Project 1

**Battleship**

Joseph Alipranti

CSC-17A 49285

10/25/2020

**Introduction**

The game I have created with the implementation of the concepts learned in the previous chapters is the classic game, Battleship. I created this game because it used to be a childhood favorite game of mine, and even though the rules and gameplay are simple, I saw it as a challenge to program, and took it upon myself to create it. The rules for the game are simple. Take turns with the computer on guessing where the other player has their ships placed, and the first person to sink all of their enemy’s ships wins the game.

**Summary**

Lines of Code: roughly 425 (not including the .h file)

Number of Variables: around 35 (some variables repeat in different functions)

Utilizes concepts from Chapters 9 through 12

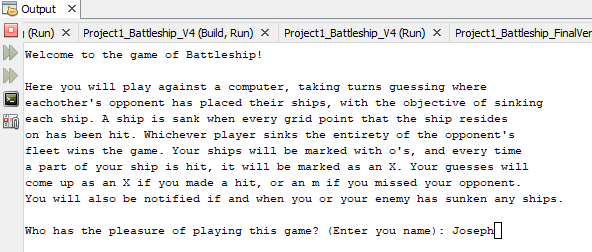
Programming this project was quite a challenge, as there were many things to validate and ensure certain actions/decisions made by the user or computer would not step out of bounds of the grid. One huge part was to ensure that the placement of the ships were random, but would not be placed diagonally, on top of each other, or placed half-off of the grid of play. A lot of time and thought went in to figuring out a way to implement it correctly while making sure placement was random for optimal game experience. Other concepts were difficult to program as well, such as determining whether or not an entire ship has been sunk, or only part of it. With a lot of brain storming and error testing, I was able to come up with ways to solve these complex programming issues and make my game functionable.

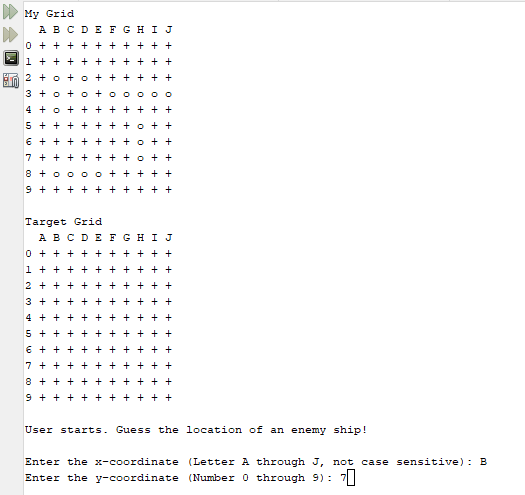
**Description**

For programming the solution to the problem (game), I brushed up on the basic rules of Battleship, the pieces/grids used by each player, and the methodology of the gameplay and how the grids get marked for reference. Programming this project took me about a week to complete, slowly everyday chipping away at the little functions and tasks that I needed to get done in order to have a near-replica of the actual game.

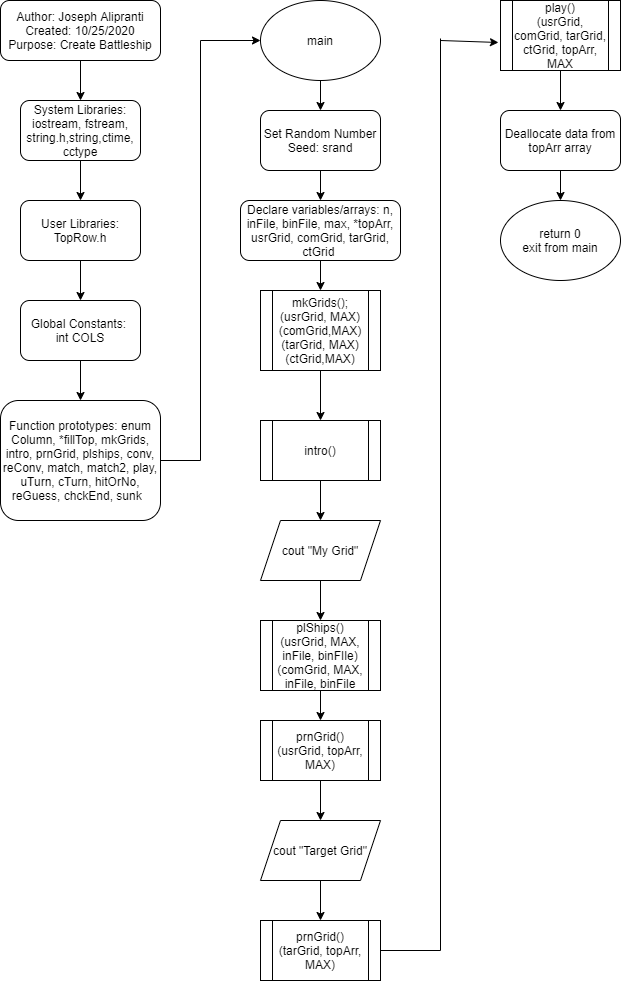
**Sample Input/Output**

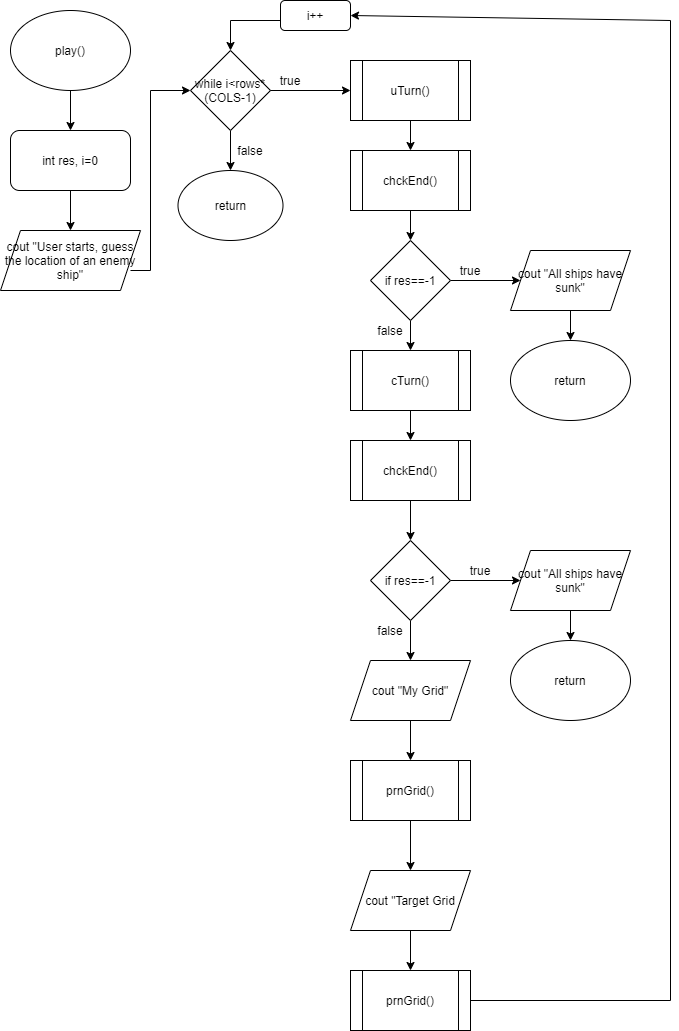
The only thing that you (the user) will be entering in the program for play, is your name at the beginning, and then the coordinates that are asked of you by the program for your guesses:





**Flowcharts/Pseudocode**





*Start*

*Include Libraries*

*Declare function prototypes*

*Initialize Variables*

*Make grids()*

*Intro()*

*Place Ships()*

*Show Grids()*

*Play()*

*Delete dynamic memory*

*Return 0*

*Play()*

*Declare/initialize res, i=0*

*Output user starts*

*While I is less than rows\*(cols-1)*

*uTurn()*

*chckEnd()*

*if res==-1*

*output ships sank*

*return*

*cTurn()*

*chckEnd()*

*if res==-1*

*output ships sank*

*return*

*output my grid*

*print grid()*

*output target grid()*

*pring grid()*

*i++*

**Variables**

Const int COLS: global, above function prototypes to pass 2D arrays to functions

Const int MAX: main(), for the rows of each grid

Ifstream inFile: main(), to retrieve the ship pieces from a text file

Fstream binFile: main(), to send the ships to binary file for confirmation after input

Char getC: \*fillTop(), to get character and store in array after converting an integer

Int lBound: mkGrids(), to turn integers into characters using ASCII conversion for grid axis

Int ships: plShips(), number of ships to red in from text file

Int r: Every function using input from user during game, to store the row chosen

Int c: Every function using input from user during game, to store the row chosen

Int test, test1: plShips(), test if ship placement has gone off grid or is overlapping

Int ret: all return functions, to hold the return integer for every variable that called it

Int res, res2: uTurn(), cTurn(), to hold the return values checking for ships sunk

**Concepts**

Included in checkoff sheet

**References**

All code is original code, however, I have used cplusplus.com and w3schools.com for reference on syntax and use.

**Program**

/\*

\* File: main.cpp

\* Author: Joseph Alipranti

\* Created on October 25th, 2020, 6:05 PM

\* Purpose: Create a game of Battleship between the user and the computer

\*

\* Version 5 (Final Version): Implement required structures, arrays

\* files, enumerators, etc. to complete checklist

\*/

//System Libraries

#include <iostream> //I/O Library

#include <fstream> //File Stream Library

#include <string.h> //Char Array Manipulation Library

#include <cctype> //For toupper operation

#include <ctime> //For rand operation

#include <string> //String Library

using namespace std;

//User Libraries

#include "TopRow.h"

//Global Constants Only

//Well known Science, Mathematical and Laboratory Constants

const int COLS = 11;

//Function Prototypes

enum Column {A,B,C,D,E,F,G,H,I,J};

topRow \*fillTop(int); //Create top row from a pointer within a structure

void mkGrids(char [][COLS],const int); //Create the grid framework

void intro(); //Display intro text

void prnGrid(char [][COLS],topRow \*,const int); //Print grids

void plShips(char [][COLS],const int,ifstream &,fstream &); //Place the ships on the grids

int conv(char &); //Convert char input into integer to search through arrays

char revConv(int &); //Reverse convert int input into char to display guess

int match(const int [],const int,const int); //P1 of matching input for ship placement

int match2(const char [][COLS],const int,const int,const int); //P2 of matching input for ship placement

void play(char [][COLS],char [][COLS],char [][COLS],char [][COLS],topRow \*,const int); //Function stepping through game

void uTurn(char [][COLS],char [][COLS]); //Function for user to make their guesses

void cTurn(char [][COLS],char [][COLS]); //Function for computer to make their guesses

int hitOrNo(char [][COLS],const int,const int); //Determine if hit or miss

int reGuess(char [][COLS],const int,const int); //Evaluate if user/computer guess is repeated

int chckEnd(char [][COLS],const int); //Check if all ships sunk

int sunk(char [][COLS],const int,const int); //Check if a ship has sunk

//Execution of Code Begins Here

int main(int argc, char\*\* argv) {

//Set the random number seed here

srand(time(0));

//Declare all variables for this function

int n=0;

ifstream inFile;

fstream binFile;

const int MAX = 10;

topRow \*topArr=fillTop(MAX);

char usrGrid[MAX][COLS]; //Define user grid

char comGrid[MAX][COLS]; //Define computer grid

char tarGrid[MAX][COLS]; //Define target grid

char ctGrid[MAX][COLS]; //Define computer target grid

//Initialize all known variables

//Process Inputs to Outputs -> Mapping Process

//Maps known values to the unknown objectives

mkGrids(usrGrid,MAX);

mkGrids(comGrid,MAX);

mkGrids(tarGrid,MAX);

mkGrids(ctGrid,MAX);

//Display the Inputs/Outputs

intro();

cout<<"My Grid"<<endl;

plShips(usrGrid,MAX,inFile,binFile);

plShips(comGrid,MAX,inFile,binFile);

prnGrid(usrGrid,topArr,MAX);

cout<<endl<<"Target Grid"<<endl;

prnGrid(tarGrid,topArr,MAX);

cout<<endl;

play(usrGrid,tarGrid,comGrid,ctGrid,topArr,MAX);

//Clean up the code, close files, deallocate memory, etc....

delete []topArr[n].topGrid;

delete []topArr;

//Exit stage right

return 0;

}

topRow \*fillTop(int max){

int n=1;

int f=0;

int c=1;

char getC;

topRow \*a=new topRow[n];

a[f].size=max; //Set size

a[f].topGrid=new char[a[f].size]; //Set new size for topGrid array

for(int i=A;i<=J;i++){

getC=revConv(c);

a[f].topGrid[i]=getC; //Fill topGrid array with chars

c++;

}

return a;

}

//Function Implementations

void mkGrids(char grid[][COLS],const int rows){

int lBound=47, //Set to ASCII decimal of the char before 0

first=0;

for(int r=0;r<rows;r++){

lBound++;

grid[r][first]=lBound; //y-axis of grid made up of numbers in the char array

for(int c=1;c<COLS;c++){

grid[r][c]='+'; //Fill rest of grid with +'s

}

}

}

void intro(){

int max=35;

string name;

char title[35]="Welcome to the game of Battleship!";

//Introductory comments, displaying basic rules and objectives of the game

//and how it will run

cout<<title<<endl<<endl<<

"Here you will play against a computer, taking turns guessing where"<<endl<<

"eachother's opponent has placed their ships, with the objective of sinking"<<endl<<

"each ship. A ship is sank when every grid point that the ship resides"<<endl<<

"on has been hit. Whichever player sinks the entirety of the opponent's"<<endl<<

"fleet wins the game. Your ships will be marked with o's, and every time"<<endl<<

"a part of your ship is hit, it will be marked as an X. Your guesses will"<<endl<<

"come up as an X if you made a hit, or an m if you missed your opponent."<<endl<<

"You will also be notified if and when you or your enemy has sunken any ships."<<endl<<endl;

cout<<"Who has the pleasure of playing this game? (Enter you name): ";

getline(cin,name);

cout<<endl;

//Comments leading to the display of the starting version of user's grid

//and target grid

cout<<name<<", here is your grid ('My Grid'), as well as the grid where you guess the"<<endl<<

"location of your opponent's ships ('Target Grid'). You both will have"<<endl<<

"your ships set up on the grid randomly, so you may begin playing."<<endl<<endl;

}

void prnGrid(char grid[][COLS],topRow \*arr,const int rows){

int n=0;

cout<<" ";

for(int i=0;i<rows;i++)

cout<<" "<<arr[n].topGrid[i]; //Format the top x-axis of the grid

cout<<endl;

for(int i=0;i<rows;i++){

for(int j=0;j<COLS;j++){

cout<<grid[i][j]<<" "; //Format the rest of the grid below

}

cout<<endl;

}

}

void plShips(char grid[][COLS], const int rows, ifstream &file,fstream &binFile){

binFile.open("file.bin",ios::out|ios::binary); //Prime the file for output, binary mode

file.open("ships.txt");

int max=6, //Max length of char array if including null terminator

ships=5, //Number of ships used for game

size=0, //Size of ship, determined when read from file

r=0, //Initializing rows

c=0, //Initializing columns

j=0, //Initializing increment variable for 1st part of if/else

k=0; //Initializing increment variable for 2nd part of if/else

int test,

test1;

int \*rTaken=new int[max-1]; //1D Dynamic Array to hold random row designations as they come

for(int i=0;i<(max-1);i++)

rTaken[i]=0; //Fill array with 0's to be out of range of the flag

char ship[max]; //Char array to get input from file

if(file){

for(int i=0;i<ships;i++){

file.getline(ship,max); //Read line from file

size=strlen(ship); //Determine size of line to get the size of ship

binFile.write(reinterpret\_cast<char \*>(&size),sizeof(int)); //Send size of char array

binFile.write(ship,size\*sizeof(char)); //Then send char data to file

if(size>3){

r=rand()%9+0; //Random row from 0 to 9 to avoid exceeding grid

test=match(rTaken,(max-1),r); //Compare coordinate to array to avoid overlapping ships

if(test==-1){

while(test==-1){

r=rand()%9+0; //Repeat of lines 143/144 until open to place ship on grid

test=match(rTaken,(max-1),r);

}

}

rTaken[j]=r; //Update the dynamic array with a new value to compare with after each iteration

c=rand()%6+1; //Random column from 1 to 6 to stay within grid

for(int i=c;i<(c+size);i++)

grid[r][i]='o'; //Add current ship to the grid

j++;

}

else{

c=rand()%10+1; //Random column from 1 to 10 to stay within grid

r=rand()%7+0; //Random row from 0 to 7 to stay within grid

test=match2(grid,rows,r,c); //Compare random coordinates to coordinates already occupied

if(test==-1){

while(test==-1||test1==-1){

c=rand()%10+1; //Repeat lines 158/159/160

r=rand()%7+0;

test=match2(grid,rows,r,c);

}

}

for(int i=r;i<(r+size);i++)

grid[i][c]='o'; //Add current ship to the grid

k++;

}

}

}

file.close(); //Close file

binFile.close();

delete []rTaken; //Deallocate memory from dynamic array

}

int conv(char &a){

char b=toupper(a); //Make every character capitalized

int r=static\_cast<int>(b-64); //Subtract from decimal ASCII value to get grid axis

return r;

}

char revConv(int &a){

int b=a+64; //Add 64 to find char correspondent

char r=static\_cast<char>(b); //Cast the decimal int as a char

return r;

}

int match(const int array[],const int tot,const int n){

int ret=n;

for(int i=0;i<tot;i++){

if(n==array[i])

ret=-1; //If random coordinate already used/placed in array, return -1

}

return ret;

}

int match2(const char grid[][COLS],const int max,const int r,const int c){

int ret=0;

if(grid[r-1][c]=='o'||grid[r][c]=='o'||grid[r+1][c]=='o'||grid[r+2][c]=='o'||grid[r+3][c]=='o')

ret=-1; //If random coordinates are already used/placed in array, return -1

if(grid[r-1][c+1]=='o'||grid[r][c+1]=='o'||grid[r+1][c+1]=='o'||grid[r+2][c+1]=='o'||grid[r+3][c+1]=='o')

ret=-1; //If random coordinates are already used/placed in array, return -1

if(grid[r-1][c-1]=='o'||grid[r][c-1]=='o'||grid[r+1][c-1]=='o'||grid[r+2][c-1]=='o'||grid[r+3][c-1]=='o')

ret=-1; //If random coordinates are already used/placed in array, return -1

return ret;

}

void play(char uGrid[][COLS],char tGrid[][COLS],char cGrid[][COLS],char ctGrid[][COLS],topRow \*topArr,const int rows){

int res;

int i=0;

cout<<"User starts. Guess the location of an enemy ship!"<<endl;

while(i<(rows\*(COLS-1))){

uTurn(tGrid,cGrid);

res=chckEnd(cGrid,rows);

if(res==-1){

cout<<endl<<"You have sunk all of the computer's ships, and won the game!"<<endl;

return;

}

cTurn(uGrid,ctGrid);

res=chckEnd(uGrid,rows);

if(res==-1){

cout<<endl<<"Oh no! The computer has sunk your fleet! Computer has won the game."<<endl;

return;

}

cout<<"My Grid"<<endl;

prnGrid(uGrid,topArr,rows);

cout<<endl<<"Target Grid"<<endl;

prnGrid(tGrid,topArr,rows);

i++;

}

}

void uTurn(char tGrid[][COLS],char cGrid[][COLS]){

int r, //Row input

c, //Column input

res, //Get return from check uGuess and cGuess functions

res2, //Get return from sunk function

hit; //Get return from hitOrMiss

char cChar; //Hold char to switch between integer values

cout<<endl<<"Enter the x-coordinate (Letter A through J, not case sensitive): ";

cin>>cChar;

if(cChar<65||cChar>74&&cChar<97||cChar>106){ //If char received is not decimal representation of A to J

while(cChar<65||cChar>74&&cChar<97||cChar>106){

cout<<"Not a valid x-coordinate. Please enter a letter A through J: ";

cin>>cChar;

}

}

c=conv(cChar); //Convert char to integer

cout<<"Enter the y-coordinate (Number 0 through 9): ";

cin>>r;

if(r<0||r>9){ //If integer received is not between 0 and 9

while(r<0||r>9){

cout<<"Not a valid y-coordinate. Please enter a number 0 through 9: ";

cin>>r;

}

}

res=reGuess(tGrid,r,c);

if(res==-1){

while(res==-1){

cout<<"You have previously guessed that coordinate. Please enter a new coordinate"<<endl<<

"Enter the x-coordinate (Letter A through J, not case sensitive): ";

cin>>cChar;

if(cChar<65||cChar>74&&cChar<97||cChar>106){ //If char received is not decimal representation of A to J

while(cChar<65||cChar>74&&cChar<97||cChar>106){

cout<<"Not a valid x-coordinate. Please enter a letter A through J: ";

cin>>cChar;

}

}

c=conv(cChar);

cout<<"Enter the y-coordinate (Number 0 through 9): ";

cin>>r;

if(r<0||r>9){ //If integer received is not between 0 and 9

while(r<0||r>9){

cout<<"Not a valid y-coordinate. Please enter a number 0 through 9: ";

cin>>r;

}

}

res=reGuess(tGrid,r,c);

}

}

cout<<endl;

hit=hitOrNo(cGrid,r,c); //Call function to determine if guess is a hit or not

if(hit!=-1){

cout<<"You have hit an enemy boat at ("<<cChar<<

","<<r<<")!"<<endl;

cGrid[r][c]='X'; //Mark x on computer's grid if ship is hit

res2=sunk(cGrid,r,c);

if(res2==-1)

cout<<"You have sunk an enemy ship!"<<endl;

tGrid[r][c]='X'; //Mark x for hit in target grid

}

else{

cout<<"You missed at ("<<cChar<<","<<r<<")."<<endl;

tGrid[r][c]='m'; //Mark m for miss in target grid

}

}

void cTurn(char uGrid[][COLS],char ctGrid[][COLS]){

int r, //Row input

c, //Column input

res, //Get return from check uGuess and cGuess functions

res2, //Get return from sunk function

hit; //Get return from hitOrMiss

char cChar; //Hold char to switch between integer values

c=rand()%10+1; //Computer picks random x-axis between 1 and 10 (A and J)

cChar=revConv(c); //Convert x-axis integer to char

r=rand()%9+0; //Computer picks random y-axis between 0 and 9

res=reGuess(ctGrid,r,c);

if(res==-1){

while(res==-1){

c=rand()%10+1; //Computer picks random x-axis between 1 and 10 (A and J)

cChar=revConv(c); //Convert x-axis integer to char

r=rand()%9+0; //Computer picks random y-axis between 0 and 9

res=reGuess(ctGrid,r,c);

}

}

hit=hitOrNo(uGrid,r,c); //Call to determine if guess is hit or miss

if(hit!=-1){

cout<<"The computer has hit your boat at ("<<cChar<<

","<<r<<")!"<<endl<<endl;

uGrid[r][c]='X'; //Mark x in user's grid if user ship is hit

res2=sunk(uGrid,r,c);

if(res2==-1)

cout<<"The enemy has sunk one of your ships!"<<endl<<endl;

ctGrid[r][c]='X'; //Fill computer's target grid

}

else{

cout<<"The computer missed at ("<<cChar<<","<<r<<")."<<endl<<endl;

ctGrid[r][c]='m'; //Fill computer's target grid

}

}

int hitOrNo(char grid[][COLS],const int row,const int col){

int ret=1;

if(grid[row][col]=='o') //If part of ship resides in the guessed coordinate

grid[row][col]=='X'; //Mark x in grid if ship is hit

else

ret=-1; //Return -1 if miss

return ret;

}

int reGuess(char grid[][COLS],const int r,const int c){

int ret=1;

if(grid[r][c]=='X'||grid[r][c]=='m')

ret=-1; //If coordinate shows up as x or m, coordinate has been previously guessed

return ret;

}

int chckEnd(char grid[][COLS],const int rows){

int ret=-1;

for(int i=0;i<rows;i++){

for(int j=1;j<COLS;j++){

if(grid[i][j]=='o') //If no coordinate holds an 'o', then all ships have sunk

ret=1;

}

}

return ret;

}

int sunk(char grid[][COLS],const int r,const int c){

int ret=1;

char hold;

int hit=0;

for(int i=(r-2);i<(r+2);i++){

if(i>=0&&i<10){

hold=grid[i][c];

if(hold=='X')

hit++;

if(hold!='X')

hit=0;

if(hit==3)

return -1; //Ship is sunk if 3 consecutive vertical coordinates are X's

else if(hit==2&&grid[i+1][c]!='o')

return -1; //Ship is sunk if 2 consecutive vertical coordinates are X's with no bordering 'o'

}

}

hit=0;

for(int i=(c-4);i<(c+4);i++){

if(i>=1&&i<=10){

hold=grid[r][i];

if(hold=='X')

hit++;

if(hold!='X')

hit=0;

if(hit==5)

return -1; //Ship is sunk if 5 consecutive horizontal coordinates are X's

else if(hit==4&&grid[r][i+1]!='o')

return -1; //Ship is sunk if 4 consecutive horizontal coordinates are X's with no bordering 'o'

}

}

return ret;

}