Project 3

Nicole Ju

DATA STRUCTURES

To store ship information in the game class, I created a struct called “ship” to store each ship’s length, symbol, and name. I then created a vector to store all of the game’s ship objects. The order that the ships are added into this vector correspond to the ship id, so vector[0] would refer to the ship with an id of 0.

In the board class, I represented the board using a 2x2 grid of characters. I also stored the shipIds in a vector to reference them in the member functions.

For HumanPlayer, MediocrePlayer, and GoodPlayer, I stored my already chosen points in a vector to make sure I wouldn’t recommend a point that has been chosen before. For my GoodPlayer, I also stored points in a “checkerboard” in a vector so I could choose from those points first.

GOODPLAYER STRATEGIES

placeShip:

To place the ships, I created sections in the board and placed the first four ships in each section so that the ships would be somewhat spread out. For the remaining ships, I placed them randomly on the board. To make sure it didn’t crash, I exited each of my while loops after 50 tries, and if that happened, I switched over to MediocrePlayer’s placeShips algorithm.

recommendAttack and recordAttack:

I had 3 different states. In state 1, the player can attack anywhere in a “checkerboard” (which includes every other spot on the board). The reason for this is because in standard battleship, no ship has a length of less than 2. After hitting a ship, the player is now in state 2. It will first target the spots adjacent to the first hit (one spot away horizontally and vertically). After another hit is made, the direction of the attack is saved, and the player is now in state 3. The player will attack in that direction first. If it gets to a point where there is no ship left in that direction, it returns to the original hit and goes in the opposite direction (e.g. if it went up before, now it goes down). If all the points adjacent to the first hit have been chosen already, we widen the shooting range to a cross with a length of 2, then 3, then 4.

GAME

**bool** GameImpl::addShip(**int** length, **char** symbol, string name)

//the Game class implements the required restrained for this function call to work

create a ship object with the specified length, symbol, and name

add the ship to the vector of ships

increment the number of ships

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

if placeShips returns false for either player

return nullptr

while each player still has ships left

display player two’s board (don’t display ships if player one is human)

call player one’s recommendAttack function and store the resulting Point

make the attack on player two’s board

have player one record the result of this attack

create a string for the result

if the attack missed

set result to “missed”

if the attack hit a ship

set result to “destroyed the…” if it destroyed the ship and “hit something” otherwise

cout print statement and display the result

if player one won

display result and board if the losing player was human

return pointer to player one

if shouldPause

wait for user to hit enter

repeat with the roles of player one and two reversed

BOARD

**void** BoardImpl::block()

find half the number of total spots on the board

while the number of blocked spots < half the total spots

find a random r and c

if this spot on the grid is ‘-’

continue

change the spot to ‘-’ and increment the number of blocked spots

**void** BoardImpl::unblock()

for each spot on the grid

if it equals ‘-’

change it to unblocked (‘.’)

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

if ship id is invalid

return false

if dir is horizontal

if leftmost point or rightmost point is invalid

return false

if any part of the ship overlaps with a ship or blocked position

return false

repeat for vertical (checking highest and lowest points)

if the ship’s id is in the shipId vector

return false

find the ship’s symbol and update the grid accordingly

add the ship to the shipId vector

return true

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

if ship id invalid

return false

find the symbol and length of the ship you are trying to unplace

if the ship is not in bounds

return false

for each spot on the grid where the ship should be

if the spot is not equal to the ship symbol

return false

for each spot on the grid where the ship is

change that spot to ‘.’

create an iterator and set it to point to the first element in the shipId vector

iterate through the shipId vector

if the element is equal to the ship id passed in

erase it from the vector

return true

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

set shipDestroyed and shotHit to false

if the point is invalid

return false

if the grid contains ‘X’ or ‘o’ at that point

return false

if the spot on the grid is not an ‘X’, ‘o’, or ‘,’ (it has to be a ship symbol)

set shotHit to true

iterate through the ship id vector

create a bool symbolFound and set to false

if the symbol corresponding to the ship id is found on the board

symbolFound is true

if symbolFound is false

set shipDestroyed to true

set shipId to the index of the ship id vector

erase the shipId from the vector

else if the spot on the grid is ‘.’

set shotHit to false

change the spot to ‘o’

return true

HUMANPLAYER

**bool** HumanPlayer::placeShips(Board& b)

display the empty board

for each of the ships in the game

prompt user for direction of the ship of that length

while the direction entered is not ‘h’ or ‘v’

prompt user for direction

store direction

while the ship cannot fit in that direction

prompt user for direction

prompt user for row and column to place the ship

while the getLineWithTwoIntegers function returns false

prompt user for row and column

while the ship cannot be placed at the entered location

prompt user for row and column

while getLineWithTwoIntegers function returns false

prompt user for row and column

display the board

return true

Point HumanPlayer::recommendAttack()

prompt user for point to attack

while getLineWithTwoIntegers returns false

prompt user for point

return point

MEDIOCREPLAYER

helper function for placeShips

**bool** putship(**const** Game& g, Board& b, **int** ship)

if ship is greater than the number of ships

return true

for each spot on the board

if you can place the ship on the board horizontally

if putship for the next ship returns true

return true

if you can place the ship on the board vertically

if putship for the next ship returns true

return true

if shipid is zero

return false

then backtrack:

for each spot on the board

call the unplaceShip function

return false

**bool** MediocrePlayer::placeShips(Board& b)

if the number of ships is zero

return false

for 50 times

block half the board

if the putship function returns true (pass in ship id 0)

unblock the board

return true

unblock the board

return false

**bool** MediocrePlayer::allChosen() //checks if all the points in the cross have been chosen

for 4 times

if the point above is valid

check if it is in the chosen points vector

if it has not been chosen

return false

else

break

for 4 times

if the point above is valid

check if it is in the chosen points vector

if it has not been chosen

return false

else

break

repeat for horizontal directions

Point MediocrePlayer::recommendAttack()

if the state is 1 or the state is 2 and all the points in the cross have been chosen

set state to 1

while the chosen spot has already been chosen

choose a random spot on the board

set p equal to this point

else if state is 2

while the chosen spot has already been chosen

randomly choose the number of spots to move

if you randomly choose vertical

set the point to the position that many spots above or below the target point

if you randomly choose horizontal

set the point to the position that many spots left or right of the target point

if the point is invalid

continue

check if the point has already been chosen

push the point onto the chosen points vector

return the point

**void** MediocrePlayer::recordAttackResult(Point p, **bool** validShot, **bool** shotHit,

**bool** shipDestroyed, **int** shipId)

if the state is 1

if the shot was valid and it hit something but didn’t destroy the ship

set state to 2

set target to the point passed in

else if state is 2

if the shot was valid and it hit something and destroyed the ship

set state back to 1

GOODPLAYER

**bool** GoodPlayer::placeShips(Board& b)

if the number of ships is <= 5 and the rows and columns are at least 10

for 4 times

if the index equals the number of ships

break

randomly choose the direction

while the ship cannot be placed in that point

based on the index

set the row and column to randomly chosen point in a different corner of the board

if there are less than 4 ships

return true

find a random point on the board for the fifth ship

randomly choose a direction

while the ship cannot be placed at that point

choose another row and column

if all of these while loops run less than 50 times

return true

unplace all the ships(something went wrong above)

use MediocrePlayer’s placeShips recursive algorithm

**bool** GoodPlayer::allChosenLine()

if the direction is vertical

check the spots above and below newtarget and see if they have been chosen

if not, return false

if the direction is horizontal

check the spots right and left of newtarget and see if they have been chosen

if not, return false

return true

**bool** GoodPlayer::allChosen(**int** x)

similar to allChosenLine but widens to a cross with size x(1,2,3). evaluates on target instead of newtarget

Point GoodPlayer::switchToCross(**int** x)

exactly the same as the state 2 algorithm in MediocrePlayer’s recommend attack, except allows for different sized crosses (rather than 4)

finds a point in the cross

while the point is already chosen

find another point in the cross

add the chosen point to the chosen vector

return the point

Point GoodPlayer::recommendAttack()

if state is 1 or state is 2/3 and all the points in the 4 cross have been chosen

if the checkerboard vector is empty

while the point has already been chosen

choose a random point on the board

check if the point is in the chosen vector

else

while the point has already been chosen

save the front of the checkerboard vector

check if that point is in the chosen vector

if it has already been chosen

erase that point from the checkerboard vector

else

push that point onto the chosen vector

erase that point from the checkerboard vector

return the point

else if state is 2

if all the spots in the 3 cross have been chosen

return a spot in the 4 cross

else if all the spots in the 2 cross have been chosen

return a spot in the 3 cross

else if all the spots in the 1 cross have been chosen

return a spot in the 2 cross

otherwise

while the point has already been chosen

choose horizontal or vertical randomly

choose up/down/left/right randomly

if that point is invalid

continue

check if that point is in the chosen points vector

else if state is 3

if all the spots next to newtarget (in the line) have been chosen

set newtarget to the original target

if all the spots in the line have still all been chosen

change directions

if all the spots in the line have still all been chosen

if all the spots in the 2 cross have not been chosen

return a spot in the 2 cross

else if all the spots in the 3 cross have not been chosen

return a spot in the 3 cross

else if all the spots in the 4 cross have not been chosen

return a spot in the 4 cross

else

while the point has already been chosen

if the attacking direction is vertical

choose a point up or down from newtarget

if the attacking direction is horizontal

choose a point left or right of newtarget

if the point is invalid

continue

check if the point is in the chosen points vector

push the point into the chosen points vector

return the point

**void** GoodPlayer::recordAttackResult(Point p, **bool** validShot, **bool** shotHit,

**bool** shipDestroyed, **int** shipId)

if state is 1

if it was a valid shot and it hit a ship but didn’t destroy it

set state to 2

set target and newtarget to p

else if state is 2

if it was a valid shot and it hit a ship but didn’t destroy it

set state to 3

set newtarget to p

set the attacking direction based on the points comparison with target

else if it destroyed the ship

set state to 1

else if state is 3

if it was a valid shot and hit and destroyed a ship

set state to 1

if it was a valid shot and hit a ship but didn’t destroy it

set newtarget to p