COMP30024 Artificial Intelligence Project Part B Report – Freckers Agent

Frank (Feifan) Xia, Student ID: 1347999

Jiaxin He, Student ID: 1455893

1. Introduction

This report showcases the design and strategic implementation of an AI agent for the game Freckers, developed for Project Part B of COMP30024. The agent utilises the alpha-beta pruning algorithm, with a well-designed heuristic evaluation function to guide decision-making. The design significantly improves the efficiency of a regular minimax algorithm and performs well under different constraints.

2. Selection of strategy

Initially, we put our focus on both the MCTS and the alpha-beta pruning algorithm. We implemented a complete MCTS on the Frecker game, however, the algorithm did not perform well, as it takes on average 140 turns to complete the game, and sometimes the game would end in a draw as it reaches 150 moves.

We then worked on the optimisation of the MCTS algorithm, such as adding a heuristic function that encourages reaching the goal state, penalises slow movement and rewards longer jumps, and adjusting the exploration constant used in the UCB1 formula, and time budget tuning that can result in deeper search. We have optimised the results to around 120 turns, but we thought that there is still a large room for improvement in terms of complexity and the number of turns. Thus, we decided to continue implementing our alpha-beta pruning algorithm.

Eventually, we found success with our alpha-beta pruning algorithm, as our version of alpha-beta outperforms the MCTS algorithm both in the time taken for each move and also in the number of turns taken to win a game.

3. Evaluation (heuristic) Function Design

We have designed a few different versions of our heuristic function and tested them against our own agents. However, the results are not what we expected:

A more complex heuristic does not outperform a relatively simple, generic heuristic function.

We did some research to see what the reason was behind it; we found out that it is related to something called overfitting in machine learning. Alpha-beta pruning relies on relative comparisons, not precise scores, so a complex heuristic function may add noise into the process and overfit in special situations. Moreover, a more complex function often means longer evaluation times. Since we have a time budget (1 second in our case), this could reduce the number of states the agent can explore, which in turn weakens the quality of decision-making, especially in the early or mid-game.

Therefore, we have tested and chosen an evaluation function that combines multiple factors and is relatively simple and generic:

- Distance to goal: Encourages frogs to approach their target row. For RED, this is row 7; for BLUE, row 0. The score rewards reduced aggregate distance.
- Mobility: Computes the difference in the number of legal moves between the agent and its opponent. A higher value suggests tactical flexibility.
- Pad adjacency bonus: Rewards frogs positioned next to lily pads. This encourages formations that enable future moves or jumps and discourages isolation.
- Terminal states: If the game is over, a win results in a score of +∞ and a loss in -∞.

These components are combined as:
Evaluation Score = 50 * Distance Score + 10 * Mobility Score + Pad Bonus

Weights were manually tuned through informal test games to ensure proper balance.

4. Performance and Testing

The agent was tested by running games against our agent itself with the command prompt: “python -m referee agent agent”.

Previous MCTS agent performance:

Our initial MCTS design can finish the game with around 140 moves, and sometimes can even end in a draw. After multiple attempts at optimisation and testing, we reduced it to 120 moves, however, it is still not good enough.

Alpha-beta pruning performance:

The first version of our alpha-beta pruning algorithm, on the other hand, can finish the game with a winner in around 80 turns. With further optimisation of the heuristic function and the implementation of the Action_priority() function, we have successfully reduced the moves down to 50 moves when running games against itself. Moreover, we have manually tested and tuned the depth of the alpha-beta search to be 3, as this is the best-performing value; the time taken for the algorithm to run has significantly decreased with this version. To tune the constants being used in the heuristic function, we conducted multiple trial games to test different weight combinations in the heuristic function. For instance, setting the mobility term too high led to slower progress and longer games (~65 turns), whereas

increasing the distance weight to 50 and reducing mobility to 10 achieved more efficient forward movement (~50 turns on average). These results are very stable when both players use the same agent.

5. Technical Details, Optimisations, and Complexity Analysis

Several implementation highlights of the alpha-beta pruning agent:

- Immutable game state cloning: Each state is copied for simulation, avoiding shared mutable state and bugs.

- Separate GameState class: This class is responsible for the basic game logic. This separation makes the code easier to modify or tune. For example, if anything needs to be changed, it can be modified in the Agent class, and it will not affect the basic game logic.
- Action prioritisation in get_legal_actions(): A function inside the get_legal_actions() function that prioritises jump moves and up and down movements over left and right movements.
- Visited landings tracking during jump sequence generation prevents infinite loops and redundant paths.
- Minimal reliance on unnecessary external data structures keeps memory use low and lookup times fast.

- A well-designed heuristic function that considers multiple factors.

Optimisation:

We utilised several methods to optimise the performance of the alpha-beta pruning agent. As we know, the alpha-beta pruning agent will prune the search tree when alpha is larger than or equal to beta, which means choosing a good move to start with in a search is very important.

After observing the game, we discovered that to move to the state more efficiently, jump moves and up and down movements are more crucial compared to left and right movements. Therefore, we came up with a function action_priority() inside the get_legal_actions() function to prioritise these movements.

Complexity Analysis

- **Baseline Minimax Complexity**: Without pruning, the time complexity is exponential in the search depth d and the branching factor b, i.e., $O(b^d)$. In Freckers, b varies depending on the game phase and available jumps, often ranging from 5 to 25.

- **Alpha-Beta Best Case Complexity**: With perfect move ordering, alpha-beta pruning reduces the effective complexity to $O(b^{d/2})$, allowing the algorithm to search twice as deep for the same computational cost.

- **Effect of Action Prioritisation**: By ordering actions such that jump moves and vertical progress are considered first, the agent often triggers pruning earlier. This achieves near-best-case behaviour in many game states, especially during early and late phases of the game.
- **Space Complexity**: The space required is primarily due to recursion depth and cloned game states, resulting in an overall space complexity of $O(d)$, which is manageable at a fixed depth of 3.

6. Useful resources

We have reused a large amount of code with some modifications from the previous assignment of this subject, as the basic game logic and movement remain the same. Furthermore, the supplementary material – coding up Minimax in Python was very useful as it provided a very detailed step-by-step guide on how to build an agent similar to alpha-beta pruning.

In terms of teamwork and communication aspects, we used git for the version control, and GitHub allowed both of us to work on the same piece of code and collaborate in the same repository.

7. Conclusion

This project demonstrates a clean, strategic application of alpha-beta pruning to a novel game setting. While not using machine learning, the handcrafted heuristic and efficient search yield competitive results. We acknowledge the guidance from course materials, textbooks, in particular the Week 10 Minimax walkthrough, which provided a helpful foundation for our alpha-beta pruning implementation.