**Mini Checkers – Guide and Info**

**By Francis Maurice B. Miranda & Dominic Riley C. Alegre**

**How to start the game:**

1. Simply open the game.py file and run it on your python program (Python 3)
2. When opening the file, a prompt will open up asking if you want to go first or not. Simply type y or n to choose.
3. After choosing whether to go first or not, the game will prompt what difficulty do you want. Simply choose from the options given.
4. After that, the game window will open in a new window.

**How to play the game:**

The game uses a point and click mechanic. Simply click the desired piece and click onto the desired spot that you want the piece to move to. Note that the piece will not move if the move is invalid (refer to rules).

When capturing a piece, simply click the empty space diagonally adjacent to the desired piece that you want to capture. The game will automatically capture the piece and put you in the adjacent empty space.

**Rules of the game:**

This game draws inspiration from the classic game of checkers but with a twist. Instead of the classic 8 by 8 board and eight starting pieces, this game only has 6 by 6 board with six starting pieces. The game follows the same standard rule of checkers. Here are the rules of the game:

1. Each player starts with 6 pieces each
2. Player can choose whether to go first or second
3. Player can move diagonally to an adjacent square
4. Player can capture a piece by jumping over the said piece and landing on an empty square.
5. Player can only capture pieces in a forward direction and no backwards movement can be done
6. Every opportunity to capture a piece must be done
7. Player wins if he/she captures all of the opponent’s pieces
8. If no players have no valid moves left, then the player with the most amount of pieces win
9. If any of the two players have the same number of pieces, it results in a draw

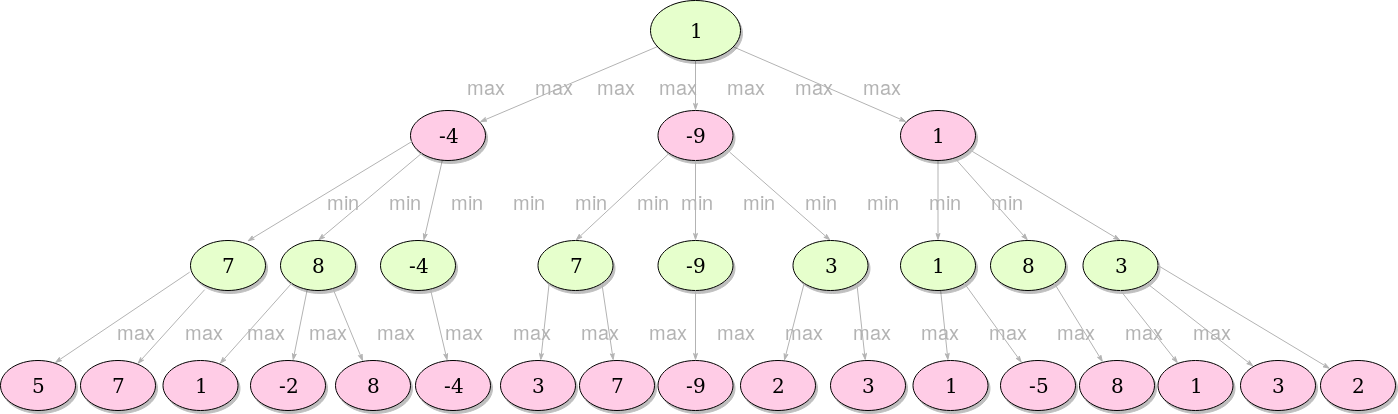
**About the game and the concepts behind it:**

The game is coded using Python 3 and utilizes the **Tkinter** Library as its graphical user interface. The game implements an AI which uses alpha beta pruning. Alpha Beta pruning is an improved version of the Minimax algorithm, a simple algorithm used in decision-making in game theory and artificial intelligence.

**Minimax Algorithm**

To further explain this, the minimax algorithm is widely used in two player based games. Examples of these are Tic-Tac-Toe or Chess. The two players will be referred as the maximizer and the minimizer. The maximizer tries to get the highest possible score while the minimizer does the opposite. Every board state has a value associated with it. In this case, it is checkers. In a given state if the maximizer has upper hand then, the score of the board will tend to be some positive value. If the minimizer has the upper hand in that board state then it will tend to be some negative value. The values of the board are calculated by some heuristics which are unique for every type of game.

The Minimax algorithm can be best represented using a game tree:

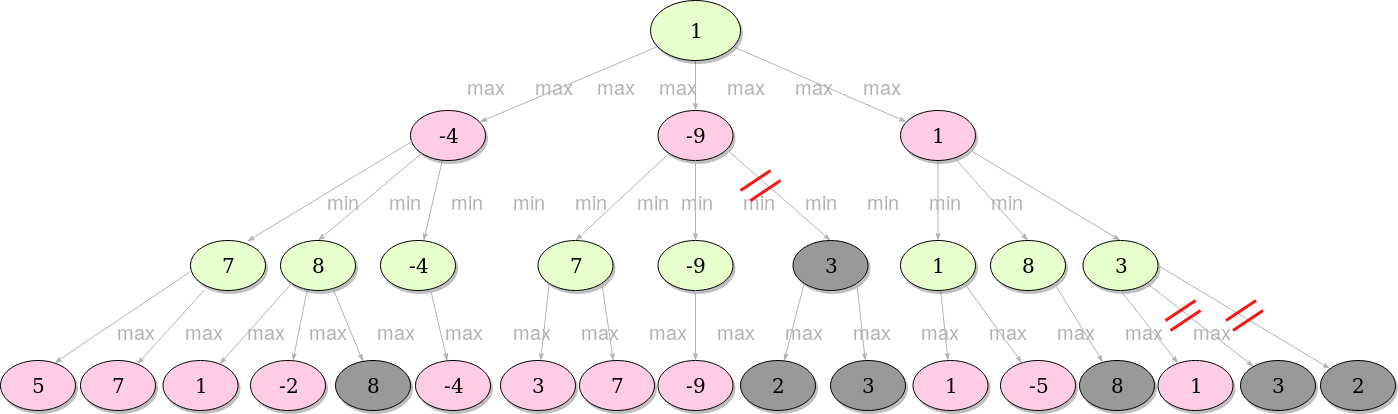


In this example we've assumed that the green player seeks positive values, while the pink player seeks negative. The algorithm primarily evaluates only nodes at the given depth, and the rest of the procedure is [recursive](https://en.wikipedia.org/wiki/Recursion_(computer_science)). The values of the rest of the nodes are the maximum values of their respective children if it's green player's turn, or, analogously, the minimum value if it's pink player's turn. The value in each node represents the next best move considering given information.

**Alpha Beta Pruning**

As said before, alpha beta pruning is the more efficient and faster method of the minimax algorithm. The difference it sets from the minimax algorithm is that it stops evaluating a move when it makes sure that it's worse than previously examined move. Such moves need not to be evaluated further. When added to a simple minimax algorithm, it gives the same output, but cuts off certain branches that can't possibly affect the final decision - dramatically improving the performance.

Here is the same tree but with alpha beta pruning:



When the search comes to the first grey area (8), it'll check the current best (with minimum value) already explored option along the path for the minimizer, which is at that moment 7. Since 8 is bigger than 7, we are allowed to cut off all the further children of the node we're at (in this case there aren't any), since if we play that move, the opponent will play a move with value 8, which is worse for us than any possible move the opponent could have made if we had made another move.

Overall, Alpha-beta pruning makes a major difference in evaluating large and complex game trees. Even though tic-tac-toe is a simple game itself, we can still notice how without alpha-beta heuristics the algorithm takes significantly more time to recommend the move in first turn.