**Ying Chou :D**

**CS 32: Project 3**

**Design of Data Structures:**

* Board representation: a 2D character array
* Probability board representation: a 2D integer array
* Ships placed on the board: an int vector, with each shipID corresponding to its index in the vector
* Ships in the game: a ship vector that stored every ship added
  + Ship is a struct I made, containing length, symbol, and name variables
* Positions that have already been attacked: a Point vector that stores every point attacked
* Positions in the set of valid points after mediocre player enters state 2: a Point vector generated after the mediocre player enters state 2
* Positions to target after good player enters state 2: a Point stack that adds points every time a new segment is hit

**Good Player Strategies:**

* For placing ships, the ships are mostly placed onto the board randomly. But after each ship is placed, two of its surrounding points will be blocked off, in an attempt to spread the ships out a bit more, to hopefully thwart mediocre player’s cross strategy
* For attacking, the good player is either in the hunt state or the target state. If in hunt state, it will pick points on the board with the highest probability of containing a ship, derived from a probability density map. This map updates with every move that is made. If in the target state, it will pick a point out of the 4 adjacent points to the ship just hit. If another segment is hit again and no ship is sunk yet, the adjacent points for that segment will also be added to the valid points to attack

**Pseudocode:**

BoardImpl::block()

for half of the total points on board

find a random point to block that has not been blocked yet

block that point

BoardImpl::placeShip(Point topOrLeft, int shipId, Direction dir)

if shipId or topOrLeft is invalid

return false

for both horizontal and vertical ships

check if ship will be partially off the board or overlap another ship

return false

check if the ship has already been placed

return false

if not, place ship

add shipId to placedShips vector

return true

BoardImpl::unplaceShip(Point topOrLeft, int shipId, Direction dir)

if shipId or topOrLeft is invalid

return false

for both horizontal and vertical ships

check if ship will be partially off the board

return false

check if the ship is at the location indicated

return false

if not, unplace ship

remove shipId from the placedShips vector

return true

BoardImpl::display(bool shotsOnly)

print top row of numbers

looping through the entire board array

if shotsOnly = true

if it’s a shot position

print

print ‘.’

if shotsOnly = false

print

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

if p not on the board or already been hit

return false

if p is a point with no ship

record p as hit

return true

if ship was hit

record p as hit

check if there are more of that symbol on the board

if yes, ship is destroyed

if ship is destroyed

find its id

return true

GameImpl::play(Player\* p1, Player\* p2, Board& b1, Board& b2, bool shouldPause)

if ships cannot be placed

return null

repeatedly until one player has no ships left

display board

attack a position with recommendAttack()

if shot invlalid

cout corresponding line

record the result

if ship destroyed/if ship is hit but not destroyed/no ship was hit

cout corresponding line

record the result

display board

check if there are still ships left

if not, player wins

return player

(repeat for second player)

HumanPlayer::placeShips(Board& b)

for all the ships in the game

display board

until valid direction is entered

cout prompt

if valid, set values

if not, cout and break

until valid point is entered

cout prompt

if valid, place ship

if not, cout and break

return true

MediocrePlayer::recursivePlace(Board& b, int shipID)

if shipID is at the last ship of the list (base case)

return true

loop through the entire board

if ship can be placed either horizontally or vertically

if next ship (shipID + 1) can also be placed (recursive call)

return true

else, unplace the original ship and try the next point

if no points work, return false

MediocrePlayer::placeShips(Board& b)

for 50 iterations

block half the board

if recursive place returns true

unblock the board

return true

if not, unblock board and try again

return false if unsuccessful after 50 tries

MediocrePlayer::recommendAttack()  
 if in state 1

until a position that hasn’t been attacked before is found

pick a random point and check if it has been attacked yet

return found point

if in state 2

until a position that hasn’t been attacked before is found

pick a random point from the pool of valid points

remove that point from the pool of valid points

check if it has been attacked yet

return found point

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

add p to list of already attacked points

if in state 1

if ship was hit but not destroyed

switch to state 2

generate pool of valid points to hit

if in state 2

if pool of valid points to empty

switch back to state 1

if ship is destroyed

switch back to state 1

clear the pool of valid points

MediocrePlayer::isIn(vector<Point> a, Point x)

loops through the vector

if x is in the vector

return true

return false otherwise

MediocrePlayer::validPos(Point p)

generates at most 16 points in a cross from the original point

if generated point is valid

add to the pool of valid points

return the pool of valid points

GoodPlayer::placeShips(Board &b)

until all ships are placed

for each ship

until the ship is placed

generate random point

check if point is not allowed

if yes, break

if ship can be place either horizontally or vertically

ship is placed

push two of the surrounding points onto a vector that keeps track of not allowed points

return true

GoodPlayer::recommendAttack()

if in state 1

calculate the cell probabilities

return point with the highest probability

if in state 2

until it finds a non-hit point or the stack is empty

take the top point off the stack

if point has already been hit

pop off the stack and find new point

if not, pop off the stack and return the point

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

add p to the list of already attacked elements

if in state 1

if ship was hit and not destroyed

change to state 2

generate the 4 adjacent points to push onto the stack

if in state 2

if ship was hit and not destroyed

add the new adjacent points for this hit onto the stack

if stack is empty

change back to state 1

if ship was hit and destroyed

change back to state 1

clear out the stack

GoodPlayer::validPos(Point p)

creates the 4 adjacent points to p

checks if each point is valid

push onto stack

GoodPlayer::calculateProbability()

until every ship has been checked

first try to place the ship horizontally by looping through the board

if ship can be placed

probability value will be incremented by 1 for every point the ship covers

then try to place the ship vertically by looping through the board

if the ship can be placed

probability value will be incremented by 1 for every point the ship covers

GoodPlayer::highestProb()

loop through the entire probability board

compare each point with the highest probability found so far

if the point has yet to be attacked, set point as highest so far

return highest point found