COMP90054 Group Project Report

Yi Li 990073

Yiting Wang 1047500

Hao Huang 1037462

Video link: https://youtu.be/EsyqFw75fZE

1 Overview

Azul, a famous strategy-oriented board game, is designed for two to four players. Regarded as a planning problem of artificial intelligence, some AI planning techniques can be applied on this game to design autonomous agent. Depth-First search, Greedy-First search, A* search, Monte Carlo Tree Search, Minimax search (in game theory) have been implemented. Beyond the class content, Alpha-beta pruning is also applied Minimax algorithm to meet the time limit requirement of this project. In section 2, all these techniques will be described in detail. With comparation of performance of different techniques implemented, we use [] in the final tournament.

2 Methodology

2.1 Greedy

2.1.1 Description

Greedy algorithm can be applied on movement selection in this game. It is a process of exploring locally optimal solution. Here, we implement greedy algorithm with two different evaluation function. The first one is selecting the movement that gives the highest score to current player. Another evaluation function involves simulating the process of opponent player selecting movement. We assume opponent player will select the movement with highest score as well. Then the movement of current player which provides the highest difference of score between two players.
When calculating score, we add up the score one player would get at the end of a round and the bonus. In this way, when selecting movement, the player not only considers about getting high score in every round, but also final bonus.

2.1.2 Weakness

Due to the locally optimal property of greedy first algorithm, it only cares about the current score. However, the lower score of current movement can help gain higher score in following steps. So we introduce depth first search.

2.2 Depth-First Search

2.2.1 Description

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. The algorithm starts at the root node (selecting some arbitrary node as the root node in the case of a graph) and explores as far as possible along each branch before backtracking. In this game, the root node is the state of current player. The goal is the sequence of movement with the highest score (Figure 1).

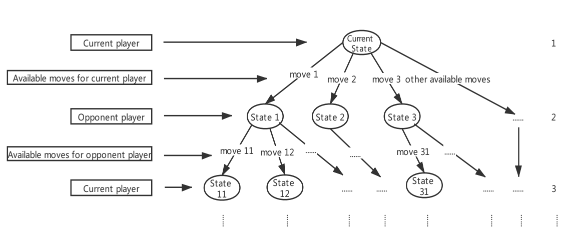

Figure 1: DFS applied on Azul

To save running time, we simplify the process to Figure 2.

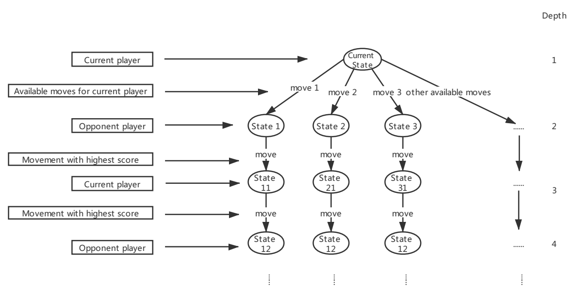

Figure 2: Simplified DFS

Here, the score of each player can be used to represent the state. Firstly, the available moves for current player and the game state-which contains all of the information about current and opponent player- are sent to algorithm of selecting move as input. Next, the DFS algorithm will traverse every movement in available move set for current player. For each movement, we will get a score for current player, and the algorithm will simulate the process of selecting move for opponent player. This process will be repeat to the depth we set. To better simulating the process, we will choose the movement with the highest score in the following process. we assume the sum score of current player is Sc and the sum score of opponent player is So. The evaluation of selecting movement is max(Sc - So).

Theoretically, the end of searching of this algorithm should be no remaining tile. However, due to the time limit, we only consider about maximum depth of 5. In real implementation, the performance of DFS with depth of 3 is better than 5. The disadvantage of this algorithm can explain this.

2.2.2 Weakness

The uncertainty of opponent player: The first one is the uncertainty of opponent player. We have no idea how opponent player will select movement. It most likely not select the movement we imagine. Then we cannot get high score after five steps.

The break with previous calculation: Even if opponent player select the movement as we imagined, the algorithm cannot provide ideal result. Because every time turning to our player, without the memory of previous calculation, it will recalculate condition of the following five steps. This is not only time-consuming, but break with previous calculation.

2.3 A* Search

2.3.1 Description

A* is an informed search algorithm. It aims to find a path by pre-processing the move and choosing the move with the highest score based on the heuristic function, which is the board score defined in the game rule. The process starts with the current game state, and traversing all available moves to find the best move.

An modified A* is based on a hardcode heuristic particular for Azul game. This heuristic function is calculated as a reward and penalty system, consisting of five parts.
For AI player:

- Penalty for putting too many tiles into floor
- line
- Reward current score difference
- Reward for having the first go token
- Penalty for unfinished pattern line but reward for unfinished pattern line in opponent game board
- Reward for game state towards the bonuses, which are the column completion and colour set completion. For example, if there are two moves to fill up the same pattern line, the AI should choose the one whose wall destination has more filled squares in column.
- Reward for finishing the game.

This heuristic function emphasizes gaining more bonus rather than focusing on the scores from placing tiles on the wall.

2.3.2 Weakness

Azul is a rotational chess, as a result, we need to also consider the available moves of the opponent. We get a optimal path to get a highest score after a complete search. However, it's common that the next game state will not be the same as we assume in the optimal path because the opponent's choice will affect the whole game state. As a result we need to calculate a new optimal path, which seems redundant and stupid. So we just apply greedy search to get a highest score in every iteration of available moves.

2.4 Minmax Search

2.4.1 Description

Minimax algorithm is a game theory strategy which aims to find the move with minimal loss, widely used for two-player games. The basic idea starts with setting up two players, MAX and MIN, the algorithm calculates the heuristic function for all possible moves then backtracking the choices from the end of the game. For MAX, the optimal move is the one with the highest score, while for MIN, the optimal move is the one that leaves MAX the worst scenario. In general, the minimax is a pessimistic strategy which focuses on searching for the best move in the worst condition that the opponent left to us. In this game, the MAX is our player's turn and MIN is the opponent's turn. Since Azul randomly initialises the game state in each round, the algorithm only considers one round situation.

However, in order to ensure the AI player chooses the most optimal move, pure minimax requires to search through each turn in the round, which is infeasible within the time constraint. Hence, we implemented alpha-beta pruning to reduce the search field.

Alpha-beta pruning records two values, with Alpha representing the minimum score in MAX turn and Beta referring to the highest score from MIN turn. For MAX turn, if the current alpha has reached the Beta, which means the current move is the best move that MIN turn left for our player, hence there is no need to search the rest moves. For MIN turn, if the current Beta has reached Alpha, which means the current move is the best possible move.

By applying this pruning, the program is able to reduce the search width in each round but increase search depth with the same time complexity, compared to the pure minimax process.

2.4.2 Weakness

Since there are too many move candidates for each move, the search width is really large. With time limitation, the program is unable to search through the whole round, even after implementing alpha beta pruning. However, the most outstanding advantage of minimax is to make the best move by reviewing the entire move chain of the round. If the minimax only considers the first few move chains, it might ignore the moves which earn less scores in the current move but could gain more scores in the future turn, thus decreasing the effectiveness of minimax.

2.5 Monte Carlo Tree Search (final submission)

2.5.1 Description

Monte Carlo Tree Search (MCTS) is a lookahead search. We make the board game as a (Markov Decision Process) MDP problem according to its rules and implement MCTS to solve this problem. It consists of four important steps to construct a Monte Carlo Tree: Select, Expand, Simulation, Backpropagate.

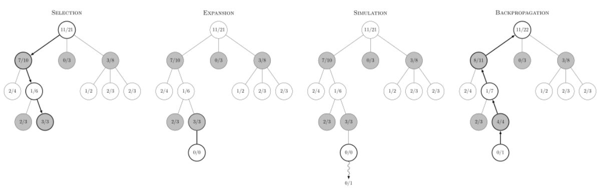

Figure 3: Monte Carlo Tree Search

2.5.2 Process

The focus of MCTS is on the analysis of the most promising moves, expanding the search tree based on random sampling of the search space [Wikipedia]. In games, the application of MCTS is based on different playouts of players and game states. Each node in the tree represents a specific game state/player state, which we change accordingly when we make different selections.
Selection & Expansion: Considering time limit and large amount of available moves, instead of UCT (Upper Confidence Tree), we use random choice for the expansion and selection. We do this to try to explore as many situations as possible in case we missed the most valuable move.
Simulation: In simulation, select one random playout and complete that until the game is finished (in Azul board game, one player finishes at least a row in the final grid). To prune and shorten the time, the value after each simulation is consisting of two parts, reward and cost. We design our policy to evaluate every state not only using the Score the player will obtain.
Backpropagation: After finishing a simulation, the result/value of the playout will update the information in the nodes on the path from root to the last node. In this case, when we need to update the value of the same node, we do not use Bellman equation to do this but calculate the average of value of the same node.

Only faster enough times of simulation, a tree contains almost all states of the game and the average value of each node will converge close to the actual value. Then the search tree can guide people to select the step with the biggest value under every circumstance.
The strength of this algorithm is it is trying to explore all the states of game and provide a guide.

2.5.3 Weakness

In this case, at the first few steps of each round, each player will have more than 100 available move operations. It's time costly to apply UCT to select and expand without leaving any possible move.
The opponent's movement is unpredictable which requires us to select for them in the simulation.
The result of the simulation and performance will vary when we apply different policy to choose

opponent's movement. In this case we apply a greedy policy for choosing the opponent's movement. In this case we apply a greedy policy for choosing the opponent's movement.

Win:Lose(%)	A*	Heuristic	PureMCTS	Pure Minimax	Alpha-beta Minimax	MCTS+ Greedy	Minimax+ Greedy
A*	N/A	40:57.5	45:52.5	85:15	80:17.5	37.5:57.5	62.5:35
Heuristic	57.5:40	N/A	45:55	95: 5	85:12.5	57.5 : 40	77.5:20
Pure MCTS	52.5:45	55:45	N/A	72.5:22.5	70:30	67.5:32.5	47.5:50
Pure Minimax	15:85	5:95	22.5:72.5	N/A	42.5:55	35:62.5	35:62.5
Alpha-beta Minimax	17.5:80	12.5:85	30:70	55:42.5	N/A	22.5:77.5	40:60
MCTS+ Greedy	57.5:37.5	40 : 57.5	32.5:67.5	62.5:35	77.5:22.5	N/A	60:40
Minimax+ Greedy	35:62.5	20:77.5	50:47.5	62.5:35	60:40	40:60	N/A
Naive player	25:75	17.5:80	15:82.5	27.5:72.5	20:77.5	12.5:87.5	30:67.5

MCTS requires a large amount number of simulations for the average value to converge. Obviously in this case, because of the time limit of the game, the times of simulation are far from enough.

2.5.4 Improvement

Simply we will calculate the score of players when each round ends as the value of the playout. Because of the time limit, we are unable to conduct a simulation to the end. As a result, it's hard to get the bonus marks. So we came up with our own policy to calculate the reward and cost of different choices of movement according to the game rules.

3 Improvement

Because of the time limit, we are unable to conduct a Minimax with depth equal to 2, which it's far way to the end of the game in most situations. So pureMinimax can only guarantee a local optimal solution. To improve this, we build a hybrid algorithm combining greedy and Minimax, allowing the AI player to earn more score in the first few steps and lose less score in the last step of each round. It's also a local optimal solution that performs better.

4 Performance and evaluation

The performance of different algorithms can be found in Table 1.

5 Expectation

If we have more time to work on this project, we will try to improve the performance of every technique implemented here. We will explore methods to improve the effectiveness of our techniques.

Self reflection

Name: Hao Huang   Student id: 1037462

As a virtual team in this special period, we worked with harmony and efficiency. I found that it's much easier for us to understand how the project works. Every time when I get a misunderstanding of the project, my teammates will immediately offer help to me. It's helpful to have a good start on my assigned part. About artificial intelligence, it gives me the deepest impression that it will easily become an artificial idiot if learning times or learning depth is not enough. For what I did, I implemented MCTS and found that the performance would be totally different when I train it with different times, depth and reward function. In this turn-based game with a time limit, there is also a trade-off between the simulation times and depth. AI planning is not that easy as just implementing an algorithm.As a member in the team, my main contribution is about coding. Not only I finished my assigned part, but also offered help to my teammate on coding work like improving the Minimax with alpha-beta pruning.For me, I am not so familiar with how to apply neural networks in practice. If I want to do more exploration and development in AI Planning problems like this assignment,I believe getting familiar with the Neural Network will be important.

Self reflection

Name: Yi Li   Student id: 990073

In a team, it is important to cooperate with team members. Most importantly, to do better time management, it is necessary to allocate different work to every team member at the start of the project. Moreover, the benefit of working in a team is discussing problems with team members.

About artificial intelligence, I have learned some useful planning algorithms and methods such as A* Monte Carlo Tree Search etc. All the knowledge I've learned from this course can help me solve the problem about AI planning afterwards.

I implement two algorithms in this project. In this team, I discuss with team member positively, and answer question in time.

Firstly, my ability of reading others' code need to be improved. And, the time effectiveness of writing code can be improved as well.

Self-reflection

Name: Yiting Wang   Student ID: 1047500

This is the first time that I finished a group project without any face-to-face communication and discussion, but there are still many online methods allowing us to leave message and reply even we might not all have same available time. Communication is always the most important factor that allow a group to work more efficient, during this project, whenever a member has a problem either in academic or technique, other members will always provide solutions or give opinions. If no one is sure about the answer, we would discuss it together. In addition, the job assigning is crucial as well. An unclear arrangement of the parts might lead to duplicate work and also cause confusions.

For artificial intelligence part, we've tried many different algorithms. The most interesting thing I found is that comparing to all complicate strategies, a hardcode heuristic in greedy algorithm seems to have the most effective performance. As it is really simple to implement and has very low time complexity, but it outperforms mostly of all other algorithm we tried. From this view, it is obvious that there is no best algorithm, but only the most suitable algorithm for particular problem. And the reason why hardcode heuristic works well is because it purely evaluates the score based on game rules and provide reward and penalty for all most each rule.

Regarding the project, I was trying to provide more ideas including the hardcode heuristic and minimax algorithm, but I might not successfully offer practical output in coding implementations, although I tried my best. Hence, I've provided more effort in report and presentation. I think I'll need to put more hardwork to improve my understanding of this subject content and the ability to work alone as well. Sometimes, I might lose the direction of the current stage and always need someone else to point out and help me back to the right track.