COMP30024: ARTIFICIAL INTELLIGENCE
EXPENDIBOTS PLAYER

Jason Polychronopoulos
Student ID: 921565

Deevesh Shanmuganathan
Student ID: 912364

October 12, 2020

ABSTRACT

Our agent took what we found was strong in part A of our project, creating large stacks of tokens to overcome obstacles, and we defined a strategy revolving around 'sniper' play where pressure on the opponent is applied from a distance. We also took inspiration from the MP-MIX Algorithm [1], specifically by applying different search strategies at different stages of the game.

1 Approach

Our agent was built upon the negamax adversarial algorithm, a variant form of minimax which takes advantage of the zero-sum property of two-player Expendi-bots to allow for a more compact implementation. This is achieved by avoiding the need for duplicate if statements for the maximiser and minimiser. This implementation is based on the fact that max(a,b) = -min(-a,-b) in alpha-beta pruning

1.1 Early game

The early game focuses on getting presence in the middle of the board. We experimented with book learning to derive a set of strong opening moves. The first 4 moves focus on moving pieces in rows zero or seven up to rows three or four depending on which colour the player is. This allows the agent to provide some frontline coverage in the early stages of the game. These moves were stored as dictionaries for each player and would be depleted, at which point the mid game would begin.

1.2 Mid game

The early game focuses on getting presence in the middle of the board. We experimented with book learning to derive a set of strong opening moves. The first 4 moves focus on moving pieces in rows zero or seven up to rows three or four depending on which colour the player is. This allows the agent to provide some frontline coverage in the early stages of the game. These moves were stored as dictionaries for each player and would be depleted, at which point the mid game would begin.

1.3 End game

While the MP-MIX algorithm has no special properties in two-player games (as MaxN degrades to minimax and paranoid is intended to degrade n-player situations to two-player), we took the idea of switching search strategies depending on our agents evaluation of the game state. In particular, our end-game strategies would switch depending on how much of a piece advantage our agent has over the opponent.

1.3.1 End game scenarios

Triggering an endgame would be the result of running near the last 10 seconds of our agent's computation time, which would result in various situations we found commonly occurred during our testing:

- If our agent is winning by a significant margin or losing by a small margin, we would continue to play our 'sniper' strategy, albeit with higher weightings on aggressive play.
- If our agent is losing by a significant margin, they would have to 'make a play', which would result in taking a greedy action. This action would simply be looking to cause the largest amount of damage possible.
- If our agent is winning by a small margin, we would switch to a 'defensive' playstyle which would involve keeping our pieces as far away from the opponent as possible while maintaining as close to a one-two block boom radius as possible in order to avoid collateral damage.

2 Heuristics

We used the following functions and relations to evaluate states throughout the game. We designed them with zero-sum in mind to work with Negamax. Weightings for meta-strategies were determined experimentally.

2.1 Baseline

Calculate the absolute difference between the players' pieces and the opponents. We use this both in more complex strategies and to generally evaluate the game board to determine which strategy to use in the end-game.

$$\#player\ pieces - \#opponent\ pieces$$

2.2 Multiple stacks

Promotes the formation of large token stacks in order to expand the possible number of moves. This was zero-sum calculated by finding the difference between the white and black team's evaluation scores.

This was evaluated by finding the sum of squares of the number of tokens in each team's stacks, divided by 12 (the maximum number of tokens for any one player).

$$MS(x) = \frac{\sum x_i^2}{12}$$

where x_i is the i$^{\text{th}}$ remaining piece in a players arsenal.

By squaring, larger stacks would be seen as more favourable, as a stack of 5 and 3 would be seen as more favourable compared to stacks of 4, 3 and 1:

$$MS([5,3]) = (25+9)12 = 2.833$$
$$MS([4,3,1]) = (16+9+1)12 = 2.166$$

2.3 Isolation

This is the sum of manhattan distances between all pieces between the two teams. We observed that a positive isolation score resulted in formation of large stacks, while multiplying by -1 would spread tokens out.

2.4 Split board

Weight stacks that are closer to their end of the board more heavily. This is done by measuring how far away a piece is from their end of the board, being 0 and 7 for white and black respectively. In the denominator this is expressed as the absolute difference between the row of a piece p_i and the players end of the board. These distances are summated and the inverse of the sum is taken.

$$Split(p) = \sum_{i=1}^{P} \frac{1}{|row(p_i) - start|}$$

where p_i is the i$^{\text{th}}$ piece in p, P is the number of pieces in p and $start$ is 0 if the player is white and 7 if the player is black.

Section 5.1.3 of Miki and Alex's report [2] describe a similar heuristic for checkers, where row number is taken into account.

2.5 Boom score

Evaluate a boom action by finding the difference between the number of opponents pieces destroyed minus collateral. Agents will avoid moves that cause more of its own pieces to explode.

2.6 Boom potential

Evaluate the biggest possible boom that can be actioned in the next move. Moves with large boom potential will be weighed to overshadow any other moves.

The purpose of this is to motivate tokens to spend their turn moving to a location that can result to a desirable boom on the following turn. This gives the agent more 'foresight' for potential booms.

2.7 Defensive

This is a meta-strategy, which involves the combination of baseline, isolation and boom potential heuristics.

This strategy promotes playing defensively by keeping as many pieces alive and far away from the opponent. The agent will boom only if advantageous, such as the case when pressure is applied.

This strategy weights the heuristics as follows:

$$defensive = 1.5 \times baseline + 0.2 \times isolation + 25 \times boom\ potential$$

2.8 Sniper

This is the main mid-game strategy to prioritise the formation of large stacks of tokens, keeping them on the players side of the board and only shooting to kill.

This is a combination of baseline, multiple stacks, split board and boom potential.

This strategy weights the heuristics as follows:

$$sniper = 1 \times baseline + 1 \times multiple\ stacks + 1 \times split\ board + 100 \times boom\ potential$$

2.9 Mortar

This is a more aggressive variation of the sniper strategy, where higher weighting is placed on perceived aggressive behaviour such as booming and less weighting on split board.

This strategy weights the heuristics as follows:

$$mortar = 1.75 \times baseline + 1 \times multiple\ stacks + 0.5 \times split\ board + 150 \times boom\ potential$$

2.10 Tactical Nuke

Inspired by the Tactical Nuke in MW2. The agent moves as close to the opponent as possible to trigger a boom. This behaviour encapsulates very destructive greediness, where the goal is to cause as much damage as possible.

This is a combination of boom potential and isolation, and is weighted as follows:

$$nuke = 150 \times boom\ potential - isolation$$

2.11 Additional heuristics

The following heuristics were developed but were not used in our agents decision making:

2.11.1 Blow the bridge

This is the difference between the number of bridges an opponent has and how many the player has. A bridge is a position on the board which joins two or more disjoint groups of pieces together such that they can be impacted by a boom action. This is illustrated in figure 1.

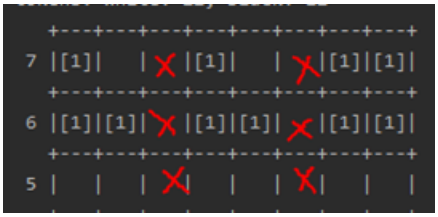

Figure 1: Spaces marked in red show bridges

2.11.2 Evaluation control

This is a summation of baseline and a random integer between -10 and 10. Effectively behaves like a random agent, this was used to evaluate our agents performance against random behaviour.

3 Optimisations

Throughout our development of the agent we found many interesting optimisations and research findings that enabled us to produce a more efficient agent. These include:

3.1 Transposition table

We constructed a transposition table to aid in pruning by storing states and their evaluated values for each strategy. This allowed the program to have constant access time to the values of previously visited board states, saving us the cost of calling negamax.

Our table stored evaluations for both white and black players in order to aid in negamax's zero sum calculations. Because we based our evaluation on the idea of a zero sum game, the heuristic value of a given hashed state is the same for both players, except that the minimiser takes the negative version of that value

3.2 Hashing

We used a hash key based on zobrist hashing [3] to store states in our transposition table.

We were able to store values for a majority of static board configuration evaluations. However, whenever a piece blows up, a unique calculation is required as the value of the move comes from evaluating the differences from the previous board state to the current board state and does not solely depend on the current board state. This was accounted for by making a special evaluation function for booms that does not record a hashed state.

3.3 Negamax

We took advantage of the zero-sum aspect of two player Expendi-bots to implement negamax, which allowed us to simplify our implementation of alpha-beta pruning and optimise our code.

As a result of our usage of Negamax, we approached evaluating the board state from a single point of view, creating zero-sum evaluation functions.

3.4 Move ordering

We ordered our actions in our search tree to prune leaf decisions that may not be as impactful.

In line with the sniper strategy, we programmed the player to build stacks of tokens, we then prioritised launching a single piece the maximum amount of space from the origin so that it could blow up opposing pieces from a distance without endangering friendly pieces.

We arranged the action order of each token stack to be:

1. Boom
2. Then, we would attempt to launch a single piece the maximum distance, decreasing the number of spaces in each iteration.
3. The process is then repeated with an increasing amount of tokens to launch on each iteration

3.5 Booking

We employed a book of opening moves to reduce search space in the initial stages of the game by moving pieces into the centre of the board to get early map control.

We found that this was more effective than simply taking random move actions.

3.6 Response to game stages

Our agent performs a high level evaluation of the game state in order to respond to changing situations. This includes tracking if it has a piece advantage over the opponent, the magnitude of that piece advantage and a timer in order to shift tactics accordingly.

3.7 Unimplemented optimisations

These are a few things we tried but did not feature in our final submission, as they did not improve the performance of the agent.

3.7.1 Minimax

We implemented minimax with alpha-beta pruning but found it was difficult to keep a record of evaluation states that can arise from a board configuration, as the heuristic value of a board is highly dependent on the colour of the player.

For this reason, we chose the zero sum approach with negamax, as it allowed to easily hash and store board states, as the evaluation values are negated for each player.

3.7.2 Asymmetric evaluation functions

Writing from the perspective of minimax, our evaluation functions were initially non-zero-sum which would be evaluated differently depending on which player was evaluating the state.

3.7.3 Manhattan heuristic

Our Manhattan heuristic from part A, which was the sum of all manhattan distances between pairs of assigned pieces, did not feature as a main heuristic as we found it was not an appropriate feature to evaluate a game state.

3.7.4 IDA*

Initially we used our implementation of IDA* to act as a greedy agent, along with just manhattan distance as our admissible heuristic. We encountered Key Errors with dictionaries during gameplay, and we eventually found it was much more effective to use the tactical nuke heuristic as our greedy evaluation function as it more properly embodied the idea of a destructive agent.

4 Performance

We experimented with our player against random players, greedy players and adversarial players using a variety of our heuristics. We tested our agent with 1,2 and 4-ply depth and found that 4-ply was quite time consuming even with alpha-beta pruning and move ordering. We opted to submit a 2-ply agent as it struck the balance between computation time and strategic complexity.

4.1 Random opponents

Against a random player, we found our agent was easily able to capitalise on the random players' actions (or lack thereof), as the random player would typically move around in its own base.

We also tested against a random agent with baseline as its evaluation function (effectively our version of a mixed random-greedy player) and found similar success albeit the random+greedy agent was more competitive as it would obviously hit back.

4.2 Greedy opponents

We constructed a purely greedy player from using just baseline, and found similar success to the random-greedy player, as our agent would capitalise on favourable booms more often than the baseline player.

4.3 Adversarial opponents

We used the agent itself as our adversarial test bed. Testing the agent against itself, we identified instances where a stalemate would occur, with both agent colours staying away from each other until a tie was declared. We rectified this by implementing more aggressive strategies in these circumstances to combat 'scared' agents.

We found many instances where these agents would perform tactical moves, such as moving large stacks out of blast radiuses and shooting a single piece across the board into enemy territory where an opponent's boom would cause more harm than good. This was the desired result.

5 Future considerations

Ideally for the future we would like to implement a few extra things to expand on our current agent:

- The killer heuristic [5]: with our implementation of a transposition table, the killer heuristic would be good to implement to further optimise our tree search.
- Temporal-difference learning in order to learn which specific moves result in more wins.
- Endgame databases in order to take advantage of the full 100MB of space allocation.
- Monte-Carlo methods such as determining optimal opening moves and at least very good move ordering in order to allow for a 4-ply depth.

References

[1] Zuckerman, Inon Felner, Ariel. (2011). The MP-MIX Algorithm: Dynamic Search Strategy Selection in Multiplayer Adversarial Search. IEEE Trans. Comput. Intellig. and AI in Games. 3. 316-331. 10.1109/TCIAIG.2011.2166266.

[2] Mika, Alex, `https://cs.huji.ac.il/~ai/projects/old/English-Draughts.pdf`

[3] Chessprogramming.org, `https://www.chessprogramming.org/Zobrist_Hashing`

[5] https://www.chessprogramming.org/Killer$_{H}euristic$