An MCTS-Based Agent for Freckers

In project part B, we are required to extend on our search algorithm in part A further, and the game rules have been extended. We now have to move all 6 frogs to the goal row instead of just moving one frog, and grow actions are allowed and necessary to proceed the game. In this case, the branching factor is very large. Each frog has 5 valid move directions, resulting in at least 30 for the branching factor. Frogs can also have chain jumps, further increase the branching factor, and every turn the agent can also choose a grow action. The branching factor in this scenario can be very large and dynamic, given that we have a maximum computation time limit of 180 seconds per player per game, the simple breadth-first search algorithm we used in part A is insufficient to finish the game, and an adversarial search algorithm is necessary.

The two adversarial search algorithms we learnt from lectures are Minimax with ***αβ*** pruning and Monte Carlo Tree Search (MCTS). We learnt from lectures that Minimax assumes opponents to behave rationally and make rational moves, and is optimal if this assumption is held. However, we have opponents that make random moves and greedy opponents who only select the most promising move at each turn. Minimax algorithm may not perform very well, especially against greedy opponents. Additionally, Minimax needs a good evaluation function to calculate the payoff of each move. In the game Freckers, we can have a chain jump, that may jump several rows, and may potentially jump to goal row in one action. A simple move may have a very low payoff in the short term, but could lead to a very useful chain jump in future, and thus result in a higher long term payoff. Therefore, designing a good evaluation function for Minimax might be difficult to achieve.

By contrast, MCTS requires no evaluation function. It simulates the utility of a current state by playing multiple games from that state, and back-propagates the utility all the way up to the root of the search tree. It also treats each player symmetrically without assuming optimal moves. Furthermore, complexity of playout in MCTS takes linear time, while Minimax takes exponential time complexity. These properties make MCTS a natural fit for Freckers' large, dynamic branching factor and stochastic or sub-optimal opponent behaviour, so we decided to use it as our core search algorithm.

We have made several modifications to the normal MCTS algorithm to make it fit Freckers better.

- First, we cap each simulate at a depth of 15—just over two full board heights—which lets playouts reach endgames without exhausting our per-move wall-clock. Second, we budget 400 search cycles (selection → simulation → backpropagation) per move. This gives us enough simulations for statistical confidence while still returning before timeout.
- We have tested several strategies to pick the best child at the end of an MCTS run. We initially pick the child that has the highest number of visits. We also tried the highest average value (Q-Value) strategy and the highest root UCB value strategy. It turned out to be using the highest number of visits strategy resulting in a better and more efficient performance.
- Unlike the normal MCTS algorithm that does not have an evaluation function, we introduced a simple heuristic evaluation that gives a weight of each move by

calculating vertical move distance and the length of jump, and we prune the move that has low weight.

- Normally, MCTS plays multiple games of current state all the way to terminal state. Given the time constraints, we introduced an early utility evaluation when a playout reaches its depth limit without terminating. We score the non-terminal leaf node by using a simple utility function that measures goal-row progress. This avoids long simulations while still providing a meaningful reward signal.
- We introduced a reroot mechanism and limit the allowed moves in the beginning of the game to improve game efficiency.

To evaluate the effectiveness of our agent, we tested our MCTS algorithm extensively through self-play and against baseline agents. In particular, we ran our agent as both red and blue to ensure balanced evaluation. Initially, we observed significant asymmetry, where red consistently performed much better than blue. This was caused by the agent making inefficient decisions for frogs that had already reached the goal row, and improper update of rewards at blue's turn. We resolved this by using a tuple to store utility in tree nodes, and modifying the move generation heuristic to deprioritize actions involving goal-row frogs, resulting in much more balanced gameplay.

We have compared multiple versions of MCTS algorithm with and without evaluation heuristics. Initially, no evaluation heuristics was used, and the search result was purely determined by UCB value. More than 100 moves in total were required for one side to win, and the search speed was very slow. We had to reduce the number of cycles and depth limit to finish within time constraints. Our enhanced MCTS limits each playout to a fixed depth of 15 and evaluates non-terminal states using a heuristic utility function. This ensures simulations are fast and still informative, improving performance under strict per-move time constraints.

We also implemented a *rerooting mechanism* in our MCTS agent. After each move, instead of reconstructing the search tree, the agent searches for the child node corresponding to the new board state and promotes it to the root. This allows the agent to reuse previously simulated paths and maintain valuable statistics, such as visit counts and utility estimates, reducing redundant computation.

Moreover, to prevent unnecessary lateral movement in the early and mid-game, we restricted left and right moves to only be allowed when frogs are close to their respective goal rows (specifically, row 7 for red and row 1 for blue). This encourages forward progress during most of the game and prevents agents from wasting moves by moving sideways when it is not useful, as well as reducing the branching factor. Although restricting lateral movement in early stages may occasionally prevent exploring useful alternate paths, in practice, this rarely happens and the benefit in avoiding indecisive frog behaviour far outweighs this limitation. Together, these modifications significantly improved our agent's efficiency, enabling us to increase our MCTS rollout count from 50 to 400 within the same time budget. Overall, our final agent consistently completes full games within the 180-second limit, avoids illegal moves, and performs competitively in both self-play and test matches, demonstrating that MCTS with informed simulation and selection strategies is well-suited for the complexity of Freckers.