Project 2

Title:

**Battleship**

Due Date:

December 13, 2017

Author:

Jorge Rubi

**Title:**

Battleship

**Introduction:**

Battleship is a game where the player takes turns attempting to guess the location of their opponent’s hidden ships which are scattered throughout a gridded 10x10 game board typically labeled using alphabetical letters along the Y-axis and numbers along the X-axis. The objective of the game is to be the first to find the location of all your opponent’s battleships, which allows the player to “sink” the battleship once its area is located. The first to sink all of their opponent’s battleships wins!

**How to play:**

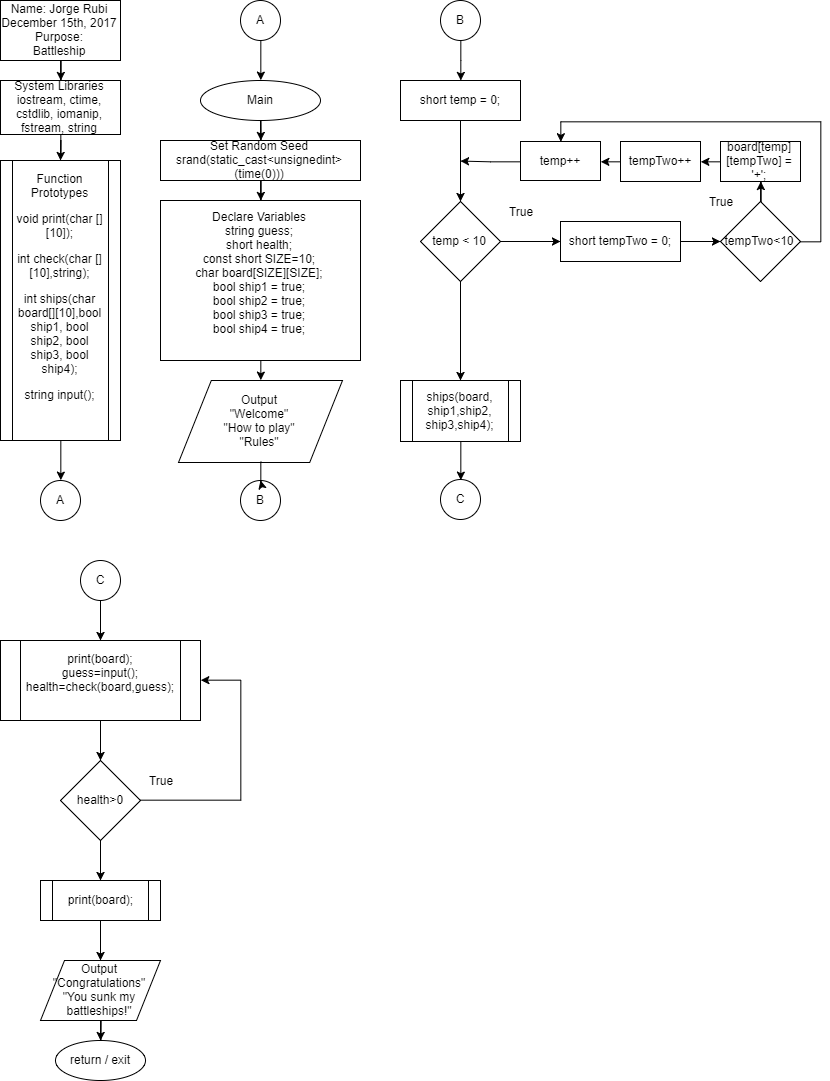
In this c++ version of the game, the user is the one facing off against the computer as they attempt to guess the location of the computer’s ships. The game starts off by introducing the player to the objective of the game as well as giving them a couple instructions on how they should input their guesses. Additionally, the program provides descriptions for what certain symbols represent throughout the game (ie: a “+” is an unexplored location on the board, an “\*” is a location which has been guessed but was found to have no ships aka a “miss”, and an “X” marks an explored location which happened to have a ship aka a “hit”!). The program will then present the game board to the user and will constantly update it as the user attempts to guess various locations and eventually beat the game. To conclude, the user must enter their guesses in a “B4”,”C5”,”F7” manner and they must acquire a total of 14 “X”’s or “hits” to beat the game! It’s that simple.

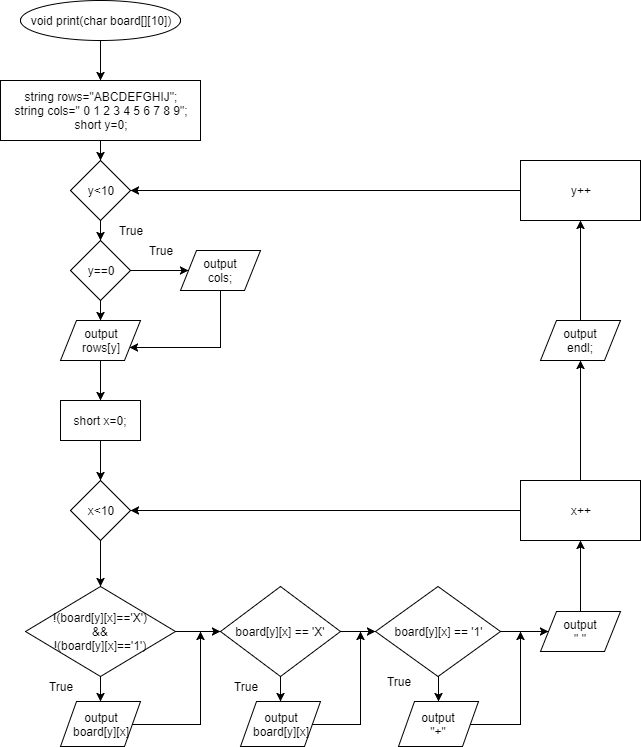
**Summary:**

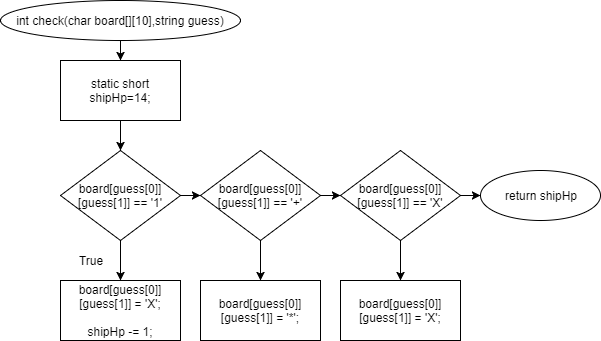
As I made this program I wanted to completely upgrade and convert my first project into something much more functional and advanced. I was able to do so by implementing loads more features into my project as well as functions and multidimensional arrays which functioned a lot more efficiently and effectively.

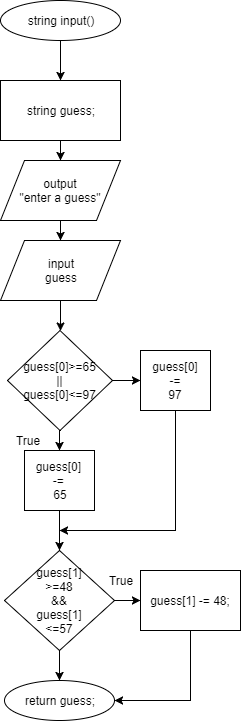
**Project Size:** about 380 lines

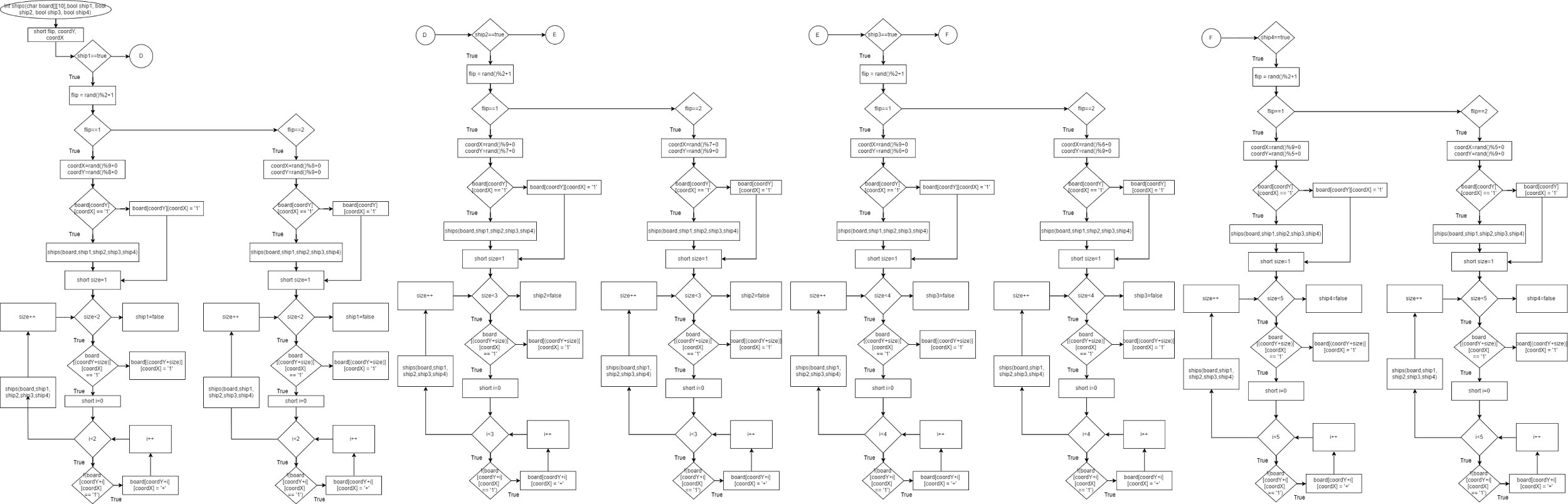
**Flow chart:**

****

****

****

****

****

**Pseudo Code:**

The program begins by creating a 10x10 array and proceeds to fill it with plus signs to act as placeholders. It then presents the user with a greeting and calls on a ship generation function. The ship generation function creates 4 ships, each of a different size, which generates randomly on the game board. If an overlap is detected during ship gen the program regens the ship which overlapped and re-attempts to create the ship in an area which does not overlap. The game then proceeds to continually request input of the user and will progressively display whether the user hits a ship or misses until the user can sink all of the battleships!

**References:**

1. Cplusplus the online resource
2. Gaddis 8th Edition

**Code:**

/\*

\* File: main.cpp

\* Author: Jorge Rubi

\* Created on November 4, 2017, 7:00 PM

\* Purpose: Project 2 : Battleship!

\* Functions edition

\* Independent boat generation

\* Anti-overlap generation

\*/

//System Libraries

#include <iostream> //I/O Stream Library

#include <ctime>

#include <cstdlib>

#include <iomanip>

#include <fstream>

#include <string>

using namespace std; // The standard name-space for system libraries

//User Libraries

//Global constants - Physics/Math/Conversions only

//Functions Prototypes

void print(char [][10]);

int check(char [][10],string);

int ships(char board[][10],bool ship1, bool ship2, bool ship3, bool ship4);

string input();

//Execution Begins Here!

int main() {

//Random # seed

srand(static\_cast<unsigned int>(time(0)));

//Variable Declaration

string guess;

short health;

//Create an array for the game board.

const short SIZE = 10;

char board[SIZE][SIZE]; //Array to create the functional game board

//Expressions to ensure a ship is generated as ship function is called

bool ship1 = true; //Size 2

bool ship2 = true; //Size 3

bool ship3 = true; //Size 4

bool ship4 = true; //Size 5

//Input / Variable Initialization

cout<<" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ "<<endl;

cout<<"| W E L C O M E |"<<endl;

cout<<"| T O |"<<endl;

cout<<"| BATTLE SHIP |"<<endl;

cout<<"|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|"<<endl;

cout<<"To play, enter your guess for the ";

cout<<"location of your opponent's ships!"<<endl;

cout<<"You can do so in a 'B4', 'F7', and 'D9' manner."<<endl;

cout<<"When you hit a ship the location will be marked with an 'X'!"<<endl;

cout<<"When you miss, an asterisk '\*' will appear at that location."<<endl;

cout<<"The game ends when all ships have been sunk."<<endl;

cout<<"Good luck!"<<endl<<endl;

//Process / Calculations

//Board reset

for(short temp = 0; temp < 10; temp++){

for(short tempTwo = 0; tempTwo < 10; tempTwo++){

board[temp][tempTwo] = '+';

}

}

for(short temp = 0; temp < 10; temp++){

for(short tempTwo = 0; tempTwo < 10; tempTwo++){

board[temp][tempTwo] = '+';

}

}

//Places hidden battleships on the game board

ships(board,ship1,ship2,ship3,ship4);

do{

print(board); //Prints game board

guess=input(); //Allows for user input

health=check(board,guess);

cout<<endl;

}while(health>0);

print(board); //Print the final state of the board once final ship is sunk

//Output

cout<<endl<<endl;

cout<<"Congratulations!"<<endl;

cout<<"You sunk my battleships!"<<endl;

//Exit function main, end of program

return 0;

}

void print(char board[][10]){

string rows = "ABCDEFGHIJ";

string cols = " 0 1 2 3 4 5 6 7 8 9";

//Creates the displayed game board

for(short y=0; y<10; y++){

if(y==0){

cout<<cols<<endl;

}

cout<<rows[y]<<" ";

for(short x=0; x<10; x++){

//Prints board according to Hit or Miss

//Prints '+' or '\*' by default

if((!(board[y][x] == 'X'))&&(!(board[y][x] == '1'))){

cout<<board[y][x];

}

else if(board[y][x] == 'X'){ //Prints X when a ship has been hit

cout<<board[y][x];

}

else if(board[y][x] == '1'){ //Conceals the ship locations

cout<<"+";

//Note for testing: cout<<board[y][x];

}

cout<<" ";

}

cout<<endl;

}

}

int check(char board[][10],string guess){

static short shipHp = 14;

//If ship placement lies in guessed location, "Hit"!

if(board[guess[0]][guess[1]] == '1'){

board[guess[0]][guess[1]] = 'X';

shipHp-=1;

}

//If ship placement does not lie in guessed location, "Miss"!

else if(board[guess[0]][guess[1]] == '+'){

board[guess[0]][guess[1]] = '\*';

}

else if(board[guess[0]][guess[1]] == 'X'){

board[guess[0]][guess[1]] = 'X';

}

return shipHp;

}

string input(){

string guess;

//User inputs a position they hope a battleship to lie in

cout<<"Enter a guess: ";

cin>>guess;

cin.ignore();

//Convert guessed letter into readable numbers for game

//Board starts at 0, Adjust Y due to ASCII character values

(guess[0]>=65||guess[0]<=97) ? (guess[0] -= 65) : (guess[0] -= 97);

//Adjust X due to ASCII character values

if(guess[1]>=48&&guess[1]<=57){

guess[1] -= 48;

}

return guess;

}

int ships(char board[][10],bool ship1, bool ship2, bool ship3, bool ship4){

//Initialize ship creation functions

short flip; //Temporary value for determining orientation of a ship

short coordY,coordX;//Temporary values for setting ship coordinates on board

//Ship 1 - Size 2

if(ship1==true){ //Generates a coin toss for orientation of ship

flip = rand()%2+1; //[1,2]

if(flip==1){ //Vertical Position

coordX = rand()%9+0; //Creates coordinates for ships

coordY = rand()%8+0; //[0,8] to avoid board collision

if(board[coordY][coordX] == '1'){ //Checks for generation collision

ships(board,ship1,ship2,ship3,ship4); //Regenerates if collision

}

else{

board[coordY][coordX] = '1'; //Places ship position on board

}

for(short size=1;size<2;size++){ //Completes the ship's size

if(board[(coordY+size)][coordX] == '1'){

for(short i=0;i<2;i++){

if(!(board[coordY+i][coordX] == '1')){

board[coordY+i][coordX] = '+'; //reset the ship spawn

}

}

ships(board,ship1,ship2,ship3,ship4);//Re-attempt generation

}

else{

board[(coordY+size)][coordX] = '1';

}

}

}

if(flip==2){ //Horizontal Position1

coordX = rand()%8+0; //Creates coordinates for ships

coordY = rand()%9+0; //[0,8] to avoid board collision

if(board[coordY][coordX] == '1'){ //Checks for generation collision

ships(board,ship1,ship2,ship3,ship4); //Regenerates if collision

}

else{

board[coordY][coordX] = '1'; //Places ship position on board

}

for(short size=1;size<2;size++){ //Completes the ship's size

if(board[coordY][(coordX+size)] == '1'){

for(short i=0;i<2;i++){

if(!(board[coordY][coordX+i] == '1')){

board[coordY][coordX+i] = '+'; //reset the ship spawn

}

}

ships(board,ship1,ship2,ship3,ship4);//Re-attempt generation

}

else{

board[coordY][(coordX+size)] = '1';

}

}

}

ship1=false;

}

//Ship 2 - Size 3

if(ship2==true){

flip = rand()%2+1; //[1,2]

if(flip==1){ //Vertical

coordX = rand()%9+0;

coordY = rand()%7+0; //[0,7] to avoid board collision

if(board[coordY][coordX] == '1'){ //Checks for generation collision

ships(board,ship1,ship2,ship3,ship4); //Regenerates if collision

}

else{

board[coordY][coordX] = '1'; //Places ship position on board

}

for(short size=1;size<3;size++){ //Completes the ship's size

if(board[(coordY+size)][coordX] == '1'){

for(short i=0;i<3;i++){

if(!(board[coordY+i][coordX] == '1')){

board[coordY+i][coordX] = '+';//reset the ship spawn

}

}

ships(board,ship1,ship2,ship3,ship4);//Re-attempt generation

}

else{

board[(coordY+size)][coordX] = '1';

}

}

}

if(flip==2){ //Horizontal Position

coordX = rand()%7+0; //[0,7] to avoid board collision

coordY = rand()%9+0;

if(board[coordY][coordX] == '1'){ //Checks for generation collision

ships(board,ship1,ship2,ship3,ship4); //Regenerates if collision

}

else{

board[coordY][coordX] = '1'; //Places ship position on board

}

for(short size=1;size<3;size++){ //Completes the ship's size

if(board[coordY][(coordX+size)] == '1'){

for(short i=0;i<3;i++){

if(!(board[coordY][coordX+i] == '1')){

board[coordY][coordX+i] = '+';//reset the ship spawn

}

}

ships(board,ship1,ship2,ship3,ship4);//Re-attempt generation

}

else{

board[coordY][(coordX+size)] = '1';

}

}

}

ship2=false;

}

//Ship 3 - Size 4

if(ship3==true){

flip = rand()%2+1; //[1,2]

if(flip==1){ //Vertical Position

coordX = rand()%9+0;

coordY = rand()%6+0; //[0,6] to avoid board collision

if(board[coordY][coordX] == '1'){ //Checks for generation collision

ships(board,ship1,ship2,ship3,ship4); //Regenerates if collision

}

else{

board[coordY][coordX] = '1'; //Places ship position on board

}

for(short size=1;size<4;size++){ //Completes the ship's size

if(board[(coordY+size)][coordX] == '1'){

for(short i=0;i<4;i++){

if(!(board[coordY+i][coordX] == '1')){

board[coordY+i][coordX] = '+';//reset the ship spawn

}

}

ships(board,ship1,ship2,ship3,ship4);//Re-attempt generation

}

else{

board[(coordY+size)][coordX] = '1';

}

}

}

if(flip==2){ //Horizontal Position

coordX = rand()%6+0; //[0,6] to avoid board collision

coordY = rand()%9+0;

if(board[coordY][coordX] == '1'){ //Checks for generation collision

ships(board,ship1,ship2,ship3,ship4); //Regenerates if collision

}

else{

board[coordY][coordX] = '1'; //Places ship position on board

}

for(short size=1;size<4;size++){ //Completes the ship's size

if(board[coordY][(coordX+size)] == '1'){

for(short i=0;i<4;i++){

if(!(board[coordY][coordX+i] == '1')){

board[coordY][coordX+i] = '+';//reset the ship spawn

}

}

ships(board,ship1,ship2,ship3,ship4);//Re-attempt generation

}

else{

board[coordY][(coordX+size)] = '1';

}

}

}

ship3=false;

}

//Ship 4 - Size 5

if(ship4==true){

flip = rand()%2+1; //[1,2]

if(flip==1){ //Vertical

coordX = rand()%9+0;

coordY = rand()%5+0; //[0,5] to avoid board collision

if(board[coordY][coordX] == '1'){ //Checks for generation collision

ships(board,ship1,ship2,ship3,ship4); //Regenerates if collision

}

else{

board[coordY][coordX] = '1'; //Places ship position on board

}

for(short size=1;size<5;size++){ //Completes the ship's size

if(board[(coordY+size)][coordX] == '1'){

for(short i=0;i<5;i++){

if(!(board[coordY+i][coordX] == '1')){

board[coordY+i][coordX] = '+';//reset the ship spawn

}

}

ships(board,ship1,ship2,ship3,ship4);//Re-attempt generation

}

else{

board[(coordY+size)][coordX] = '1';

}

}

}

if(flip==2){ //Horizontal

coordX = rand()%5+0; //[0,5] to avoid board collision

coordY = rand()%9+0;

if(board[coordY][coordX] == '1'){ //Checks for generation collision

ships(board,ship1,ship2,ship3,ship4); //Regenerates if collision

}

else{

board[coordY][coordX] = '1'; //Places ship position on board

}

for(short size=1;size<5;size++){ //Completes the ship's size

if(board[coordY][(coordX+size)] == '1'){

for(short i=0;i<5;i++){

if(!(board[coordY][coordX+i] == '1')){

board[coordY][coordX+i] = '+';//reset the ship spawn

}

}

ships(board,ship1,ship2,ship3,ship4);//Re-attempt generation

}

else{

board[coordY][(coordX+size)] = '1';

}

}

}

ship4=false;

}

}