Approach to Infexion

For the full two-player version of Infexion, we utilised the **α–β algorithm** to choose the best strategy in playing the game. We decided on this algorithm due to its potential against deterministic and perfect information games (as is the case for Infexion) as well as its reduction of branching factors and in turn, the reduction of its time and space complexity without compromising the optimality of the best move.

We also considered the Monte Carlo Tree Search, but since it requires a large space to store playouts, the time and space limitations of the game would affect its optimality as proven by studies which showed how it had a worse performance in some deterministic and perfect information games (Kato et al., 2015; Papadopoulos et al., 2012). Hence, the α–β algorithm would be a better choice.

Implementation of Alpha-Beta Algorithm

When building the minimax tree, we utilised two priority queues:

1. **curr_pq** to keep track of nodes to be expanded in the **current depth** of nodes we're expanding.
2. **next_pq** to keep track of **child nodes** to expand once we've 'ran out' of expanding nodes in the current depth.

In addition, since we only have 180 seconds to compute solutions for the whole game, we set an internal time limit for building the tree with each turn set to a maximum of **referee["time_remaining"]/turns_left**.

Below are the main steps to our approach:

1. In order to select an action, the program uses **breadth-first search** in order to search for all possible moves from the current board state by popping nodes from **curr_pq** one by one. These then generate **child nodes**.
2. The **child nodes** are stored in **next_pq**, with the highest priority move being the best one in terms of our evaluation function.
3. Once nodes in curr_pq run out, it means we have finished conducting breadth-first search on a certain depth of the tree. Set **curr_pq = next_pq**, and repeat steps 1-2 until the internal time limit is reached.
4. Conduct α–β pruning on the built minimax tree.
5. Return the action which results in the best score for MAX (since we programmed our MiniMax tree such that our agent would always be MAX).

Time and space Complexity

With regards to time and space complexities, we define the following terms:

- b: the maximum branching factor of the search tree; in other words, the maximum number of possible actions the player can take
- m: the maximum depth of state space

The algorithm has a space complexity of $O(bm)$. While it does not ensure perfect ordering when pruning, it performs much better than minimax. Hence, we can conclude that its time complexity is a lot higher than $O(b^m)$ and slightly lower than $O(b^{m/2})$.

Evaluation function

With regards to the main evaluation function, we calculated the ratio between the total power of the MAX and MIN tokens in the board (the colour of our minimax agent is always regarded as MAX).

$$TokenRatio(board) = \frac{TotalPower(MAX\ tokens)}{TotalPower(MIN\ tokens)}$$

Our experiments indicate that spreading too early (more specifically, when the ratio is not satisfied) could lead to a loss in the long run. Initially, the evaluation function is the ratio between the number of MAX and MIN tokens, but we found that the current ratio worked better.

In addition, since there tends to be several board states with the same token ratio value, we also used the total token power our agent has at the board state as a tiebreaker. Using this as a tiebreaker gives the agent incentive to use moves which increase power so it can take more tokens in the future. The performance of the agent improved after this tiebreaker was implemented, winning against its non-tiebreaker counterpart.

We'd like to note that token ratio is still the main evaluation function over token power because even if our agent has a lot of power in a certain board state, the opponent agent could still have even *more* power and in turn, more offensive capabilities against our agent – token power is unable to account for this.

Search Optimizations

1. Intentionally generated spread moves before spawn moves
After playing games of Infexion, we noticed that spread moves tend to result in bigger shifts in the game (e.g. being able to take over a large number of enemy tokens, or having many of our tokens taken), resulting in more extreme evaluation function scores compared to spawn moves. Thus, generating spread moves first (they would be at the 'left half' from their parent node, while spawn moves would be at the 'right half') helps α–β pruning.

2. Ignored moves which spreads a token of power 1 to an empty cell
Based on our experiments with playing the game, we found that such moves serve no utility to winning the game.

3. α–β pruning
The α–β pruning algorithm itself is an optimization of the minimax search strategy. With perfect ordering, it would have double the depth of search in a shorter amount of time. While we were unable to implement perfect ordering, our algorithm is still much better than minimax, taking about 0.01 seconds on average to find the best eval-score move to return from the built MiniMax tree as compared to an average of 0.7 seconds on normal MiniMax.

4. Priority queues to build minimax tree instead of queue
Typically when building a minimax tree, nodes to expand are just added and popped to the same queue throughout. For our algorithm when generating the child nodes of all nodes in a certain depth *x*, we instead place the children into a *priority* queue (separate queue from nodes of depth *x*) so that the more promising nodes can be expanded first, considering the time constraints of building the tree.

5. Limited the number of nodes per depth of the minimax tree
There is a tradeoff between the space complexity and the optimality of the algorithm. Through experimentation, we found an optimal node limit that balances these two factors, which is 10^4 nodes. Since we expanded nodes by their priority value as discussed in previous sections (where highest priority is the best move), the algorithm does not lose much by disregarding the nodes beyond this optimal node limit.

Performance Evaluation

To have a baseline as well as a better understanding of how the base code works, we created random, greedy, and normal minimax algorithms in such an order before going for the α–β algorithm.

We've tested these programs against each other at least 5 times with each player having the chance to play both as red and blue tokens. There are no limits, aside from the constraints specified in the project specification. The results of which are seen in Table 1 as shown below:

Win Table

VS	Main strategies				Total Win Rate
	Random	**Greedy**	**MiniMax**	**α–β**	
Random		L	L	L	0%
Greedy	W		L	L	33%
MiniMax	W	W		L	66%
α–β	W	W	W		100%

Table 1: This table shows how the algorithms on the first column fares against those on the first row.

It is important to note that the outcome was the same regardless of which player goes first. Hence, we can conclude that the α–β algorithm is the most effective.

Aside from the α–β algorithm, these are the other search strategies we attempted which did not work as well, from worst to best:

1. **Random algorithm**: Spawns and spreads randomly around the board without any strategy.
2. **Greedy algorithm**: Calculates the best spread move which is determined by the number of opponent's tokens that have been conquered. It spawns only if the power limit has not been reached and none of the opponent's tokens have been conquered. The location for spawning would be around the current token and is not within the range of any of the opponent's possible spread actions. Likely didn't perform as well since it only looks one move ahead.
3. **MiniMax algorithm**: Beats the greedy and random agents easily, however still loses to α–β pruning because it takes a lot longer to find the best move in the built minimax tree.

References

Kato, H., Fazekas, S. Z., Takaya, M., & Yamamura, A. (2015). Comparative study of monte-carlo tree search and alpha-beta pruning in amazons. In *Information and Communication Technology: Third IFIP TC 5/8 International Conference, ICT-EurAsia 2015, and 9th IFIP WG 8.9 Working Conference, CONFENIS 2015, Held as Part of WCC 2015, Daejeon, Korea, October 4-7, 2015, Proceedings 3* (pp. 139-148). Springer International Publishing.

Papadopoulos, A., Toumpas, K., Chrysopoulos, A., & Mitkas, P. A. (2012, September). Exploring optimization strategies in board game abalone for alpha-beta search. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*(pp. 63-70). IEEE.