How to use:

To use the software I wrote as part of this assignment one simply needs to launch the terminal, navigate to the folder that contains the 2 files (AI_player.py, Node.py), and the 2 files required for playing (play.py, board.py). After navigating to the folder a symbolic link needs to be made between the given player and the piece it will be playing, this can be done by typing as an example:

```
ln -s AI_player.py Red
```

Inside the code for AI_player one needs to set parameters for searching; the depth always needs to be an even number as the algorithm in node.py assumes even numbered depths are moves made by that player and odd number depths are those made by an opponent. The argument to Get_move that's made in AI_player states whether to use the easy heuristic or the hard heuristic (1 for easy anything else for hard).

The algorithm for AI_player.py will generate all the necessary arguments for Node.py based off the board (apart from the two stated earlier) and will write the result to the given move needed. Node.py will do the scoring of the board and all child states (less pruning).

Results:

Below I've laid out a comparison on the results of the easy ai heuristic for the first 10 moves:

Depth	**2**	**4**	**6**
Time	0.1 to 0.5 Seconds	0.5 to 1 Second	1.5 to 3 Seconds
Total states	200 to 800	1500 to 3000	4000 to 12000

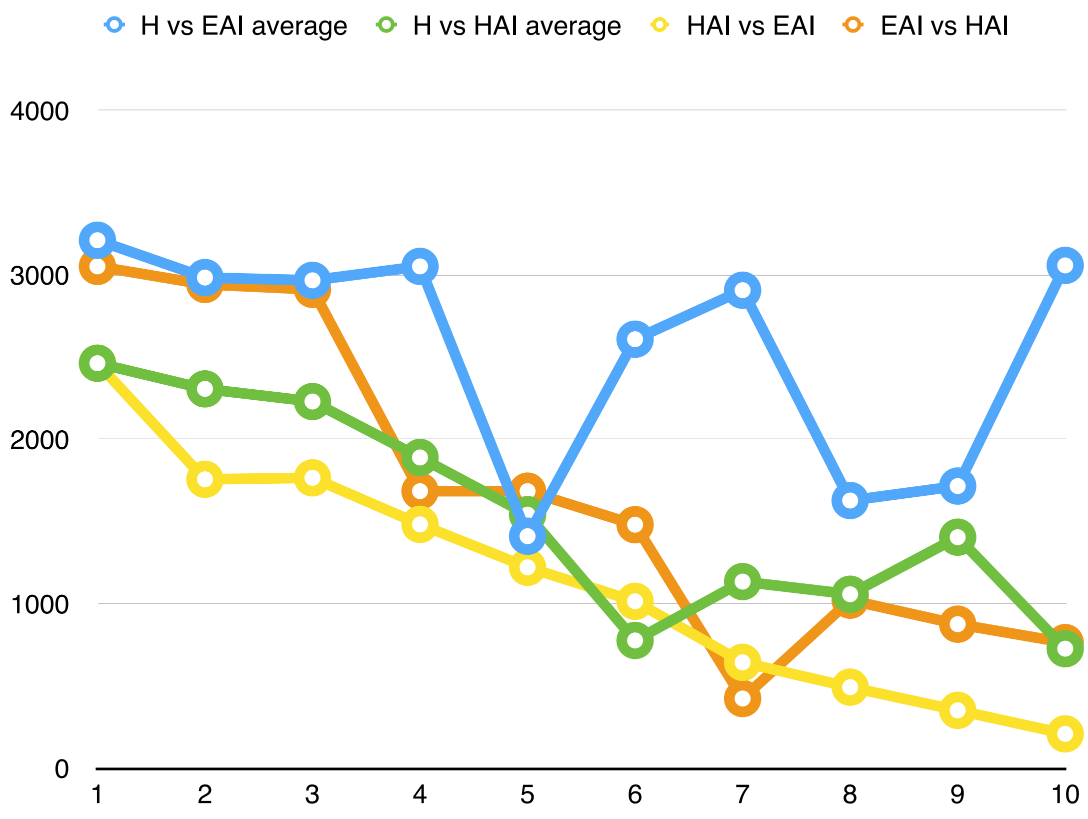
And here is a comparison of the harder scoring heuristic for the first 10 moves:

Depth	**2**	**4**	**6**
Time	0.1 to 0.5 Seconds	0.1 to 1 Second	.5 to 3 Seconds
Total states	120 to 800	300 to 3000	1500 to 12000

As you can see both from the tables and the charts below although they have similar maximum bounds for both the time and total states visited in the turn the lower bounds of the two are quite different. The harder heuristic steadily declines the number of states it looks at by a fairly constant rate while the easy ai tends to stay looking at the maximum state unless there is a potential win condition for either it or its opponent.

One interesting thing to note; the way the algorithm is currently setup if two ai players play against each other they tend to make the same moves until one has a potential win condition. This is due to the fact that the algorithm will automatically fill up the columns left to right unless another player forces it to make another decision (such as playing in the middle rows or at the end). This could be solved by choosing a random column for the first couple moves rather than choosing 0 (by default).

The chart directly below compares the average number of moves across 10 games each between a Human (H) easy heuristic (EAI) and hard heuristic (HAI) in different combinations at a depth of 4 (2 moves prediction by the AI and 2 by its opponent). The spikes in the H vs EAI game are caused by boards that result in a potential win state by either the computer or myself.



<u>Other notes:</u>

While the AI is generally reactive rather than predictive there are several instances where the AI chooses not to react due to bugs in the code. One way of seeing this is if you lead the AI to a potential win state for yourself several times it will block each turn until it decides to focus on leading itself to a win state rather than blocking your move (generally leading to you winning)

In a human vs AI game, most of the wins are by the human with the ai taking the win at random times when it sees a move you as a player don’t that was made up of its blocking; this tends to occur approximately 30-40% of the time when playing against the hard heuristic and 15-20% of the time when playing against the easy heuristic.