Formatting:

Tasks with a \* should be prioritized since other tasks rely on them

Mark in green if the task has been completed and pushed to github

Mark in red if you are currently working on the task

Mark in yellow if a change has been made to a previously completed task (also note what commit contains the changes)

NOTES:

* des (short for description)
* While writing code, make lots of comments and ask questions whenever needed
* Look at the header file before you implement a method. In the header files, functions that are tabbed under another function, use the functions below them in their definitions. Example:

setUP() // ← don’t write this function

setUpIO() //← until these

shipIO() //← functions are written

* OUR PROGRAM WILL BE STRESS TESTED WHEN GRADED. Make sure the user can’t break the program. Especially when taking input (see below).
* OUR PROGRAM WILL BE GRADED ON HOW USER-FRIENDLY IT IS. When writing IO portions, make things clear and pretty.
* When taking inputs, ensure that invalid input won’t make the program crash. I think the simplest solution to this is to take in a string and then convert the type as needed. Otherwise the program will crash if it’s expecting an int and it gets a char. ADDITIONALLY, trap users in a loop if they give invalid input.

To do list:

1. Check invalid cases for all input.
2. Flip axis back to how they should be, letters on top, numbers on the side. Correct in turnIO, shipIO and draw board methods
3. Update html documentation
4. Cite html generation program
5. Store google drive documents in github
6. Tally accounted hours
7. Finished!

\*ship.h

des: ship conscructor, sets the member variables to the parameters taken in

params: size - length of the ship | orien - the orientation (vertical/horizontal) | Loc - the (x,y) coordinate of the ship's upper-left spot

ship(int size, bool orien, int xLoc, int yLoc)

des: increase the ship hit count by 1

void incrementShipHitCount()

des: sink check

returns: true - ship is sunk | false - ship is not sunk

bool sinkCheck()

int getSize()

bool getOrien()

int getXLoc()

int getYLoc()

\*board.h

All changes to board were done by Kai, in board polish commit

des: board constructor, initialize board values to '.'

board()

des: constructs a string that formats the ship board in a 10x10 grid with labeled axis, this will be used in game.shipIO()

return: the string that formats the board

std::string drawShipBoard()

des: constructs a string that formats both boards in a user friendly manner, this is used during every turn of the game

return: the string that formats the boards

std::string drawFullBoard()

game.h

des: the main game loop

void play()

des: calls the functions that allow the users to set up their boards

void setUp()

des: gets the name of the a player and their number of ships

params: p - the player being modified

void setUpIO(player\* p)

des: allows the user to place their ships

params: p - the player being modified

void shipIO(player\* p)

des: this functions completes a full turn (a full turn happens after both players finish their turn)

void fullTurn()

des: gathers the input for a half turn, sets m\_shotX and m\_shotY based on input, prints full board

params: p - the player being modified

void turnIO(player\* p)

des: after a winner has been determined, print a closing screen stating the winner

void closingScreen()

des: prints 70 "\n"'s to clear the screen

void clearScreen()

des: waits for the next player to press enter then switches players

void switchPlayerPrompt()

//not needed

Des: automatically sets up the players ships and names to static values, only used for testing purposes

\*Void autoSetUp()

player.h

des: player constructor, initializes member variables to input params

params: name - the name of the player i.e. "Kai" | shipCount - how many ships the player wants

\*player(std::string name, int shipCount);

des: checks for valid input, if valid: creates a ship object in the m\_ships array, then places the ship into the shipGrid, does nothing on invalid input

params: the member variables of a ship object

return: true - input is valid and ship was placed | false - invalid input

bool buildAndPlaceShip(int size, bool orien, int xLoc, int yLoc)

des: allows a player to shoot a shot and updates the player's shot grid

params: (x,y) the coordinate being shot at

void playerTurn(int x, int y)

des: checks for valid input

returns: true - valid input | false - invalid input

bool shoot(int x, int y)

des: during player1's turn, this will be called on player2 to update their board and ships, and get information about the shot

params: (x,y) the coordinate being shot at

returns: "\*\*HIT!\*\*", "\*\*Miss\*\*", "Ship of size <s> destroyed!"

std::string updatePlayerShotAt(int x, int y)

des: checks to see if a shot was a hit

params: (x,y) the coordinate being shot at

returns: true - shot was a hit | false - miss

bool hitCheck(int x, int y)

des: if the shot is a hit, updates the ships and

returns: return the ship size if a ship is sunk, returns 0 if no ship is sunk

int updateShips()

des: given a coordinate, find the ship that occupies that coordinate

returns: the ship that occupies the coordinate

ship shipIdentifier(int x, int y)

des: checks if the player has lost i.e. (shipCount == sinkCount)

returns: true - if the player has lost | fasle - if they player has yet to lose

bool loserCheck()

des: returns the full board of the player as a string

std::string printBoard()

std::string getName()