Project 1

Title

ASCII BattleShip!
An ASCII Battleship Simulation

Course

CSC-17A

Section

42636

Due Date

April 17, 2017

Author

Scott Parker

1 Introduction

BattleShip is a classic board game that has multiple layers of complexity. This simulation features 5 ships of varying sizes: BattleShip which takes 5 hits to sink, Destroyer which takes 4 hits to sink, Submarine which takes 3 hits to sink and a PT Boat which takes 2 hits to sink for a total of 14 hits to sink the entire fleet.

2 Game Play and Rules

The game is played by first placing 5 ships into the ocean. The ocean is made up of a 2D array of 26 x 12 lines with each row and column representing an x,y coordinate pair.

After the first player places their ships then the second player (or computer if vs. computer) will place all available ships. Players then alternate guesses in an attempt to sink the other players fleet. The victory conditions occur when one player first scores 14 hits (sinks the entire enemy fleet).

The player is given the opportunity to save the game at the end of each round, after which the game goes back to the initial loading screen.

3 Development Summary

Lines of Code	**~850**
Comment Lines	**~750**
Blank Lines (White space)	**~50**
Total Lines of Source File	**931**

The project was completely coded from scratch. All of the code, headers files, functions, and features were 100% developed in-house without copying any external sourcecode. The project took approximately 70 hours to develop over a 8 day period (not including documentation).

3.1 Comments on Development

Working with 2D arrays in a structure was a challenging objective and eventually I found no other way than to use literals as the size definitions for the arrays. It would have been easier to use dynamic arrays and the use of literals could have been avoided, however due to time constraints rewriting the entire project to accommodate this was not feasible.

Saving and restoring games was also a challenge incorporating two distinct features. The first feature is the bones file which holds a list of valid save-game files. The second feature is the binary save file which uses the name of the save file. Duplicate file names overwrite existing files and the size of the bones file is dynamicall managed to avoid file-creep (excessive blank lines added at the end of the file).

3.1.1 Cheat Feature

Entering out-of-range data while being asked to enter a column in vs. computer mode will print out the enemy fleet layout. This was intentionally left in the program to aid in debugging and testing.

3.1.2 Display Elements

Even though this is a text-based ASCII program the games does feature an extensive color element. It uses ASCII color codes and is known to work correctly on Windows 10 and Macintosh systems.

3.1.3 Other New Features

I also added some other new features to the application including:

- Difficulty Levels – a novice user can start from “Easy” which is pretty much similar to the first version of the application with the occasional second bomb and make their way to “Advanced” which contains multiple words and letters.
- Accuracy – the player’s accuracy is now calculated and displayed at the end of the game.

4 Features

4.1 Pseudocode

```
/*
 * File:   main.cpp
 * Author: Scott Parker
 * Created on April 10, 2017, 2:00 PM
 * Purpose: Battleship game Project 1
 * Notes: In the ship array 0 indicates ocean (cyan O), 1 indicates a ship (black X),
2 indicates
 *     a ship that has been hit (red X) and -1 is a guess from the enemy player
 *
 *        In the the guess array 0 indicates ocean (cyan O), 1 indicates a miss
(black X) and 2
 *     indicates a hit (red X)
 */

//System Libraries
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <iomanip>
#include <cctype>
#include <fstream>
#include <string>
#include <cctype>
using namespace std;

//User Libraries
#include "colors.h"
```

```
#include "player.h"

//Constant to hold 2D array columns

//function to display the game board
//function to clear the gameboard
//Function to place a ship on the gameboard
//Function to see if ship is being placed in valid position
//Function to place all ships on the gameboard
//function to display the ship map only
//function to play the game
//function to reset all current game data
//Play a game with two human players
//Play a game versus the computer
//function to place all ships for computer player
//function for player to enter a guess
//function for computer to enter a guess
//Function to save the game
//Function to load a saved game
//Function to resume a saved game
//Function to resume a Human vs Human game
//Function to resume a Human vs Comp game
//Function to initiate player data and start game

//Executable code begins here! Always begins in Main
    //Set random seed

    //Declare Variables
    //menu variable for choices
    //constant for number of rows

    //Player1 structure
    //Player2 structure

    //Game menu
    do{

        //Output switch menu screen
        //enter 1 to resume saved game
        //enter 2 for a new game
        //0 or unlisted number to exit

        //Loop to validate input
           //Resetting flags
           //ignore contents of buffer
           //keep requesting input until valid

        //Switch to determine the Problem
           //start a new game
           //Function to load a saved game
           //default option - exit menu
   //show menu while choices all active

    //Exit stage right! - This is the 'return 0' call end of main
```

```
//*************************   newPlyr   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Function to initiate player data and start game
//** Inputs: none
//** Outputs: none
//*****************************************************************************
    //player structure
    //loop through rows
        //loop through columns
           //set guess array value to 0
           //set ship array value to 0
    //return structure

//*************************   resHum   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Function to resume a Human vs Human game
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //2 structures to track player hits and guesses
   //string to use for pausing screen
   //for loop to loop through rows
       //for loop to loop through columns
           //each non 0 occurance is a guess
               //increment total guesses
               //if guess value == 2 that guess was a hit
                   //increment number of hits
           //check if guess (value of array not zero)
               //increment total guesses if array value is a guess
               //check to see if guess was a hit
                   //increment hit counter if value was hit (array value 2)
   //begin do loop
       //output round number (guesses so far plus 1)
       //pause function to delay until <enter> pressed
       //for loop to output new lines to clear screen
       //call function to display player 1 game status
       //call function to enter guess and increment p1 hits if hit
       //increment total number of guesses so far
       //notify of pause
       //clear buffer
       //pause until enter
       //for loop to clear screen
       //notify of pause
       //pause until enter pressed
       //clear screen
       //call function to display game map for player 2
       //call function to guess for player 2 - increment hits if hit
       //increment player 2 number of guesses
       //notify of pause
       //clear buffer
       //pause until enter pressed
       //clear screen
```

```
       //output player 1 hits and guesses
       //output player 2 hits and guesses
       //notify of pause
       //clear buffer
       //pause until enter pressed
   //repeat loop until a player gets 14 hits or saves game
   //if game ends because of save then call save function
   //exit game message

//**************************   resComp   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Function to resume a Human vs Comp game
//** Inputs:  int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**          int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //string to be used for pausing
   //loop through rows
       //loop through columns
           //check if guess (value of array not zero)
               //increment total guesses if array value is a guess
               //check to see if guess was a hit
                   //increment hit counter if value was hit (array value 2)
           //check if guess (value of array not zero)
               //increment total guesses if array value is a guess
               //check to see if guess was a hit
                   //increment hit counter if value was hit (array value 2)
   //begin do loop
       //display player 1 game map
       //call guess function, increment player 1 hits if guess hits
       //increment number of player 1 guesses
       //call comp guess function increment player 2 hits if guess hits
       //increment player 2 guesses
       //output player 1 hits and guesses
       //output player 2 hits and guesses
       //notify of pause
       //clear buffer
       //pause until enter pressed or save char entered
   //loop until 14 hits scored by player or game saved
   //call game save function if necessary

//**************************   resGame   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Function to resume a saved game
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //variable used for choice options
   //Load a saved game
   //choose computer or human opponent
   //force variable to upper case
   //loop to validate input type and range
```

```
       //output data range options
       //input choice
       //force to upper case
   //if choice is to play computer game
       //call function to resume game vs computer
   //otherwise
       //call function to resume game vs human

//*************************   lodGame   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Function to load a saved game
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //string to hold file names from bones file. and hold names while counting lines
   //create filestream object
   //clear any game data in memory
   //open bones file in input mode
   //output error if file did not open correctly
       //loop through file line by line
           //get file line, increment to next line
           //output text from that line in file
   //close file
   //enter name of savefile to open
   //open binary file name provided
   //read in binary file data to populate game array data
       //read file to fill player 1 ship data
       //read file to fill player 1 guess data
       //read file to fill player 2 ship data
       //read file to fill player 2 guess data

//************************   savGame   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Function to save the game
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //hold file names from bones file and to hold names while counting lines
   //array of strings to hold file names
   //count variable and temp variable
   //create filestream object
   //open bones file in input mode
   //output error if file unable to open
   //otherwise
       //find the number of lines in the file
           //get file line, increment to next line
           //increment counter of the number of lines
   //close file
   //create dynamic array to hold bones file data
   //open bones file in input mode
   //output error if file fails to open
```

```
   //otherwise
       //loop through file line by line
           //get file line, increment to next array element
   //close file
   //loop through array line by line
       //display array element if not empty line
   //enter the name of new savefile
   //check for duplicate filename
        //add filename to end of array if no duplicate
        //overwrite if duplicate
    //loop through array
        //output contents of array
   //open bones file in output mode
   //output error message if file fails to open
   //otherwise
       //loop through array line by line
           //if line is not blank
               //output array element as line in file
   //close file
   //clean up memory and delete dynamic array
   //open binary file in output mode to save game data
   //write player 1 ship data to file
   //write player 1 guess data to file
   //write player 2 ship data to file
   //write player 2 guess data to file
   //close file


//***************************   entGues   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //function for player to enter a guess
//** Inputs: int pGuess[][COLS], int opShip[][COLS], int rows
//** Outputs: 0 if miss, 1 if hit
//*****************************************************************************
   //character variable to hold column data for guess
   //variable to hold converted character variable as int
   //variable to hold row data for guess
   //variable to set to return 1 if hit, 0 if miss
   //bool variable set to false until successful guess entered
   //loop until successful guess
       //enter column
       //loop to validate that input was valid and in range
           //cheat and show enemy ships
           //activate cheat if out of range data input
           //input data for column to guess
       //if upper case data entered
           //subtract 65 from value and set column number guess to new value
        //else data is lower case character
           //subtract 97 from value and set column number guess to new value
       //input row number for guess
       //loop to validate input range and type
           //input row guess until valid entry
       //if guess array value not zero this value already guessed so repeat guess
       //otherwise if enemy ship array value = 1 HIT
```

```
            //set value of player guess array to 2
            //set value of enemy ship array to 2
            //set hit value to 1
            //set boolean as true since successfully entered guess
        //otherwise the guess was a miss
            //set player guess array value to 1 to show miss
            //set enemey ship array value to -1 to show miss
            //set hit value to 0 (since coordinate was a miss)
            //set boolean to true since guess was successfully made without error
    //return hit value from function

//************************   comGues   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //function for computer to enter a guess
//** Inputs: int pGuess[][COLS], int opShip[][COLS], int rows, int gues
//** Outputs: 0 if miss, 1 if true
//*****************************************************************************
    //random column guess value
    //random row guess value
    //default hit value zero
    //boolean false until successful guess
        //if total guesses > 0 and guesses%11==0 then give comp a free hit
            //loop through rows while boolean false
                //loop through columns while boolean false
                    //if player has not guessed this coordinate already
                        //if this coordinate has an enemy ship that has not been hit
                            //set player guess value to 2 (hit)
                            //set opponent ship value to 2 (hit)
                            //set boolean to true
                            //set hit value to 1
        //otherwise if player has already guessed this position
            //get new column guess
            //get new row guess
        //otherwise if enemy has ship here
            //set player guess value to 2 (hit)
            //set enemy ship map value to 2 (hit)
            //set hit value to 1
            //set boolean to true (successfully entered guess)
        //otherwise position not guessed but no ship here
            //set player guess value to 1 (miss)
            //set enemy map value to -1 (to show enemy shot missed)
            //set hit value to zero
            //set boolean to true (guess attempted successfully)
    //return hit value

//*************************   pGameH   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Play a game with two human players
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
    //string used for pauses
```

```
   //begin do loop
       //clear cin flags
       //notify of pause
       //pause until enter key pressed
       //clear screen
       //call function to display player 1 game data
       //call function for player 1 to enter a guess - increment hits if needed
       //increment player 1 guess count
       //notify of pause
       //clear screen
       //notify of pause
       //pause until enter key pressed
       //display player 2 game data
       //call function for player 2 to enter a guess - increment hits if needed
       //increment player 2 guess count
       //pause until enter key pressed
       //clear screen
       //output player 1 hits, player 1 guesses
       //output player 2 hits, player 2 guesses
       //pause until enter key pressed
   //loop until player has 14 hits or game saved
   //save game if required

//***************************   pGameC   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Play a game versus the computer
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //string for pausing
   //begin do loop

       //call function for player 1 to enter a guess - increment hits if needed
       //increment player 1 guess count
       //call function for computer guess - increment hits if needed
       //increment player 2 guess count
       //call function to display player 1 game data
       //output player 1 hits and player 1 guesses
       //output player 2 hits and player 2 guesses
       //notify of pause
       //pause until enter pressed
   //loop until player has 14 hits or saves game
   //save game if necessary

//*****************************   pShipC   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //function to place all ships for computer player
//** Inputs: int ship[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //variable to hold choice
   //variable to hold column number
```

```
   //variable to hold row number
   //boolean false until ship successfully placed
   //loop down from max ship size to min
       //output PT Boat if size = 2
       //output SUBMARINE if size = 3
       //output DESTROYER if size = 4
       //output BATTLESHIP if size = 5
       //begin do loop
           //random column value
           //random row value
           //randomly decide vertical or horizontal placement
           //if ship placement vertical
               //check to see ship will fit or overlap
                   //boolean false if ships overlap or out of bounds
               //otherwise
                   //Loop through rows from start position to ship size
                       //set array value to 1 (ship present)
                       //boolean true, ship successfully placed
           //otherwise ship placement horizontal
               //check to see if ship will fit or overlap
                   //set boolean false if ships overlap or out of bounds
               //otherwise
                   //loop through columns
                       //set array value to 1 (ship present)
                       //boolean true, ship successfully placed
       //loop until ship placement successful


//************************   newGame   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //function to play a new game
//** Inputs:  int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**          int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //choice variable
   //pause input
   //Choose to play against a human or the computer
   //validate data
       //output available choices
       //input choice again until valid
   //if human opponent chosen
       //notify Player 1 to place ships
       //Place all ships on for player 1
       //pause until enter pressed
       //clear screen
       //notify Player 2 to place ships
       //pause until enter pressed
       //clear screen
       //call function to Play the game with two humans
   //otherwise
       //Place all ships on for player 1
       //place all ships for computer player
       //call function to Play the game vs. computer
```

```
//************************   clrData   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //function to reset all current game data
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //Reset game data for player 1
   //Reset game data for player 2

//************************   disShip   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //function to display a players ship data only
//** Inputs: int ship[][COLS], int rows
//** Outputs:  none
//*****************************************************************************
   //loop through rows
       //output row number
       //loop through columns
           //if no ship present
               //output cyan 'O'
           //if undamaged ship present
               //output black 'X'
           //if damaged ship present
               //output red inverse 'H'
           //if other player shot this location but missed
               //output magenta 'o'

//************************   pAllShp   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Function to place all ships on the gameboard
//** Inputs: int ships[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //loop down through size from Battleship to PT Boat
       //if size = 2 output PT Boat
       //if size = 3 output SUBMARINE
       //if size = 4 output DESTROYER
       //if size = 5 output BATTLESHIP
       //only output this message if error occurs
       //call function to place ship
       //call function to show ships map data

//************************   rngFind   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Function to check if the ship will fit on the board and is not
//**           placed on top of another ship
//** Inputs: int ships[][COLS], int rows, int size, int putRow,
//**         int intCol, char verHor
//** Outputs: true or false (boolean)
```

```
//*************************************************************************
   //boolean false if ship off board or overlapping another ship
   //if vertical placement
       //check to see if ship will be off the map
           //if ship off map set boolean to false
       //otherwise check overlap
           //loop through rows to check size
               //check to see if ship will overlap another
                   //if ships overlap set boolean to false
                   //set to true if ships overlap
                   //set boolean true if ship placement not overlap
   //otherwise placement horizontal
       //if ship will extend off the map
           //set boolean to false
       //otherwise check for overlap
           //loop through columns to check size
               //check to see if position will overlap
                   //set boolean to false ship placement not possible
                   //set overlap to true
                   //set boolean true if ship can be placed here
   //return ship placement boolean

//************************   putShip   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //Function to place a ship on the gameboard
//** Inputs: int ships[][COLS], int rows, int size
//** Outputs: none
//*************************************************************************
   //variable to choose ship column placement
   //variable to choose horizontal or vertical placement
   //variable to hold char converted to int for column
   //variable to choose row placement
   //boolean set to false until ship successfully placed
   //loop while ship not placed
       //input column for ship placement
       //loop until range and type okay
           //input column for ship placement
       //if column choice is upper case letter
           //set int for conversion to value - 65
       //otherwise choice was lowercase
           //set int for conversion to value - 97
       //choose row to guess
       //loop check data type and range
           //input row until valid range and type
       //input vertical or horizontal placement
       //loop check data type and range
           //input vertical or horizontal until within range
       //if vertical placement
           //call function to check overlap and map ranges
               //if overlap or out of map set ship placemnt boolean to false
           //otherwise place the ship
               //loop through rows for size of ship
                   //set array value to 1 (ship present)
                   //set ship placement to true
```

```
        //otherwise placement is horizontal
            //call function to check overlap and map ranges
                //set ship placement to false if overlap or off map
            //otherwise place ship
                //loop through columns for ship size
                    //set array value to 1 (ship present)
                    //set ship placement to true

//*************************   setGame   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //function to clear the gameboard and set all values to 0
//** Inputs: int ship[][COLS], int guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
   //run through the rows
        //run through the columns
            //set array for guesses all to false
            //set array for ships all to false

//*************************   disGame   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456789
//000000001111111111222222222233333333334444444444555555555566666666667777777777
//** Purpose:  //function to display the game board
//** Inputs: int guess[][COLS], int ship[][COLS], int rows
//** Outputs:
//*****************************************************************************
   //output map legend
   //loop through rows
        //output row number
        //loop through columns
            //if guess array value zero
                //output cyan 'O'
            //if guess array value 1
                //output black 'M'
            //otherwise for array value 2 output red inverse 'H'
   //output map legend
   //loop through rows
        //output row number
        //loop through columns
            //if array value 0
                //output cyan 'O'
            //if array value 1
                //output black 'X'
            //if array value 2
                //output red inverse 'H'
            //otherwise for array value -1
                //output magenta 'o'
```

4.2 Advanced Concepts Used

- Dynamic Arrays
- Static Arrays

- Reading Binary Files
- Writing Binary Files
- Reading Text Files
- Searching Arrays
- Custom Structures
- Structures containing arrays
- Passing structure into function
- Returning structures from functions
- Boolean functions
- Memory management

5 References

Extensive information was researched on Google.
The most valuable repositories of information was from:

www.stackoverflow.com
www.cplusplus.com

6 Source Code

colors.h //header file for using ASCII color in terminal

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * File:   colors.h
 * Author: scott_r_parker
 *
 * Created on April 10, 2017, 8:57 PM
 */

#ifndef COLORS_H
#define COLORS_H


#define RESET   "\033[0m"
#define BLACK   "\033[30m"      /* Black */
#define RED     "\033[31m"      /* Red */
#define GREEN   "\033[32m"      /* Green */
#define YELLOW  "\033[33m"      /* Yellow */
#define BLUE    "\033[34m"      /* Blue */
#define MAGENTA "\033[35m"      /* Magenta */
#define CYAN    "\033[36m"      /* Cyan */
#define WHITE   "\033[37m"      /* White */
```

```
#define BOLDBLACK   "\033[1m\033[30m"      /* Bold Black */
#define BOLDRED     "\033[1m\033[31m"      /* Bold Red */
#define BOLDGREEN   "\033[1m\033[32m"      /* Bold Green */
#define BOLDYELLOW  "\033[1m\033[33m"      /* Bold Yellow */
#define BOLDBLUE    "\033[1m\033[34m"      /* Bold Blue */
#define BOLDMAGENTA "\033[1m\033[35m"      /* Bold Magenta */
#define BOLDCYAN    "\033[1m\033[36m"      /* Bold Cyan */
#define BOLDWHITE   "\033[1m\033[37m"      /* Bold White */

#define INVERSE     "\033[7m"     /* Swap Bacground and Text colors */
#define UNDERLINE       "\033[4m"     /* Underline Single */

#define BGBLACK     "\033[40m"      /* BLACK Background */
#define BGRED       "\033[41m"      /* RED Background */
#define BGGREEN     "\033[42m"      /* GREEN Background */
#define BGYELLOW    "\033[43m"      /* YELLOW Background */
#define BGBLUE      "\033[44m"      /* BLUE Background */
#define BGMAGENTA   "\033[45m"      /* MAGENTA Background */
#define BGCYAN      "\033[46m"      /* CYAN Background */
#define BGWHITE     "\033[47m"      /* WHITE Background */

#endif /* COLORS_H */
```

player.h //File that holds player structure info

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/*
 * File:   Player.h
 * Author: scott_r_parker
 *
 * Created on April 13, 2017, 2:36 PM
 */

#ifndef PLAYER_H
#define PLAYER_H

struct Player{
    int guesses=0;
    int hits=0;
    const int ROWS=12;
    const int COLS=26;
    int guess[12][26]={};
    int ship[12][26]={};
};
```

```
#endif /* PLAYER_H */
```

main.cpp //Main program file

```
/*
 * File:   main.cpp
 * Author: Scott Parker
 * Created on April 10, 2017, 2:00 PM
 * Purpose: Battleship game Project 1
 * Notes: In the ship array 0 indicates ocean (cyan O), 1 indicates a ship (black X), 2 indicates
 *     a ship that has been hit (red X) and -1 is a guess from the enemy player
 *
 *        In the the guess array 0 indicates ocean (cyan O), 1 indicates a miss (black X) and 2
 *     indicates a hit (red X)
 */

//System Libraries
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <iomanip>
#include <cctype>
#include <fstream>
#include <string>
#include <cctype>
using namespace std;

//User Libraries
#include "colors.h"
#include "player.h"

//Global Constants
//Such as PI, Vc, -> Math/Science values
//as well as conversions from one system of measurements to another
const int COLS=26;

//Function Prototypes
void disGame(int [][COLS], int [][COLS], int); //function to display the game board
void setGame(int [][COLS], int [][COLS], int); //function to clear the gameboard
void putShip(int [][COLS], int, int); //Function to place a ship on the gameboard
bool rngFind(int [][COLS], int, int, int, int, char); //Function to see if ship is being placed in valid
position
void pAllShp(int [][COLS], int); //Function to place all ships on the gameboard
void disShip(int [][COLS], int); //function to display the ship map only
void newGame(Player, Player, int); //function to play the game
void clrData(int [][COLS], int [][COLS], int [][COLS], int [][COLS], int); //function to reset all
current game data
void pGameH(Player, Player, int); //Play a game with two human players
void pGameC(Player, Player, int); //Play a game versus the computer
void pShipC(int [][COLS], int); //function to place all ships for computer player
```

```
int entGues(int [][COLS], int [][COLS], int); //function for player to enter a guess
int comGues(int [][COLS], int [][COLS], int, int); //function for computer to enter a guess
void savGame(int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS], int
p2guess[][COLS], int rows); //Function to save the game
void lodGame(int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS], int
p2guess[][COLS], int rows); //Function to load a saved game
void resGame(Player, Player, int rows); //Function to resume a saved game
void resHum(Player, Player, int rows); //Function to resume a Human vs Human game
void resComp(Player, Player, int rows); //Function to resume a Human vs Comp game
Player newPlyr(); //Function to initiate player data and start game


//Executable code begins here! Always begins in Main
int main(int argc, char** argv) {
    //Set random seed
    srand(static_cast<unsigned int>(time(0)));

    //Declare Variables
    int choice=0;
    const int ROWS=12;

    Player p1=newPlyr();
    Player p2=newPlyr();

    //Game menu
    do{
        //Output switch menu screen
        cout<<"Choose from the list <non-numeric data will be ignored>"<<endl;
        cout<<"Enter 1 to resume a saved game"<<endl; //enter 1 to resume saved game
        cout<<"Enter 2 to start a new game"<<endl; //enter 2 for a new game
        cout<<"Enter 0 (zero) or a number not listed to exit."<<endl; //0 or unlisted number to exit
        cin>>choice;
        while (cin.fail()) { //Loop to validate input
            cout<<"INPUT INVALID! "<<endl;
            cin.clear();  //Resetting flags
            cin.ignore(256, '\n'); //ignore contents of buffer
            cout<<"Please enter a valid choice!:"<<endl;
            cin>>choice;
        }

        //Switch to determine the Problem
        switch(choice){
            case 2:{newGame(p1, p2, ROWS);break;} //start a new game
            case 1:{resGame(p1, p2, ROWS);break;} //Function to load a saved game
            default:
                cout<<"You are exiting the game"<<endl; //default option - exit menu
        }
    }while(choice>=1&&choice<=2); //show menu while choices all active

    //Exit stage right! - This is the 'return 0' call
    return 0;
```

```
}

//*************************  newPlyr  ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //Function to initiate player data and start game
//** Inputs: none
//** Outputs: none
//*****************************************************************************
Player newPlyr(){
   Player a;
   for (int i=0;i<a.ROWS;i++){
      for (int j=0;j<a.COLS;j++){
         a.guess[i][j]=0;
         a.ship[i][j]=0;
      }
   }
   return a;
}


//************************  resHum  ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //Function to resume a Human vs Human game
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
void resHum(Player p1, Player p2, int rows){
   string pauser=""; //string to use for pausing screen
   for (int i=0;i<rows;i++){ //for loop to loop through rows
      for (int j=0;j<COLS;j++){ //for loop to loop through columns
         if (p1.guess[i][j]!=0){ //each non 0 occurance is a guess
            p1.guesses++; //increment total guesses
            if (p1.guess[i][j]==2){ //if guess value == 2 that guess was a hit
               p1.hits++; //increment number of hits
            }
         }
         if (p2.guess[i][j]!=0){ //check if guess (value of array not zero)
            p2.guesses++; //increment total guesses if array value is a guess
            if (p2.guess[i][j]==2){ //check to see if guess was a hit
               p2.hits++; //increment hit counter if value was hit (array value 2)
            }
         }
      }
   }
```

```
   do { //begin do loop
      cin.clear();
      cout<<"Beginning round: "<<p1.guesses+1<<endl; //output round number (guesses so far
plus 1)
      cout<<"Player 1 enter coordinates, Player 2 turn your back:"<<endl;
      cout<<"<Press enter to clear screen>"<<endl; //notify of pause
      getline (cin, pauser); //pause function to delay until <enter> pressed
      for (int i=0;i<5000;i++){ //for loop to output new lines to clear screen
         cout<<endl;
      }
      disGame(p1.guess, p1.ship, rows); //call function to display player 1 game status
      p1.hits+=entGues(p1.guess, p2.ship, rows); //call function to enter guess and increment p1
hits if hit
      p1.guesses++; //increment total number of guesses so far
      cout<<"<Press enter to clear screen>"<<endl; //notify of pause
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter
      for (int i=0;i<5000;i++){ //for loop to clear screen
         cout<<endl;
      }
      cout<<"Player 2 enter coordinates, Player 1 turn your back:"<<endl;
      cout<<"<Press enter to clear screen>"<<endl; //notify of pause
      getline (cin, pauser); //pause until enter pressed
      for (int i=0;i<1000;i++){ //clear screen
         cout<<endl;
      }
      disGame(p2.guess, p2.ship, rows); //call function to display game map for player 2
      p2.hits+=entGues(p2.guess, p1.ship, rows); //call function to guess for player 2 - increment
hits if hit
      p2.guesses++; //increment player 2 number of guesses
      cout<<"<Press enter to clear screen>"<<endl; //notify of pause
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter pressed
      for (int i=0;i<1000;i++){ //clear screen
         cout<<endl;
      }
      cout<<"Current Game Status:"<<endl;
      cout<<"Player 1 hits: "<<p1.hits<<" guesses: "<<p1.guesses<<endl; //output player 1 hits
and guesses
      cout<<"Player 2 hits: "<<p2.hits<<" guesses: "<<p1.guesses<<endl; //output player 2 hits
and guesses
      cout<<"<Press enter to continue, 'N' or 'n' to save and exit>"<<endl; //notify of pause
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter pressed
   } while (p1.hits<14 && p2.hits<14 && pauser!="n" && pauser!="N"); //repeat loop until a player
gets 14 hits or saves game
   if (pauser=="n" || pauser=="N") savGame(p1.ship, p1.guess, p2.ship, p2.guess, rows); //if
game ends because of save then call save function
   cout<<"Exiting Game!"<<endl; //exit game message
}
```

```
//**************************  resComp  ************************************
//23456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
//00000000111111111122222222223333333333444444444455555555556666666666777777777
//** Purpose:  //Function to resume a Human vs Comp game
//** Inputs:  int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**          int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
void resComp(Player p1, Player p2, int rows){
   string pauser=""; //string to be used for pausing
   for (int i=0;i<rows;i++){ //loop through rows
      for (int j=0;j<COLS;j++){ //loop through columns
         if (p1.guess[i][j]!=0){ //check if guess (value of array not zero)
            p1.guesses++; //increment total guesses if array value is a guess
            if (p1.guess[i][j]==2){ //check to see if guess was a hit
               p1.hits++; //increment hit counter if value was hit (array value 2)
            }
         }
         if (p2.guess[i][j]!=0){ //check if guess (value of array not zero)
            p2.guesses++; //increment total guesses if array value is a guess
            if (p2.guess[i][j]==2){ //check to see if guess was a hit
               p2.hits++; //increment hit counter if value was hit (array value 2)
            }
         }
      }
   }
   do { //begin do loop
      disGame(p1.guess, p1.ship, rows); //display player 1 game map
      cout<<"Player 1 please enter your guess:"<<endl;
      p1.hits+=entGues(p1.guess, p2.ship, rows); //call guess function, increment player 1 hits if guess hits
      p1.guesses++; //increment number of player 1 guesses
      cout<<"Computer guessing now:"<<endl;
      p2.hits+=comGues(p2.guess, p1.ship, rows, p2.guesses); //call comp guess function increment player 2 hits if guess hits
      p2.guesses++; //increment player 2 guesses
      cout<<"Current Status: "<<endl;
      cout<<"Player 1 hits: "<<p1.hits<<" guesses: "<<p1.guesses<<endl; //output player 1 hits and guesses
      cout<<"Player 2 hits: "<<p2.hits<<" guesses: "<<p2.guesses<<endl; //output player 2 hits and guesses
      cout<<"<Press enter to continue, 'N' or 'n' to save and exit>"<<endl; //notify of pause
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter pressed or save char entered
   } while (p1.hits<14 && p2.hits<14 && pauser!="n" && pauser!="N"); //loop until 14 hits scored by player or game saved
   if (pauser=="n" || pauser=="N") savGame(p1.ship, p1.guess, p2.ship, p2.guess, rows); //call game save function if necessary
```

```
   cout<<"Exiting Game!"<<endl;
}


//************************  resGame  ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //Function to resume a saved game
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
void resGame(Player p1, Player p2, int rows){
   char chooser='\0'; //variable used for choice options
   lodGame(p1.ship, p1.guess, p2.ship, p2.guess, rows); //Load a saved game
   cout<<"Playing versus (C)omputer or (H)uman?"<<endl;
   cout<<"**Player 1 always Human in vs. computer!!!**"<<endl;
   cout<<"Enter C/H now!"<<endl;
   cin>>chooser; //choose computer or human opponent
   chooser=toupper(chooser); //force variable to upper case
   while (cin.fail() || (chooser!='C' && chooser!='H')){ //loop to validate input type and range
      cin.clear();
      cin.ignore(256, '\n');
      cout<<"Enter 'C' or 'H' to continue!"<<endl; //output data range options
      cin>>chooser; //input choice
      chooser=toupper(chooser); //force to upper case
   }
   if (chooser=='C') { //if choice is to play computer game
      resComp(p1, p2, rows); //call function to resume game vs computer
   } else { //otherwise
      resHum(p1, p2, rows); //call function to resume game vs human
   }
}


//************************  lodGame  ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //Function to load a saved game
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
void lodGame(int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS], int
p2guess[][COLS], int rows){
```

```
   string line=""; //hold file names from bones file. also used as temp to hold names while
counting lines
   fstream fil; //create filestream object
   clrData(p1ship, p1guess, p2ship, p2guess, rows); //clear any game data in memory
   cout<<endl;
   fil.open ("bones.txt", ios::in); //open bones file in input mode
   if (fil.fail()){ //output error if file did not open correctly
      cout<<"ERROR! Unable to open BONES file!"<<endl;
   } else {
      cout<<"List of *Savefiles*: "<<endl;
      while (fil){ //loop through file line by line
         getline(fil, line); //get file line, increment to next line
         cout<<"*"<<line<<"*"<<endl; //output text from that line in file
      }
   }
   fil.close(); //close file

   cout<<"Enter name of game to load: <Do not include * characters>"<<endl;
   cin.ignore(256, '\n');
   getline (cin, line); //enter name of savefile to open
   fil.open(line+".bin", ios::in | ios::binary); //open binary file name provided
   if (fil.fail()){
      cout<<"ERROR! Unable to open file!"<<endl;
      cout<<"Randomly creating new game state!"<<endl;
      pShipC(p1ship, rows); //randomly place player 1 ships
      pShipC(p2ship, rows); //randomly place player 2 ships
   } else { //read in binary file data to populate game array data
      fil.read(reinterpret_cast<char *>(p1ship), (rows*COLS)*sizeof(int)); //read file to fill player 1
ship data
      fil.read(reinterpret_cast<char *>(p1guess), (rows*COLS)*sizeof(int)); //read file to fill player
1 guess data
      fil.read(reinterpret_cast<char *>(p2ship), (rows*COLS)*sizeof(int)); //read file to fill player 2
ship data
      fil.read(reinterpret_cast<char *>(p2guess), (rows*COLS)*sizeof(int)); //read file to fill player
2 guess data
      fil.close();
   }
}


//**********************   savGame   ****************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //Function to save the game
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*******************************************************************************
```

```
void savGame(int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS], int
p2guess[][COLS], int rows){
   string line=""; //hold file names from bones file. also used as temp to hold names while
counting lines
   string *gName=nullptr; //array of strings to hold file names
   int count=0, temp=0; //count variable and temp variable
   fstream fil; //create filestream object
   fil.open ("bones.txt", ios::in); //open bones file in input mode
   if (fil.fail()) {
      fil.open ("bones.txt", ios::out);
      fil.close();
   }
   if (fil.fail()){ //output error if file unable to open
      cout<<"ERROR! Unable to open BONES file!"<<endl;
   } else { //otherwise
      while (fil){ //find the number of lines in the file
         getline(fil, line); //get file line, increment to next line
         count++; //increment counter of the number of lines
      }
   }
   fil.close(); //close file

   cout<<"Count value: "<<count<<endl;
   if (count<2) count=2;
   gName=new string [count]; //create dynamic array to hold bones file data
   fil.open ("bones.txt", ios::in); //open bones file in input mode
   if (fil.fail()){ //output error if file fails to open
      cout<<"ERROR! Unable to open bones file!"<<endl;
   } else { //otherwise
      for (int i=0;i<count;i++){ //loop through file line by line
         getline(fil, gName[i]); //get file line, increment to next array element
      }
   }
   fil.close(); //close file
   cout<<"Current Save Files"<<endl;
   for (int i=0;i<count;i++) { //loop through array line by line
      if (gName[i]!=""){ //display array element if not empty line
         cout<<"Save file name: "<<gName[i]<<endl;
      }
   }

   cout<<BOLDRED<<"WARNING! Duplicate names will overwrite!"<<RESET<<endl;
   cout<<BOLDRED<<"WARNING! Only enter alphanumeric characters!"<<RESET<<endl;
   cout<<"Enter the game name to save: "<<endl;
   getline(cin, line); //enter the name of new savefile

   temp=0;
   while (gName[temp]!=line && temp<count){
      temp++;
      if (temp == count-1) {
         gName[temp]=line;
```

```
      }
   }
   if (temp<count-1){
      cout<<"file "<<line<<" overwritten!"<<endl;
   }

   cout<<"file names in array"<<endl;
   for (int i=0;i<count;i++){
      cout<<gName[i]<<endl;
   }

   fil.open("bones.txt", ios::out); //open bones file in output mode
   if (fil.fail()) { //output error message if file fails to open
      cout<<"ERROR! Unable to open bones file!"<<endl;
   } else { //otherwise
      for (int i=0;i<count;i++){ //loop through array line by line
         if (gName[i]!=""){ //if line is not blank
            fil<<gName[i]<<endl; //output array element as line in file
         }
      }
   }
   fil.close(); //close file
   delete [] gName; //clean up memory and delete dynamic array
   cout<<"Saving Game!"<<endl;
   fil.open(line+".bin", ios::out | ios::binary); //open binary file in output mode to save game data
   fil.write(reinterpret_cast<char *>(p1ship), (rows*COLS)*sizeof(int)); //write player 1 ship data
to file
   fil.write(reinterpret_cast<char *>(p1guess), (rows*COLS)*sizeof(int)); //write player 1 guess
data to file
   fil.write(reinterpret_cast<char *>(p2ship), (rows*COLS)*sizeof(int)); //write player 2 ship data
to file
   fil.write(reinterpret_cast<char *>(p2guess), (rows*COLS)*sizeof(int)); //write player 2 guess
data to file
   fil.close(); //close file
}


//**************************   entGues   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //function for player to enter a guess
//** Inputs: int pGuess[][COLS], int opShip[][COLS], int rows
//** Outputs: 0 if miss, 1 if hit
//*******************************************************************************
int entGues(int pGuess[][COLS], int opShip[][COLS], int rows){
   char putCol='\0'; //character variable to hold column data for guess
   int intCol=0; //variable to hold converted character variable as int
   int putRow=0; //variable to hold row data for guess
   int hitMiss=0; //variable to set to return 1 if hit, 0 if miss
```

```
   bool shipSet=false; //bool variable set to false until successful guess entered
   while (!shipSet){ //loop until successful guess
      cout<<"Enter Column (A through Z)"<<endl;
      cin>>putCol; //enter column
      while (cin.fail() || !isalpha(putCol)){ //loop to validate that input was valid and in range
         cin.clear(); //clear cin error flags
         cin.ignore(256, '\n'); //clear buffer
         cout<<"CHEAT ACTIVATED! SHOWING COMPUTER SHIPS!"<<endl;
         disShip(opShip, rows); //activate cheat if out of range data input
         cout<<"You must enter a letter A to Z..."<<endl;
         cin>>putCol; //input data for column to guess
      }
      if (isupper(putCol)){ //if upper case data entered
         intCol=(putCol-65); //subtract 65 from value and set column number guess to new value
      } else { //else data is lower case character
         intCol=(putCol-97); //subtract 97 from value and set column number guess to new value
      }
      cout<<"Enter Row (0 to 11)"<<endl;
      cin>>putRow; //input row number for guess
      while (cin.fail() || putRow<0 || putRow>=rows){ //loop to validate input range and type
         cin.clear(); //clear cin flags
         cin.ignore(256, '\n'); //clear buffer
         cout<<"Enter a row from 0 to 11.."<<endl;
         cin>>putRow; //input row guess until valid entry
      }
      if (pGuess[putRow][intCol]!=0) { //if guess array value not zero this value already guessed
so repeat guess
         cout<<"Position already guessed!"<<endl;
      } else if (opShip[putRow][intCol]==1) { //otherwise if enemy ship array value = 1 HIT
         pGuess[putRow][intCol]=2; //set value of player guess array to 2
         cout<<"Hit!"<<endl;
         opShip[putRow][intCol]=2; //set value of enemy ship array to 2
         hitMiss=1; //set hit value to 1
         shipSet=true; //set boolean as true since successfully entered guess
      } else { //otherwise the guess was a miss
         pGuess[putRow][intCol]=1; //set player guess array value to 1 to show miss
         opShip[putRow][intCol]=-1; //set enemey ship array value to -1 to show miss
         cout<<"Miss!"<<endl;
         hitMiss=0; //set hit value to 0 (since coordinate was a miss)
         shipSet=true; //set boolean to true since guess was successfully made without error
      }
   }
   return hitMiss;  //return hit value from function
}


//************************  comGues  ***************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
```

```
//** Purpose:  //function for computer to enter a guess
//** Inputs: int pGuess[][COLS], int opShip[][COLS], int rows, int gues
//** Outputs: 0 if miss, 1 if true
//*************************************************************************
int comGues(int pGuess[][COLS], int opShip[][COLS], int rows, int gues){
   int intCol=rand()%COLS; //random column guess value
   int putRow=rand()%rows; //random row guess value
   int hitMiss=0; //default hit value zero
   bool shipSet=false; //boolean false until successful guess
   while (!shipSet){
      if (gues>0 && gues%11==0){ //if total guesses > 0 and guesses%11==0 then give comp a
free hit
         for (int i=0;i<rows && !shipSet;i++) { //loop through rows while boolean false
            for (int j=0;j<COLS && !shipSet;j++){ //loop through columns while boolean false
               if (pGuess[i][j]==0){ //if player has not guessed this coordinate already
                  if (opShip[i][j]==1){ //if this coordinate has an enemy ship that has not been hit
                     cout<<"Lucky Hit!"<<endl;
                     pGuess[i][j]=2; //set player guess value to 2 (hit)
                     opShip[i][j]=2; //set opponent ship value to 2 (hit)
                     shipSet=true; //set boolean to true
                     hitMiss=1; //set hit value to 1
                  }
               }
            }
         }
      } else if (pGuess[putRow][intCol]!=0) { //otherwise if player has already guessed this
position
         cout<<"Position already guessed!"<<endl;
         intCol=rand()%COLS; //get new column guess
         putRow=rand()%rows; //get new row guess
      } else if (opShip[putRow][intCol]==1) { //otherwise if enemy has ship here
         pGuess[putRow][intCol]=2; //set player guess value to 2 (hit)
         cout<<"Hit!"<<endl;
         opShip[putRow][intCol]=2; //set enemy ship map value to 2 (hit)
         hitMiss=1; //set hit value to 1
         shipSet=true; //set boolean to true (successfully entered guess)
      } else { //otherwise position not guessed but no ship here
         pGuess[putRow][intCol]=1; //set player guess value to 1 (miss)
         opShip[putRow][intCol]=-1; //set enemy map value to -1 (to show enemy shot missed)
         cout<<"Miss!"<<endl;
         hitMiss=0; //set hit value to zero
         shipSet=true; //set boolean to true (guess attempted successfully)
      }
   }
   return hitMiss; //return hit value
}


//*************************   pGameH   ***************************************
//23456789012345678901234567890123456789012345678901234567890123456789012345678901234567
789
```

```
//0000000011111111112222222222333333333344444444455555555556666666666777777777
//** Purpose:  //Play a game with two human players
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*******************************************************************************
void pGameH(Player p1, Player p2, int rows) {
   string pauser=""; //string used for pauses
   do { //begin do loop
      cin.clear(); //clear cin flags
      cout<<"Beginning round: "<<p1.guesses+1<<endl;
      cout<<"Player 1 enter coordinates, Player 2 turn your back:"<<endl;
      cout<<"<Press enter to clear screen>"<<endl; //notify of pause
      getline (cin, pauser); //pause until enter key pressed
      for (int i=0;i<1000;i++){ //clear screen
         cout<<endl;
      }
      disGame(p1.guess, p1.ship, rows); //call function to display player 1 game data
      p1.hits+=entGues(p1.guess, p2.ship, rows); //call function for player 1 to enter a guess -
increment hits if needed
      p1.guesses++; //increment player 1 guess count
      cout<<"<Press enter to clear screen>"<<endl; //notify of pause
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter pressed
      for (int i=0;i<1000;i++){ //clear screen
         cout<<endl;
      }
      cout<<"Player 2 enter coordinates, Player 1 turn your back:"<<endl;
      cout<<"<Press enter to clear screen>"<<endl; //notify of pause
      getline (cin, pauser); //pause until enter key pressed
      disGame(p2.guess, p2.ship, rows); //display player 2 game data
      p2.hits+=entGues(p2.guess, p1.ship, rows); //call function for player 2 to enter a guess -
increment hits if needed
      p2.guesses++; //increment player 2 guess count
      cout<<"<Press enter to clear screen>"<<endl; //notify of pause
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter key pressed
      for (int i=0;i<1000;i++){ //clear screen
         cout<<endl;
      }
      cout<<"Current Game Status:"<<endl;
      cout<<"Player 1 hits: "<<p1.hits<<" guesses: "<<p1.guesses<<endl; //output player 1 hits,
player 1 guesses
      cout<<"Player 2 hits: "<<p2.hits<<" guesses: "<<p2.guesses<<endl; //output player 2 hits,
player 2 guesses
      cout<<"<Press enter to continue, 'N' or 'n' to save and exit>"<<endl; //notify of pause
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter key pressed
   } while (p1.hits<14 && p2.hits<14 && pauser!="n" && pauser!="N"); //loop until player has 14
hits or game saved
```

```
   if (pauser=="n" || pauser=="N") savGame(p1.ship, p1.guess, p2.ship, p2.guess, rows); //save
game if required
   cout<<"Exiting Game!"<<endl;
}


//**************************   pGameC   *************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //Play a game versus the computer
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*******************************************************************************
void pGameC(Player p1, Player p2, int rows) {
   string pauser=""; //string for pausing
   do { //begin do loop
      cout<<"Player 1 please enter your guess:"<<endl;
      p1.hits+=entGues(p1.guess, p2.ship, rows); //call function for player 1 to enter a guess -
increment hits if needed
      p1.guesses++; //increment player 1 guess count
      cout<<"Computer guessing now:"<<endl;
      p2.hits+=comGues(p2.guess, p1.ship, rows, p2.guesses); //call function for computer
guess - increment hits if needed
      p2.guesses++; //increment player 2 guess count
      cout<<"Current Status: "<<endl;
      disGame(p1.guess, p1.ship, rows); //call function to display player 1 game data
      cout<<"Player 1 hits: "<<p1.hits<<" guesses: "<<p1.guesses<<endl; //output player 1 hits
and player 1 guesses
      cout<<"Player 2 hits: "<<p2.hits<<" guesses: "<<p1.guesses<<endl; //output player 2 hits
and player 2 guesses
      cout<<"<Press enter to continue, 'N' or 'n' to save and exit>"<<endl; //notify of pause
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter pressed
   } while (p1.hits<14 && p2.hits<14 && pauser!="n" && pauser!="N"); //loop until player has 14
hits or saves game
   if (pauser=="n" || pauser=="N") savGame(p1.ship, p1.guess, p2.ship, p2.guess, rows); //save
game if necessary
   cout<<"Exiting Game!"<<endl;
}


//****************************   pShipC   ************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //function to place all ships for computer player
//** Inputs: int ship[][COLS], int rows
```

```
//** Outputs: none
//*****************************************************************************
void pShipC(int ship[][COLS], int rows){
   char verHor='\0'; //variable to hold choice
   int intCol=0; //variable to hold column number
   int putRow=0; //variable to hold row number
   bool shipSet=false; //boolean false until ship successfully placed

   for (int size=5;size>=2;size--) { //loop down from max ship size to min
      cout<<"Randomly Placing... ";
      if (size==2) cout<<"PT BOAT!"<<endl; //output PT Boat if size = 2
      else if (size==3) cout<<"SUBMARINE!"<<endl; //output SUBMARINE if size = 3
      else if (size==4) cout<<"DESTROYER!"<<endl; //output DESTROYER if size = 4
      else if (size==5) cout<<"BATTLESHIP!"<<endl; //output BATTLESHIP if size = 5
      do { //begin do loop
         intCol=rand()%COLS; //random column value
         putRow=rand()%rows; //random row value
         rand()%2==0?verHor='v':verHor='h'; //randomly decide vertical or horizontal placement
         if (verHor=='v'){ //if ship placement vertical
            if (!rngFind(ship, rows, size, putRow, intCol, verHor)){ //check to see ship will fit or
overlap
               shipSet=false; //boolean false if ships overlap or out of bounds
               cout<<"Invalid location! Try again!"<<endl;
            } else { //otherwise
               for (int i=0;i<size;i++){ //Loop through rows from start position to ship size
                  ship[putRow+i][intCol]=1; //set array value to 1 (ship present)
                  shipSet=true; //boolean true, ship successfully placed
               }
            }
         } else { //otherwise ship placement horizontal
            if (!rngFind(ship, rows, size, putRow, intCol, verHor)){ //check to see if ship will fit or
overlap
               shipSet=false; //set boolean false if ships overlap or out of bounds
               cout<<"Invalid location! Try again!"<<endl;
            } else { //otherwise
               for (int i=0;i<size;i++){ //loop through columns
                  ship[putRow][intCol+i]=1; //set array value to 1 (ship present)
                  shipSet=true; //boolean true, ship successfully placed
               }
            }
         }
      } while (!shipSet); //loop until ship placement successful
   }
}


//************************  newGame  ***************************************
//2345678901234567890123456789012345678901234567890123456789012345678901234567890123456
789
//0000000011111111112222222222333333333344444444445555555555666666666677777777
777
```

```
//** Purpose:  //function to play a new game
//** Inputs:  int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**          int p2guess[][COLS], int rows
//** Outputs: none
//*******************************************************************************
void newGame(Player p1, Player p2, int rows){
   char chooser='\0'; //choice variable
   string pauser=""; //pause input
   cout<<"(H)uman or (C)omputer for player 2? <Enter H or C>"<<endl;
   cin>>chooser; //Choose to play against a human or the computer
   while (cin.fail() || (chooser!='h' && chooser!='H' && chooser!='c' && chooser!='C')) { //validate
data
      cin.clear(); //clear cin flags
      cin.ignore(256, '\n'); //ignore contents of buffer
      cout<<"Please enter 'H' or 'C'!"<<endl; //output available choices
      cin>>chooser; //input choice again until valid
   }
   if (chooser=='h' || chooser=='H') { //if human opponent chosen
      cout<<"<Press enter for Player 1 to place ships>"<<endl; //notify Player 1 to place ships
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter pressed
      pAllShp(p1.ship, rows); //Place all ships on for player 1
      cout<<"<Press enter to clear screen>"<<endl; //notify of pause
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter pressed
      for (int i=0;i<1000;i++){ //clear screen
         cout<<endl;
      }
      cout<<"<Press enter for Player 2 to place ships>"<<endl; //notify Player 2 to place ships
      getline (cin, pauser); //pause until enter pressed
      pAllShp(p2.ship, rows); //Place all ships on for player 2
      cout<<"<Press enter to clear screen>"<<endl; //notify of pause
      cin.ignore(256, '\n'); //clear buffer
      getline (cin, pauser); //pause until enter pressed
      for (int i=0;i<1000;i++){ //clear screen
         cout<<endl;
      }
      cout<<"Starting game..."<<endl;
      pGameH(p1, p2, rows); //call function to Play the game with two humans
   } else { //otherwise
      cout<<"Player 1: Place your ships on the map!"<<endl;
      pAllShp(p1.ship, rows); //Place all ships on for player 1
      cout<<"Computer placing ships now..."<<endl;
      pShipC(p2.ship, rows); //place all ships for computer player
      pGameC(p1, p2, rows); //call function to Play the game vs. computer
   }
}


//************************   clrData   ***************************************
```

```
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //function to reset all current game data
//** Inputs: int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS],
//**         int p2guess[][COLS], int rows
//** Outputs: none
//*******************************************************************************
void clrData(int p1ship[][COLS], int p1guess[][COLS], int p2ship[][COLS], int p2guess[][COLS],
int rows){
   setGame(p1guess, p1ship, rows);  //Reset game data for player 1
   setGame(p2guess, p2ship, rows);  //Reset game data for player 2
}


//************************  disShip  ***************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //function to display a players ship data only
//** Inputs: int ship[][COLS], int rows
//** Outputs:  none
//*******************************************************************************
void disShip(int ship[][COLS], int rows){
   cout<<"YOUR SHIP POSITIONS"<<endl;
   cout<<"   ABCDEFGHIJKLMNOPQRSTUVWXYZ"<<endl;
   for (int i=0;i<rows;i++){ //loop through rows
      cout<<setw(3)<<left<<i; //output row number
      for (int j=0;j<COLS;j++){ //loop through columns
         if (ship[i][j]==0){ //if no ship present
            cout<<CYAN<<"O"<<RESET; //output cyan 'O'
         }
         else if (ship[i][j]==1){ //if undamaged ship present
            cout<<BLACK<<"X"<<RESET; //output black 'X'
         } else if (ship[i][j]==2) { //if damaged ship present
            cout<<RED<<INVERSE<<"H"<<RESET; //output red inverse 'H'
         } else { //if other player shot this location but missed
            cout<<MAGENTA<<"o"<<RESET; //output magenta 'o'
         }
      }
      cout<<endl;
   }
   cout<<endl;
}


//************************  pAllShp  ***************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
```

```
//000000001111111112222222222333333333344444444455555555556666666666777777777
//** Purpose:  //Function to place all ships on the gameboard
//** Inputs: int ships[][COLS], int rows
//** Outputs: none
//*****************************************************************************
void pAllShp(int ships[][COLS], int rows){
   for (int i=5;i>=2;i--){ //loop down through size from Battleship to PT Boat
      cout<<"Choose where to place your ";
      if (i==2) cout<<"PT BOAT!"<<endl; //if size = 2 output PT Boat
      else if (i==3) cout<<"SUBMARINE!"<<endl; //if size = 3 output SUBMARINE
      else if (i==4) cout<<"DESTROYER!"<<endl; //if size = 4 output DESTROYER
      else if (i==5) cout<<"BATTLESHIP!"<<endl; //if size = 5 output BATTLESHIP
      else cout<<"ERROR! SHOULD NEVER GET TO THIS MESSAGE!"<<endl; //only output
this message if error occurs
      putShip(ships, rows, i); //call function to place ship
      disShip(ships, rows); //call function to show ships map data
   }
}


//*************************  rngFind  **************************************
//2345678901234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111112222222222333333333344444444455555555556666666666777777777
//** Purpose:  //Function to check if the ship will fit on the board and is not
//**          placed on top of another ship
//** Inputs: int ships[][COLS], int rows, int size, int putRow,
//**         int intCol, char verHor
//** Outputs: true or false (boolean)
//*****************************************************************************
bool rngFind(int ships[][COLS], int rows, int size, int putRow, int intCol, char verHor){
   bool shipOk=false; //boolean false if ship off board or overlapping another ship
   bool overlap=false;
   if ((verHor=='v') || (verHor=='V')){ //if vertical placement
      if ((putRow+size)>rows){ //check to see if ship will be off the map
         cout<<"Ship off the world!"<<endl;
         shipOk=false; //if ship off map set boolean to false
      } else { //otherwise check overlap
         for (int i=0;i<size && !overlap;i++) { //loop through rows to check size
            if (ships[putRow+i][intCol]!=0) { //check to see if ship will overlap another
               cout<<"Ship collision!"<<endl;
               shipOk=false; //if ships overlap set boolean to false
               overlap=true; //set to true if ships overlap
            } else {
               shipOk=true; //set boolean true if ship placement not overlap
            }
         }
      }
   } else { //otherwise placement horizontal
```

```
        if ((intCol+size)>COLS){ //if ship will extend off the map
            cout<<"Ship off the world!"<<endl;
            shipOk=false; //set boolean to false
        } else { //otherwise check for overlap
            for (int i=0;i<size && !overlap;i++) { //loop through columns to check size
                if (ships[putRow][intCol+i]!=0) { //check to see if position will overlap
                    cout<<"Ship collision!"<<endl;
                    shipOk=false; //set boolean to false ship placement not possible
                    overlap=true; //set overlap to true
                } else {
                    shipOk=true; //set boolean true if ship can be placed here
                }
            }
        }
    }
    return shipOk; //return ship placement boolean
}


//*************************   putShip   ***************************************
//234567890123456789012345678901234567890123456789012345678901234567890123456
789
//000000001111111111222222222233333333334444444444555555555566666666667777777
777
//** Purpose:  //Function to place a ship on the gameboard
//** Inputs: int ships[][COLS], int rows, int size
//** Outputs: none
//*****************************************************************************
void putShip(int ships[][COLS], int rows, int size){
    char putCol='\0'; //variable to choose ship column placement
    char verHor='\0'; //variable to choose horizontal or vertical placement
    int intCol=0; //variable to hold char converted to int for column
    int putRow=0; //variable to choose row placement
    bool shipSet=false; //boolean set to false until ship successfully placed
    while (!shipSet){ //loop while ship not placed
        cout<<"Which column and row should hold the bow of the ship?"<<endl;
        cout<<"Enter Column (A through Z)"<<endl;
        cin>>putCol; //input column for ship placement
        while (cin.fail() || !isalpha(putCol)){ //loop until range and type okay
            cin.clear(); //clear cin flags
            cin.ignore(256, '\n'); //clear buffer
            cout<<"You must enter a letter A to Z..."<<endl;
            cin>>putCol; //input column for ship placement
        }
        if (isupper(putCol)){ //if column choice is upper case letter
            intCol=(putCol-65); //set int for conversion to value - 65
        } else { //otherwise choice was lowercase
            intCol=(putCol-97); //set int for conversion to value - 97
        }
        cout<<"Enter Row (0 to 11)"<<endl;
        cin>>putRow; //choose row to guess
```

```
      while (cin.fail() || putRow<0 || putRow>11){ //loop check data type and range
         cin.clear(); //clear cin flags
         cin.ignore(256, '\n'); //clear buffer
         cout<<"Enter a row from 0 to 11.."<<endl;
         cin>>putRow; //input row until valid range and type
      }
      cout<<"Place the ship vertically or horizontally (V/H)?"<<endl;
      cin>>verHor; //input vertical or horizontal placement
      while (cin.fail() || (verHor!='v' && verHor!='V' && verHor!='h' && verHor!='H')){ //loop check
data type and range
         cin.clear(); //clear cin flags
         cin.ignore(256, '\n'); //clear buffer
         cout<<"Please enter 'V' or 'H'!"<<endl;
         cin>>verHor; //input vertical or horizontal until within range
      }
      if (verHor=='v' || verHor=='V'){ //if vertical placement
         if (!rngFind(ships, rows, size, putRow, intCol, verHor)){ //call function to check overlap
and map ranges
            shipSet=false; //if ship overlap or out of map range set ship placemnt boolean to false
            cout<<"Invalid location! Try again!"<<endl;
         } else { //otherwise place the ship
            for (int i=0;i<size;i++){ //loop through rows for size of ship
               ships[putRow+i][intCol]=1; //set array value to 1 (ship present)
               shipSet=true; //set ship placement to true
            }
         }
      } else { //otherwise placement is horizontal
         if (!rngFind(ships, rows, size, putRow, intCol, verHor)){ //call function to check overlap
and map ranges
            shipSet=false; //set ship placement to false if overlap or off map
            cout<<"Invalid location! Try again!"<<endl;
         } else { //otherwise place ship
            for (int i=0;i<size;i++){ //loop through columns for ship size
               ships[putRow][intCol+i]=1; //set array value to 1 (ship present)
               shipSet=true; //set ship placement to true
            }
         }
      }
   }
}


//************************  setGame  ***************************************
//23456789012345678901234567890123456789012345678901234567890123456789012345678901234567890123456
789
//00000000111111111122222222223333333333444444444455555555556666666666777777777
//** Purpose:  //function to clear the gameboard and set all values to 0
//** Inputs: int ship[][COLS], int guess[][COLS], int rows
//** Outputs: none
//*****************************************************************************
```

```
void setGame(int ship[][COLS], int guess[][COLS], int rows){
   for (int i=0;i<rows;i++){ //run through the rows
      for (int j=0;j<COLS;j++){ //run through the columns
         guess[i][j]=0; //set array for guesses all to false
         ship[i][j]=0; //set array for ships all to false
      }
   }
}


//*************************   disGame   ************************************
//2345678901234567890123456789012345678901234567890123456789012345678901234567890123456789
//0000000011111111112222222222333333333344444444445555555555666666666677777777777
//** Purpose:  //function to display the game board
//** Inputs: int guess[][COLS], int ship[][COLS], int rows
//** Outputs:
//*****************************************************************************
void disGame(int guess[][COLS], int ship[][COLS], int rows){
   cout<<"Your GUESS Map   Ocean="<<CYAN<<"O"<<RESET<<"
Miss="<<BLACK<<"M"<<RESET
         <<"  Hit="<<RED<<INVERSE<<"H"<<RESET<<endl; //output map legend
   cout<<"   ABCDEFGHIJKLMNOPQRSTUVWXYZ"<<endl;
   for (int i=0;i<rows;i++){ //loop through rows
      cout<<setw(3)<<left<<i; //output row number
      for (int j=0;j<COLS;j++){ //loop through columns
         if (guess[i][j]==0){ //if guess array value zero
            cout<<CYAN<<"O"<<RESET; //output cyan 'O'
         }
         else if (guess[i][j]==1){ //if guess array value 1
            cout<<BLACK<<"M"<<RESET; //output black 'M'
         }
         else cout<<RED<<INVERSE<<"H"<<RESET; //otherwise for array value 2 output red
inverse 'H'
      }
      cout<<endl;
   }
   cout<<endl;
   cout<<"Your SHIPS   Ocean="<<CYAN<<"O"<<RESET<<"  Enemy
Miss="<<MAGENTA<<"o"
         <<RESET<<"  Hit Ship="<<RED<<INVERSE<<"H"<<RESET<<"  Ship Position="
         <<BLACK<<"X"<<RESET<<endl; //output map legend
   cout<<"   ABCDEFGHIJKLMNOPQRSTUVWXYZ"<<endl;
   for (int i=0;i<rows;i++){ //loop through rows
      cout<<setw(3)<<left<<i; //output row number
      for (int j=0;j<COLS;j++){ //loop through columns
         if (ship[i][j]==0){ //if array value 0
            cout<<CYAN<<"O"<<RESET; //output cyan 'O'
         }
         else if (ship[i][j]==1){ //if array value 1
```

```
            cout<<BLACK<<"X"<<RESET; //output black 'X'
        } else if (ship[i][j]==2) { //if array value 2
            cout<<RED<<INVERSE<<"H"<<RESET; //output red inverse 'H'
        } else { //otherwise for array value -1
            cout<<MAGENTA<<"o"<<RESET; //output magenta 'o'
        }
      }
      cout<<endl; //output linebreak
   }
   cout<<endl; //output linebreak
}
```