|  |
| --- |
| **Paradigmas de programacion**  **Informe** |
|  |
| **Andres Muñoz Bravo**  **19646487-5** |

|  |  |
| --- | --- |
|  | Profesor: |
|  | Roberto Gonzales Ibañes |
|  |  |
|  |  |
|  |  |

|  |  |  |
| --- | --- | --- |
|  | Santiago - Chile |  |
|  | 2016 |  |

Tabla de Contenidos

[Tabla de Contenidos I](#_Toc462437007)

[Índice de Figuras I](#_Toc462437008)

[Índice de Tablas I](#_Toc462437009)

[CAPÍTULO 1. Introducción 3](#_Toc462437010)

[CAPÍTULO 2. Analisís del problema 4](#_Toc462437011)

[CAPÍTULO 3. Diseño de la solución 5](#_Toc462437012)

[CAPÍTULO 4. Implementación 7](#_Toc462437013)

[CAPÍTULO 5. Resulados 7](#_Toc462437014)

[CAPÍTULO 6. Conclusión 7](#_Toc462437015)

[CAPÍTULO 7. REFERENCIAS 7](#_Toc462437016)

Índice de Figuras

[Ilustración 1: Archivos del programa 5](file:///C:\Users\Andres\Desktop\Programacion\C\Paradigmas\Paradigmas-C\Informe.docx#_Toc462436951)

[Ilustración 6: Archivo con la información de un tablero 6](#_Toc462436952)

Índice de Tablas

Tabla de porcentajes de las funcinones realizadas………………………………………7

# Introducción

Para esta primera experiencia del curso Paradigmas de Programación, se ha pedido programar el juego Battleship (Batalla naval), utilizando el paradigma de programación imperativa procedural. Este paradigma se caracteriza por estar basado en la “Máquina de Turing” y utilizar variables las cuales se modifican, por funciones o procedimientos a medida que avanza un programa. Para llevar a cabo la implementación de este juego se hace uso del lenguaje de programación “C” el cual se rige por el paradigma anteriormente mencionado. Este lenguaje es fuertemente tipificado y además es compilado, lo cual hace que el tiempo de ejecución sea menor comparado con otros lenguajes que no son tipificados e interpretados (Ej: Scheme).

En este juego participan dos jugadores, cada jugador posiciona embarcaciones en un tablero, estas posiciones son solamente conocidas por quien ubica dichos barcos. La modalidad de juego se basa en turnos. En cada uno de estos turnos un jugador efectua ataques a una posición del tablero enemigo, si este disparo o ataque llega donde se ubica un barco o parte de este, destruye esa parte o barco en caso de ser la última parte de él.

# Analisís del problema

Para poder llevar cabo la implementación de este juego o llamado programa desde ahora, es necesario conocer los requerimientos que debe cumplir dicho programa. Para esto se dividió este problema en subproblemas, para simplificar y hacer de alguna forma metódica la búsqueda de la solución. Ahora bien, estos su subproblemas serán enumerados a continuación:

1. Crear un tablero donde se pueda jugar
2. Guardar dicho tablero en un archivo de texto.
3. Cargar el tablero de un archivo guardado.
4. Poner embarcaciones en el tablero creado o cargado.
5. Efectuar un disparo.
6. Juntar las soluciones anteriores para desarrollar un juego.
7. Guardar un juego en procedimiento.
8. Cargar un juego guardado.

Cabe recalcar que estos subproblemas fueron divididos en más subproblemas para llevar a cabo la verdadera implementación de este programa que será explicada en el siguiente capítulo.

# Diseño de la solución e Implementacion

Una vez mencionados los puntos anteriores y después se un extenso análisis se procedió a diseñar las soluciones respectivas para cada problema descompuesto del problema principal.

El programa está compuesto por diferentes archivos:

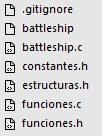


Ilustración 1: Archivos del programa

Dentro del archivo battleship.c se encuentra el main del programa, aquí se hacen los llamados delas funciones creadas que se encuentran en el archivo funciones.c, este ultimo utiliza las estructuras ubicadas en el archivo estructuras.h y además las constantes definidas en el archivo constantes.h.

Como se puede apreciar existe un archivo .gitignore, este archivo existe porque se ocupó Github para llevar a cabo la implementación del programa. De esta forma se hacia el traspasó a un equipo con sistema operativo Linux.

El primer problema presentado al crear el tablero, fue ubicar los barcos de forma aleatoria en la matriz, para ello fue necesario crear un arreglo el cual es contenido en la estructura params (arreglo\_ship\_cpu). Éste fue rellenando con barcos que contienen posiciones aleatorias con la condición que no se repitieran dichas posiciones, utilizándose el siguiente algoritmo;

\*For i=0 hasta la cantidad de barcos:

\*Crear un barco con posiciones aleatorias.

\*Mientras las posiciones del barco superpongan a otro dentro del arreglo:

\*Modificar las posiciones del barco con nuevas posiciones aleatorias.

\*Agregar al arreglo.

Una vez creado el arreglo se procede a ubicar dentro de la matriz de caracteres en las posiciones guardadas en cada barco. El principal problema de este algoritmo es que el random utiliza el tiempo como semilla y al estar dentro del while, tiene que pasar aproximadamente un segundo para que genere un random diferente. Esto hace que el tablero se demore unos segundo en crearse.

**Guardando tablero:**

Para guardar el los datos primero es necesario crear una estructura para guardar el respectivo registro. Luego se convierte la estructura a un string, esto es posible porque ella está formada solamente por caracteres o cadena de caracteres.

Donde los primero dos numero es el tipo de registro:

1. 00: indica la fecha y hora
2. 01:indica: dimensiónFila/dimensionColumna/cantidadShip/cantidadCuadros/cuadrosMinimos

dimensiónFila; cantidad de filas de la matriz guardada.

dimensionColumna: Cantidad de columnas de la matriz guardada.

cantidadShip: Cantidad de barcos de la computadora

cantidadCuadros: Son los cuadros ocupados por los barcos de la Computadora

cuadrosMinimos: Cantidad de cuadros mínimos que deben multiplicar las dimensiones de las matrices (sirve para verificar que caben todos los barcos)

1. 02: tipoShip/orientación/fila/columna

tipoShip: Carácter a poner en el tablero

orientación: indica si el barco esta puesto horizontal o vertical

fila: posición fila en la que esta ubicada la cabeza del ship

columna: posición columna en la que esta ubicada la cabeza del ship

**Cargando tablero:**

De forma inversa al guardado de la información de un tablero, se realiza el cargar un tablero. Primero se recorre el archivo preguntando por el tipo de registro (00,01,02), en el caso de que sea 01 el tipo de registro, se vuelca el buffer que va leyendo el archivo, en una estructura para ese respectivo registro. De la misma forma se realiza para los registro 02 que son los barcos. De esta forma se pueden obtener los datos rápidamente.

# Resulados

Se probó cada funcionalidad por separado y los resultados son los siguientes:

|  |  |
| --- | --- |
| **Función** | **Porcentaje** |
| Crear tablero | 100% |
| Guardar tablero | 100% |
| Cargar tablero | 50% |
| Verificar tablero | 100% |
| Atacar | 100% |
| Posicionar embarcaciones | 100% |
| Visualizar | 100% |

Como se puede apreciar la función cargar tablero es la única que genera conflicto, el error se desconoce. Lo más probable es que después de ocupar la función cargar tablero el programa deje de funcionar, también existen casos en los que no deja de funcionar pero después al utilizar otra función deja de funcionar el programa.

Las funcionalidades extras como guardar juego cargar juego y otra no se alcanzaron a terminar por cuestiones de tiempo. Cabe destacar que se estaba implementado una batalla, pero tampoco se alcanzó a terminar y se deshabilitó del menú (quedaron las funciones incompletas).

# Conclusión

El paradigma de programación utilizado no tuvo mayores limitaciones para poder enfrentar el problema, solo fue necesario pensar como representar lo que se debía mostrar en pantalla, y los datos para poder trabajar con ellos de una forma coherente.

Se espera que en las próximas entregas independiente del paradigma de programación se llegue por lo menos a un 90% de totalidad.

# REFERENCIAS