Project 2

CIS-17A (45434)

**Battleship**

Due Date:

July 27, 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.

For example, 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 player attempts to guess various locations and eventually beat the game. There are a total of 5 ships that the player must destroy. Each ship has its own name, hit points/health values, amongst various other properties.   
The game will display when a certain type of ship has been sunk. Sink all of them to win!

**Summary:**

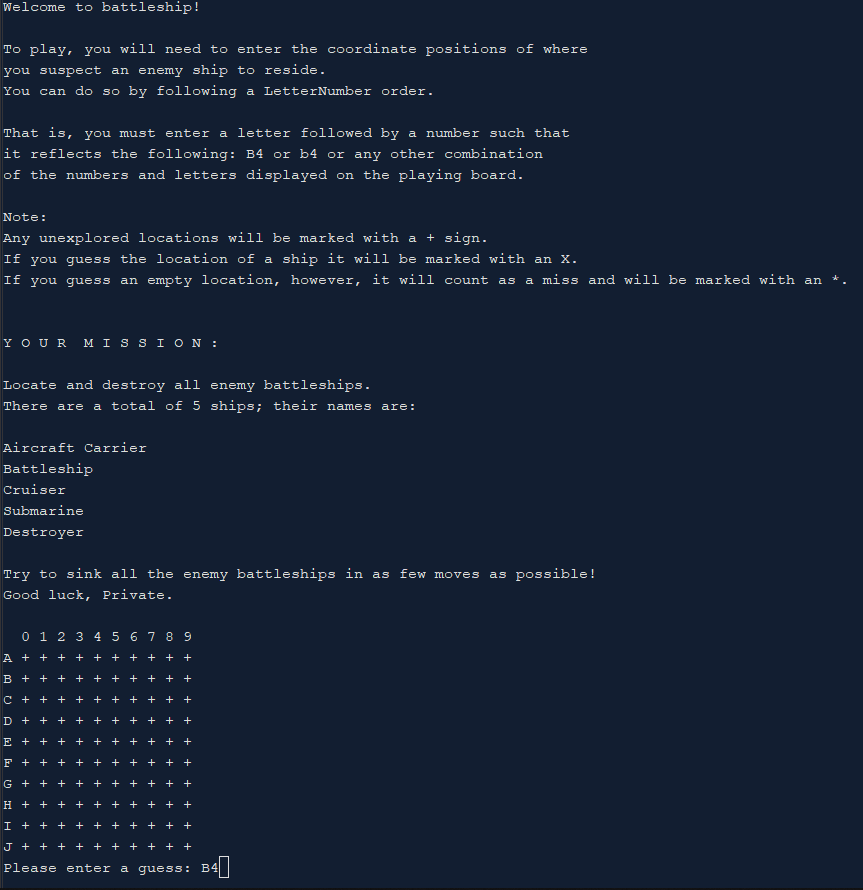
As I made this program I wanted to completely upgrade and convert my first project into something much more functional and advanced. I was able to do so by implementing loads more features into my project as well as incorporating newly learned material such as Classes, Inheritance, Abstract Classes, Operator overloading, amongst many other concepts covered throughout the semester.

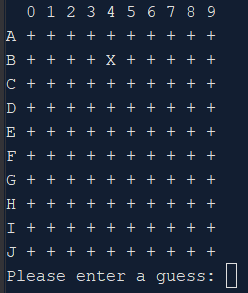
**Project Size:** about 1400 lines

**Description:**

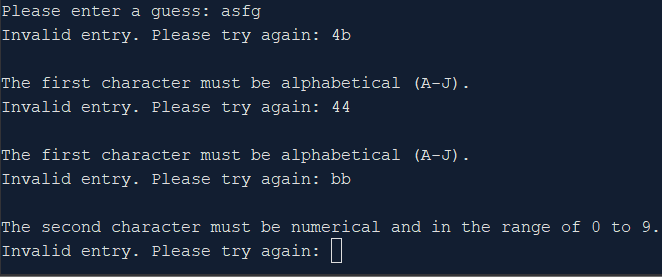
**Sample Input/Output:**

Input: B4 (enter)

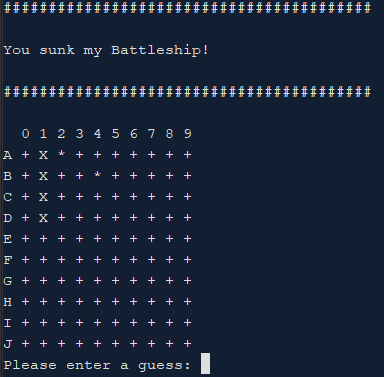


Output updates accordingly:  
 

Invalid input yields error prompts to the user:



Destroying a ship outputs which ship the player sank:



**Variables:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Type** | **Variable Name** | **Description** | **Location** |
| class Game | board | Holds game board values and properties | init(), shipGen(), run(), print(), update(), bordGen() |
|  | letters | Holds string of letters for output on the Y axis | print() |
|  | hp | Holds total hit points possible | play() |
| class  Ship | name | Holds the ship name/type | update() |
|  | length | Holds the length/size of the ship | update(),shipGen() |
|  | orient | Holds the ship orientation | shipGen() |
|  | hp | Holds the amount of hit points the ship can take | update(),shipGen() |
|  | coords | Holds X & Y Spawnpoint coordinates for the ship | update(),shipGen() |
|  | place | Holds the placement coordinates in a one dimensional array for easy access | update(),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 |  |  |  |
|  | 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:**

Most of the concepts included in this project involved the topics covered in class nearing the end of the semester. Using the Project2CheckOffSheet file provided, I tried to include as many of the topics in this new version of Battleship. Unfortunately, there were some topics I was unable to include due to the fact that I was either unsure of how to incorporate them into the project; or, I simply wasn’t familiar with them. That being said, I think making this project really helped in allowing me to employ all the material I’ve learned from this class, this semester; and, I look forward to the challenges that lay before me as I make my way through Computer Science.

**UML:**

*See included html file within Project\_2 Folder*

**References:**

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

**CheckOffSheet:**

**Cross Reference for Project 2**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Chapter** | **Section** | **Topic** | **Where Line #''s** | **Pts** | **Notes** |
| 13 |  | Classes |  |  |  |
|  | 1 to 3 | Instance of a Class | main() line 24 | 4 |  |
|  | 4 | Private Data Members | Game.h lines 56-59 | 4 | Never Public |
|  | 5 | Specification vs. Implementation | Game.h vs Game.cpp | 4 | .h vs. .cpp files Always split |
|  | 6 | Inline | Battleship.h line 20 | 4 |  |
|  | 7, 8, 10 | Constructors | Battleship.h lines 4-16 | 4 | Overloading |
|  | 9 | Destructors | Game.cpp line 23 | 4 |  |
|  | 12 | Arrays of Objects | n/a | 4 |  |
|  | 16 | UML | in attached html file | 4 |  |
|  |  |  |  |  |  |
| 14 |  | More about Classes |  |  |  |
|  | 1 | Static | Game.h line 56 | 5 |  |
|  | 2 | Friends | Battleship.h line 14 | 2 |  |
|  | 4 | Copy Constructors | Battleship.cpp lines 18-24 | 5 |  |
|  | 5 | Operator Overloading | Battleship.h, Cruiser.h, Destroyer.h line 20 | 8 | Overload 3 operators |
|  | 7 | Aggregation | Game.h lines 62-66 | 6 |  |
|  |  |  |  |  |  |
| 15 |  | Inheritance |  |  |  |
|  | 1 | Protected members | Ship.h lines 22-33 | 6 |  |
|  | 2 to 5 | Base Class to Derived | Battleship.h line 12 | 6 |  |
|  | 6 | Polymorphic associations | n/a | 6 |  |
|  | 7 | Abstract Classes | Ship.h lines 7-34 | 6 |  |
|  |  |  |  |  |  |
| 16 |  | Advanced Classes |  |  |  |
|  | 1 | Exceptions | n/a | 6 |  |
|  | 2 to 4 | Templates | n/a | 6 |  |
|  | 5 | STL | n/a | 6 |  |
|  |  |  |  |  |  |
|  |  | Sum |  | 100 |  |

Page 1

**Program:**

/\*

\* File: main.cpp

\* Author: Jorge Rubi

\* Purpose: Project 2 - Battleship Revamped x 2

\*/

//System Libraries Here

#include <iostream>

#include "Game.h"

using namespace std;

//User Libraries Here

//Global Constants Only, No Global Variables

//Like PI, e, Gravity, or conversions

//Function Prototypes Here

//Program Execution Begins Here

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

//Seed random number

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

Game game;

game.play();

//Exit

return 0;

}

/\*

\* File: Game.cpp

\* Author: Cokes PC

\*

\* Created on July 26, 2018, 7:55 PM

\*/

#include <iostream>

#include <cctype>

#include <fstream>

using namespace std;

#include "Game.h"

string Game::letters = "ABCDEFGHIJ";

Game::Game() {

}

Game::Game(const Game& orig) {

}

Game::~Game() {

}

void Game::play(){

//Store the number of moves made in order to win

int moves = 0;

//Store the user's guess to pass it to the update function

char \*guess;

intro();

init(); //Generate the board and ships

do{

print(); //Print the playing board

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

update(guess);

moves++;

}while(hp > 0);

//Output the board in it's final (winning) state.

print();

//Record the player's score and then end the game

endGame(moves);

}

void Game::intro(){

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

cout << "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\n"

"Y O U R M I S S I O N : \n\n"

"Locate and destroy all enemy battleships. \n"

"There are a total of 5 ships; their names are: \n\n"

"Aircraft Carrier\n"

"Battleship\n"

"Cruiser\n"

"Submarine\n"

"Destroyer\n\n"

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

"Good luck, Private." << endl << endl;

}

void Game::init(){

//Generate the playing board once

bordGen();

//Generate enemy battleship positions on the playing board once

shipGen();

}

void Game::bordGen(){

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

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

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

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

}

}

}

void Game::shipGen(){

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

// ///\*\*\*Extreme redundancy function\*\*\*/// //

// Due to abstract data type inheritance //

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

//Boolean to determine generation success

bool gen = true;

//Temporary variables for ship placement

int sizeTemp, xVal, yVal, count, length, \*coords;

//Generate Aircraft Carrier ship1

do{

//Get the spawn coords with every loop in case they are modified or reset

ship1.setCoords();

coords = ship1.getCoords();

length = ship1.getLength();

//Reset temporary variables for each ship loop

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

//Attempt horizontal generation

if(ship1.getOrient() == 0 ){

//Edge collision detection through sufficient space checking

sizeTemp = 10 - coords[1] - length;

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

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

coords[1] += sizeTemp;

}

//After potentially updating the coordinate values

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

yVal = coords[0];

xVal = coords[1];

//Begin ship placement

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

//If an attempted overlap occurs, undo everything

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

//Hold the position generation was at in order to undo

count = x;

//Set gen to false and break to attempt gen again

gen = false;

break;

}

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

//Place a ship onto the board if there's space to do so

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

//Store placement coordinate

ship1.place[x + x] = yVal;

ship1.place[x + x + 1] = xVal + x;

//Set gen to true to avoid undo loop

gen = true;

}

}

//If generation was unsuccessful, undo the attempted generation actions

if(!gen){

for(int x = coords[1]; x < coords[1] + count; x++){

board[yVal][x] = '+';

//Undo storing of placement coordinate

ship1.place[x + x] = 0;

ship1.place[x + x + 1] = 0;

}

}

}

//Attempt vertical generation

else if(ship1.getOrient() == 1){

//Edge collision detection through sufficient space checking

sizeTemp = 10 - coords[0] - length;

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

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

coords[0] += sizeTemp;

}

//After potentially updating the coordinate values

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

yVal = coords[0];

xVal = coords[1];

//Begin ship placement

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

//If an attempted overlap occurs, undo everything

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

//Hold the position generation was at in order to undo

count = y;

//Set gen to false and break to attempt gen again

gen = false;

break;

}

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

//Place a ship onto the board if there's space to do so

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

//Store placement coordinate

ship1.place[y + y] = yVal + y;

ship1.place[y + y + 1] = xVal;

//Set gen to true to avoid undo loop

gen = true;

}

}

//If generation was unsuccessful, undo the attempted generation actions

if(!gen){

for(int y = coords[0]; y < coords[0] + count; y++){

board[y][xVal] = '+';

//Undo storing of placement coordinate

ship1.place[y + y] = 0;

ship1.place[y + y + 1] = 0;

}

}

}

else{

cout << "An error occurred" << endl;

exit(EXIT\_FAILURE);

}

}while(!gen);

//Debugging

// for(int i = 0; i < length; i++){

// cout << ship1.place[i + i] << ship1.place[i + i + 1] << endl;

// }

//Generate Battleship ship2

do{

//Get the spawn coords with every loop in case they are modified or reset

ship2.setCoords();

coords = ship2.getCoords();

length = ship2.getLength();

//Reset temporary variables for each ship loop

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

//Attempt horizontal generation

if(ship2.getOrient() == 0 ){

//Edge collision detection through sufficient space checking

sizeTemp = 10 - coords[1] - length;

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

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

coords[1] += sizeTemp;

}

//After potentially updating the coordinate values

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

yVal = coords[0];

xVal = coords[1];

//Begin ship placement

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

//If an attempted overlap occurs, undo everything

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

//Hold the position generation was at in order to undo

count = x;

//Set gen to false and break to attempt gen again

gen = false;

break;

}

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

//Place a ship onto the board if there's space to do so

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

//Store placement coordinate

ship2.place[x + x] = yVal;

ship2.place[x + x + 1] = xVal + x;

//Set gen to true to avoid undo loop

gen = true;

}

}

//If generation was unsuccessful, undo the attempted generation actions

if(!gen){

for(int x = coords[1]; x < coords[1] + count; x++){

board[yVal][x] = '+';

//Undo storing of placement coordinate

ship2.place[x + x] = 0;

ship2.place[x + x + 1] = 0;

}

}

}

//Attempt vertical generation

else if(ship2.getOrient() == 1){

//Edge collision detection through sufficient space checking

sizeTemp = 10 - coords[0] - length;

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

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

coords[0] += sizeTemp;

}

//After potentially updating the coordinate values

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

yVal = coords[0];

xVal = coords[1];

//Begin ship placement

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

//If an attempted overlap occurs, undo everything

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

//Hold the position generation was at in order to undo

count = y;

//Set gen to false and break to attempt gen again

gen = false;

break;

}

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

//Place a ship onto the board if there's space to do so

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

//Store placement coordinate

ship2.place[y + y] = yVal + y;

ship2.place[y + y + 1] = xVal;

//Set gen to true to avoid undo loop

gen = true;

}

}

//If generation was unsuccessful, undo the attempted generation actions

if(!gen){

for(int y = coords[0]; y < coords[0] + count; y++){

board[y][xVal] = '+';

//Undo storing of placement coordinate

ship2.place[y + y] = 0;

ship2.place[y + y + 1] = 0;

}

}

}

else{

cout << "An error occurred" << endl;

exit(EXIT\_FAILURE);

}

}while(!gen);

//Debugging

// for(int i = 0; i < length; i++){

// cout << ship2.place[i + i] << ship2.place[i + i + 1] << endl;

// }

//Generate Submarine ship3

do{

//Get the spawn coords with every loop in case they are modified or reset

ship3.setCoords();

coords = ship3.getCoords();

length = ship3.getLength();

//Reset temporary variables for each ship loop

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

//Attempt horizontal generation

if(ship3.getOrient() == 0 ){

//Edge collision detection through sufficient space checking

sizeTemp = 10 - coords[1] - length;

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

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

coords[1] += sizeTemp;

}

//After potentially updating the coordinate values

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

yVal = coords[0];

xVal = coords[1];

//Begin ship placement

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

//If an attempted overlap occurs, undo everything

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

//Hold the position generation was at in order to undo

count = x;

//Set gen to false and break to attempt gen again

gen = false;

break;

}

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

//Place a ship onto the board if there's space to do so

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

//Store placement coordinate

ship3.place[x + x] = yVal;

ship3.place[x + x + 1] = xVal + x;

//Set gen to true to avoid undo loop

gen = true;

}

}

//If generation was unsuccessful, undo the attempted generation actions

if(!gen){

for(int x = coords[1]; x < coords[1] + count; x++){

board[yVal][x] = '+';

//Undo storing of placement coordinate

ship3.place[x + x] = 0;

ship3.place[x + x + 1] = 0;

}

}

}

//Attempt vertical generation

else if(ship3.getOrient() == 1){

//Edge collision detection through sufficient space checking

sizeTemp = 10 - coords[0] - length;

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

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

coords[0] += sizeTemp;

}

//After potentially updating the coordinate values

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

yVal = coords[0];

xVal = coords[1];

//Begin ship placement

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

//If an attempted overlap occurs, undo everything

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

//Hold the position generation was at in order to undo

count = y;

//Set gen to false and break to attempt gen again

gen = false;

break;

}

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

//Place a ship onto the board if there's space to do so

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

//Store placement coordinate

ship3.place[y + y] = yVal + y;

ship3.place[y + y + 1] = xVal;

//Set gen to true to avoid undo loop

gen = true;

}

}

//If generation was unsuccessful, undo the attempted generation actions

if(!gen){

for(int y = coords[0]; y < coords[0] + count; y++){

board[y][xVal] = '+';

//Undo storing of placement coordinate

ship3.place[y + y] = 0;

ship3.place[y + y + 1] = 0;

}

}

}

else{

cout << "An error occurred" << endl;

exit(EXIT\_FAILURE);

}

}while(!gen);

//Debugging

// for(int i = 0; i < length; i++){

// cout << ship3.place[i + i] << ship3.place[i + i + 1] << endl;

// }

//Generate Cruiser ship4

do{

//Get the spawn coords with every loop in case they are modified or reset

ship4.setCoords();

coords = ship4.getCoords();

length = ship4.getLength();

//Reset temporary variables for each ship loop

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

//Attempt horizontal generation

if(ship4.getOrient() == 0 ){

//Edge collision detection through sufficient space checking

sizeTemp = 10 - coords[1] - length;

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

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

coords[1] += sizeTemp;

}

//After potentially updating the coordinate values

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

yVal = coords[0];

xVal = coords[1];

//Begin ship placement

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

//If an attempted overlap occurs, undo everything

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

//Hold the position generation was at in order to undo

count = x;

//Set gen to false and break to attempt gen again

gen = false;

break;

}

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

//Place a ship onto the board if there's space to do so

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

//Store placement coordinate

ship4.place[x + x] = yVal;

ship4.place[x + x + 1] = xVal + x;

//Set gen to true to avoid undo loop

gen = true;

}

}

//If generation was unsuccessful, undo the attempted generation actions

if(!gen){

for(int x = coords[1]; x < coords[1] + count; x++){

board[yVal][x] = '+';

//Undo storing of placement coordinate

ship4.place[x + x] = 0;

ship4.place[x + x + 1] = 0;

}

}

}

//Attempt vertical generation

else if(ship4.getOrient() == 1){

//Edge collision detection through sufficient space checking

sizeTemp = 10 - coords[0] - length;

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

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

coords[0] += sizeTemp;

}

//After potentially updating the coordinate values

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

yVal = coords[0];

xVal = coords[1];

//Begin ship placement

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

//If an attempted overlap occurs, undo everything

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

//Hold the position generation was at in order to undo

count = y;

//Set gen to false and break to attempt gen again

gen = false;

break;

}

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

//Place a ship onto the board if there's space to do so

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

//Store placement coordinate

ship4.place[y + y] = yVal + y;

ship4.place[y + y + 1] = xVal;

//Set gen to true to avoid undo loop

gen = true;

}

}

//If generation was unsuccessful, undo the attempted generation actions

if(!gen){

for(int y = coords[0]; y < coords[0] + count; y++){

board[y][xVal] = '+';

//Undo storing of placement coordinate

ship4.place[y + y] = 0;

ship4.place[y + y + 1] = 0;

}

}

}

else{

cout << "An error occurred" << endl;

exit(EXIT\_FAILURE);

}

}while(!gen);

//Debugging

// for(int i = 0; i < length; i++){

// cout << ship4.place[i + i] << ship4.place[i + i + 1] << endl;

// }

//Generate Destroyer ship5

do{

//Get the spawn coords with every loop in case they are modified or reset

ship5.setCoords();

coords = ship5.getCoords();

length = ship5.getLength();

//Reset temporary variables for each ship loop

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

//Attempt horizontal generation

if(ship5.getOrient() == 0 ){

//Edge collision detection through sufficient space checking

sizeTemp = 10 - coords[1] - length;

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

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

coords[1] += sizeTemp;

}

//After potentially updating the coordinate values

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

yVal = coords[0];

xVal = coords[1];

//Begin ship placement

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

//If an attempted overlap occurs, undo everything

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

//Hold the position generation was at in order to undo

count = x;

//Set gen to false and break to attempt gen again

gen = false;

break;

}

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

//Place a ship onto the board if there's space to do so

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

//Store placement coordinate

ship5.place[x + x] = yVal;

ship5.place[x + x + 1] = xVal + x;

//Set gen to true to avoid undo loop

gen = true;

}

}

//If generation was unsuccessful, undo the attempted generation actions

if(!gen){

for(int x = coords[1]; x < coords[1] + count; x++){

board[yVal][x] = '+';

//Undo storing of placement coordinate

ship5.place[x + x] = 0;

ship5.place[x + x + 1] = 0;

}

}

}

//Attempt vertical generation

else if(ship5.getOrient() == 1){

//Edge collision detection through sufficient space checking

sizeTemp = 10 - coords[0] - length;

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

//by as many spaces as necessary and proceed

if(sizeTemp < 0){

coords[0] += sizeTemp;

}

//After potentially updating the coordinate values

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

yVal = coords[0];

xVal = coords[1];

//Begin ship placement

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

//If an attempted overlap occurs, undo everything

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

//Hold the position generation was at in order to undo

count = y;

//Set gen to false and break to attempt gen again

gen = false;

break;

}

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

//Place a ship onto the board if there's space to do so

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

//Store placement coordinate

ship5.place[y + y] = yVal + y;

ship5.place[y + y + 1] = xVal;

//Set gen to true to avoid undo loop

gen = true;

}

}

//If generation was unsuccessful, undo the attempted generation actions

if(!gen){

for(int y = coords[0]; y < coords[0] + count; y++){

board[y][xVal] = '+';

//Undo storing of placement coordinate

ship5.place[y + y] = 0;

ship5.place[y + y + 1] = 0;

}

}

}

else{

cout << "An error occurred" << endl;

exit(EXIT\_FAILURE);

}

}while(!gen);

//Debugging

// for(int i = 0; i < length; i++){

// cout << ship5.place[i + i] << ship5.place[i + i + 1] << endl;

// }

}

void Game::print(){

//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 = 10;

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(board[y][x] == '1'){

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

}

else{

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

}

}

cout << endl;

}

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

}

char \*Game::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;

}

void Game::update(char\* guess){

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

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

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

//If hit

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

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

hp--;

//Check all ship coordinates to determine which ship type was hit

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

for(int i = 0; i < (ship1.length \* 2); i++){

if(ship1.place[i + i] == y && ship1.place[i + i + 1] == x){

ship1--;

}

}

for(int i = 0; i < (ship2.length \* 2); i++){

if(ship2.place[i + i] == y && ship2.place[i + i + 1] == x){

ship2--;

}

}

for(int i = 0; i < (ship3.length \* 2); i++){

if(ship3.place[i + i] == y && ship3.place[i + i + 1] == x){

ship3--;

}

}

for(int i = 0; i < (ship4.length \* 2); i++){

if(ship4.place[i + i] == y && ship4.place[i + i + 1] == x){

ship4--;

}

}

for(int i = 0; i < (ship5.length \* 2); i++){

if(ship5.place[i + i] == y && ship5.place[i + i + 1] == x){

ship5--;

}

}

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

}

//If miss

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

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

}

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

//Output when a ship has been sunk!

if(ship1.hp == 0){

//Turn value negative to avoid outputting with every update

ship1--;

cout << "#########################################" << endl << endl;

cout << "You sunk my " << ship1.getName() << "!" << endl << endl;

cout << "#########################################" << endl << endl;

}

if(ship2.hp == 0){

//Turn value negative to avoid outputting with every update

ship2--;

cout << "#########################################" << endl << endl;

cout << "You sunk my " << ship2.getName() << "!" << endl << endl;

cout << "#########################################" << endl << endl;

}

if(ship3.hp == 0){

//Turn value negative to avoid outputting with every update

ship3--;

cout << "#########################################" << endl << endl;

cout << "You sunk my " << ship3.getName() << "!" << endl << endl;

cout << "#########################################" << endl << endl;

}

if(ship4.hp == 0){

//Turn value negative to avoid outputting with every update

ship4--;

cout << "#########################################" << endl << endl;

cout << "You sunk my " << ship4.getName() << "!" << endl << endl;

cout << "#########################################" << endl << endl;

}

if(ship5.hp == 0){

//Turn value negative to avoid outputting with every update

ship5--;

cout << "#########################################" << endl << endl;

cout << "You sunk my " << ship5.getName() << "!" << endl << endl;

cout << "#########################################" << endl << endl;

}

}

void Game::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! "<< '\r' << '\n' << endl;

file.close();

}

#include "AircraftCarrier.h"

using namespace std;

//Constructor

AircraftCarrier::AircraftCarrier() {

name = "Aircraft Carrier";

length = 5;

orient = rand() % 2;

hp = 5;

coords[0] = 0;

coords[1] = 0;

//Initialize placement coordinate array

place = new int[length \* 2];

for(int i = 0; i < (length\*2); i++){

place[i] = 0;

}

}

//Copy constructor

AircraftCarrier::AircraftCarrier(const AircraftCarrier& orig) {

name = orig.name;

length = orig.length;

orient = orig.orient;

hp = orig.hp;

\*coords = \*orig.coords;

}

//Destructor

AircraftCarrier::~AircraftCarrier() {

}

///\*\*\*Getters\*\*\*///

void AircraftCarrier::setCoords(){

coords[0] = rand() % 10;

coords[1] = rand() % 10;

}

string AircraftCarrier::getName(){

return name;

}

int AircraftCarrier::getLength(){

return length;

}

int AircraftCarrier::getOrient(){

return orient;

}

int AircraftCarrier::getHp(){

return hp;

}

int \*AircraftCarrier::getCoords(){

return coords;

}

#include "Battleship.h"

using namespace std;

Battleship::Battleship() {

name = "Battleship";

length = 4;

orient = rand() % 2;

hp = 4;

coords[0] = 0;

coords[1] = 0;

//Initialize placement coordinate array

place = new int[length \* 2];

for(int i = 0; i < (length\*2); i++){

place[i] = 0;

}

}

Battleship::Battleship(const Battleship& orig) {

name = orig.name;

length = orig.length;

orient = orig.orient;

hp = orig.hp;

\*coords = \*orig.coords;

}

Battleship::~Battleship() {

}

///\*\*\*Getters\*\*\*///

void Battleship::setCoords(){

coords[0] = rand() % 10;

coords[1] = rand() % 10;

}

string Battleship::getName(){

return name;

}

int Battleship::getLength(){

return length;

}

int Battleship::getOrient(){

return orient;

}

int Battleship::getHp(){

return hp;

}

int \*Battleship::getCoords(){

return coords;

}

#include "Cruiser.h"

using namespace std;

Cruiser::Cruiser() {

name = "Cruiser";

length = 3;

orient = rand() % 2;

hp = 3;

coords[0] = 0;

coords[1] = 0;

//Initialize placement coordinate array

place = new int[length \* 2];

for(int i = 0; i < (length\*2); i++){

place[i] = 0;

}

}

Cruiser::Cruiser(const Cruiser& orig) {

name = orig.name;

length = orig.length;

orient = orig.orient;

hp = orig.hp;

\*coords = \*orig.coords;

}

Cruiser::~Cruiser() {

}

///\*\*\*Getters\*\*\*///

void Cruiser::setCoords(){

coords[0] = rand() % 10;

coords[1] = rand() % 10;

}

string Cruiser::getName(){

return name;

}

int Cruiser::getLength(){

return length;

}

int Cruiser::getOrient(){

return orient;

}

int Cruiser::getHp(){

return hp;

}

int \*Cruiser::getCoords(){

return coords;

}

#include "Destroyer.h"

using namespace std;

Destroyer::Destroyer() {

name = "Destroyer";

length = 2;

orient = rand() % 2;

hp = 2;

coords[0] = 0;

coords[1] = 0;

//Initialize placement coordinate array

place = new int[length \* 2];

for(int i = 0; i < (length\*2); i++){

place[i] = 0;

}

}

Destroyer::Destroyer(const Destroyer& orig) {

name = orig.name;

length = orig.length;

orient = orig.orient;

hp = orig.hp;

\*coords = \*orig.coords;

}

Destroyer::~Destroyer() {

}

///\*\*\*Getters\*\*\*///

void Destroyer::setCoords(){

coords[0] = rand() % 10;

coords[1] = rand() % 10;

}

string Destroyer::getName(){

return name;

}

int Destroyer::getLength(){

return length;

}

int Destroyer::getOrient(){

return orient;

}

int Destroyer::getHp(){

return hp;

}

int \*Destroyer::getCoords(){

return coords;

}

#include "Submarine.h"

using namespace std;

Submarine::Submarine() {

name = "Submarine";

length = 3;

orient = rand() % 2;

hp = 3;

coords[0] = 0;

coords[1] = 0;

//Initialize placement coordinate array

place = new int[length \* 2];

for(int i = 0; i < (length\*2); i++){

place[i] = 0;

}

}

Submarine::Submarine(const Submarine& orig) {

name = orig.name;

length = orig.length;

orient = orig.orient;

hp = orig.hp;

\*coords = \*orig.coords;

}

Submarine::~Submarine() {

}

///\*\*\*Getters\*\*\*///

void Submarine::setCoords(){

coords[0] = rand() % 10;

coords[1] = rand() % 10;

}

string Submarine::getName(){

return name;

}

int Submarine::getLength(){

return length;

}

int Submarine::getOrient(){

return orient;

}

int Submarine::getHp(){

return hp;

}

int \*Submarine::getCoords(){

return coords;

}

/\*

\* File: Ship.cpp

\* Author: Cokes PC

\*

\* Created on July 26, 2018, 7:57 PM

\*/

#include "Ship.h"

Ship::Ship() {

}

Ship::Ship(const Ship &orig) {

}

Ship::~Ship() {

}

/\*

\* File: AircraftCarrier.h

\* Author: Cokes PC

\*

\* Created on July 26, 2018, 7:58 PM

\*/

#include "Ship.h"

#ifndef AIRCRAFTCARRIER\_H

#define AIRCRAFTCARRIER\_H

class AircraftCarrier : public Ship {

friend class Game;

public:

AircraftCarrier();

AircraftCarrier(const AircraftCarrier& orig);

virtual ~AircraftCarrier();

//Overloaded postfix -- operator

AircraftCarrier operator--(int){

hp--; //Updates the Hp value by subtracting one

return \*this;

}

//Setters for modifying values

void setCoords(); //Creates a new spawnpoint

//Getters for returning the data in the member

string getName();

int getLength();

int getOrient();

int getHp();

int \*getCoords();

private:

};

#endif /\* AIRCRAFTCARRIER\_H \*/

/\*

\* File: Battleship.h

\* Author: Cokes PC

\*

\* Created on July 26, 2018, 7:58 PM

\*/

#include "Ship.h"

#ifndef BATTLESHIP\_H

#define BATTLESHIP\_H

class Battleship : public Ship {

public:

friend class Game;

Battleship();

Battleship(const Battleship& orig);

virtual ~Battleship();

//Overloaded postfix -- operator

Battleship operator--(int){

hp--; //Updates the Hp value by subtracting one

return \*this;

}

//Setters for modifying values

virtual void setCoords(); //Creates a new spawnpoint

//Getters for returning the data in the member

string getName();

int getLength();

int getOrient();

int getHp();

virtual int \*getCoords();

private:

};

#endif /\* BATTLESHIP\_H \*/

/\*

\* File: Cruiser.h

\* Author: Cokes PC

\*

\* Created on July 26, 2018, 7:59 PM

\*/

#include "Ship.h"

#ifndef CRUISER\_H

#define CRUISER\_H

class Cruiser : public Ship {

friend class Game;

public:

Cruiser();

Cruiser(const Cruiser& orig);

virtual ~Cruiser();

//Overloaded postfix -- operator

Cruiser operator--(int){

hp--; //Updates the Hp value by subtracting one

return \*this;

}

//Setters for modifying values

void setCoords(); //Creates a new spawnpoint

//Getters for returning the data in the member

string getName();

int getLength();

int getOrient();

int getHp();

int \*getCoords();

private:

};

#endif /\* CRUISER\_H \*/

/\*

\* File: Destroyer.h

\* Author: Cokes PC

\*

\* Created on July 26, 2018, 7:59 PM

\*/

#include "Ship.h"

#ifndef DESTROYER\_H

#define DESTROYER\_H

class Destroyer : public Ship {

friend class Game;

public:

Destroyer();

Destroyer(const Destroyer& orig);

virtual ~Destroyer();

//Overloaded postfix -- operator

Destroyer operator--(int){

hp--; //Updates the Hp value by subtracting one

return \*this;

}

//Setters for modifying values

void setCoords(); //Creates a new spawnpoint

//Getters for returning the data in the member

string getName();

int getLength();

int getOrient();

int getHp();

int \*getCoords();

private:

};

#endif /\* DESTROYER\_H \*/

/\*

\* File: Submarine.h

\* Author: Cokes PC

\*

\* Created on July 26, 2018, 7:59 PM

\*/

#include "Ship.h"

#ifndef SUBMARINE\_H

#define SUBMARINE\_H

class Submarine : public Ship {

friend class Game;

public:

Submarine();

Submarine(const Submarine& orig);

virtual ~Submarine();

//Overloaded postfix -- operator

Submarine operator--(int){

hp--; //Updates the Hp value by subtracting one

return \*this;

}

//Setters for modifying values

void setCoords(); //Creates a new spawnpoint

//Getters for returning the data in the member

string getName();

int getLength();

int getOrient();

int getHp();

int \*getCoords();

private:

};

#endif /\* SUBMARINE\_H \*/

/\*

\* File: Game.h

\* Author: Cokes PC

\*

\* Created on July 26, 2018, 7:55 PM

\*/

#include <string>

using namespace std;

#include "Ship.h"

#include "AircraftCarrier.h"

#include "Battleship.h"

#include "Submarine.h"

#include "Cruiser.h"

#include "Destroyer.h"

#ifndef GAME\_H

#define GAME\_H

class Game {

friend class Ship;

public:

Game();

Game(const Game& orig);

virtual ~Game();

//Executes and maintains game processes

void play();

//Outputs a player introduction and rules

void intro();

//Calls for ship and board initialization functions

void init();

//Generates & Initializes the playing board

void bordGen();

//Generates complete ship on the playing board

//Warning: Absurdly redundant due to every ship having its own gen loop

void shipGen();

//Print the playing board in its current state

void print();

//Get user to input a guess

char \*input(string);

//Update the board throughout play

void update(char \*);

//Get user ID and store the number of moves it took for them to win the game

void endGame(int);

private:

//Store Board, Vars, and maybe player structs from proj 1 in here

static string letters;

//char \*\*board; or board[10][10].

char board[10][10];

int hp = 17;

//Ship members

AircraftCarrier ship1;

Battleship ship2;

Submarine ship3;

Cruiser ship4;

Destroyer ship5;

};

#endif /\* GAME\_H \*/

/\*

\* File: Ship.h

\* Author: Cokes PC

\*

\* Created on July 26, 2018, 7:57 PM

\*/

using namespace std;

#include <string>

#ifndef SHIP\_H

#define SHIP\_H

class Ship {

public:

Ship();

Ship(const Ship &orig);

virtual ~Ship();

private:

protected:

//Holds the ship name/type

string name;

//Holds the length/size of the ship

int length;

//Holds the ship orientation (0 is horizontal, 1 is vertical)

int orient;

//Holds the amount of hit points the ship can take

int hp;

//Holds X & Y Spawnpoint coordinates for the ship

int coords[2];

//Holds the placement coordinates in a one dimensional array for easy access

int \*place;

};

#endif /\* SHIP\_H \*/