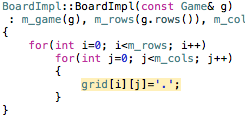
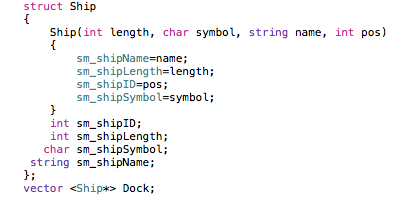
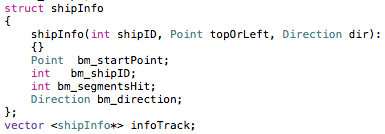
a.

I created the board using a two-dimensional array of characters. All elements of the array are set to a dot character by the constructor of the board. Macintosh HD:Users:Gageek:Desktop:Screen Shot 2013-05-23 at 7.25.45 AM.png

Game keeps track of new ships that are added with a Ship struct and a vector of pointers to Ships called “Dock”.



When a ship is placed onto the Board, a pointer to a new “ShipInfo” struct (which holds the ship’s ID, start point, direction, and number of damaged segments) is pushed back onto a vector called “InfoTrack”. When a ship is hit, that ship’s Id is matched with it’s infoTrack placement, and segmentsHit is incremented. When segmentsHit equals that ship’s length, we know it has been destroyed.



Since players do not have access to the board’s elements, MediocrePlayer and GoodPlayer have their own two dimensional array of characters (same size as the board) to keep track of shots made. The constructor of each sets all elements of the array to ‘n’. When the MediocrePlayer recommends an attack at a point, that point on the array is changed to ‘y’.

b.

**GoodPlayer**

The GoodPlayer’s PlaceShips function calls block to block out half of the spots on the board then calls the recursive helper function to fit the ships into the limited board. The “PlaceRecursively” function is given the shipID and reference to the board to place it on. Starting from the bottom right corner of the board and moving to the top right, the function attempts to place the ship vertically then if unsuccesful attempts to place it horizontally.

If a ship is successfully placed, the function calls itself for the next shipID. If the recursive call to the function returns true, then it means all ships were successfully placed and the base case was reached. The base case is when shipID is equal to the number of ships; this does not happen unless all ships were placed. After placing all ships succesfully, placeShips unblocks the board and returns true. If the recursive function fails to place the ships, placeShips returns false.

c.

**Pseudocode**

MediocrePlayer Ship Placement

bool MediocrePlayer::placeShips(Board &b)

{

block half of the board

call placeRecursively;

if(it fails)

unblock the board;

return false;

unblock the board;

return true;

}

bool MediocrePlayer::placeRecursively(Board &b, int shipID)

{

if (shipID is equal to number of ships)

return true; //the final ship’s ShipID is one less

//than the number of ships

beginning at top right and stopping at bottom right

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;

}

MediocrePlayer Attack!

Point MediocrePlayer::recommendAttack()

{

if in state 1

repeatedly

pick a random point on the board

if point has not been attacked

mark as attacked;

return point;

if in state 2

search for points marked as targets

mark as attacked;

return point;

}

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 5 points to the right of p as targets

mark 5 points to the left of p as targets

mark 5 points above p as targets

mark 5 points below p as targets

switch to state 2;

if attack failed

stay in current state

if ship was destroyed

remove unattacked targets from board

switch to state 1

}

GoodPlayer Ship Placement

bool GoodPlayer::placeShips(Board &b)

{

block half of the board

call placeRecursively;

if(it fails)

unblock the board;

return false;

unblock the board;

return true;

}

bool GoodPlayer::placeRecursively(Board &b, int shipID)

{

if (shipID is equal to number of ships)

return true; //the final ship’s ShipID is one less

//than the number of ships

beginning at bottom right and stopping at top left

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;

}

GoodPlayer Attack

Point GoodPlayer::recommendAttack()

{

if in state 1

if this is the first turn

attack middle of board

mark as attacked

increment turn

return point

repeatedly

pick random point on board

if row is even and column is odd

add one to row

else if row is odd and column is even

add one to column

if point has not been attacked

mark as attacked

return point

if you get here

call avoidWaste() function

if in state 2

while stack is not empty

pop off point

if point has not been attacked

mark as attacked

return point

if you get here

call avoidWaste() function

}

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

}

Point GoodPlayer::avoidWaste()

{

repeatedly

pick random point on board

if row is even and column is odd

add one to row

else if row is odd and column is even

add one to column

if point has not been attacked

mark as attacked

return point

}