Assignment 2 - Part 4

Angela Luo - 50731

May 2020

Abstract

Transitioning from a static game world to a dynamic one, the following sections describe the steps I took to adapt my search to be more robust. Section 5 lists the functions mentioned in this report for clarity.

1 Task Recalculation

Since the objects in the world can move around, we have to make sure that the object we are trying to get to is still there, especially next to the agent after a successful $move/1$ call. This can be checked using $map_adjacent(P_n, \_, O)$ as a success criteria after $move/1$ call. If this or $move/1$ fails, $get_identity/3$ is called with unchanged arguments as a fallback.

2 Atomized Movement

Originally, the A* algorithm calculates the shortest path from the agent's current position to the desired position, P_n as a list of positions, $[P_1, P_2, ..., P_n]$. The agent moves in one call, $agent\_do\_moves(Agent, [P_1, ..., P_x])$. This will fail if there is another agent or a change in environment that results in a wall blocking the path. To solve this problem, the movement of the agent is altered to be atomic, i.e. moving one space in the list at a time. To check that the next position in the queue is still available, the agent calls $agent\_do\_moves(Agent, [P_x])$, if this fails, find an alternative path as a fallback.

3 Path Splicing

Instead of re-computing the shortest route, we can reuse the original path. In a list of positions that describes the shortest route, $[P_1, P_2, ..., P_n]$, if at the time of execution, P_2 is a wall and agent cannot move to that position, $get\_alternative\_path/1$ is called, which finds the shortest path to the next available position in $[P_3, ..., P_n]$. If there is a shortest path $[P_x, P_y, P_3]$ to P_3 for example, the new route for the agent is then a concatenation of the new path and the original path, $[P_x, P_y, P_3, ..., P_n]$.

4 Conclusion

In conclusion, the solution that I have implemented relies on creating safety nets for when functions fail, thus creating a more robust system even when there are other active agents in the game.

5 List of Functions and Their Functions

- $get\_alternative\_path/1$ - recursively calls on it self until the shortest path to the next available position in the remaining path is found or fail if $Path$ is empty.
- $get\_identity/3$ - contains main computation that determine what the agent will do next depending on its current energy.
- $move/1$ - takes a list of positions, attempts to move the agent, calls itself with left over path.