 Stack pointer 5](#_Toc439721271)

[2.1.4 Registros 5](#_Toc439721272)

[2.2 Descripción de los métodos utilizados 5](#_Toc439721273)

[2.3 Desarrollo de la solución 5](#_Toc439721274)

[2.3.1 Análisis 5](#_Toc439721275)

[2.3.2 Diseño 6](#_Toc439721276)

[2.3.3 Implementación 6](#_Toc439721277)

[2.3.4 Pruebas 6](#_Toc439721278)

[2.3.5 Uso 6](#_Toc439721279)

[2.4 Discusión de los resultados 6](#_Toc439721280)

[CAPÍTULO 3. Conclusion 6](#_Toc439721281)

[3.1 Fallas 6](#_Toc439721282)

[3.1.1 Victoria al llenar el tablero con banderas 6](#_Toc439721283)

[3.1.2 Creacion de vista mayor a dimension maxima 6](#_Toc439721284)

[3.1.3 Permite que la cantidad de bombas sea mayor a la cantidad de casillas 6](#_Toc439721285)

[3.2 Mejoras 6](#_Toc439721286)

[3.2.1 Conteo de banderas puestas 6](#_Toc439721287)

[3.2.2 Posibilidad de ingresar cualquier archivo de texto 6](#_Toc439721288)

[CAPÍTULO 4. Bibliografía 7](#_Toc439721289)

# Introducción

Todo software tiene como base una programación binaria, algo más alto nivel está el lenguaje *assembler*, algo similar a esto es MIPS.

Se propuso realizar un juego con MARS, un entorno para programar en el lenguaje de ensamblado MIPS.

## Antecedentes y motivación

Principalmente, considero este proyecto un reto, tanto de aprendizaje como de logros personales, además me enfrento a un nuevo lenguaje y a un nuevo paradigma lo cual me motiva a trabajar en este software.

## Descripción del problema

Se requiere simular el juego Busca Minas, desde ahora también el juego, logrando implementar la parte visual y funcional del juego, donde es de libre elección la parte visual, es decir, puede o no corresponder al juego clásico

## Solución propuesta

Se desarrolla un software de una versión modificada del Busca Minas, implementando la parte visual y funcional del juego con MARS, interprete del lenguaje de ensamblado MIPS, donde el modo de juego estará limitado por entradas por teclado.

## objetivos y alcances del proyecto

### Objetivo general

Simular el juego Busca Minas

### Objetivos específicos

Utilizar MIPS para la generación funcional del juego

Utilizar MIPS para la visualización del juego

### Alcances

Como parte del trabajo, sólo se limita a la funcionalidad que entrega MIPS, excluye la de MARS, por ende no se utilizan métodos propios de este último. Así mismo, se excluye de este trabajo todo complemento visual que no sea propio de MARS.

## Metodologías y herramientas utilizadas

Para el desarrollo del software se programa como si fuese un método de programación extrema, es decir, se realiza en la marcha e intentando generar etapas en el corto plazo.

Para la programación se utiliza MARS intérprete del ensamblador MIPS

## Organización del documento

## El juego

El juego consiste en un “busca minas” (*Minesweeper*), juego que se volvió famoso con Windows 3.1.Este consiste en que el jugador debe encontrar las minas ocultas en un tablero que muestra números que representan la cantidad de bombas en los cuadrados adyacentes a él.

### ¿Cómo se juega en esta versión?

En esta versión similar a la original, la condición de victoria se da por colocar las banderas en las casillas donde hayan bombas, no si están todas las casillas que no son bombas abiertas, además solo se manejan las casillas mediante consola es decir para abrir casillas se hace mediante la consola al igual que colocar banderas.

### ¿Cuándo se gana? ¿Cuándo se pierde?

En este juego se ha decidido no imitar el busca minas, se gana cuando cada bomba en el tablero tiene una bandera puesta en él, en cambio, en el juego original, se gana una vez que se abren todas las posiciones con números,

# Desarrollo

El funcionamiento del tablero se hace mediante 2 capas, la vista la cual ve el usuario y otro tablero interno que contiene las posiciones y bombas, para jugar el programa solicitara el método de ingreso del laberinto, los cuales pueden ser aleatorio o por texto, si es por texto se debe tener un archivo de texto de nombre único, llamado “lectura.txt”

Para la condición de victoria deben estar puestas las banderas en las bombas (pese a que originalmente el juego no es así), y para saber si se gana cambian las caras de las esquinas al igual que al perder cambian las caras en las esquinas.

Ideas

## Fundamentos teóricos y estado del arte del dominio del problema

### Syscall

Se utiliza para métodos de I/O, entrada y salida, con el cual se puede leer o escribir en consola o en un documento de texto.

Los *syscall* utiliza el registro $v0 como entrada y algunos otros dependiendo de cuál sea la función.

### Memoria

Es el espacio donde se almacenan memoria, en MIPS se utiliza la palabra .data como identificador de elementos guardados en memoria.

### Stack pointer

Tipo de memoria que funciona como una pila, que sirve para almacenar datos temporalmente.

### Registros

#### Temporales

Los registros temporales son aquellos que son de la forma $tn donde n es un numero desde 0 hasta 9.

#### Globales

Los registros globales son aquellos que son de la forma $sn donde n es un numero desde 0 hasta 7

#### Entrada a funciones

Los registros que se utilizan como entrada a funciones son de la forma $an donde n es un numero desde 0 hasta 4.

#### Salida de funciones

Los registros que se utilizan como salida de las funciones son de la forma $vn donde n puede ser 0 o 1.

## Descripción de los métodos utilizados

## Desarrollo de la solución

### Análisis

El usuario debe poder ver un tablero

El usuario debe poder interactuar, cargar tablero o generar aleatorio

El usuario debe poder colocar banderas, sacar banderas, abrir casillas

Debe haber una condición de victoria y de perdida

### Diseño

Forma del tablero interno

El tablero interno o mascara, está compuesto lógicamente por nodos, en realidad compuestos por tres espacios de memoria, que contienen posición x, correspondiente a las filas, posición y, correspondiente a las columnas y el dato correspondiente al número de bombas, de 1 a 8 o 9 si es bomba.

Mostar una imagen

Mostar una fila

Mostar tablero

Colocar imagen en posición

Agregar bomba tablero

Generar tablero aleatorio

Cambiar dato

Leer un archivo

Rellenar tablero

Abrir casillas tablero

Colocar y sacar banderas

### Implementación

### Pruebas

### Uso

## Discusión de los resultados

# Conclusión

Se ha logrado el objetivo de programar el juego y lograr un funcionamiento satisfactorio, se conoce gran parte de las fallas y se tiene un conocimiento de como arreglarlos.

## Fallas

Las fallas que posee el programa son:

### Victoria al llenar el tablero con banderas

Si se desea se puede colocar banderas en todo el tablero y llegara a la victoria, sin siquiera jugar, esta es una falla producida por la condición de victoria que considera solo que las bombas posea banderas, pero no considera que la cantidad de banderas sea igual a la cantidad de bombas.

### Creacion de vista mayor a dimension maxima

Es posible crear un tablero mayor a la dimensión de 16x16 establecida, esto puede ser solucionado fácilmente con un *branch.*

### Permite que la cantidad de bombas sea mayor a la cantidad de casillas

Al igual que en el punto anterior es fácil solucionar con un *branch*, si se replica esta falla el programa entrara en un ciclo recursivo infinito.

## Mejoras

### Conteo de banderas puestas

Mejora que se debe implementar para hacer la comparación de banderas y resolver la falla 4.1, el problema está al descontarla, es decir, al querer sacar una bandera.

### Posibilidad de ingresar cualquier archivo de texto

Una mejora no implementada debido a problemas desconocidos por mí, desconozco la dificultad, pero esta implementado correctamente la lectura de archivo.

# Bibliografía

Missouri State University. (28 de 08 de 2014). *Missouri State University.* Obtenido de http://courses.missouristate.edu/KenVollmar/MARS/