CAPTURE THE FLAG

Shamith Achanta
University of Illinois at Urbana-Champaign
email: shamith2@illinois.edu

ABSTRACT
This paper is to demonstrate dynamic mission planning using multi-domain systems. We create a demonstrative platform for simulation multi-domain systems (robots or agents). High level algorithms are tested on these robots by dividing them into teams and simulating them to play 'Capture the Flag (CtF)' game. Further, adversaries (in form of gray agents or randomized variables in the environment) are induced to observe how teams create and execute innovative strategies to accomplish their mission objectives.

PURPOSE/AIM & BACKGROUND
The objective of this paper is to demonstrate dynamic mission planning using robotic teams. Dynamic Mission Planning deals with planning required to make robots accomplish their mission objective. This requires robots to accomplish their task in a 'dynamic' environment (with adversaries and randomness) in optimal time. For accomplishing the mission objective, robots need algorithms (also called as policies) to operate and interact with agents in the environment. The Capture the Flag environment is used to implement and test this. Python (programming language) has been used to build this environment.

METHODOLOGY
The training of the teams uses a computerized demonstrative platform - 'Capture the Flag (CtF)' environment. This environment comprised of dividing a fair board/map into two backgrounds. Obstacles, flags and agents are placed evenly in both the backgrounds in a fair map. Algorithms (also called as policies) are tested on these teams and the decision-making of the teams are observed. Further, adversaries (in form of gray agents or randomized variables in the environment) are induced to observe how teams create and execute innovative strategies to accomplish their mission objectives.

INDIVIDUAL AND GLOBAL MEMORY
Policies, alone, cannot optimize the agents to accomplish their mission objectives. The agents need to locate the flag and trace its path to the flag or in case of adversaries, communicate with its team agents for optimizing its game plan. Hence, there is a need to explore the environment as well as communicate within a team. This problem is solved by Individual Memory of the agent and Global Memory of the team. Individual Memory gives a map of the environment explored by the agent. This helps the agent remember the path traced, observe the environment variables (agents, flags and obstacles) and take appropriate action decisively. It provides a medium to interact with environment variables. Global Memory gives a cumulative individual memory map (map of the environment explored by all the agents) of the team. In case of adversaries, agents can effectively communicate with other agents since each agent has a copy of the environment explored by the other agents.

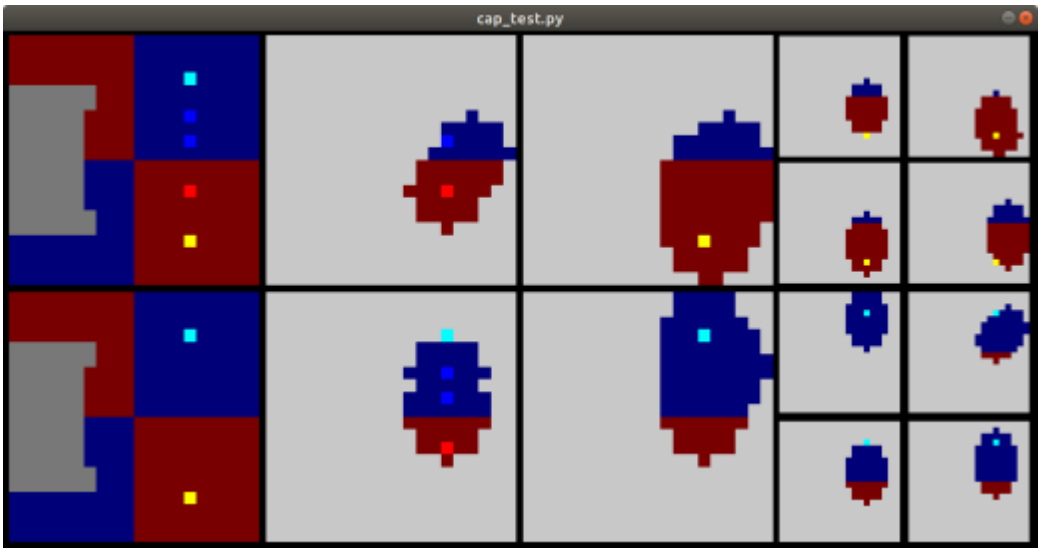


Figure 1: The leftmost maps represent fair maps. The rightmost eight maps represent the individual memory of eight agents (four in each team). The bottom four maps represent the individual memory (environment explored) by blue agents and the upper four maps represent individual memory of red agents. The maps left of individual memory are global memory (environment explored by all agents in a team) and the teamwise positions of the global memory are similar to those of individual memory (as above). Here, the blue agents move towards the yellow flag while the red agents move towards the blue flag.

POLICIES

Once the agent is able to explore the environment and communicate effectively in a team, the next task is to test a policy/algorithm on the agent and observe the decision-making and the game play of the agents. A search based heuristic – AStar and a defensive as well as counter-attacking policy – Fighter are discussed in this paper.

AStar (A*)

The mission objective of the A* policy is to capture the enemy flag. AStar policy uses the A* search based heuristic [1]. Starting from a specific starting node of a graph, A* aims to find a path to the given goal node having the smallest cost (least distance traveled, shortest time, etc.). It does this by maintaining a tree of paths originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied.

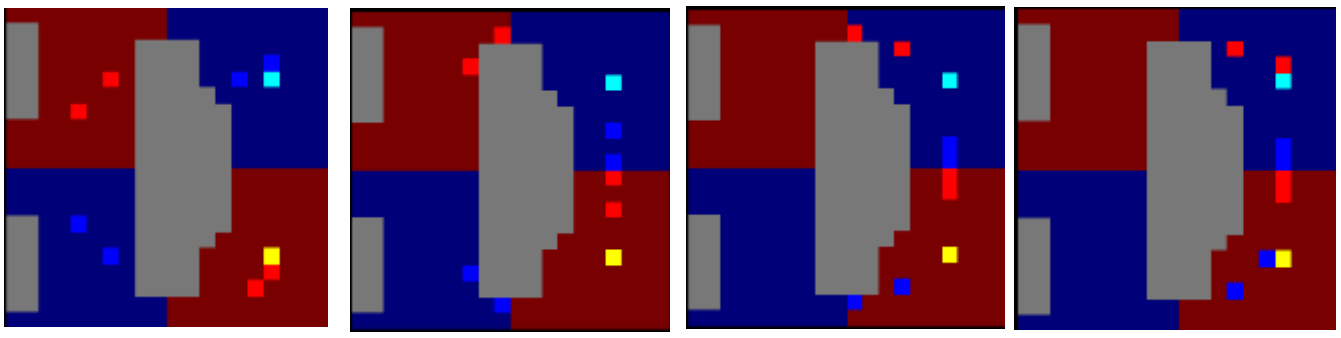


Figure 2: As seen above, the blue agents and the red agent take the minimal path to capture the enemy team flag (yellow flag is red team flag and light blue flag is blue team flag) using A* policy.

Fighter

The mission objective of the Fighter policy is to fight back against the enemy. For this, the Fighter policy teams agents into two different categories, with each team deploying a different attacking strategy. There are two types of agent: aggressive (aggr) agents and defensive (def) agents. Aggressive agents tend to capture the nearest enemy using A* policy whereas defensive agents tend to guard the flag within a prescribed guard radius and capture an enemy when the enemy ventures into their prescribed down radius.

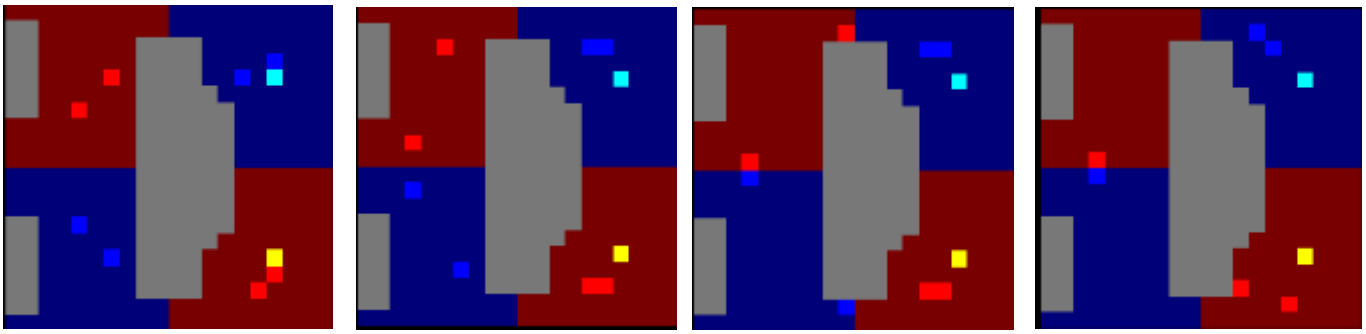
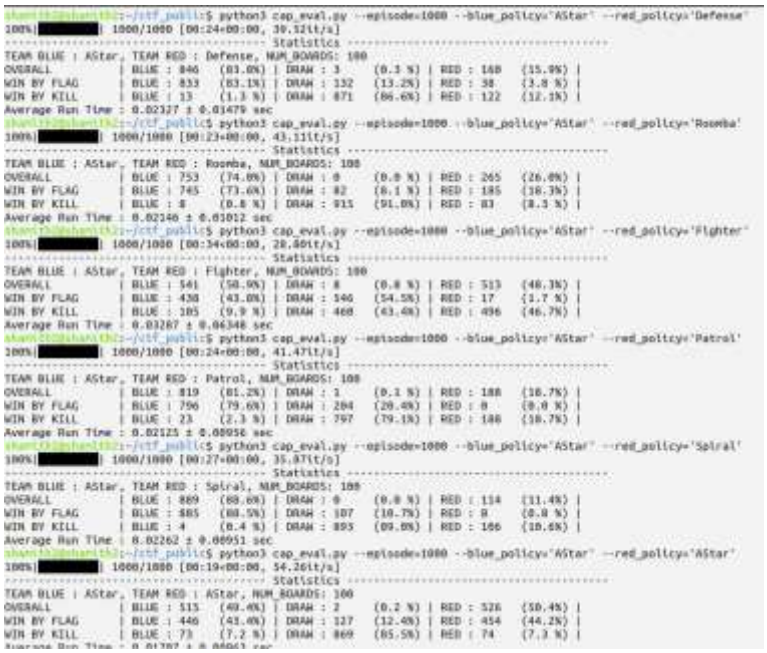


Figure 3: As seen above, two blue 'aggr' agents and the two red 'aggr' agent take the minimal path to capture the nearest enemy using A* policy (second and third map) and the remaining two 'def' agents (with guard radius of 4 units and down radius of 6 units) in each team guard the flag within the guard radius capture the 'aggr' agents when they come in the vicinity of the home flag (within their down radius) (the last map).

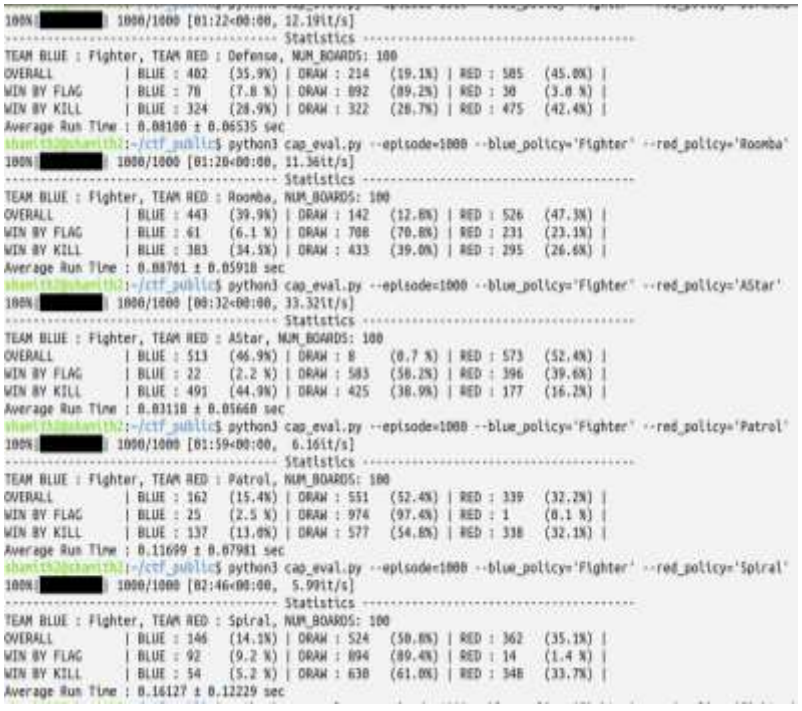
FINDINGS/RESULTS

For the analysis, 100 different fair boards/maps were used for 1000 different episodes and the results were analyzed. This is to make sure that the results are unbiased. On analyzing, A* policy was found to be highly capable of capturing the enemy flag. It was found to win 85% of the times irrespective of the policy of the enemy team. On the other hand, Fighter was found to be highly dependent on the number of aggressive agents and defensive agents. Fighter can be customized to defeat any policy by adjusting the number of aggressive agents and defensive agents. For instance, having all aggressive agents makes Fighter similar to A* whereas having all defensive agents makes Fighter similar to an all defensive policy.



```
$ python3 cap_eval.py --episode=1000 --blue_policy='AStar' --red_policy='Defense'
100%| 1000/1000 [00:24<00:00, 39.32it/s]
---------- Statistics ----------
TEAM BLUE : AStar, TEAM RED : Defense, NUM_BOARDS: 100
OVERALL          | BLUE : 846   (83.6%) | DRAW : 3     (0.3 %) | RED : 168   (15.9%) |
WIN BY FLAG      | BLUE : 833   (83.1%) | DRAW : 132   (13.2%) | RED : 38    (3.8 %) |
WIN BY KILL      | BLUE : 13    (1.3 %) | DRAW : 871   (86.6%) | RED : 122   (12.1%) |
Average Run Time : 0.02327 ± 0.01479 sec
$ python3 cap_eval.py --episode=1000 --blue_policy='AStar' --red_policy='Roomba'
100%| 1000/1000 [00:23<00:00, 43.11it/s]
---------- Statistics ----------
TEAM BLUE : AStar, TEAM RED : Roomba, NUM_BOARDS: 100
OVERALL          | BLUE : 753   (74.8%) | DRAW : 0     (0.0 %) | RED : 265   (26.0%) |
WIN BY FLAG      | BLUE : 745   (73.6%) | DRAW : 82    (8.1 %) | RED : 185   (18.3%) |
WIN BY KILL      | BLUE : 8     (0.8 %) | DRAW : 911   (91.8%) | RED : 83    (8.3 %) |
Average Run Time : 0.02140 ± 0.01012 sec
$ python3 cap_eval.py --episode=1000 --blue_policy='AStar' --red_policy='Fighter'
100%| 1000/1000 [00:34<00:00, 28.86it/s]
---------- Statistics ----------
TEAM BLUE : AStar, TEAM RED : Fighter, NUM_BOARDS: 100
OVERALL          | BLUE : 541   (50.9%) | DRAW : 8     (0.8 %) | RED : 513   (48.3%) |
WIN BY FLAG      | BLUE : 438   (43.8%) | DRAW : 546   (54.5%) | RED : 17    (1.7 %) |
WIN BY KILL      | BLUE : 105   (9.9 %) | DRAW : 468   (43.4%) | RED : 496   (46.7%) |
Average Run Time : 0.03287 ± 0.06348 sec
$ python3 cap_eval.py --episode=1000 --blue_policy='AStar' --red_policy='Patrol'
100%| 1000/1000 [00:24<00:00, 41.47it/s]
---------- Statistics ----------
TEAM BLUE : AStar, TEAM RED : Patrol, NUM_BOARDS: 100
OVERALL          | BLUE : 819   (81.2%) | DRAW : 1     (0.1 %) | RED : 188   (18.7%) |
WIN BY FLAG      | BLUE : 796   (79.6%) | DRAW : 204   (20.4%) | RED : 0     (0.0 %) |
WIN BY KILL      | BLUE : 23    (2.3 %) | DRAW : 797   (79.1%) | RED : 188   (18.7%) |
Average Run Time : 0.02125 ± 0.00856 sec
$ python3 cap_eval.py --episode=1000 --blue_policy='AStar' --red_policy='Spiral'
100%| 1000/1000 [00:27<00:00, 35.87it/s]
---------- Statistics ----------
TEAM BLUE : AStar, TEAM RED : Spiral, NUM_BOARDS: 100
OVERALL          | BLUE : 889   (88.6%) | DRAW : 0     (0.0 %) | RED : 114   (11.4%) |
WIN BY FLAG      | BLUE : 885   (88.5%) | DRAW : 107   (10.7%) | RED : 8     (0.8 %) |
WIN BY KILL      | BLUE : 4     (0.4 %) | DRAW : 893   (89.8%) | RED : 166   (10.6%) |
Average Run Time : 0.02262 ± 0.00951 sec
$ python3 cap_eval.py --episode=1000 --blue_policy='AStar' --red_policy='AStar'
100%| 1000/1000 [00:19<00:00, 54.26it/s]
---------- Statistics ----------
TEAM BLUE : AStar, TEAM RED : AStar, NUM_BOARDS: 100
OVERALL          | BLUE : 515   (49.4%) | DRAW : 2     (0.2 %) | RED : 526   (50.4%) |
WIN BY FLAG      | BLUE : 446   (43.4%) | DRAW : 127   (12.4%) | RED : 454   (44.2%) |
WIN BY KILL      | BLUE : 73    (7.2 %) | DRAW : 869   (85.5%) | RED : 74    (7.3 %) |
```

Figure 4: Results for A* Policy tested on team Blue. The last result (A* vs A*) shows that the results are unbiased.



```
100%| 1000/1000 [01:22<00:00, 12.19it/s]
---------- Statistics ----------
TEAM BLUE : Fighter, TEAM RED : Defense, NUM_BOARDS: 100
OVERALL          | BLUE : 402   (35.9%) | DRAW : 214   (19.1%) | RED : 505   (45.0%) |
WIN BY FLAG      | BLUE : 78    (7.8 %) | DRAW : 892   (89.2%) | RED : 30    (3.0 %) |
WIN BY KILL      | BLUE : 324   (28.9%) | DRAW : 322   (28.7%) | RED : 475   (42.4%) |
Average Run Time : 0.08100 ± 0.06535 sec
$ python3 cap_eval.py --episode=1000 --blue_policy='Fighter' --red_policy='Roomba'
100%| 1000/1000 [01:28<00:00, 11.36it/s]
---------- Statistics ----------
TEAM BLUE : Fighter, TEAM RED : Roomba, NUM_BOARDS: 100
OVERALL          | BLUE : 443   (39.9%) | DRAW : 142   (12.8%) | RED : 526   (47.3%) |
WIN BY FLAG      | BLUE : 61    (6.1 %) | DRAW : 708   (70.8%) | RED : 231   (23.1%) |
WIN BY KILL      | BLUE : 383   (34.5%) | DRAW : 433   (39.0%) | RED : 295   (26.6%) |
Average Run Time : 0.08701 ± 0.05918 sec
$ python3 cap_eval.py --episode=1000 --blue_policy='Fighter' --red_policy='AStar'
100%| 1000/1000 [00:32<00:00, 33.32it/s]
---------- Statistics ----------
TEAM BLUE : Fighter, TEAM RED : AStar, NUM_BOARDS: 100
OVERALL          | BLUE : 513   (46.9%) | DRAW : 8     (0.7 %) | RED : 573   (52.4%) |
WIN BY FLAG      | BLUE : 22    (2.2 %) | DRAW : 583   (58.2%) | RED : 396   (39.6%) |
WIN BY KILL      | BLUE : 491   (44.9%) | DRAW : 425   (38.9%) | RED : 177   (16.2%) |
Average Run Time : 0.03118 ± 0.01660 sec
$ python3 cap_eval.py --episode=1000 --blue_policy='Fighter' --red_policy='Patrol'
100%| 1000/1000 [01:59<00:00,  6.16it/s]
---------- Statistics ----------
TEAM BLUE : Fighter, TEAM RED : Patrol, NUM_BOARDS: 100
OVERALL          | BLUE : 162   (15.4%) | DRAW : 551   (52.4%) | RED : 339   (32.2%) |
WIN BY FLAG      | BLUE : 25    (2.5 %) | DRAW : 974   (97.4%) | RED : 1     (0.1 %) |
WIN BY KILL      | BLUE : 137   (13.0%) | DRAW : 577   (54.8%) | RED : 338   (32.1%) |
Average Run Time : 0.11699 ± 0.07981 sec
$ python3 cap_eval.py --episode=1000 --blue_policy='Fighter' --red_policy='Spiral'
100%| 1000/1000 [02:46<00:00,  5.99it/s]
---------- Statistics ----------
TEAM BLUE : Fighter, TEAM RED : Spiral, NUM_BOARDS: 100
OVERALL          | BLUE : 146   (14.1%) | DRAW : 524   (50.8%) | RED : 362   (35.1%) |
WIN BY FLAG      | BLUE : 92    (9.2 %) | DRAW : 894   (89.4%) | RED : 14    (1.4 %) |
WIN BY KILL      | BLUE : 54    (5.2 %) | DRAW : 630   (61.0%) | RED : 348   (33.7%) |
Average Run Time : 0.16127 ± 0.12229 sec
```

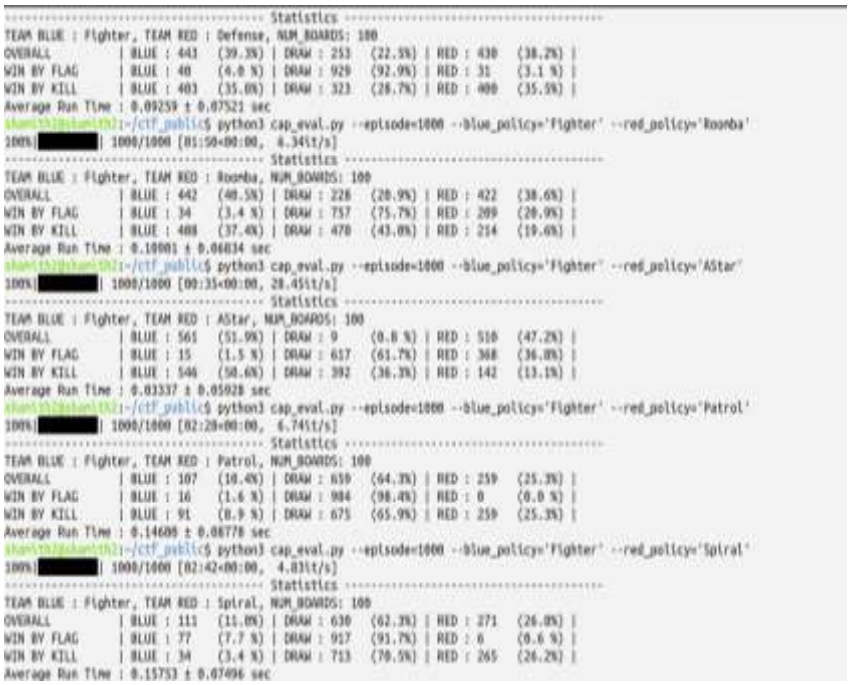


Figure 5: Results for Fighter Policy tested on team Blue with two aggressive agents and two defensive agents and one agressive agent and three defensive agents respectively. As seen above, Fighter with one aggressive agents and three defensive agents defeats A* while Fighter with two aggressive agents and two defensive agents doesnot.

CONCLUSION

Individual Memory and Global Memory support policies in making the agents accomplish their mission objective. Individual Memory gives a map of the environment explored by the agent helping it observe and interact with the environmental variables. Global Memory, on the other hand, gives a map of the environment explored by all the agents in the team. It further aids agents communication in a team in case of induced adversaries and randomness in the environment. Further, on analyzing the results on the policies, A* policy was found to be highly capable of capturing the enemy flag. It was found to capture the flag in O(1) time (constant Big-O). On the other hand, Fighter was found to be highly dependent on the number of aggressive agents and defensive agents. Fighter can be customized to defeat any policy by adjusting the number of aggressive agents and defensive agents. For instance, this happens in Figure 5 since more defensive agents means more difficulty for a enemy agent to penetrate the guarding radius and capture the flag.

ACKNOWLEGDMENTS

I would like to thank Prof. Huy Trong Tran, Research Assistant Professor, Aerospace Engineering at UIUC, for giving me the opportunity and mentoring my research. I would like to thank Seung Hyun Kim for guiding me during my research. I would like to further thank Defence Advanced Research Projects Agency (DARPA) and Illinois Space Grant Consortium for giving me an opportunity to conduct research and fund my research. Their help has been crucial in completing my research.

REFERENCES

[1] Wikipedia. “A* search algorithm”. https://en.wikipedia.org/wiki/A*_search_algorithm.