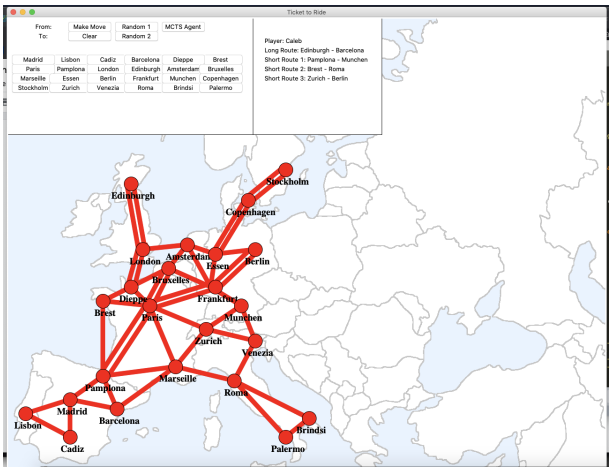
Monte Carlo Search Tree Ticket to Ride Agent

Caleb Johnson
Artificial Intelligence
Salt Lake City, UT
calebdeejohnson@gmail.com

Introduction

For my project, I decided to design a Monte Carlo Tree Search agent to play the game Ticket to Ride. The basis of the game is to build trains, finish routes, and compete to see who can achieve the highest score. I spent about 20 hours on the project, with about 3 spent doing initial research, about 8 being setting up the platform and code base to just be able to play Ticket to Ride/making an evaluation function, etc. The rest of the time was spent creating different agents. A lot of prototyping was done on how fleshed out to make the game and we reverted back. I then explored a couple of different agents that will be discussed in the paper.



Game Simplifications

For the purpose of this project, I made the following simplifications to the traditional game of Ticket to Ride. In essence, I have completely changed much of the core of the game, however we maintain the goal of route building and having the longest acyclic "line".

- There is no drawing of cards and all lines between two cities have the same cost. At each turn you simply choose to build a line between two connected cities. In the future I would want to make an exploitation / exploration aspect where you either draw or play, but for the scope of this project I was not able to.
- You simply get assigned your longest route and shorter routes randomly and do not get a choice over what they are. I hope to fix this later but again, and should genuinely not be too hard, but for this project, that is what I decided upon.
- I only included a little over half of the possible cities and routes from the European version of ticket to ride. I tried adding them all, but the state space became too larger to run and test reasonably on my old MacBook Pro.
- There is no way to build stations to help connect routes that you were too slow on.

Score Evaluation

I explored two different ways of determining my reward for my Monte Carlo Tree Search rollouts, one in which the agent tried to simply maximize its own final score and one where the agent was competing in an adversarial sense to "win" the game. Score maximization seemed to be the closest to how the real game plays out among amateur users, but I will describe how I did my set up for both types

- Score maximization: In this case I ran the rollout until there were fewer free available routes than there were players (which departs from the game a bit, but allowed me to normalize the number of turns for everyone). I then tabulated the agent's score using a depth first search function to determine if they completed each route and add the values for all completed routes (I ignored negative values for failed routes because I have not set up a route selection interface yet). Because we are operating with no longer of the opponents placements for this I did not try to calculate the longest route.
- For this evaluation, I did my rollout for each state until the board was full was again, but this time the reward for the MCTS agent was simply a (+1) if the agent won, a (-1) if the agent lost to any of the players, and (0) for a tie (my simplified model without trains being counted at the end increased the chance of ties). For this agent, I created another depth-first search function that determined the longest acyclic route that a user finished and awarded (+7) for it (I decreased the value slightly because of the score simplification made it a bit too powerful).

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Conclusion

I was able to obtain results and my MCTS agent was able to out-perform random agents that simply randomly chose an available route quite often. I was still able to beat the agent nearly every time, but I truly believe that really my only limitation was computation power as even with the drastically reduced route space of 47, my agent took a long time and forced me to set a low iteration value. One of my main takeaways was how enterprising these solutions can be the first time. I read the paper and watched the documentary on AlphaGo and have seen write-ups for DeepBlue, and the computation they use is jaw dropping. They there after get optimized like crazy, but I definitely see why the initial choice is to throw all the computation power at it. If I were to do this again, I would really commit I suppose. I had a time restraint with other classes and projects that prevented me from being able to see how far I can take it.