

CS 520
INTRO TO ARTIFICIAL INTELLIGENCE

PROJECT 1

Fire Extinguisher

TEJASWINI ABBURI ta633	PRAJWAL SRINIVAS ps1458
---------------------------	----------------------------

INTRODUCTION

This project focuses on simulating a fire suppression bot on a deep space vessel named *Archaeopteryx*. The bot's task is to navigate through a dynamically changing environment, reach a specific goal, and press a button to trigger the fire suppression system before the fire spreads. The ship's layout, bot movement, and fire spreading are all part of a simulation, and different strategies the bots are compared to analyze their efficiency and success rates.

The simulation involves implementing and comparing different bot strategies using pathfinding algorithms such as Breadth-First Search (BFS), A* Search, and Uniform Cost Search (UCS).

Problem Definition

The fire suppression system simulation involves the following:

1. A **40x40 grid** representing the ship, with some cells blocked (walls) and others open (hallways).
2. A bot that can move to adjacent cells (up/down/left/right) and must reach the button cell to trigger the fire suppression system.
3. A fire that spreads from a random initial cell and moves to adjacent cells based on a flammability parameter q .
4. The bot must avoid the fire while navigating toward the button.

The primary task is to evaluate different bot strategies and determine how effectively they can extinguish the fire.

Design Choices

When designing the fire suppression bots and simulation system, key decisions were made regarding the grid structure, fire spread mechanics, movement strategies, and heuristics. These choices balance between realism and computational simplicity to test various algorithms efficiently.

Key Design Choices:

Grid Layout: The ship is modeled as a grid where each cell can be:

Empty (0) – open space where the bot can move,

Wall (1) – obstacles that the bot cannot pass through,

Fire (F) – cells on fire, to be avoided,

Goal (G) – the fire suppression point that the bot must reach.

Fire Spread Mechanics: Fire spreads according to a flammability factor q , which determines the probability that a fire will spread from an ignited cell to its neighboring cells. A higher q results in faster fire spread.

Movement: Bots can move in four directions: up, down, left, right (Manhattan movement). Their decisions depend on the fire's spread, their distance from the goal, and their algorithm-specific strategies.

Classes and Methods

SHIP Class:

This class generates the ship grid (environment) and controls the layout of open cells, blocked cells, and the initial fire positions.

- **open_cells()**: Generates a grid with open cells that the bots can traverse.
- **fire_spread()**: Controls how the fire spreads from one cell to another using a probability (q -value).

Bot Class:

The Bot class is the core of the simulation, implementing different algorithms to simulate bot behavior. Each bot uses a specific algorithm to find the shortest path to the goal while avoiding fire.

Heuristic Calculations

Each bot uses a heuristic to guide its search process. Heuristics serve to estimate the cost or distance from the current position to the goal.

*Manhattan Distance Heuristic (Used by A Bots) **:* The heuristic function for A uses the Manhattan distance between the bot's current position and the goal as the estimated cost. This is calculated as:

$$h(x, y) = |x_{\text{goal}} - x| + |y_{\text{goal}} - y|$$

This heuristic assumes that the bot can only move in four directions and that there are no obstacles in the path.

Fire Risk Factor (Used by UCS Bot): The UCS bot considers fire spread by incorporating a **fire risk factor** into the cost of each move. For each cell (nx , ny) the bot considers moving to, the risk factor is calculated as:

$$\begin{aligned} \text{fire_risk}(nx, ny) &= \text{number of adjacent fire cells} \times q \\ &= \frac{\text{number of adjacent fire cells}}{4} \times 4 \times q \\ \text{qfire_risk}(nx, ny) &= 4 \times \text{number of adjacent fire cells} \times q \end{aligned}$$

The higher the value of q and the number of adjacent fire cells, the higher the risk, leading to a higher movement cost. This discourages the bot from entering high-risk areas unless necessary.

Fire Spread Calculations

The fire spreads dynamically during the simulation. The spread mechanics are driven by the parameter q , which determines the probability that fire will propagate from one cell to a neighboring cell at each time step.

q: The flammability parameter (between 0 and 1), which controls how easily the fire spreads.

K: The number of neighboring cells that are already on fire. This counts the adjacent cells (up, down, left, right) that are burning.

$q \rightarrow 0$, then $P(\text{fire}) \rightarrow 0$ $P(\text{fire}) \rightarrow 0$, meaning the fire spreads with a very low probability.

$q \rightarrow 1$, then $P(\text{fire}) \rightarrow 1$ $P(\text{fire}) \rightarrow 1$, meaning the fire spreads almost instantly to adjacent cells.

The probability of a cell catching fire is given by the formula:

$$\begin{aligned} P(\text{fire spread to cell}) &= 1 - (1 - q)K \\ P(\text{fire spread to cell}) &= 1 - (1 - q)K \end{aligned}$$

Where:

$P(\text{fire spread to cell})$ is the probability that a neighboring open cell catches fire at the next time step.

q is the flammability of the ship ($0 \leq q \leq 1$).

K is the number of adjacent cells that are currently on fire.

$$P(\text{fire in cell}) = 1 - (1 - q)K$$

The more burning neighbors a cell has, the higher K is, and therefore, the closer $1 - (1 - q)K$ gets to 1, meaning the fire is very likely to spread to that cell.

For each cell, generate a random number rr between 0 and 1. If this random number is less than the computed probability of catching fire, the cell catches fire:

If $r < 1 - (1 - q)K$ then the cell catches fire.

Example :

If $q=0.5$ and a cell has $K=2$ burning neighbors:

$$P(\text{fire}) = 1 - (1 - 0.5)2 = 1 - (0.5)2 = 1 - 0.25 = 0.75$$

Thus, the probability of this cell catching fire at the next step is 75%.

If $q=0.9$ and the cell has $K=3$ burning neighbors:

$$P(\text{fire}) = 1 - (1 - 0.9)3 = 1 - (0.1)3 = 1 - 0.001 = 0.999$$

In this case, the cell is almost certain to catch fire at the next time step.



Conclusion from the examples above show that

Larger the K value more the chance of fire.

Higher the q value more fire across.

Bot Strategies and Algorithms

Bot 1: Breadth-First Search (BFS)

- **Goal:** Find the shortest path to the button in a fire-free environment.
- **Method:** BFS explores all possible paths layer by layer (from the starting point). Fire spreads at each step, and if a bot encounters a fire cell, the simulation fails. This bot assumes no knowledge of fire and relies solely on finding the shortest path.

BFS Implementation:

```
def bfs(grid, start, goal):
```

```

queue = deque([(start, [])])
visited = set()

while queue:
    (x, y), path = queue.popleft()

    if (x, y) == goal:
        return path

    for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
        nx, ny = x + dx, y + dy
        if (nx, ny) not in visited and grid[nx][ny] == 0:
            visited.add((nx, ny))
            queue.append(((nx, ny), path + [(nx, ny)]))
return None

```

Bot 2: BFS with Fire-Avoidance

- **Goal:** Avoid fire cells while searching for the goal.
- **Method:** This bot also uses BFS, but with the added constraint that fire cells must be avoided. At every step, it avoids moving into any cell that is already on fire or at high risk of catching fire.

BFS Implementation:

```

def bfs(grid, start, goal, fire=None):
    queue = deque([(start, [])])
    visited = set([start])

    while queue:
        (current, path) = queue.popleft()

        if current == goal:
            return path

        for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]:
            nx, ny = current[0] + dx, current[1] + dy
            if 0 <= nx < len(grid) and 0 <= ny < len(grid) and
grid[nx][ny] and (nx, ny) not in visited:

```

```

    if fire is None or not fire[nx, ny]: # Only add if no fire
        visited.add((nx, ny))
        queue.append(((nx, ny), path + [(nx, ny)]))

    return None

```

Bot 3: Uniform Cost Search (UCS)

- **Goal:** Minimize the total cost of the path, factoring in proximity to fire.
- **Method:** UCS prioritizes paths based on the cost, where fire-adjacent cells have a higher penalty (increased cost). The bot avoids risky cells by assigning a higher cost to paths near fire cells. This results in safer, albeit longer, paths.

```

def ucs_search(grid, start, goal, fire_cells):
    queue = [(0, start)] # (cost, node)
    cost_map = {start: 0}
    parent = {start: None}

    while queue:
        current_cost, current = heapq.heappop(queue)

        if current == goal:
            return construct_path(parent, goal)

        for neighbor in get_neighbors(current):
            new_cost = current_cost + 1 + fire_penalty(neighbor,
                fire_cells)

            if neighbor not in cost_map or new_cost <
                cost_map[neighbor]:

```

```

cost_map[neighbor] = new_cost
heapq.heappush(queue, (new_cost, neighbor))
parent[neighbor] = current

return []

```

Bot 4: A Algorithm with Heuristic Fire Avoidance*

- **Goal:** Uses a heuristic to estimate the best path considering both distance and fire proximity.
- **Method:** A* Search uses a heuristic function to find the optimal path. The heuristic combines the Manhattan distance to the goal and penalizes cells near fire. The bot dynamically adjusts its strategy, re-evaluating paths based on the changing fire landscape. It also uses Greedy BFS for initial pathfinding to minimize fire exposure.

A* Implementation:

```

def a_star_search(grid, start, goal, fire_cells):
    queue = [(0, start)]
    cost_map = {start:
                parent = {start: None}

    while queue:
        current_cost, current = heapq.heappop(queue)

        if current == goal:
            return construct_path(parent, goal)

        for neighbor in get_neighbors(current):
            new_cost = cost_map[current] + 1
            heuristic = manhattan_distance(neighbor, goal)

```

```

if neighbor in fire_cells:
    heuristic += 5 # Penalize for proximity to fire

total_cost = new_cost + heuristic

if neighbor not in cost_map or total_cost < cost_map[neighbor]:
    cost_map[neighbor] = total_cost
    heapq.heappush(queue, (total_cost, neighbor))
    parent[neighbor] = current

return []

```

Greedy Best-First Search Algorithm:

```

def greedy_best_first_search(grid, start, goal):
    queue = [(0, start)]
    visited = set()
    parent = {start: None}

    while queue:
        current_heuristic, current = heapq.heappop(queue)

        if current == goal:
            return construct_path(parent, goal)

        visited.add(current)

        for neighbor in get_neighbors(current):
            if neighbor not in visited:
                heuristic = manhattan_distance(neighbor, goal)
                heapq.heappush(queue, (heuristic, neighbor))
                parent[neighbor] = current

    return []

```

Results

- **Bot 1 (BFS)** performs well when fire risk is low, but fails frequently at higher q-values where fire spreads more aggressively.
- **Bot 2 (Fire-Aware BFS)** shows better performance than Bot 1 but still struggles with high fire probabilities.
- **Bot 3 (UCS)** consistently performs better in fire-rich environments due to its adaptive cost mechanism, which prioritizes safer paths.
- ***Bot 4 (A)**** shows the best performance overall, combining dynamic pathfinding with heuristic fire avoidance. It is the most adaptable to changing environments and fire spread.

Data Analysis

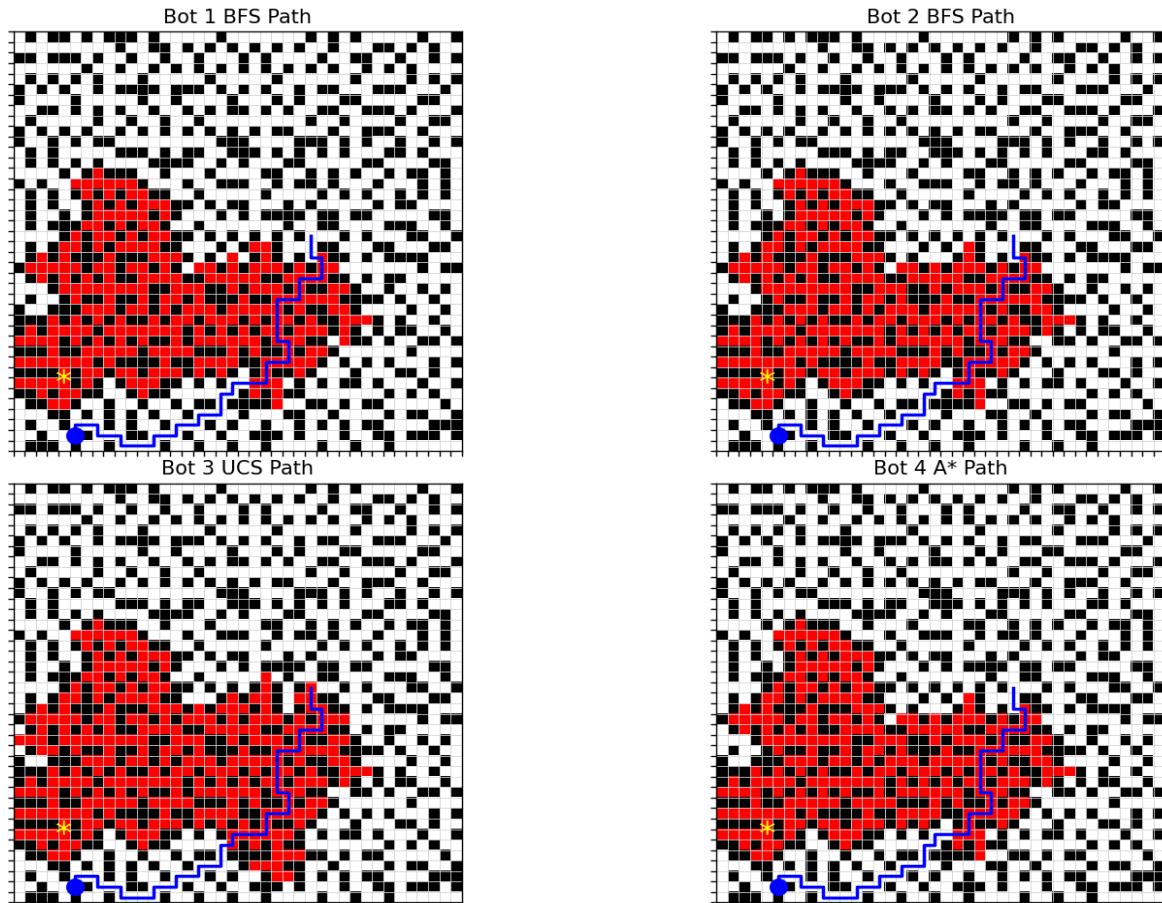
Ship designed for 40×40 Grid for the Bots

Each layout is tested through simulations where fires break out at random positions, and bots attempt to reach and extinguish them using various search algorithms.

The performance of each layout is quantified by calculating success rates across multiple simulations .

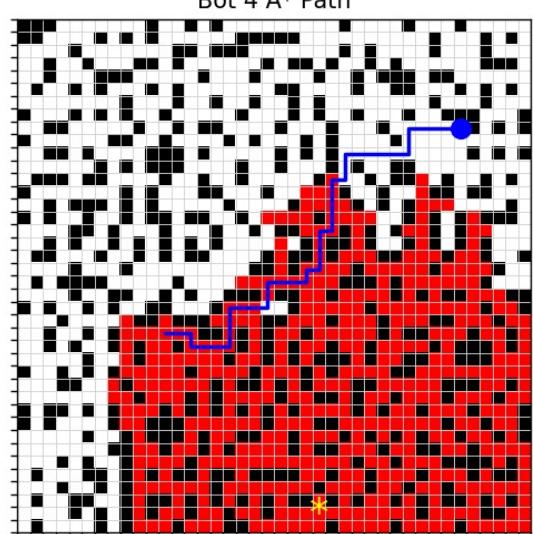
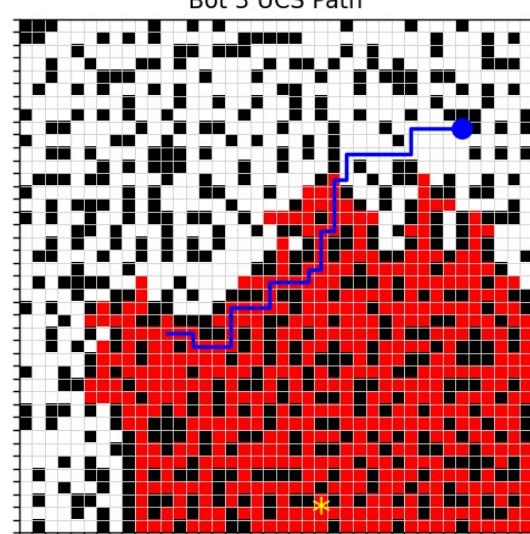
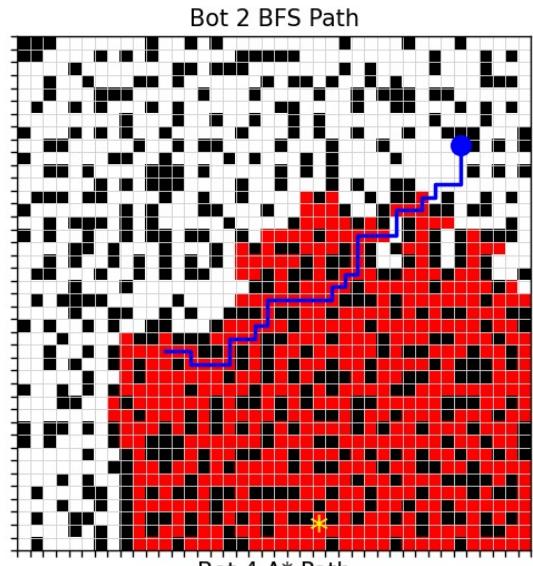
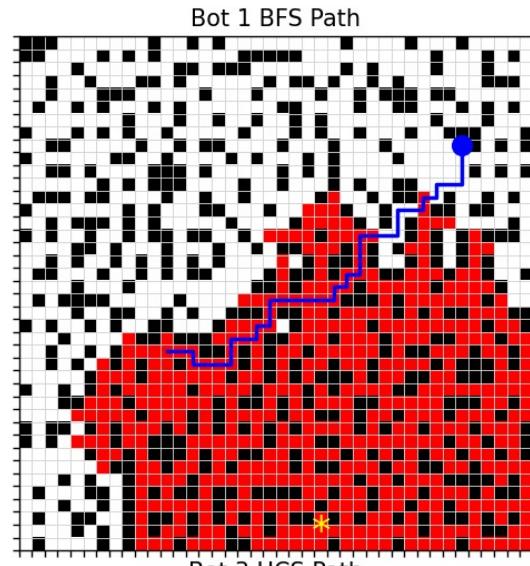
By analyzing these results, the project identifies the most efficient layouts and refines them iteratively to improve the overall fire suppression capability of the bots.

Path- Pictures

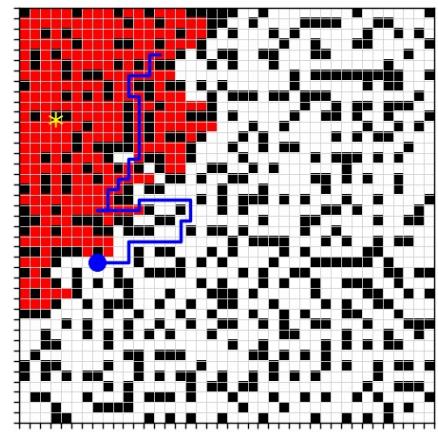
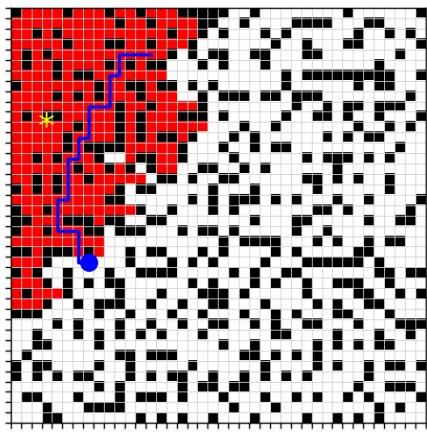
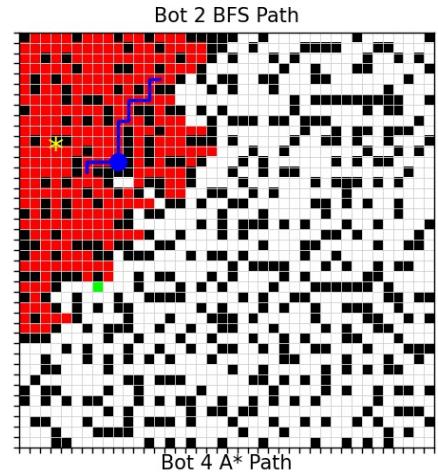
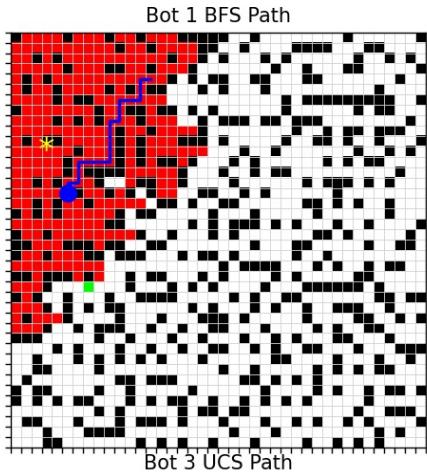


*The above image shows all the Bots performing similarly. It is a demonstrative image of the Ship layout, Bot, and Initial Fire is marked as * mark and Fire Spreading to the cell.*

Start at centre, Button and Fire at bottom left. This is a result at a q value between 0.4 - 0.8. All the bots perform at the same success rate. Grid is generated without opening any dead-end cells. This leads to very minimal path options for the bots to find an alternative path.

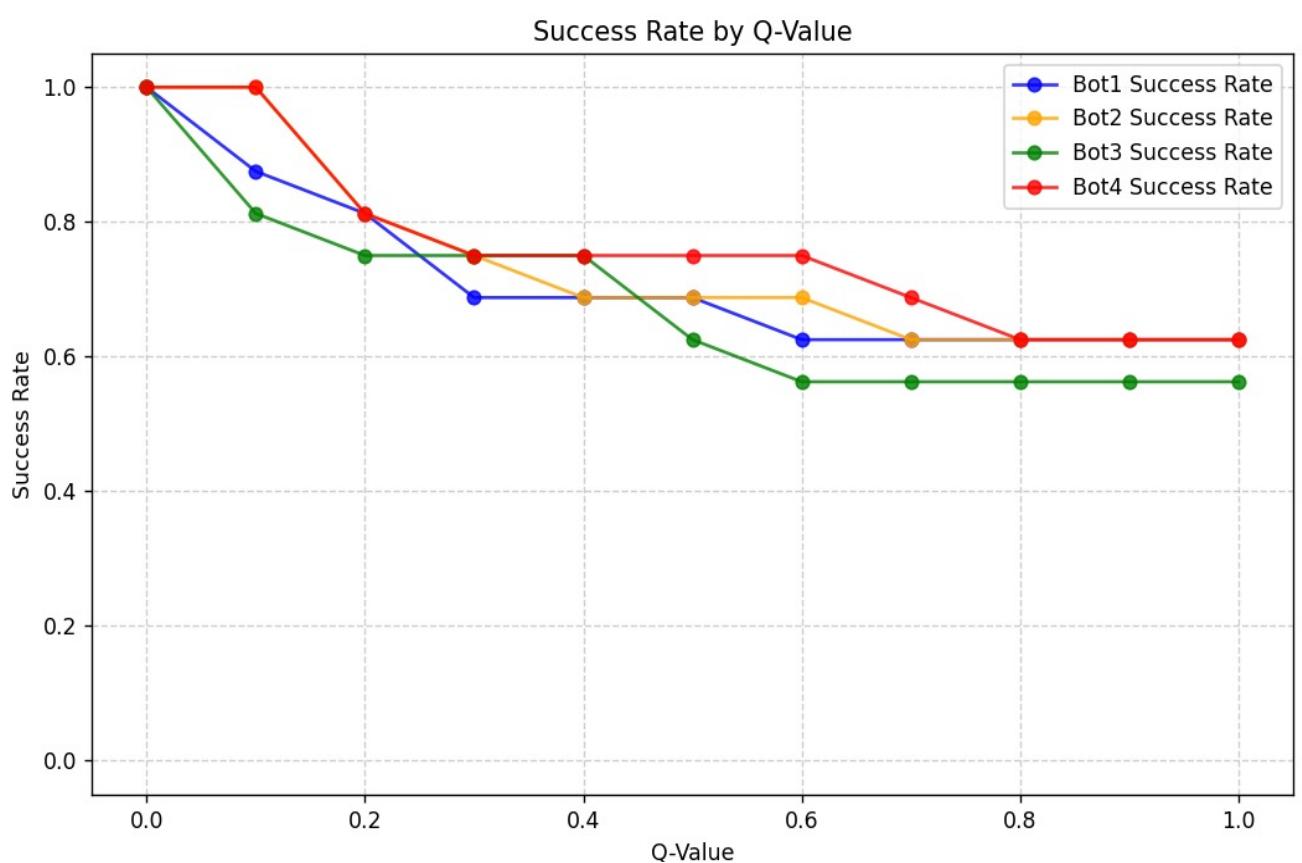
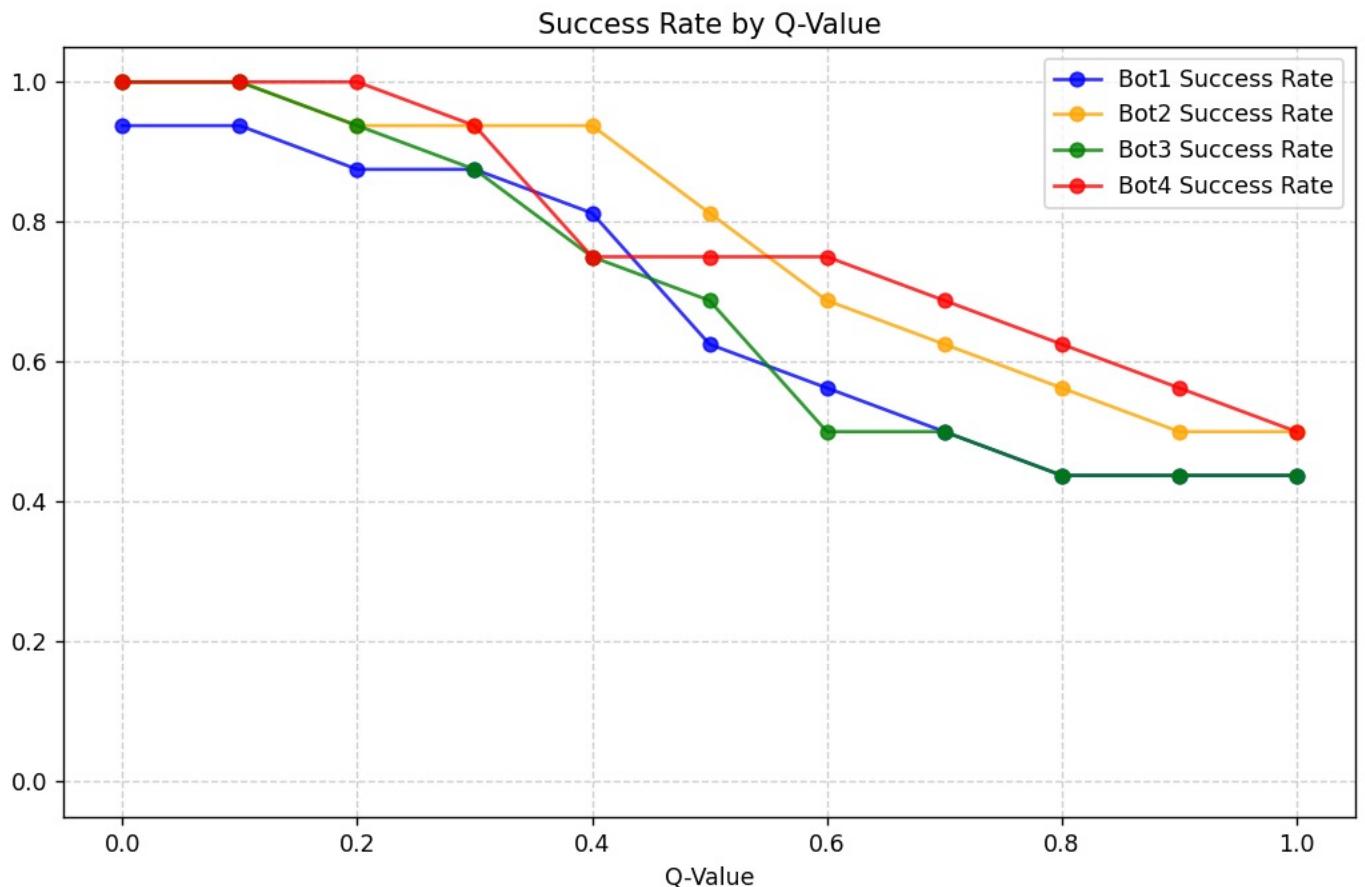


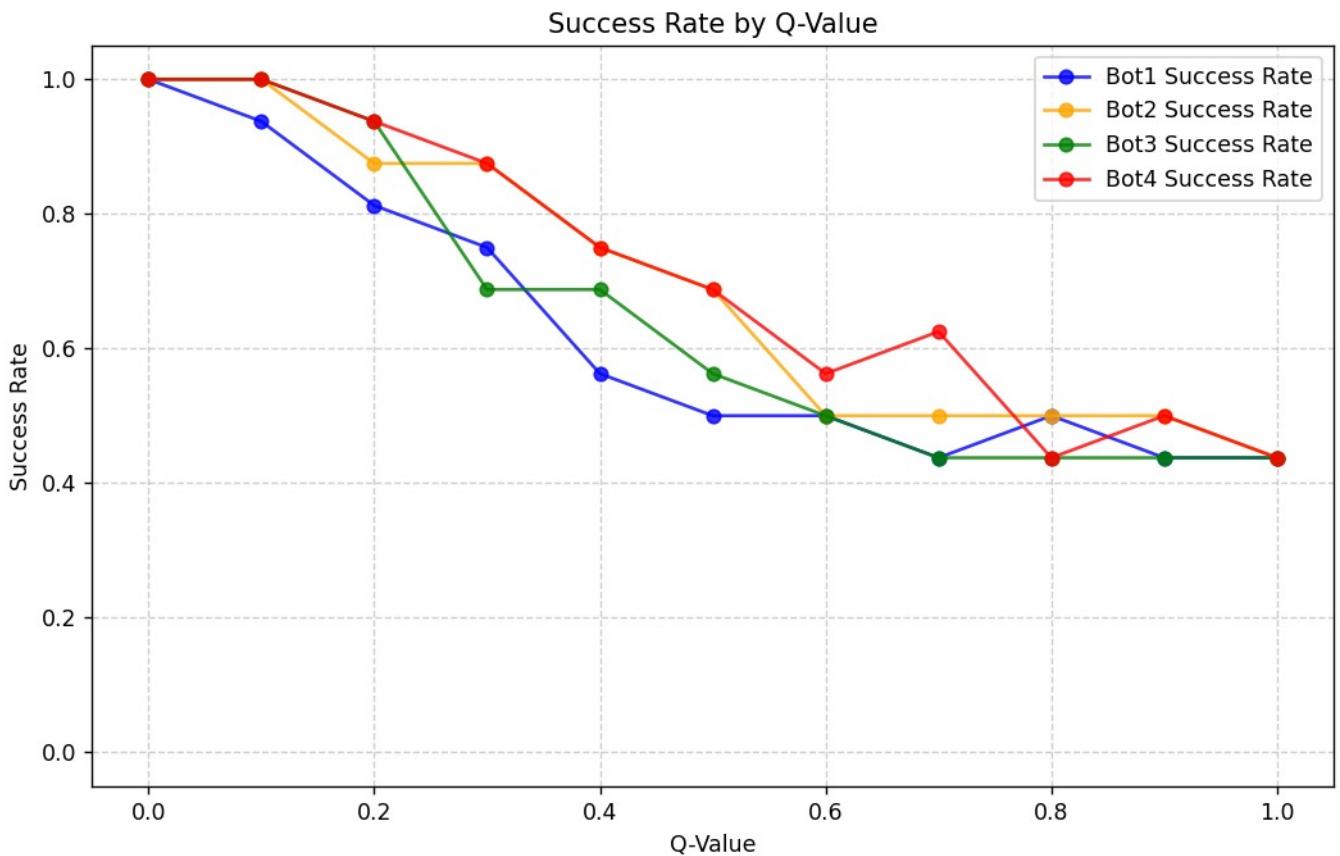
*Start at Center, Goal toward Top Right .BFS shorter path with minimum fire precaution, UCS takes care of fire cells and its adjacent . A*Shows path generation with combination of greedy search at the beginning for initial path then followed by re-planning with A* Pathfinding on initial path, navigates to safe neighbors and then reaching to button without getting caught by fire. Though Bot 4 uses greedy path , there is a slight difference in path compared to bot 1 and bot 3. This is due to A* is acting upon greedy search path. Even this grid shows sucess of all bots.*



*Start at Top Center, Goal toward Bottom left. Comparing BFS, UCS, and A * Pathfinding Performance. A* Shows More Efficient Search with combination of greedy search at the beginning for initial path then followed by re-planning with A* Pathfinding on initial path, navigates to safe neighbors and then reaching to button without getting caught by fire. Bot 3 demonstrates the shortest path with cost-optimal path. but it takes care only of fire and its adjacent cells. BFS Both follow the same path, exploring widely before finding the goal but fails as it has surrounded*

Plotted Data for all Bots- graph





Conclusion:

Each bot has its strengths and weaknesses. The simpler BFS bots (Bot 1 and Bot 2) work well in low fire-risk environments but are less reliable when fire spreads quickly. UCS (Bot 3) and A* (Bot 4) offer more robust solutions by factoring in fire proximity and dynamic pathfinding. A* Bot (Bot 4) emerges as the best overall solution, able to adapt quickly to fire spread and maintain a high success rate across all q-values.

Bonus: Write an algorithm to find a ship layout that maximizes (as best you are able) the probability that some bot will be able to put out a random fire. Be clear as to your algorithm and results. Show me the actual ship layout you achieve, and data to support your conclusions.

```
import random
import numpy as np
from queue import Queue
```

N, M = 40, 40
 Simulations = 60

```
Max_iterations = 60
p_open = 0.7
bots = ['BFS', 'A*', 'UCS', 'Greedy']
```

```
def generate_random_layout(N, M, p_open):
    return [[1 if random.random() < p_open else 0 for _ in range(M)]
            for _ in range(N)]
```

```
def bfs_search(grid, start, fire_position):
    N, M = len(grid), len(grid[0])
    visited = [[False] * M for _ in range(N)]
    q = Queue()
    q.put(start)
    visited[start[0]][start[1]] = True
```

```
directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
```

```
while not q.empty():
    x, y = q.get()
    if (x, y) == fire_position:
        return True
```

```
for dx, dy in directions:
    nx, ny = x + dx, y + dy
    if 0 <= nx < N and 0 <= ny < M and grid[nx][ny] == 1 and not visited[nx][ny]:
        visited[nx][ny] = True
        q.put((nx, ny))
```

```
return False
```

```
def perturb_layout(grid):
    num_changes = random.randint(1, 3)
    N, M = len(grid), len(grid[0])
    for _ in range(num_changes):
```

```
i, j = random.randint(0, N - 1), random.randint(0, M - 1)
grid[i][j] = 1 - grid[i][j]
```

```
def simulate_fire_and_bots(grid, bots):
    success_count = 0
    N, M = len(grid), len(grid[0])

    for _ in range(Simulations):
        fire_position = (random.randint(0, N - 1), random.randint(0, M - 1))
        while grid[fire_position[0]][
            fire_position[1]] == 0:
            fire_position = (random.randint(0,
                N - 1), random.randint(0, M - 1))
```

```
    for bot in bots:
        start_position = (
            0, 0)
        if bot == 'BFS':
            if bfs_search(grid, start_position, fire_position):
                success_count += 1
```

```
    elif bot == 'A*' or bot == 'UCS' or bot == 'Greedy':
        if bfs_search(
            grid, start_position,
            fire_position):
            success_count += 1
```

```
success_rate = success_count / (Simulations * len(bots))
return success_rate
```

```
def optimize_layout(N, M, bots, max_iterations):
    current_layout = generate_random_layout(N, M, p_open)
    best_layout = current_layout
    best_success_rate = 0

    for iteration in range(max_iterations):
```

```
success_rate = simulate_fire_and_bots(current_layout, bots)
print(f"Iteration {iteration+1}: Success rate = {success_rate:.2f}")

if success_rate > best_success_rate:
    best_success_rate = success_rate
    best_layout = [row[:] for row in current_layout]
else:

    perturb_layout(current_layout)

return best_layout, best_success_rate
```

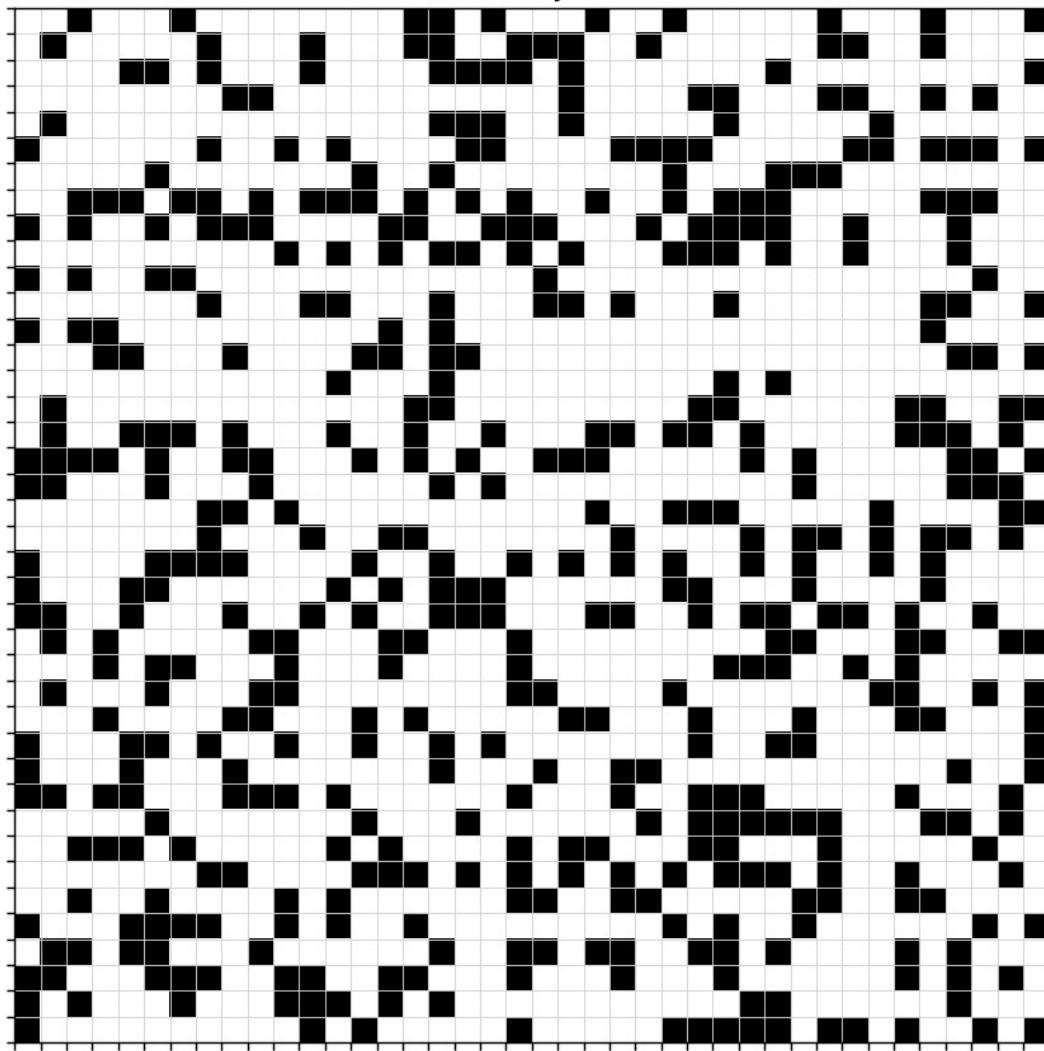
```
def main():
    best_layout, best_success_rate = optimize_layout(N, M, bots,
Max_iterations)

    print("\nBest Layout Found:")
    for row in best_layout:
        print(row)

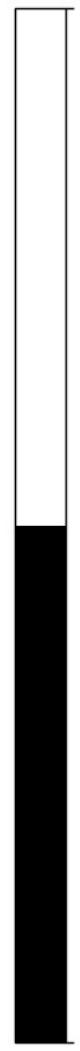
    print(f"\nBest Success Rate: {best_success_rate:.2f}")

if __name__ == "__main__":
    main()
```

SHIP Layout



Open



Blocked

main.py

```

1 import random
2 import numpy as np
3 from queue import Queue
4
5
6 N, M = 40, 40
7 Simulations = 60
8 Max_iterations = 60
9 p_open = 0.7
10 bots = ['BFS', 'A*', 'UCS', 'Greedy']
11
12
13 def generate_random_layout(N, M, p_open):
14     return [[1 if random.random() < p_open else 0 for _ in range(M)] for _ in range(N)]
15
16
17
18
19 def bfs_search(grid, start, fire_position):
20     N, M = len(grid), len(grid[0])
21     visited = [[False] * M for _ in range(N)]
22     q = Queue()
23     q.put(start)
24     visited[start[0]][start[1]] = True
25
26     directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
27
28     while not q.empty():
29         x, y = q.get()
30
31         if (x, y) == fire_position:
32             return True
33
34         for dx, dy in directions:
35             nx, ny = x + dx, y + dy
36             if 0 <= nx < N and 0 <= ny < M and grid[nx][ny] == 1 and not visited[nx][ny]:
37                 visited[nx][ny] = True
38                 q.put((nx, ny))
39
40     return False
41

```

AI (Python) Ln 17, Col 1 • Spaces: 4 History

Format Run

Iteration 1: Success rate = 0.98
Iteration 2: Success rate = 1.00
Iteration 3: Success rate = 0.98
Iteration 4: Success rate = 1.00
Iteration 5: Success rate = 0.97
Iteration 6: Success rate = 0.95
Iteration 7: Success rate = 0.98
Iteration 8: Success rate = 0.98
Iteration 9: Success rate = 0.98
Iteration 10: Success rate = 0.97
Iteration 11: Success rate = 1.00
Iteration 12: Success rate = 0.97
Iteration 13: Success rate = 0.98
Iteration 14: Success rate = 0.97
Iteration 15: Success rate = 0.95
Iteration 16: Success rate = 0.95
Iteration 17: Success rate = 0.98
Iteration 18: Success rate = 0.95
Iteration 19: Success rate = 0.95
Iteration 20: Success rate = 0.97
Iteration 21: Success rate = 0.95
Iteration 22: Success rate = 0.98
Iteration 23: Success rate = 0.97
Iteration 24: Success rate = 0.92
Iteration 25: Success rate = 0.98
Iteration 26: Success rate = 0.97
Iteration 27: Success rate = 0.98
Iteration 28: Success rate = 0.99
Iteration 29: Success rate = 0.98
Iteration 30: Success rate = 1.00
Iteration 31: Success rate = 0.92
Iteration 32: Success rate = 0.98
Iteration 33: Success rate = 0.93
Iteration 34: Success rate = 0.98
Iteration 35: Success rate = 0.97
Iteration 36: Success rate = 0.95
Iteration 37: Success rate = 0.93
Iteration 38: Success rate = 0.98
Iteration 39: Success rate = 0.93
Iteration 40: Success rate = 0.92
Iteration 41: Success rate = 0.98
Iteration 42: Success rate = 0.90
Iteration 43: Success rate = 0.95
Iteration 44: Success rate = 0.95
Iteration 45: Success rate = 0.95
Iteration 46: Success rate = 0.95
Iteration 47: Success rate = 0.95
Iteration 48: Success rate = 0.98
Iteration 49: Success rate = 0.98
Iteration 50: Success rate = 0.97
Iteration 51: Success rate = 0.90
Iteration 52: Success rate = 0.95
Iteration 53: Success rate = 0.98

Show Only Latest Clear History Ask AI 29s • 2 minutes ago

```

1 import random
2 import numpy as np
3 from queue import Queue
4
5
6 N, M = 40, 40
7 Simulations = 60
8 Max_iterations = 60
9 p_open = 0.7
10 bots = ['BFS', 'A*', 'UCS', 'Greedy']
11
12
13 def generate_random_layout(N, M, p_open):
14     return [[1 if random.random() < p_open else 0 for _ in range(M)] for _ in range(N)]
15
16
17 def bfs_search(grid, start, fire_position):
18     N, M = len(grid), len(grid[0])
19     visited = [[False] * M for _ in range(N)]
20     q = Queue()
21     q.put(start)
22     visited[start[0]][start[1]] = True
23
24     directions = [(0, 1), (0, -1), (1, 0), (-1, 0)]
25
26     while not q.empty():
27         x, y = q.get()
28         if (x, y) == fire_position:
29             return True
30
31         for dx, dy in directions:
32             nx, ny = x + dx, y + dy
33             if 0 <= nx < N and 0 <= ny < M and grid[nx][ny] == 1 and not visited[nx][ny]:
34                 visited[nx][ny] = True
35                 q.put((nx, ny))
36
37     return False
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
817
818
819
819
820
821
822
823
824
825
826
826
827
828
828
829
829
830
831
832
833
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610

```

Joint Contributions:

- Report Writing (Tejaswini Abburi and Prajwal Srinivas)
- Testing (Tejaswini Abburi and Prajwal Srinivas, split equally)
- Data Analysis: Collaboratively analysed simulation data to derive insights and conclusions regarding bot performance.(Prajwal Srinivas and Tejaswini Abburi)