CSL-3090 ARTIFICIAL INTELLIGENCE PROJECT REPORT

SOUMIK ROY (B20AI042)
STUTI ASWANI (B20AI065)
YASH BHARGAVA (B20AI050)

Github Repo | Demo

Abstract

Emanuel Laskar once said, “When you see a good move, look for a better one.” Well, what if there was an AI that could do that for you? Presenting Stalemate, our attempt to build a robust AI that can pit 2 chess pieces against each other and use classic heuristics to determine who will win.

The game comes with 4 player options:
1. Human Player
2. AI with Minimax
3. AI with Alpha-Beta Pruning
4. Random Player

There are also 4 types of heuristics involved:
1.. Open Move Score (OMS)
2. Improved OMS
3. Weighted OMS
4. Farsighted Score

Problem Statement

The goal of this project is to build an AI that can feasibly and optimally play a game of stalemate between 2 chess pieces that can be any of the following: Knight, Bishop, Queen and Rook.

Background Of The Game

The 2-player game begins with both players deciding which piece they will be using. When using AI, at every point, the best possible move will be chosen as the next move.

The players can be one of 4 types:
1. An AI that uses a minimax algorithm to determine its next move.
2. An AI that uses the alpha-beta pruning algorithm to determine its next move.
3. A Human Player (user inputs the position they want to move to)
4. A player that randomly selects a move from available legal moves.

The game will then end when there are no possible moves left for a player to play (no valid moves), or a player is unable to make a move in the given time(forfeit). In either case, the player unable to make a move will lose.

Motivation

One of the most famous questions in classical chess is which one of the Bishop and the Knight is more powerful. As humans through the decades have pondered on and debated this problem, we decided to come up with a game that puts these 2 pieces of chess against each other. We then built an AI agent that could use modified versions of the Heuristics and Pruning techniques taught to us in the course CSL3090 - Artificial Intelligence, that can play the game by itself, making the best possible decisions and trying to win the game.

This would also enable us to gain insight into these techniques and give us a greater understanding of them, as we translate our penned class notes into a functioning code.

Not only do we now have an answer to the question, but our enthusiasm also led us to expand the set of pieces to include the Queen and the Rook, and allowed us to pit all these pieces against each other. As a result, we ended up developing a full-fledged game where you can have 2 users go against each other or 2 AI agents go against each other, or even better, have a user go against an AI agent trying to beat it!!

Study / Experiment Setting

We went through several studies and concluded that Minimax and Alpha Beta pruning would be the best-used AI algorithms for the given deterministic game problem. We also went through several heuristics that are commonly used to evaluate the score for a given state of a chess game and found the following as best suited for the problem at hand:

1. **Open Move Score (OMS)**: Given a player, the number of legal moves possible for it is known as OMS.
2. **Improved OMS**: Given a player and its opponent, improved-OMS is {number of legal moves possible for the player - number of legal moves possible for its opponent}.
3. **Weighted OMS**: It is similar to Improved OMS, just with the difference that here the score is
   {legal moves possible for the player - (weight) * number of legal moves possible for its opponent}
4. **Farsighted Score**: It takes into account the moves possible by the opponent after the current player makes any possible move. The score is
   {legal moves possible for the player - (weight) * number of legal moves possible for its opponent after any of the above legal moves is made}
5. **Null Score**: It returns +∞ if a given player has won, -∞ if lost, else 0.

Demonstration

The entire project codebase is at https://github.com/Soumik-Roy/StaleMate.

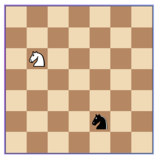
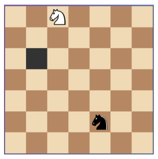
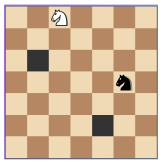
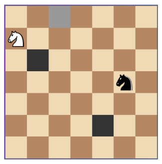
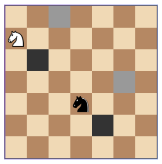
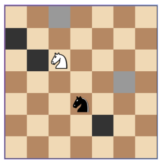
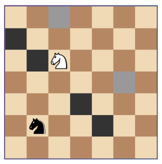
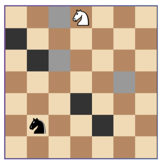
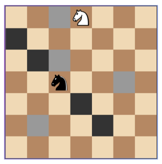
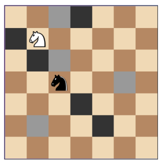
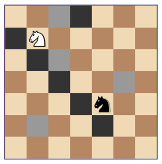
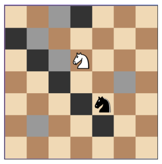
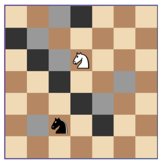
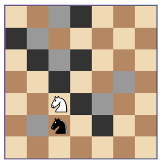
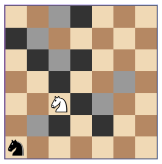
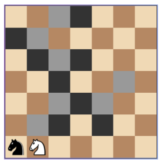
Once you have the project folder, open it in the terminal, then install all dependencies using the following command.

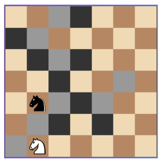
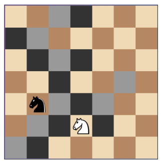
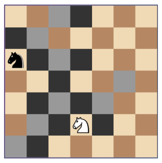
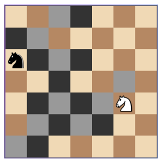
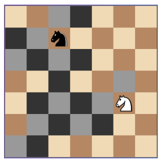
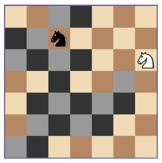
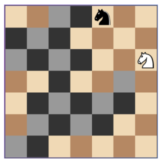
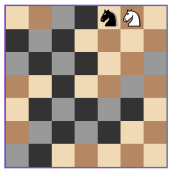
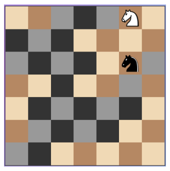
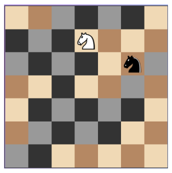
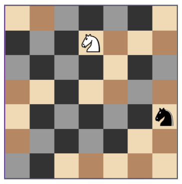
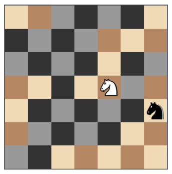
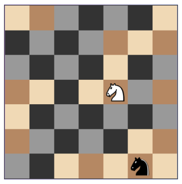
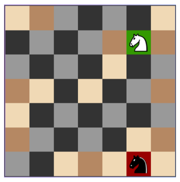
```
pip install -r requirements.txt
```

Now to play the game use the command:

```
python play.py
```

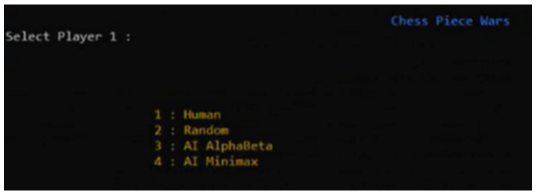
Below you can see a move by move simulation of a Knight vs Knight game, played between AI Minimax and Alpha-Beta agents.
Also, a demonstration video of a similar match can be seen at this link.

Move 1		Move 2	
Move 3		Move 4	
Move 5		Move 6	
Move 7		Move 8	
Move 9		Move 10	
Move 11		Move 12	
Move 13		Move 14	
Move 15		Move 16	

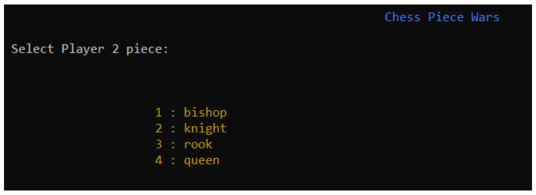
Move 17		Move 18	
Move 19		Move 20	
Move 21		Move 22	
Move 23		Move 24	
Move 25		Move 26	
Move 27		Move 28	
Move 29		Move 30	

Here are a few more screenshots of the game terminal:

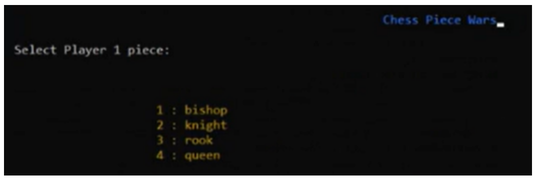
Selection of players 1, 2 and their pieces:



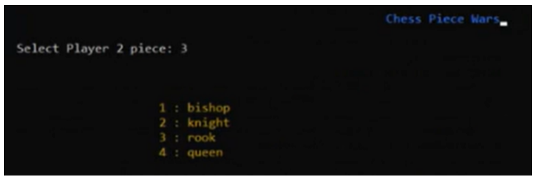
```
                                                Chess Piece Wars
Select Player 1 :



            1 : Human
            2 : Random
            3 : AI AlphaBeta
            4 : AI Minimax
```



```
                                                Chess Piece Wars
Select Player 2 piece:



            1 : bishop
            2 : knight
            3 : rook
            4 : queen
```

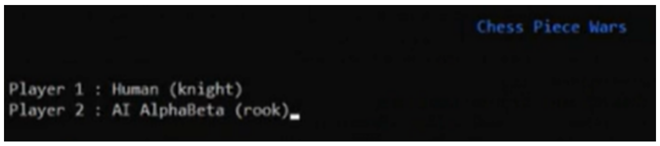


```
                                                Chess Piece Wars
Select Player 1 piece:



            1 : bishop
            2 : knight
            3 : rook
            4 : queen
```



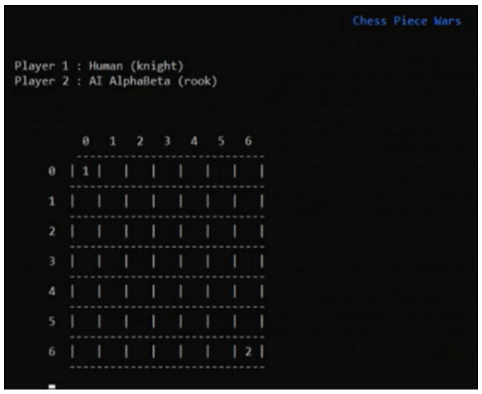
```
                                                Chess Piece Wars
Select Player 2 piece: 3



            1 : bishop
            2 : knight
            3 : rook
            4 : queen
```

The confirmation of players 1 and 2:



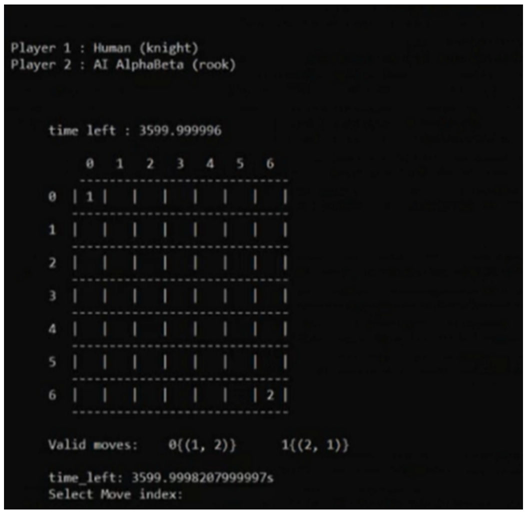
```
                                                Chess Piece Wars


Player 1 : Human (knight)
Player 2 : AI AlphaBeta (rook)
```

Starting positions:



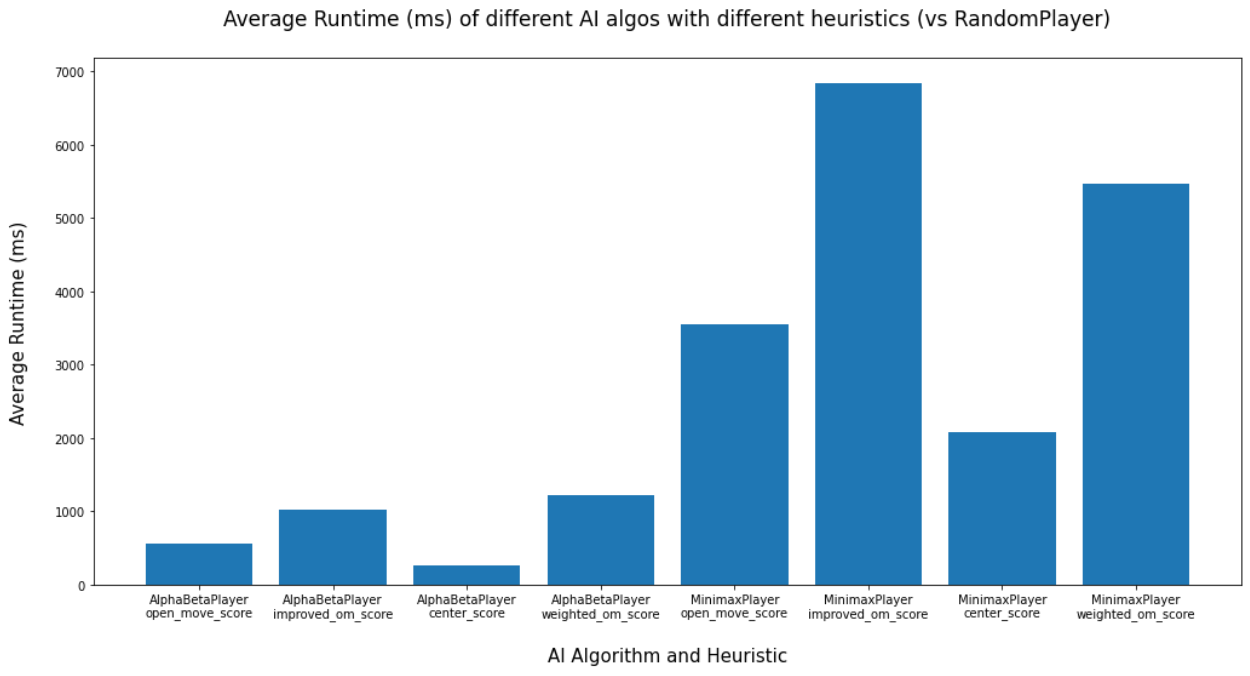
```
                                                Chess Piece Wars

Player 1 : Human (knight)
Player 2 : AI AlphaBeta (rook)


        0   1   2   3   4   5   6
      -----------------------------
  0   | 1 |   |   |   |   |   |   |
      -----------------------------
  1   |   |   |   |   |   |   |   |
      -----------------------------
  2   |   |   |   |   |   |   |   |
      -----------------------------
  3   |   |   |   |   |   |   |   |
      -----------------------------
  4   |   |   |   |   |   |   |   |
      -----------------------------
  5   |   |   |   |   |   |   |   |
      -----------------------------
  6   |   |   |   |   |   |   | 2 |
      -----------------------------
```

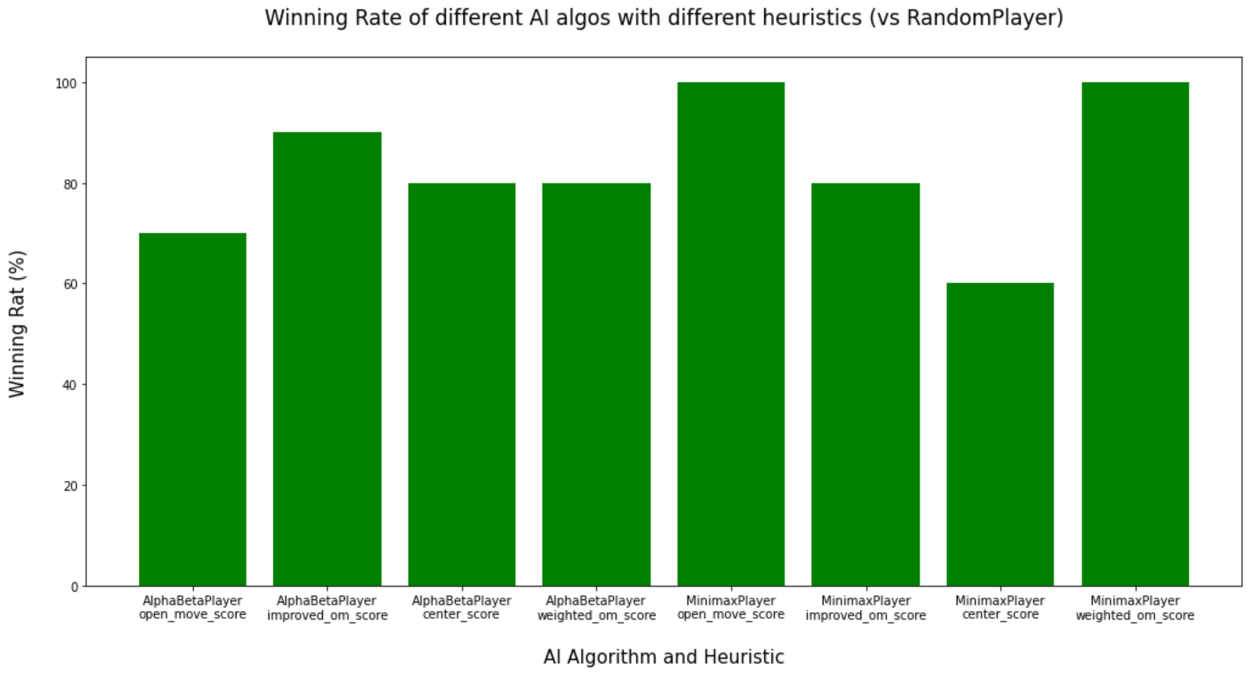
Mid-game board, showing valid moves to the human player:

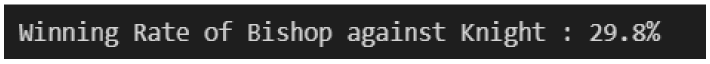


```
Player 1 : Human (knight)
Player 2 : AI AlphaBeta (rook)


    time left : 3599.999996

          0   1   2   3   4   5   6
        -----------------------------
    0   | 1 |   |   |   |   |   |   |
        -----------------------------
    1   |   |   |   |   |   |   |   |
        -----------------------------
    2   |   |   |   |   |   |   |   |
        -----------------------------
    3   |   |   |   |   |   |   |   |
        -----------------------------
    4   |   |   |   |   |   |   |   |
        -----------------------------
    5   |   |   |   |   |   |   |   |
        -----------------------------
    6   |   |   |   |   |   |   | 2 |
        -----------------------------

    Valid moves:    0{(1, 2)}      1{(2, 1)}

    time_left: 3599.9998207999997s
    Select Move index:
```

Results







```
Winning Rate of Bishop against Knight : 29.8%
```

Conclusion and Future Work

We reached the goal that we had initially set for ourselves. After rigorous testing, we can confirm that the AI indeed does work. As seen from the graphs, the Alpha-Beta pruning algorithm takes lesser time to run than the Minimax algorithm.
Also, in case you were wondering, the Knight is stronger than the Bishop, in this particular game.
Our future work will revolve around transforming the architecture to support multiple players instead of just 2. This will enable us to visualize more complicated battles between pieces.

References

We referred to the material provided for the course, as well as the following papers:

https://www.ijcai.org/Proceedings/75/Papers/048.pdf

https://www.whitman.edu/documents/Academics/Mathematics/2019/Felstiner-Guichard.pdf

https://vision.unipv.it/IA1/ProgrammingaComputerforPlayingChess.pdf

https://arxiv.org/pdf/1208.1940.pdf

https://www.globalvoxinc.com/wp-content/uploads/2021/06/AI_term_paper.pdf

Individual Roles

The initial idea, player distribution and choice of heuristics were discussed and chalked out collectively.
Soumik implemented the playing agents, Stuti implemented the heuristics and they worked collectively with Yash to integrate them as a single unit.
Yash also made the online simulation of the game.
The report had equal contributions from everyone.