Younjee Kang
1005978621

CSC384 - A2: Describing the Game Tree Strategies

Depth Limit:

We have implemented the Minimax with alpha-beta pruning which also has a depth limit. We have constructed the max depth to be at 8, assuming our heuristic estimates about the non-terminal positions are good enough. This way, we could have a more functional and practical program.

Caching States:

Using dictionary structure in python, we implement state caching that maps states(that also contains which player of that state was) to the depth, alpha, beta, move, and value. Then, if the program encounters the state we have seen before, it checks if the current (alpha, beta) is within a range we have already seen before. That is, for a max node, if the current beta is smaller than the cached beta, we can use the cached value as it would be pruned anyway (as we pruned when value >= beta), and for a min node, the current alpha should be greater than the cached alpha to use the cached value as it would be pruned as well(similarly as we pruned when value <= alpha). Plus, it compares the current depth and the cached depth. As the cached values evaluated at a deeper level in the tree than the current depth would be less accurate, we cannot use the cached value even if it satisfies the alpha-beta condition.

Your Own Heuristic

For a complex heuristic function, three basic ideas are taken into account:

1. How many pieces are **stable** in terms of the opponent's capture and how many are not?
   a. How many are shielded at the edge of the board
   b. How many are shielded(protected) by the other pieces(eg. pyramid formation)
   c. How many are extremely vulnerable, i.e., can be captured by just a single jump
2. How many pieces are **preventing** the opponent's chance to become stronger?
   a. How many pieces are blocking the first row, i.e., preventing an opponent's piece from becoming a king
3. How many are more **promising in terms of attack**
   a. How many kings

Younjee Kang
1005978621

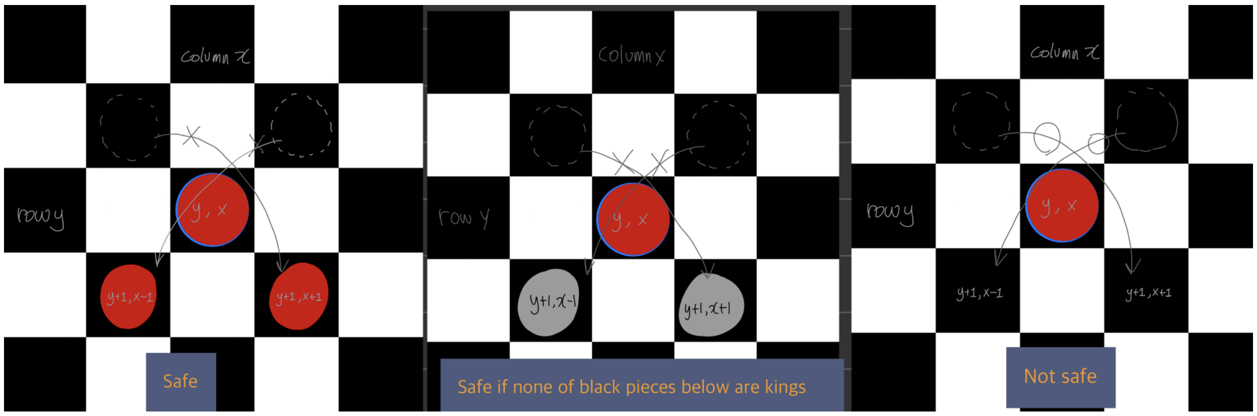
b. How many are at the centre of the board

Note we specifically looked at the red pieces, as we can consider Max's payoff and the Min player can find the move that minimizes it in the Minimax algorithm.

Detailed Procedure:

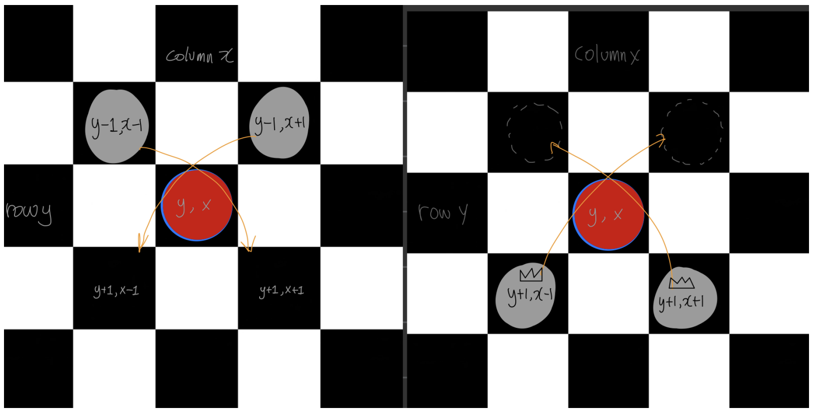
1-a) If the red piece is on the edge of the board, then it is safe from capture. Therefore, it is counted as a safe piece.

1-b) Consider the diagram below:



Those are examples of when a red piece is safe. Let y, x be the coordinates of a red piece that we are looking at. Then, the red piece is safe from captures as long as the two pieces below are red, or black but pawns. If at least one of them is empty or a black king piece, then it is unstable. Count the safe piece.

1-c) Consider the diagram below:



These are example cases of when a red piece can be captured by one jump of an opponent. No matter the piece upper the red piece is a king or not, the red piece can be easily captured if its adjacent is empty. If a black piece adjacent to the red piece is a king, then as it can hop upwards, the red piece is vulnerable also when they are below them.

Subtract the number of those risky pieces from the number of stable pieces. As this is a good estimate, this value may be worth more than the other values to speed up the runtime.

2-a) Black pieces become kings when it reaches the last row. So the red pieces on the last row are not only on the edge so it is safe from capture, but also essentially preventing the opponents to have greater power in the game. Count the loyal pieces. Each is worth 1, but note that they are one of the safe pieces, so counted twice.

3-a) Count the red king pieces as each can create greater gains in one move. Each is worth 8 points, whereas each pawn piece is worth 4.

3-b) As the game proceeds, taking control of the centre is a beneficial strategy. It sometimes forces an opponent piece to move in a certain way, or it is simply good to capture the opponent's pieces. If a piece is on either of the two middle rows, it gains 1.5 points, and 0.5 more if it is even located in the middle 4 columns.

Node Ordering Heuristic

For both player MAX and MIN, we know it is better to explore the best possible child node first: the node has the most MAX's payoff for MAX and the least MAX's payoff for MIN. Or in other words, a node that is promising for that each player. Here, we are given a set of possible moves, and we need to decide which move is likely to have a better state to themselves, and this should be computed ideally quickly. Therefore, we implemented a node ordering using these ideas:

1. A move that captures a lot of pieces is likely to bring victory.
2. A move that brings a piece to the edge(i.e., the first and the last rows) benefits the player. (as it can keep opponents to be a king, or it can be a king, or in all cases, it is still protected from the captures.)
3. Taking control of the centre of the board is beneficial.

Note that these variables are known and computed quickly as an object Move contains as its attribute, with no need to create an after-move-state.

Detailed Procedure

- The number of captured pieces of the block is doubled and added to the points.
- By trial and error, we have found that exploring the move that has brought a piece around the centre speeds up the algorithm. Each of the centre-located pieces is worth 21 points.
- If a move brings a piece to the closest row from the start, it gets 2 points.

Younjee Kang
1005978621

- If a move brings a piece to the furthest row, where it lets a piece be a king, the move gets 3 points.
- If a move brings a piece to the furthest 3 rows, it gains 2 points.

Additional Improvements

When generating possible moves, we made the function build a heap that contains possible moves(either jump or non-jump moves) by pushing whenever a new potential move is discovered. This is to ideally speed up when picking the node to explore first in the AlphaBeta Minimax algorithm, rather than using a list and finding a max or min each time.

Plus, when generating possible moves, the program looks at the possible jump moves first for all nodes. If there is none, it then looks at the possible non-jump moves. This is to eliminate time finding all the possible non-jump moves that wouldn't be used, following the mandatory jump rules.