|  |
| --- |
| Battleship: The Game |
| Project 1 |
| Ryan Melendez CSC 17A 10/23/15 |

**Introduction:**

The project I created is a two player battleship game simulator in which two players must guess the locations of the other persons ships placed on a 10 x 15 game board. At the start of the game, the program prompts the user, based off player numbers, to input the location of the 5 ships of varying sizes. As each turn is taken, the opposing players board is stored in memory and the proper board pertaining to that player is displayed. I chose to create this program because it was a game that I remember playing growing up and felt that my personal experience from playing this game would allow me to recreate it with accuracy.

Ship Size: Cruiser: 2 units, Frigate: 3 units, Submarine: 3 units, Escort: 4 units, Battleship: 5 units

Game board is 10 x 15:

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14

0 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

1 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

2 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

3 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

4 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

5 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

6 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

7 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

8 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

9 ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~ ~

**Summary:**

Program size: 428 lines

Number of variables: around 20

Number of functions: 8

IDE: Visual Studios 13

Overall this program took me around 1 week to create and for the most part gave me the opportunity to incorporate the materials covered in Chapters 9-12. When it came to the few exceptions, I was able to find a way to demonstrate how they were used in somewhat of a relevant manner.

I enjoyed creating this project because it allowed me to recreate a game I played early on and my life and still to this day. The fact that I now have a playable game on my computer is something I look forward to showing to friends and family as the opportunities arise. Some of the issues I ran into while creating the project were mainly dealing with dynamic memory allocations and pointers soley based on the fact that I had not fully grasped the concepts. I had previously attempted to create this program while taking CSC-5 in the Spring of 2015 but was not able to complete it. I had to expand my previous psuedocode, for the sake of creating a stable game environment.

**Description:**

Refer to attached FlowChart.

**Psuedocode:**

*Start*

*-preset ship values*

*-using a struct[5]*

*-create board*

*-using an array[][]*

*Input*

*-ask players to input location of ships*

*- asks for placement dealing with 2 separate boards*

*-after users input their ships, the game begins*

*- prompts players to input coordinates of attack on the opposing players board*

*- based off x and y coordinates*

*-all of this loops until all of one players ships are sunk*

*Output*

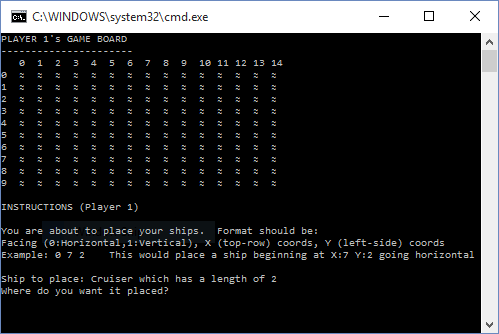
*-outputs the players names and the winner of the two*

*-outputs the total hits and misses*

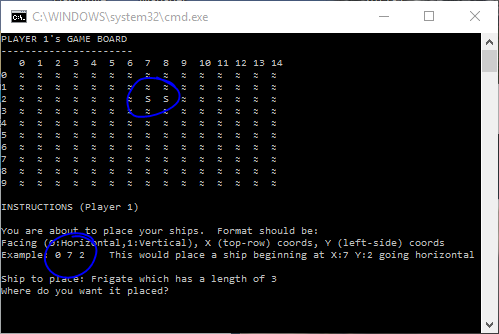
*-outputs them to both txt and dat files*

**Gameplay examples:**

Simple picture of how the game play is initialized

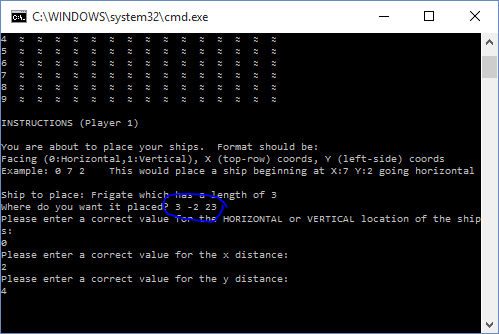
****

Example case: inputting the coordinates of 0 (Horizontal) 7 (X-direction) 2 (Y-direction)



Error Checking: if the user inputs incorrect values, user it asked to input them again

In this case, the incorrect values of (3, -2, 23)



**New Concepts:**

-Chapter 9 Pointers

* Pointer to Variables:
  + Line 248: ptrd - variable for direction (d)
  + Line 249 ptrx – variable for x position (x)
  + Line 250 ptry – variable for y position (y)
* Pointers to Struct:
  + Line 94 - pointer to struct outInfo
  + Line 117 - pointer to struct PlaceShips

-Chapter 10 CStrings

* String Objects
  + Line 387 – compared string objects to alphabetize them

-Chapter 11 Structured Data

* Enumerated Data:
  + Line 48- enum data for horizontal or vertical position
* General Structs:
  + Line 21 – struct POINT
  + Line 29 – struct SHIP
  + Line 41- struct PLAYER
  + Line 49 – struct PLACESHIP
  + Line 55 – struct outInfo
* Array of Structs:
  + Line 39 – ship[5] sets info for 5 ships
  + Line 44 - player[3] sets boards for both players
* Array within a Struct:
  + Line 36 –onGrid[5]
  + Line 38 – hitFlag[5]

-Chapter 12 Advanced File Operations

* Reading from a .txt file
  + Line 90 - calls the function dispIntro
  + Line 413 - within the function accessing the WelcomeMessage file
* Outputting to a .txt file
  + Line 201 – sends info to the function otpt
  + Line 369 – majority of the function outputs the game stats
* Outputting to a .dat file
  + Line 377 – creates dat file
  + Line 378 - sends the names to that dat file

**References**

* Gaddis 8th edition: Getting started with C++
* <http://www.hasbro.com/common/instruct/Battleship.PDF> : Game instructions

**Code**

/\*

Ryan Melendez

Battleship Game

CSC-17A

Visual Studios 13

10/23/15

\*/

#include <iostream>

#include <string>

#include <fstream>

#include <cctype>

using namespace std;

const char isWATER = 247; // ASCII Character Code

const char isHIT = 'X'; // Constant chars used to display the board

const char isSHIP = 'S'; // chose constant since it would be referenced more than once

const char isMISS = '0'; // and it would allow the user to customize the game board to their preference

struct POINT

{

//A location on the grid defined

//by X(horizontal) Y(vertical) coordinates

int X;

int Y;

};

struct SHIP

{

//Ship name

string name;

//Total points on the grid

int length;

//Coordinates of those points

POINT onGrid[5]; //0-4 max length of biggest ship

//Whether or not those points are a "hit"

bool hitFlag[5]; // array within structures

}ship[5]; // array of structs

struct PLAYER

{

char grid[15][10];

}player[3]; //Ignore player 0, just using player's 1 & 2

//array of structs

enum DIRECTION { HORIZONTAL, VERTICAL }; // keep track of which direction to place the ship

struct PLACESHIPS

{

DIRECTION direction;

SHIP shipType;

};

struct OUTINFO

{

int missCount = 0; // keeps track misses

int hitCount = 0; // keeps track of hits

};

bool gameRunning = false;

//Functions

void LoadShips();

void ResetBoard();

void DrawBoard(int);

PLACESHIPS UserInputShipPlacement();

bool UserInputAttack(int&, int&, int);

bool GameOverCheck(int);

void otptStat(int, int, int);

void dispIntro();

int main()

{

string names[2];

dispIntro(); // reads input from a txt file

LoadShips();

ResetBoard();

struct OUTINFO \*ptr,otFile; // pointer to a struct

ptr = &otFile;

//"PLACE SHIPS" phase of game

//Loop through each player...

for (int aplyr = 1; aplyr<3; ++aplyr)

{

//Loop through each ship type to place

for (int thisShip = 0; thisShip<5; thisShip++)

{

//Display gameboard for player

DrawBoard(aplyr);

//Give instructions

cout << "\n";

cout << "INSTRUCTIONS (Player " << aplyr << ")\n\n";

cout << "You are about to place your ships. Format should be:\n";

cout << "Facing (0:Horizontal,1:Vertical), X (top-row) coords, Y (left-side) coords\n";

cout << "Example: 0 7 2 This would place a ship beginning at X:7 Y:2 going horizontal\n\n";

cout << "Ship to place: " << ship[thisShip].name << " which has a length of " << ship[thisShip].length << "\n";

cout << "Where do you want it placed? ";

//Get input from user and loop until good data is returned

PLACESHIPS \*ptr2, aShip;

ptr2 = &aShip;

ptr2->shipType.onGrid[0].X = -1;

while (ptr2->shipType.onGrid[0].X == -1)

{

aShip = UserInputShipPlacement();

}

//Combine user data with "this ship" data

ptr2->shipType.length = ship[thisShip].length;

ptr2->shipType.name = ship[thisShip].name;

//Add the FIRST grid point to the current player's game board

player[aplyr].grid[ptr2->shipType.onGrid[0].X][ptr2->shipType.onGrid[0].Y] = isSHIP;

//Determine ALL grid points based on length and direction

for (int i = 1; i<ptr2->shipType.length; i++)

{

if (ptr2->direction == HORIZONTAL){

ptr2->shipType.onGrid[i].X = ptr2->shipType.onGrid[i - 1].X + 1;

ptr2->shipType.onGrid[i].Y = ptr2->shipType.onGrid[i - 1].Y;

}

if (ptr2->direction == VERTICAL){

ptr2->shipType.onGrid[i].Y = ptr2->shipType.onGrid[i - 1].Y + 1;

ptr2->shipType.onGrid[i].X = ptr2->shipType.onGrid[i - 1].X;

}

//Add the remaining grid points to our current players game board

player[aplyr].grid[ptr2->shipType.onGrid[i].X][ptr2->shipType.onGrid[i].Y] = isSHIP;

}

//Loop back through each ship type

}

//Loop back through each player

}

//Ready to play the game

gameRunning = true;

int thisPlayer = 1;

do {

//Because we are attacking , the

//opposite players board is the displayed

int enemyPlayer;

if (thisPlayer == 1)

enemyPlayer = 2;

if (thisPlayer == 2)

enemyPlayer = 1;

system("cls");

DrawBoard(enemyPlayer);

//Get attack coords from this player

bool crctIn = false;

int x, y;

while (crctIn == false)

{

crctIn = UserInputAttack(x, y, thisPlayer);

}

//Check board; if a ship is there, set as HIT.. otherwise MISS

if (player[enemyPlayer].grid[x][y] == isSHIP)

{

player[enemyPlayer].grid[x][y] = isHIT;

ptr->hitCount++; // keeps track of how many were hit

}

if (player[enemyPlayer].grid[x][y] == isWATER)

{

player[enemyPlayer].grid[x][y] = isMISS;

ptr->missCount++; // keeps track of how many were missed

}

//Check to see if the game is over

//If 0 is returned, nobody has won yet

int aWin = GameOverCheck(enemyPlayer);

if (aWin != 0) {

gameRunning = false;

break;

}

//Alternate between each player as we loop back around

thisPlayer = (thisPlayer == 1) ? 2 : 1;

}while (gameRunning);

system("cls");

cout << "\n\nCONGRATULATIONS!!! PLAYER " << thisPlayer << " HAS WON THE GAME!\n\n\n\n";

// output files to GameStats.txt file

// outputs usernames to a binary file

otptStat(ptr->missCount, ptr->hitCount, thisPlayer); // sending miss count and hit count as pointers.

system("pause");

return 0;

}

bool GameOverCheck(int enemyPLAYER)

{

bool winner = true;

//Loop through enemy board

for (int w = 0; w<15; w++)

{

for (int h = 0; h<10; h++)

{

//If any ships remain, game is NOT over

if (player[enemyPLAYER].grid[w][h] = isSHIP)

{

winner = false;

return winner;

}

}

}

//If we get here, somebody won, game over!

return winner;

}

bool UserInputAttack(int& x, int& y, int theplayer)

{

cout << "\nPLAYER " << theplayer << ", ENTER COORDINATES TO ATTACK: ";

bool goodInput = false;

cin >> x >> y;

if (x<0 || x >= 15)

return goodInput;

if (y<0 || y >= 10)

return goodInput;

goodInput = true;

return goodInput;

}

PLACESHIPS UserInputShipPlacement() //return values to a struct that initializes my Ship placement function

{

int d, x, y; // changing variables d x and y to general pointers

int \*ptrd = nullptr;

int \*ptrx = nullptr;

int \*ptry = nullptr;

ptrd = &d;

ptrx = &x;

ptry = &y;

PLACESHIPS tmp;

//Using this as a bad return

tmp.shipType.onGrid[0].X = -1;

//Get 3 integers from user

cin >> \*ptrd >> \*ptrx >> \*ptry;

while (\*ptrd > 1 || \*ptrd < 0) //checks for valid input of the ship in the HORZ or VERT direction

{

cout << "Please enter a correct value for the HORIZONTAL or VERTICAL location of the ships: " << endl;

cin >> \*ptrd;

}

while (\*ptrx > 14 || \*ptrx < 0) // checks for valid input of the ship in the x direction

{

cout << "Please enter a correct value for the x distance: " << endl;

cin >> \*ptrx;

}

while (\*ptry > 9 || \*ptry < 0) // checks for valid input of the ship placement in the y direction

{

cout << "Please enter a correct value for the y distance: " << endl;

cin >> y;

}

if (\*ptrd != 0 && \*ptrd != 1)

return tmp;

if (\*ptrx<0 || \*ptrx >= 15)

return tmp;

if (\*ptry<0 || \*ptry >= 10)

return tmp;

//Good data

tmp.direction = (DIRECTION)\*ptrd;

tmp.shipType.onGrid[0].X = \*ptrx;

tmp.shipType.onGrid[0].Y = \*ptry;

return tmp;

}

void LoadShips()

{

// Sets the default data for the ships

// we plan to include in the game

ship[0].name = "Cruiser"; ship[0].length = 2;

ship[1].name = "Frigate"; ship[1].length = 3;

ship[2].name = "Submarine"; ship[2].length = 3;

ship[3].name = "Escort"; ship[3].length = 4;

ship[4].name = "Battleship"; ship[4].length = 5;

}

void ResetBoard()

{

//Loop through each player

for (int plyr = 1; plyr<3; plyr++)

{

//For each grid point, set contents to 'water'

for (int w = 0; w<15; w++)

{

for (int h = 0; h<10; h++)

{

player[plyr].grid[w][h] = isWATER;

}

} //Loop back to next player

}

}

void DrawBoard(int thisPlayer)

{

//Draws the board for a player (thisPlayer)

cout << "PLAYER " << thisPlayer << "'s GAME BOARD\n";

cout << "----------------------\n";

//Loop through top row (board\_width) and number columns

cout << " ";

for (int w = 0; w<15; w++) {

if (w < 10)

//Numbers only 1 character long, add two spaces after

cout << w << " ";

else if (w >= 10)

//Numbers 2 characters long, add only 1 space after

cout << w << " ";

}

cout << "\n";

//Loop through each grid point and display to console

for (int h = 0; h<10; h++)

{

for (int w = 0; w<15; w++)

{

//If this is the FIRST (left) grid point, number the grid first

if (w == 0)

cout << h << " ";

//If h was 1 character long, add an extra space to keep numbers lined up

if (w<10 && w == 0)

cout << " ";

//Display contents of this grid (if game isn't running yet, we are placing ships

//so display the ships

if (gameRunning == false)

cout << player[thisPlayer].grid[w][h] << " ";

//Don't show ships, BUT show damage if it's hit

if (gameRunning == true && player[thisPlayer].grid[w][h] != isSHIP)

{

cout << player[thisPlayer].grid[w][h] << " ";

}

else if (gameRunning == true && player[thisPlayer].grid[w][h] == isSHIP)

{

cout << isWATER << " ";

}

//If we have reached the border.. line feed

if (w == 15 - 1)

cout << "\n";

}

}

}

void otptStat(int miss, int hits, int winner) // writes output files to a text file & dat binary file

{

// keep track of how many times the shots are fired, missed, names of the two players.

ofstream outFile;

fstream file;

string first, last;

string name1, name2;

cout << "Player 1 please enter your name: ";

getline(cin, name1);

cout << "Player 2 please enter your name: ";

getline(cin, name2);

outFile.open("GameStats.txt", ios::out); // file for txt

file.open("GameStats.dat", ios::out|ios::binary); // creates a binary file that mimics the .txt file

file.write(reinterpret\_cast<char \*>(&name1), sizeof(name1));

file.write(reinterpret\_cast<char \*>(&name2), sizeof(name2));

//file.write(reinterpret\_cast<char \*>(&names), sizeof(names)); // can be used to keep track of the people who use the game

// for say maybe in an arcade setting

outFile << "============================================" << endl;

// alphabetizing the string objects using operators

if (name1 > name2)

{

last = name1;

first = name2;

outFile << " " << first << " and " << last << " " << endl;

}

else if (name2 > name1)

{

first = name1;

last = name2;

outFile << " " << first << " and " << last << " " << endl;

}

outFile << " Thanks for playing Battleship! " << endl;

outFile << "============================================" << endl;

outFile << "Player 1: " << name1 << endl;

outFile << "Player 2: " << name2 << endl;

outFile << "The total amount of shots missed was: " << miss << endl;

outFile << "The total amount of shots hit was: " << hits << endl;

outFile << "The winner was Player " << winner +1 << " !!" <<endl;

outFile.close();

system("cls");

}

void dispIntro() // reads input from a txt file

{

string getCntnt;

ifstream openFile;

openFile.open("WelcomeMessage.txt");

while (openFile >> getCntnt)

{

cout << getCntnt << " ";

}

cout << endl << endl;

openFile.close();

}