Group 14: Project 1 Documentation

Team Members: Alice Kuang, Thresa Kelly, Minwoo Lee, Justin Sizoo, Maggie Swartz

***Overview:***

The following program implements a game of battleship for two players. Players will agree upon the number of ships they would like to place and then place their ships. Then, the first player will then prompted to take a shot; he will be notified whether a ship was hit, sunk, or missed. If the player hits or sinks a ship, he will be allowed another turn; if not the second player will be allowed to take a shot. The game will proceed by alternating between players until one player sinks all their opponent's ships. This player will be declared the winner.

***Files: (Located in Battleship branch on master branch)***

Main.cpp: Runs the program

Board.h/Board.cpp: performs board initialization and operations such as insertion, shot marking, sink evaluation, win evaluation, and display board.

Executive.h/Executive.cpp: stores game logic, responsible for alternating the players, calling methods to update the board, and validating user input.

**Main**:

To run the program, the player should open the command window and navigate to the folder where the program is stored. They should then use the makefile to compile the program and after compiler write ./Battleship followed by the number of ships they would like to select in the command line (i.e. ./Battleship 3).

If the user does not give two arguments, an error message will remind them of correct format of the program. If the second argument is an invalid number of ships (not between 1 and 5), they will also receive an error message. If the format is correct, main creates an instance of the Executive class and calls the run function.

**Board Class**:

*Board will store the player number, size of the board, row and column information, and three 2-dimensional character arrays (initialGrid, placeGrid, hitGrid). The 2-dimensional arrays will be used to track the initial placement of ships, mark ships hit the opponent hit on the player’s board, and the player’s winning status against his opponent.*

1. **Board constructor**: The board constructor will initialize three 10x10 2-dimensional character arrays. The character ‘0’ will be initialized as the value at each location.
2. **Board deconstructor**: deletes double array at the end of the game.
3. **insertShip method**: accepts the ship size, coordinate location (row and column), and orientation. Performs error checking to make sure ship is within bounds of board and a ship has not been placed at the location. If the ship is able to be inserted, store the character version of the ShipNumber in placeGrid and InitialGrid; return true.
4. **noCollisions method**: noCollisions is a helper method to insertShip which verifies that given a ship size, coordinate location, and orientation, no collisions with other ships will occur if the specified ship is placed.
5. **shootShot method**: accepts the coordinate location (row and column) and the opponent’s board. Using the opponent’s board, call the isHit method to check if there was a hit on the opponent’s board at the given location. If there was a hit, insert an ‘H’ at the location and return true. Else, store an ‘M’ to represent a miss and return false.
6. **isHit method**: accepts the coordinate location (row and column) and checks whether the ship was hit or not on the placeGrid. If the ship was hit (a number between 1 and 5 was stored at the location), insert an ‘S’ on the placeGrid and return true. Else, return false.
7. **sinkStatus method:** accepts the coordinate location (row and column) and returns a bool of whether the ship was sunk or not. This method extracts the value of the shipNumber stored at the location in the initialGrid, and runs a double forloop over the entire board to count the number of times the placeGrid has an ‘S’ that the initialGrid had the value at agiven location . When the count and the element number are the same, the ship is sunk so return true. If the entire board is traversed and the condition was not met, return false.
8. **checkWin method:** traverses the board with a double forloop to check If there is character between ‘1’ and ‘5’ on the placeGrid. If a value between ‘1’ and ‘5’ was found on the placeGrid, then there are still ships to sink so return false. Else, return true.
9. **printBoard method:** takes a string which determines the board type and prints a formatted board showing the row and column names along with the data at each position.

**Executive Class:**

*The executive class stores the boards for the two players, a bool status to track the active player, and an integer for the size of the board.**This class executes the game, alternating turns between the players, prompting ship placement, and validating input.*

1. **Executive constructor:** The executive constructor initializes the board size, two player boards, a variable to alternate the players’ turns, an initial board, and runs the chooseShipLoc to populate both players board with the initial ship placements.
2. **Executive deconstructor:** The executive deconstructor deallocates the memory used by the two player boards.
3. **run method:** The run method regulates all functionalities associated with the gameplay; these include performing input validation, calling board methods such as sinkStatus and checkWin to return a status of the hit, and calling printBoard to display the data, and alternating between players.
4. **chooseShipLoc method:** Given a player’s board and the number of ships they would like to select, chooseShipLoc asks the player to input a ship location and orientation. After the orientation is specified and successful (passing validateLoc and validateDirection successively), the insertShip method is called to insert the ship into the object board. This process until the specified number of ships are inserted; then the status for player turn (pTurn), will be reversed to alternate to the other player’s playing board.
5. **charToInt method:** converts character to integers; because the boardClass is a 2-D character array, all numbers as stored as characters into the board. Conversion of the character to integer allows for better manipulation of data.
6. **validateLoc method:** The validateLoc method takes in a string of placement input and evaluates if it is a valid location. The checks performed including restricting the length of the string to two or three characters, for cases of a digit followed by a character or two digits followed by two characters (i.e. 9a and 10a),if the string is of proper format (i.e. not a10), if the letter characters are between A and J, handling case sensitivity of letters, and the case of entries with only digits (11 or 111).
7. **validateDirection method:** The validateDirection checks to make sure the character input is valid; If the input was not ‘h’ (horizontal) or ‘v’ (vertical), the user will be prompted again for an input, until a valid input is given.