COMP30024 Project B Report

Arsam Samadi Bahrami (1263236)
Saaiq Ahmed (1272488)
Granth Saxena (1183511)

I. Approach

Search Algorithm

After various attempts at experimentation to create an agent that is both effective (makes desirable move decisions) and efficient (successfully runs within the time limit), we finally came to implement a minimax algorithm of depth 2, optimised via alpha-beta pruning and an initial sorting function, and made more effective by prioritising ‘Spread’ moves. In an attempt to meet the time and space constraints posed by the program, alongside the real-world time constraints of the assignment deadline, we ruled out various other algorithms including Depth 3 minimax, dynamic depth minimax, and Monte Carlo Search Tree.

Evaluation Function

Our implementation’s evaluation function measures the power differential between our agent and the opponent where losing positions are assigned a value of negative infinity while winning positions are assigned a value of positive infinity. This approach allows us to prioritize the most desirable moves for our agent while avoid moves that may increase the opponent's chance of success.

The decision-making capabilities of our agent was also improved through the strategic prioritisation of ‘Spread’ moves. Through manual testing and rigorous analysis of various agent programs and move sequences, we observed a consistent advantage associated with executing ‘Spread’ moves. While a ‘Spawn’ action on its own can only allow for a $+1$ increase in power, a ‘Spread’ move can take over multiple enemy cells, giving advantage to our agent and taking away board power from the opponent. A ‘Spread’ move can also be used as a defensive mechanism when a high-powered cell can spread itself in an attempt to escape being taken over by a higher-powered enemy cell. As a result, we chose to prioritize ‘Spread’ moves during the search process in order to give us a slight advantage over opponents with similar implementations.

We made efforts to expand on the efficiency of our evaluation function. While the power-based evaluation provided decent results, we also recognized the need to consider other factors that could influence the outcome of the game. Incorporating additional heuristics, such as piece positioning or board control, could potentially improve the overall performance of our program. However, devising and fine-tuning these heuristics would require extensive experimentation and analysis which we regrettably did not have the time to achieve.

II. Performance Evaluation

Sorting

We attempted to implement an *order_moves* sorting function to choose the best potential move as decided by our evaluation function of the power differential. However, applying the sorting function at every call of the *get_moves* function was computationally expensive and significantly slowed down the performance of our algorithm, occasionally even timing out after reaching the time constraint. Subsequently, we decided to limit the use of the sorting function to the beginning of the search process which boosts our agent's chance of success by prioritising the most potentially desirable moves without the drawbacks of the previous implementation. When testing the sorting function on a Depth 3 minimax, our program went from timing out in six moves to being able to reach ninety-five moves.

Depth 3 Minimax

We also experimented with the depth of the minimax search itself. Attempting to implement a Depth 3 minimax search led to significant performance issues and timing out alongside potential risks in long-running games. As the game progressed and new cells were added to the board, the number of possible moves grew exponentially (a single player cell has 6 possible 'Spread' moves while an empty cell has only the one possible 'Spawn' move) resulting in a time-out generally beyond the 80^{th} move, rendering the approach impractical.

Dynamic Depth Minimax

To address these issues, we explored dynamic depth, where the search depth is adjusted based on the player's advantage. When the player had a substantial lead, we limited the depth to 2, as looking further ahead was unnecessary. However, this approach still suffered from slow performance in the early game and would prove to be ineffective against an opponent of similar competence. In a scenario where both players end up with a similar number of pieces, using Depth 3 would result in timing out.

Threshold-based Minimax

A potential threshold-based approach was considered but never implemented due to lack of time to conduct proper testing. This approach would utilize Depth 3 searches until a certain threshold of cells on the board were taken before switching to Depth 2 searches. This could potentially address some of the performance issues, but its actual effectiveness remains untested and a potential point of future research.

Machine Learning

We investigated the potential of training a neural network to make move decisions based on historical gameplay data to create an agent based on a Machine Learning model. The network could learn from past moves and games and adjust its decision-making process accordingly. While this approach could greatly improve desirability of moves, we did not have the time to train a sufficiently large and diverse dataset of games, or to implement a model that could operate within the project constraints.

Multiple Algorithms

We briefly attempted to explore the concept of having multiple agents that would locally compete with each other based on their own strategies and ultimately decide the most desirable move. This proved to be beyond the scope for this project and would potentially not follow the time and space constraints but is an interesting avenue of exploration for future improvement.

Advanced Testing Tools

We would have liked to develop additional diagnostic tools to visualise and analyse various aspects of Infexion such as move patterns and the effectiveness of different heuristics. These tools could have allowed insights into the dynamics of the game and our agent's decision-making which would allow us to find natural advantageous strategies in the game, fine-tune our evaluation function and pruning, and to potentially explore the impact of other searches or algorithms. Although beyond our scope, we did make some progress into creating these diagnostic tools but chose to prioritise completing the actual search program instead.

Monte Carlo Tree Search

Additionally, we explored how a Monte Carlo Tree Search could have been implemented for our game, especially due to its proven advantages in strategy games. However, we found that our implementation was timing out even at Depth 2 simulations, running slower than our Depth 2 minimax algorithm. Perhaps the Monte Carlo approach would have been preferred if time and space constraints were different or if parallel processing was allowed, but under the current scope of the project, we decided to instead implement a minimax depth of 2, striking a balance between efficiency and accurate decision-making.