a)

There are many data structures that are used. One data structure that is used is called the boardShip class. This was a class variable that held all of the information for a board, including its name, length, symbol, ID, and points. In board, there was a vector of pointers to boardShips to store all the ships, that way it was easy to allocate and delete boardShips dynamically. The game class had a similar class called ship, however, in this class, the ship did not have points attached with it. Finally in player, the different players were children of the parent class player. This allowed for efficiency in calling functions.

b)

My goodplayer strategy included choosing only 50% of the points, in which none of the points are directly touching eachother. Next, it did a very similar strategy to mediocre player in which the left, right, and top, and bottom of the points were checked(if there was a hit), however, it adds points to a priority set of points to hit, if there is a duplicate in points. This priority point then becomes the next point that gets hit. This algorithm is better because the amount of “random” shots will top off at 50, compared to the possibly many more from mediocre player.

c)

Game non-trivial functions:

Player\* play(Player\* p1, Player\* p2, bool shouldPause = true);

Check if p1 and p2 are playable by calling placeships function

If either of them aren’t return nullptr

Otherwise until one of the boards are destroyed

display the b2 board depending on if p1 is a human

call recommendattack()to get a point

call attack() to actually attack

call recordAttacResult() to record the result

display the board based off the attack

Repeat with b1 and p2

Display who wins!

Board non-trivial functions:

void block();

Check if the blocked points vector size is equal to half the number of row times number of columns.

If not, get a point to block by choosing randomly at 50% probability

void unblock();

Repeatedly until go through whole board  
 Check if point is blocked

If so, iterate and erase the point from vector

bool placeShip(Point topOrLeft, int shipId, Direction dir);

Check if shipID and topOrLeft is valid

Check if the length of the board goes outside of the board from topOrLeft

Gather points of where ship will be by using topOrLeft, dir, and the ship length

Check if the ship points are blocked or already used by another ship

Check if the ID is already used

If above code shows no issues, allow ships to be placed down

Add the shipID to usedID’s vector

Create a new BoardShip

Add the ship points to the BoardShip

Add the symbols to the board

Push the boardShip into the Boardship vector

bool unplaceShip(Point topOrLeft, int shipId, Direction dir);

Check if valid shipID and if ship is actually on the board

delete the ship

Erase the boardship based off the id

Erase the id from the IDs vector

bool attack(Point p, bool& shotHit, bool& shipDestroyed, int& shipId);

Check if a position has been shot before by looking through shotPoints vector

Check in each boardship

The points and compare it to to inputted points

If it is the same point, there is a hit!

Set shot hit = true, set the symbol to ‘X’, and decrement theBoardShip of that point’s health by 1

If the health is equal to zero

Set the ship to destroyed, and set the shipID to the dead ID

If it isn’t a hit then set the symbol to ‘o’

push the point p into shotpoints

bool allShipsDestroyed() const;

Check if the vector of boardships is not mpty

If so, return false, otherwise return true

Player non-trivial functions:

HumanPlayer

bool placeShips(Board& b);

Repeatedly until the counter equal the size

Asks for direction and checks if valid, if not asks for it again

Asks for point and converts it from string to a point(see recommendAttack() on how it is implemented)

If not valid, asks for another point

Displays the board

Point recommendAttack();

Repeatedly until a correct point is given

If it isn’t integers or empty, ask for points again

Get the firstCharacter

Check for a space

Get the SecondCharacter

Convert the first and second character to ints

Return the point

MediocrePlayer

int placeShipsRecursion(Board &b, int ships, int& counter, vector<Point> theShipCoords);

//This is a helper function for placeShips

Check if the shipID = -1

return counter+50

Check if the shipID is greater than the num of ships

return counter

Check if the counter is greater than or equal to 50

return counter

Check ships have been placed yet

If yes, and the shipID is zero

unplace the ship, and increment counter

If yes, and the shipID is not zero

Get the cords of the ship

erase the coords

unplace the ship

Repeatedly until ship is placed or couldn’t place ship

Increment the row/column to next one

If reached the end, switch to vertical

If reached the end and is vertical, say couldn’t place ship

If we couldn’t place the ship

the return value = placeShipsRecursion with shipID -1

If we could place the ship

Add the top or left coords and directions to vectors

the return value = placeShipsRecursion with shipID + 1

Return the return value

Point MediocrePlayer::recommendAttack()

Check if inStateOne

If so pick a random point and check if it is used yet

Add to the usedpoints vector

Return the point

If not(must be in stateTwo)

Repeatedly until got a valid point to return

Check the pointsToHit array, and save it as point p

Erase the point from the array

Check if the point is used already

If so, choose another point

pushback new point into array

void MediocrePlayer::recordAttackResult(Point p, bool validShot, bool shotHit, bool shipDestroyed, int shipId)

Check if the shot is valid

If so, add to the usedPoints vector

Check if shothit

If yes

For each of the four direction, add the next four points in the direction to the pointsToHit vector.

Check if in State One and if the ship is destroyed

If yes, stay in stateOne

If not, check if the ship is destroyed

If yes, add to destroyedShips vector, and pop the points to hit to empty., and set back to stateOne.

GoodPlayer

vector<Point> GoodPlayer::combinedRowsAndCols(vector<int> rows, vector<int> cols)

Combines to vectors of ints into vectors of points

bool GoodPlayer::placeShipsRecursion(Board &b, int shipID, int zeroeth, bool& submarinePlaced)

Places the ships in specific places, which have been found to be the best possible board arrangement against the mediocre player

bool GoodPlayer::placeShips(Board &b)

Calls placeShipsRecursion 5 times for the 5 boats

Point GoodPlayer::recommendAttack()

Same as recommendAttack of mediocre player

Has specific points from beforehand

Uses those points before going random

void GoodPlayer::recordAttackResult(Point p, bool validShot, bool shotHit, bool shipDestroyed, int shipId)

Same as mediocre player