Matthew Laikhram

Mini Camelot

Installation

Non-Windows machines

1. Unzip the project
2. Navigate to the src directory
3. Do “javac minicamelot/MiniCamelot.java”
4. Do “java minicamelot.MiniCamelot”

Any machine

1. Unzip the project
2. Install NetBeans
3. Open project in NetBeans
4. Press F6 (Run Project)

High Level Description

Once the program is started, the player will be presented with the new game panel, where they will be able to select a difficulty and the starting player from a set of radio buttons. Pressing start will cause the new game panel to be replaced with the board GUI. This displays the board as an array of icons.

When the player clicks on one of their pieces, the program calculates all of the possible moves of that piece and highlights the appropriate tiles in green. Once a green tile is selected, the program determines whether or not a chain move can be made. If it can, then the possible chain moves are calculated and highlighted in green. If the chain move is optional, then the current selected piece is also highlighted, indicating that you can click on the piece again to cancel the chain move. This is repeated until there are no more chain moves, at which point a timer is triggered to let the AI know that it is time to make a move.

The timer is set to 1 millisecond just to visually stagger the player’s move with the AI’s move. Once that timer expires, the AI creates a new 10 second timer and runs the alpha-beta search algorithm in a new thread. An iterative deepening approach is taken to allow the AI to have a valid move, even if it runs out of time while the algorithm is being run. At each depth limit, the AI stores all relevant statistics and the best move before incrementing the depth limit.

Once time runs out, the AI interrupts the alpha-beta search algorithm and makes its move on the board GUI. It makes each move one at a time if there is a chain of moves to be made. A 1 millisecond timer is once again used to visually stagger any chain moves the AI has made. This makes it easier for the player to see what set of moves the AI made to get it to the final board configuration. In addition, the AI will also display the relevant statistics as well as a phrase representing the AI's confidence in its chosen move.

This process is repeated until an endgame state is reached, where the program will disable user input to the board GUI and display who won, or if it was a draw. At this point, or any point in the game, you can click on the end game button to display the new game panel once again. Pressing the end game button while on the new game panel will close the program.

Special Rules

The following rules were added to disambiguate some rulings when chaining is implemented:

Capture Moves are mandatory whenever they are *immediately* available.

You can chain together multiple Canter and Capture Moves. Chaining Canter Moves together will prevent you from landing on any tile you've landed on in the current chain. However, if a Capture Move is made in the chain, then any tiles landed on prior to the Capture Move will be allowed in the chain once again.

While Capture Moves that are beyond a chain are not required, if you chain moves up to the point of having a Capture Move immediately available, then you must take the Capture Move.

Difficulties

Easy: Limits the depth of the search tree to 1

Medium: Limits the depth of the search tree to 2

Expert: No limit on the depth of the search tree

Evaluation Function

The evaluation function first checks if the current state is an endgame state, in which case it will return -1000, 0, or 1000 respectively.

Otherwise, it will count all of the white pieces and all of the black pieces. It will also calculate the distance each piece is to its respective home castle and sum those up (positive for black pieces and negative for white pieces). So, the farther a piece is away from its home castle, the more points it will contribute to this value. This is done to give incentive to move your pieces forward.

It will also keep track of the distances of the two pieces for each color that are farthest from their home castles. If the 2nd place black piece has a greater distance than the 1st place white piece, then the evaluation function gives a positive bonus. This prioritizes winning the "race to the castle" if there is a clear path for both pieces to get to the end. A point deduction is given if the opposite is true, however, which would prioritize either jumping ahead in the race or capturing the pieces that are closer to the castle.

If there is only one black piece remaining, then the evaluation ignores any distance related calculations and just uses the number of pieces remaining as a basis. This will remove any distance-based incentive since reaching the castle is no longer an achievable endgame.