Anderson Truong

205676321

CS32 Spring 2022

CS32 Project 3 Report

**Data Structures**

**Game Board:**

The Board class contains a private two-dimensional char array of size MAXROWS by MAXCOLS. The array is used to represent the ships and attacked positions on a player’s board.

When a shot is made, the array at the shot position is updated to be ‘o’ if the shot misses and ‘X’ when it hits.

**ShipType in GameImpl:**

The ShipType struct (defined in “utility.h”) holds a ship type’s length (int), symbol (char), and name (string).

Every GameImpl object has a private ShipType vector that stores unique ShipTypes based on name and symbol. Unique ShipTypes are enforced in GameImpl::addShip(). GameImpl’s methods can query the vector to access a ShipType’s members based on its ID. A ShipType’s ID in the vector is its index.

**ShipInstance in Board:**

The ShipInstance struct (defined in “Board.cpp”) holds a ship’s ID, top/left position, and direction spanning from the top/left position.

Every BoardImpl object has a private ShipInstance vector that stores every placed ship’s ID, position, and direction. This allows for BoardImpl to know which grid blocks a ship is on.

**GoodPlayer Strategies**

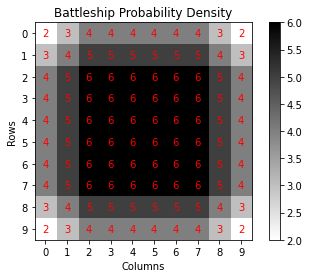
For ship placement, the GoodPlayer simply places ships at random points.

For attacking, the GoodPlayer uses a probabilistic hunt and target strategy. The player can operate in a hunting mode (picking a random position that has the highest probability of a ship) or a targeting mode (picking a position in the crosshairs of a previously hit position). These strategies achieve around a 95% win rate against the MediocrePlayer.

**Hunting with probability and parity:**

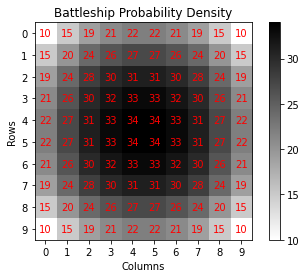
The player produces a two-dimensional int array of probabilities based on previously missed attack positions. The array is populated during each GoodPlayer::recommendAttack() call. Blocks in the crosshairs of a missed position have a lower probability.

The total probability array is created from a superposition of every ship’s probability array. For example, here is the initial probability density heatmap for a ship of length 3 (visualized in Python):



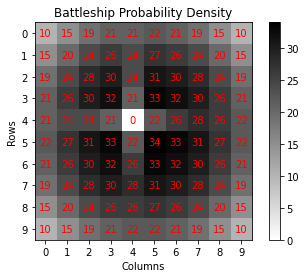
The probabilities are calculated by attempting to place a ship length at a certain point in different configurations. For example, in the corner, there are two possible configurations for ship placement: horizontal with the leftmost point in the corner, and vertical with the topmost point in the corner.

Here is the superposition of the initial probability densities for the standard ship lengths of 5, 4, 3, 3, 2:



The points nearest to the center have the highest probability of a ship due to the higher number of possible configurations.

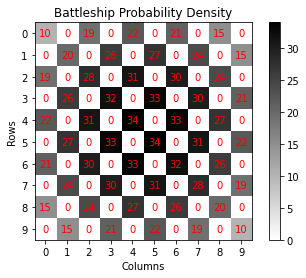
The GoodPlayer attacks a point with the highest probability (one of the four points with 34 probability). In the case of a miss, the array is updated accordingly:



Points in the crosshair of a missed shot now have a lower probability of a ship.

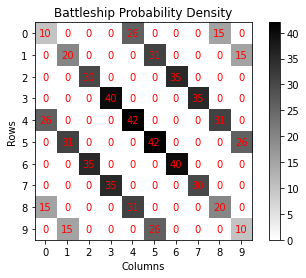
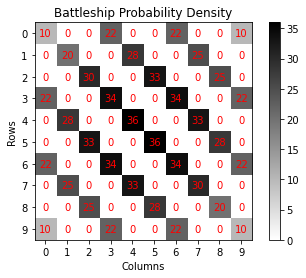
To further optimize point selection, the GoodPlayer performs a parity selection to remove unnecessary choices. First, it determines the smallest ship length (N) out of all undestroyed ships. Then, it only keeps every other N points. This narrows down the selection of points without losing much information.

Parity with smallest length 2:



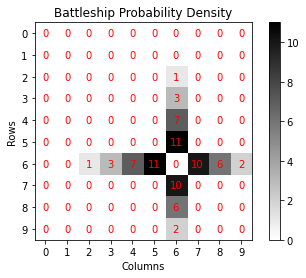
Every ship of length 2 or greater is guaranteed to have a point on every other 2 points.

As the smallest ship length increases (as the smallest ships are destroyed), the parity becomes more aggressive in its selection. Here are the parity-adjusted arrays with smallest lengths 3 and 4:

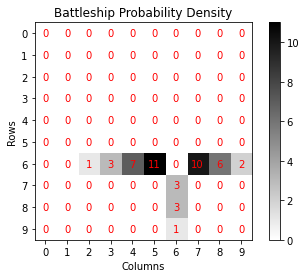


**Targeting with probability and direction/proximity weights:**

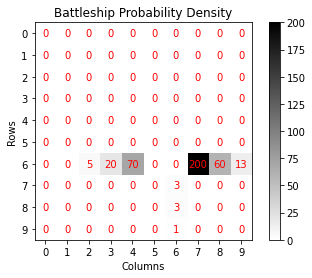
Once a ship is hit, the GoodPlayer switches to targeting mode. This mode only calculates the probabilities for the positions in the crosshair of the point that caused the mode switch. The calculation process is the same as the Hunting mode, causing points closer to the targeted point to have higher probability.



If a shot misses on one of the branches, the probability for the rest of the points along that branch is zero, effectively pruning the branch. Here is the crosshair when a shot misses:



Once a second point is a hit, the targeting mode determines the direction of the ship based on the line created between the targeted point and the second point. Points along this line have their weights doubled, effectively ignoring points along the other perpendicular. The point weights are further increased based on their proximity to the targeted point with the function f(x) = 10/x, where x is the distance.



Once a ship is destroyed, the GoodPlayer moves all the points between the last destroyed point and the targeted point to the list of missed points, so as to not target them again. However, there may still be points left in the hit but not destroyed list of points. If this is the case, then the GoodPlayer remains in targeting mode and begins to target those points in the same process as above. If the list is empty, then it switches back to hunting mode.

**Non-Trivial Algorithms**

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

for ship type in all ship types:

if name matches ship type’s name

return false

push back new ship to all ship types

return true

bool GameImpl::playerAttack(Player\* attacker, Player\* attacked, Board& attackedBoard, bool shouldPause)

if attacker is a human

print out attacked board with shots only

else

print out attacked board

get recommended point from attacker

attack the attacked player’s board at recommended point

attacker records result of attack

attacked records result of attack

if attack was valid

print: attacked

if shot hit a ship

if ship was destroyed

print: destroyed ship

else

print: hit something

else

print out shot missed

if attacker is human

print out attacked board with shots only

else

print out attacked board

else

print out: wasted a shot

if all ships destroyed on attacked board

return true

if shouldPause

wait for enter key

return false

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

if p1 or p2 cannot place ships

return nullptr

loop indefinitely

if p1 attacks and destroys all p2’s ships

print: p1 wins

return p1

if p2 attacks and destroys all p1’s ships

print: p2 wins

return p2

return nullptr

void BoardImpl::clear()

for row in max rows

for col in max cols

set the board at row, col to ‘.’

void BoardImpl::block()

create vector of blocked points

block count = number of cells in half the board

loop while block count is not zero

create point at random row and column

if point already blocked

continue

block out point on board

decrement block count

void BoardImpl::unblock()

for row in max rows

for col in max cols

if point is blocked

unblock point

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

if shipId not in 0 to nShips - 1

return false

for every placed ship

if ship has same ID as placed ship

return false

current point starts at topOrLeft

loop up to ship length times

if current point is invalid

return false

if current point is already occupied

return false

if dir is horizontal

increment current point column

if dir is vertical

increment current point row

push back ship in placed ships

current point starts at topOrLeft

loop up to ship length times

set board at current point to ship’s symbol

if dir is horizontal

increment current point column

if dir is vertical

increment current point row

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

if shipId, topOrLeft, and dir do not match any placed ship

return false

current point starts at topOrLeft

loop up to ship length times

set board at current point to empty ‘.’

if dir is horizontal

increment column

if dir is vertical

increment vertical

erase ship from placed ships

return true

void BoardImpl::display(bool shotsOnly)

print out column indices

for row in max rows

print row index

for col in max cols

if shots only and point is a ship symbol

print empty ‘.’

else

print board at row, col

bool BoardImpl::shipInstanceDestroyed(const ShipInstance& instance)

current point starts at ship instance’s topOrLeft point

loop up to ship length times

if board at current point matches ship symbol

return false

if ship direction is horizontal

increment current point column

if ship direction is vertical

increment current point row

return true

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

if p is invalid or p is already attacked (‘X’ or ‘o’)

shotHit = false

shipDestroyed = false

shipId = -1

return false

if p doesn’t hit a ship

set board at p to missed ‘o’

shotHit = false

shipDestroyed = false

shipId = -1

return true

create matchedShip

loop through all ships on board

if symbol at p matches ship’s symbol

matchedShip = ship

Break

mark board at p as a hit ‘X’

if hit destroyed a ship

shipDestroyed = true

else

shipDestroyed = false

shotHit = true

shipId = matchedShip ID

return true

bool BoardImpl::allShipsDestroyed()

loop through all ships

if ship is not destroyed

return false

return true

bool HumanPlayer::placeShips(Board& b)

loop up to number of ships times

display player’s board

do

get direction from user input

until valid direction

do

get row and column from user input

until valid row and column

place ship at row and column in direction

return true

Point HumanPlayer::recommendAttack()

do

get row and column

until row and column is valid

return point with row and column

bool MediocrePlayer::recursivePlace(Board& b, int shipId)

if shipId equals number of ships in game

return true

for point (row, col) on board

if able to place ship horizontally or vertically

if able to recursivePlace next ship with shipId + 1

return true

else

unplace ship in both directions

return false

bool MediocrePlayer::placeShips(Board& b)

loop up to 50 times

block board

if able to recursivePlace starting at shipId 0

unblock board

return true

unblock board

return false

bool MediocrePlayer::pointNotChosen(Point& p)

for point in previously attacked points

if p matches point

return false

return true

Point MediocrePlayer::recommendAttack()

if in move state 1

loop indefinitely

generate random point

if point not chosen

push back point to previously attacked points

return point

if in move state 2

loop through all ships

if ship length is greater than or equal to 6

set move state to 1

return call to recommendAttack()

create vector of possible points

loop from -4 to 4 with i

create point at transitionPoint with same row + i

if point is valid and not chosen

push back points to possible points

create point at transitionPoint with same col + i

if point is valid and not chosen

push back points to possible points

pick random point from possible points

push back point to previous attacks

return point

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

if move state is 1 and the shot hit but didn’t destroy a ship

set move state to 2

store Point p as the transition point

if move state is 2 and the shot destroyed a ship

set move state to 1

bool GoodPlayer::recursivePlace(Board& b, int shipId)

if shipId equals the number of ships in the game

return true

loop up to 50 times

pick a random point on the grid

pick a random direction (vertical or horizontal)

if able to place ship at point in direction

if able to recursivePlace next ship with ID shipId + 1

return true

else

unplace ship

return false

bool GoodPlayer::recursivePlace(Board& b, int shipId)

if shipId equals number of ships in game

return true

loop up to 50 times

pick a random point on the boad

pick a random direction (vertical or horizontal)

if able to place ship at point in direction

if able to recursivePlace next ship with ID shipId + 1

return true

else

unplace ship

return false

bool GoodPlayer::validPoint(Point p)

if point is out of bounds

return false

loop through all missed attacks

if point matches missed attack point

return false

return true

bool GoodPlayer::validPlace(Point p, int shipLength, Direction dir)

if point is not validPoint

return false

if dir is vertical

loop with i from point’s row to point’s row + ship length

if point at (i, point’s col) is not validPoint

return false

return true

if dir is horizontal

loop with i from point’s col to point’s col + ship length

if point at (point’s row, i) is not validPoint

return false

return true

return false

void GoodPlayer::huntProb()

set probability array to all zeros

loop through all ships in undestroyed ships

loop through all points (row, col) on board

loop with r from row - ship length + 1 to row

if validPlace ship at (r, col) vertically

increment probability array at (r, col)

loop with c from col - ship length + 1 to col

if validPlace ship at (row, c) horizontally

increment probability array at (row, c)

find smallest ship length N

loop through all points on board

if point is not every other N points

set probability array at point to zero

void GoodPlayer::targetProb()

set probability array to all zeros

get point that triggered target mode as (row, col)

loop through all undestroyed ships

loop along vertical crosshair up to ship length - 1 points

if able to place ship at crosshair point

loop from point’s row up to ship length - 1 points

increment probability array at the position

loop along horizontal crosshair up to ship length - 1 points

if able to place ship at crosshair point

loop from point’s col up to ship length - 1 points

increment probability array at the position

if at least two points were hit during target mode

if both points on the same row

loop through all points on row

double weights on point

increase weight on point by 10/(distance from trigger)

if both points on the same col

loop through all points on col

double weights on point

increase weight on point by 10/(distance from trigger)

loop through all hit undestroyed ship points

set probability array at point to zero

Point GoodPlayer::recommendAttack()

if no undestroyed ships left

return point at (0, 0)

if on hunt mode

calculate probabilities with huntProb()

if on target mode

calculate probabilities with targetProb()

loop through probability array

find point with highest probability

return point with highest probability

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

if no undestroyed ships left

return

if shot hit a ship

push back Point p int to destroyed points

set mode to target

else

push back Point p to missed points

if attack destroyed a ship

erase ship from vector of undestroyed ships

move all points between Point p and trigger point from destroyed points to missed points

if no points left in destroyed points

set mode to hunt