Evagelos Petropoulos

U75564437

Project 1 Part 3

Problem Formulation

Starting off in this project, the first thing I did was change how state was represented in the code. Since there are now two players at all times competing for resources, the state had to represent that. The position, backpacks, and what materials an agent had delivered as part of the state now had to be divided between Player A and Player B. Also, I implemented a turn variable so that way whose players' turn it was could be recorded for action and resource purposes.

My action functions(move_agent and get_neighbor) are very similar to before in theory, they just needed to be edited in order to support two players in a state this time instead of just one. Turn is constantly checked to make sure the right player will move. The get_neighbor function is the main one responsible for determining next states available and which are viable and optimal, and if so, calls move_agent to go to them. This includes checking the Conflict Rule, which has the turn player having priority over which tile it wants to occupy over the other player trying to access it. The four viable moves a player can make are up, down, left, and right, as long as the tile it is trying to access is on the map and not occupied by the other player

The utility function is located in the minimax file. It checks the state and returns the zero-sum payoff described in class and in project requirements. This is defined as Player A winning if the difference between Player A's delivered total and Player B's delivered total is positive, Player B winning if it is negative, and a tie if it is zero.

Heuristic Design

A heuristic function estimates and calculates the cost and reward of each player's proximity to both resources and the base, if they are carrying resources in their backpacks ("Heuristics & Approximate Solutions | AP CSP (Article)"). It incentivizes carrying resources

back to base once any are in their backpack, by attaching a strong negative score to other states that would not lead to their respective base. If the player has no resources, then it incentives going towards states that would let it reach a tile to pick up a resource. The Manhattan Distance equation is again used in the heuristic to help carry this out.

I created the heuristic like this so it could be admissible(never overestimating the actual cost needed), so it could pick up resources and return them to base quickly so a player would not be interrupted on the path to a second resource, and feature again the zero-sum rule.

Results

Minimax search algorithms use recursion to allow a player character to move optimally against an opponent, where moves can be predicted depending on the depth allowed("Artificial Intelligence"). Due to this, however, a large number of states can be generated that create a great computational overhead and reduced runtime. Alpha-beta pruning is used here to trim down the possible number of states that are considered by the algorithm to keep the algorithm running smoothly and without too much difficulty. States that are not optimal are not considered by the algorithm.

Discussion

The heuristic was designed to optimize resource collection and then immediately return to base right after. There is an issue where sometimes a player with the minimax algorithm, after collecting either one or two resources, will still keep circling another resource tile instead of returning to base. I am not sure why it does this, but it could be due to an over emphasis on base distance. The random agent does not have this behavior.

Despite the project requesting a depth of 3-5, I found a slightly larger depth helped noticeably with the search algorithms. I had depth set to 11 for both the minimax vs minimax simulation and minimax vs random simulation. Depth did not have a noticeable effect on runtime on the minimax vs random simulation till about a depth of 15, where it slowed down the runtime significantly, to about a second needed per turn. Depth was also set to 11 for minimax vs

minimax. After this, each increase in depth added half a second or more of time needed for each turn to be computed by the program. I believe needing additional depth might be due to a less than optimal minimax or heuristic function. I managed to improve them from needing the depth of 15 to function to 11 by working on the alpha-beta pruning technique.

Works Cited

“Heuristics & Approximate Solutions | AP CSP (Article).” *Khan Academy*, www.khanacademy.org/computing/ap-computer-science-principles/algorithms-101/solving-hard-problems/a/using-heuristics.

“Artificial Intelligence .” *Http://Www.Rnlkwc.Ac.In/*, www.rnlkwc.ac.in/pdf/study-material/comsc/Ai.pdf. Accessed 2025.