**Approach**

The agent was segmented into 3 components, namely the *Evaluation Function, Search (MiniMax) Algorithm,* and *Optimization (Alpha-Beta Pruning, Transposition Table with Zobrist Hashing)*. These three components were developed iteratively.

**Data Structures**

The agent perceives the state of the board as a 6x6 2-Dimensional list of characters, where ‘B’ represents a Black pawn, ‘W’ represents a White pawn, and ‘\_’ represents an empty tile.

The Initial State is represented where the first 2 rows (0, 1) are filled with ‘B’, middle 2 rows (2, 3) are filled with ‘\_’ and last 2 rows (4, 5) are filled with ‘W’.

The Goal States are represented where the last row (5) contains one ‘B’ anywhere in its nested list.

The agent also converts the 2-Dimensional list into a into a **Zobrist Hash** integer value, which is stored in a **Transposition Table** to facilitate memoization. This will be elaborated on later.

**Evaluation Function**

The Evaluation Function is calculated by iterating through the 2-Dimensional list state representation, to find the position of all the pawns. The score of one side of the board (Black or White) is the accumulative sum over all pawns, of the distance travelled toward the last row. Through experimentation, I found that adding weights yielded more accurate evaluations, the threat of a pawn progressing down the board increases exponentially. The weights are as follows:

Row 4 -> Distance \* 3

Row 5 -> Distance \* INFINITY

The scores of each side of the board (Black or White) are calculated and the final evaluation score is the difference between Black’s and White’s score (i.e., score\_black – score\_white), since the agent perceives the state from Black’s perspective. Breakthrough being a zero-sum game, this Evaluation Function calculates the scale in which the game’s state is tilted toward one side’s favor.

In addition to the Evaluation Function, each state is tested if there exists a pawn on the second last rank on the turn. This is an autowin condition and a the first legal move forward will be immediately returned.

**Minimax Search Algorithm**

The agent utilizes an Alpha Beta Pruning Minimax Algorithm searching 5 ply each turn. (The agent is able to search 7 ply without exceeding the time constraint on my own machine but is limited to a depth of 5 on Coursemology). The search algorithm is also complemented with a **Transposition Table** using **Zobrist Hashing** to memoize already evaluated states, while minimizing memory usage.

The Minimax Algorithm recursively calls its minimization or maximization function based on the turn, up to a depth of 5. In this agent’s case, the Black player aims to maximize the evaluation score while the White player aims to minimize it. Within each iteration of the minimization or maximization function calls, moves are generated from the state by evaluating all possible legal moves for that given state and are evaluated by a recursive call of the minimization or maximization function. Once the depth of 5 is reached, or the game reaches a terminal state (game over), the state is evaluated by the Evaluation Function. The recursive calls bubble upward, and the best move and evaluation is taken from the maximal or minimal evaluated subtree, depending on whether it is Black or White’s turn at that depth.

**Alpha-Beta Pruning**

Upon initializing the search, two main values, alpha and beta (representing Black and White) are initialized to negative infinity and infinity respectively, (i.e. both players start with their worst possible score). Whenever the maximum score that the player White (beta) is assured of becomes less than the minimum score of the player Black (alpha), i.e., beta <= alpha, the Search will not need to consider the subsequent subtree generated by the state, since it will never be reached in actual play.

**Zobrist Hashing**

Each 2-Dimensional list representation of the board undergoes Zobrist Hashing, to represent the board as a 16-bit binary integer, returned as a decimal integer. Upon initialization of the agent, a Zobrist Hash Table is generated where each square is on the board has a random bitstring assigned to it, based on whether it has a Black piece or a White piece.

Each state is then iterated through, and the Zobrist Hash Value of the state is XOR-ed with the bitstring assigned to the colored pawn on each square it appears on. The final hash value is then returned as the integer yielded after the cumulative XOR operations. The total number of possible values is 2^16, so the probability of two states being hashed to the same value is negligible.

**Transposition Table**

A Transposition Table with key: Zobrist Hash of state, value: best move and evaluation is implemented. During the Minimax search, if a new state is evaluated using the evaluation function, its Zobrist Hash is calculated and stored in the Transposition Table. Otherwise, if the state’s Zobrist Hash exists in the Transposition table, its best move and evaluation are retrieved directly from the table, saving search time on re-expanding that state.

In addition to this, by using a Zobrist Hash to represent the state of the board, we avoid storing the states as large 2-Dimensional lists, which reduces space usage considerably.

**Conclusion**

The agent is utilizes adversarial search to identify the optimal moves per turn considering both player’s optimal play up to a certain depth. With a straight forward Evaluation Function and heavy focus on time and space optimization, the agent is built to predict play into the future as far as possible, in order to yield the best move for the turn.