Chess Bot - An Automated Chess Player Using Reinforcement and a Double Deep Q-Learning Network

Sriram Ayalavarapu, Sidharth Rao, Ashvath Suresh Kumar

1. Introduction

1.1 Inspiration:

Chess is an age old sport that has paved the way for many future games. Chess is considered to be one of the most complex games, consisting of an unfathomable number of board states and combinations of pieces. Using just brute force, a computer would still not be able to compute all of the different possible moves and find the best way to win, so we made it our goal to make an effective Chess Player using Reinforcement Learning.

1.2 Basic Project Description:

Our project, in its simplest form, is an automated chess player. There are 2 major components to this: the game setup and the AI. For the AI, we used Reinforcement Learning, specifically Double Deep Q-Learning, which will be explained later in this paper. This project was coded in python, utilizing Google Collaboratory. In order to do this project, we needed to research previous projects in this field to draw inspiration. We then needed to create the three main components: Game Setup, Learning Algorithm, and Training. Finally we needed to decide on future developments to pursue.

2. Literature Review:

The Literature review is an essential part of any project. It is in this stage that we research previous projects and draw some inspiration. This stage also helps us understand the task at hand, as we see what others have done, allowing us to plan our course of action. Below is a quick summary of the research that we have done on a few of the major chess AIs.

2.1 Bernstein Program:

The Bernstein Program, developed in 1957, was the first complete chess program. It was a project of IBM and was developed by Alex Bernstein, Michael de V. Roberts, Timothy Arbuckle, and Martin Belsky as programmers, Nathaniel Rochester, who served the role of supervisor, and Arthur Bisguier, a chess grandmaster. This project effectively introduced the idea of a chess AI and led the way for many future projects to come.

2.2 Deep Thought:

Deep thought was the next major chess AI, being one of the first Chess AI to beat a “grandmaster”. It was developed in 1985 by Feng-hsiung Hsu, Thomas Amaranathan, Murray Campbell, Andreas Nowatzyk, Mike Browne, and Peter Jansen. In 1988, it won a match against Ben Laster, one of the greatest grandmasters of the time, marking a revolutionary moment for AI.

2.3 Deep Blue:

Deep Blue is perhaps the most popular chess robot. It is the IBM-branded successor to Deep Thought, developed by the same crew. It is known for being the first chess computer to beat a world champion under normal time constraints in both a single game and an overall match. It managed to beat Garry Kasparov in a single game on Feb. 10, 1996 but lost the overall match. It did, however, win a rematch in 1997, with a score of 3.5 - 2.5. This was perhaps the most important moment in Chess AI history. It marked the ultimate victory of not only Chess AIs, but to many AI enthusiasts, the victory of machine over man.

2.4 Stockfish 11:

Stockfish 11 has the highest chess rating till date. It has achieved a monstrous chess rating of 3452. To put this in perspective, the highest chess rating of a human is Magnus Carlsen's, at 2882. That's more than a 500 point difference! It is a free and open source chess engine. We are planning on, after training our AI for a few more weeks, using Stockfish to train it, allowing it to become exposed to players that are far beyond the level of even a grandmaster. This will show our AI much more effective movement sets, including many different openings.

3. Summary:

3.1 Base/Game Setup:

We started our project by making a system for playing chess.

3.1.1 Moves:

Our approach to defining the moves was simple. We created functions for each type of piece that defined every "raw" possible move that the piece could perform. We then trimmed this set to all moves that were on the board. Below is an example of our code for all basic moves of the knight:

```
  def knight(piece , board): # piece is the coordinates of the knight and board
is the actual current state of the board.
    # assigning temporary variables to each component of the coordinates
    row = piece[0]
    column = piece[1]
    # all raw moves: move forward 1 space, move 2 spaces to the left ...
    possiblemoves = [[row+1, column+2],[row+1,column-2],[row-1, column-2],[row-
1, column+2],[row+2, column-1], [row+2, column+1], [row-2, column+1], [row-2,
column-1]]
    # trims to fit the board
    for possiblemove in possiblemoves:
      possiblemove.append(moves.verify(possiblemove, piece, board))
    return possiblemoves
```

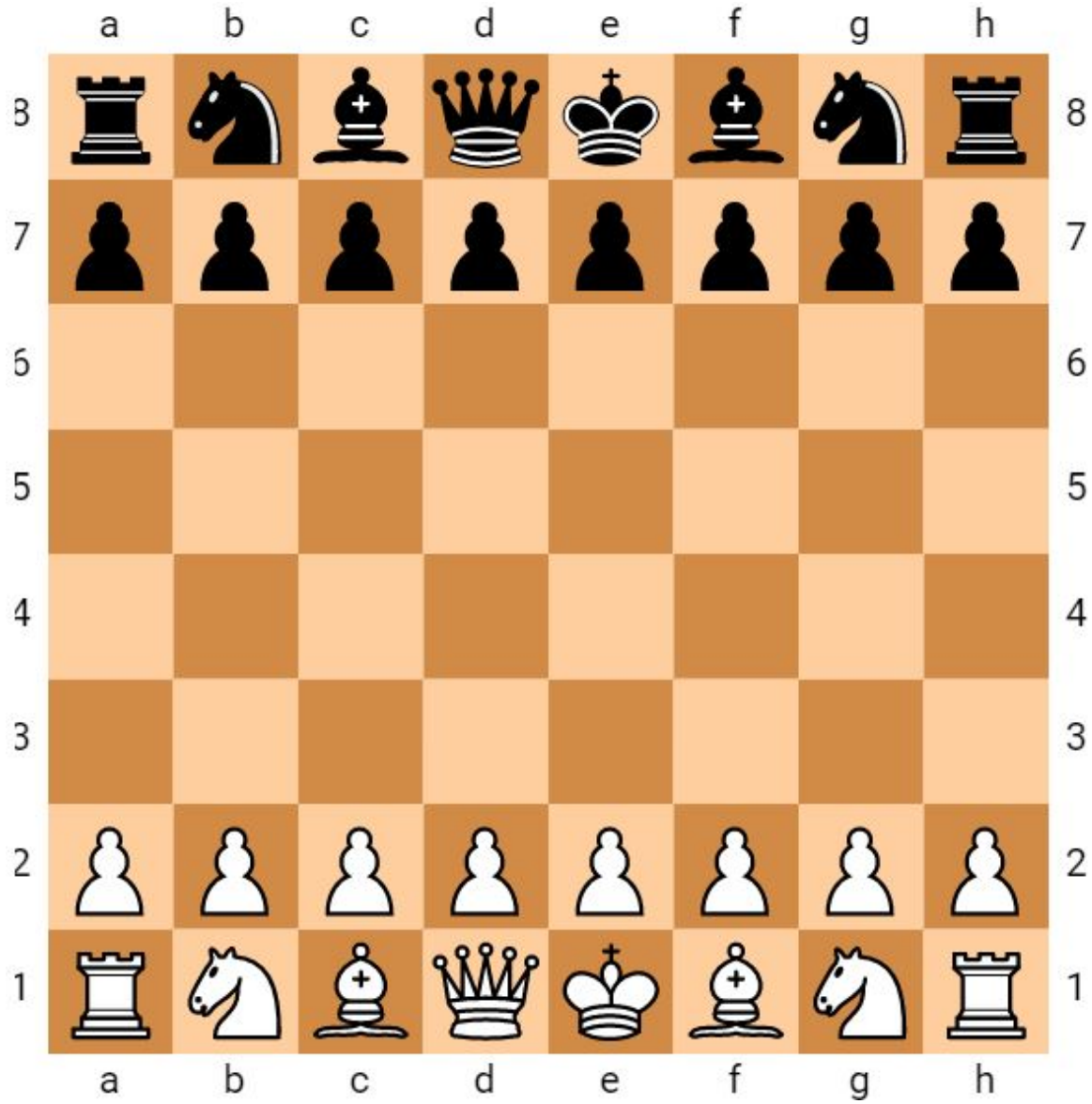
Our next step was to send these move lists through a set of filters that eliminated illegal moves, such as moves that caused the king to go into check. Then we created a function that merged all of these moves for all pieces at any given time. We also added many other rules and restrictions, such as moves in check, pawn promotion, and of course, checkmate. The final step was to create a function that executed a move.

3.1.2 Display:

For the board, we originally defined a numpy array to serve as the Chess Board:

```
board = [['Rb1', 'KNb1', 'Bb1', 'Qb' , 'Ib' , 'Bb2', 'KNb2', 'Rb2' ],
         ['Pb1', 'Pb2' , 'Pb3', 'Pb4', 'Pb5', 'Pb6', 'Pb7' , 'Pb8' ],
         ['blk', 'blk' , 'blk', 'blk', 'blk', 'blk', 'blk' , 'blk' ],
         ['blk', 'blk' , 'blk', 'blk', 'blk', 'blk', 'blk' , 'blk' ],
         ['blk', 'blk' , 'blk', 'blk', 'blk', 'blk', 'blk' , 'blk' ],
         ['blk', 'blk' , 'blk', 'blk', 'blk', 'blk', 'blk' , 'blk' ],
         ['Pw1', 'Pw2' , 'Pw3', 'Pw4', 'Pw5', 'Pw6', 'Pw7' , 'Pw8' ],
         ['Rw1', 'KNw1', 'Bw1', 'Qw' , 'Iw' , 'Bw2', 'KNw2', 'Rw2' ]]
```

Now, this doesn't look very visually appealing, so we chose to use an already existing chess database (PyChess) to present the current chess board to the user:



We then repurposed the numpy array board to allow the code to interact with a simple board that had no images, only text, making it easier for the code to interpret the board. We synced the numpy array and the displayed board so that any update on the numpy array would be reflected in the display board.

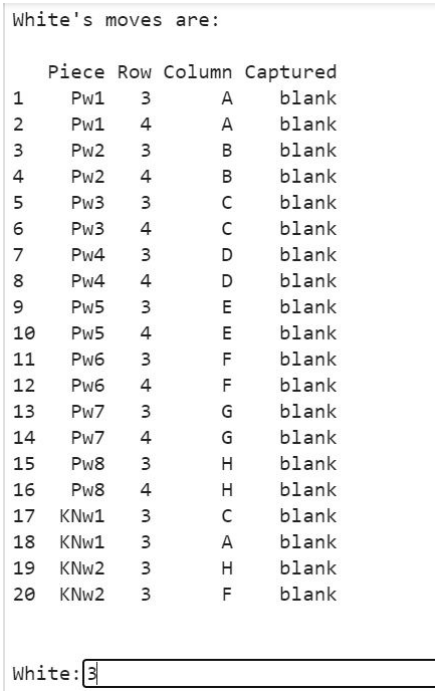
3.1.3 Player Input:

The final part of the basic game setup is to create a method for players to input moves. For this, we went to 3 methods:

- Entering an index for the move:

- This method had us display a table of moves that showed the move ID - a number from 1-x, with x being the number of possible moves at the time- and some basic information about the move, including which piece would be moved, where it would be moved to, and which piece the move would capture.
- The actual input would be the move ID.
- We decided not to use this approach because it took up too much space and, in occasions where there are many possible moves, it becomes difficult to read all of the moves to decide which one is the desired move.

Here is a picture of this method:



White's moves are:

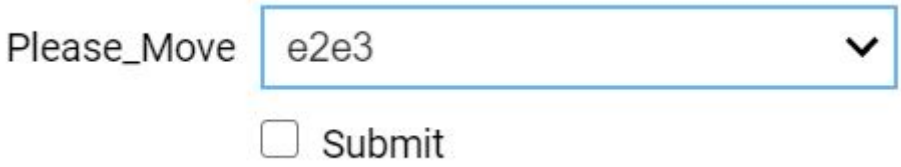
	Piece	Row	Column	Captured
1	Pw1	3	A	blank
2	Pw1	4	A	blank
3	Pw2	3	B	blank
4	Pw2	4	B	blank
5	Pw3	3	C	blank
6	Pw3	4	C	blank
7	Pw4	3	D	blank
8	Pw4	4	D	blank
9	Pw5	3	E	blank
10	Pw5	4	E	blank
11	Pw6	3	F	blank
12	Pw6	4	F	blank
13	Pw7	3	G	blank
14	Pw7	4	G	blank
15	Pw8	3	H	blank
16	Pw8	4	H	blank
17	KNw1	3	C	blank
18	KNw1	3	A	blank
19	KNw2	3	H	blank
20	KNw2	3	F	blank

White: 3

- Type in the move in long notation:
  - Long notation is a form of documenting a chess move by writing the coordinates at which the piece started and where the piece will move to. For example, e6e7 means "move the piece at column "e" and row 6 to column "e" and row 7". Please note that, for Pawn Promotion, you must append "q" for queen, "r" for rook, "b" for bishop, etc. at the end of the move. In long notation, castling on the King's side is noted as "O-O", while the queen's side is "O-O-O".
  - We chose to use this approach because it takes little space and looks more visually pleasing.
  - However, it does require some pre-requisite knowledge of whether or not a move is legal, though it does return that the move is invalid and allows one to enter another move.

  Here is a picture of this approach:

- Select from a Dropdown:
  - This option is perhaps the simple. The player simply selects their move form a list of all of the dropdown options, which are displayed in Long Notation.
  - The cons to this approach was that the dropdown blocked the board, so one could not look at the move list and scrutinize the board at the same time. Another issue was that the dropdown could be changed after the move has been finalized. In other, the move can be undone.
  - The pros, however is that it is both visually appealing and requires no prerequisites.



3.2 Learning:

3.2.1 Q-Learning:

For our project, we decided to use Q-Learning. Q-Learning is a system in which an AI, or the **agent** plays multiple games, and after each game, updates all the rewards for the moves in that game. A **reward** is the instantaneous benefit of an action. The **value** is a quantity which allows the computer to compare 2 moves. The equation is as follows:

$V^{\pi}(s) = \Sigma_{t=0}^{\infty} \gamma^t r(t)$

This equation says that the value of a policy is equal to the discounted sum of the rewards that result from it. The discounted sum can be explained with an example. γ is a discount factor between 0 and 1, non-inclusive. If the reward is 1 for the first step, 10 for the second, and 100 for the third, with a discount of 0.1, the value would simply be $1 \cdot 0.1^0 + 10 \cdot 0.1^1 + 100 \cdot 0.1^2 = 3$.

The agent compares two **actions** and determines the superior one by finding the **optimal policy**, using the following equation:

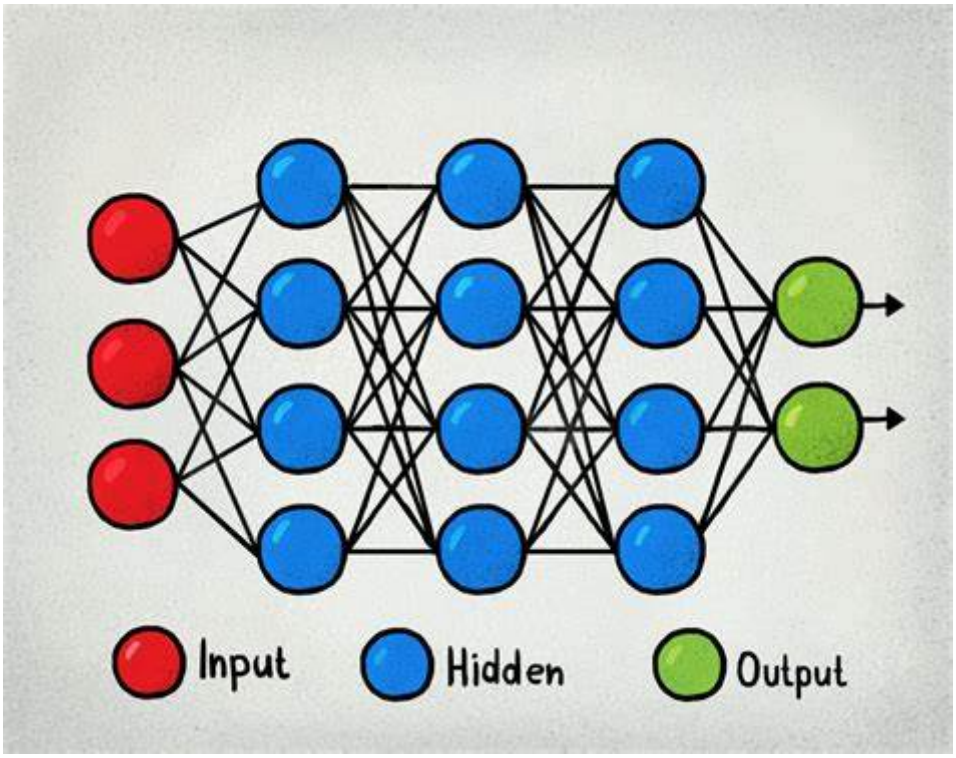
$V^{\pi^*}(s) = max_a(R(s,a) + \gamma V^{\pi^*}(T(s,a)))$

This equation essentially says that the optimal policy is equal to the policy that take action a, where a produces the highest discounted sum of rewards

When deciding every move, the computer will go through all the moves ever taken and will determine the best one with the highest reward. After each game, the AI will update its list of moves based on both the instantaneous reward and the long-term **value**, a quantity which allows it to understand the future repercussions of an action.

3.2.2 Neural Networks:

A neural network is a series of algorithms that try to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. It essentially takes an input, in some form or another, and connects every element in the input to an element in the hidden layer, which do the actual processing. These hidden layers then send out an output. For example, in image recognition, the input you be the pixels. There would be a connection between each neuron of layer 1 and each pixel. The output of layer one would become the input of layer two and so on, with the output of the final layer being the overall output. Please refer to the diagram below to understand this better.



3.2.3 Deep Double Q-Learning:

As previously stated, we created a DDQN Model but did not have enough time to train it. Hence, we reverted back to q learning for the time being. Essentially, the DDQN (Double Deep Q Networks) agent uses two neural networks to learn and predict what action to take at each step. The first network, referred to as the Q network, is used to predict what to do when the agent encounters a new state. It takes in the state as input and outputs Q values for the possible actions that could be taken. The second network, or the deep network, solves the problem of overestimation of the Q values by calculating the q-target $Q^*(s_t, a_t) \approx r_t + \gamma Q_\theta(s_{t+1}, argmax_{a'} Q_\theta(s_{t+1}, a^{'}))$. As a result, Double Deep Q Networks help us reduce the overestimation of Q values, help us train faster, and allow for more stable and consistent learning.

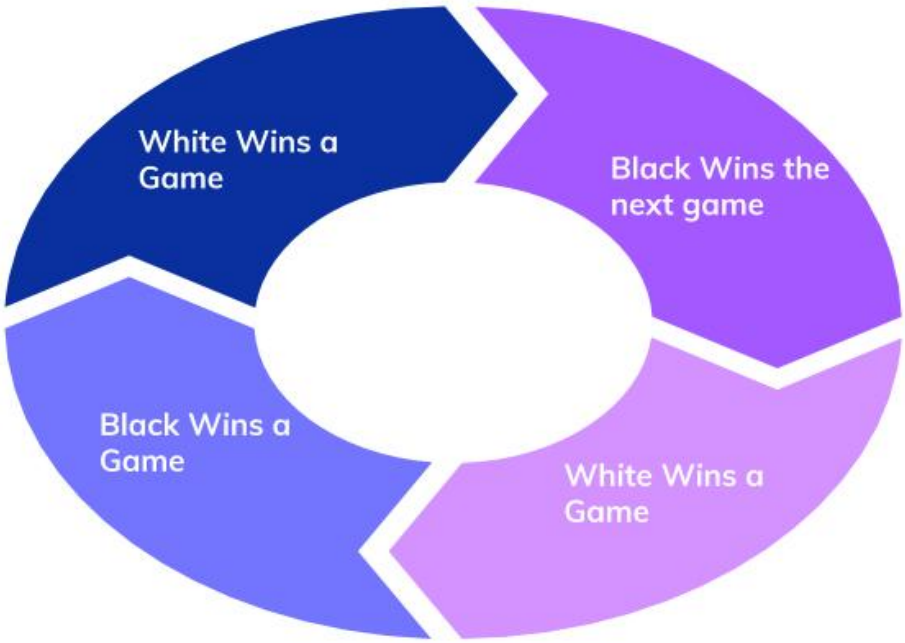
3.3 Training:

Training is perhaps the most important part of the Reinforcement Learning process.

3.3.1 Setup:

The most important question when discussing training is “How?”. We trained our program using an AI vs. AI setup. This was for two major reasons. One alternative was having the code read previous games, but that means that we must collect data from various sources and make them interpretable by our AI. This would cost us too much time and we would not be able to train the AI as effectively as we expected. Furthermore, analyzing various previous games would cause the AI to sacrifice some of the hands-on experience that it would gain by playing games by itself. The second alternative was to train it using a Player vs. AI setup. This option isn’t feasible because it requires us to play hundreds of games with the AI in just a few days. Therefore, partially through the process of elimination, we chose to create an AI vs. AI setup.

Another factor that we considered and ultimately led to us deciding to use an AI vs. AI setup was the fact that this form of training would double the results. To further explain this, allow me to use the diagram below.



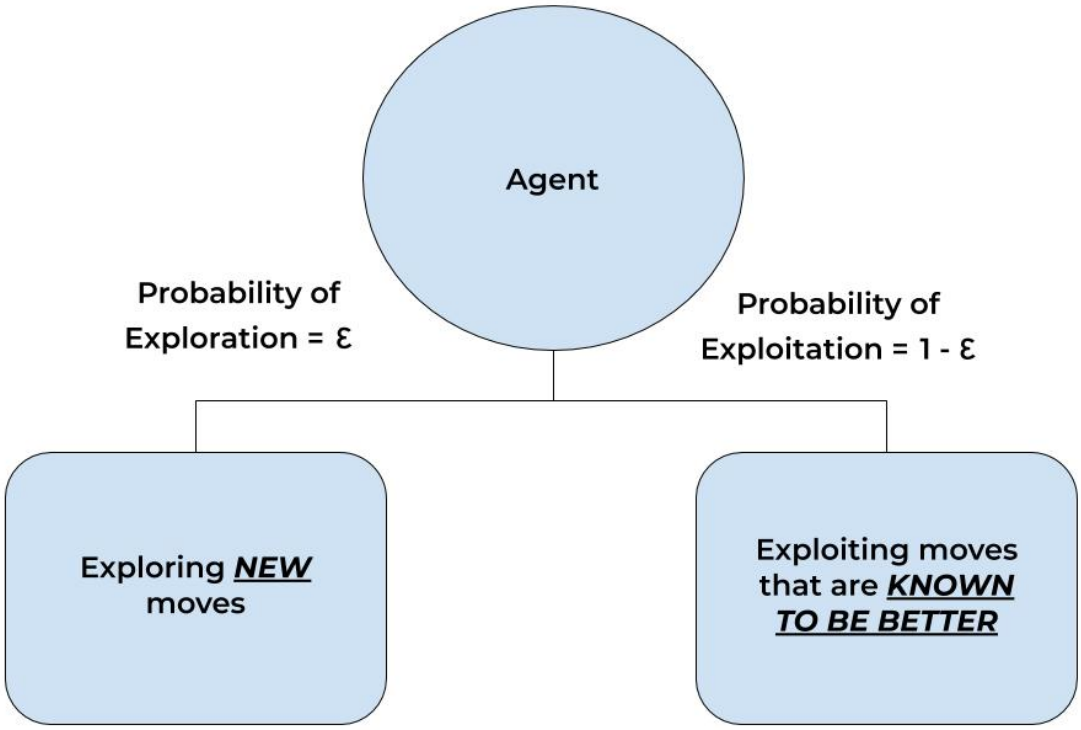
This diagram illustrates the ideal situation for this AI vs. AI setup. Starting at the dark blue arrow, we see that white won the first game. This causes white to get a reward for checkmate. Black, however, lost, giving it a point reduction. This causes black to analyze the policies and update them to create an optimal course of action. By doing this, black beats white. Now, white lost, and received a point reduction. This causes it to analyze policies and update their values to find the best one. With the best course of action in mind, white returns and beats black, starting the cycle over again. Since both the AIs are the same, the gain in knowledge of one AI contributes to that of the other. This causes the learning to double.

3.3.2 Speed Enhancements using GPU's:

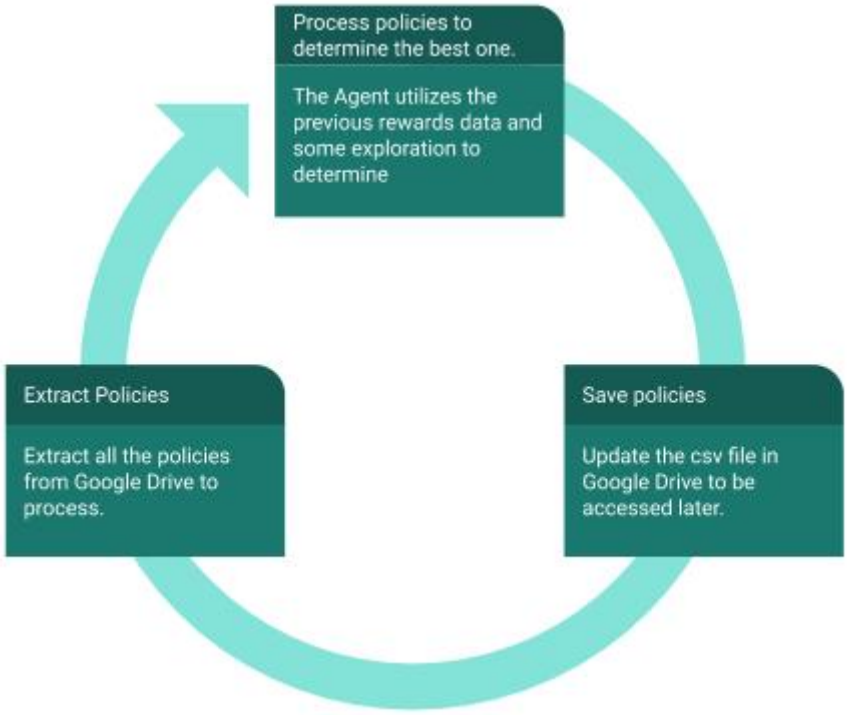
GPUs are simple hardware accelerators that can speed up computer processing at an exponential level. This allows the training to finish in one-tenth of its normal time, making it more efficient and eliminating any unnecessary time consumption. For the sake of our code, we utilized a NVidia G-Force 70 and Google Collaboratory's built in GPU, the NVidia Tesla K80.

3.3.3 Policy Management:

The main function of training in Q-Learning is to explore and update the policies so that there are many different paths explored and documented for when the AI actually plays. The exploration portion ties into the Epsilon Greedy Function. This promotes exploration while giving minimal weightage to the exploitation of moves that are known to produce wanted rewards. This allows the AI to develop its own personal move sets while finding other, perhaps better, moves.



As for the storage of policies, we chose to create a csv file, which essentially stores the policies in a spreadsheet. This file had to be made easily accessible to all of us, so we decided to save the file in Google Drive. This allowed all of us to manage the file at our own convenience. Below is a picture of how the policies are used throughout the training:



Here is a portion of the CSV File that stores the policies:

The first column stores the board, while the second stores the move. The last column stores the reward.

[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'lw', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 1, 7, 'blank', 'possible', 'None']	48.2582793
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 2, 7, 'blank', 'possible', 'None']	50.49697869
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 1, 6, 'blank', 'possible', 'None']	52.82485359
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'lw'	['lw', 0, 6, 'blank', 'possible', 'None']	49.05301046
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'lw', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 1, 6, 'blank', 'possible', 'None']	49.14420589
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'lw', 'blk'	['lw', 0, 7, 'blank', 'possible', 'None']	52.2810701
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'lw'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 0, 6, 'blank', 'possible', 'None']	49.88432878
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'lw', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 0, 7, 'blank', 'possible', 'None']	51.84847085
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'lw'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 0, 6, 'blank', 'possible', 'None']	53.86011365
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'lw', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 0, 7, 'blank', 'possible', 'None']	57.29799325
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'lw'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 1, 7, 'blank', 'possible', 'None']	58.75210792
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 2, 7, 'blank', 'possible', 'None']	60.9396864
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 1, 7, 'blank', 'possible', 'None']	69.81633467
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 2, 7, 'blank', 'possible', 'None']	72.50429892
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 3, 7, 'blank', 'possible', 'None']	74.31032194
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 2, 7, 'blank', 'possible', 'None']	74.05014575
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 1, 7, 'blank', 'possible', 'None']	87.29315629
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 2, 7, 'blank', 'possible', 'None']	92.86505988
[['blk', 'blk', 'lb', 'blk', 'blk', 'blk', 'blk', 'blk'], ['blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk', 'blk'	['lw', 1, 7, 'blank', 'possible', 'None']	98.7926169

4. Final Words and Conclusion:

4.1 Future Improvements:

Our future goals for this project are to complete and train the DDQN Code, create an app for this project, make the UI better, and hopefully make the project public for anyone to use.

4.2 Conclusion:

After 2 weeks of programming and training, we ended up with a Chess Player that used Q-Learning, because we didn't have enough time to train the DDQN, to play against anyone in Google Collab. It had, on its own, developed a preference to control the center at the beginning of the game, and if trained more, would have undoubtedly become better. Through this project, we greatly improved our knowledge of Python, different libraries, and our understanding of Reinforcement Learning.

5. Links and References

5.1 Citations:

- Bernstein, Alex, and Michael De V. Roberts. “Computer v. Chess-Player.” Scientific American, vol. 198, no. 6, 1958, pp. 96–106., doi:10.1038/scientificamerican0658-96.
- Monsters, Data. “7 Types of Artificial Neural Networks for Natural Language Processing.” *Medium*, Medium, 26 Sept. 2017, medium.com/@datamonsters/artificial-neural-networks-for-natural-language-processing-part-1-64ca9ebfa3b2.
- Smith, Mark. “Computer Chess: How the Ancient Game Revolutionised AI.” The Guardian, Guardian News and Media, 19 May 2020, www.theguardian.com/plug-into-hybrid/2020/may/19/computer-chess-how-the-ancient-game-revolutionised-ai.
- The Editors of Encyclopaedia Britannica. “Deep Blue.” Encyclopædia Britannica, Encyclopædia Britannica, Inc., 20 Apr. 2012, www.britannica.com/topic/Deep-Blue#ref1120534.
- Violante, Andre. “Simple Reinforcement Learning: Q-Learning.” *Medium*, Towards Data Science, 1 July 2019, towardsdatascience.com/simple-reinforcement-learning-q-learning-fcddc4b6fe56.

5.2 Links:

Github: https://github.com/sidharthmrao/Chess_AI_Bot

Presentation: https://docs.google.com/presentation/d/1Rs-Dt2OOBlcp-DYM2ur3KOiE_0HqbBLdsrzzHctRWHY/edit?usp=sharing

Demo: https://drive.google.com/file/d/1DDPPMCKcMKG1AZOSFKPjJ-yTaMB1VYkJ/view?usp=sharing