COMP90054 ASSIGNMENT AZUL GROUP 03 REPORT

A PREPRINT

Xulin Yang
904904

Xinyao Niu
900721

Wenrui Zhang
872427

June 3, 2020

1 Oral presentation link

`https://youtu.be/kd_EsLFdQzE`

2 Implemented techniques

In this section we will introduce the techniques we used. We tried Monte Carlo tree search and documented in the section 2.1. We tried Greedy search and documented in the section 2.2. We tried Minimax algorithm and documented in the section 2.3. The solution to the challenges we faced when implementing the following techniques are documented in the section 3. The possible improvements for these techniques will be discussed in the section 5.

2.1 Monte Carlo tree search

You can see the implementation for this algorithm from the *MCTS* tag in our repository with /players/StaffTeamEasy/myPlayer.

2.1.1 Description of the design decisions

Compared to the value iteration and policy iteration, MCTS has a lower memory usage cost. Because there are a huge amount of game states for the Azul game and averagely 5-120 available actions for a game state. As a result, value iteration and policy iteration requires a huge computer memory demand to store the state action pair value table for iterations. In other words, value iteration and policy iteration is impossible nor suitable for selecting an action for the azul game. Consequently, although the MCTS might not be so robust compared to the iteration methods, the impossibility of memorizing state action table force us to pick MCTS among these choices.

2.1.2 Strength

Firstly, as mentioned in the design decision above, the MCTS have a lower memory usage cost compared to the value iteration and policy iteration who are storing the whole state space. Secondly, as we solve online each time when we encounter a state, we can pause the iteration of MCTS when we meet the 1 second time limit but still get a policy to apply. Thirdly, if we apply upper confidence bounds (UCB1) with MCTS to have Upper confidence tree (UCT) and set $C_p = 1/\sqrt{2}$. We can approximate to the Minimax algorithm. In other words, MCTS can enable us to pick the global optimal action with the consideration of opponent's actions. Fourthly, we don't need an explicit evaluation function which requires our deep understanding on the Azul game.

2.1.3 Weakness

The weakness for the MCTS is that it might be not so robust if the iteration we applied is not enough as described in the description part.

2.1.4 Challenges experienced

Firstly, unlike the examples in the lecture, we don't have the exact probability of opponent's action when we made an action. Instead, we need assign our own designed Markov decision process for opponent's action probability. Secondly, we need to design our own reward function for the MCTS. What we did is that we use the difference between our player and the opponent player's score as the reward function. Thirdly, the expansion operation in MCTS is very time consuming in python as we need to create new node in the tree for the MCTS. The solution is given in the section 3.2. Fourthly, if the branching factor is too large, then the MCTS cannot explore enough to give a sensible policy before terminated by time limit. We will use the technique described in the section 3.3 to solve this problem when the computation power is extremely limited as in this case.

2.2 Greedy search

2.2.1 Description of the design decisions

Greedy search is a extremely powerful search algorithm that the algrithm goes straightly to the goal that we set, and runes pretty much very fast (almost in linear time to the move it got) comparing to BFS, DFS, A*. The algorithm could be found in tag "greedy" and under StaffTeamEasy folder in Player named *greedy.py*.

2.2.2 Strength

It is fairly easy to implement as we only need one linear-time function to evaluate how go is a move and then return the best move. This also implies that it is fairly fast in terms of computational time, which means it will hardly timeout for the 1 second time limitation. Furthermore, we can mostly focus on the design of the evaluation function when implementing the algorithm. That is, we could let the greedy algorithm imitate what we (as human) are thinking when playing the game without paying too much attention on the infrastructure that make the algorithm run since it is quite easy when comparing to the implementation of MiniMax algorithm.

2.2.3 Weakness

The weakness for the greedy algorithm is that it just consider current state with the evaluation function which sometimes does not give an optimal action in a long run. Even thought we are adding a lot more features to the greedy linear function, it still only have a very limited visions.

2.2.4 Challenges experienced

The design of the linear evaluation function is difficult. We spend a lot of time on selecting good features and fine-tuning the weight, but still could hardly get huge improve from iteration to another iteration. However, the good thing here is that the function could be used not only in the greedy algorithm but also for other algorithms that rely on the evaluation function. The detail has been documented in the section 3.4.

2.3 Minimax algorithm

2.3.1 Description of the design decisions

Minimax algorithm will work as it maximize score for our player in a relatively long run. It takes the consideration for the opponent actions and evaluate its effect in later state while the greedy algorithm won't. In this game, we have the 2-player contest which makes the minimax algorithm work efficiently.

2.3.2 Strength

Firstly, our player plays optimal action in relatively long run if the opponent agent also plays optimally. Secondly, the algorithm is complete as the Azul game has finite state aciton pair. Last but not least, we could integrate our greedy function in order to prune some apparently bad action.

2.3.3 Weakness

The time complexity for the algorithm is $O(b^m)$ where b is the branching factor and m is the maximum search depth. The b can be a number from 5 to 120 (maximum) and m can range from 1 to 15. So there will be a huge state space to be considered. Secondly, we need a good evaluation function for the algorithm, otherwise the algorithm will output some unexpected choices.

2.3.4 Challenges experienced

The biggest challenge is apparently that we need to make the minimax algorithm fit the 1 second time limit. To solve this problem, in one hand, we can try the $\alpha - \beta$ algorithm which uses the returned value from each child minimax node to prune the nodes to be searched, however, this normally do really well. On the other hand, we could use our linear greedy function in this case. For the Max part, we could use this greedy function to filter out the move that really bad, so that the branching factor will reduce dramatically. For the Min part, instead of listing all the possible action for our opponent, we use this greedy function to select action for them in order to save time. As the linear function are already quite good, the best action are highly likely be included by our filter. More detail of this technique will be discussed in the section 3.5. What should not be neglected is that the design of evaluation function is quite challenging. The detail has been documented in this section 3.4.

2.4 Final tournament agent

2.4.1 Overview

The final submission is a mini-max based search algorithm integrated with a $O(n)$ heuristic function for reducing the branching factor as well as return a decent result even run out of time.

2.4.2 Strength

The strength of our program is most about how to save time and approximation. Our program guarantee to return a decent result that compute by a pretty strong greedy function even we run out of time in exploring the mini-max tree. Moreover, that greedy function are used for filtering candidate that we are going to explore. By doing this, we could dramatically reduce the branching factor (down to maximum of 5) for each max layer. Further more, we use the greedy function to simulate the action the our opponent without explore the actual min layer as this method work fine in most case, and more importantly, it is fast. Combining those techniques we mentioned above, our algorithm could look further to see action 2 steps away and gain huge benefit from that.

Apart from time saving, we use two difference standard to evaluate how good is our action by comparing the game state two steps away with the root game state. If we are going to lose the game, that is our expected end of game score is lower than our opponent, we are going to choose the action which minimise our expected score difference. Whereas, if we are leading the game, we would prioritise the action that can give us higher expect score at the end of the game as the action can expand the score difference might not give us much benefit on earning score for ourselves. By doing this, we could both gain benefit no matter weak or strong opponent we are facing.

2.4.3 Weakness

The drawback is pretty clear for our program. As we mentioned before, the simulation technique that we used for simulating opponent action does failed at some situation where our opponent is too strong or too weak that we cannot hardly predict their action. However, in the context of this subject, we might not facing and opponent which is far stronger than us as we tested our greedy algorithm and get a pretty satisfied ranking in mock contest. For the other one, we might miss the change to pick the optimal action that can boost our score but are unlikely to failed the game. The other weakness is we are not using the information if we break our mini-max search in the middle due to time-out. Last but not least, we are using two different evaluation function to filter the candidate and return a score for the leaf of the search tree. It may encounter some unexpected behavior as the final score are determine which action are selected but we also want to use the feature inside the filter to restrict the action chosen when there are very few actions.

3 Alternative techniques

3.1 Q learning with man-made features

This is the primary idea that we would like to build for this project, as it is really fast to compute by using a linear function and convergence guaranteed. We also try to modify the runner so that I will call a update function at the end of the game. However, the weight always converge to a strange array which the overall Q-learning could be easily beaten by the same algorithm with the same feature but using our find-tuned feature. This may caused by wrong updating function. Unfortunately, we are running out of time on this, so we did not dig further to see what might go wrong.

3.2 MCTS with depth limit

As we found the expansion in MCTS is time consuming, we design a variant version of MCTS with a depth limit for expansion. When the expansion hits the limit, it return the current MCTS node instead keeps expand new nodes. As a result, we can save computation time. In this case we look 2 depth of actions ahead.

3.3 MCTS with branching factor limit

When the branching factor is too large, there will be too much states to be generated in the node creation. Thus will cause timeout even before the MCTS algorithm starts its iterations. So we add a *branching factor limit* before the execution of the MCTS. If the state's branching factor is larger than the *branching factor limit*. We execute the greedy algorithm instead of the MCTS. The reason is that when the state space is huge, computation for looking forward will be limited by the given time limit which makes it no difference compared to the greedy algorithm.

3.4 Evaluation function designing

How we make sure the feature we come up with is useful? When we decided to add a new feature, we will run the agent against other baseline agent with 500 games to ensure that we eliminate the randomness in the testing and see whether the new feature helps us to increase the winning rate and increase the average score we earned. Moreover, the weight of the feature is also tested in order to make sure all the features has the weights that reflect their importance. For example the weight for filling lots of tile is larger than the weight

of filling the line which has already been occupied by some tiles. The choose of the weight reflects the strategy we used in this project. Overall, the features used in this project are round score, end of game bonus, the penalty for filling few tiles in a long line, bonus for filling more tiles and the bonus to try to complete one line.

3.5 Top n actions selection

Since the minimax used in this project requires a lot of time for running. It makes the performance better if we can do the pruning. By using the heuristic function, we can first choose some moves that have high heuristic value. In that case, we can simply expand these states and usually it leads to a good result. In this project, we choose 5 moves to expand, since if we choose a lot expand it will have a long running time and the increase of performance is slightly, if we choose fewer moves to expand it might leads to missing the best move.

4 Experimental results

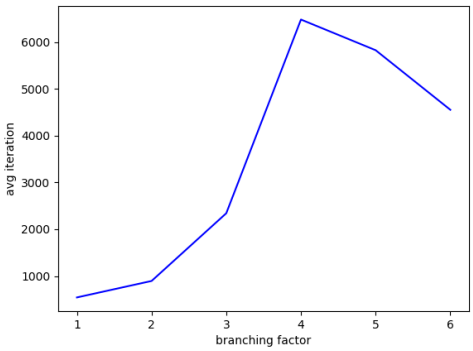

Figure 1: MCTS average iterations v.s. branching factor

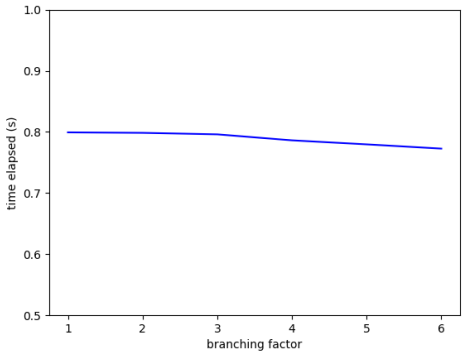

Figure 2: MCTS time usage for branching factor

We sample the MCTS v.s. naive player in 50 games and collect experimental data about working MCTS. As you can see from the figure 2, we use 0.8 seconds for each action selection iterations in MCTS. And from the figure 1, if the branching factor increases, the number of iterations will drop. If there are too few iterations run. The MCTS won't give a sensible outcome. In other words, even greedy algorithm is better in this case. So our alternative techniques for MCTS mentioned in the section 3.2 and the section 3.3 makes MCTS work with respect to the 1 second time limit without losing effectiveness.

	naive	MCTS	Greedy	Minimax	baseline1	baseline2
MCTS	93%	Nan	55%	14%	48%	49%
Greedy	93%	42%	Nan	17%	55%	51%
Minimax	94%	85%	82%	Nan	90%	72%

Table 1: Experimental result

The table 1 shows how our agent (row) compete with other agent (column)'s winning rate as well as the baseline agent we developed. The baseline 1 is the agent we developed for the May 30 contest and rank at 16. The baseline 2 is the agent we developed for the June 01 contest and rank at 9. We simulate 100 test contests between different agent. As you can see from the table 1, the minimax algorithm performs the best compared to other agents we have developed. Take the running of Minimax v.s. naive as example, the variances of the experiment contest size with 10, 50 and 100 are 248.09 219.01 198.06. As you can see, the increasing of the experiment size helps to reduce the randomness. Thus we can choose the final agent based on this experiment result. Although we would like to provide how our final version of these algorithms performs in the practice contests. But we always found bugs after our submission. So we didn't have the final version data in the practice contest. We only have buggy agent result. The baseline 1 in the May 30 contest and rank at 16 is the buggy greedy agent. The baseline 2 in the June 01 contest is the buggy minimax algorithm.

5 Future improvements

If we have more time, we might use the Q-learning algorithm with function approximation. We believe that we have explored good features for the Azul game. However, there is only a few days left before the due. As a result, we didn't have enough time to learn the weight for the features we discovered by the Q-learning.

If we have more time, we will apply the $\alpha - \beta$ pruning algorithm for the minimax algorithm to save the computation time. Thus gives a chance to make the minimax agent search deeper.

6 Xulin Yang individual report

6.1 What did I learn about working in a team?

Firstly, the team should have a clear goal. If the team doesn't have a clear goal, team members will have high probability to do the things they think are correct. Consequently, the team will be lack of discussion. So, every member in the team should be clear what the team is doing and working for the common goal for the team. Secondly, each individual should be assigned clear responsibility. If each team member is clear about what he is responsible for, then there will be high probability that the team will reinvent wheels for the project. Furthermore, some tasks might be ignored because no one think he should be responsible for it. As a result, the work delivered by the team will fail if responsibility is not assigned clearly to each team member.

6.2 What did I learn about artificial intelligence?

From this subject, I learn that artificial intelligence isn't magic. It is realized by some theoretical formula for computing a result for a task what we think only human are capable of. The goal artificial intelligence are deduction and reasoning, knowledge representation, planning, natural language processing (NLP), learning, perception, and the ability to manipulate and move objects. Long-term goals of AI research include achieving Creativity, Social Intelligence, and General (human level) Intelligence. What we focus in this subject is majorly planning.

From this project, I tried algorithms from the subject. And discover the limitation for these algorithm and the performance bottleneck. For example, In implement the Monte Carlo tree search algorithm for out team. During my implementation, the major challenge is that I need to make the algorithm fit the 1 second time limit. What I have done is to come up with the techniques I made as mentioned in the section 3.3 and the section 3.2.

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

Firstly, I watch a lot of replay for the team to identify some buggy action for our agent. And report the result to the team members in the communication channel. Analysis in which part of the algorithm influence the agent to make such an action.

Secondly, I attend all the team meetings to discuss the project with the team members. And I make sure my status of work is not only pushed to the GitLab repository but aslo my progress on works pushed to my team members. So that they can have a rough idea of what I am doing.

Thirdly, I implement the Monte Carlo tree search algorithm for the team. Although it is not selected by the team for the contest, it still can be used as a bench mark agent for other agents. I finish my development of the Monte Carlo tree search algorithm early. Then I help other team members in the coding part. I peer programmed with other team mebers though the zoom meeting.

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

Not pretty much. The major one is that the 4 subjects I choose this semester are COMP90015, COMP90020, COMP90024 and this subject which are 4 main subjects in master computing subject in the Unimelb master degree. As a result, I didn't have enough time for every subject if I didn't select so much time consuming subject where COMP90020 has a semester long project and COMP90024's project is very time consuming. Consequently, I starts this project one week before the due. If possible, I should organize subject selection more wisely.

7 Xinyao Niu individual report

7.1 What did I learn about working in a team?

We can finish the job that we normally could not finish by each individual if everyone is really progressing in a team. A leader is also very important when individuals can not agree. Everyone will do their part if they want to and then merge the work back in together. Team work is more challenging in the context of current situation where each of our team member is working remotely. In this case, more frequently group meeting is necessary as we need to know everyone is doing fine without knowing some big problems until last minutes.

7.2 What did I learn about artificial intelligence?

In terms of this project, I learnt that to come up with a good feature and good weight associate with each feature a difficult. That also enforce the idea why we are going to have model-free algorithms which might free us from doing this.

In terms of the subject materials, I gain better understanding about classical planning and reinforcement learning. I did not know that we have so many sophisticated way to come up with a heuristic before taking this subject. Moreover, the online interactive video of the second half semester gives me an excellent idea of how those algorithms work. From a more broader perspective, this subject gives me an idea that many part of world are now working under artificial intelligence algorithm which gives me extra motivation to learn more. Apart from those, I notice that their is a big difference in understanding/how to use the AI algorithm and knowing how to implement them.

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

In terms of non-programming part, I regularly update my progress with my teammate which reduce the change that multiple people doing the same thing. Also, I am the maintainer of the the git repository which help my teammates to resolve conflicts and versioning problems.

From programming perspective, I primarily develop a pretty strong greedy algorithms and some useful features that becomes the new baseline and basis of any other new algorithms that we implemented later including MCTS and Minimax. Furthermore, I put huge effort (almost rewrite the key part) on improving the performance of our Minimax algorithm based on the work of Wenri, which made the algorithm usable and pretty competitive, and eventually becomes our final submission. Moreover, I did a lot on solve the time-out issue for our minimax algorithm. We initially use the func_timeout function that predefined in utils, however, that function actually requires extra invocation time where it may takes extra 0.1-0.3 second to exit. What is worse that we can hardly test our code on a server-like environment to see how likely it will time-out in 100 matches, but I made me realise that I might have a better CPU than the server. Me and my teammate then find a workaround that we test our code on a worse laptop and see how it goes. We eventually find 0.7 is quite safe with 500 match no time-out (we modify the *advanced_model.py* and the program will exit once any timeout occur).

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

It is better for us to decide a regular time for zoom meeting instead of pretty random at the moment which gives us a chance to planning our agenda before the meeting. Furthermore, I need to improve my ambition of using new algorithms. I actually tried to implement Q learning at the very beginning as it is fast to make decisions and what we need to do is load the weight in the competition. However, due to too many change made to the part that we should not change in order to train the weight and few days without much progress, we decided to fall back to Minimax which we familiar the most eventually.

8 Wenrui Zhang individual report

8.1 What did I learn about working in a team?

First of all, I learned that the communication between team members is important. For example, it is helpful to discuss about the feature and share some useful strategy. Regular group meeting after the tournament is also helpful, some undesired moves can be extracted and the strategy using by the opponent can be analysed to improve our agent. Moreover, the competition between team members makes me stay motivated. The usage of git makes the code easier to be uploaded and analysed. Each of us working on our own branch and we can keep uploading the newest version of the code without leading confusion. Overall, this group working is valuable for me.

8.2 What did I learn about artificial intelligence?

In this project, what I first implemented is the algorithm of minimax, and then our group improved it to be the final submission. The main logic of minimax and why we use it is mentioned in the main part of the report. I learned 3 aspects of improving this searching algorithm. Firstly, the choice of features is important, some good features could lead the minimax perform in a better way. Secondly, the weight of features is also very important, all the weight should be reasonable and it is hard to choose. In this project, we choose the weight based on our strategy and experience which may be reasonable but definitely not perfect. Thirdly, the depth of searching is one of the most significant approaches to improve the performance. Therefore the pruning is worth considering because of the time limit. Combining all of three, the minimax should have a good performance.

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

The best aspect is that I contributed the structure of minimax before our group worked on the minimax, which had some creative features, such as the closer to the centre leads to the higher heuristic value, penalty on trying to fill long line with few tiles. After several days of competition within the group, the minimax is chosen to be the submitted one. Then when we analysed the battle during the group meeting, I suggested some possible strategies and features. After implement them and some features suggested by my team members, our group submitted my program and got a nice grade of the 9th position at the next practice battle, which is just below the top team of the staff. After then, I explained the detail of my strategy and my function to my team members and then we came up some great ideas to look deeper. Since it is expensive to expand every state, we may first choose some moves which have the high score and expand them only. By implementing this logic, the depth that the minimax can search increased by two. This increase of depth is crucial, because it makes our agent perform better by predicting better result of opponent move.

By the last few days of the project due, I noticed that our agent may have the error of timeout on the competition platform sometimes, which is emergent to be solved. The main reason that happens is that our computer may have better devices than the one perform competition. Even though the score of our agent was still great after one random move, we had to come up with one solution to handle such problem. Therefore, I checked some online replays, and notice that such problem usually happened when there are lots of possible move to choose. The time limitation function we use cannot work in the way what we expect and I came up two solutions. One is to do greedy moves for the first two steep, another one is to reduce the timeout value from 0.7 to 0.5. Both of them would lead to the decrease of the performance but I had to choose one to implement and explain to the team member why this is the better solution than another. By considering that the timeout may still happen and the performance drops even if we set the timeout lower, I suggested to use two greedy steps at the first of each turn to make sure that we can be in time limit. Then, my heuristic function is used to calculate the best action. Since I designed the feature and weight well, the performance may not drop much even though there will definitely a drop. It is a trade-off between performance and time.

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

The area that I need to improve is the ambition to use the new technology that I haven't tried before. I am familiar with the logic of minimax and max-n, therefore I decided to implement them at first. However, since the special restriction of 1 second limit per step, it is hard to expand all branches or expand deeper. Therefore, I realised it might be helpful to train a policy and load the load policy. I reviewed the lecture and searched for some documents, but I didn't implement it at the end. One reason is that it may takes a long time to train an excellent policy, since I haven't tried to train one before and I cannot estimate the time needed for policy training under the problem domain of this project, it might be risky to implement it compared to using minimax which has been implemented. Another reason is that the minimax had some relatively good features that I spent a long time on it and its performance is acceptable. Overall, I shall be more ambitious on using new learned technologies when it is suitable.