COMP90054 AI Planning and Autonomy
Group Project, Semester 1 2020

Azul_project_group_29

Demo Video: https://youtu.be/WMWYJLRdlIs

1. Introduction

Artificial Intelligence has a variety of uses in all aspects of life. There are numerous problems AI can solve, but the most fun topic is to play games with AI. The goal of the project is to implement simple intelligent agents that are capable of playing the famous board game "Azul" in 2-player mode. Azul has several characteristics which make it an interesting but also not too complex playground for AI implementations. They are important for shaping AI strategies.

- All players have global knowledge.
- All player actions are deterministic. One certain action will lead to one certain state.
- All players take actions in turns, not simultaneously.
- The performance of a player is clearly measured by a discrete number of scores.
- Azul is not strictly a zero sum game. Although usually a higher score results in a higher chance to win, the sum of all players' scores is not a fixed number.

To implement the agents, multiple algorithms are carried out including Best First Search, Monte Carlo Tree Search and also the Reinforcement Learning. We used the Monte Carlo Tree Search agent in the final tournament of this project. In this report, we will discuss the performances of agents in different technologies. The agent 'naive player' will be used as our baseline agent, who will try to put the most number of tiles in a pattern line. The results are analyzed to help interpreting the strengths and weaknesses of the algorithms.

2. Implementations

2.1 Best First search (BFS)

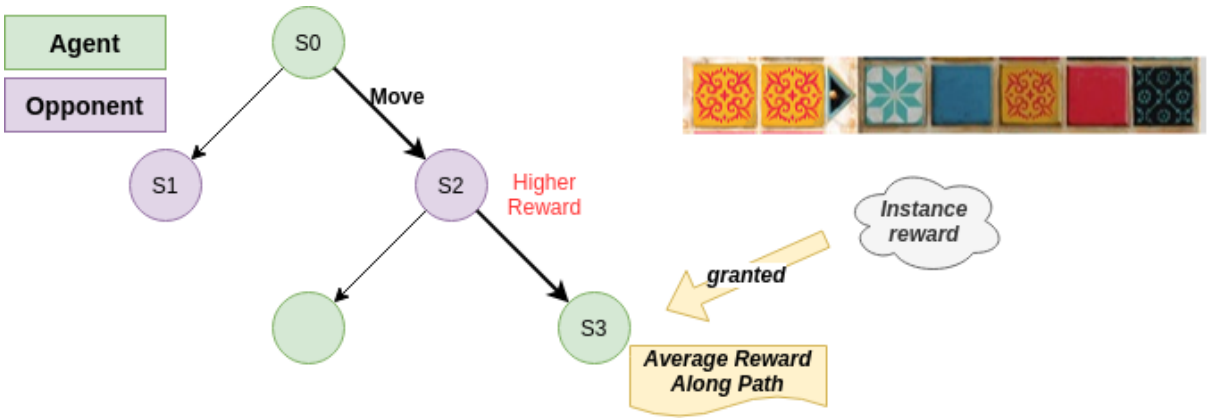

Figure 1. BFS with instance reward

Best First Search is the search algorithm that explores a graph by expanding the most promising child according to a specific heuristic. It performs well in satisfying planning.
In the Azul board game, we can view the gaming process as a tree search problem, where states are connected by moves taken. In order to win the game, the agent should try to get to the leaf node with the highest final score. There are several challenges to solve the problem with BFS: large state space, 1 second limitation on execution time, and most significant: no instant reward on each move as the player can get scores only when the round ends or the game ends.

2.1.1 Design Choices

In order to address the challenges, we have developed an instance reward algorithm to give scores for *every move* the agent made before the end of rounds. Even if the tiles have not yet been painted to the wall, agents can get a future reward in advance, as shown in figure 1. The instance reward will be used as the heuristic evaluation function for BFS. Also, under the 1 second move selection limitation, the

agent would be unable to search completely for the tree. Therefore, the agent will choose its next move based on the average rewards along the path for all the nodes explored until limitations are met.
In later state, we modified the BFS search to only 2 steps move, and also to include simulations for the opponent as in game theory. The purpose is to get the agent to focus on the short-term planning and it turns out to have a great improvement on the performance.

2.2.2 Analysis

The strength of BFS is that it will have a good performance in short term planning, like in the next 2 steps, since it will try to get the highest instant rewards. However, it lacks the ability to plan for a longer path due to the inefficiency in searching for a large state space.
The method can be improved by rewarding in longer term result, for example, if a set could be collected of 4 by the move, it can be granted a reward of ⅘ on the final bonus of full set tile collection.

2.2 Monte Carlo Tree Search (MCTS)

MCTS with Upper Confidence Bound (UCB) is used for our final submission It has several advantages deemed suitable for this project.

- It may be hard to estimate immediate reward for a move because scores are only calculated at the end of a round. MCTS can easily handle it because it does a simulation until a stage where scores can be calculated.
- MCTS can be terminated at any point and still gives a result, which is helpful since only given 1 second is given for each move.
- MCTS doesn't require any knowledge about the game besides rules and end conditions.
- Players take actions in turns so a tree search structure is appropriate.

2.2.1 Design choices

In the implementation, we simultaneously learn a policy for our agent and for the opponents. We assume the opponent tries to get the highest rewards as a variety of the tournament opponents would use such a heuristic. For our own agent, we use a reward of our score minus scaled opponent score, in order to encourage the agent to try to get highest and win the game earlier, as shown in figure 2.
Also, we find that the performance of MCTS depends on the policy we apply when doing simulation. A pure random strategy is very common but under time constraint, but it can access rare nodes. We use the given "naive player", which is still simple and quick but much more effective than a random player. Moreover, figuring out an outstanding simulation policy is almost equivalent to solving the planning problem itself, which is hard to achieve.
For each simulation we only continue to the end of a round even if the game doesn't finish. The reason is that tiles are randomly selected for a new round. Although we can include this randomness in our search tree by estimating probabilities of tiles appearing, it will make the tree unnecessarily huge and impossible to give a meaningful answer in 1 second. However, when calculating rewards, we do include the end of game score. This is to encourage our agent to collect tiles in a column or of the same colour.

2.2.2 Strengths and Weaknesses

In addition to the strengths already listed in section 2.1.1 Why MCTS, MCTS is also easy to understand and implement. If an agent uses too much human knowledge as input, sometimes it ends up being a big cluster of logics because the developer tries too hard to tell the agent what to do in certain situations. MCTS doesn't have this problem because it doesn't require much domain knowledge.
Compared to other tree searches, MCTS is asymmetric. It visits nodes it deems worthy more often and doesn't need an explicit algorithm to eliminate branches.
The biggest problem with MCTS is probably its performance under time constraint. Without exploring enough nodes, it is not reliable to provide an optimal solution, that's why in the beginning round when state space is still large, the MCTS tends to make bad initial moves. Here we tried to address the problem by using naïve search at the start first few moves of the game.
In the end, since our MCTS agent only plans round based, some end of round move were not the best strategy as the agent will try to avoid putting tiles on floor, but sometimes, it's better to put tiles on floor to leave space for next round.

2.2.3 Challenges and Improvements

One challenge is the resource limitation. With a fixed amount of time, there is a trade-off between simulation complexities and number of simulations that can be performed. If we have more time, we can implement a variety of strategies and perform many trials to measure their relationship and find the best policy. Also, at the early stage of the game, when the state space remaining is still large, we can use other algorithms to do the planning, like reinforcement learning, and in later states, MCTS can dominate the performance.

One more improvement could be estimating appearances of tiles before the start of a new round. This task gets easier when the remaining tiles get less. We could effectively reach a stage where the number of possible situations is so small that they can be incorporated into a search tree. Sometimes, we may also do a partial estimation. For example, if we know there are less than 5 red tiles remaining, we should probably avoid putting a red tile in line 5.

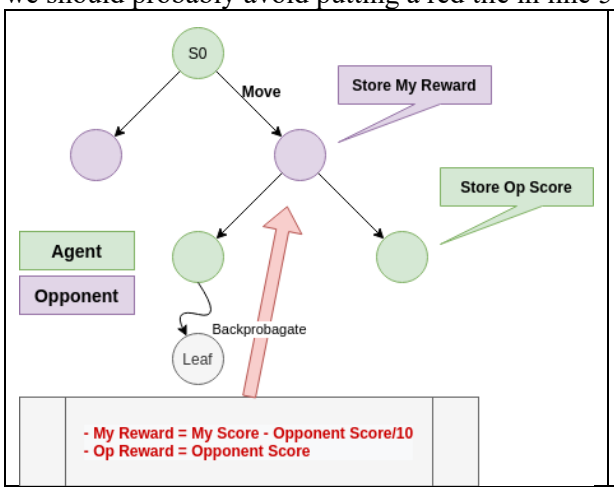

Figure 2. MCTS

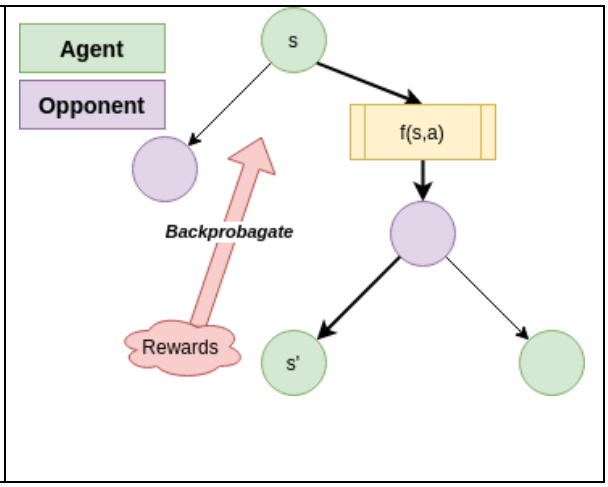

Figure 4. RL

2.3 Reinforcement Learning

Instead of using tree search algorithms to solve the problem, we can also implement Reinforcement Learning to develop an effective agent. Unlike other Markov Decision Process algorithms, RF does not assume knowledge of an exact mathematical model of the MDP.

As the Azul game has a large state space, it is impossible for us to store all state-action pairs as in normal Q Learning, therefore, we will use a linear approximation function **[2]** to approximate the Q function with features and weights.

2.3.1 Design choices

There are 45 normalized features used in our designed to address the game states, including percentage of each tile in the bag and factories, and line fill percentage of target move and wall collection percentage for both players.

We implement a model free reinforcement learning agent, like shown in figure 4, where the update will be delayed until the opponent makes the move. Both Q learning and SARSA are implemented and we use SARSA in our experiments in section 3.

2.2.2 Strengths and Weaknesses

With Reinforcement learning, we can backpropagate the end of game bonus to update the weights through Markov Decision Process. Therefore, RL agents will theoretically be able to plan for a whole game move based on the features.

The problem with RF agents is that it is hard to explain the selections and make adjustments to improve the agent. Also it is hard to address the problem of random tiles draw at start of each round, in the point of a linear approximation Q function, the state changed because of its action, but in fact, it is just changed by the environment, but the effect will be updated to the function weights.

What's more, as the agent is built under the project framework, where if the game ends, the reward will not pass back to the agent, therefore our agent will never observe the outcoming state and reward, thus it will never successfully plan for a game end move.

2.2.3 Challenges and Improvements

The major challenge for implementing reinforcement learning is to design the features for estimating the states. This is a very important part of RF, since we have increased our features from 19 to 45, and the win rate of agents against baseline was rising 30%.

For future improvements, the 45 features are still a 'plain' explanation on the game state. We believe features involving counter play should be included to address the current relations between different players and these may improve the performance.

Although we have chosen the model free RL in the end, we implement a single Q learning agent at the beginning without considering the opponent's move, and the performance was quite bad. We think a Double Q learning should also be tried in this problem, so as to learn both for ourselves and the opponents like we did in the MCTS process.

3. Experiment and Evaluation

We have run competitions between three main agents developed and the baseline. The winning rate and average scores of 100 games run are listed in table 1.

Agent 1	**Agent 2**	**Win Rate (%)**	**Avg. Scores**
BFS 2 step	baseline	82% - 18%	49.45 – 33.09
MCTS	baseline	97% - 3%	59.19 – 29.88
Model Free RF	baseline	72% - 26%	46.09 – 37.62
BFS 2 step	MCTS	29% - 70%	53.81 – 62.57
MCTS	Model Free RF	86% - 13%	58.51 – 42.93
Model Free RF	BFS 2 step	36% - 63%	46.99 – 56.84

From the table, we can see that MCTS has a dominating performance of all agents. That's probably because its great performances on planning when the state space starts becomes smaller. Also, our design on rewards to encourage high score with wining intension works pretty well as MCTS can get a high average score on baseline model, who always ends the game in 5 or 6 rounds. While, RF has a lower winning rate than expected, and that's probably because our feature space is not good enough and the training is not complete. BFS 2 step focusing on maximum rewards on short term and it also works pretty well.

In the trial tournaments, Model Free RF gets a relatively low wining rate of 106-8-122 but good score 11036, making it ranks a position higher than staff medium agent. While, MCTS was submitted with a reward encouraging winning the game instead of gain highest score, therefore in the leader board, we have a relatively good win rate 166 -5-73, but less scores 9767. But MCTS ranks much higher than staff medium. In final tournament, we will modify the reward in MCTS as described in section 2.2, to balancing the trade-off of higher score and wining the game quicker.

4. Conclusion

In the project, we have tried out different technologies to develop an intelligent agent to play the board game Azul. Monte Carlo Tree search has given the best performances in our experiment under the project limitations. Meanwhile, reinforcement learning still has space for us to explore in the future.

Reference

[1] Browne, Cameron B., et al. "A survey of monte carlo tree search methods." *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012): 1-43.
[2] Melo, Francisco S., Sean P. Meyn, and M. Isabel Ribeiro. "An analysis of reinforcement learning with function approximation." *Proceedings of the 25th international conference on Machine learning*.

Self Reflection - Rongbing Shan, 945388

What did I learn about working in a team?

Working in a team requires good communications and co-operations, and obviously, it will be effective only when all members are active to find problems and solve them. It is important that team members communicate and share their ideas and progress. And we should always try to have a late start on any sort of projects.

What did I learn about artificial intelligence?

Artificial Intelligence is a fun topic. Programming the agent to solve problems automatically is a challenging, but interesting task. With proper utilization, AI will dominate a lot of industries in the future in all aspects of life. And also, math does matter a lot in the world of AI, as it felt so hard to design the rewards and heuristics for our agent to trade-off between ending the game and gaining high scores. And I have never imaged MCTS will have such good performance in planning, it can even outrun reinforcement learning.

What was the best aspect of my performance in my team?

I am very responsible for the project. I spent a lot of time exploring different techniques for agents and shared them with the team. I was willing to research and try out methods to improve our agent performances and was able to convert the theoretical algorithms learned in this subject into programs. The best-performing agent of our team is developed mainly by me. Also, I have implemented a BFS search player and multiple reinforcement learning agents. What's more, I have created the video demo for the project.

What is the area that I need to improve the most?

I believe the most critical part is that we should have started the project a bit earlier to test out all desired methodologies. Also, the communication of our team was not very effective, and I should definitely do more on that. And I should be better at resources managements to allocate the tasks more reasonably so I will have time testing out other algorithms interested.

Self Reflection -Congxin Zhang, 954515

What did I learn about working in a team?

Working in a team requires more communication with teammates and presenting your own ideas. When encountering difficulties, you can ask your teammates for help. It's better to start working as soon as possible, and try to keep up with the progress of your teammates.

What did I learn about artificial intelligence?

Artificial intelligence contains a lot of aspects. In this subject, I learn how to program an agent and solve problems automatically. Additionally, I learn many techniques such as blind search, heuristic search, Monte Carlo tree search, reinforcement learning, game theory and so on.

What was the best aspect of my performance in my team?

I tried the heuristic search technique in this project. I developed three versions including two steps, three steps and two steps with some game strategies taken by the agent. The first has the highest winning rate. It can always perform well when competing with other agents.

What is the area that I need to improve the most?

I should communicate more with my teammates, and put forward my own ideas. Secondly, starting the project as soon as possible, and leaving more time for testing every technique we have developed.

Self Reflection - Xinzhe Li, 825797

What did I learn about working in a team?

I learnt it is important to leave each other space. Under the current situation where everyone is working at home, we haven't had too much communication. We each picked a technique at the start, went on doing our own implementations and only had discussion until a later stage. I personally prefer this way because sometimes keeping close contacts puts pressure on individuals. Different individuals move at different paces and being able to work at my own pace really eases the pressure of being chased around by deadlines and brings better outcomes.

What did I learn about artificial intelligence?

I used to only believe in advanced techniques such as MCTS and reinforcement learning, thinking classical planning is mainly for teaching purposes and is not widely applicable in real life. I was proved wrong. Under time constraint and without a proper simulation policy, MCTS may not even beat a naïve player which doesn't use any ai planning technique. It is important to understand how an algorithm works but also why it works or not.

What was the best aspect of my performance in my team?

I explored the principles behind MCTS and wrote the first version of our MCTS implementation, which is later improved by Rongbing. I tried to use different simulation policies such as pure randomness, naïve search and best first search to investigate its effectiveness.

What is the area that I need to improve the most?

I need to be more familiar with python. I haven't systematically studied python. I've only picked some knowledge through prior study in other subjects. Although the algorithms are straightforward to understand, I spent more time than expected implementing it because I often tripped myself over simple grammars.