**Technical Internship Report**

**Student : Aymen Tlili**

**Firm : DataCamp Training & consulting , Sfax**

**Enet’com Supervising Teacher : Bassem ben Hamed**

**University Year : 2021/2022**

Summary :

I-Company Presentation ………………………………………………………………………………. 3

II-Project Description and Outlines : What is a Chess Engine ?...............................4

III-Specifics concerning the game of Chess……………………………………………………….6

IV-Reinforcement Learning Round-up……………………………………………………………..8

V-Project Implementation Details …………………………………………………………………..10

1-Data Structures………………………………………………………………………………….10

2-Project Structure Using OOP………………………………………………………………11

3-Board representation………………………………………………………………………….11

4-Search ……………………………………………………………………………………………..13

5-Evaluation…………………………………………………………………………………………14

6-End-Game TableBases……………………………………………………………………….16

VI-References………………………………………………………………………………………………..17

I-Company Presentation :

DataCamp Taining & Consulting is a Firm based in Sfax aimed at training and consulting High End profiles and providing them with the Knowledge they need to face the ever rising challenges of the Data-Driven World . The center is approved by the Tunisian Minestry of professional training and employment under the agreement N°61/377/20 and they operate as training member of the Tunisian CNFCPP ,giving their graduates access to internationally recognized certifications thanks to multiple partnerships.

This firm provides training to Students and already in the line of work experts who are in search of catching on the needed skills to Handle Big Data and make informed Decisions As well as help Drive Research and Explore new technologies that can be of great aid to companies in several domains .

The Founding Members include Dr Bassem Ben Hamed & Dr Heni Bouhamed , are experienced instructors with more than 25 years of teaching experience between them combined .

* Dr Bassem Ben Hamed:

Professor in Applied Mathematics at ENET'Com

Co-founder & Data Scientist at DataCamp Training & Consulting

Senior Instructor Huawei Artificial Intelligence

Instructor CDOSS Machine Learning & Deep Learning

* Dr Heni Bouhamed :

Phd In computer Science from the university of Rouen 2013

Chef assistant Professor at Sfax’s Faculty for Economic Sciences and Management

Certified in Big Data by Cloudera and Business Intelligence by IBM

Big Data Consultant for Elitech Consulting

For more Info: https://datacamp-training.com/a-propos-de-nous/

DataCamp Training & Consulting

**Adresse:** Avenue Bourguiba maison 178, Gremda Sfax 3062

**Email:** contact@datacamp-training.com

**Phone:** +216 95563389 +216 98550000

Nous sommes agréés par le Ministère de la Formation Professionnelle et de l’Emploi (Agrément N°61/377/20) et opérateur de formation inscrit à la CNFCPP (Centre National de Formation Continue et de Promotion Professionnelle)



II-Project Description and Outlines : What is a Chess Engine ?

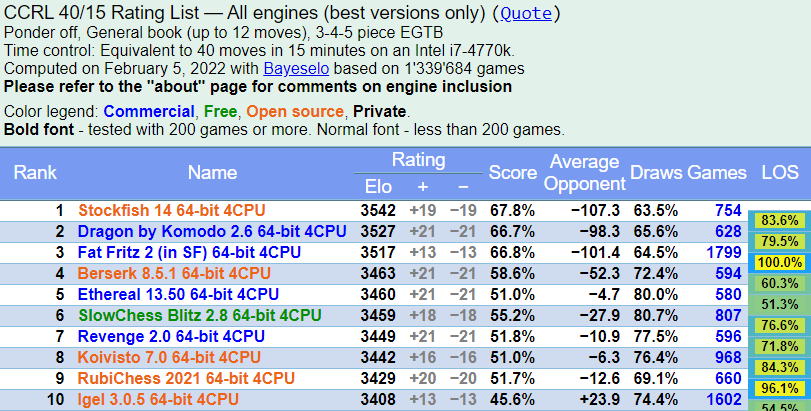
A chess Engine is a Piece of Software is capable of Artificial intelligence for solving and playing Games of Chess , usually relying on Reinforcement Learning and the process of learning to improve , it has underlying complicities and optimization problems seeing the large number of Actions to take in a given state/position .Engineers around the world compete to improve their Engines and not only prove supremacy compared to normal human standards but what a technological feat they keep making towards solving one of the highest branching game trees.



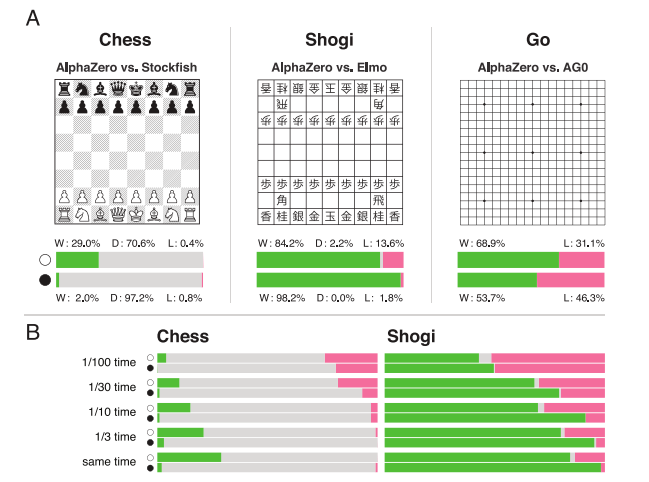
Example of a Computer Chess Championship and how the Evaluation , Depth of Search for the next best move changes, the number of Nodes/Positions considered by the game tree as well as TableBase Hits (predefined best moves for a particular position)

CCRL is a site that provides weekly refreshing rankings of the world’s top 500 Chess Engines, Here are some stats:





Now that we know what a chess engine , you might be wondering which is best .In 2016 DeepMind had released an AI called Alpha Zero capable of playing and beating Human Grand-Masters and professionals at each sport as well as other chess Engines , here’s a summary :



III-Specifics concerning the game of Chess :

1. Chess is a game with perfect Information
2. Chess is a 0-Sum Game
3. Chess has a Ton of positions

1.Chess is a game with perfect Information :

In [game theory](https://en.wikipedia.org/wiki/Game_theory), a [sequential game](https://en.wikipedia.org/wiki/Sequential_game) has **perfect information** if each player, when making any decision, is perfectly informed of all the events that have previously occurred, including the "initialization event" of the game (e.g. the starting hands of each player in a card game).

Perfect information is importantly different from [complete information](https://en.wikipedia.org/wiki/Complete_information), which implies [common knowledge](https://en.wikipedia.org/wiki/Common_knowledge_(logic)) of each player's utility functions, payoffs, strategies and "types". A game with perfect information may or may not have complete information.

This means that with enough computational power we’d be able to “solve chess” if we could list all possible positions possible

2.Chess is a 0-Sum Game:

**Zero-sum game** is a [mathematical representation](https://en.wikipedia.org/wiki/Mathematical_model) in [game theory](https://en.wikipedia.org/wiki/Game_theory) and [economic theory](https://en.wikipedia.org/wiki/Economic_theory) of a situation which involves two sides, where the result is an advantage for one side and a loss for the other

--A zero-sum game is a situation where, if one party loses, the other party wins, and the net change in wealth is zero.

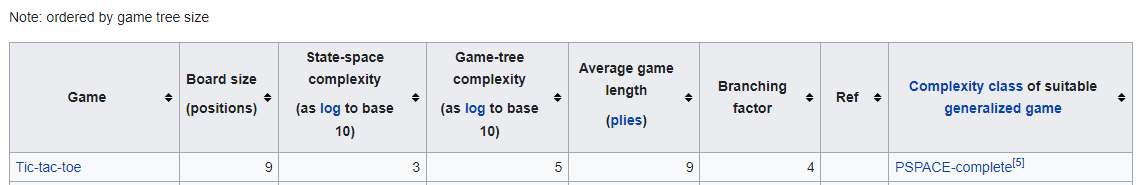
--Zero-sum games can include just two players or millions of participants.

--In financial markets, futures and options are considered zero-sum games because the contracts represent agreements between two parties and, if one investor loses, then the wealth is transferred to another investor.

--Most transactions are non-zero-sum games because the end result can be beneficial to both parties.

This property allowes us to use different variants of the MinMax algorithm called Negamax , discussed Later

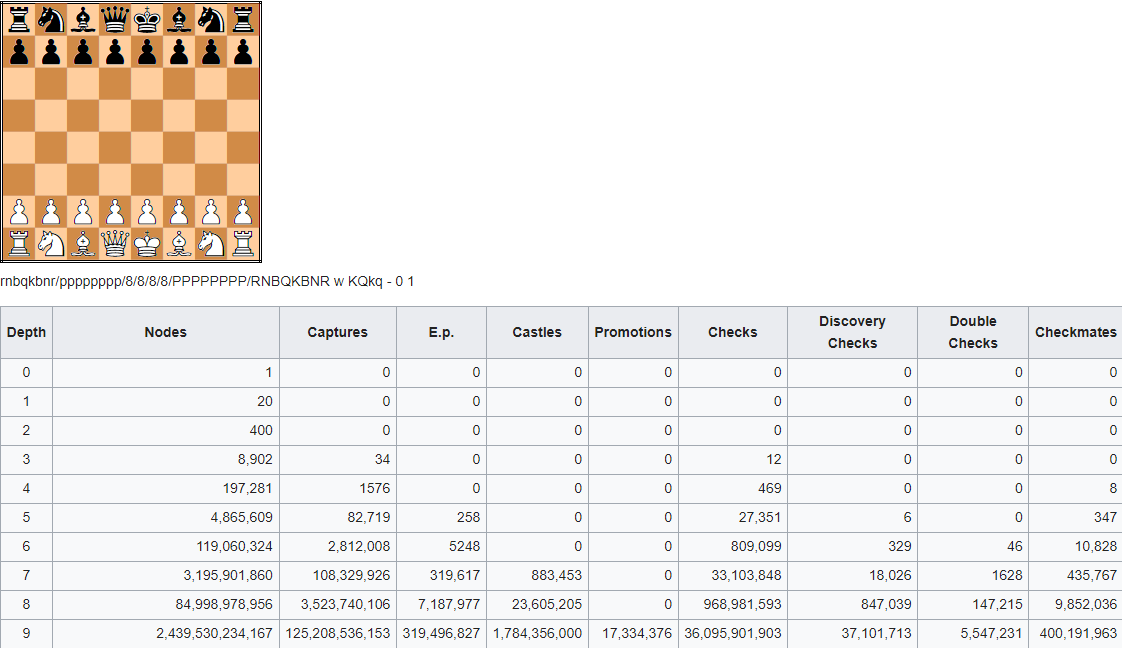
3.Chess has a Ton of positions:





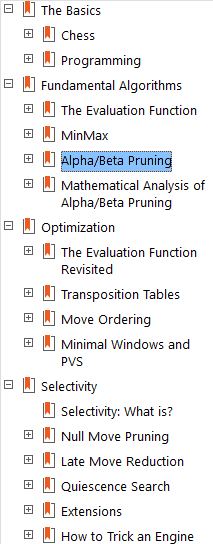


Taken from wikepedia :Game Complexity



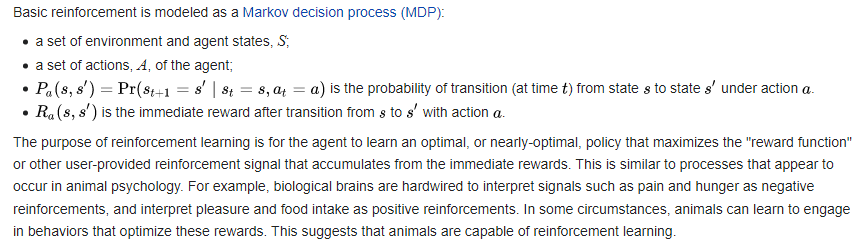
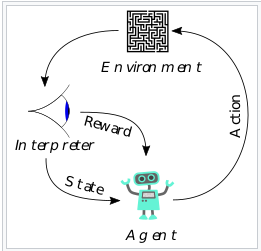
Taken from Chess Programming Wiki : Perft Result

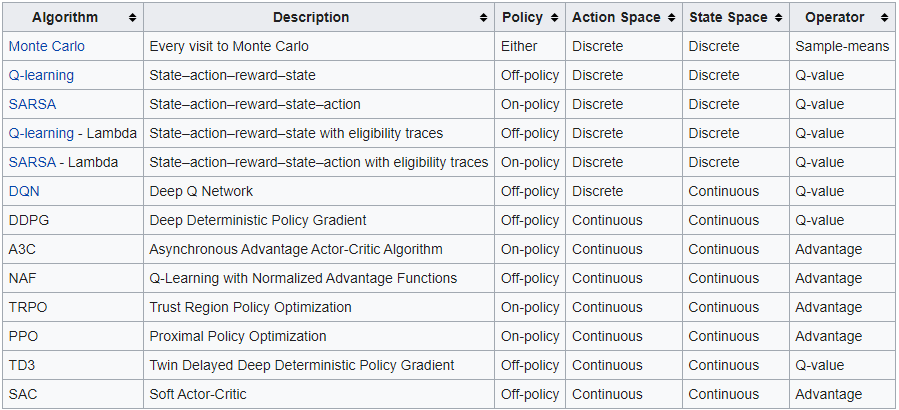
With this knowledge seeing that each position needs to be evaluated and the depth of search needs constant changing the fact that chess is a perfect information game means that solving for all positions will take computationally exponential times will not end during one’s lifetime even when using today’s cutting-edge technology . Thus the Need for Heuristics:

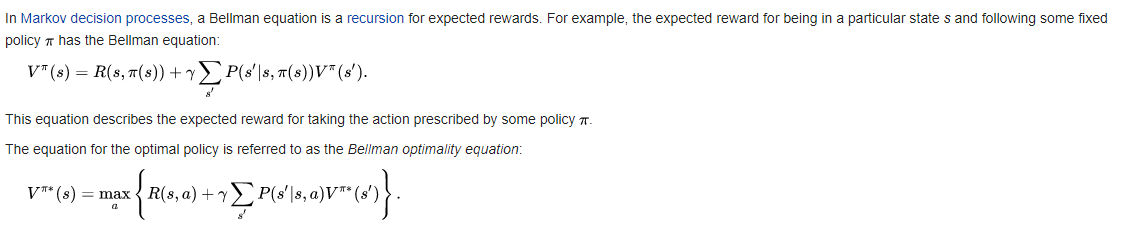


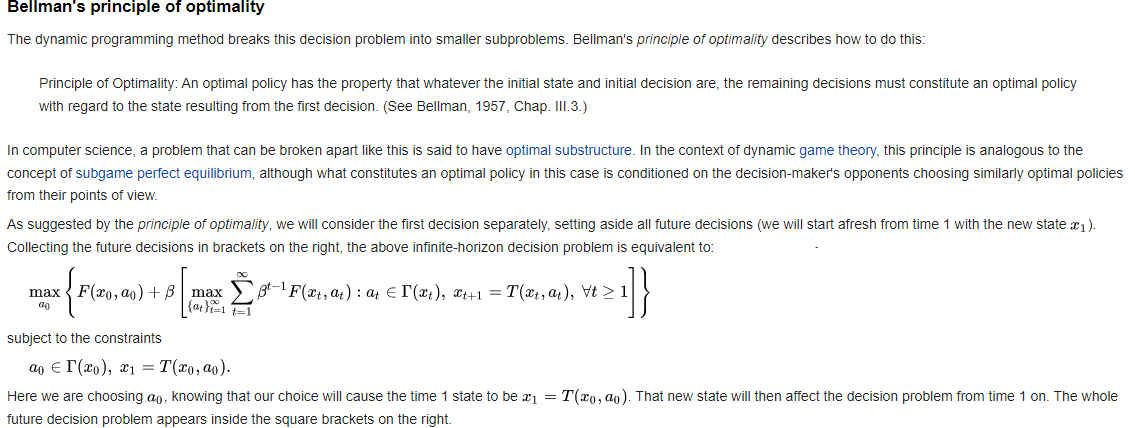
Taken from the book : Chess algorithms by Noah Caplinger

IV-Reinforcement Learning Round-up

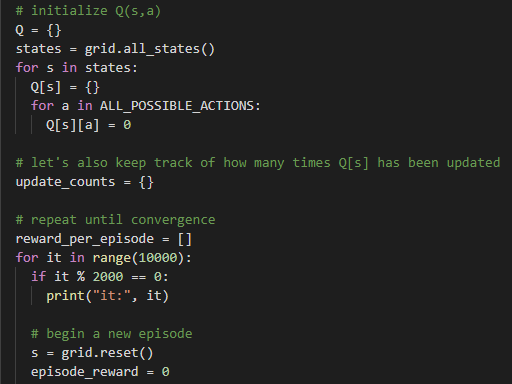
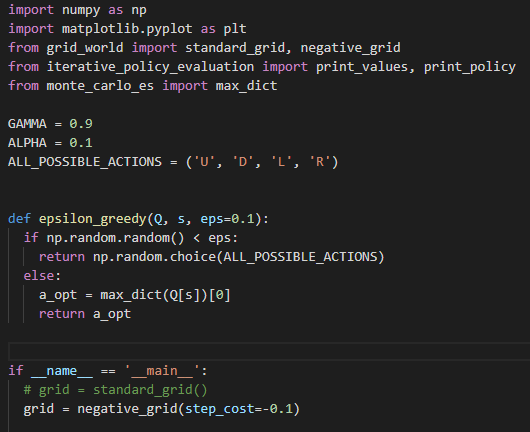


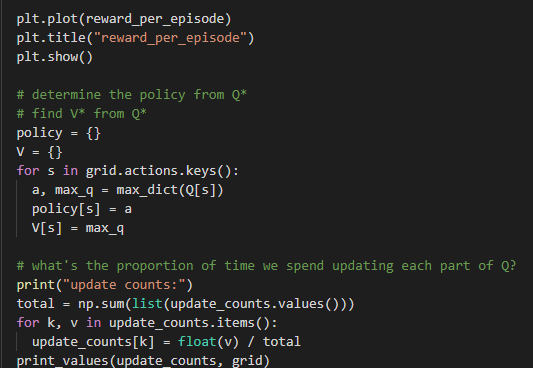
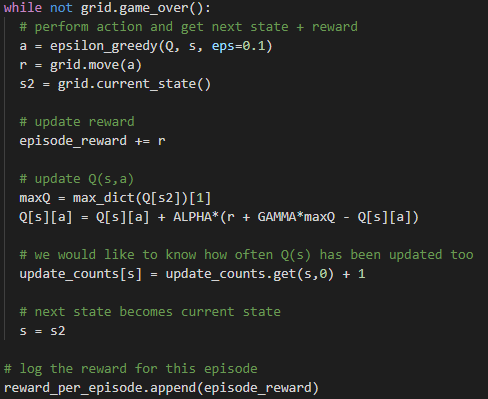






Q-Learning in Code for the Gridworld problem [a simple introductory example] :



****

V-Project Implementation Details

1-Data Structures :

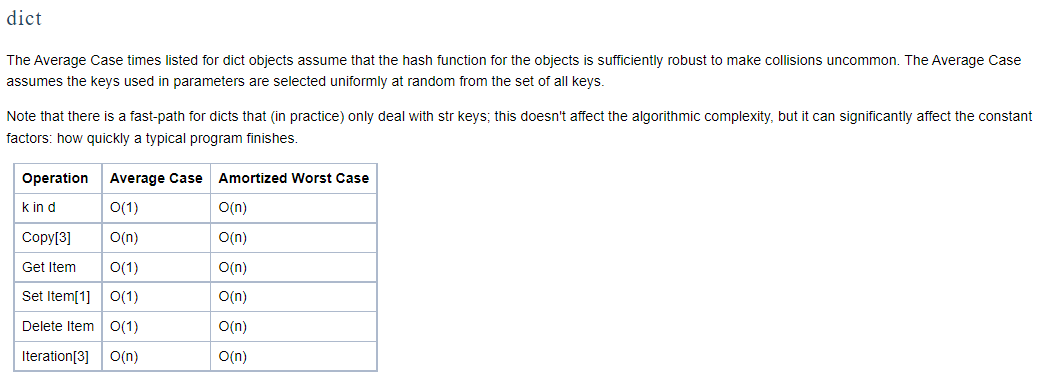
My chess engine was expected to go through multiple positions applying operations quickly and evaluating the resulting then either splitting the results into a different structure or keeping an old copy of past positions . To do this I first saught out to build a 8 by 8 Matrix using Python Numpy library optimized for linear algebra and other precise scientific calculations.

After further investigation I discovered that a more optimal to represent a chess board was using what is called a Bit Board which was usually represented by an unsigned int64 binary string .if a pice was present on a specific number that bit would be set Hot or to 1 ,if not it will remain at 0 .This was through Wes Doyle’s Youtube Channel where he further explains the uses of bitboards in other games like checkers , shogi and the likes.

This is particularly fast and nor memory expensive that that each piece would be represented by a single bit in the computer’s memory making access and treatment of multiple positions at the same time feasible .Using bit operations is also known to be fast and has constant lookup times with the addition of possible logical operations to distinguish different types of moves or desired result binary strings .

These bitboards would represent positions , we opted to make a class in Python called Piece and attach to each instance a set of attributes among which are the initial position bitboards,possible movement bitboard , incoming attacks bitboard and so on . Conveniently enough the 64 bit representation is enough to describe the entire board .

We also found that do to the incoming hashing and large number of positions we would rather sacrifice Disk space/memory as a decent tradeoff to getting better evaluation and search speeds thus the need for Dictionaries as a preferred Data type came to be , with constant lookup times and the ability to make dicts “inert” to the operating system , it was possible to handle the exponentially growing number of positions as the game went on .



Taken from Python wiki : Time complexity

2-Project Structure Using OOP:

The use of Object Oriented programming is not usually recommended when dealing with high complexity problems that may require several instance method calls which are taxing execution time wise . In fact one of the ideas behind this Project was to rewrite the entire Chess engine with optimization using Julia Programming language and relying on functional paradigm to reach high performance.

In this case OOP was used for better visibility and code clarity . such code can be re-written in Java or scala but regardless of the languages . the perceived intended functionality was :

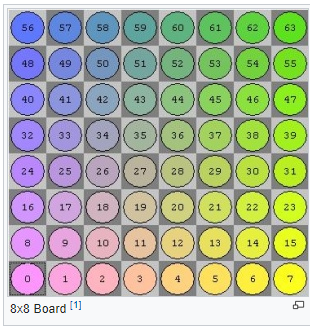
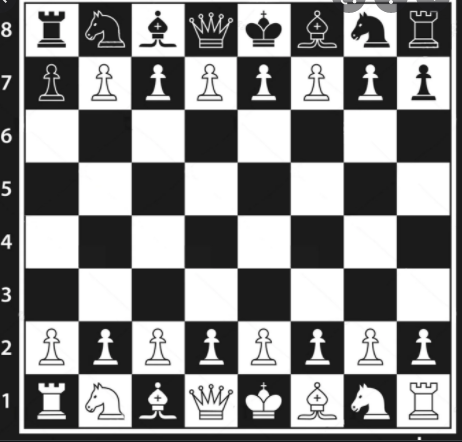
* Class Agent : would create 2 agents ready to execute Q-learning code when called
* Class Game : this would hold the 2 players , the current game position as well as starting the game and other historical events and will hold a match history and Boolean flags indicating when castling is possible , en passant moves etc
* Class Position : this class will hold multiple bitboard and will be subject to iterative deepening to create more positions , it will hold it’s own evaluation and create instance of the class Move and destroy them
* Class Move : will specify if the move is legal or pseudo-legal , from where to which square the move is going to be made and by which piece
* Class Piece : hold the piece’ value , symbol , tapered positional value , allowed moves and a bitboard of possible moves on a empty board (using logical operations this can be extended to full moves)
* Class Board : gives the representation of the current board as well as useful information like FEN and PGN
* Class UCI parser: to use Commercial issued GUI’s , it is possible to adapt the returns of our functions to a certain protocol called UCI which gives access to many feature and a better user experience overall, this transition is the job of this class

3-Board representation :

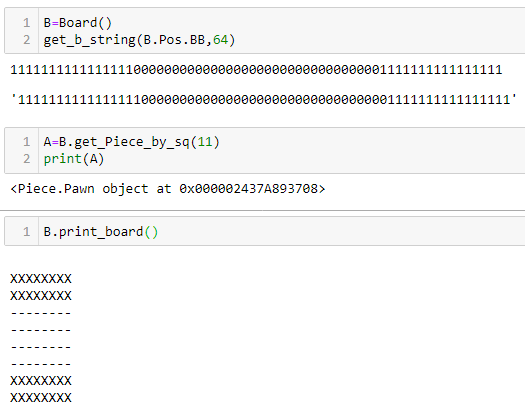
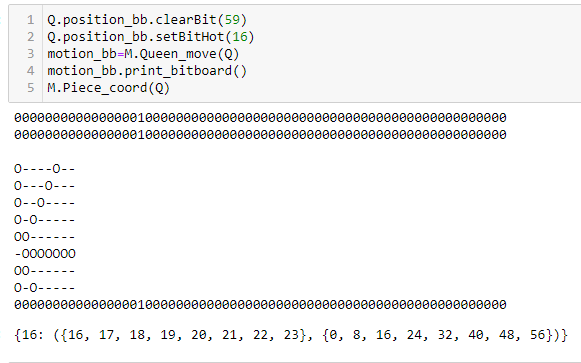
The **8x8 Board** as basic square-centric board representation is either a two-dimensional [array](https://www.chessprogramming.org/Array) of [bytes](https://www.chessprogramming.org/Byte) (or integers), containing [piece](https://www.chessprogramming.org/Pieces) and empty square codes, indexed by [file](https://www.chessprogramming.org/Files) and [rank](https://www.chessprogramming.org/Ranks) index, or more commonly a one-dimensional array indexed by a [square](https://www.chessprogramming.org/Squares) in a 0..63 range which combines rank or file indices in three consecutive bits each . Such a board representation is often used redundantly in [bitboard](https://www.chessprogramming.org/Bitboards) programs to answer the question which piece (if any) resides on a square efficiently. It has to deal with [square mapping](https://www.chessprogramming.org/Square_Mapping_Considerations) accordantly.

Alternative representation :

As a lone board representation, the 8x8 board has some efficiency issues with [move generation](https://www.chessprogramming.org/Move_Generation) related to off the board test. Therefore more common are approaches dealing with that, that is [10x12 board](https://www.chessprogramming.org/10x12_Board) with surrounding ranks and files, and [Vector Attacks](https://www.chessprogramming.org/Vector_Attacks) with its cheap test and unique square difference property with respect to [distance](https://www.chessprogramming.org/Distance) and [direction](https://www.chessprogramming.org/Direction)

Idea behind the representation

Jupyter Notebook Hand-made representation



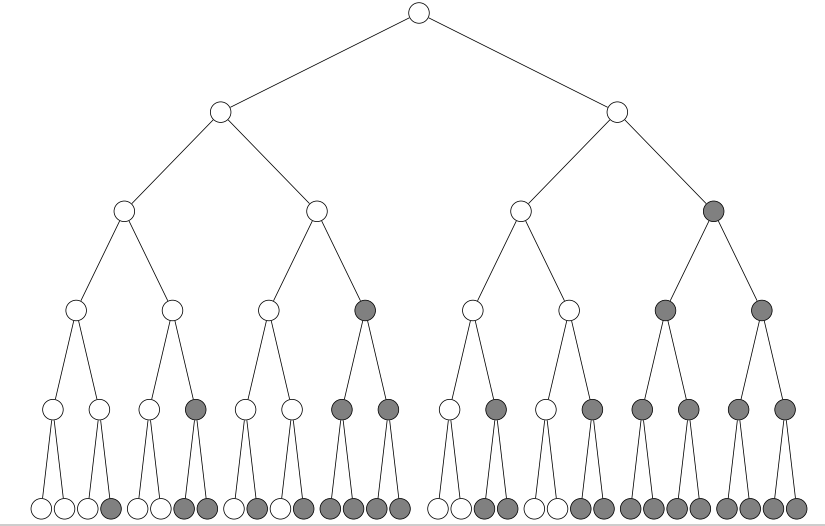
Representation provided by GUIs

4-Search :

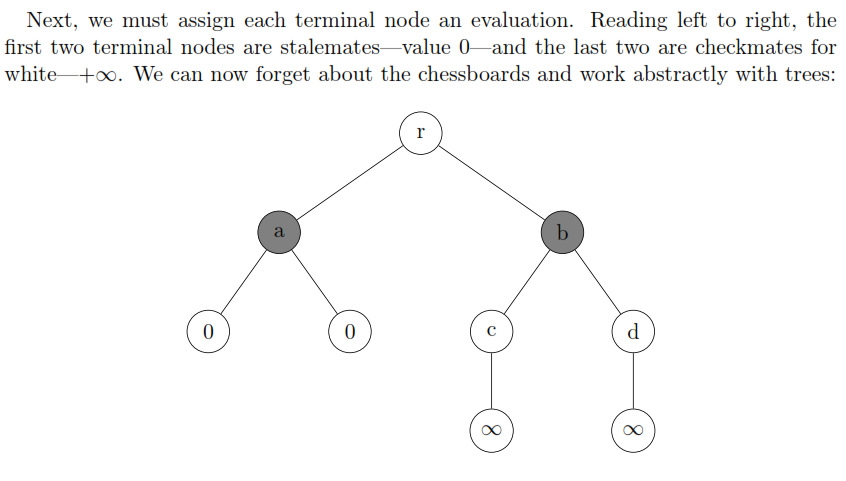
After generating a selected a number of moves we use dictionaries as what’s called a “Transposition Table” . After each episode of learning we try to avoid assessing the same position multiple times to so that we only evaluate them once and store the result for later use , accelerating the number of iterations possible per second .

The search step handles a set number of positions , searches through the game tree and prepares everything with Negamax for a smooth decision process .

Suppose our Game tree is something like this , holding a branching factor of 2 passing to each next depth :



As you can see even with a smaller branching factor than chess’s true factor this tree is huge and this just for 1 starting position and looking 4 plys in the future ; So applying the MinMax algorithm on the sorted tree yields the following in a separate example

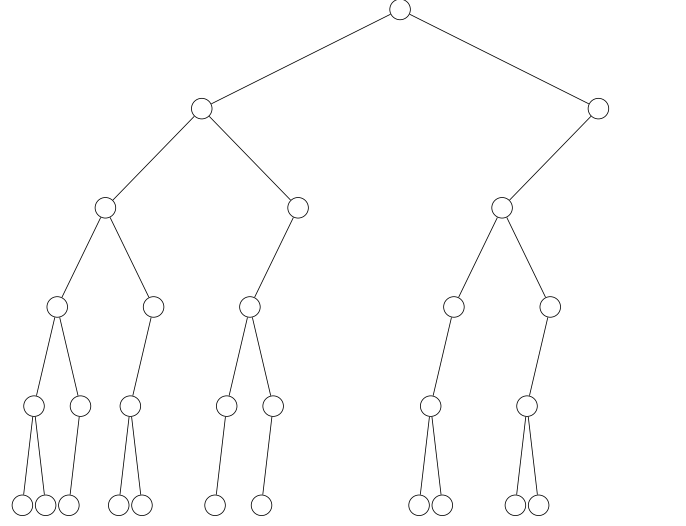


**Theorem 1 (Nodes searched in best case).** Let T be a perfectly ordered game tree of uniform branching factor b. At level l of this tree, alpha/beta pruning will examine exactly



To remedy the large Number of Nodes in the game Tree we must use Alpha Beta pruning as well as several ordering algortithms to bring about a better execution time .

**Alpha/Beta pruning : (looking back at the first example)**



Using Multiple other heuristic including the:

- Killer heuristic :finds the moves that give the attacking moves out of a set , these moves are denoted by high volatility in overall board value

-Null-move Heuristic :Suppose u were allowed to not make a move , any good player would tell us and (proof is attached in refrences) that not moving is most likely a waste of tempo or a missed opportunity but during the late game of a chess match , it is sometimes preferable to waste moves to force the opposing player into a stalemate ; this heuristic takes this supposition as the null hypothesis and performs a test which yields a p value of whether it’s better to waste tempo or not , making the agent choose one of the lower volatility moves

-Silent move heuristic : to avoid under-exploring the possible position the agent will sometimes choose a mediocre move and hope that it reaches a new optimum in the near future just to avoid being predicatble

5-Evaluation:

Each piece has it’s own material evaluation as decribed in Larry Kauffmann’s Article that assumes several hypothesis like whether the position is open or closed the presence of a pair of bishops Vs a pair of knights and what stage of the game it is a and so on . this is the value initialized in the Piece class and as the game progresses it needs to detect these features and change the value of the piece accordingly

P = 100

N = 320

B = 330

R = 500

Q = 900

K = 20000

Each type of piece has a positional evaluation . our aim is to get our pieces to positions where they are most effective and control a larger territory allowing for easier checkmating .For this purpose Tables have been Set that describe these positional advantages for both mid and late game considered all matches start off from the position in classical Chess

Examples of Mid-game of Piece-Square Tables: [PeSTO for Knight Piece]

int mg\_knight\_table[64] = {

-167, -89, -34, -49, 61, -97, -15, -107,

-73, -41, 72, 36, 23, 62, 7, -17,

-47, 60, 37, 65, 84, 129, 73, 44,

-9, 17, 19, 53, 37, 69, 18, 22,

-13, 4, 16, 13, 28, 19, 21, -8,

-23, -9, 12, 10, 19, 17, 25, -16,

-29, -53, -12, -3, -1, 18, -14, -19,

-105, -21, -58, -33, -17, -28, -19, -23,

};

int eg\_knight\_table[64] = {

-58, -38, -13, -28, -31, -27, -63, -99,

-25, -8, -25, -2, -9, -25, -24, -52,

-24, -20, 10, 9, -1, -9, -19, -41,

-17, 3, 22, 22, 22, 11, 8, -18,

-18, -6, 16, 25, 16, 17, 4, -18,

-23, -3, -1, 15, 10, -3, -20, -22,

-42, -20, -10, -5, -2, -20, -23, -44,

-29, -51, -23, -15, -22, -18, -50, -64,

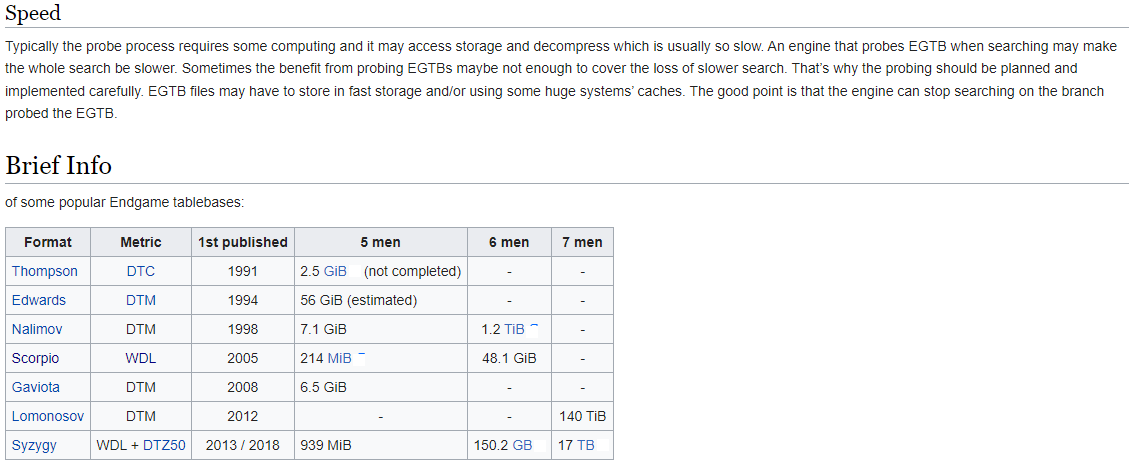
};

An improvement to the evaluation function is to make the transition from mid-game to late game dependant on number of pieces left and gradually converge towards the end-game tables .

I have accomplished this in my work by doing the transition linearly in a 11 step operation by which connecting the end value to positions that can be solved with End game TableBases

6-End-Game TableBase:

When the number of pieces left for each player is diminished to the range of 4-7 pieces with or without pawns , the recommendation is to recur to the usage of pre-calculated table bases for the best position to play in order to win , learning can be disabled as the calculation have already been done but seeing the same problem as before this requires a lot of RAM , computational time to read all the files/parse them but it is algorithmically better than doing the calculations all over again .



In my work I chose the Scorpio 6 piece table base and performed checksums to verify the integrity of the files before parsing through the dedicated software but have not completed the transfert learning step

VI-References:

Chess Programming wiki

Wikipedia : Reinforcement Learning ,0 sum game , Game complexity

# StackOverflow: [What are some good resources for writing a chess engine? [closed]](https://stackoverflow.com/questions/494721/what-are-some-good-resources-for-writing-a-chess-engine)

Bitwise Operations https://realpython.com/python-bitwise-operators/

Chess Algorithms Noah Caplinger

# Chess Programming Part II: Data Structures [www.gamedev.net/tutorials/](http://www.gamedev.net/tutorials/)

Article by GM Larry Kaufmann - www.chess.com/article/view/the-evaluation-of-material-imbalances