Student Information

Course: COMP90054 AI Planning for Autonomy

Semester: Semester 2, 2024

Student:

Yu-Tse Ling - 1466145 – YUTSE

Self Evaluation

Part 1

What are the design decisions you made? (3 Marks)

The technique I decided to utilize for designing the agent is the heuristic search. As the goal state for Azul is to find the actions that can sum up to the highest score to beat the competitors, the function I chose to use is the Greedy Best First Search. It tends to search for all possible actions after each round of the game and prioritises selecting the actions that promise to obtain the higher score regardless of whether it is the shortest path. To implement the problem as the state-space model, the initial state for the agent, which I treat as the rootstate in the coding program, is starting with observing the pattern board and let the model to plan what to do in the first round. For the first action, I simply assign the agent to find the tiles that can fill up the pattern line to achieve the first point since the pattern board is empty at the first round. In addition, the agent will avoid actions that result to the floor line unless it is mandatory as to try lessening the points being deducted. To ensure little probability for the agent to input tiles to the floor line, I added more weights in score calculating function to the penalties it gets when it is necessary for the agent to append tiles to floor line.

For the score calculation, I separated the scores to 3 parts: increased_points, decreased_points, and bonus_points in the azulheuristic() function. The increased_points represents the extra points the agent gets when the tiles are adjacent to other tiles horizontally or vertically. The decreased_points indicates the penalties it gets when there are tiles remaining on the floor line. Lastly, the bonus_points can be obtained by completing rows, columns, or the same set of tiles. The summation of the score is the key of the SelectAction() function to return a valid and best action in each turn in every round. If the latter calculated score of the action is bigger than the current max score, it updates the max score and returns the action as the best action.

Part 2

What are the challenges you have **solved** when implementing that technique? (1 Mark)

Once my designed agent started to play, I discovered that it kept selecting the tiles and add to the floor line even when all the pattern lines are empty. Therefore, I added another decision tree to the original SelectAction() function to restrict the agent from appending the tiles to the floor line if it isn't mandatory. Also, I constrained the agent to select only the tiles that will lead to at least 1 point as the first action in the first round to secure the points. I implemented the technique successfully, and the agent can beat the staff agent in 6 rounds out of 10 rounds.

However, I still found some errors. When my agent is done selecting the tiles that can achieve points for filling up the pattern lines, it takes the smaller amount of tiles and put to the pattern lines with more tile spaces, for example, it puts 1 white tile to the pattern line with 5 spaces even when there are 3 blue tiles remaining in the factory or the center for it to choose. Choosing the 3 blue tiles and add to the 5 spaces pattern line is obviously the better option since it requires less tiles to fulfill on the next round. Therefore, I tried to assign the agent to focus on completing the pattern as the first action during the rest of the rounds, but it ends up with a worse outcome. Additionally, I plan on making the agent focus on completing the rows, columns, and pattern sets to gain bonus points. The results appeared to be winning 4 out of 10 games. The previously mentioned conditions all sound reasonable to me to win the game, I considered that it was because of the additional constraints interrupting the functionality of greedy best first search. It also suggests that this algorithm might not be the optimal solution to the problem, so I decided to change to another technique to enhance better performance for my agent.

Part 3

What are the future improvements you plan to make for your approach if you have more time? (1 Mark)

For the next group submission, I plan to change my Greedy Best First Search to Monte Carlo Tree Search algorithm. Its decision making process contains random sampling and statistic evaluation, which implies that it can learn deeply from experience. Unlike Greedy Best First Search, it explores and exploits widely to every nodes in the states to check which will be the optimal path. In the SelectAction() function, instead of only checking the previously deepcopied state in my original program, my intention will be allowing the agent to backpropagate if the first randomly selected action isn't the best action after learning. Monte Carlo Tree Search can contain more complex functions for learning, so it would be a more suitable solution to solve multiple limitations and develop a comprehensive game strategy.

In the learning procedure, I plan on implementing the constraints such as focusing on completion of rows, columns, and sets of pattern tiles. Especially, prioritising completing the columns because it gets higher bonus points. While completing the columns, the agent can get more points if the tiles are adjacent, so that's why I choose to prioritise

completing columns before completing sets of pattern tiles. I also plan on adding one more constraint about having the ability to observe the pattern board of the competitor and try to block its way to victory. Lastly, collaboration is beneficial in enhancing the knowledge to the game, so our team plans to discuss about our current implementation and how we can improve in the future.