COMP 424 Final Project Report

Group adeideidei: Byron Chen (260892558) and Kevin Li (260924898)
Emails: Byron Chen (yifei.chen@mail.mcgill.ca) and
Kevin Li (kevin.li3@mail.mcgill.ca)

1. Introduction

In this project, an Artificial Intelligence Agent is implemented for playing the game Colosseum Survival[1]. This project aims to win as many games as possible with this rule without knowing who would be the opponent. Throughout the whole development of this project, many approaches for choosing the action have been implemented and tested, i.e. Monte Carlo Tree (MCT) Search, Minimax Search, Breadth-First Search (BFS), and Best-First Search.

2. Technical Approach

In this project, an action (position and direction for the barrier) is defined as a move for the agent through the game. Furthermore, a state (node) for the search algorithms is defined as the players' positions and barriers on the chessboard. Through the development of this project, Monte Carlo Tree Search, Minimax Search, Breadth-First Search, and Best-First Search are implemented and tested to choose the action for the AI Agent. In the final version of this project, all the methods have been combined except Monte Carlo Tree Search. In the rest of this section, each approach's implementation will be explained in detail with some theoretical basis.

2.1 Breadth-First Search (BFS)

Breadth-First Search is a basic uninformed search algorithm. This algorithm traverses all the nodes in a graph layer by layer. This method was implemented by the start code provider[1] and is modified in this project. This search algorithm is used to get all the valid actions that can be made according to the agent's current position.

2.2 Monte Carlo Tree (MCT) Search

Monte Carlo Tree Search is a standard and efficient method among all the search algorithms. This algorithm is combined with the search tree and random simulations. While doing this search, the agent will first choose several actions according to the tree policy and then do the random simulations according to the default policy till the end of the search. The final results for each simulation will be used to choose the best action by back-propagation. As a result, this algorithm is an expensive search to use.

A basic version of this algorithm is implemented and experimented with a random agent in this project. Since the agent is only allowed to take 30 seconds for the first step and 2 seconds for each later move, the algorithm cannot simulate many steps before or during the

1. Check the detailed description of this game in the project description [1].

game. Although all the actions simulated during the first 30 seconds are stored and used as references for future moves, whenever a new state appears that has not been searched before the first action, the agent's move would act with no difference from the random moves. Because of the limited time, Monte Carlo Tree Search is not entirely appropriate in this case, and this algorithm is then deleted from the list of approaches.

2.3 Best-First Search

Best-First Search is an informed search algorithm that traverses the graph by expanding the most promising node according to several evaluation criteria. This method was primarily used for selecting the most profitable action based on the heuristic values, namely scores, of each action, and several criteria calculate the scores. The rest of this subsection will explain how different situations could affect the heuristic value of each state.

2.3.1 Distance between the Agent and the Middle of the Board

The first criterion is related to the distance between the destination of the agent after the action and the middle point $(\frac{boardsize}{2}, \frac{boardsize}{2})$ of the chessboard. And this distance is compared with that between the original position and the middle point. If the distance is smaller than the previous one, the score of this action is increased by $(20 - distance)$; decreased by $(20 - distance)$ if it is bigger than the previous one in the opposite situation. The constant 20 in this criterion got from many experiments. This criterion gives the agent the bias of moving towards the middle.

2.3.2 Distance between the Agent and the Opponent

This criterion is related to the distance between the agent and the opponent. It is similar to the previous criteria; if it is closer to the opponent than the original position, the score would be increased by $(30 - distance)$ and decreased by $(30 - distance)$ in the opposite situation. This criterion would give the agent the bias of moving toward the opponent, i.e. the agent becomes more aggressive. The constant 30 in this criterion got from many experiments.

2.3.3 The Placement of the Barrier

This criterion is related to the the direction of the barrier to put. The score would be increased by 40 if this action placed the barrier on the side with the opponent. Furthermore, this criterion combines the previous heuristic function, which gives the agent the tendency to block the opponent.

2.3.4 Number of Barriers Around

This criterion is related to the number of barriers around the destination of the action. The score would be decreased by $(60 \times No.\ Barriers\ Around\ Destination - 1)$. This criterion is set to prevent the agent from entering a position with multiple barriers, increasing the danger of getting blocked by the opponent. Therefore, the selected action would be safer compared to other actions.

2.3.5 Positions Next to the Borders or Same Positions

This criterion is related to the destination positions. Action's score will be decreased by 20 for every border next to the destination and 10 if it does not move. If an action moves the chess to the corner of the chessboard, the score would be deducted by 40 in total. This criterion is done since such action might increase the chance of getting blocked by the opponent and decrease the number of possible actions in general cases.

2.3.6 Blockade Unnecessary Blocks

This criterion ensures that the selected action does not tend to blockade an area that might be counted as its score for the match at the end. The action score would be deducted by $(40 \times No.\ Blocks\ Blockaded)$.

2.3.7 Result of the game

This criterion is to verify if an action can win the game directly. If such action exists, this action would be directly selected. However, if an action can lead to failure, it would be deleted from the checklist.

2.4 Minimax Search

After implementing the Best-First Search, a little Minimax Search is also implemented and included inside the Best-First Search, which can also affect the heuristic value of the different states. Minimax Search is an algorithm that minimizes the possible loss in a worst-case scenario. This method was implemented during the phase of selecting the actions.

Due to the time constraint, it is impossible to implement the entire Minimax Search. Therefore, the Minimax Search Tree is only expanded with a depth of 2, and only two steps are considered in the selection process.

The first process of the Minimax Search is to pick the 4 best actions based on the heuristic value of the action if the total number of valid actions is larger than 10. Moreover, since there are a lot of available actions at the beginning of each game, it may take much time to compute the heuristic value of each action. Therefore, considering that many available actions exist for the agent, it is unlikely that the agent will choose a risky move. A threshold of 30 is set to minimize the required time to select an action. If there are more than 30 valid actions for the agent, only 2 of the best actions would be chosen for the following process. In the second process, for each selected action, the opponent would choose the best action based on their heuristic values with the new position of the agent and the updated chessboard. In the third process, the agent would select a further action based on the opponent's action. The score for the following two actions would be added and minus by the score of the opponent's action at last. The best action for this state would be chosen based on the newly calculated score.

This Minimax Search assures that the agent does not lose the game in two steps, assuming that the opponent has the same goal as the agent does unless no more available actions are available.

3. Motivation for Technical Approach

At first, Monte Carlo Tree Search was chosen as the primary function to be implemented and optimized since 30 seconds are given before the first move of the AI agent, and it can reach deeper compared to Minimax Search. The time was considered enough to run many states by Monte Carlo Tree Search. However, after implementing and experimenting with a basic version of Monte Carlo Tree Search, only a few states are simulated. Moreover, the agent's behaviour is similar to the random agent most of the time since only limited nodes could be simulated due to the time constraint. This result changed the focus point for this project from Monte Carlo Tree Search to others.

After playing a couple of games with different people, some strategies for winning this game were brought forth. By ranking different strategies and assigning the values for each strategy, the idea of using the Best-First Search was put forward.

During the implementation of the Best-First Search algorithm, the time efficiency for this algorithm was found to be high. This feature allows predicting the possible opponent moves and the agent's following action based on the opponent moves, which is how a Minimax search works.

4. Pros/cons of Chosen Approach

At first, Monte Carlo Tree Search was chosen as the primary function. Compared with the Monte Carlo Tree Search, the chosen approaches are much more time-efficient which is more appropriate for this project. Apart from this, the chosen approaches remove all the uncertainty during the search procedures. However, the results for these approaches are not entirely optimal. Monte Carlo Tree Search allows the search to go deeper than the Minimax Search since it requires less computation. Therefore, it is more forward-looking than Minimax Search.

Furthermore, the heuristic values were defined based on the team's understanding. They only follow the team's strategies considered to be useful, which are very subjective. So if it plays against an opponent that plays in different strategies that are not considered before, it might perform poorly. For the same reason, the Minimax Search is also not optimal. Furthermore, due to the depth limit, if the agent plays against an opponent that is more forward-looking, it might also lead itself to failure without any recognition.

Moreover, the only method to improve the performance of the chosen approach is to have a better heuristic function. However, the parameters for the heuristic functions are only modified according to the human experience in this project. As a result, the chosen heuristic functions are not optimal.

5. Further Improvements

Since the limited development time for this project, some ideas have not been achieved in this project. The rest of this section will introduce each idea for future improvement.

5.1 α-β Pruning

Minimax Search is used in this project, but this agent predicts only a few steps due to the time limit. As a result, it is vital to implement α-β Pruning to make the agent more time-efficient. This algorithm is a better version of Minimax Search since it would prune some of the branches in the search tree to save time.

5.2 Combination with Improved Monte Carlo Tree Search

Although the Monte Carlo Tree Search is a time-consuming algorithm, it is still an optimal algorithm if many states are searched by it. As a result, it is a good idea to focus on reducing the time consumed by using the Monte Carlo Tree Search. Furthermore, current approaches can also be implemented as the tree policy for the Monte Carlo Tree Search for higher efficiency.

5.3 Optimization for the Heuristic Functions

This agent mainly depends on the heuristic values for the Best-First Search, namely score, for different situations. However, all the scores for different situations are assigned by a human after times of experiments. As a result, the scores assigned for each situation can not be guaranteed to be the optimal ones. It means that it is essential to collect all the features together and create a heuristic function. After this, the heuristic function can be optimized using optimization algorithms. Many optimization algorithms can be used to get the optimal heuristic functions, i.e. Simulated Annealing, Parallelism and Beam Search. These two optimization algorithms are worth using in future experiments to get the most appropriate heuristic function instead of manually setting values.

5.4 Combination with Convolutional Neural Network (CNN)

After improving all other approaches, combining them with Convolutional Neural Network (CNN) could be meaningful. By training CNN for recognizing different states of the chessboard, it will be much more efficient for the agent to get the information directly from the chessboard for choosing the action to move. Furthermore, combining CNN with Monte Carlo Tree Search is also worth to be experienced. If this is achieved successfully, all the heuristic functions can be ignored, and higher efficiency can be achieved[2].

6. Conclusion

In this project, an agent combined with Breadth-First Search, Best-First Search with many heuristic functions and Minimax Search is implemented for playing the Colosseum Survival game. After many experiments, the heuristic functions were optimized. Furthermore, the basic Monte Carlo Tree Search was found to be inappropriate for this project according to the time limit of each action.

2. This idea is enlightened by Memo Akten [2]

7. Work Distribution

Both Byron and Kevin participated in all parts of the project. Byron is mainly responsible for implementing Breadth-First Search and Best-First Search and report management. Kevin is mainly responsible for implementing Monte Carlo Tree Search, Minimax Search, and optimization.

References

[1] Project Description.

[2] Collaborative creativity with Monte Carlo Tree Search and Convolutional Neural Networks (and other image classifiers). Memo Akten, Dec 14, 2016. Website:

https://memoakten.medium.com/collaborative-creativity-with-monte-carlo-tree-search-and-convolutional-neural-networks-and-other-69d7107385a0