CSC 320 Programming Assignment 4
Konane Player
Due Date 1: Monday October 1, noon

For this assignment you will implement the minimax adversarial search algorithm with alpha-beta pruning on the game of Konane, also known as Hawaiian Checkers. The game is played on an N by N board of black and white pieces. N may be as small as 4 or as large as 8. The board shown below is 8 by 8, and uses 'B' for black pieces and 'W' for white pieces.

```
   0 1 2 3 4 5 6 7
0  B W B W B W B W
1  W B W B W B W B
2  B W B W B W B W
3  W B W B W B W B
4  B W B W B W B W
5  W B W B W B W B
6  B W B W B W B W
7  W B W B W B W B
```

In order to play Konane, the black player goes first and removes one black piece from the corner or the center of the board. For an 8 by 8 board, this equates to positions (0,0), (3,3), (4,4), or (7,7). Next the white player removes a white piece adjacent to the space created by the first move. Then the players alternate moves, each jumping one of their own pieces over one horizontally or vertically adjacent opponent's piece, landing in an empty space on the other side, and removing the jumped piece. If desired, this may be continued in a multiple jump, as long as the same piece is moved in a straight line. The game ends when one player can no longer move and that player is considered the loser.

I will provide you with a Python implementation of the Konane game. You will implement minimax and then focus on finding a good evaluation function for estimating the value of a given Konane board. It has been noted that "the performance of a game-playing program is dependent on the quality of its evaluation function" (Russell and Norvig, page 171).

Implementing Minimax with Alpha-Beta Pruning

Download the relevant files

1. Open the file konane.py and read through the definitions provided. The board size must be specified in the constructor for the Konane class. There is also a base class called Playerand there are three specializations of this class: HumanPlayer, SimplePlayer, andRandomPlayer. Each of these player definitions inherits from both the Player class and theKonane class. Therefore, the board size must also be specified in the constructors for these classes.
2. Modify the calls at the bottom of the konane.py to play a game between the RandomPlayerand the SimplePlayer on an 8 by 8 board.
3. In the file minimax.py, implement a MinimaxNode class to hold the necessary information for a search node in a minimax game tree. It should include the following data:
    - state: the current board configuration
    - operator: the move that resulted in the current board configuration
    - depth: the depth of the node in the search tree
    - player: maximizer or minimizer

4. In the file minimax.py, implement a MinimaxPlayer class that inherits from both thePlayer class and the Konane. Your class must override the abstract methods initialize andgetMove from the Player class. It should also implement several other methods, which are described in the file.

   Notice that the first line of the MinimaxPlayer constructor calls the constructor of theKonane base class. Also notice that the depth limit for the search is passed in as an argument to the class. Therefore you will not need to pass it as an argument to the minimax method. You should be able to use your MinimaxPlayer in the following way:

   ```
   game = Konane(8)
   game.playNGames(2, MinimaxPlayer(8, 2), MinimaxPlayer(8, 1), False)
   ```

   Executing the above commands should demonstrate that the minimax player which searches to depth 2 will outperform a minimax player which only searches to depth 1.

   ```
   Game 0
   MinimaxDepth2 wins
   Game 1
   MinimaxDepth2 wins
   MinimaxDepth2 2 MinimaxDepth1 0
   ```

5. Implement the alpha-beta minimax search based on the **pseudocode** linked to from the class web page
6. Implement an appropriate static evaluator for Konane. The key difference between Konane players will be in how they evaluate boards. Based on our reading, the static evaluator should:
   - Order the end states correctly
   - Be time efficient to compute
   - Be strongly correlated with the actual chances of winning

   Explain your static evaluator in the file evalDescription.txt Discuss the various features you considered including as part of the evaluation function. How did you determine which features were the most effective? This should be between one and two pages of single-spaced text.

Tournament Play

We will hold two tournaments. One tournament will have both a depth limit of 4 and a time limit of 5 seconds. The other tournament will have only a time limit of 5 seconds.

Each Konane player should conduct a minimax search and return a move within 5 seconds. If a move is not returned within the time limit, then a random move will be selected. In addition, each Konane player may not use more than half a gigabyte of memory. This can be monitored using thetop command.

Each Konane player will play every other Konane player in the tournament twice, one time going first and the other time going second. Each player will receive a +1 for a win. The winner of the tournament will be determined by the cumulative score over all of these games. It will be possible to have multiple winners.

During class on Monday, October 10, we will hold the tournament to determine whose program is the ultimate Konane player. If for some reason your program is not operational, **you** must compete as a human in the tournament.

Submit: Turn in all relevant files via nexus.