**INFORME SOBRE DESARROLLO Y EJECUCIÓN DEL TRABAJO PRÁCTICO N° 1**

*“Batalla Naval”*

**INTEGRANTES**

Cáceres, Juan Manuel (41411969)

Castro, David (4924164)

*ACLARACIÓN: las funciones y variables en el código están en inglés, sin embargo en este informe las llamamos en español para que sea más fácil la interpretación.*

**DESARROLLO DEL MAIN**

- Primero generamos un puntero a un objeto tipo juego que es la clase encargada de controlar las funciones principales del programa.

- Generamos también un contador de barcos que se irá incrementando a medida que se vayan encontrando barcos (siempre que consiga el primer averiado de una serie o logro hundir un submarino, cuenta como que descubro un barco) se inicializa en -1 para que al encontrar el primer barco incremente a cero (posición 0)

- Llamo a la función que rellena la cola con pilas de posiciones con una posición random en cada pila (entonces tendré una cola, con 100 pilas donde cada pila tendrá una sola posición al azar cada una)

- En el while se desarrollará toda la metodología del juego en sí, y se repetirá siempre y cuando aún no haya descubierto los 10 barcos.

- Creo un elemento posición que me ayudará a guardar posiciones de manera auxiliar.

- Ingreso a un do while (para que se ejecute al menos una vez) donde tomo la posición próxima a ser disparada (no la desapilo) y veo si esa posición ya fue usada (disparada) en algún momento (revisandola en la matriz de booleanos) donde en caso de que esa posición ya fue usada en algún momento, la desapilo y así itero hasta encontrar una posición que no fue usada aún, saliendo del do while.

- Efectúo un disparo (donde la función disparo en sí registra el tiro en la matriz de booleanos, y desapila la posición devolviéndola como return de la función) y almacenamos la posición disparada en la variable posición que ayuda de auxiliar.

- Generamos una variable char que almacena la entrada de teclado y luego mediante el if analizamos si es un “v”, en ese caso hemos descubierto un barco y tenemos la primera posición que lo compone y la guardamos en una variable nueva llamada posición inicial, y luego sumamos en uno el contador de barcos descubiertos (en el primer barco descubierto pasaría a ser 0)

- Con la función registrar barco, almacenamos en un barco (guardado en el arreglo de 10 posiciones, interno en el objeto juego) la posición descubierta donde conseguimos una “v” en el barco que actualmente estamos descubriendo (barco 0 si es el primero), y luego ejecuto la funcion que apila las 4 posiciones siguientes a probar (las apila en caso de ser posible, donde si la posición a apilar se sale del tablero o es una posición que ya fue disparada no la apila)

- Ingreso nuevamente a un do while que igual que antes se encarga de disparar, guardar la posición disparada en posición y pedir una respuesta hasta que se consiga un caracter distinto de “a”.

- Analizo el caracter donde si es “v” guardo la posición encontrada en el barco que actualmente estoy descubriendo, y ejecuto la funcion que apila las 2 posiciones siguientes a ser probadas (igual que antes, las apila si es que son posiciones que no fueron utilizadas o no se salen del tablero) donde a la función se le debe pasar por parámetro la posición inicial del barco, que fue descubierta al principio, y la posición donde conseguí el próximo “v”.

- Nuevamente entro a un do while para disparar y buscar el próximo caracter, comparo con el if si es un “v” y en caso positivo guardo la posición del averiado encontrado en el barco que estoy descubriendo actualmente, y cargo las 2 posiciones posibles para seguir disparando, en caso de ser posible.

- Ingreso al do while para disparar e ingresar por teclado el caracter y luego entro al ultimo if que analiza que el caracter sea un “h” ya que la última alternativa que queda es que sea un barco -destructor- donde se consigue con 3 “v” y finalmente el “h”

- Dentro del if guardo la posición donde conseguí el “h” en el barco que me encuentro descubriendo actualmente, y luego tomo del arreglo de los 10 barcos, el barco en la posición que estoy descubriendo actualmente y lo asigno como descubierto, donde esa función automáticamente determina el tipo de barco que es dependiendo de la cantidad de posiciones que componen al barco, y le cambia un booleano interno que tiene el objeto barco, a descubierto.

- Luego el ciclo for es para mostrar en pantalla lo siguiente:

*se descubrió un -TIPO DE BARCO- en las posiciones: [x1][y1] [x2][y2] [x3][y3] [x4][y4]*

Que son las posiciones donde fui encontrando “v” y finalmente el “h”

- Finalmente antes de salir de ese if, seteo la variable que almacena las entradas de teclado con el caracter “a” para que no me cree conflictos con los demás if.

- A partir de acá quedan los else que son de cada if que verificaba si el caracter ingresado era “v” u otra cosa (esa otra cosa solo puede ser un “h” ya que si era un “a” volvia a pedir otra entrada) por lo que si es un “h” solo queda, igual que antes, guardar la posición encontrada en el barco que actualmente estoy descubriendo, luego también setearlo como descubierto para asignarle un nombre dependiendo de la cantidad de posiciones que lo conforman y cambiar su booleano descubierto a true.

- Ingreso al ciclo for que me genera la misma salida que antes:

*se descubrió un -TIPO DE BARCO- en las posiciones: [x1][y1] [x2][y2] [x3][y3]*

(en este caso tendrá solo 3 posiciones) y seteo la variable que almacena los caracteres ingresados en “a”

- Continuamente ingreso a un else que realiza lo mismo que el bloque anterior, pero para el caso donde primero conseguí un “v” y luego un “h” que sería el caso de un cañonero (2 posiciones), y luego con el for lo muestro en pantalla.

- Continuó luego ingresando a un if (en caso de que no ingrese al primer if al principio del programa) donde entrará si es que logro como primer entrada un “h” donde significa que hundí un barco a la primera, donde eso sería un submarino, entonces, sumo uno en la variable que me cuenta barcos descubiertos y guardo la posición donde conseguí el “h” en el barco que descubrí actualmente y seteo el barco como descubierto, asignandole así el tipo de barco y colocando su booleano en true.

- Luego ingreso al for para generar y mostrar el mensaje:

*se descubrió un -TIPO DE BARCO- en la posición: [x1][y1]*

- Finalmente salgo de la cadena de if else y desencolo de la cola la pila donde fui agregando y quitando posiciones, para luego volver al inicio del while.

- Una vez que descubre los 10 barcos, el while finaliza y se ejecutarán 3 funciones, la primera función imprime una matriz (tablero) por pantalla que muestra de forma gráfica los barcos encontrados a lo largo de la partida, la segunda función también imprime de forma gráfica una matriz (tablero) que será la matriz de booleanos donde se registraron todos los disparos de la partida, y por último la última función que mostrará las posiciones que no fueron disparadas y sobraron de la partida.

**DESARROLLO POR CLASES Y MÉTODOS**

**- struct POSICIÓN**

**- bool flag:** Determina si el objeto struct efectivamente representa una posición.

**- int x:** Almacena la posición en x.

**- int y:** Almacena la posición en y.

**- class NODE**

**- <T> data:** Almacena el dato del nodo.

**- Node\* node:** Puntero al nodo siguiente.

**- Node():** Constructor sin parámetros.

**- Node(<T>):** Constructor con parámetro, que es el dato a cargar.

**- setData(<T>):** Setea la variable que contiene el dato.

**- setNext(Nodo\*):** Setea el nodo siguiente.

**- <T> getData():** Retorna el valor del dato.

**- Node\* getNext():** Retorna el puntero al nodo siguiente.

**- bool isEmpty():** Retorna true si no apunta a un siguiente nodo.

**- class LIST**

**- Node<T>\* start:** Es un puntero que apunta al primer nodo de la lista.

**- List():** Constructor sin parámetros.

**- List(Node<T>\*):** Constructor con parámetro donde este es un puntero que apunta al primer nodo de la lista.

**- add(<T>):** Agrega un nodo a la lista.

**- dlte():** Elimina el primer nodo (nodo cabeza)

**- dlteLast():** Elimina el último nodo.

**- <T> head():** Retorna el dato del nodo de la cabeza.

**- <T> tail():** Retorna el dato del nodo de la cola.

**- List\* rest():** Retorna la lista actual sin el nodo cabeza.

**- bool isEmpty():** Retorna true si la lista está vacía.

**- string toPrint(string):** Recorre y retorna todos los datos de la lista en un string agregando al final el string pasado por parámetro.

**- class PILE**

**- Pile():** Constructor sin parámetros, que genera una lista.

**- stack(<T>):** Apila un elemento.

**- unstack():** Desapila un elemento.

**- <T> take():** Retorna el elemento próximo a salir (no lo elimina)

**- bool pileIsEmpty():** Retorna true si la pila está vacía.

**- string showPile(string):** Recorre y retorna todos los datos de la pila en un string agregando al final el string pasado por parámetro.

**- class QUEUE**

**- Queue():** Constructor sin parámetros, que genera una lista.

**- PutInQueue(<T>):** Encola un elemento.

**- TakeOutOfQueue():** Desencola un elemento.

**- <T> take():** Retorna el elemento próximo a salir (no lo elimina)

**- bool queueIsEmpty():** Retorna true si la cola está vacía.

**- string showQueue(string):** Recorre y retorna todos los datos de la cola en un string agregando al final el string pasado por parámetro.

**- class BOAT**

**- POSITION\* position[4]:** Guarda los casilleros donde se encuentra el barco.

**- string boat:** Define el tipo de barco.

**- bool discovered:** Es true si el barco ya fue descubierto.

**- Boat():** Constructor sin parámetros.

**- setPosition(POSITION\*):** añade una posición más al barco en el arreglo position.

**- setDiscovered():** Cambia la variable discovered a true y dependiendo de cuántos casilleros en position ocupa define el tipo de barco.

**- POSITION\* getPosition(int ):** Retorna la posición del barco almacenada en la posición pasada por parámetro del arreglo.

**- string getBoat():** Retorna el string del tipo de barco.

**- bool getDiscovered():** Retorna true si el barco ya fué descubierto.

**- class POSITIONSTACK**

**- PILE<POSITION>\* positions\_stack:** Almacena las posiciones a ser disparadas en una pila.

**- PositionsStack():** Constructor sin parámetros.

**- PositionsStack(POSITION\*):** Constructor de la clase que recibe una posición inicial por parámetro y la almacena.

**- stackPosition(POSITION\*):** Agrega una posición a la pila.

**- unstackPosition():** Desapila una posición de la pila.

**- POSITION\* take():** Devuelve la posición próxima a salir (no la elimina).

**- class PILESQUEUE**

**- QUEUE<PositionsStack\*>\* piles\_queue:** Almacena las pilas de posiciones en la cola de pilas.

**- PilesQueue():** Constructor sin parámetros.

**- POSITION\* generateRandomPosition():** Genera un elemento posición con un “x” e “y” completamente al azar.

**- fillQueue():** Llena la cola con pilas de una sola posición aleatoria.

**- PositionsStack\* take():** Devuelve el dato próximo a salir (no lo elimina).

**- printPilesQueue():** (usado para test) Imprime las posiciones almacenadas en cada pila de posiciones, de todas las pilas de la cola (las muestra desapilando una por una, vaciando las pilas)

**- class GAME**

**- PilesQueue\* piles\_queue:** Almacena la cola de pilas de posiciones.

**- Boat\* boats[10]:** Almacena los 10 barcos que deberán ser descubiertos.

**- bool shots[10][10]:** Matriz de booleanos donde se guardan las posiciones de los disparos que se fueron realizando.

**- Game:** Constructor de la clase donde se genera la cola de pilas, inicializa el arreglo de barcos con los 10 barcos a descubrir y por último inicializa todos los elementos de la matriz de booleanos en falso.

**- fillWithRandomPositions():** Llena la cola de pilas con posiciones aleatorias.

**- POSITION\* takePosition():** Retorna la posición siguiente de la pila de posiciones que actualmente se está utilizando.

**- POSITION\* shoot():** Dispara eliminando la posición de la pila, registra el disparo en la matriz de booleanos (shots) y finalmente retornando la posición disparada.

**- recordShot(POSITION\*):** Registra en la matriz de booleanos (shots) donde se realizó el disparo que es pasado por parámetro.

**- recordBoat(POSITION\*, int):** Almacena en la posición del arreglo que contiene los barcos, pasada por parámetro, una posición para ser almacenada.

**- bool checkShotPosition(POSITION\*):** Retorna true si la posición pasada por parámetro es true en la matriz de booleanos.

**- stackPositions1(POSITION\*):** Apila las 4 direcciones siguientes (en caso de ser posibles) alrededor de una posición pasada por parámetro.

**- stackPositions2(POSITION\*, POSITION\*):** Apila las 2 direcciones siguientes (en caso de ser posibles) según las 2 posiciones pasadas por parámetro.

**- stackPositions3(POSITION\*, POSITION\*):** Apila las 2 direcciones siguientes (en caso de ser posibles), según las 2 posiciones pasadas por parámetro.

**- unstackPosition():** Desapila una posición de la pila de posiciones.

**- takeOutOfQueue():** Desencola una pila de posiciones de la cola de pilas.

**- Boat\* takeBoat(int):** Devuelve el barco en la posición pasada por parámetro, del arreglo que almacena los 10 barcos.

**- recordBoat(POSITION\*, int):** Almacena en el barco de la posición pasada por parámetro, la posición de un barco.

**- printPilesQueue():** Imprime todas las posiciones iniciales generadas recorriendo toda la cola y todos los elementos de las pilas (desapilandolas y desencolandolas)

**- printBoats():** Muestra el tablero con los casilleros de todos los barcos descubiertos.

**- printShots():** Muestra el tablero con los booleanos donde se realizaron disparos.