Project 3 Report

1. The board was created using a two-dimensional array of characters, with ‘.’ representing a space with open water, an ‘X’ marking a hit shot and an ‘o’ marking a missed shot. To keep track of shots that were hit, GoodPlayer used its own two-dimensional char array, filled with ‘n’ by default, with ‘y’ marking the points that were hit. MediocrePlayer used a vector list of Point objects to keep track of the points that were hit.
2. The GoodPlayer’s Placeships function calls an auxiliary function recusivelyPlaceShips that uses a recursive algorithm to place ships on random points on the board in directions that would fit each ship. If placing the ship fails, the function backtracks and tries to place the ship in a different random point again to ensure that all ships are properly placed on the board.

When recommending moves, the GoodPlayer uses a two-state algorithm similar to the one MediocrePlayer uses, but improves on the order in which points are selected to be shot. During the first five turns, the GoodPlayer attacks the middle of the board, along with four points, one in each quadrant of the board, equally spaced. Once the first five turns are over, if the player is in state 1, then recommend attack will return a randomly generated point in a checkerboard pattern, rather than just any randomly generated point. The result of every attack is recorded in a two-dimensional char array and used to determine the next course of action based on the player’s state.

**Pseudocode**

bool MediocrePlayer::placeShips(Board& b)

{

Push back ships

Trying 50 times…

Block half of the board

recursivelyPlaceShips

unblock

if successful

return true

if unsuccessful

return false

}

bool MediocrePlayer::recursivelyPlaceShips(Board& b, vector<int> &shipsToPlace, int index) {

//base case…

if the index equals number of ships

return true;

//start here...

Get the ship’s length at the first index

while placing the ship is unsuccessful…

tryToPlace:

if (ship can be placed vertically here)

if (placeRecursively can place all remaining

ships with current ship being vertical)

return true; //all ships were placed

if you get here, unplace the current ship;

if (ship can be placed horizontally here)

if (placeRecursively can place all remaining

ships with current ship being horizontal)

return true; //all ships were placed

if you get here, unplace the current ship;

if you get here, return false;

}

Point MediocrePlayer::recommendAttack()

{

if the state is 2 and there’s nothing left to hit

state is 1;

if state is 1

try hitting a random point

if state is 2

hit a point in state2Targets

}

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

{

if attack from state 1 was successful, but ship not destroyed

if possible

mark 4 points to the right of p as targets

mark 4 points to the left of p as targets

mark 4 points above p as targets

mark 4 points below p as targets

switch to state 2;

if attack failed

stay in current state

if ship was destroyed

clear state2Targets

switch to state 1

}

bool GoodPlayer::placeShips(Board& b)

{

Make a vector of all the ships

Place them recursively and return true if successful, false if not.

}

bool GoodPlayer::recursivelyPlaceShips(Board & b, vector<int> &shipsToPlace, int index)

{

if index equals number of ships

return true;

tryPlacing:

get a random point

if the ships fits there vertically

if the other ships can be placed recursively

return true;

unplace the ship

else

if the ship fits there horizontally

if the other ships can be placed recursively

return true;

unplace the shi

else

try again

return false if reached this point

}

Point GoodPlayer::avoidWaste()

{

repeatedly

pick random point on board

if row is even and column is odd

add 1 to row

else if row is odd and column is even

add 1 to column

if point has not been attacked

mark

return point

}

Point GoodPlayer::recommendAttack()

{

if state is 1

if turn is 1

hit middle

if something was hit

mark it

if turn is 2

hit first quadrant

if something was hit

mark it

if turn is 3

hit second quadrant

if something was hit

mark it

if turn is 4

hit third quadrant

if something was hit

mark it

if turn is 5

hit fourth quadrant

if something was hit

mark it

Hit a random checkerboard point

avoidWaste

if state is 2

while there’s something to check

check the top

pop the top

mark if hit

avoidWaste

}

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

bool shotHit, bool shipDestroyed,

int shipId)

{

if shot was successful but ship not destroyed

add on point left, right, up, and down from point to stack

switch to state 2

if stack is empty

switch to state 1

if shot was not successful

stay in current state

if ship destroyed

if stack is not empty

switch to state 2

else

switch to state 1

}