Trabajo Práctico Final - Lenguaje de Programación 1

Documentación

Autor: Giovani Dueck

1- Estructuras de Datos

Tableros: Matrices de 10x10 de enteros. Estas guardan un valor diferente para celdas de cada tipo. Los tipos de celda son: vacía/agua/desconocida, acertada, fallada, hundida. Hay dos tipos de tablero para cada jugador, 4 en total. Por un lado están los tableros de flota, en estos se guardan las posiciones de la flota del jugador y sirven para visualizarla; guardan además valores específicos para cada celda ocupada por un barco de un cierto tipo. El Segundo tablero es el de disparos, y no guarda tipo de barco; sobre este se realizan disparos. Existen varios tableros auxiliares usados por algunas funciones, pero funcionan de forma semejante.

Posiciones: una estructura “struct coord” que contiene dos enteros x e y para guardar posiciones.

Barcos: Los barcos de los jugadores se guardan en punteros a estructuras “struct ship”, que contienen todas las informaciones necesarias sobre cada barco de la flota, incluyendo posición, tamaño y cantidad de celdas no disparadas. En el modo PC vs PC solo se guardan los barcos del programa actual. Los punteros apuntan a bloques de memoria dinámica realojada nuevamente cada vez que el juego inicia o reinicia.

Celdas de tablero para la inteligencia: estas celdas son estructuras que guardan la posicion y estado de la celda, similar al valor que guardan los tableros normales. Tableros de 10x10 compuestos por estas celdas son usados por las inteligencia para calcular sus movimientos, pero estos solo se diferencian de los demás tableros en que guardan valores de posicion *y* estado, y en el caso del algoritmo de densidad de probabilidades, un valor de densidad.

Configuración: una estructura es usada para guardar los valores configurables del juego antes de aplicar los cambios.

Vectores de cadenas: entro otros, paths a imagenes y cadenas de mensajes estan guardados en vectores de cadenas.

Cola: el algoritmo de búsqueda e intercepcion (se activa en dificultad 1 o “Capitan”) utiliza una cola para determinar la posicion a dispararse en el siguiente turno si es que se encuentra un barco enemigo.

2- Funciones

(Obs.: Muchas funciones no funcionan para el modo PC vs PC, por lo que existen funciones aparte)

**main.c:**

int main(int argc, char \*argv[]): Contiene las inicializaciones de variables necesarias para crear la interfaz. Ejecuta gtk\_main().

Param: Estandar main()

Ret: 0 si no hay errores, otro numero si los hay

**board.c:**

int count\_ship\_type(int player, int type): Cuenta la cantidad de barcos vivos del tipo type en la flota del jugador player.

Param: player: 0 (jugador)/(programa actual), 1 (oponente); type: 0-4 (tamaño de barco)

Ret: cantidad de barcos vivos del tipo type para el jugador player.

int count\_ships(int player): Cuenta la cantidad de barcos vivos en la flota del jugador player.

Param: player: 0 (jugador)/(programa actual), 1 (oponente)

Ret: cantidad de barcos vivos totales para el jugador player.

int mark\_sunk(int player): Marca barcos hundidos para distinguirlos de los barcos disparados pero vivos para el jugador player.

Param: player: 0 (jugador)/(programa actual), 1 (oponente)

Ret: 0-4 (tamaño de barco hundido) si se hunde un barco, -1 si no se hunden barcos

int place\_ship(int player, int ship\_size, int ship\_n, int orient, int x, int y): Ubica un barco sobre el tablero y genera un struct ship.

Param: player: 0 (jugador)/(programa actual), 1 (oponente); ship\_size: 0-4 (tamaño de barco); ship\_n: 0-3 (número de barco de un tipo); orient: 0-1 (horiz.-vert.); x: 0-9; y: 0-9 (coordenadas)

Ret: 0 (sin errores), -1 (fuera del tablero), -2 (solapamiento de barcos)

void print\_board(int player, int tipo): Imprime el tablero especificado para el jugador player.

Param: player: 0 (jugador)/(programa actual), 1 (oponente); tipo: 0 (flota), 1 (disparos)

Ret: -

void reset(): Resetea todos los tableros a {0}

Param: -

Ret: -

void set\_fleetsize(int n\_fra, int n\_des, int n\_aco, int n\_sub, int n\_por, int n\_tot): Permite configurar la cantidad de cada tipo de barco que tendra cada jugador.

Param: n\_fra: 0-4; n\_des: 0-4; n\_aco: 0-4; n\_sub: 0-4; n\_por: 0-4; n\_tot: 1-9. (cantidades de cada tipo de barco y al final del total de barcos)

Ret: -

**gameloop.c**

int ai\_shoot(int \*i, int \*j): Llama a la funcion de disparo correspondiente a la dificultad. La dificultad puede ser una de tres: 0 (algoritmo random), 1 (algoritmo de busqueda random e intercepcion), y 2 (algoritmo de densidad de probabilidades).

Param: i: puntero a entero que guarda la fila disparada, j: punter a entero que guarda la columna disparada

Ret: 0 o 1 (resultado de disparo: fallo o acierto)

void delay(float seconds): Funcion auxiliar que detiene ejecucion por un tiempo determinado. Ya no es usada.

Param: seconds: cantidad de segundos a detener la ejecucion.

Ret: -

int gameloop():Ejecuta el juego en la version basada en texto. Ya no usada.

Param: -

Ret: 0

void shoot(int player, int fila, int col): Efectua un disparo, es decir actualiza los tableros y barcos dependiendo del resultado del disparo. No revisa si el disparo es valido.

Param: player: 0-1; fila: 0-9; col: 0-9

Ret: -

int validateShot(int player, int fila, int col): Verifica un disparo y retorna el resultado. No efectua el disparo.

Param: player: 0-1; fila: 0-9; col: 0-9

Ret: 0: fallo; 1: acierto; -1: fuera del tablero; -2: celda ya disparada

int winner():Verifica cual jugador es el ganador.

Param: -

Ret: 0 (jugador)/(programa actual), 1 (oponente)

**gtk\_callbacks.c**

void board\_cb(GtkWidget \*event\_box, GdkEventButton \*event, gpointer data): Funcion de callback para un evento button-press-event en un tablero. Contiene las funciones de posicionamiento de barcos y de disparos para el jugador humano.

Param: Estandar callback

Ret: -

void change\_status(int status\_no): Cambia el status\_label para mostrar el mensaje indicado por status\_no. Mensajes guardados en un vector de cadenas.

Param: status\_no: 0-10

Ret: -

int chk\_config():Verifica si la configuracion es valida.

Param: -

Ret: 0 (no valida), 1 (valida)

void config\_ok\_cb(GtkWidget \*widget, gpointer data): Callback para el boton OK en config\_dialog. Caso la configuracion sea valida, se aplica y se cierra el dialogo.

Param: Estandar callback

Ret: -

void config\_reset\_cb(): Callback para el boton config\_reset. Reinicializa las configuraciones a su valor inicial.

Param: -

Ret: -

void confirm\_quit(GtkWidget \*widget, gpointer data): Si el juego está en progreso, pregunta por una confirmación antes de salir del programa. Se llama al usar la opcion de menu "Salir".

Param: Estandar callback

Ret: -

gboolean confirm\_quit\_delete(GtkWidget \*widget, GdkEvent \*event, gpointer data): Si el juego esta en progreso, pregunta por una confirmacion antes de salir del programa. Se llama al crear el evento "delete-event" en window.

Param: Estandar callback

Ret: TRUE: confirmar; FALSE: cancelar

void confirm\_restart(GtkWidget \*widget, gpointer data): Si el juego esta en progreso, pregunta por una confirmacion antes de reiniciar el juego. Se llama al usar la opcion de menu "Nuevo".

Param: Estandar callback

Ret: -

GtkWidget \* create\_board\_ju(): Retorrna el tablero del jugador usando un eventbox e imagenes de fondo.

Param: -

Ret: eventbox: tablero con callback para evento click

GtkWidget \* create\_board\_op(): Retorrna el tablero del oponente usando un eventbox e imagenes de fondo.

Param: -

Ret: eventbox: tablero con callback para evento click

GtkWidget \* create\_col(): Crea las filas que indican la columna del tablero.

Param: -

Ret: col: columna de imagenes numeradas

GtkWidget \* create\_fila(): Crea las columnas que indican la fila del tablero

Param: -

Ret: fila: fila de imagenes numeradas

void difficulty\_cb(GtkWidget \*widget, gpointer data): Callback para diff\_sel en en config. Permite cambiar la dificultad.

Param: Estandar callback

Ret: -

void end\_game(): Termina el juego y muestra el mensaje de fin de juego.

Param: -

Ret: -

long file\_len(FILE \* f): Funcion auxiliar para medir la longitud de un archivo.

Param: f: archivo

Ret: long: tamaño en bytes

void game\_over\_new\_cb(): Callback para game\_over\_new. Reinicia el juego.

Param: -

Ret: -

void game\_over\_quit\_cb():Callback para game\_over\_quit. Cierra el programa.

Param: -

Ret: -

void game\_over\_stats():Escribe las extadisticas de la partida sobre el game\_over\_dialog. Esto incluye la cantidad de disparos y aciertos.

Param: -

Ret: -

void gamemode\_cb(GtkWidget \*widget, gpointer data): Cambia el gamemode seleccionado en el dialogo de configuracion.

Param: Estandar callback

Ret: -

int get\_shots():Verifica que la entrada a shot\_entry sea valida, es decir, no hay caracteres no numericos. shot\_entry es un campo de entrada de texto en configuracion.

Param: -

Ret: -1 (no valido), cantidad de disparos (valido)

void init\_config():Carga la configuracion inicial.

Param: -

Ret: -

void init\_instruc():Inicializa el dialogo instrucciones

Param: -

Ret: -

void instruc\_ok\_cb(GtkWidget \*widget, gpointer data): Cierra el cuadro de dialogo de instrucciones. Callback de boton OK.

Param: Estandar callback

Ret: -

void new\_game():Inicializa los tableros y la configuracion del juego como para iniciar un juego nuevo.

Param: -

Ret: -

void orient\_button\_cb():Cambia la orientacion seleccionada en la colocacion de barcos

Param: -

Ret: -

void path\_ok\_cb():Callback para path\_ok. Verifica que un directorio se selecciona en PC vs PC.

Param: -

Ret: -

void place\_ship\_gui(int x, int y): Posiciona los barcos del jugador en las posiciones que elige si hacer los cambios definitivos. La idea es poder visualizar las posiciones pero posibilitar cambios.

Param: x: 0-9; y: 0-9 (coordenadas)

Ret: -

void place\_ship\_ok\_cb():Confirma la seleccion de posicion y la guarda en la matriz tablero

Param: -

Ret: -

void run\_about():Muestra el dialogo about

Param: -

Ret: -

void run\_config():Funcion callback para mostrar el dialogo de configuracion de partida

Param: -

Ret: -

void run\_game\_over():Muestra el dialogo de fin de juego

Param: -

Ret: -

void run\_instruc(GtkWidget \*widget, gpointer data): Funcion callback para mostrar el dialogo de instrucciones

Param: Estandar callback

Ret: -

void run\_path():Muestra el dialogo path\_dialog en PC vs PC

Param: -

Ret: -

void run\_stats():Muestra el dialogo stats

Param: -

Ret: -

void save\_player\_stats():Guarda las estadisticas del juego de manera permanente

Param: -

Ret: -

void set\_ships\_images():Asigna imagenes de acuerdo a las flotas al grid de barcos de la interfaz grafica

Param: -

Ret: -

void ship\_add(GtkWidget \*widget, gpointer data): Aumenta el contador del barco especificado en el dialogo de configuración

Param: Estandar callback

Ret: -

void ship\_sub(GtkWidget \*widget, gpointer data): Decrementa el contador del barco especificado en el dialogo de configuración

Param: Estandar callback

Ret: -

int shoot\_cpu\_gui():Dispara como PLAYER\_2 en Jugador vs PC

Param: -

Ret: -

void shot\_cb(GtkWidget \*widget, gpointer data): Callback para shot\_entry. Revisa el valor introducido para el limite de disparos

Param: Estandar callback

Ret: -

void show\_stats():Muestra estadisticas de jugadores guardadas, si existen

Param: -

Ret: -

void start\_cb(GtkWidget \*widget, gpointer data): Cambia el jugador que comienza el juego en configuracion

Param: Estandar callback

Ret: -

void start\_game():El juego progresa a la fase de disparos. Desbloquea funciones de disparos y bloquea funciones de configuracion.

Param: -

Ret: -

void stats\_close\_cb(GtkWidget \*widget, gpointer data): Cierra el dialogo stats

Param: -

Ret: -

void update\_board\_images():Reasigna imagenes de acuerdo al tablero interno al tablero de la interfaz grafica

Param: -

Ret: -

void update\_ships\_images():Reasigna imagenes de acuerdo a las flotas al grid de barcos de la interfaz grafica

Param: -

Ret: -

**hunt\_target.c**

(Obs.: Algoritmo de busqueda e intercepcion, activo en dificultad 1 o “Capitan”)

int HTshoot(int \*i, int \*j): Dispara una celda dependiendo del modo active

Param: i: puntero a entero que guarda la fila disparada, j: puntero a entero que guarda la columna disparada

Ret: 0 (fallo), 1 (acierto)

int hunt\_shoot(int \*i, int \*j): Modo busqueda: dispara una celda al azar. Caso se acierta, las celdas adyacentes se agregan a la cola de objetivos si no se ha disparado ya, y el modo de intercepcion se activa

Param: i: puntero a entero que guarda la fila disparada, j: puntero a entero que guarda la columna disparada

Ret: 0 (fallo), 1 (acierto)

void init\_HT(): Inicializa el tablero interno del algoritmo

Param: -

Ret: -

void mark\_cell(int x, int y, int state): Cambia el componente state de la celda en la posicion (x,y)

Param: x: 0-9; y: 0-9 (coordenadas); state: HIT, MISS o SUNK

Ret: -

struct cell\* pot\_read(): retorna el siguiente elemento en la cola

Param: -

Ret: celda a dispararse

void pot\_write(int x, int y): agrega un elemento a la cola

Param: x: 0-9; y: 0-9 (coordenadas)

Ret: -

int target\_shoot(int \*i, int \*j): Modo intercepcion: dispara la siguiente celda en la cola. Caso acertar, agrega las celdas adyacentes a la cola, si no fueron disparadas ya. Si la cola se vacia, se cambia al modo busqueda.

Param: i: puntero a entero que guarda la fila disparada, j: puntero a entero que guarda la columna disparada

Ret: 0 (fallo), 1 (acierto)

**pcvspc.c**

(Obs.: funciones para el modo PC vs PC)

void mark\_mystery(int x, int y, int res): Marca la posicion indicada en el tablero desconocido del oponente

Param: x: 0-9; y: 0-9 (coordenadas), res: 0 (fallo), 1 (acierto), 2 (hundimiento)

Ret: -

void pvp\_clean():Elimina archivos con los nombres usados si es que existen

Param: -

Ret: -

void pvp\_end\_game():Crea el archivo p\_name.txt con la linea FIN

Param: -

Ret: -

int pvp\_get\_response(int \*x, int \*y): Lee el archivo respuesta a un disparo y registra los resultados sobre el tablero propio

Param: i: puntero a entero que guarda la fila disparada, j: puntero a entero que guarda la columna disparada

Ret: 0 (fallo), 1 (acierto), 2 (hundimiento), -1 (ninguna respuesta), -2 (error de formato)

void pvp\_log\_turn():Registra el turno en el historial

Param: -

Ret: -

void pvp\_reset\_historial():Borra el archivo historial.txt y crea uno nuevo

Param: -

Ret: -

int pvp\_respond(int \*x, int \*y): Realiza el diparo del oponente sobre los tableros propios. Escribe los resultados a un archivo de respuesta.

Param: i: puntero a entero que guarda la fila disparada, j: puntero a entero que guarda la columna disparada

Ret: 0 (fallo), 1 (acierto), 2 (hundimiento), -1 (ninguna respuesta)

void pvp\_set\_path(gchar \*folder): Guarda el path al directorio en el que se juega

Param: folder: camino a directorio

Ret: -

void pvp\_setup\_board():Coloca los barcos del programa usando ciertas reglas

Param: -

Ret: -

void pvp\_shoot(int \*x, int \*y): Realiza un disparo, crea un archivo de disparo.

Param: i: puntero a entero que guarda la fila disparada, j: puntero a entero que guarda la columna disparada

Ret: -

int pvp\_turn():Llama funciones segun turn; genera disparos o respuestas. Retorna TRUE o FALSE segun las condiciones de victoria. TRUE indica a g\_timeout que el juego continúa

Param: -

Ret: TRUE, FALSE

void reset\_mystery():Reinicializa el tablero de disparos del programa

Param: -

Ret: -

**prob\_density.c**

(Obs.: algoritmo de densidad de probabilidades, dificultad 2 o “AI”)

void calculate\_density(): calcula posiciones probables para cada barco vivo

Param: -

Ret: -

void init\_PD(): inicializa el tablero del algoritmo

Param: -

Ret: -

void mark\_sunk\_PD(int size): resta un barco de tamaño size de la cuenta de barcos vivos

Param: size: 0-4 (tamaño de barco)

Ret: -

void PD\_place\_ship(int ship\_size, int x, int y, int orient): prueba la posicion y orientacion para ver si es possible colocar el barco especificado. Caso si, las densidades de las celdas ocupadas incrementa.

Param: ship\_size: 0-4 (tamaño de barco); x: 0-9; y: 0-9 (coordenadas); orient: 0-1 (horiz.-vert.)

Ret: -

int PDshoot(int \*i, int \*j): dispara sobre la celda con mayor densidad de probabilidades. Caso haya multiples con valor igual, una es disparada al azar.

Param: i: puntero a entero que guarda la fila disparada, j: puntero a entero que guarda la columna disparada

Ret: 0 (fallo), 1 (acierto)

void print\_density\_grid(): function auxiliary que imprime un tablero con los valores de densidad en consola para visualizer el mapa creado.

Param: -

Ret: -

void reset\_density(): reinicializa la densidad de cada celda en 0

Param: -

Ret: -

void setup\_board\_PD():Coloca los barcos del programa usando ciertas reglas

Param: -

Ret: -

void sim\_placements(int ship\_size): llama PD\_place\_ship con cada coordenada, orientacion y tipo de barco vivo.

Param: ship\_size: 0-4 (tamaño de barco)

Ret: -

**rand.c**

(Obs.: algoritmo aleatorio, dificultad 0 o “Cadete”)

int gen\_num(int lower, int upper): Genera un numero entre lower y upper, inclusive

Param: lower: limite inf.; upper: limite sup.

Ret: numero generado

void seed():Cambia la semilla del generador para que sea basado en el reloj

Param: -

Ret: -

void setup\_board\_rand():Coordina llamadas de place\_ship() para el jugador aleatorio

Param: -

Ret: -

int shoot\_rand(int \*i, int \*j): Llama a validateShot() y shoot() para el programa usando valores aleatorios

Param: i: puntero a entero que guarda la fila disparada, j: puntero a entero que guarda la columna disparada

Ret: 0 (fallo), 1 (acierto)

**ui.c**

(Obs.: funciones con interfaz basada en texto. Funciones durmientes en la version final)

void ai\_shot\_msg(int x, int y, int res): imprime un mensaje para informar sobre el turno del programa

Param: x: 0-9; y: 0-9 (coordenadas); res: 0 (fallo), 1 (acierto)

Ret: -

void brief(int disp, int disp\_ai): Imprime un reporte para el usuario. Incluye informaciones sobre las dos flotas y cantidad de disparos

Param: disp: cantidad de disparos del jugador; disp\_ai: cantidad de disparos del programa

Ret: -

void game\_over\_msg(int winner): Imprime un mensaje de fin de juego y los resutados

Param: winner: ganador: 0 (jugador), 1 (programa)

Ret: -

void place\_ship\_ui(int player, int ship\_size, int ship\_n): llama place\_ship con una interfaz de texto

Param: player: 0-1; ship\_size 0-4; ship\_n: 0-4 (numero de barco del tipo ship\_size)

Ret: -

void print\_help():Imprime instrucciones de juego y aclara las reglas.

Param: -

Ret: -

void print\_ref():Imprime referencias a simbolos que aparecen sobre el tablero conforme avance el juego

Param: -

Ret: -

int select\_difficulty():Permite al usuario seleccionar la dificultad del oponente

Param: -

Ret: dificultad (0-2)

int select\_gamemode():Permite al usuario seleccionar el modo de juego

Param: -

Ret: modo de juego (0 – Jug. vs PC, 1 – PC vs PC)

void set\_fleetsize\_ui(): Version con interfaz de texto de set\_fleetsize

Param: -

Ret: -

void set\_max\_shots():Pregunta si el jugador desea poner un limite de disparos, al llegar al limite termina el juego

Param: -

Ret: -

void set\_turn():Pregunta al usuario cual jugador debe empezar a disparar

Param: -

Ret: -

void setup\_board():Coordina llamadas de place\_ship\_ui()

Param: -

Ret: -

int shoot\_ui():Llama a validateShot() y shoot() con interfaz para el jugador.

Param: -

Ret: 0 (fallo), 1 (acierto)

void ui\_settings():Junta todas las configuraciones iniciales e imprime el mensaje de bienvenida

Param: -

Ret: -

3- Inteligencia/Estrategia

En el modo Jugador vs PC se puede seleccionar la dificultad del oponente, y se tiene 3 opciones: “Cadete”, el algoritmo aleatorio, “Capitan”, el algoritmo de busqueda e intercepcion, y “AI”, el algoritmo de densidad de probabilidades.

El Capitan opera en dos modos: busqueda e intercepcion. En el primero, el algoritmo dispara sobre el tablero de forma aleatoria hasta encontrar una celda con un barco. Al marcarse una celda como acierto, las celdas adyacentes, siempre que no hayan sido disparadas hasta entonces, se agregan a una cola de posibilidades, y el algoritmo cambia al modo intercepcion. En el modo intercepcion, el algoritmo toma la siguiente celda en la cola y la dispara. Cuando la cola se vacie, el algoritmo vuelve al modo de busqueda.

Las debilidades de este algoritmo son son principales. En primer lugar, no optimiza la busqueda, ya que no sabe los tamaños de barco que busca ni cuales ya se hundieron, y en segundo lugar, ya que no sabe si un barco fue hundido o no, es forzado a disparar todas las celdas alrededor de un acierto para asegurarse de que no deja ninguna celda potencialmente ocupada atras, bajando si precision. Por esta razon, decicí escribir un segundo algoritmo, que rodea los dos problemas del algoritmo de busqueda e intercepcion.

La AI es un algoritmo basado en una funcion de densidad de probabilidades. Esto significa que, para cada celda en el tablero, calcula las posibles maneras en las que pudiese ocuparse por cualquiera de los barcos vivos. Esto causa, por ejemplo, que en el comienzo de la partida, el algoritmo dispare a una de las 16 celdas centrales del tablero, ya que estas pueden ser utilizadas en la mayor cantidad de posicionamientos posibles.

El algoritmo tambien adapta sus calculos al tablero cambiante. Cuando una celda es disparada, esta mantiene una probabilidad de 0, ya que no puede ser disparada otra vez. Cuando una celda es marcada como acertada, las probabilidades de celdas colineales son infladas, ya que existe un barco en alguna de esas celdas con certeza, por lo que el algoritmo dispara sobre una de estas celdas. Si una celda es marcada como hundida, ya no infla la probabilidad de celdas adyacentes, ya que no se sabe con certeza que hay un barco cerca, la celda actua como una celda fallada.

A medida que los barcos se hunden, hay menos posibilidades de colocacion. Si, por ejemplo, de una flota solo queda un portaaviones, el algoritmo solo dispara sobre celdas que son parte de una linea de 4, ya que el barco no cabe en otras posiciones. Esto le otorga una clara ventaja sobre el algoritmo de busqueda aleatoria.

La principal debilidad de este algoritmo se presenta cuando un oponente posiciona sus barcos en los bordes, ya que en el inicio, esas celdas tienen menos valor de probabilidad y el algoritmo tarda en llegar a ellas.

Este algoritmo tambien respeta algunas reglas de posicionamiento de barcos: el barco de menor tamaño es colocado en uno de los bordes del tablero para ser mas dificil de encontrar, y el resto de los barcos se colocan de forma aleatoria, pero sin tocarse entre si, asi se evita que el oponente encuentre varios barcos en un bloque.

4- Puntos Fuertes

El principal fuerte de mi programa es la inteligencia, que es superior a muchos jugadores humanos gracias a su precision y rapidez de calculo.

Otro punto fuerte, en mi opinion, es una interfaz clara, simple y limpia.

5- Puntos debiles

Hasta donde yo sé, mi programa no tiene significantes puntos débiles, pero para el modo PC vs PC, el programa no revisa el formato de la ultima entrada el historial de jugadas.

6- Dificultades encontradas

Las mayores dificultades se encontraron al hacer el testing y debugging para el modo PC vs PC. Para solucionar esta dificultad fue necesario ejecutar dos instancias del programa y hacer que juegen una contra la otra.

7- Guia de utilizacion

**Inicio**

El programa inicia con la ventana principal que contiene los dos tableros, la barra de menus, y un boton “Nueva partida”.

Para iniciar una nueva partida ir a Juego > Nuevo o hacer click sobre “Nueva Partida” en la ventana de inicio. Una ventana de configuracion aparece con una configuracion preseleccionada. Para cambiar la configuracioc usar los botones y entradas de texto. Para reestablecer la configuracion inicial, hacer click sobre “Reestablecer”. Para confirmer la configuracion, hacer click sobre “OK”.

**Colocacion de barcos**

Para colocar un barco sobre el tablero, hacer click sobre la primera celda que ocupa el barco seleccionado, indicado por el texto en el centro de la ventana. Para cambiar de orientacion, hacer click sobre el boton que indica la orientacion actual. Para confirmer la posicion, hacer click sobre “Colocar”. Barcos no pueden salir del tablero ni solaparse, en estos casos la posicion no es valida.

Al colocar todos los barcos, hacer click sobre el boton “Comenzar” para comenzar a disparar.

**Disparos**

Para disparar, hacer click en la celda objetivo. No puede disparar sobre celdas ya disparadas.

Celdas disparadas toman tres formas: fallos (punto gris), aciertos (cruz roja), y hundimientos (cruz gris).

Al acertar o hundir un barco, puede disparar otra vez. Al fallar, pasa el turno al siguiente jugador. El turno es indicado por el texto en el centro de la ventana.

La integridad de las dos flotas es visible debajo de su tablero, y si un limite de turnos se introdujo en la configuracion, los turnos restantes tambien se muestran.

**Fin de la partida**

Al hundir un jugador la flota de su oponente o alcanzarse el limite de turnos, el juego termina, y el resultado es anunciado. Si hay un ganador, se muestra en pantalla, si hay un empate, esto tambien se muestra en pantalla.

La ventana de fin de partida muestra algunas estadisticas del juego.

Para jugar otra vez, hacer click sobre “Juego nuevo".

Para cerrar el programa, hacer click sobre “Salir”.

**Menus**

La barra de menus es eccesible en todo momento.

Juego > Nuevo: Inicia una nueva partida, si un juego ya esta en progreso, confirmar accion.

Juego > Estadisticas: Muestra las estadisticas guardadas por nombre de jugador. Para guardar estadisticas, introducer un nombre de jugador en configuracion de juego nuevo.

Juego > Salir: Cierra el programa, si un juego ya esta en progreso, confirmar accion.

Ayuda > Instrucciones: Muestra las reglas del juego e instrucciones de uso.

Ayuda > Acerca de: Muestra informacion acerca del programa.