COMP30024 - Project Part B **Date:** 13/05/2024
Report

Team members:
Emilio Ortiz
Kate Griffiths

Team name: Purple

Agent Strategy
The strategy used for creating the agent was Minimax with alpha-beta pruning. Minimax works well as a strategy for two-player zero-sum games (which the game of Tetress is), since it chooses the move that will benefit it the most by looking at all possible game states a number of moves ahead. Minimax relies on being able to see as many moves ahead as possible to be able to make good predictions. This requires a lot of memory, resources and takes a lot of time to search which won't always be available. That is why alpha-beta pruning was also included, so that branches that show a better move for the opponent than existing branches are pruned, saving us time by not exploring the rest of that branch.

The time complexity of minimax is O(b^d) and the space complexity is O(bd), where b is the branching factor/number of legal moves at each point and d is the maximum depth of the tree.

When using minimax with alpha-beta pruning the worst case time complexity occurs when there are no nodes that get pruned, which gives a time complexity of O(b^d). In the best case scenario, where the best moves are searched first the time complexity becomes O(b^(d/2)).

To maximise the effectiveness of alpha-beta pruning, the program sorts the playable moves at the current board state, in each step of the algorithm, according to how desirable they are (i.e. how likely they are to lead down the search tree to the highest utility values), before expanding successor board states that are reached by performing one of those moves. This is done under the premise that the number of pruned branches is increased if the most promising moves are explored first. The metric used to measure the desirability of moves is explained in the Optimisation section.

Choosing Evaluation Method for Minimax
Minimax also relies on the evaluation method used to determine which moves are considered to be good. In part A of the project the method used valued moves that caused line removal. This could have been adapted for part B to favour lines that removed more of the opponents colour than its own from the game as better, and any move that got closer to doing that was also valued higher. This evaluation method was disregarded as it did not consider some of the other game mechanics that are involved when playing with another player. Instead the method chosen opted to focus on reducing the number of playable moves the opponent was able to make, while also maximising our own possible moves. One of the downsides of using this as an evaluation method is that it will result in an increase in the branching factor of our own moves, which could lead to increased search times.

Evaluation Method
The evaluation method used by the agent to determine the value of a given board state follows the following pseudo code.

```
EVAL_METHOD(board, player) -> float:
    IF (max_turns_reached):
        IF (player_tokens > opponent_tokens):
            // inf represents a winning state
            return float('inf')
        ELIF (player_tokens < opponent_tokens):
            // -inf represents a losing state
            return float('-inf')
        ELSE: //equal tokens
            return 0
    ELIF (playable_moves(player) == 0) or (playable_moves(opponent) == 0):
        IF (playable_moves(player) == 0):
            // ran out of moves, we lose
            return float('-inf')
        ELSE: //opponent ran out of moves, we win
            return float('inf')
    ELSE:
        // higher number is more favourable
        return (num_playable_moves(player) - num_playable_moves(opponent))
```

The evaluation method computes the difference in the number of possible moves each player can make. A positive number indicates that the agent has more playable moves than the opponent and a negative number indicates that the opponent has more playable moves, and therefore, the upperhand; in non-terminal board states at which the depth limit of the minimax function has been reached, this difference is returned. The function can also return infinity or -infinity to represent the utility of a terminal game state that has either resulted in a win or a loss.

This was the chosen evaluation method because we deemed board states where the player has more playable moves than their opponent advantageous, and more likely to conduct to the victorious terminal state where the player's opponent is unable to make more moves.

Optimisation
While testing the agent we found that it was taking too long to search for a good move. Over an entire game the agent was taking more than the specified maximum time to make moves. In an attempt to reduce the search time, the list of possible moves was sorted to benefit from alpha-beta pruning's reduced time complexity. In the list of the playable_tetrominos preference was given to moves that had a lower number of adjacent tokens belonging to the player; and therefore moves which more greatly allowed the player to be more spread out in the board, occupying more space, and to possibly have a higher number of possible moves, were picked first.

Additionally, at every Minimax step, every playable tetromino that has the same level of preference as another is discarded (so no two tetrominos have the same preference level in the move list), this in order to lower the branching factor. This modification allowed us to increase the baseline for the number of moves the Minimax algorithm can look ahead.

To adjust for long search times while using Minimax, a function to dynamically calculate and change the depth of the Minimax algorithm was added. This function increased the depth of the search in states that had a smaller number of moves to take advantage of the reduced time complexity that lower possible moves allows for. It conversely reduced the depth of the search when a board state had a higher number of possible moves so that search times would not be excessively long as the higher branching factor would become too high to perform a deep Minimax search. Additionally, it also considers the agent’s remaining time, as indicated by the referee (in the cases where there is a time limit) to choose the Minimax depth limit, allowing a deeper search when more time is had.