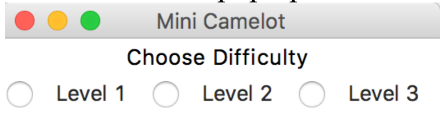
Kevin Foo
CS 4613
Final Project: Mini Camelot
Write Up

To run the game

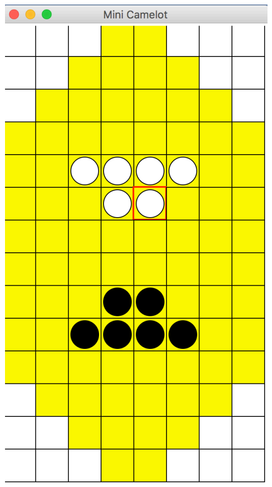
- Need to make sure you have the Tkinter library (usually included in Python build)
- Run the python script in any Python interpreter or,
- In terminal, `cd` to the folder where file is located and run "`python mini_camelot.py`"

To play the game

1. After running the program, "Choose Difficulty" window should pop up. Choose a level.


Mini Camelot
Choose Difficulty
Level 1 Level 2 Level 3

2. After choosing a level, the game board should appear and begin by selecting a white piece and a spot to move it to.



Description of program

The game was developed using the Python library Tkinter which also was used to create the GUI. The game starts out as asking the user for the difficulty level from a scale of 1 to 3 (3 being the hardest), which also determines the depth limit for the Iterative Deepening Search that is used to essentially create the tree. For example, if the player selected a difficulty of 3, this would indicate that we would be searching a depth level of 4 on the tree when running the Alpha-Beta Algorithm.

To start playing, the user makes the first move and clicks on a piece and a position that the player desires to move to. Through Tkinter, each time a player selects a piece, it triggers a function call, and when a space on board is selected another function is called. In order to determine if the move the player wants to make is possible, a possible moves list is created which includes all the possible coordinates the piece can travel to (capturing moves and cantering moves included). While creating a possible moves list, capturing moves also must be looked for since they are required to be taken. When combined the player move functionality is brought down to: checking if the desired position is range of game board, checking for terminal state (if two of a player's pieces are in the castle), determining if there is a capturing move and what it is, finding all the possible moves a piece can take, checking if a player or AI has won, being able to select a piece and board position to play.

After the player makes their move, it becomes the AI's turn and begins by calling the MaxValue function which then calls the MinValue function. In the Max and Min value functions, the AI has to first determine if a terminal state has been reached, so we can just return from there. Then it checks if there is a capturing move to take first because if there is, it is forced to take it and wouldn't have to search really beyond the first depth. If it's not a capturing move, then just like the player, the AI would have to go through a possible moves list for each piece it has and based on the utility value assigned to it and it is passed back in a list along with the v value. After the function returns from the MinValue or MaxValue function calls, it checks the values of alpha and beta to see if pruning is necessary.

The evaluation function is the main decision maker for the moves the AI performs. The evaluation function considers the distance of the piece to the castle and the number of pieces a player has remaining. The evaluation function also tries to prioritize the castle win over a capturing win. The function stems from the distance formula where the piece's distance is calculated from their own castle and the goal castle. The function also uses the player's remaining number of pieces to determine the weight of the move, since there should be a different play strategy when a player has even less pieces. For all the player's pieces, the function determines how close it is from the two goal coordinates and how far it is from its own goal coordinates. When distance is calculated, we now use the player's number of pieces remaining to calculate a weight. The formula to calculate the `nearVal` and `awayVal` takes into account minimizing the value of `awayVal` and maximizing `nearVal`. The way it was designed was so that when it becomes the hardest level, Level 3, the AI begins to play a defensive strategy right away by safeguarding its castle spaces, thus forcing the player to attack.