Mini-Checkers

David Tsai • CS 4613 Artificial Intelligence • Professor Ed Wong • Spring 2018

Running the game

Install Python 3.6 with Tcl/Tk. Extract the ZIP file to a location of your choice. In the Command Prompt (on Windows) or Terminal, change into the directory where you extracted the ZIP file. Make sure that the “minicheckers” folder is in this directory; it should have been created when the ZIP file was extracted. Do not change into the “minicheckers” directory. Run the following command:

```
python minicheckers
```

If the default version of Python on your system is not 3.6, then substitute the name of the Python 3.6 interpreter in the place of “python” in the command. If Python is not in your PATH environment variable, then use the full path or a valid relative path to the Python 3.6 interpreter.

You should see the Mini-Checkers window appear. By default, the computer plays as red, and you play as black. By default, black moves first. If you would like to play as black but let the computer move first, simply tick the “Red to move” radio box in the panel on the right. Click the New Game button at the bottom of that panel to start playing.

The design

The graphical user interface (GUI) displays the current positions of pieces on the board and allows the user to move one of his or her pieces when it is his or her turn. The GUI only allows legal moves. If one player has no legal moves to make, that player’s turn is forfeited, regardless of whether that player is the user or the computer.

A panel on the right side of the board allows the user to change which players the computer should play as, whose turn it is, and the strength of the computer. The game can be configured to allow two users to play against each other, to make the computer play against itself, or to let one user play against the computer. In the latter case, the user can choose to be red or black. The default settings satisfy the project requirements, which state that the computer is the red player and that the user is the black player. The player can choose to move first or second by adjusting whose turn it is before starting a new game.

The computer player is powered by an alpha-beta search. The cutoff depth is, for every search, initially set to 6 levels. After each alpha-beta search, if there is time left, another alpha-beta search is started again with the same parameters but with a deeper cutoff depth. Parts of the search are cached so that they need not be recomputed repeatedly. A separate thread waits 14.7 seconds, just under the limit of 15 seconds, kills the search, and returns the result from the last search that was not killed.

For every node in the alpha-beta search, a utility value is calculated. All utility values are of the Python float type. If the node is a terminal state, then its utility value is calculated according to the rules in the next section in this document. If the node is not a terminal state but the cutoff depth has not been reached, then search continues. Otherwise, the evaluation function is used to estimate the utility value. For more information on the evaluation function, see the following section in this document.

Terminal states

If all of one player's pieces are captured, then the other player wins. If neither player can make any legal moves, then the game ends and the player with more pieces wins. If neither player can make any legal moves but the players have an equal number of pieces, then the game ends in a draw.

If the red player wins, the utility value is ∞. If the black player wins, the utility value is $-\infty$. If the game ends in a draw, the utility value is 0. The float type in Python can represent positive and negative infinity natively.

The evaluation function

A *friendship* is defined as any one of these cases:

- A red piece is in the top row.
- A black piece is in the bottom row.
- A red piece is in the row above and in the column directly to the left or right of another red piece. In other words, a red piece is diagonally above another red piece.
- A black piece is in the row below and in the column directly to the left or right of another black piece. In other words, a black piece is diagonally below another black piece.

This function first calculates six parameters:

- $\ln\left(\frac{N_r}{N_b}\right)$, where:
    - N_r is the number of red pieces.
    - N_b is the number of black pieces.
- $\ln\left(\frac{F_r}{F_b}\right)$, where:
    - F_r is the number of red friendships.
    - F_b is the number of black friendships.
- $\ln\left(\frac{J_r}{J_b}\right)$, where:
    - J_r is the number of captures that the red player could make from the current arrangement of the pieces if it were his or her turn.
    - J_b is the number of captures that the black player could make from the current arrangement of the pieces if it were his or her turn.
- $\ln\left(\frac{M_r}{M_b}\right)$, where:
    - M_r is the number of moves that the red player could make from the current arrangement of the pieces if it were his or her turn.
    - M_b is the number of moves that the black player could make from the current arrangement of the pieces if it were his or her turn.
- $\frac{\sum(2.5-Y_r)+\sum(2.5-Y_b)}{2.5}$, where:
    - Y_r contains the row numbers of the red pieces, with the top row being 0 and the bottom row being 5.
    - Y_b contains the row numbers of the black pieces, with the top row being 0 and the bottom row being 5.
    - 2.5 is the middle of the board. It is between rows 2 and 3.

- $\frac{\sum(2.5-|X_r-2.5|)-\sum(2.5-|X_b-2.5|)}{2.5}$, where:
    - X_r contains the column numbers of the red pieces, with the left column being 0 and the right column being 5.
    - X_b contains the column numbers of the black pieces, with the left column being 0 and the right column being 5.
    - 2.5 is the center of the board. It is between columns 2 and 3.

Each parameter is then multiplied by a weight, and the total of the products is returned as the utility value. The weights change with the strength of the computer; to see what weights are used in each difficulty setting, see the next section in this document.

Levels of difficulty

The difficulty of the computer player can be adjusted in the panel on the right of the window. The computer's difficulty is internally adjusted by changing the weights that the evaluation function uses. These are the weights that are used for each difficulty level:

Difficulty	Log of Ratio of Numbers of Pieces	Log of Ratio of Numbers of Friendships	Log of Ratio of Numbers of Captures	Log of Ratio of Numbers of Moves	Average Row	Distance to Center Column
Easy	0.6979	0.2415	0.0835	0.0289	0.0100	0.0035
Medium	0.5495	0.7160	0.1225	0.3750	-0.1995	1.4420
Hard	0.4174	0.8370	0.0456	0.1986	0.1112	0.3588

The weights in the easy setting are simple: they are all $\frac{\ln^x 18}{100}$, where x ranges from 4 to -1 from left to right in the table above. Although there are 36 squares on the board, only 18 are squares where a piece could reside.

The other weights were obtained through trial and error. An automated script played through thousands of games where one player used the weights from the easy setting and where the other player used randomly-generated weights. Of the sets of weights that resulted in victories, the average was taken, and the same script was used again to match new random weights against them. Of the sets of weights that resulted in victories, the average was taken again. This was repeated several times.

After manually playing against the computer with each averaged set of weights, two were hand-picked to be the sets of weights for the medium and hard settings. These choices were made by feel.