Xiande Wen - Qing Feng Chye - Mingda Dong

Resources

Git repository: https://gitlab.eng.unimelb.edu.au/linkleagend/comp90054-2020s1-azul
Youtube video: https://youtu.be/veLgldG3fz0

Azul report by Salted Egg

Executive Summary

Azul is a board game designed and released in 2017 by Plan B Games, where two to four players try to fill out as much of their personal 5x5 tile boards as possible before the game ends when one player finishes an entire row on his/her board [1].

As newfangled as it can possibly be, Azul seems to possess great potential in game theory and in strategisation, and thereby sparking our team's interest in building an Artificial Intelligence (AI) to compete against human operators.

Among the techniques we have learnt so far, the Planning Domain Definition Language (PDDL) approach was ruled out from the beginning, for we were convinced that it might end up being very complicated and prone to errors. Following the PDDL, blind search became irrelevant and unable to provide insights on the choices the players make. The research was then divided into three approaches, using heuristic search (HS), Breadth-First Width Search (BFWS), and game theory method, which, specifically speaking, is the minimax algorithm with alpha-beta pruning. It is then found out that heuristic search became intriguing in terms of defining a good evaluation function which required a great amount of domain knowledge of the problem to perform well, as the game came out relatively new and our team had no expert for the game. The reason why we chose BFWS over some reinforcement learning such as Q-learning and SARSA is because the lecturer said, to our surprise, that a BFWS agent tends to outperform a Q-learning agent. However, due to the scarcity of resources and relevant information, we excused us from BFWS and moved to using Monte-Carlo Tree Search (MCTS), whereas we moved from Heuristic Search to Q-learning. After some experimentations, we have gained new knowledge over different techniques and developed a deeper understanding of the Azul board game from the behaviour of the AI agent.

Minimax

Minimax is the most used game theory method. It is designed for the two-player adversarial game. And it selects the best move by expanding the search tree by all the actions that a player can take at certain state, evaluating scores for all the terminal states and back propagate in a way such that our goal is to select the move resulting the maximum score while assuming the opponent is trying to minimising our score. Its performance would be perfect with unlimited resources in a zero-sum game.

In our case, intuitively the gain of one player is not exactly balanced by the loss of the other but to some extent, we can still observe that the player can select a move which may cause more loss for the other. For example, we can make a move that blocks the opponent from being able to get 5 tiles in a line and force them to lose the bonus marks. In that way, the loss of the opponent can be regarded as our gain which could possibly give us more chance to win at the end of the rounds. The minimax is guaranteed to have its place in the competition as long as the two players have interactions so that we chose it as one of our strategies.

The performance of the minimax is largely determined by the depth of the search tree. If it can go deeper in the tree, the evaluation of each move would be more accurate. But the search space is growing exponentially with the depth at most O(b^d) so more depth with limited resources means that we need to find a way to reduce the search space. One common technique that always comes along with the minimax is the alpha-beta pruning. It works by recording the best scores we are assured of with the alpha and beta when expanding the search tree. And we can disregard a group of subtrees if we found that expanding them would not give a better score than alpha or beta. With optimal ordering, the alpha-beta can achieve search space of O(((b^d))^(0.5)) [7], which means we reduce the search space by its square root so that we can go twice deeper or so.

With the alpha-beta pruning, our minimax is only able to achieve depth of 2 under the 1 second time limit. And it can achieve a winning rate of 93% over 100 games against the naive player and an upper middle position in the tournament. But it is not hard to see that the reason for the good performance against the naive player is because the naive player works in a similar way to our assumption of the opponent. Our minimax player assumes the perfect rationality for the opponent [8], which means that the opponent would follow the same strategy when encountering the same situation, which is not always the case. This is one of the limitations of the minimax player in general games. Another limitation is that even with alpha-beta pruning, the minimax is not guaranteed to have a smaller search space because of the ordering of the states. In the worst case, the search space will remain the same.

Many techniques have been introduced to compensate for these limitations like the randomised minimax algorithm where we try to model a non-rational player and also the preferential ordering of the states in order to give the best ordering for alpha-beta pruning. Therefore, we can see that even with a very harsh time limit, there are still many things we can do to further improve the minimax player.

Monte Carlo Tree search

Monte Carlo Tree Search is one of the most popular strategies that has been widely used in the Artificial Intelligence world. This strategy has been implemented in the game of Go, which is the well known 'AlphaGo'. In March 2016, AlphaGo beat the world best Go player - Lee Sedol, with the score of 4-1.[2] We are amazed by the incredible performance of Monte Carlo Tree Search, in terms of its ability to win one of the hardest games to be played in human history. Hence, we were having a big dream to implement Monte Carlo Tree Search in our project, to imitate the behaviour that AlphaGo used in beating the world champion.

When we first started designing the Monte Carlo Tree search, the challenge for us is to define a proper model. As Monte Carlo Tree search is not model-free based learning, we have to come up with a design that is different from the given model in Azul specification. In each state, Monte Carlo Tree search model contained Q value, N value (number of visited), and children expanded in the node. Initially, we tried to design our own monte carlo search tree. Gladly, we found some useful resources that have provided us the template of Monte Carlo Tree search, and we need to define the abstract classes ourselves. The algorithm of Monte Carlo Tree search used in this project is the same as discussed in the lecture: Select, expand, simulate, and backpropagation. We also used the 'multi-armed bandit' strategy, and used the formula of upper confidence tree (UCT), to decide whether to do exploitation or exploration. We spent 2 weeks to figure out how to set up the system, including writing our own reward function, a function to generate new state, a function to choose random move, a function to make a move (in order to simulate the game), a function to choose a winner, a function to set up our own board system, and most importantly, incorporating with the 4 steps in MCTs. [3] In selection, we choose the node with the highest UCT value. When we meet the terminal state or unexplored node, we would return the path that has been recorded along the way. In expansion, we find the children of the selected node by expanding them using the function that we have written. In simulation, MCT would use the function designed by us to do simulation and get a reward after the simulation hit the terminal state. In this Azul project, a terminal state is defined by the end of round. We were thinking to use the end of the game as our terminal state, however, we found that it is very inconsistent with the state of nodes. By using the end of the round as our terminal state, we would be able to cut the depth of the tree and be more efficient in constructing the MCT and choosing the best move. After getting the reward from simulation, MCT will then backpropagate the reward back to every single parent node[4].

In the testing phase, we realized that MCT did take a long time to do rollout before choosing a move. In addition, MCT's performance is highly dependent on the number of rollouts done before choosing the best move. For example, by setting the time of rollout to be 1000 (do rollout for 1000 times before choosing the move), MCT is able to beat the minimax strategy. However, given the time limitation in Azul project specification (1 second to make a move), only maximum 30 times of MCT rollouts can be done. In this case, MCT strategy cannot even beat a naive player. We have tried to remove the time limit by ourselves, and run testing on MCTs. The result is quite surprising, as running 1000 rollouts for MCTs performed the best compared to any other value. Our hypothesis is that since we are using the end of round to be our terminal state, there is a high chance for MCT to have an overfitting tendency, without thinking of the strategy to achieve the best of the final game score. By choosing 1000 rollouts for MCTs, and running 100 games against minimax algorithms, MCTs won 35 games and minimax won 65 games. However, the whole experiment process took 4 hours to be completed.

After finding a way to search for a strategy to do offline search before going to online competition, we found that we have to use a linear approximation together with Q-learning method, to save necessary value to an external file. However, given the time that we have for this project, this goal could not be achieved. If we have more time given on this project, we would definitely implement the Q-learning together with best move probability approximation, to do offline training and use it for tournaments.

Deep Q-Learning

Q-learning is one of the most basic model-free algorithms that utilise Bellman equation in the reinforcement learning (RL). The fact of being model-free provides Q-learning the ability to train over some dataset without prior domain knowledge, so as to learn from the pattern which the agent makes choices.
Half way through implementing the Q-learning algorithm, an inevitable problem emerged -- the limitation of a Q-table. In the current setting of the game, where there are 5 colours with 5 factories, each factory contains at most 4 tiles, we have at least 5 * 4 * 5 configurations of the factory. Besides, each player has a 5 x 5 grid state and a 1 x 7 floor state, which gives us (2 ^ 25 * 2 ^ 7) number of states. This could lead to a disastrously large two dimensional array of lookup Q table. Therefore, we moved on from Q-learning and went to do further research.

Deep Q Network (DQN), which has been implemented in many Atari games, was our second choice. Although DQN sounds very similar to Q-Learning, these two algorithms differ quite much. It utilises the exploration-exploitation method from the Q-Learning, storing the transition --- old state, action taken, new state, reward and the status that shows if the game has ended. The transition is deemed as a kind of memory. When the amount of memory accumulates to a certain amount, the model starts to replay over the value and update the Q values.
One obvious advantage about the DQN, compared to a Q-Learning method, is that it does save space when computing the input states (number of states) and output states (number of actions) . Since DQN contains multiple layers, each layer then contains various numbers of neurons, this is very subjective to personal experience and might lead to performance discrepency. Therefore, to achieve an optimal performance, we might need to change the number of layers and the number of neurons per layer several times.

While implementing the DQN, we have encountered a few difficulties. Firstly, there is the lack of proper resources other than simple tutorials over games like a 2D game called ‘cartpole’. Thus, we spent a lot of time reading up the documentation of keras framework. In addition, since this is a two-player game, it is a bit difficult to simulate each player’s behaviour without using the game engine. Hence, we integrated the online learning algorithm into a player and let it select moves based on epsilon-greedy technique. Moreover, how to define proper rewards and unstable target states have become another concern, we treat rewards simply by subtraction between the player's score for the new state and ending of each round as one episode. Additionally, we are using the Keras framework, which has a strict input standard for models and high requirements for laptop hardwares. Our laptop with Ubuntu 18.04 LTS and NVidia GTX 1060 installed has failed to initialise a tensorflow instance, for it kept raising errors about CUDA drivers. On the other hand, we preprocess each game state to the player's grid state concatenated with the player's floor state, to meet the input requirement into a DQN model.

Although our model has not been able to run due to the time constraint and the limitation of hardware, we have read up on further research papers and have found areas of improvement. For example, a double Q learning, by creating another model as target model, might further reduce the overestimation tendency of our DQN [5]. We can also improve our judgement by modelling the opponent's state [6] and predicting their actions as well, so that we can undermine or sabotage their plan where we tend to gain scores quite a lot. Meanwhile, the longer training time for the DQN model would definitely make the weights converge better to the optimal allocations.

Conclusion

In conclusion, the best strategy to be used in board games is combined technique, which we do not have enough time to explore with it. However, we did stand on the shoulder of the giant to implement some of the strategies that we have learnt. Minimax, Monte Carlo Tree search, and deep Q-learning have their own advantages and disadvantages while being used in Azul board games. The future in Artificial intelligence is bright. With the experiment of implementing all these strategies, we learnt to appreciate the hard work computer scientists have put in in the research field of AI. Minimax, MCTs, and deep Q-learning are all famous and widely used AI techniques in everyday life around us. Most importantly, it is the same kind of feeling of bearing a baby when creating our own AI: joy, fun, anticipation, and a hope for the better future.

References:

[1] Law, Keith (September 28, 2017). "It's Time for Some Game Theory with the Beautiful Azul". *Paste Magazine*.
[2] Michael C. Fu (2018). "MONTE CARLO TREE SEARCH: A TUTORIAL"
[3] Anno Accademico (2013-2014). 'Monte Carlo Tree Search algorithms applied to the card game Scopone"
[4] SIMON KLEIN (2015). "Attacking SameGame using Monte-Carlo Tree Search"
[5] Hasselt et al.(2015). "Deep Reinforcement Learning with Double Q-learning".
[6] He He, JG, KK, HD (September 18, 2016). "Opponent Modelling in Deep Reinforcement Learning".
[7] Samuel H. Fuller, John G Gaschnig(1973). Analysis of the alpha-beta pruning algorithm
[8] Silvia Garc´ıa D´ıez, J´erˆome Laforge, Marco Saerens (2012). Rminimax: An optimally randomized MINIMAX algorithm

Individual reflection --- Mingda Dong

1) The most important thing I learnt about working in a team is the importance of a vibrant and self-disciplined atmosphere in the team. Due to the outbreak of the COVID-19, many of the project works have been shifted online and group mates have to meet online via Zoom. Our group, SaltedEgg, is the only group among all my other groups where everyone proactively joins the discussion in the Wechat group, willingly gives suggestions to help other teammates whenever they get stuck and need inspiration, and actively opens the mic and the camera when joining a zoom meeting. All of these behaviours have created a vibrant atmosphere for the team, which is conducive for unhindered communication and thereby greater efficiency.
2) A lot of the techniques and methods we have learnt so far are not model-free, which means we need to have good domain knowledge and thus come up with a good feature engineering process. Weight allocation then becomes somewhat like alchemy, where adjusting the values bit by bit to get good results. After getting in contact with a model-free algorithm --- Deep Q Network, I find that it opens up the horizon of things we can build, as it is really powerful enough to explore areas that we do not yet have sound domain knowledge. From my perspective, deep learning is really promising.
3) I did not succeed in building a successful DQN model in the end, so I would like to say that my best aspect of performance in the team is my relentless effort to keep trying. As one of my teammates swapped from heuristic search to game theory strategies, I chose to leave game theory and explore further into Reinforcement Learning such as Q learning, which I later found to be inappropriate for training. Therefore, I moved onto building a Deep Q Network. Although the model cannot run by itself in the end, I have learnt a lot more since the journey started.
4) The area that needs dire improvement is the habit of designing the implementation before putting into action. For example, one of our teammates did Monte-Carlo Tree Search, he planned his design out and finished within 1 day. My way of doing DQN was a bit of being scattered around, doing coding all over the place then trying to find a way to link them all together in the end. I guess that is also part of the reason why I did not get to get the DQN agent up and running. Therefore, in the future, I would say I will do planning and designing for the most of the time, once a rough idea is formulated, writing code will have become so much easier.

Individual Reflection --- Xiande Wen

1) I made two wonderful friends with this teamwork. I learnt that we can be more creative by exchanging ideas and ambitious thoughts. By having at least one working method,

we can freely explore more challenging algorithms. Also, driven by others' expectations, everyone in the team wanted to have the best efforts to achieve an even better result. And with scheduled meetings and deadlines, we were able to maintain a high working efficiency throughout the whole process. More importantly, with such a situation this year, this teamwork really brings us a sense of being involved in the community which had been absent for a long time!

2) From this project, I learnt that Artificial Intelligence is nothing mysterious but can be achieved by various kinds of mysterious algorithms! I learnt how we can follow the procedure in other researchers' reports to build more challenging methods and how we can use their statistics and conclusions to analyse the behaviour of our own methods.
3) I think my best performance in our team is that I chose a simple method to work with and made it working as our baseline method. Although my teammates' methods are more powerful and have more potential, they require more time and efforts to make the methods work with the time limit and to make further improvements. So we are glad that we got a not-the-best but working-as-expected method in the end while we also got a chance to explore more challenging algorithms!
4) The area that I need to improve the most is that I need to manage my own time more wisely. During the time doing this assignment, I also got many others on which I put a lot more time. If I can arrange the time wisely, I might get more time to improve the method for a better performance. The evaluation function for the minimax can be further improved as we only used the difference between our player's score and the opponent's score with nothing else being considered. Also before we recursively explore the search tree, we can give the available moves based on our knowledge of the game in order to further reduce the search space in addition to alpha-beta pruning.

Individual Reflection --- Qing Feng Chye

1) Working in a team is absolutely different from working as an individual. First of all, working in a team required a heavy load of communication. Second, delegation of the task should be fair and the difficulties of the tasks delegated should be equal. Third, handling the flow of the project is more challenging than working as an individual. However, teamwork makes the dream work. In this group, we used wechat as a way of

communication. We have a proper meeting and a stand up meeting every week by using zoom. Besides, we list out the tasks needed to be done, and we all voluntarily choose what is our interest. The delegations of work were done extremely well, and the atmosphere in the Salted Egg team is incredibly joyful and loving. Working in a team also meant that we will face challenges and conflicts. However, that does not stop us from communicating and heading towards the right direction. We always provide constructive feedback for one another.

2) Artificial Intelligence is incredible. By implementing the MCTs strategy that was mentioned in the lecture, I get to see the 'behind the scene' work that has been done in this field. I was always amazed by the incredible performance of artificial intelligence, and never thought I can create one on my own. This project has truly challenged me to step out in faith and believe that I can do it. In order to learn more of the MCTs strategy I was going to implement, I have to do some research on the resources related to MCTs. Also, there are challenges along the way, when I just did not know how to start and construct a proper model. However, I would say that I have overcome the fear and finally produced the reasonable end product.
3) One of the best aspects of my performance is producing a MCT player. I was inspired by the performance of AlphaGo, and thought of what if we can do something similar to it. Furthermore, I set up a zoom meeting for everyone whenever we need to meet up.
4) In terms of timing, I felt that I can do better at estimating the time needed to perfect the strategy. I have successfully implemented the strategy 3 days before the due date. However, I realized that we need to do offline training in order to eliminate the restriction on time out. The steps required us to explore Q-learning and incorporate with linear regression, to assign the probability for moves. By the end of the project, I did not manage to finish it. If I can do this, I am confident that we will be one of the best teams in the Azul board game tournament.