Final Project Report
Surviving a Zombie Apocalpyse Using Map-Based Multi-Agent Planning

Joshua Ramos, Sarah Stapleton, Anna Yu

May 4, 2023

1 Abstract

The zombie apocalypse can be a difficult place to navigate as a group. With brain eating monsters everywhere, when your group gets split up it is important to get everyone where they need to be without losing anyone. In order to simulate this scenario, we have a map with obstacles, searchers, lost humans, and zombies which act as dynamic obstacles. The map is discretized into a grid and each grid space contains an obstacle, a lost human, a searcher, a zombie, or is a free space. Levels are randomly generated based on the given difficulty which decides number of obstacles, zombies, and searchers/lost humans. We simulate the zombies as moving randomly, unless a searcher is within their sight, then they will greedily move towards them. The lost humans are immobile and invisible to the zombies, but those searching know their location to simulate gps trackers. Once found, the lost humans become searchers. Searchers have two modes: searching for a lost human and searching for the exit point. In order to accomplish both of these tasks, an LRTA* algorithm is used to prioritize the immediate surroundings. We found that having an LRTA* horizon of 3 to be most effective. Using ideal methods, the simulation was able to complete with all humans escaping 75% of the time, with an average planning execution time of 0.1912 milliseconds, and an average total simulation time of 1.2417 seconds.

2 Introduction

The zombie apocalypse is here and we are now living in a zombie-ridden world. We're with a group of friends but somehow we're split up, some with the locations of those who are lost. The lost people have found a way to hide in place to avoid the detection of nearby zombies. The searchers are all tasked to find lost peers, only after which they can successfully exit the map. Would you be able to find all your friends and escape alive? Or will you face the wrath of hungry zombies?

With a goal of multi-agent planning, the aim of this game is to mainly implement and show the LRTA$*$ algorithm at work.

Figure 1: Images of a Minecraft Steve, Villager, and Zombie

3 Method/Algorithm

3.1 Searcher

The algorithm for searchers is divided up into 2 scenarios: searching and heading towards the exit.

In the searching case, while there are still lost people hiding, searchers are assigned the nearest lost person's location as their goal. It uses the LRTA$*$ algorithm to head towards the lost person while avoiding getting eaten by zombies. This is achieved by treating zombies as dynamic obstacles, such that instead of modifying the LRTA$*$ planner to maximize the distance of a searcher from a zombie, we can treat the zombie's C-Space Transform as a moving obstacle that the searchers cannot walk into and will avoid with the aid of LRTA$*$ extended horizon.

The searcher's goal changes once it is able to find the lost person it was assigned. Consequently, if there are still lost people hiding who have not yet been assigned to any searchers, then the nearest lost person to the searcher will be assigned to it. However, in the case where all lost people have been assigned to searchers, the rest of the searchers will head for the exit. This avoids aliasing searchers in our plan.

The planning of the searchers is centralized. In our algorithm, every searchers action is determined in one full swing, per time step, and then applied at once. This allows us to have more control over the assignment of searchers to lost people and it emulates the scenario where our searchers can converse among one-another over radio to carefully determine their next moves to best guarantee everyone's safety.

State	(x,y)
Goal 1	Searcher(x,y) == LostPerson(x,y)
Goal 2	Searcher(x,y) == Exit(x,y)
Cost	Distance from current grid space to next (8-connected)
Heuristic	max(dx,dy)*0.4*min(dx,dy)

LRTA$*$ algorithm when expanding $N \geq 1$:

1. *expand N states*
2. *update h-values of expanded states by Dynamic Programming(DP)*
   (a) *initialize:* $h(s\prime) = \infty$ *for all states in closed list*
   (b) *repeat:* $h(s\prime) = min_{s \in succ(s\prime)} c(s\prime, s) + h(s)$
3. *move on the path to state* $s = argmin_{s\prime \in OPEN} g(s\prime) + h(s\prime)$

3.2 Zombie

The algorithm for zombies is also divided into 2 scenarios: random and chase.

When the zombie doesn't detect any searchers within its vicinity, defined with a radius of 5 block spaces away, the zombie will randomly move within a 4-connected grid.

When the zombie does detect a searcher within its proximity, the zombie will greedily chase it while also using a 4-connected grid.

The Zombie-Planner is a decentralized planner. We implemented this by running each zombie as its own process/thread in the simulation, such that it can act independently of other zombie's targets or positions. The time is takes for a zombie to make a decision and move is 50ms. (Zombie-Planner Execution Time)

State	(x,y)
Goal	Zombie(x,y)==Searcher(x,y)
Cost	(not applicable)
Heuristic	(not applicable)

3.3 Lost Person

Lost people begin hidden and static. They do not move and zombies do not see or chase after them.

Once a lost person is found by a searcher, the lost person becomes a searcher and takes on the searcher algorithm. Searcher tasks would then be assigned.

State	(x,y)
Goal	(only relevant when becoming a searcher)
Cost	(only relevant when becoming a searcher)
Heuristic	(only relevant when becoming a searcher)

3.4 Gameplay

The simulation visualizer was setup using the Pygame Python library.

The map size, number of searchers, number of lost people, and number of zombies can vary. The obstacles are randomly generated each time depending on map size each time to test the robustness of the implemented algorithms. An example of a Level 2 map can be seen on the right with blue, green, and red dots representing searchers, lost people, and zombies respectively. On simpler levels with smaller maps, a Minecraft Steve, Villager, and Zombie represent searchers, lost people, and zombies respectively.

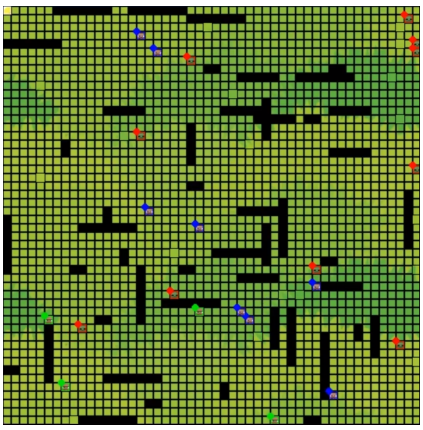

Figure 2: Still frame of apocalypse visualizer

Zombies are implemented to move slower than searchers to emulate their relatively sluggish movement, like in movies or other video games.

4 Results

The algorithms are implemented through a small game visualizer. Different map sizes and number of searcher, lost people, and zombies show different "levels." A test was done to experimentally find what the optimal LRTA$*$ horizon was to maximize the success rate of the planner. A success is defined when all lost people are found by searchers and no one is eaten by a zombie. 100 trials were performed for each horizon of 1, 3, and 10. The testing environment is described as the following:

Environment created for tests:

- Obstacle: 15
- Map Size: 30x30
- Searchers: 1
- Lost: 5
- Zombies: 3

# Trials	N-Step Horizon	Success	Avg Sim Time	Searcher Plan Time	Zombie Plan Time
100	1	67%	1.1642s	0.1861ms	50ms
100	3	75%	1.2417s	0.1912ms	50ms
100	10	60%	2.2041s	0.2557ms	50ms

Table 1: Varying algorithm horizon test results

From the results, it can be observed that by allowing a horizon of 3, the success rate of the planner is on average 75%. Allowing an expansion of 1 and 10 decreased the success rate by over 8%. When analyzing other variables, the average simulation time and searcher planner time both increase as the horizon increases. This is expected as there is more computation and planning necessary when increasing the allowed number of expansion.

Overall, the LRTA$*$ implementation for searchers is very effective even in a scenario where one agent had multiple goals.

5 Discussion

During implementation, there were many approaches we could have taken for implementing each planner type. We decided to use a centralized planner for the Searcher algorithm and a decentralized planner for the Zombie algorithm primarily because we wanted to highlight our utilization of LRTA$*$ in the Searcher planner. Although the Searcher planner is directing multiple agents, which would normally consume a bit of time, while the Zombie planner is distributed to each Zombie entity/thread to run independently of one another, our searchers still succeed in escape 75% of the time because of its quick action computation time.

In testing, we found that increasing the horizon of our LRTA$*$ resulted in an increase in search time, and as a result, total computation time. Moreover, it also resulted in a lower success rate. Over careful observation, it is evident that our planner overestimated the goal states and updated unnecessary states. And resulted in our searchers appearing to wander aimlessly at times. This fault in our implementation is most likely due to faulty updating of the heuristic values of our searcher's map during horizon expansion. This is something we will improve upon, and once improved should show significant enhancement in our plan optimality. Our main goal with the extended horizon approach was to achieve corner-avoidance. The idea that we can avoid walking into corners and getting trapped by zombies.

On the other hand, with a short horizon, our planner suffered in success rate due to its susceptibility to cornering. Thus, after tuning, we determined that a horizon of 3 states out was best fit for our approach. It provided us with a 75% success rate, very similar average execution times to the 1 step horizon, and was able to more quickly escape/avoid corners similar to the 10 step horizon.

Moving forward, our simulation can branch out in many directions. We envisioned decentralizing the searcher planner and have each searcher work independently to emulate a divide and conquer based-approach. Additionally, we believe that expanding upon our current Searcher-Planner by introducing searcher-redirection property, such that no lost person is left behind, even if the searcher he was assigned was eaten. Because at the moment, if a searcher was a assigned a lost person, but they die on their search, the rest of the searchers will leave that lost person for dead. Gruesome, right? But yes, we would like to extend our algorithm to take into account these liabilities in our approach. Moreover, down below, is an explicit list of future improvements:

5.1 Future Improvements

- As mentioned before, it is noticed that LRTA$*$ is not the fastest at updating heuristics as it only changes when making the next move. An interesting exploration would be investigating how to find ways of improving the update property of LRTA$*$. This may include researching extensions of LRTA$*$.

- Implementing fun upgrades throughout the map will increase the complexity of the searcher's cost function and the overall implementation of the searcher's algorithm.
- Optimizing the distribution of the lost people to searchers would also be useful. This will minimize the risk of a lost person never being found if its searcher was eaten by a zombie.
- Improving the avoidance property of the LRTA$*$ implementation would help searchers avoid being cornered by a zombie or zombies. As LRTA$*$ slowly updates the map's heuristics, a greedy zombie may get to the searcher before the searcher can successfully find its way out of a corner.

6 Conclusion

For multiple agents traversing an environment, LRTA$*$ is able to efficiently plan around dynamic obstacles. There were 3 major components in the simulation: searchers, zombies, and lost people. Each entity had its own algorithm implemented. The LRTA$*$ algorithm was implemented for searchers. Random movement followed by a greedy search when a searcher was within its proximity was implemented for zombies. And lost people were stationary when hidden but then took on the role of a searcher when they were found. This combination created an interesting visualization of the algorithm at work and highlighted the algorithms advantages and disadvantages in this apocalyptic environment.

A test was performed by varying the horizon of the LRTA$*$ algorithm. At each horizon, 100 trials were run and a success rate, average total simulation time, searcher planner time, and zombie planner time was recorded. The results showed that a horizon of 3 was the optimal in maximizing the average planner success rate at 75%.

7 References

16-350 planning techniques for Robotics. (n.d.). Retrieved April 27, 2023, from https://www.cs.cmu.edu/~maxim/classes/robotplanning/

Pygame front page. Pygame Front Page - pygame v2.4.0 documentation. (n.d.). Retrieved April 27, 2023, from https://www.pygame.org/docs/

Subprocess - subprocess management. Python documentation. (n.d.). Retrieved May 3, 2023, from https://docs.python.org/3/library/subprocess.html

Threading - thread-based parallelism. Python documentation. (n.d.). Retrieved May 3, 2023, from https://docs.python.org/3/library/threading.html

Time - Time Access and conversions. Python documentation. (n.d.). Retrieved May 3, 2023, from https://docs.python.org/3/library/time.html