Project 1

CIS-17A (45434)

**Battleship**

Due Date:

July 17, 2018

Author:

Jorge Rubi

**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 10x10 gridded 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. Once you manage to do so, you win!

**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 17 “X”’s or “hits” to beat the game! It’s that simple.

**Summary:**

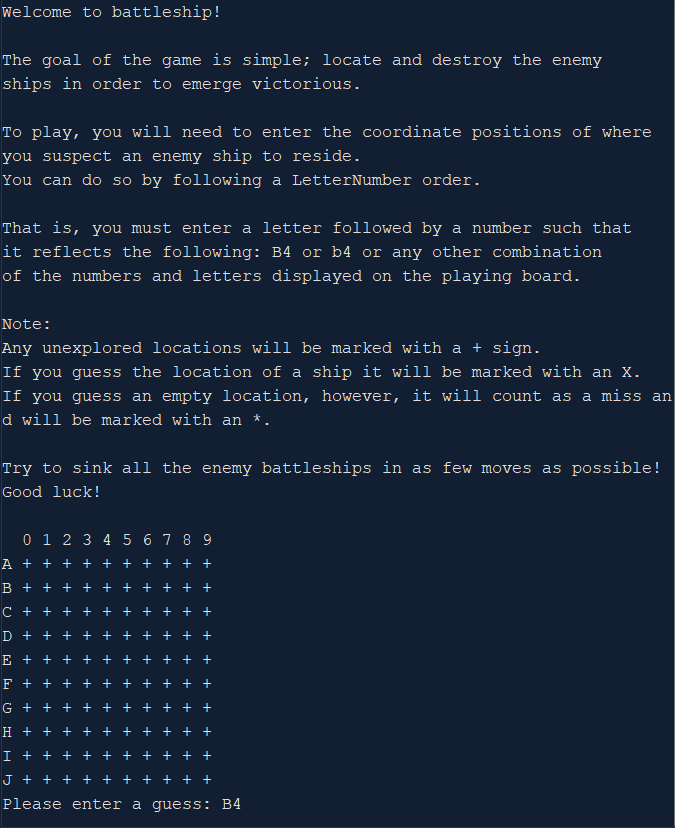
As I made this program I wanted to completely upgrade and convert my first project into something much more functional and advanced. I was able to do so by implementing loads more features into my project as well as functions and multidimensional arrays which functioned a lot more efficiently and effectively.

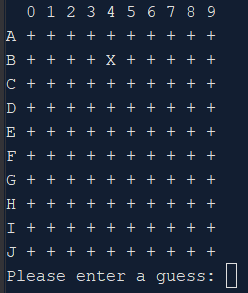
**Project Size:** about 400 lines

**Description:**

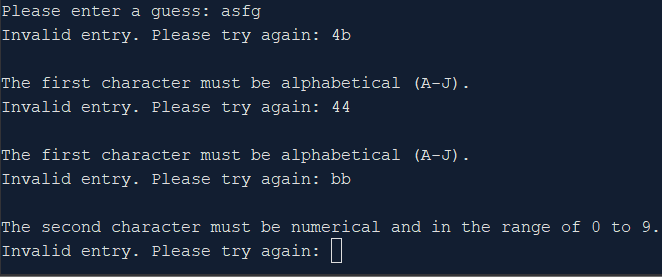
**Sample Input/Output:**

Input: B4 (enter)



Output updates accordingly:  
 

Invalid input yields error prompts to the user:



**Variables:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable Name** | **Description** | **Location** |
| Board | arr | Holds game board values and properties | main(), init(), shipGen(), run(), print(), update() |
|  | arrray | Initializes game board to a blank state | bordGen() |
| Vars | ships | Array of structures to hold ship properties | shipGen() |
| Integer | size | Holds the game board size | bordGen(), print() |
|  | sizeTemp | Stores size of overlap, if there happens to be any during generation | shipGen() |
|  | xVal | Stores random x value for generation | shipGen() |
|  | yVal | Stores random y value for generation | shipGen() |
|  | count | Used to undo ship placement in case of overlap | shipGen() |
|  | moves | Counts the number of moves taken by the user | run() |
| Char | guess | Pointer to hold the user's guess for the enemy ship position | run() |
|  | temp | C-string of processed user input | Input() |
| bool | gen | Ensures proper ship generation | shipGen() |
|  | valid | Ensure proper input validation | Input(), endGame(), |
| string | letters | Holds string of letters for output on the Y axis | print() |
|  | str | Holds raw user input prior to validation and processing | Input() |
|  | name | Holds user’s initials for highscore file | endGame() |
| fstream | file | Variable used for file I/O | endGame() |

**Concepts:**

The majority of new concepts applied in this project involved structures. Getting used to the syntax and incorporating two-dimensional arrays, arrays of structures, and similar ideas took a bit of getting used to. Upon completing this project, however, I felt a lot more comfortable with structures; as well as classes, despite not incorporating them into this project. The majority of the newfound structure syntax can be found in the shipGen function. Additionally, some newfound file I/O syntax can be found within the endGame function.

**References:**

1. Cplusplus the online resource
2. Gaddis 9th Edition
3. CIS-5 Project 2 (For algorithm/order ideas – no code was borrowed)

**Program:**

/\*

\* File: main.cpp

\* Author: Jorge Rubi

\* Purpose: Project 1 : Battleship, Revamped

\*/

//System Libraries Here

#include <iostream>

#include <ctime> //For pseudo-random number generation

#include <cctype> //For input validation

#include <fstream> //For file I/O

using namespace std;

//User Libraries Here

#include "GAME.h"

//Global Constants Only, No Global Variables

//Like PI, e, Gravity, or conversions

//Function Prototypes Here

//Note: For Project 2 plan to create user-board and playability versus computer

Board run(Board &); //Runs program and maintains processes

Board init(); //Calls for board gen and ship gen

void print(Board &); //Print the playing board (Executes throughout game for UI)

Board bordGen(); //Generate the playing board (Declare, Initialize, Pass)

Board shipGen(Board &); //Generate hidden ship positions on board (Occurs once)

Board update(char \*, Board &); //Update the playing board & check game conditions

char \*input(string); //Get user input as string and convert to chars

void endGame(int); //Get user ID and number of moves taken to win game

void intro(); //Outputs an introduction and tells player the rules

//Program Execution Begins Here

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

//Random number seed

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

Board arr;

//Initialize game variables

arr = init();

//Output Prompt

intro();

//Run the game

run(arr);

//Exit

return 0;

}

Board init(){

//Intialize the playing board once

Board arr = bordGen();

//Generate enemy battleship positions on the playing board once

shipGen(arr);

return arr;

}

Board run(Board &arr){

//Play the game!

int moves = 0; //Count the number of moves made by the user

char \*guess; //Hold the user's guess for the enemy ship position

do{

print(arr);

guess = input("a guess"); //Get user input

arr = update(guess,arr);

moves++;

}while(arr.hp > 0); //will return false when all enemy ships have been sunk

//Output final board state

print(arr);

//Record high score then end the game

endGame(moves);

}

Board bordGen(){

//Executes once to initialize the playing board

Board array;

int size = array.boardSize;

//Loop initializes character array spaces to '+' as a placeholder

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

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

array.board[i][j] = '+';

}

}

return array;

}

Board shipGen(Board &bArr){

//Generate ships and their properties

////////////////////////////////////////////////////////////////////////////

//Create ships variable

Vars ships;

//Dynamic allocation of ship quantity (create array of ship structures)

ships.array = new Ship[ships.size];

//Boolean to determine generation success

bool gen = true;

//Temporary variables for ship placement

int sizeTemp, xVal, yVal, count;

//Create coordinate array as such to store the ship's starting point

/\* 0|1 |2

\* s0|y0|x0

\* s1|y1|x1

\* s2|y2|x2

\* s3 ...

\* s4 ...

\*/

ships.array->coords[ships.size][3];

//Loop until all ships are created

for(int i = 0; i <= ships.size - 1; i++){

//If the ship orientation is 0: horizontal gen. if 1: vertical gen.

ships.array[i].orient = rand() % 2;

//Set the length of the ship, independent of orientation

ships.array[i].length = ships.lngthArr[i];

//Reset temporary variables for each ship loop

sizeTemp = 0, xVal = 0, yVal = 0, count = 0;

//Loop until valid ship placement

do{

//"Spawnpoint" generation

////////////////////////////////////////////////////////////////////

//Generate random y value & store the y coordinate

ships.array[i].coords[i][1] = rand() % 10;

//Generate random x value & store the x coordinate

ships.array[i].coords[i][2] = rand() % 10;

////////////////////////////////////////////////////////////////////

//Attempt horizontal generation

////////////////////////////////////////////////////////////////////

if(ships.array[i].orient == 0){

//Edge collision detection through sufficient space checking

sizeTemp = bArr.boardSize - ships.array[i].coords[i][2] - ships.array[i].length;

//If there isn't enough space, move back the spawn point

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

ships.array[i].coords[i][2] += sizeTemp;

}

//After potentially updating the coordinate values

//due to space checking, define the temporary placement vars

yVal = ships.array[i].coords[i][1];

xVal = ships.array[i].coords[i][2];

//Begin ship placement

for(int x = 0; x < ships.array[i].length; x++){

//if bArr == '1', undo everything, gen = false, break

if(bArr.board[yVal][xVal + x] == '1'){

//Count what position generation was at in order to undo

count = x;

//Set gen to false so the function may try again

gen = false;

break;

}

//If there is a free '+' space, place a ship marker

else if(bArr.board[yVal][xVal + x] == '+'){

bArr.board[yVal][xVal + x] = '1';

//Set gen to true to avoid undo loop

gen = true;

}

}

//if generation unsuccessful (gen == false) undo ship placement

if(!gen){

for(int x = count; x > ships.array[i].coords[i][2]; x--){

bArr.board[yVal][xVal + x] = '+';

}

}

}

////////////////////////////////////////////////////////////////////

//Attempt vertical generation

////////////////////////////////////////////////////////////////////

if(ships.array[i].orient == 1){

//Edge collision detection through sufficient space checking

sizeTemp = bArr.boardSize - ships.array[i].coords[i][1] - ships.array[i].length;

//If there isn't enough space, move back the spawn point

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

ships.array[i].coords[i][1] += sizeTemp;

}

//After potentially updating the coordinate values

//due to space checking, define the temporary placement vars

yVal = ships.array[i].coords[i][1];

xVal = ships.array[i].coords[i][2];

//Begin ship placement

for(int y = 0; y < ships.array[i].length; y++){

//Overlap detection

if(bArr.board[yVal + y][xVal] == '1'){

//Count what position generation was at in order to undo

count = y;

//Set gen to false so the function may try again

gen = false;

break;

}

//If there is a free '+' space, place a ship marker

else if(bArr.board[yVal + y][xVal] == '+'){

bArr.board[yVal + y][xVal] = '1';

//Set gen to true to avoid undo loop

gen = true;

}

}

//if generation unsuccessful (gen == false) undo ship placement

if(!gen){

for(int y = count; y > ships.array[i].coords[i][1]; y--){

bArr.board[yVal + y][xVal] = '+';

}

}

}

////////////////////////////////////////////////////////////////////

}while(!gen);

////////////////////////////////////////////////////////////////////////////

//Debugging line reveals ship generation coordinates

//cout<<ships.array[i].coords[i][1]<<ships.array[i].coords[i][2]<<endl;

}

return bArr;

}

void print(Board &array){

string letters = "ABCDEFGHIJ";

//Output Nums on top of board

////////////////////////////////////////////////////////////////////////////

for(int i = 0; i < 10; i++){

if(i == 0 ) cout << " ";

cout << " " << i;

if(i == 9 ) cout << endl;

}

////////////////////////////////////////////////////////////////////////////

//Output the playing board in its current state

////////////////////////////////////////////////////////////////////////////

int size = array.boardSize;

for(int y = 0; y < size; y++){

//Output letters on left side of board

cout << letters[y] << " ";

//Output board characters

for(int x = 0; x < size; x++){

if(array.board[y][x] == '1'){

cout << "+" << " "; //"Hide" the ship positions to user

}

else{

cout << array.board[y][x] << " "; //Output the char ie: +,\*,X

}

}

cout << endl;

}

////////////////////////////////////////////////////////////////////////////

}

char \*input(string action){

bool valid = true;

string str = "";

char \*temp = new char[2]; //For C-string

int size;

//Input validation

///////////////////////////////////////////////////////////////////////////

cout << "Please enter " << action << ": ";

do{

if(!valid){

cout << "Invalid entry. Please try again: ";

}

cin >> str;

cin.ignore();

size = str.length();

valid = true;

if(size > 2){

valid = false;

continue;

}

//Check if values entered are alphanumeric

if(isalnum(str[0]) || isalnum(str[1])){

//Check if the first character is alphabetical and do changes

if(isalpha(str[0])){

str[0] = tolower(str[0]);

if(str[0]>'j'){

valid = false;

continue;

}

else{

str[0] -= 49;

}

}

else{

cout << endl;

cout << "The first character must be alphabetical (A-J). " << endl;

valid = false;

continue;

}

//Check if second character is a number and if it's <= 9

if(!(isdigit(str[1]) && (str[1] > 9))){

cout << endl;

cout << "The second character must be numerical and in the";

cout << " range of 0 to 9. " << endl;

valid = false;

continue;

}

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

temp[j] = str[j];

}

}

else{

valid = false;

}

}while(!valid);

///////////////////////////////////////////////////////////////////////////

return temp;

}

Board update(char \*guess, Board &arr){

//Convert chars into ints and set them equal to temp variables

int y = guess[0] - '0';

int x = guess[1] - '0';

//If hit

if(arr.board[y][x] == '1'){

arr.board[y][x] = 'X';

arr.hp--;

}

//If miss

if(arr.board[y][x] == '+'){

arr.board[y][x] = '\*';

}

//Note: If already hit ('X') or a miss ('\*'), do nothing.

return arr;

}

void endGame(int moves){

string name = "";

bool valid = true;

int size;

fstream file;

cout << "Congratulations, you sunk my battleships!" << endl;

cout << "Your total moves were: " << moves << endl;

cout << "Enter your initials to add yourself to the leaderboard. " << endl;

cout << "(Note: Your initials must be 3 characters) : ";

do{

if(!valid){

cout << "Invalid entry. Please try again: ";

}

valid = true;

getline(cin, name);

size = name.length();

//Ensure initials are 3 characters long

if(!(size==3)){

valid = false;

continue;

}

//Ensure characters are alphabetical

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

if(!(isalpha(name[i]))){

valid = false;

continue;

}

}

}while(!valid);

cout << "Your highscore has been saved. Thanks for playing! \n" << endl;

//Open or create highscores file

file.open("highscores.txt", fstream::in|fstream::out|fstream::app);

file << "High scores: ";

//Add initials to the file

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

file << name[i];

}

file << " with " << moves << " moves! " << endl;

file.close();

}

void intro(){

cout << "Welcome to battleship!" << endl <<endl;

cout << "The goal of the game is simple; locate and destroy the enemy \n"

"ships in order to emerge victorious. \n\n"

"To play, you will need to enter the coordinate positions of where\n"

"you suspect an enemy ship to reside. \n"

"You can do so by following a LetterNumber order.\n\n"

"That is, you must enter a letter followed by a number such that \n"

"it reflects the following: B4 or b4 or any other combination \n"

"of the numbers and letters displayed on the playing board.\n\n"

"Note: \nAny unexplored locations will be marked with a + sign. \n"

"If you guess the location of a ship it will be marked with an X.\n"

"If you guess an empty location, however, it will count as a miss and will be marked with an \*.\n\n"

"Try to sink all the enemy battleships in as few moves as possible!\n"

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

}

/\*

\* File: GAME.h

\* Author: Cokes PC

\*

\*/

using namespace std;

#ifndef GAME\_H

#define GAME\_H

struct Board{

static const int boardSize = 10; //Holds the board size to be used in the array definition

char board[boardSize][boardSize];//Game board

int hp = 17;

};

struct Player{

string name; //Stores player name for leaderboard/highscore output

int score; //Stores player's score

};

struct Ship{

int orient; //Stores ship value for horizontal or vertical generation

int length; //Stores ship length for generation

int coords[][3]; //Stores ship coords for generation

};

struct Vars{

Ship \*array; //Hold array of ships and their values from Ship struct

int size = 5; //Hold size of ships array aka the number of ships

int lngthArr[5] = {2,3,3,4,5}; //Length array to be used in ship generation

};

#endif /\* GAME\_H \*/