CS-171 Checkers Final AI Report

Team name: BasilAI

Member #1: Serena Tse -- sctse -- 21907178

Member #2: Jenny Xie Ng -- jxieng -- 60748939

I. In about 1/2 page of text, describe what you did to make your Final AI agent "smart."

Our Final AI agent uses a minimax algorithm in order to choose the next best move. In our minimax algorithm, we chose to use backtracking with the undo function provided to us. We decided to use the minimax algorithm because we wanted our AI to search for all the moves and choose the best move possible rather than cutting off a possible better move with alpha-beta pruning. We have a set maximum depth of three to stop the minimax algorithm from searching more than three turns ahead. We chose three as our depth because although a higher depth does better, it made our AI too slow at making decisions, so we decided that three was a deep enough search to do reasonably well, while not taking up too much time to choose which move to make.

In order to make our AI smarter, we have an evaluation function with multiple heuristics that add to a score variable. This evaluation function is called to evaluate (or score) the current board when either a player has won or if the depth reaches zero while searching for the next best move. One heuristic we used was counting the number of kings we have and subtracting the number of opponent kings. We weighed this heuristic more heavily compared to normal pawn pieces because the kings have more freedom to move around and hunt down the opponent's pieces. Another heuristic we utilized was counting the number of pawns we possessed and subtracting the number of pawns our opponent had. To weigh this heuristic, we added the row number that our pawn was currently on so that pawns that were closer to the opponent's side have a better score. This would make it so that the pawns could move towards the opponent's side to become kings.

II. In about 1/4 page of text, describe problems you encountered and how you solved them.

We first wrote our code in Java and everything seemed correct while going through our code, but our AI kept losing, even to the RandomAI. We decided to rewrite it in Python using the same heuristics that we had in Java and our AI started winning. We think our syntax while writing in Java using == and .equals when comparing ints and strings was incorrect. To fix this, we wrote our code in Python so we knew that our syntax when comparing ints and strings was correct.

Another problem we had was in our evaluation function. We noticed that our AI was not performing as well as it should have, even though we had a lot of heuristics which should be improving the performance of our AI. We realized that we forgot to subtract "points" when the opponent had good checker placements. After fixing this, our AI did a lot better. Additionally, changing the weights of some of our heuristics helped to improve our AI.

III. In about 1/4 page of text, provide suggestions for improving this project.

One suggestion we have for this project would be showing us a quick example of how to run our AI's. The gitlab page was a bit confusing and we felt that if an example were shown to us (in lecture, discussion, or on the gitlab page) we would not have wasted time trying to figure out the parameters for the commands.

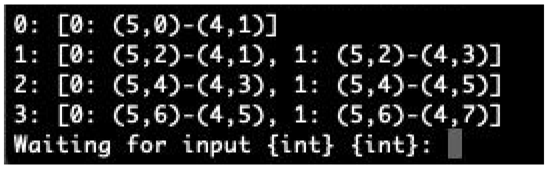
It would also help if we were told that a tie would occur after a maximum number of moves. We spent a while trying to figure out why we were getting a tie when it was possible for someone to win.

Another suggestion that would have been helpful would be to give us a picture of the boards being tested on, for example:

```
   0  1  2  3  4  5  6  7  8
0  .  b  .  b  .  b  .  b  .
1  b  .  b  .  b  .  b  .  b
2  .  .  .  .  .  .  .  .  .
3  .  .  .  .  .  .  .  .  .
4  .  .  .  .  .  .  .  .  .
5  .  .  .  .  .  .  .  .  .
6  .  w  .  w  .  w  .  w  .
7  w  .  w  .  w  .  w  .  w
```

When we tried to draw out the board on paper, we ended up placing the starting checkers on the wrong spots, since we were not too familiar with checkers.

Our last suggestion would be to give us an example of how the array works when trying to run the AI manually.



```
0: [0: (5,0)-(4,1)]
1: [0: (5,2)-(4,1), 1: (5,2)-(4,3)]
2: [0: (5,4)-(4,3), 1: (5,4)-(4,5)]
3: [0: (5,6)-(4,5), 1: (5,6)-(4,7)]
Waiting for input {int} {int}:
```

Looking at the picture above was confusing to us and although it was slightly explained in the get all possible moves section in the student manual, it was still confusing. Providing an example in the student manual would have been helpful. For example, 0: [0: (5,0)-(4-1)] meant that if we input 0 0, it would move the checker in row 5 col 0 to row 4 col 1.