Technical Design Document: Complex Game Systems Assignment Game: Draughts (aka Checkers)

Variation Chosen: **Single Player vs AI Opponent**

Game Rules: **English Standard**

In my version the player to move first is selectable. Simply change the "Start Player" number and then click "Reset Game". The board will be reset and the "Current Player" colour will show the colour of the player whos turn it is.

Source: http://en.wikipedia.org/wiki/Draughts

- A move consists of moving a piece diagonally to an adjacent unoccupied square. Only the dark squares of the checkered board are used. A piece may move only diagonally into an unoccupied square.
- If the adjacent square contains an opponent's piece, and the square immediately beyond it is vacant, the piece may be captured (and removed from the game) by jumping over it. Capturing is mandatory.
- The player without pieces remaining, or who cannot move due to being blocked, loses the game. For my game the first player who has no available moves in their current turn is the one that loses the game, even if both players then have no available moves.
- Once a game has been gridlocked, where only back and forth moves between same locations on the board avoid jumps, the player with the majority of free space wins the game.

Men:

- Uncrowned pieces (men) move one step diagonally forward, and capture an opponent's piece by moving two consecutive steps in the same line, jumping over the piece on the first step. Multiple opposing pieces may be captured in a single turn provided this is done by successive jumps made by a single piece; the jumps do not need to be in the same line but may "zigzag" (change diagonal direction).
- Men can capture only forward

Kings:

- When a man reaches the crownhead or kings row (the farthest row forward), it becomes a king, and is normally marked by placing an additional piece on top of the first man, and acquires additional powers including the ability to move backwards (and capture backwards). As with non-king men, a king may make successive jumps in a single turn provided that each jump captures an opponent man or king.

Variant Rules:

- In many games at the end one adversary has three kings while the other one has just one king. In such a case the first adversary must win in thirteen moves or the game is declared a draw.
- For my version, I use something similar - a user set number of non-capture moves (defaulting to 100). If no piece of any player has been captured in this number of turns, the game is declared a draw. I have given it a range of between 10 and 100.

The program will check if there are any capture moves available to the current player and only allow them to select between those available. Otherwise, any piece with an available move will be selectable.

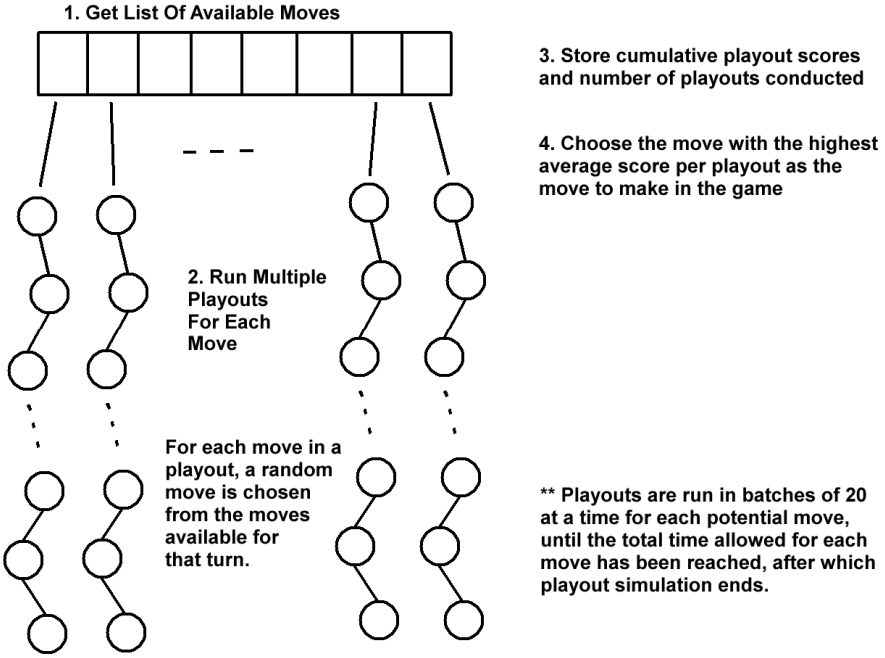
AI Technique: **Monte Carlo tree search**

More specifically, the AI technique used here is Pure Monte Carlo Game Search, though playouts are only played out up to a maximum pre-set number of moves to look ahead.

Firstly the AI will obtain the list of available moves it has for the current turn. Getting this list will return moves that are either only capture (or jump) moves, if there are any capture moves it can make, or non-capture moves if no jump moves are available.

If there is only a single move available, as in a single forced jump, or only a single normal move available, that move is immediately taken with no move simulation and scoring done, as there is no other option, so any processing time would be wasted. If there is more than one move available it will select its move using the Monte Carlo method, using the list of moves available for the current turn.

For the Pure Monte Carlo method the AI goes through all available moves from the list of moves for the current turn, and for each of them it plays through the game to a set depth (which is a user modifiable setting, defaulting to 50 moves), or until the game is won, whichever occurs first. Each play through will generate a score for that particular move. For each move available both the total cumulative score from playouts simulated for that move, and the total number of playouts conducted for that move, are recorded.



This would have allowed for a different number of playouts to have occurred for different moves, had multithreaded playout simulation been achieved properly. As it stands the final program is single threaded, so the number of playouts is always the same for each move. How many playouts are conducted is based on the number of moves being simulated for each playthrough, and the total decision time allowed for the AI to make a move decision.

The playouts are simulated and scored by a recursive function that takes in the number of moves to search to, and the GameState for that particular move. At each move during the playout simulation the AI uses a static board evaluation function that returns a score for the current board state, based on piece positions and types, or if the state is a win state for one player, returns a "high" score for that state ending the recursive move simulation for that playout.

This score feeds back up the recursive call stack, into the scores from previous moves, and ultimately back out to the score for the current move playout being simulated.

Playouts are done in batches of 20 playouts for each current move being tested, until the total time allowed for the current move has been reached. The time per move is also user adjustable.