Project Title:

“Finding the Battleships”

Supercomputing Challenge

New Mexico

Final Report

April 3, 2019

Team #87

Media Arts Collaborate Charter School

Team Member(s):

Seungbin Chung

Teacher:

Creighton Edington

Project Mentors:

Creighton Edington

Geoff Danielson

Acknowledgements:

I would like to acknowledge the following for people for assisting and supporting me throughout this project:

Creighton Edington: For teaching me the basics of how this project is done and for guiding me throughout most of this project

Geoff Danielson: For helping me with figuring out the logic of one of the most crucial search patterns and for mentoring me.

Table of Contents:

Introduction

We search for things every day. Whether it's on the internet or whether we're looking for something that's missing or hidden, these things are always a part of our everyday lives. Searching can apply to anything in a serious or fun matter. However, each search method has a different approach regarding the situation. Say we use the game, Battleship. How can we optimize a search pattern to figure out what is the most efficient way at finding ships? From what set of search methods can we use to compare and determine what has the highest hit rate?

To search further into the question, I recreated the game Battleship using Netlogo. There were two methods I used to compare data: one with spacing between the ships, and one with no spacing. I also conducted this experiment using three search pattern methods to find out which of the three are the most efficient at finding the battleships.

Materials and Methods:

To research this problem, I used 3 steps into solving the problem:

1. Creating a program that plays the game

2. Creating an Ai (artificial intelligence) which plays the game

3. Figuring out what is the most efficient search pattern.

I did this experiment with the program, Netlogo using "BehaviorSpace" for my trial.

How Battleship is played:

There is a total of five ships: the carrier, battleship, cruiser, submarine, and destroyer. Each have their own unique sizes from carrier which is the largest taking five tiles of space and the destroyer which is the smallest taking two tiles of space. The ships and the number of spaces is listed below (data 1):

Ship:	**Number of spaces:**
Carrier	5
Battleship	4
Cruiser	3
Submarine	3
Destroyer	2

(Data of ships and sizes in Battleship) (data 1)

The players place their ships down on a 10x10 grid and then take turns at guessing the coordinates of their enemy ship. The main objective of the game is to sink all enemy ships. Each player has a chance of either hitting a ship or missing a ship. Down below is an example of the layout of the game:

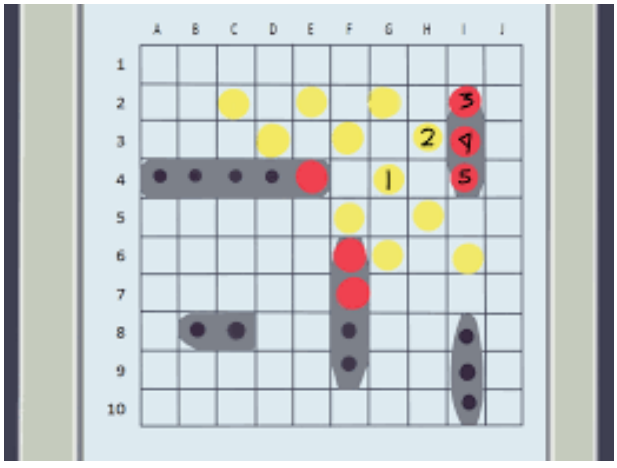
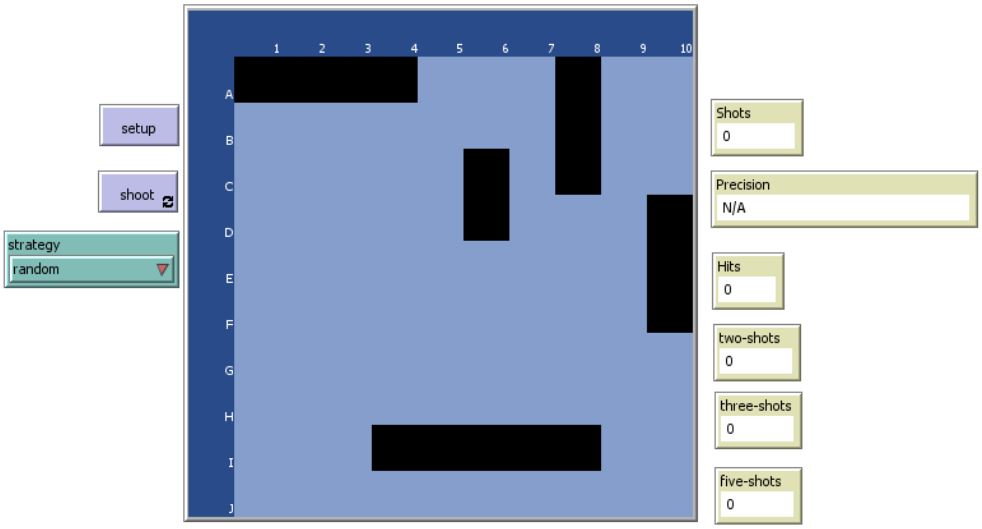


Image fetched from: https://www.wikihow.com/images/a/a9/Play-Battleship-Step-15-Version-2.jpg

The coordinates are laid out from A-J and 1-10. Let's say for example a player calls out the coordinates A4. If a ship is there, the player will call out hit, if a ship is not there then the player will call out miss. The players will place a marker usually red for hit and white for miss, to indicate whether the ship was hit or not.

Shot Patterns and Code

By using Netlogo, I recreated Battleship and created the program to play the game. The screenshot below is what my current game looks like:



(For this research, I mainly used the spaced out version to conduct my experiment.)

The following below are screenshots of the code for the search pattern algorithms:

```
to target-random-patch
  ask patch random-pxcor random-pycor ;looks somewhere random inside
  [
    ifelse (pcolor = blue + 2 or pcolor = black) ;if patch color is
    [
      ifelse (ship-here = 1) ;If there is a ship at the location (b
      [
        set pcolor red
        set shots-fired shots-fired + 1
        set hits hits + 1 ;this is the counter for the number of sh
        set last-hit-pxcor [pxcor] of patch-at 0 0
        set last-hit-pycor [pycor] of patch-at 0 0
        set last-shot? 1
        set global-targeting-sequence global-targeting-sequence + 1
        set targeting-sequence global-targeting-sequence
      ]
      [
        set pcolor white ;sets patch color to white (blue + 2)
        set shots-fired shots-fired + 1
      ]
    ]
    [
      shoot ;re-calls the shoot function command
    ]
  ]
end
```

(Random shot pattern above) (image 1)

```
to shot-pattern-two ;this is the shot function for finding pt-boat
  set search-pxcor search-pxcor + 2 ;increment pxcor by 2 spaces in
  if search-pxcor > 10
  [
   set search-pxcor search-pxcor - 11 ; resets search-pxcor back to
   set search-pycor search-pycor + 1 ; increments pycor up one row
  ]
  ask patch search-pxcor search-pycor
  [
    ifelse (pcolor = blue + 2 or pcolor = black) ;if patch color is
    [
      ifelse (ship-here = 1) ;If there is a ship at the location (bl
      [
        set pcolor red
        set shots-fired shots-fired + 1
        set hits hits + 1 ;this is the counter for the number of shi
        set last-hit-pxcor [pxcor] of patch-at 0 0 ;records the last
        set last-hit-pycor [pycor] of patch-at 0 0 ;records the last
        set last-shot? 1 ;sets the previous/current shot that hit th
        set global-targeting-sequence global-targeting-sequence + 1
        set targeting-sequence global-targeting-sequence ;this is th
        set two-shots two-shots + 1 ;this is the counter for the pat
      ]
      [
        set pcolor white ;sets patch color to white (blue + 2)
        set shots-fired shots-fired + 1
        set two-shots two-shots + 1 ;this is the counter for the pat
      ]
    ]
    [
      shoot
    ]
  ]
end

to shot-pattern-three ;this is the shot function for finding sub and
```

(Algorithm for both pt-boat and carrier) (image 2)

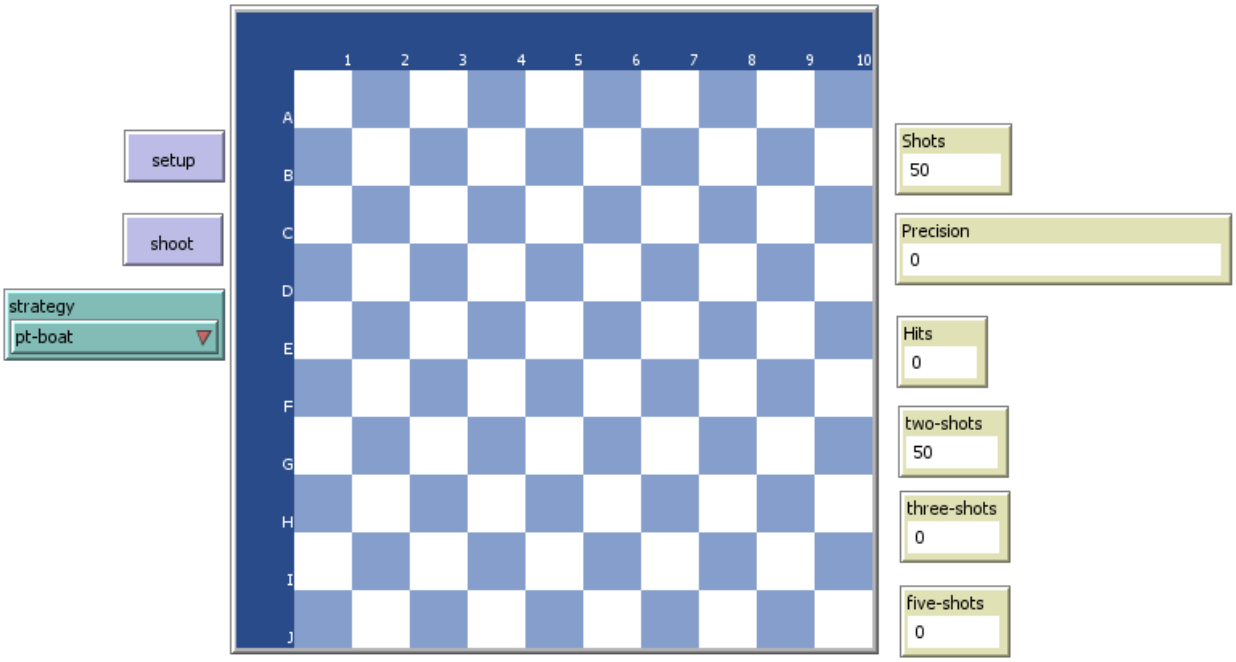
```
to shot-pattern-three ;this is the shot function for finding sub and
  set search-pxcor search-pxcor + 3 ; increment pxcor by 3 spaces in
  if search-pxcor > 10
  [
   set search-pxcor search-pxcor - 11 ; resets search-pxcor back to
   set search-pycor search-pycor + 1 ; increments pycor up one row
  ]
  ask patch search-pxcor search-pycor
  [
    ifelse (pcolor = blue + 2 or pcolor = black) ;if patch color is
    [
      ifelse (ship-here = 1) ;If there is a ship at the location (bl
      [
        set pcolor red
        set shots-fired shots-fired + 1
        set hits hits + 1 ;this is the counter for the number of shi
        set last-hit-pxcor [pxcor] of patch-at 0 0 ;records the last
        set last-hit-pycor [pycor] of patch-at 0 0 ;records the last
        set last-shot? 1 ;sets the previous/current shot that hit th
        set global-targeting-sequence global-targeting-sequence + 1
        set targeting-sequence global-targeting-sequence ;this is th
        set three-shots three-shots + 1 ;this is the counter for the
      ]
      [
        set pcolor white ;sets patch color to white (blue + 2)
        set shots-fired shots-fired + 1 ;shot counter
        set three-shots three-shots + 1 ;this is the counter for the
      ]
    ]
    [
      shoot
    ]
    if (three-shots = 20) ;if the counter reaches 20, then it will s
    [
      set shot-3 0 ;this stops the 3 shot search pattern
      set shot-2 1 ;this initializes the 2 shot search pattern
    ]
  ]
end
```

(One of the shot pattern algorithms for carrier) (Image 3)

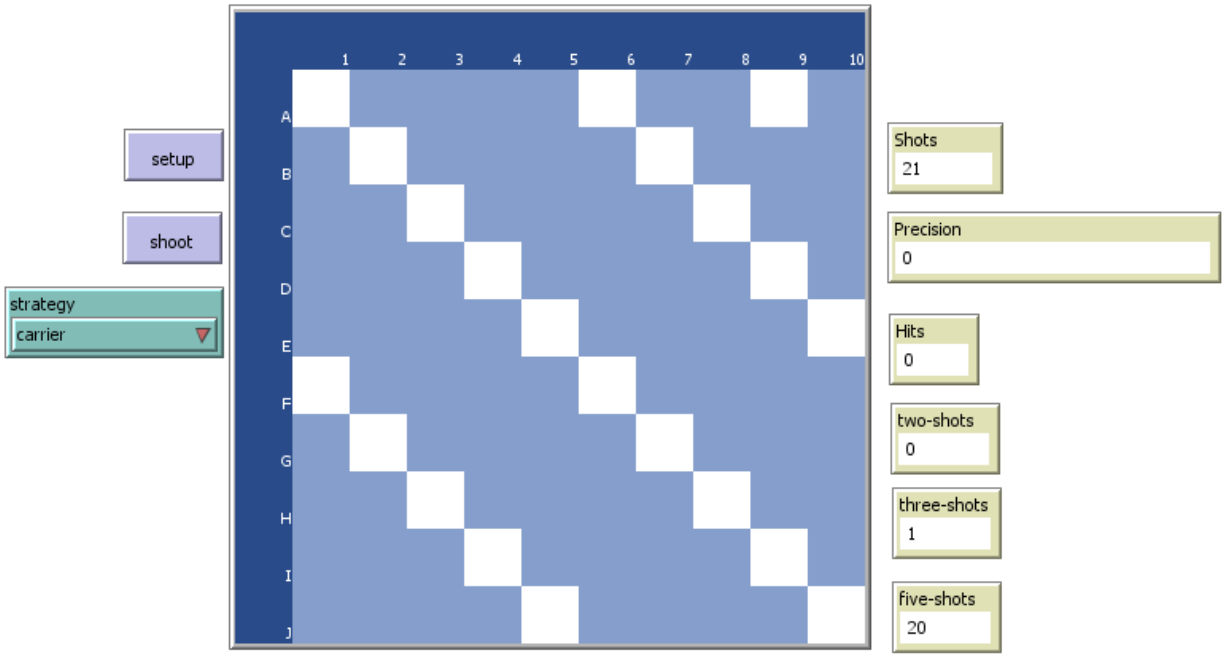
```
to shot-pattern-five ;this is the shot function for finding carrier
  set search-pxcor search-pxcor + 5 ; increment pxcor by 5 spaces in
  if search-pxcor > 10
  [
   set search-pxcor search-pxcor - 11 ; resets search-pxcor back to
   set search-pycor search-pycor + 1 ; increments pycor up one row
  ]
  ask patch search-pxcor search-pycor
  [
    ifelse (pcolor = blue + 2 or pcolor = black) ;if patch color is
    [
      ifelse (ship-here = 1) ;If there is a ship at the location (bl
      [
        set pcolor red ;sets color to red (if it hit)
        set shots-fired shots-fired + 1 ;shot counter
        set hits hits + 1 ;this is the counter for the number of shi
        set last-hit-pxcor [pxcor] of patch-at 0 0 ;records the last
        set last-hit-pycor [pycor] of patch-at 0 0 ;records the last
        set last-shot? 1 ;sets the previous/current shot that hit th
        set global-targeting-sequence global-targeting-sequence + 1
        set targeting-sequence global-targeting-sequence ;this is th
        set five-shots five-shots + 1 ;this is the counter for the p
      ]
      [
        set pcolor white ;sets patch color to white (blue + 2) (if i
        set shots-fired shots-fired + 1 ;shot counter
        set five-shots five-shots + 1 ;this is the counter for the p
      ]
    ]
    [
      shoot
    ]
    if (five-shots = 20) ;if the counter reaches 20, then it will st
    [
      set shot-5 0 ;this stops the 5 shot search pattern
      set shot-3 1 ;this initializes the 3 shot search pattern
    ]
  ]
end
```

(Main shot pattern for carrier above) (image 4)

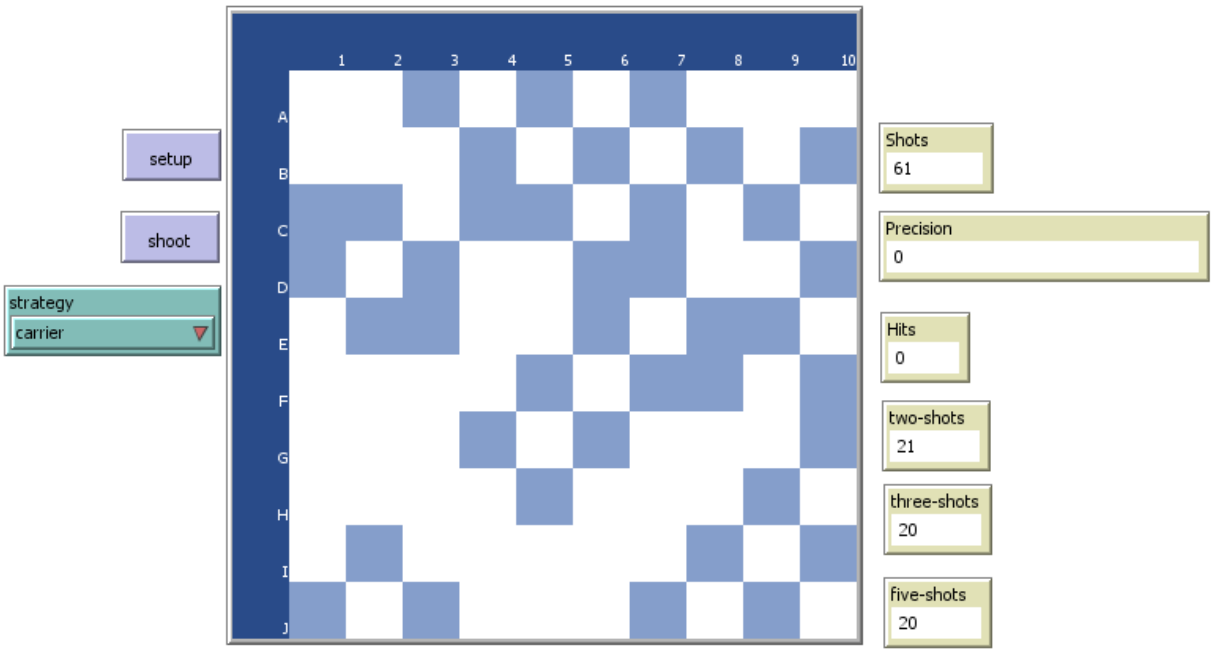
Information*: These are the main algorithms that go into the three shot patterns. The random shot pattern will fire wildly until it hits something. The pt-boat shot pattern skips every other tile and shoots (see image 6 below). The carrier shot pattern switches from 5 tiles, to 3 tiles, then to 2 tiles. The carrier will skip 5 tiles until it hits the very top of the board, then will do the same to 3, and then finally will finish off with 2 tiles. Below are examples of the pt-boat and the carrier shot pattern being used without the ships. Screenshots in respective order: pt-boat, carrier, and carrier (afterwards)



(Shot pattern for pt-boat above) (image 5)



(Shot pattern for carrier five-shots first above) (image 6)



(Shot pattern for carrier, full shot pattern above) (image 7)

Note: Carrier switches patterns from skipping every 5 tiles when the five-shots indicator hits 20, to 3, then to two (see image 6 and 7 above). The algorithm all these patterns also have is a ship searcher where if a ship gets hit, it searches around the 4 tiles right next to the hit spot. If it does find a ship, it keeps moving on trying to look for more in the vicinity. If it doesn't find another ship, then it will resume its original search pattern (See image 10 below). Down below are screen shots of the code and an example of its usage:

```
;this function searches around the ship if a ship has been hit
to next-shot
  let target-heading 0 ;target heading is the area around the ship
  while[target-heading < 360];loop breaker for when a ship has been searche
  [
    ask patch last-hit-pxcor last-hit-pycor
    [
      ask neighbors4 ;this asks the neighboring four patches around the shi
        [
          if (pcolor = blue + 2 or pcolor = black)
          [
            ifelse (ship-here = 1)
            [
              set pcolor red ;turns the tile to red if the ship is there
              set shots-fired shots-fired + 1 ;shot counter
              set hits hits + 1 ;hit counter
              set global-targeting-sequence global-targeting-sequence + 1 ;
              set targeting-sequence global-targeting-sequence ;the targeti
              set last-shot? 1 ;sets last shot shit was hit to true
            ]
            [
              set pcolor white ;if it missed, then sets color to white
              set shots-fired shots-fired + 1 ;shot counter
            ]
          ]
        ]
      set target-heading target-heading + 90 ;this is the function that rot
    if hits = 17 ;placed a stop because I noticed that it seemed to ignore
    [
      stop
    ]
    ]

  ask patch last-hit-pxcor last-hit-pycor ;this stores the last x coordinat
    [
    set targeting-sequence 0 ;resets the targeting sequence after it confir
  ]
```

(Part 1 for algorithm searching around ships when ship is hit) (image 8)

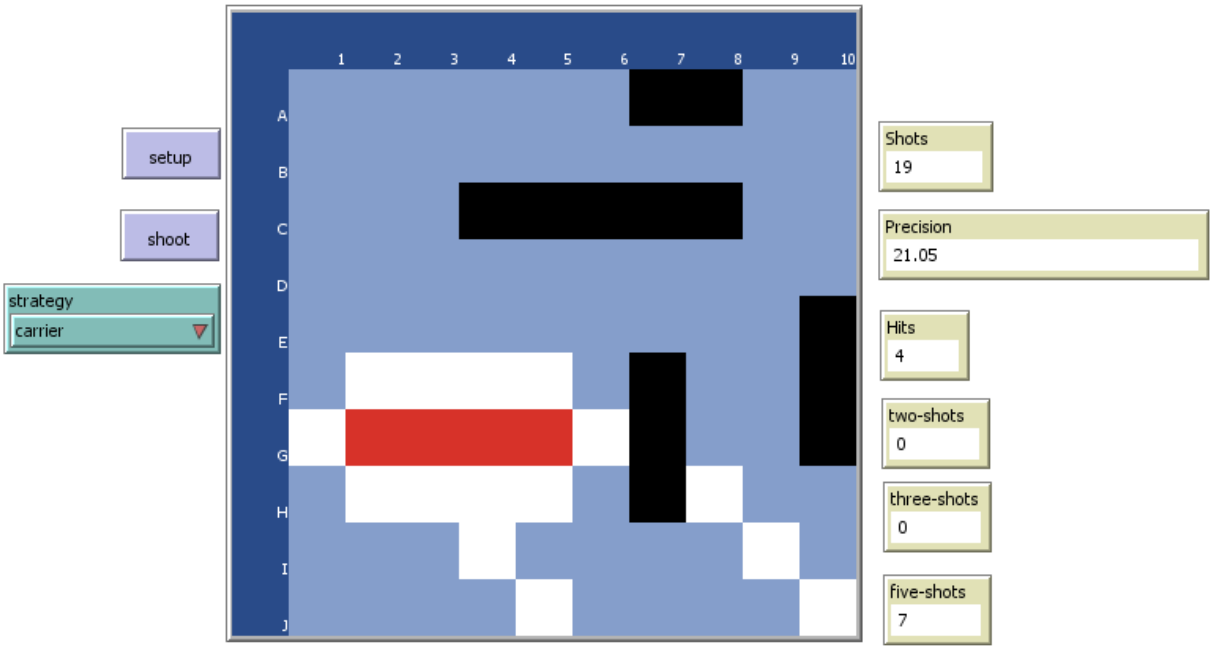
```
  ask patch last-hit-pxcor last-hit-pycor ;this stores the
    [
    set targeting-sequence 0 ;resets the targeting sequenc
  ]

  ;check-next-target checks for any ships with a targeting

  ifelse(count patches with [targeting-sequence > 0] > 0)
  [
    let x-stepper 1 ;;the x-stepper and the Y-stepper are
    let y-stepper 0
    while[y-stepper < 10] ;;This runs a loop while the y-s
    [

    ask patch x-stepper y-stepper
   [
      if(targeting-sequence  != 0) ;if a ship has been tar
      [
      set last-hit-pxcor [pxcor] of patch-at 0 0 ;records
      set last-hit-pycor [pycor] of patch-at 0 0 ;records
      set y-stepper 10
      ]
    ]
      set x-stepper x-stepper + 1 ;;adds a value to the pa
      if(x-stepper = 11) ;;if the parameter value (x) is 1
      [
       set x-stepper 1;;resets the parameter (x) value to
       set y-stepper y-stepper + 1 ;;adds to the parameter
      ]
  ]
  ]
  [
   set last-shot? 0
  ]
end
```

(Part 2 for algorithm searching around ships when ship is hit) (image 9)



(Example of next-shot being used) (image 10)

Information*: The way the next shot pattern works is it assigns the hit ship a value. The function I used to describe this method is called the targeting-sequence. It'll look around until it has searched all 4 tiles next to it and then reset the value to 0, meaning it won't look around there anymore. After that, it'll assign another value if a tile hit was also part of the ship. If there are no more ships to be found after that, all values will be reset to 0 and then it'll return to its regular shoot pattern.

Data Results:

Using BehaviorSpace allowed me to run 1000 times and record it down on an excel sheet. By doing this, I took the data given, and averaged the results of the precision of all runs for each pattern. I did this for two of my versions: the spaced and the non-spaced version. Down below is a data table of the averages and the standard deviation of both versions. (See data table 2 and 3 below)

Note: Standard deviation was used to determine which shot pattern was the most consistent.

Spaced Data Table:

Shot Patterns:	Trial #1	Trial #2	Trial #3
Carrier	Average: 24.47904 Standard deviation: 2.534173339	Average: 24.57789 Standard deviation: 2.549550481	Average: 24.24046 Standard deviation: 2.543882727
Pt-boat	Average: 23.99062 Standard deviation: 1.610049588	Average: 24.13793 Standard deviation: 1.607781445	Average: 24.06614 Standard deviation: 1.626902121
Random	Average: 23.15354 Standard deviation: 3.128043722	Average: 22.97259 Standard deviation: 3.116011059	Average: 22.97422 Standard deviation: 2.998121332

(Spaced data table) (data 2)

Non-Spaced Data Table:

Shot Patterns:	Trial #1	Trial #2	Trial #3
Carrier	Average: 27.78826 Standard Deviation: 4.636896	Average: 28.00133 Standard Deviation: 4.830557	Average: 28.23691 Standard Deviation: 4.878385
Pt-boat	Average: 26.31623 Standard Deviation: 3.081075	Average: 26.24216 Standard Deviation: 3.082028	Average: 26.28924 Standard Deviation: 3.035912
Random	Average: 27.1237 Standard Deviation: 5.683653	Average: 26.76697 Standard Deviation: 5.494594	Average: 26.18048 Standard Deviation: 5.175764

(Non-spaced data table) (data 3)

The result of the data shown above is interesting, because we can see a significance between spaced and non-spaced in terms of both average and standard deviation. Since spaced is harder to track down, it has less of an average in terms of precision (see data 2 above). Non-spaced on the other hand has possibilities of making ships easier to find which causes the average to go up (see data 3 above). What surprised me the most about the results was the non-spaced where pt-boat was the least precise between the carrier and the random shot patterns (data 3). The carrier on both data sets was the most effective one, and the second most consistent compared to the pt-boat which had the lowest standard deviation in both sets of data. Even though there were more methods to solve this by, I chose the most common methods used by people to see how effective the comparison might be.

Conclusion

In conclusion, I was able to determine from the three sets of search patterns, that the carrier had the most effective way of searching for hidden battleships. This was a surprising result, as I expected the pt-boat method to be the most effective since it was the most consistent search pattern. I believe that the carrier may have had the edge because it's wide range of search made it easier to cover spots that were less examined, and the way the code was set up, from skipping from 5 to 3 to 2 was the fastest way to get all the ships.

Bibliography

Pattern Search (Optimization), Wikipedia, February 7th, 2019

(https://en.wikipedia.org/wiki/Pattern_search_(optimization))

Tips how to win battleship, UltraBattleship

(http://www.ultrabattleship.com/tips.php)

Search Algorithm, Wikipedia, April 1st, 2019

https://en.wikipedia.org/wiki/Search_algorithm

Battleship, DataGenetics

http://www.datagenetics.com/blog/december32011/