# Project Part 2 Report, COMP30024

Patrick Butcher: 996189
Huan Zhang: 1173919
Group Name: TBD

## Introduction

The cachex game player algorithm was implemented with a composite of a Greedy algorithm and a Minimax with Alpha-Beta Pruning algorithm, both using an A* search algorithm in the evaluation score.  Originally it was hypothesised that a stand alone Minimax with Alpha-Beta Pruning method with A* would have given us an optimal solution, however after evaluating the performance within the time and space constraints, modifications were made to the algorithm to perform optimally in certain situations.

## Evaluation Score

The evaluation score for each board state or leaf node in the alpha-beta pruning tree was made up of iterations of the A* search algorithm, with a true distance as the heuristic. Rather than returning the shortest path, the shortest distance from side to side was calculated for both players in a board state. This was calculated using iterations of the algorithm. For a red player, the shortest distance for every pair of cells on the opposite sides of their colour. For example on a 5x5 board as shown in figure 1, for a red player, the shortest distance from ((0,0) and (4,0)) was calculated, as well as ((0,0) and (4,1)) and ((0,1) and (4,3)). So for every node, the A* algorithm was implemented 2*n^2 times with the lowest score being recorded for both colours, and the difference between those values being used as the evaluation score for that node. A maximal shortest distance for an opponent and a minimal distance for a player's own shortest path being desired.

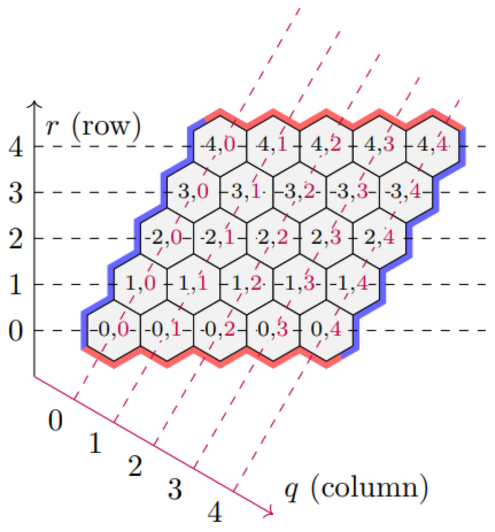

*Figure 1: example of a board with size n=5*

If an opponent's cell was on one either the start or goal cell, the algorithm would not be calculated for that combination. Cells occupied by an opponent on the board had to be avoided in the shortest path. If a player's own cells were also used in the shortest path if they were found already existing on the board, and did not contribute to the distance of the path. For example if red was to get from (0,2) to (4,2) and (1,2),(2,2) and (3,2) were already occupied by red, the shortest distance would be two rather than five, and when every possible path for winning is blocked, the distance will be set to infinity.

Strategically this function worked well and gave a complete and accurate evaluation of the board. Since the goal of the game was form a path from end to end, A* was a function that

represented the goal well. It may have been quite time expensive, given the algorithm was implemented n^2 times every board state. Another less expensive algorithm that was considered for an evaluation function was the difference between the number of a player's cells that were occupied and opponents' cells that were occupied in a given state. This would have been much more time efficient, however may not have been as accurate of an evaluation score, given the goal of the game is to form a path from one side to the other rather than occupy the most cells. Because this type of algorithm would have been more time efficient, it had the potential for a greater depth value, and hence could look more moves ahead. However from observation it was found that the A* algorithm with a lower depth score performed better.
Since minimax is an algorithm that heavily relies on the quality of the evaluation, if implemented with a stronger and more accurate evaluation for the representation (approximate value) of the state, the performance of minimax will be significantly improved. Thus, one potential improvement for our implementation is to combine the A* distance score, the number of occupied, number of opponent's cells, and including other features of the board to help better describe the game state.

## Algorithm

The alpha-beta pruning algorithm then used this score to evaluate states of the game. Every possible move (every empty cell) was considered given it was not pruned. Hence for the theoretical tree there would be a branching factor of n^2 for the number of empty cells. This value reduced as the board grew. Alpha-beta pruning was used to reduce the time complexity rather than a regular adversarial search, whilst still performing at a similar level, from our testing, the number of nodes and iterations with Alpha-beta pruning were significantly reduced compared to the one without pruning.

When testing the algorithm composed purely with alpha-beta pruning with a large cut-off depth, it was discovered that as the size of the board increased, the time taken would increase at a rate greater than n^2 seconds. Hence an algorithm was implemented to include a dynamic depth. The depth of the alpha-beta pruning method was based on both n (the size of the board) as well as nturns (the number of completed turns in the game so far). The reason for including a dynamic depth value was to have the largest depth value without exceeding the time limit of n^2 seconds per player per game. It was found that for smaller boards, the time limit would not be exceeded for large depths. For example when n=3, a depth of nine kept the algorithm time within the threshold, a depth that would almost represent a full minimax depth given that no captured or steals occured in the game. However, for a board with n=12, the algorithm would struggle to finish a game within n^2 seconds with a depth of only one. Hence for larger boards, it was decided that the depth would generally be kept at a smaller value (given nturns was taken into account as well).

It was also decided that during the early stages of the game, when nturns was low, that the depth would be smaller, and increase as the game progressed. This was done primarily to reduce the time taken in the early stages of the game, as it was decided that a deep minimax algorithm was not required in these early stages as when the game got more complex. Hypothetically, when the board is empty, there are a n^2 number of moves that will have the same A* value. Similarly when the board as many empty cells, there will be many moves each comprising of a similar A* score. Assuming all cells are equally advantageous to move on, there is less value in analysing minimax while the board is empty, and instead it is more valuable when there is a complex board setup with multiple opponents pieces being considered as well. It is clear that not all cells *are* equally advantageous, as some have more options to reach a winning state than others, however the point stands to some degree. Ideally an alpha-beta pruning algorithm would be implemented with full depth every turn, however since time was an issue, it was decided to save a greater depth for more complex board scenarios, with a greater number of occupied cells and a larger number of turns that had occurred.

Specifically, depth was a constant value of n-8 if n was less than 6. If n was greater or equal to 6, depth began as a value of zero, and increased by one every thirty percent of the maximum number of nturns. These values were chosen based on observation to try and win our game within the time limit n^2 for all board sizes.

When depth was equal to zero, the greedy algorithm was implemented. The greedy algorithm used the same A* algorithm used as the evaluation function for its heuristic function, however only took a player's own shortest distance into account and not its opponents'. This helped reduce the time complexity of our algorithm as the A* algorithm was only implemented n^2 times rather than 2*n^2 when depth == 1. It also gave some element of strategic capability rather than moving to random cells. The idea for implementing this greedy algorithm for the first 30% of games was to speedily start our path from one side before settling into a more complex alpha-beta pruning algorithm as the board populated itself.

At the beginning of the game the algorithm always chose to STEAL if given the opportunity. By observation it was hypothesised that this gave a strong advantage as having an extra tile on the board usually resulted in a win.

## Performance Evaluation

How have you judged your program's performance?

Our algorithm was tested against multiple other algorithms. It was tested against a random algorithm as well as a greedy algorithm as well as itself. This firstly helped with debugging as it produced a visualisation of the game which could be observed and analysed. It also helped with strategic evaluation. If our algorithm was not performing well against the less complex algorithms it was clear where we could improve. We will also look into special cases when the algorithm fails to help us find places to improve it, as well as actually playing ourselves against our own algorithm to see how it behaves, and we will normally choose some targeted counter moves based on the algorithm's choice. After iterations of performing each algorithm against one another and making modifications to influence optimisation, the composite algorithm with both greedy and alpha-beta pruning was chosen. This was chosen mainly based off observation. One way to improve our performance evaluation and hence the optimality of our algorithm would have been to record results and analyse them statistically. This would reduce the human error of our current method. Also, we may want to implement some other more advanced algorithms to compare, such as self-learning/machine learning algorithms, and other types of search trees (e.g. Monte-Carlo Tree search)

It was also observed that our algorithm was well below the space constraints, yet pushed the boundaries of the time constraint for larger boards. To improve our algorithm, a greater amount of space could have been used to store evaluation scores for given board states. This would have used quite a lot of space but would have reduced the time. Instead of having to calculate the evaluation score of 2*n^2 times, the states of the already explored boards could have been stored in a dictionary or hash table and have taken a much shorter time. This would have provided the opportunity to increase the depth of the alpha-beta pruning algorithm and look deeper into the game, creating a more optimal algorithm.

## Supporting work/Reference

https://github.com/duilio/c4/blob/master/c4/cache.py
Hex - Creating Intelligent Adversaries (Part 1: Minimax α-β Pruning) | by Greg Surma | Medium
221-hex-poster-final (stanford.edu)