Option 3: Build an agent using advanced search techniques (for example: killer heuristic, principle variation search (not in lecture), or monte carlo tree search (not in lecture))

Create a performance baseline using run_search.py to evaluate the effectiveness of a baseline agent (e.g., an agent using your minimax or alpha-beta search code from the classroom)

In [3]:

```
print("The effectiveness of alpha-beta search is as below:")
print("Minmax: 60%")
print("Random: 100%")
print("Greedy: 100%")
```

```
The effectiveness of alpha-beta search is as below:
Minmax: 60%
Random: 100%
Greedy: 100%
```

Use run_search.py to evaluate the effectiveness of your agent using your own custom search techniques

In [4]:

```
print("The effectiveness of principle variation search is as below:")
print("Minmax: 15%")
print("Random: 60%")
print("Greedy: 70%")
```

```
The effectiveness of principle variation search is as below:
Minmax: 15%
Random: 60%
Greedy: 70%
```

You must decide whether to test with or without "fair" matches enabled--justify your choice in your report

I decide test with "fair" matches enabled. Because in some games, picking a winning move for the opening guarantees the player a victory. Playing "fair" matches this way will balance out the advantage of picking perfect openings.

Q1:How much performance difference does your agent show compared to the baseline?

A1: My agent(principle variation search) won 5% more games than baseline(alpha-beta search).

Q2:Why do you think the technique you chose was more (or less) effective than the baseline?

A2: Because principle variation search will never examine a node that can be pruned by alpha-beta.

In [ ]:

```
```