UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Paradigmas de Programación

Paradigma imperativo Procedural

FELIPE MATURANA GUERRA

Profesor:
Roberto González

Ayudantes:
Giovanni Benussi
Mauricio Rojas
Esteban Contardo

Santiago - Chile
2-2016

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

El paradigma “Imperativo-Procedural” el cual es abordado en esta experiencia se basa en el modelo o arquitectura **Von Neumann**. En el diseño del húngaro el computador es capaz de almacenar una codificación del cálculo realizado, así como también es capaz de ejecutar una secuencia de instrucciones para modificar el contenido almacenado previamente (En el lenguaje C es posible realizar cambios en el contenido almacenado mediante el paso de la dirección de memoria de la variable). El paradigma se caracteriza principalmente por estados que definen el programa y sentencias (Instrucciones) las cuales le indican al ordenador cómo realizar una tarea y así, cambiar ese estado.

El problema que se abordará en el siguiente informe consiste en la implementación en lenguaje C el famoso juego “Battleship”, sin embargo, con algunas modificaciones propias de nuestro departamento como los disparos, ya que en éste caso los disparos son efectuados por los barcos los cuales pueden tener ataques especiales en área, por columna, etc. Este completo proyecto requiere habilidades como manejo de punteros (simples y dobles), declaración de estructuras y la tipificación de éstas, generación de tableros, la validación de éstos y además la posibilidad de visualizarlos a lo largo de la partida luego de posicionar los barcos del enemigo, los propios del jugador y luego de los ataques.

CAPÍTULO 2. ANÁLISIS DEL PROBLEMA

2.1 BATTLESHIP

Dentro del juego existen requerimientos específicos los cuales fueron considerados en la etapa del diseño de la solución partiendo desde lo más general como es la representación del tablero hasta los resultados de cada disparo efectuado desde el barco, pasando por funcionalidades como la verificación de la creación de tableros y además la visualización de éstos.

La representación más obvia del tablero es una representación matricial o de cubo, sin embargo, existen otro tipo de implementaciones las cuales pueden ser más efectivas en términos de una óptima utilización de recursos ya que en su representación matricial se debe pedir memoria para todos y cada uno de las celdas del tablero a utilizar lo cual puede ser contraproducente cuando el usuario solicite tableros muy grandes y la batalla sólo se desarrolle en una porción mínima de éste ya que la memoria para el resto de las celdas estará asignada y no se utilizará, es por esto que surgen representaciones más efectivas como pedir memoria solamente en las celdas que se vaya a poner un barco o se vaya a realizar un ataque.

CAPÍTULO 3. DISEÑO DE LA SOLUCIÓN

El código está estructurado mediante archivos de cabecera (.h) y archivos de código fuente (.c).

Los archivos de cabecera son:

a) Constants.h: Este archivo de cabecera contiene la tipificación de las enumeraciones para “**piezas**”, es decir, la representación de ‘char’ de cada barco en el tablero y las marcas que deja cada disparo efectuado por la función play (“O” si el disparo acertó pero no lo hundió, “X” para indicar que el barco fue destruido y “#” para indicar que el disparo falló). La tipificación de la enumeración de “**comandante”** indicador de pertenencia de cada barco en el tablero (J si el barco le pertenece al jugador y E si el barco pertenece al Enemigo). Enumeración y tipificación de las constantes del tipo **booleanas** (0 para FALSE y 1 para TRUE). Tipificación y enumeración de cada uno

de los errores que puedan existir como por ejemplo que no encuentre el archivo o que el tablero creado sea inválido, no exista, no haya memoria suficiente, etc. En este archivo no fue necesario importar librerías.

b) Structs.h: posee la tipificación y declaración de las estructuras que se utilizarán para la implementación del juego. En este archivo es necesario incluir la librería "constants.h" ya que la estructura Ship posee **Comandante** al igual que la estructura Board para guardar por quién fue efectuado la última jugada.

c) Funciones.h: posee la declaración de todas las funciones utilizadas en el proyecto. En este archivo es necesario incluir las librerías "constants.h" y "structs.h" ya que algunas de las funciones reciben como parámetros estas estructuras.

Los archivos de código fuente (.c) son los siguientes:

a) Funciones.c: donde ocurre la magia del proyecto, es aquí donde se desarrolla el cuerpo de las funciones previamente declaradas en "funciones.h" respetando sus respectivos parámetros y retornos. En esta librería es necesario incluir todos los archivos de cabecera anteriormente descritos además de las librerías de C: ***"stdio.h"*** ya que nuestra función para mostrar el tablero (print) utiliza la función "printf "para mostrar por pantalla el tablero y las acciones que han ocurrido en él así como disparos y barcos colocados. ***"stdlib.h"*** ya que es necesario utilizar funciones como malloc o calloc para reservar memoria y que los cambios realizados en la estructura board sean guardados en "stack" y así poder utilizarlos previniendo que el binding y el scope limiten nuestra capacidad de acción en el programa en general. Además, se incluyó la librería ***"time.h"*** ya que es gracias a ésta que es posible generar posiciones aleatorias al momento de posicionar los barcos del enemigo al momento del llamado de la función createBoard. Se declara la variable **time_t** t y se utiliza **srand** (unsigned) time(&t).

b) Battleship.c: donde se realizan los llamados a las funciones con los argumentos correspondientes, se incluyen la librería "stdio.h" porque se realizan llamados a la función printf para mostrar las pruebas realizadas con nuestras funciones además de los 3 archivos de cabecera señalados con anterioridad.

El proyecto está estructurado mediante una función que crea el tablero, una función que verifica la creación correcta de ésta y de otras funciones que modifican el estado del tablero.

3.1 ENFOQUE DE LA SOLUCIÓN

La representación del tablero será de forma matricial y la estructura de éste en el lenguaje de programación C contendrá los siguientes elementos:

a) char **matriz: Será la representación matricial de caracteres para el tablero el cuál contendrá los barcos y las marcas dónde se han realizados disparos exitosos o fallidos.

b) **Int** n, **int** m: Estos enteros representan las dimensiones de la matriz, cantidad de filas y columnas respectivamente ya que el tablero perfectamente puede ser rectangular, sin embargo, m debe ser un número par ya que la cantidad de columnas debe ser repartido de forma equitativa entre el jugador y el enemigo.

c) **Int** JuegoFinalizado: Este será un indicador de cuando el juego esté finalizado lo cual se traducirá en que el ganador será el Último jugador que realizó la jugada y además cuando el juego esté finalizado (JuegoFinalizado =1) no se podrán realizar jugadas.

d) **Int** BarcosEnemigos, **int** BarcosJugador: Estos enteros representan la cantidad inicial de barcos enemigos y la cantidad de barcos que posee el jugador, éstos datos se necesitan para verificar que al iniciar una partida y realizar un disparo la cantidad de barcos del jugador y del enemigo no se diferencien de un barco (Δ **1**), con 1 como mínimo.

e) **Int** ActualBarcosEnemigos, **Int** ActualBarcosJugador: Representan la cantidad de barcos actual que posee el enemigo y el jugador respectivamente. Cuando un barco sea destruido se descontará en una unidad estos contadores y cuándo lleguen a cero se cambiará el estado de c) **Int** JuegoFinalizado a 1.

f) **Comandante** UltimoJugador: Cada barco posee un **Comandante** el cual representa el propietario del barco (E o J) ya sea **E**nemigo o **J**ugador, esta propiedad sirve para evitar jugadas consecutivas por el mismo jugador.

g) **Ship *** Enemigos, **Ship *** Jugador: Es un puntero a **Ship** el cual contendrá cada uno de los barcos del Enemigo y el Jugador respectivamente.

La representación de los barcos en el tablero será a través de caracteres correspondiente al primer carácter y son 5 tipos.

Tipo de Barco	Largo	Ancho	Caracteristica Especial
Normal	3	1	Ataque por celda.
Acorazado	4	1	Blindaje (1): Resiste 1 ataque.
Porta Aviones	3	1	Ataque por columna.
Submarino	3	1	Ataque en área 3x3.
Buque de Guerra	2	1	Ataque por fila.

Tabla 3-1: Tipos de Barcos implementados

La representación de la estructura de los Barcos en el lenguaje de programación "C" posee los siguientes elementos:

a) **Int** Largo, **Int** Ancho: Estos enteros representan las características principales del barco que se utilizará.

b) **Int** Vida: Esta representación ayuda en la implementación del juego ya que es necesario representar de forma distinta los ataques acertados que destruyen y los que no. Cuando el ataque es acertado, el barco afectado posee **Defensa = 0** y la vida es mayor o igual a 1 se representa con un "**O** = 79". Cuando el ataque es acertado, el barco afectado posee **Defensa** = 0 y la vida es igual a 0 se representa con un "**X** = 88" y si el disparo es fallido (no impacta en un barco) se representa con "**# =** 35**".**

c) **Int** Defensa: Representa una cualidad especial de algunos barcos como el **A**corazado el cual posee una defensa de 3 disparos, es decir, es capaz de mitigar el daño de 3 ataques antes de que su vida sea reducida.

d) **Char** Barco: Es el carácter con el cual es representado en el tablero cada barco, corresponde a la inicial de cada tipo: **N, A, P, S, B.**

e) **Comandante** comandante: Representa a quién le pertenece el barco. Puede ser "**E = 69"** Si el barco es propiedad del enemigo o "**J = 74"** si el barco lo posee el jugador.

Esta característica sirve para dejar constancia de quién fue el último que realizó una jugada en el tablero y que éste no pueda realizarlo dos veces consecutivas.

f) **Int** ** PosicionesTablero: Es la representación de todas las posiciones ocupadas por el barco: PosicionesTablero[N°Barco][0 ó 1] = si es 0 muestra Fila, si es 1 muestra Columna

3.2 ALGORITMOS UTILIZADOS

Al momento de realizar una jugada a través de la función **play** en una posición **p**, si en la posición p existe un barco, es necesario identificar qué barco ha sido afectado por la jugada es por esto que se necesita saber a quién pertenece ese barco a través del atributo Comandante, por ejemplo, si el disparo fue efectuado por un barco del jugador, se revisará **Ship * BarcosEnemigos,** mediante un **algoritmo de búsqueda lineal** el cual está implementado en la función **int** BarcoAtacado(**Position** p, **Comandante** comandante, **Board** *b):

```
int BarcoAtacado(Position p, Comandante comandante, Board *b)
{
    int i,j;
    if(comandante = J)
    {
        for(i=0;i<b->BarcosEnemigos;i++)
        {
            for(j=0;j<b->Enemigos[i].Largo;j++)
            {
                if(b->Enemigos[i].PosicionesTablero[j][0]==p.Fila && b->Enemigos[i].PosicionesTablero[j][1]==p.Columna)
                {
                    return i;
                }
            }
        }
    }
    else if(comandante = E)
    {
        for(i=0;i<b->BarcosJugador;i++)
        {
            for(j=0;j<b->Jugador[i].Largo;j++)
            {
                if(b->Jugador[i].PosicionesTablero[j][0]==p.Fila && b->Jugador[i].PosicionesTablero[j][1]==p.Columna)
                {
                    return i;
                }
            }
        }
    }

    return -1;
}
```

Figura 3-1: Algoritmo de Búsqueda lineal Barco

El algoritmo retorna el número del barco al cual le corresponde en la lista de barcos que posee la estructura Board. Para luego realizar las operaciones que sean necesarias a ese barco, ya sea quitar vida, defensa según corresponda.

La función getScore calcula un puntaje de la partida, recibe como parámetro la estructura Game, la cual posee la estructura Board.

```
int getScore(Game g)
{
    int Puntaje;
    int BJI = g.board.BarcosJugador;
    int BEI = g.board.BarcosEnemigos;
    int BAJ=g.board.ActualBarcosEnemigos;
    int BAE=g.board.ActualBarcosEnemigos;
    int Dif=g.board.Dificultad;
    Puntaje=1000*(BAJ-BAE)*Dif + (BJI-BEI) * 500;


    return Puntaje;
}
```

Figura 3-2: Calculo de puntaje getScore

Para el cálculo del puntaje se consideran 5 aspectos fundamentales en la partida, La cantidad de Barcos que inicialmente tenía cada jugador y la cantidad de barcos que posee en el instante en que el puntaje es calculado. La diferencia que inicialmente existe entre los barcos(BJI-BJE) es ponderada por 500, adicionada a la diferencia que existe entre los barcos que posee actualmente multiplicada por 1000 y además ponderada por la Dificultad da como resultado el puntaje obtenido en la partida.

CAPÍTULO 4. RESULTADOS

De los requerimientos funcionales obligatorios se cumplió a cabalidad cada uno de los 8 mencionados en el informe. El código es capaz de:

1. Generar un tablero válido de dimensiones NxM el cual es divido por la mitad y se le asigna a cada jugador una mitad. Además esta función posiciona todos los barcos enemigos.

2. Guardar un tablero y generar un ID aleatorio el cual es guardado en un archivo de texto “BoardsSaved.txt” (Se incluye en la carpeta) el cual contiene las dimensiones y dificultad del tablero que se esté guardando. Cabe destacar que sólo se implementó la dificultad 1.

3. Cargar un tablero y verificar las condiciones de éste, lo retorna si es que encuentra el ID en el archivo de texto previamente mencionado que recibe como parámetro.

4. Realizar jugadas en el tablero, disparos de distintas dimensiones según la disponibilidad armamentística de los barcos de cada jugador, en este caso, se mantuvo de forma ilimitada los disparos especiales ya que le dan mayor dinamismo al juego, de lo contrario basta con cada vez que hagan un disparo en área cambiar el parámetro int TipoATK del barco que realizó ya su ataque especial a 0. Existen 5 tipos de barcos 4 tipos de ataques y el acorazado que posee blindaje.

5. Posicionar barcos en una posición dada.

6. Imprimir el tablero de forma parcial (sólo la mitad del jugador) o completa.

7. Obtener el puntaje de una partida.

8. Se generó documentación estandarizada con la herramienta Doxygen

9. Encriptación winRAR para la carpeta que contiene los archivos de código la contraseña es “PARADIGMAS22016”.

10. Se generó un archivo de texto plano “.xml” el cuál está contenido en la carpeta junto a la documentación de Doxygen.

Por no sobrepasar las páginas permitida del informe, el resto de las pruebas se realizó en el manual del usuario, sin embargo, la única potencial falla recae en la posibilidad de poner barcos incluso una vez ya se ha disparado.

CAPÍTULO 5. CONCLUSIÓN

El lenguaje de programación, o más bien, el paradigma resultó ser óptimo en la implementación del juego ya que éste permite y facilita a la vez el manejo de matrices (como fue diseñada la solución), mediante el manejo de punteros y punteros dobles es posible llevar a cabo todo el proyecto, el manejo de memoria resultó ser propicio en éste caso ya que a medida que cuando se termina de utilizar un dato, es posible liberar esa memoria previamente asignada y así optimizar los recursos ya que éstos siempre son limitados, sin embargo, esto resulta un arma de doble filo ya que si el programador no posee los conocimientos necesarios acerca del manejo de memoria resultará contraproducente. El paradigma de programación nos ofrece las herramientas necesarias para solucionar de buena manera la implementación del juego, ya que como se expuso previamente la característica principal de éste es la capacidad de cambiar el estado de una variable (estructura en este caso) a través de una secuencia de procedimientos que le indicamos al ordenador.

CAPÍTULO 6. REFERENCIAS

Referenciar páginas web:

Hasbro. (2012). *Reglas para 2 Jugadores BATTLESHIP*. (Recuperado 2016). http://www.hasbro.com/common/documents/dad261471c4311ddbd0b0800200c9a66/DC43DE785056900B109B0A81EE470A9E.pdf

Bernal, J (2013). *LENGUAJE IMPERATIVO O PROCEDURAL*. (Recuperado 2016). http://lenguajesdeprogramacionparadigmas.blogspot.cl/2013/03/lenguaje-imperativo-o-procedural.html

AHO, A., HOPCROFT, J.; ULLMAN, J. D. (1987*). Data Structures and Algorithms Addison-Wesley*. Estados Unidos de América.

Kölher, J. (2015). *Apuntes de la Asignatura Análisis de Algoritmos y Estructuras de Datos.* Chile.