Bryan Kim
Bjk3yf
CS 4710 AI
Floryan

HW3 Analysis Report

Algorithm Implemented:

My algorithm was based on a simple non-absorbing Markov Chain with all actions being deterministic. I have determined that for the game “Ticket To Ride”, there are 3 states every player will go through. The first state (S1) is when the player has to draw a card. The second state (S2) is when the player is able to claim a route, and the final third state (S3) is when the tickets in the player’s hand are completed. Associated with these states are specific actions that can be done. S1 has the designated action of a1, where the player takes a card. S2 has the action of a2, where the player claims a route. A3 is with S3, which is to obtain more train cards. The figure below shows how the AI will choose its next step based on the circumstances of the board.

$$\begin{tikzcd}
S1 \arrow[loop above, "a1"] \arrow[r, "a1"] & S2 \arrow[loop above, "a2"] \arrow[r, "a2"] \arrow[l, bend left, "a2"] & S3 \arrow[l, bend left, "a3"] \arrow[ll, bend left=60, "a3"]
\end{tikzcd}$$

The makeMove() function starts out by checking if the destination ticket list in the player’s hand is empty. This checks if the player is in S3 and will appropriately take more cards to have the player keep obtaining points. Then the AI will check if the ideal path for the destination ticket was blocked by the opponent. If the path was indeed blocked and no longer can be obtained by the AI, the path will be updated to the next shortest path for the current ticket. I had modified the

breadth first search method, “shortestPathcost”, found in the Routes class to output the shortest path rather than just the minimum cost.
The AI will then check whether it is in S1 or S2 by seeing if it can afford any routes in the current path. If the state is S2, it will claim the route with the respective color of the route. If there is no route currently affordable, S1 will be the state and will choose from the 5 faceup cards or deck based on what is needed for the next route.

Testing the AI:

To test the AI, I had personally played the AI to check for base cases. One method of playing was to always take a route from the path the AI was thinking of claiming. This is to always have the AI change their paths whenever possible. This proved to be inefficient for myself as I would constantly run out of resources and the AI would catch up and eventually take the entire path for the ticket, leaving myself to lose. Another strategy was to play for the most amount of points as a player. This strategy always led to most games leading toward the AI winning. The pathfinding from the AI has shown to be a great asset for it to decide the right moves, but also a hidden factor of how sometimes the randomness of the cards would often hinder both players at getting the hand they need at the right time. Testing slowly with myself as a player helped with debugging the AI, allowing myself to see the flaws that still needed to be ironed out.
Where the AI seems to fail or fault is when it is close to running out of train pieces. The AI’s main goal is to focus on finishing the ticket’s path so that it would avoid making more optimal choices based on the situation of the board. When the AI would go against itself, one AI would always be hindered since their routes would always eventually get blocked due to their own routing. Thus the AI would be at a standstill and not have much to do about it. Looking back on this now, I can see that the AI is limited and often is in a greedy mindset. The next steps I would take to the AI is to add some simulated annealing to help ease in the decision process to claim random routes near the end of the game to not get stuck in this situation.