# **Table Of Contents:**

[AI Method:](#h.una4ee56q422)

[AI Implementation:](#h.5umavk3e8g54)

[Changes to generic Implementation:](#h.20vzezv61a87)

[Using the AI:](#h.q9lfr0orw4lg)

# 

## AI Method:

The method for AI that I am using is Monte Carlo Tree Search(MCTS). Though Modified slightly for performance and an attempt at better predictions.

## AI Implementation:

My base implementation for the AI is as follows;

-Firstly I call my GetValidMovesForBlack function which fills out a list of all available moves for the AI

-Then save that list for reference later as potentialMoves.

-then for each available move

-for each playout(number given when creating AI)

-I Clone all the pieces

-then as long as neither the simulated list of red or black pieces is empty

-perform the first AI move

-then simulate a player turn(random, random capture if capture available)

continue to simulate player and AI turns until end of game reached or no more moves are possible for the AI

-Then delete the clone and go for the next potential Move

## Changes to generic Implementation:

Ok so firstly I added certain values to the AI’s creation, due to the time it was taking during each turn i put a depth limit on the game simulation, (chosen when AI is created) so if it hasn’t reached a end game state after a certain number of moves into a game, it will just quit out and take its score the way it is.

To reinforce that I added a heuristic value to each simulated turn, giving a +1 to the potentialMove score for each player piece capture, and a +15 for any player kings captured, but a -2 for losing a piece, and a -20 for losing a king, in an attempt to sway the AI into saving its pieces when it could.

On top of that if an end game state is reach the score is increased/decreased by 50 accordingly

## Using the AI:

to Create an Ai create a new ai with the parameters of the board you are playing on, the number of playouts you want to do for each availableMove, and the depth you want to search each playout



Then when appropriate(after the players turn for instance)call CalculatePotentialMoves();

This will do as described above, using Search(), GetValidMovesForBlack(), SimulateRedTurn(), MakeMove(), as well as the directional check functions to determine multi jump and mandatory moves for the AI.