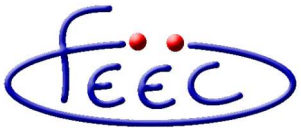

UNIVERSIDADE ESTADUAL DE CAMPINAS - UNICAMP

FACULDADE DE ENGENHARIA ELÉTRICA E COMPUTAÇÃO - FEEC

Yuki Motoyama Kuniyoshi – **RA:** 188785

Curso: Engenharia Elétrica Integral – 11

Implementação de um programa elementar de Xadrez Computacional - Relatório final

Campinas

2º Semestre / 2020

Yuki Motoyama Kuniyoshi –**RA:** 188785

Curso: Engenharia Elétrica Integral - 11

Implementação de um programa elementar de Xadrez Computacional - Relatório final

Trabalho de Fim de Curso apresentado como **requisito parcial** para obtenção do **Título de Bacharel em Engenharia Elétrica** pela Faculdade de Engenharia Elétrica e Computação da UNICAMP.

Orientador(a): Professor Romis Attux - DCA/FEEC/UNICAMP

Campinas

2º Semestre / 2020

RESUMO

A história do xadrez se concretiza pela evolução das jogadas, técnicas e principalmente pela relação com a tecnologia, nos desenvolvimentos de softwares atrelados atualmente à inteligência artificial (IA) que possibilitem destrinchar a complexidade deste jogo. Dessa forma, com o intuito de conciliar tal questão com o aprofundamento da temática de tomada de resolução perante incertezas, empregou-se de conceitos de IA e da Teoria de Jogos, atrelados aos demais conhecimentos computacionais adquiridos durante a graduação, para elaboração de um simples código para implementação de um programa capaz de jogar e buscar jogadas satisfatórias no xadrez. Este relatório final apresenta o passo a passo realizado para a execução do mesmo e, explorando as decisões escolhidas pelo algoritmo em inúmeras posições, concluiu-se que o programa, consoante seu potencial de busca em profundidade, tem a capacidade de escolher as posições as quais suas perdas seriam minimizadas, maximizando seu ganho global. Não obstante, o programa prevê espaço para melhorias futuras, visto que buscas mais apuradas requerem tempo e memória exorbitantes, as quais podem ser tratadas habilmente se inclusos algoritmos de otimização e poda na busca.

Palavras-chave: Xadrez, IA, Algoritmo Minimax, Teoria dos Jogos.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

INTRODUÇÃO

Um dos mais conhecidos jogos de tabuleiro da História, o xadrez possui uma origem ainda inexata, com seus mais antigos precursores culturais. Conforme introduzido pelo pesquisador antropólogo Celso Castro em seu artigo sobre Uma História Cultural do Xadrez, a dinâmica do jogo é muito mais do que um jogo de loteria, do acaso, pois "*ninguém ganha uma partida porque "teve sorte", nem perde porque "teve azar". Trata-se de um jogo movido apenas pelo raciocínio dos dois jogadores, que são os únicos responsáveis pelo resultado.* [...]" [1], partindo da ideia controversa de sua origem, os registros abordam o séc. VI na Índia, sendo o marco inicial do jogo conhecido na versão *Chaturanga*, traduzido como "Quatro Divisões (do Exército)" [2], até às retratadas no fim do séc. XV na Europa, com uma variante clássica mais difundida da época, mas ainda a possuir marcante semelhança com a versão dos dias atuais.

Consoante a isso, o interesse e desenvolvimento de estudos voltados à programação de rotinas que tenham a capacidade de jogar xadrez acompanharam a evolução do mesmo, desde os primeiros experimentos durante a era "Pré-Computador" com o polêmico *hoax* autômato *The Turk*, de Von Kempelen, ao recente programa *AlphaZero* de 2017, da companhia *DeepMind*, que trouxe novos patamares ao poder de cálculo das *engines* de xadrez, utilizando de redes neurais treinadas por TPUs para se aprimorar [3].

Diante disso, este projeto visa construir um algoritmo que permita ao computador realizar jogadas satisfatórias de xadrez, entrelaçando, assim, um estudo aprofundado do próprio jogo e dos materiais acerca do desenvolvimento das diferentes *engines* atualmente conhecidas, com uma breve abordagem nos tópicos relativos à Teoria dos Jogos.

JUSTIFICATIVA

A rotina implementada visa ser uma base teórica no aprofundamento e maior compreensão na criação de programas que sejam capazes de enfrentar situações que envolvam o entendimento do processo de decisão de agentes que interagem entre si, a partir da compreensão da lógica da situação em que estão envolvidos, tais como numa partida de xadrez.

OBJETIVOS

1.1 Objetivos Gerais:

Dentre os objetivos deste Trabalho de Fim de Curso (TFC), há a proposta de implementar um programa capaz de jogar xadrez a um nível decente de técnica, tendo em mente as limitações de tempo e recursos, para assim ser capaz de empregar os conhecimentos adquiridos durante a graduação, em particular no que se refere à área de Computação, num projeto de temática ímpar às comumente realizadas nas disciplinas curriculares.

1.2 Objetivos Específicos:

Realização do algoritmo de busca em árvore das jogadas possíveis a serem executadas e de uma função de avaliação do resultado final, com o intuito de se escolher a melhor jogada dentre as buscadas. Vale ressaltar que a otimização de tais códigos também será foco deste TFC, de forma a garantir um fluxo contínuo da partida e evitar longas esperas de processamento.

Por fim, difundir o tema para haver o contato do jogo de tabuleiro a todos os tipos de públicos, pois este proporciona tanto entretenimento como desenvolvimento do raciocínio cognitivo dos jogadores e espectadores, é outro dos objetivos deste projeto.

REVISÃO BIBLIOGRÁFICA

A fundamentação teórica principal deste projeto foram os estudos do matemático Claude Shannon, em específico o artigo *XXII. Programming a Computer for Playing Chess* de 1950 [4], no qual Shannon disserta a respeito da implementação de um programa capaz de jogar xadrez, suas limitações, aprimoramentos e futuras implicações. O artigo discorre as etapas a serem tomadas para criar tal algoritmo, segmentando o código em funções de forma a tornar coesa e simples a compreensão e realização das mesmas. Tais etapas foram um norte indispensável para encabeçar o projeto e programar boa parte do algoritmo. Assim como Shannon, embora "*It would seem desirable to have a number of the standard openings stored in a slow-speed memory in the machine*"[1], o interesse neste projeto é primariamente a análise de seu comportamento estratégico em posições intermediárias, não tendo sido incluída nenhuma biblioteca de aberturas.

Relativamente ao tema da função de avaliação, Marek explana:

> "*A heuristic evaluation function (usually called simply evaluation function) is that part of the program that allows comparison of positions in order to find a good move at the given position [...] it is not guaranteed that value of the function is the absolutely optimal guide"*[2]

Efetivamente, toda função de avaliação será uma aproximação da melhor estratégia que se deseja aplicar ao programa, cuja complexidade se eleva pelo fato de existirem inúmeros estilos de jogo e preferências pessoais para cada jogador, tornando o trabalho de repassar tal inclinação estratégica para o computador de forma eficiente um desafio considerável [5]. À vista disso, a função aplicada neste projeto baseou seus valores nos apresentados no algoritmo *open-source Stockfish* [6], um dos mais conhecidos e eficientes *Engines* da atualidade.

---

[1]SHANNON, Claude. **Programming a Computer for Playing Chess.** 1949. pg.15

[2] Marek Strejczek, **Some aspects of chess programming**, 2004, pg.32

METODOLOGIA E DESENVOLVIMENTO DO PROTÓTIPO

Conforme indicação do orientador, o projeto foi realizado em linguagem C, pelo fato da mesma ser de fácil manuseio no tratamento de árvores de busca. No tocante às hipóteses consideradas, disposto que o xadrez é um jogo de informação perfeita, ou seja, todas as jogadas são de conhecimento de todos os jogadores, e também um jogo de soma zero, *i.e.*, o benefício de um jogador é obrigatoriamente idêntico à desvantagem do outro, obtendo uma "soma zero" ao final, pôde-se assumir que para uma dada posição numa partida, a avaliação da mesma será vitória para as peças brancas, vitória para as peças pretas ou um empate.

Dessa forma, o programa deve analisar cada posição e avaliá-la como uma dessas opções. Contudo, uma análise absoluta até a decisão final e unânime para toda e qualquer posição não é factível com o tempo e recursos presentes na atualidade. Não obstante, é possível realizar aproximações de modo que uma função de avaliação fosse utilizada para quantificar a soberania de uma das partes para uma dada posição, fato este que será tratado mais adiante.

Tratando do código em si, os primeiros passos para sua implementação foram criar uma representação do tabuleiro de xadrez e de variáveis imprescindíveis a serem utilizadas ao longo do programa.

À vista disso, o tabuleiro fora representado por meio da matriz B[9][9] no qual o elemento [1][1] indica a posição **a1** do tabuleiro clássico, [1][2] **b1**, [1][3] **c1** e assim por diante. Ressalto aqui que a matriz declarada no programa possui dimensões 9x9 pelo fato de ter se buscado simplicidade na leitura do programa de forma que a fileira 1 do tabuleiro clássico representasse o vetor 1 da variável, não o vetor 0.

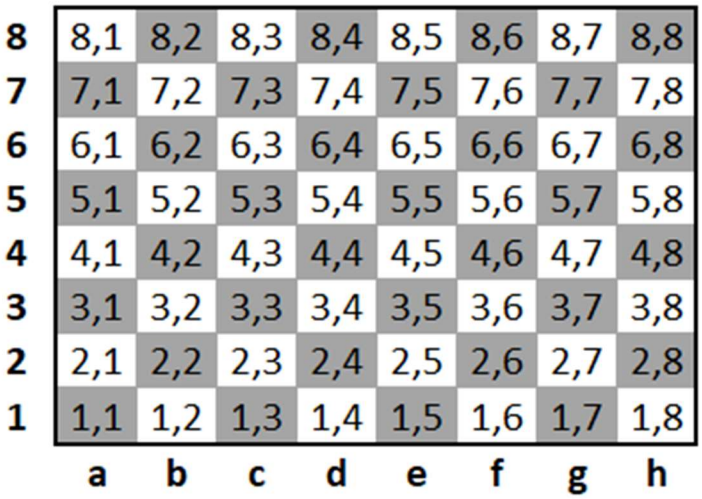

Figura 1: *Comparativo das coordenadas entre o tabuleiro clássico e a matriz controlada no programa*

Ainda no tema, tendo um vetor bidimensional para retratar o tabuleiro, uma jogada será dada por uma posição inicial, uma posição final e um número indicando a peça escolhida em caso de promoção de peões. Assim, declarou-se um *structure* de forma a armazenar esses dados:

```
typedef struct {        // VIP- Assumes "prom" as a positive number, regardless of whose piece it is
    int i1[3];          // Rank of initial position
    int i2[3];          // File of initial position
    int f1[3];          // Rank of final position
    int f2[3];          // File of final position
    int prom[3];        // Piece to be promoted to
} moves;
```

Figura 2: *Declaração da struct moves* -
Obs: As variáveis internas são vetores para que se pudesse armazenar não apenas a jogada em si, mas também a jogada-pai durante a busca em árvore.

Adicionalmente, outra *struct* semelhante à anterior também fora implementada, porém para armazenamento das jogadas possíveis em dada posição, sendo, portanto, uma lista de jogadas. A finalidade por trás de não se ter utilizado apenas uma das *struct,* afinal estas são semelhantes em formato, fora, sobretudo, para eficiência no uso de memória e maior clareza ao distinguir, ao longo do código, entre uma lista de jogadas possíveis e um movimento único armazenado. Complementarmente, o tamanho de 100

elementos fora uma escolha empírica dado que, embora estudos realizados atestem que a quantidade máxima de jogadas possíveis para uma certa posição no tabuleiro seja na ordem de 200 movimentos, testes realizados durante o projeto demonstraram que as posições alcançadas não ultrapassaram 70 movimentos. Em vista disso, uma margem foi imposta neste valor e a quantidade 100 foi preferida.

```
typedef struct {        // VIP- Assumes "prom" as a positive number, regardless of whose piece it is
    int i1[100];        // Rank of initial position
    int i2[100];        // File of initial position
    int f1[100];        // Rank of final position
    int f2[100];        // File of final positio
    int prom[100];      // Piece to be promoted to
    } lists;
```

Figura 3: *Declaração da struct lists*

Ademais, outras variáveis foram criadas, dentre elas as mais relevantes estão resumidas abaixo:

- *int turn:* de quem o turno atualmente analisado pertence (+1 = brancas -1 = pretas);
- *int CPU:* qual peça o programa estará jogando como (+1 = brancas -1 = pretas);
- *int depth:* profundidade de busca das jogadas;
- *int castle[4]*: 4 elementos para indicar a possibilidade ou não de roque de cada jogador, na ordem Brancas Roque Menor e Maior, Pretas Roque Menor e Maior (+1 = Não mais permitido 0 = Permitido);
- *int rex, rey, rix, riy*: armazenam as posições de ambos os reis, necessário para análise de xeques;
- *int move_count*: quantidade de jogadas realizadas durante a partida, sendo 1 jogada a díade 1 movimento das brancas + 1 movimento das pretas;
- int *phase*: a atual fase em que se apresenta a partida, conhecidos como *midgame* e *endgame* (0 e 1, respectivamente), indispensável para avaliação mais precisa das posições;

Em se tratando das peças, foram designadas as seguintes constantes para fins de identificação:

```
#define pawn    1
#define knight  2
#define bishop  3
#define rook    4
#define queen   5
#define king    6
```

Figura 4: *Declaração de constantes para as peças -*
Para as peças pretas, os valores são os mesmos, porém negados.

Por fim, tem-se a declaração de outra *struct* e funções suporte para a criação da árvore de busca genérica (no qual a quantidade de nós-filhos é variável). Estas parcelas do código foram compiladas numa biblioteca .h separada, *"searchtree.h"*[2] (vide anexo) para fins de clareza e organização.

```
struct GenTree {
    int eval;
    int board[9][9];
    moves move;
    struct GenTree *first;
    struct GenTree *next;
};
```

Figura 5: *Declaração da struct Tree*

Como é possível denotar da imagem acima, cada nó da árvore genérica é composto pelo seu valor de avaliação *eval*, sua respectiva jogada e posição do tabuleiro após a jogada, e ponteiros para o próximo nó-irmão **next* e para o respectivo nó-pai com **first* (apenas para o primeiro filho de cada nó-pai, pelo fato da árvore ser concatenada por meio de uma lista encadeada).

Quanto à composição do código, este comporta as seguintes rotinas principais:

- **int main ()**

Função principal do programa, esta incorpora a interface com o usuário e as chamadas principais das funções de busca e de realização do movimento no tabuleiro oficial.

- **void InitiateBoard ()**

Inicializa o tabuleiro oficial, no qual as jogadas serão de fato realizadas, e um tabuleiro auxiliar para questões de busca

- **void T0(int T [9][9], lists move, int pos, int turn)**

Esta função recebe a jogada a ser realizada por meio da posição desta na lista de jogadas, aquele que fará a jogada e o tabuleiro no qual será realizado o movimento.

- **void List_Moves (Tree* root, int turn, int depth)**

Composta por diversas sub-rotinas, esta função recebe um nó da árvore, o turno da jogada e a atual profundidade no qual a busca se encontra. A partir disso, a rotina percorre o tabuleiro repassado pelo nó e lista, inicialmente, todas as jogadas possíveis para as peças do jogador da vez por meio de um procedimento semelhante para a maioria das peças, no qual seu alcance de movimento (horizontal, vertical, diagonal) é trilhado e, em caso de se encontrar uma outra peça, este verifica a possibilidade de captura (peça adversária) ou não, e finaliza adicionando todas os movimentos possíveis para aquela peça na variável *list*. Casos específicos como o *double-step* e *en passant* dos peões e roques dos reis foram tratados de modo isolado.

Todavia, mais uma análise é necessária para seguir as regras oficiais do xadrez: uma jogada é considerada ilegal se a mesma resultar no próprio rei estar em xeque. Isto posto, com base nas variáveis *rex* e *rey* citadas anteriormente, é possível traçar as casas das quais uma peça adversária poderia estar levando o rei em xeque e deletar o movimento se o mesmo fosse ilegal.

Conjuntamente, se a jogada é válida, é realizada a avaliação (*evaluation)* da posição resultante. A despeito da função de avaliação abordada por Shannon, o mesmo sugere que *“Evaluations are based on the general structure of the*

position, the number and kind of Black and White pieces, pawn formation, mobility, etc"[3]. Assim sendo, a função heurística empregada levou em consideração os seguintes fatores:

- Atual fase da partida

Sendo um jogo de grande complexidade, o xadrez fora, de forma simplificada, dividido em 3 principais fases: abertura (*opening)*, meio-jogo (*midgame*) e finais (*endgame*), pelo fato dos objetivos de cada período serem distintos: durante a abertura, o intuito é desenvolver as peças para casas mais ativas, nas quais possam controlar e atacar o maior número de casas, e manter o rei numa casa protegida (ger. com um roque), para que no meio-jogo seja possível focar em obter vantagens posicionais e/ou atacar o rei adversário.

Durante essas primeiras fases é comum a troca de peças, de forma que o tabuleiro se apresente com uma população de peças menores, o que torna ataques diretos ao rei mais escassos e tem-se a partir de então o final, no qual um dos recursos mais valorosos são os peões, disposto que inicia-se uma "corrida" pela promoção dos peões que, por sua vez, uma vez promovidos, trazem uma vantagem absoluta, dado que muitos finais já foram cientificamente solucionados, *i.e.*, dado uma posição *Y*, existe uma lista de movimentos que leva um dos lados à vitória, independentemente das jogadas a serem realizadas pela outra parte. Em suma, nesta última fase haverá maior valoração para peões em fileiras avançadas, assim como a atividade do rei, justaposto que este se torna uma peça vital para a escolta dos peões.

Ressalta-se aqui que, para fins de simplificação, o projeto considerou apenas duas fases: o meio-jogo e o final, consoante o fato que os fundamentos aplicados nas aberturas e meio-jogos podem ser unificados numa mescla de jogabilidade razoável.

Nos demais itens a seguir, destaca-se que todas apresentam suas valorações de forma dual, contendo diferentes avaliações a depender da fase da partida, justificado pela dissertação previamente apresentada.

[3]SHANNON, Claude. **Programming a Computer for Playing Chess.** 1949. pg. 5

○ Importância de cada peça

Dentre as peças do xadrez, é comum pesar sua importância segundo a quantidade de casas as quais a peça possui controle, *i.e.*, uma torre cujo alcance engloba toda a horizontal e vertical disponível possui maior relevância aos bispos e cavalos (limitações de cor da casa e alcance minimizado, respectivamente), enquanto uma dama possui um alcance combinado de uma torre e bispo, sendo, individualmente, a peça mais valiosa. Tal valoração se apresenta no vetor *PieceValue[2][8]*

○ Posicionamento da peça

Como explicitado no tópico anterior, o controle do espaço no tabuleiro é de grande valia, o que é proporcional ao posicionamento adequado das peças em casas-chave, de forma a terem suas funções otimizadas. Assim, atribuiu-se para cada casa um valor, a depender da peça ali posta, compilada nas variáveis *WTable[2][7][64]* (para as brancas) e *BTable[2][7][64]* (para as pretas).

○ Mobilidade

Consoante ao termo Posicionamento na função, a Mobilidade indica, de forma mais precisa e exclusiva para cada posição, a mobilidade de cada peça: é produzido a Área de Mobilidade, que consiste em todas as casas do tabuleiro excetuando-se aquelas ocupadas pelos reis e rainhas e as atacadas por peões inimigos. Peões em suas casas originais também foram descartadas.

Dito isso, a função **int Mobility(x, y,piece,turn)** recebe como parâmetros a peça analisada, sua casa e o jogador atual, de modo que, após gerar a Área de Mobilidade, ela examina e retorna a quantidade de casas controladas pela dita peça dentro desta área. Salienta-se que a expressão “casas controladas” integram não apenas as casas atacadas, mas também em caso de peças como bispos e torres que podem formar uma “artilharia” em conjunto com outras (tal como o conhecido *Alekhine's gun* [7]), as casas atacadas por ambas as peças serão contabilizadas.

Por fim, o vetor *MobilityBonus[4][2][28]* fornece, para cada peça (isentando o peão e rei), a valoração da mobilidade proporcional ao número de casas retornadas pela função *Mobility*.

○ Estratégias posicionais usuais

Implementações estáticas de menor porte (independentes da fase do jogo) também foram realizadas a fim de afinar o *evaluation* das posições:

■ Bônus para roques no início da partida, para proteção do rei;

■ Ônus para peões dobrados (*i.e.* mais de um peão do mesmo jogador na mesma coluna);

■ Bônus para peões passados (*i.e.* peões sem peões adversários que poderiam impedir seu avanço), incrementado com base no quão avançado se apresenta no tabuleiro;

■ Bônus para roques no início da partida, para proteção do rei;

■ Ônus para lances com a rainha durante as primeiras jogadas (uma dica comum entre iniciantes, expor sua peça mais valiosa para ataques do adversário tende a ser prejudicial, dado que serão gastos lances para recuar a rainha ao passo que o adversário desenvolverá suas peças);

■ Roque para proteção do rei, ônus de peões dobrados e bônus de peões passados;

- **int Negamax (Tree* node,int dept, int color)**

Esta rotina faz a busca em árvore das jogadas possíveis por meio do algoritmo Minimax, resultando na melhor jogada encontrada com base na profundidade definida.

O algoritmo Minimax, cujo pseudocódigo se apresenta abaixo, teve seu teorema comprovado em 1928 pelo matemático americano John von Neumann [8], reproduzindo a concepção de minimizar a possível perda máxima. Para o

caso específico de um jogo soma-zero como xadrez, o algoritmo busca, iterativa e alternadamente, maximizar o próprio ganho e minimizar os ganhos do adversário.

A rotina *Negamax* criada possui tal nomenclatura pelo fato de se utilizar um formato mais compacto do algoritmo Minimax usual, ao utilizar-se da igualdade *max(a,b)=-min(-a,-b)* como base, possibilitando um código mais enxuto e único para tratar ambos os jogadores [9].

```
function negamax(node, depth, color) is
    if depth = 0 or node is a terminal node then
        return color × the heuristic value of node
    value := −∞
    for each child of node do
        value := max(value, -negamax(child, depth - 1, -color))
    return value
```

Figura 6: *Pseudocódigo da função Negamax*

Como é possível notar na Figura 6, a rotina percorre a árvore até que a profundidade limite seja alcançada ou se alcance as folhas (nós-filhos finais) da árvore, e retorna o maior valor encontrado, de forma a maximizar este valor para o próprio turno e minimizar para o do adversário.

RESULTADOS E DISCUSSÃO

Com base no exposto, o programa foi implementado e testado para inúmeras posições, de forma a compreender e interpretar os resultados das buscas. Durante esta fase de testes, análises em profundidades mais elevadas foram aplicadas, porém a demanda por tempo e memória por parte do programa em determinadas posições de grande atividade e possibilidades, principalmente durante o *midgame*, mostrou-se demasiado elevada mesmo empregando diferentes métodos de otimização na busca, tais como a poda Alpha-Beta [10] e o PVS (*Principal Variation Search*) [11].

Assim, separou-se algumas partidas para análise, com o programa implementado atuando como brancas contra a *Engine* disponível no site *chess.com*

> **1.** e4 c5 **2.** Qh5 d6 **3.** Nf3 Bd7 **4.** Qd5 h6 **5.** Qxb7 Bc6 **6.** Bb5 Nf6 **7.** Qxa8 Qc7 **8.** Bxc6+ Kd8 **9.** Bb7 g5 **10.** Nxg5 a5 **11.** Nxf7+ Kd7 **12.** Nxh8 e5 **13.** Nf7 Nh7 **14.** Nxh6 Ke6 **15.** Nf5 Qd8 **16.** Nxd6 Bh6 **17.** Nf5 Bg7 **18.** Nxg7+ Ke7 **19.** Nf5+ Kf6 **20.** Kf1 c4 **21.** Rg1 Qc7 **22.** d4 Nf8 **23.** dxe5+ Kxe5 **24.** Bd2 Qd8 **25.** b4 Qe8 **26.** Qxa5+ Kf6 **27.**Qa8 Ng6 **28.** c3 Ne5 **29.** Bd5 Qc8 **30.** Qa7 Nbd7 **31.** Nd4 Kg7 **32.** g4 Kh7 **33.** Kg2 Nxg4 **34.** Nc6 Kg7 **35.** f3 Kf8 **36.** fxg4 Kg7 **37.** h4 Kh7 **38.** Kh2 Qe8 **39.** Kg2 Qc8 **40.** Kh2 Qe8 **41.** Kg2 Qc8 **½-½**

Computando suas jogadas sob profundidade 3 contra uma *Engine* de Elo[4] 850 iniciou-se a partida com uma Defesa Siciliana (1. e4 c5) no tabuleiro, porém a partir do segundo lance vemos o reflexo da função de avaliação implementado. Diferente de outras *Engines*, como mencionado na Revisão Bibliográfica, este programa não possui uma biblioteca de aberturas, o que resulta na função de avaliação ter como foco principal de seus primeiros movimentos a busca por vantagem material, *i.e.*, buscar material inimigo não defendido, de forma a obter vantagem se o ataque não for suprimido. Por conseguinte, temos a captura em b7 e, logo em seguida, um lance interessante: Bb5, beneficiando-se de um *pin* no bispo adversário, indiretamente defendendo a sua rainha e desenvolvendo ofensivamente a peça.

[4] Sistema de classificação de performance no xadrez

A partida seguiu até outro ponto de atenção no lance 10: uma captura desnecessária por parte do programa, o que fora ocasionado no intuito de prevenir que, no caso de mais um avanço do peão para g4, o cavalo seria forçado a uma casa subótima (f4 ou g1) para os padrões de avaliação implementados (baixa mobilidade, posicionamento inadequado).

Concomitantemente com a baixa profundidade de busca, o programa verificou que haveria vantagem posicional e, em caso de não haver captura recíproca, um *fork* entre o rei e a torre seria possível, como acabou por transcorrer na partida. Quanto ao final da partida, averiguou-se que o programa, para uma profundidade de 3, não era capaz de encontrar vias de se obter vantagem de forma límpida, levando o programa a uma estagnação de cálculo, sucedendo-se, mesmo numa posição vencedora, a aceitar o empate por repetição (*Threefold Repetition*) no lance 41.

1. e4 c5 **2.** Qh5 e6 **3.** Nf3 Be7 **4.** Ne5 g6 **5.** Qg4 Qb6 **6.** Bd3 Qb4 **7.** c4 g5 **8.** Kf1 h6 **9.** Kg1 Na6 **10.** h4 gxh4 **11.** Rh3 Qb6 **12.** Qg7 Nc7 **13.** Qxh8 a5 **14.** Qxg8+ Bf8 **15.** Qxf7+ Kd8 **16.** Qxf8+ Ne8 **17.** Qxh6 a4 **18.** Rxh4 d6 **19.** Qg5+ Kc7 **20.** Ng6 a3 **21.** bxa3 Bd7 **22.** Rh8 Ra6 **23.** Ne7 Ba4 **24.** f3 Qa7 **25.** Qf4 Rb6 **26.** Qg5 Kd7 **27.** g3 Qa8 **28.** g4 Qa5 **29.** Kg2 Ng7 **30.** Qxg7 e5 **31.** Nd5+ Kc6 **32.** Rc8# **1-0**

Na partida contra a mesma *Engine* porém de Elo 1000, detalhada por completo acima, temos outra Defesa Siciliana, com desenvolvimento de peças aprazível até alcançarmos o lance 8. Kg1, um movimento não usual: sua justificativa se encontra no avanço dos peões adversários, levando a avaliação de manter a torre na coluna h para contrapor o avanço inimigo, mas que também considerou retirar o rei do ataque indireto da rainha inimiga. Adiante na partida, outro momento chave durante a partida fora o programa encontrar o lance vencedor 12. Qg7 que, ameaçando tanto a torre como o peão inimigo com xeque que, somado a uma defesa ineficaz por parte da *Engine*, resultou numa sequência de capturas de peças adversárias. A partir do lance 27, a *evaluation* denotou posições de avanço de peões como mais vantajosas e, com o lance 30. Qxg7, tem-se um mate forçado em 3 jogadas possível no tabuleiro e, estando nos limites de profundidade de busca, o programa é capaz de encontrar e finalizar a partida.

CONCLUSÃO

Diante do exposto, é possível inferir que o programa implementado possui suas limitações atreladas, em sua maioridade, à profundidade de busca nas jogadas.

Tal fato é de comum conhecimento na comunidade de pesquisa deste ramo, visto que a complexidade do jogo é de ordem exponencial e os recursos presentes para este projeto são finitos. Inobstante, existem métodos de otimização que, quando aplicados eficientemente, podem permitir a máquina a formular árvores de busca mais inteligentes e profundas num tempo hábil, tal como o uso de tabelas de transposição via tabelas *hash* [12], o qual utiliza do fato de várias posições de xadrez poderem ser alcançadas de diferentes maneiras, podendo, portanto, haver podas na busca armazenando tais posições e desconsiderando ramos previamente descartados.

A despeito disso, o algoritmo foi capaz de analisar conclusivamente e com rapidez as partidas e simulações realizadas, entregando uma jogabilidade razoável ao ponto de produzir resultados interessantes contra oponentes de nível iniciante a intermediário.

REFERÊNCIAS

[1] CASTRO, Celso. **Uma História cultural do Xadrez.** *Cadernos de Teoria da Comunicação*, 1994, Vol.1, n.2, p. 3-12.

[2] REMUS. **The Origin Of Chess and the Silk Road.** *Los Altos, Califórnia, US*. Disponível:http://www.silkroadfoundation.org/newsletter/volumeonenumberone/origin.html. Acesso em 08/08/2020.

[3] SILVER, David *et al.* **A general reinforcement learning algorithm that masters chess, shogi and Go through self-play.** 2018.Disponível:https://science.sciencemag.org/content/362/6419/1140. Acesso em 08/08/2020.

[4] STREJCZEK, Marek. **Some Aspects of Chess Programming.** 2004, Electrical and Electronic College, Department of Computer Science. Alemanha.

[5] SHANNON, Claude. **Programming a Computer for Playing Chess.** 1949. *Philosophical Magazine*, Ser. 7, Vol. 41, n. 314.

[6] VANDEVONDELE, Joost; NICOLET, Stéphane. **Stockfish Evaluation Guide**. 2020. Disponível:https://hxim.github.io/Stockfish-Evaluation-Guide/. Acesso em 01/11/2020.

[7] SERPER, Gregory. **The Power Of Alekhine's Gun.** 2016. Disponível: https://www.chess.com/article/view/the-alekhines-gun. Acesso em 05/12/2020.

[8] NEUMANN, John Von; MORGENSTERN, Oskar. **Theory of Games and Economic Behavior**. 1944. *Princeton University Press*.

[9] SMITH, Warren. **Fixed Point for Negamaxing Probability Distributions on Regular Trees.** 1992. *NEC Research Institute*.

[10] KOZIOL, Pawe *et al.* **Alpha-Beta.** 2020. Disponível: https://www.chessprogramming.org/Alpha-Beta. Acesso em 05/09/20.

[11] KOZIOL, Pawe *et al.* **Principal Variation Search.** 2020. Disponível:https://www.chessprogramming.org/Principal_Variation_Search. Acesso em 05/09/20.

[12] ZOBRIST, Albert. ***A New Hashing Method with Application for Game Playing***. 1970. *Technical Report #88, Computer Science Department, The University of Wisconsin, Madison, WI, USA.*

GLOSSÁRIO

Engine: Programa de computador capaz de jogar xadrez.

Evaluation: Análogo à função de avaliação.

Fork: Tática no xadrez no qual uma peça ataca duas ou mais peças adversárias, não sendo possível defender ambas.

Open-source: Software de computador com o seu código fonte disponibilizado e licenciado publicamente para ser modificado ou distribuído.

Pin: Tática no xadrez no qual uma peça atacada é impedida de se mover pois exporia outra de maior valor ao ataque.

Structure: Em C, um tipo de declaração de dados que define uma lista agrupada de variáveis sob um nome em um bloco de memória.

Tabela Hash: Estrutura de dados que associa chaves de pesquisa a valores.

TPU: Tensor Processing Unit, circuito integrado específico do aplicativo do acelerador de IA desenvolvido pelo Google.

ANEXOS

1. searchtree.h

```
// General Tree Functions
#include <stdlib.h>
#include <stdio.h>

typedef struct {            // VIP- Assumes "prom" as a positive number, regardless of whose
piece it is
        int i1[100];                // Rank of initial position
        int i2[100];                // File of initial position
        int f1[100];                // Rank of final position
        int f2[100];                // File of final positio
        int prom[100];              // Piece to be promoted to
        } lists;

                        typedef struct {            // VIP- Assumes "prom" as a
                        positive number, regardless of whose piece it is
                        int i1[3];                  // Rank of initial position
                        int i2[3];                  // File of initial position
                        int f1[3];                  // Rank of final position
                        int f2[3];                  // File of final position
                        int prom[3];                // Piece to be promoted to
                        } moves;

                        struct GenTree {
                         int eval;
                         int board[9][9];
                         moves move;
                         struct GenTree *first;
                         struct GenTree *next;
                        };

                        typedef struct GenTree Tree;

                        Tree* create (int val, int b[9][9], lists mov,int pos);
                        void insert (Tree* a, Tree* sa);
                        void print (Tree* a);
                        int searchval (Tree* a, int val);
                        void freetree (Tree* a);

                        Tree* create (int val, int b[9][9], lists mov,int pos){
                         Tree *a =(Tree *) malloc(sizeof(Tree));
                         a->eval = val;
                         memcpy(a->board,b,81*sizeof(int));                 //Board will
                        have the position after the move was played in the root
                        board
                         a->move.i1[0]=mov.i1[pos];
                         a->move.i2[0]=mov.i2[pos];
                         a->move.f1[0]=mov.f1[pos];
                         a->move.f2[0]=mov.f2[pos];
                         a->move.prom[0]=mov.prom[pos];
                                 a->first = NULL;
                                 a->next = NULL;
                                 return a;
                        }

                        void insert (Tree* a, Tree* sa) {
```

```
sa->next = a->first;
a->first = sa;
}

void print (Tree* a){

        Tree* p;
        printf("%d\n",a->eval);
        for (p=a->first; p!=NULL; p=p->next)
        print(p);
}

int searchval (Tree* a, int val){
        Tree* p;
        if (a->eval==val)
                return 1;
        else {
                for (p=a->first; p!=NULL; p=p->next) {
                        if (searchval(p,val))
                                return 1;
                }
}
        return 0;
}
void freetree (Tree* a){
        Tree* p = a->first;
        while (p!=NULL) {
                Tree* t = p->next;
                free(p);
                p = t;
        }
        free(a);
}
```

2. main.c

```
/*
Trabalho Final de Curso apresentado como requisito parcial
para obtenção do título de Bacharel em Engenharia Elétrica
pela Faculdade de Engenharia Elétrica e Computação
da Universidade Estadual de Campinas.

"Implementacao de um Programa Elementar de Xadrez Computacional"
Aluno: Yuki Motoyama Kuniyoshi          RA: 188785
Orientador: Prof. Romis Attux - DCA/FEEC/UNICAMP

*/


#include <stdio.h>
#include <stdlib.h>
#include "searchtree.h"

#define pawn     1
#define knight   2
#define bishop   3
#define rook     4
#define queen    5
#define king     6
#define infinite 100000
/*
```

```
        Rank x
                -4, -2, -3, -5, -6, -3, -2, -4;                         8
                -1, -1, -1, -1, -1, -1, -1, -1;                         7
                 0,  0,  0,  0,  0,  0,  0,  0;                                 6
                 0,  0,  0,  0,  0,  0,  0,  0;                                 5
                 0,  0,  0,  0,  0,  0,  0,  0;                                 4
                 0,  0,  0,  0,  0,  0,  0,  0;                                 3
                 1,  1,  1,  1,  1,  1,  1,  1;                                 2
                 4,  2,  3,  5,  6,  3,  2,  4                                  1

File y   1  2  3  4  5  6  7  8
*/

// White Positioning Evaluation Table- Dimensions:          Phase, Piece, Square
        int WTable[2][7][64] = {
        {
        //Midgame Values
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        //Pawn
         0,             0,              0,              0,              0,
        0,              0,              0,              //a1...a8
         3,             3,              10,             19,             16,
        19,             7,              -5,
        -9,             -15,    11,             15,             32,             22,
                5,              -22,
        -4,             -23,    6,              20,             40,             17,
                4,              -8,
        13,             0,              -13,    1,              11,             -2,
                -13,    5,
         5,             -12,    -7,             22,             -8,             -5,
                -15,    -8,
        -7,             7,              -3,             -13,    5,              -
16,     10,             -8,
         0,             0,              0,              0,              0,
        0,              0,              0,

        //Knight
        -175 , -92,     -74,    -73,    -73 ,   -74,    -92,    -175,
         -77 , -41,     -27,    -15,   -15 ,    -27,    -41,    -77,
         -61 , -17,      6,      12,     12 ,     6,   -17,      -61,
         -35 ,   8,     40,     49,     49 ,   40,      8,      -35,
         -34 ,  13,     44,     51,     51 ,    44,     13,     -34,
          -9 ,  22,     58,     53,      53,    58,     22,     -9,
         -67 , -27,      4,      37,     37 ,    4,      -27,     -67,
        -201 , -83,     -56,    -26,    -26 ,   -56,    -83,    -201,
        //Bishop
        -53,    -5,             -8,     -12,    -12,    -8,             -5,
        -53,
        -15,    8,              19,     4,              4,              19,
        8,              -15,
        -7,             21,             -5,     17,             17,             -5,
                21,             -7,
        -5,             11,             25,     17,             17,             25,
                11,             -5,
        -12,    29,             22,     31,             31,             22,
        29,             -12,
        -16,    6,              1,      11,             11,             1,
        6,              -16,
        -17,    -14,    5,      0,              0,              5,              -
```

```
14,     -17,
        -48,    1,   -14,       -23,    -23,    -14,    1,              -48,
        //Rook
        -31,    -20,    -14,    -5,             -5,             -14,    -20,    -
31,
        -21,    -13,    -8,             6,              6,              -8,
        -13,    -21,
        -25,    -11,    -1,             3,              3,              -1,
        -11,    -25,
        -13,    -5,             -4,             -6,             -6,             -4,
                -5,             -13,
        -27,    -15,    -4,             3,              3,              -4,
        -15,    -27,
        -22,    -2,             6,              12,             12,             6,
                -2,             -22,
        -2,             12,             16,             18,             18,
        16,             12,             -2,
        -17,    -19,    -1,             9,              9,              -1,
        -19,    -17,
        //Queen
         3,             -5,       -5,            4,             4,              -5,
                -5,             3,
        -3,              5,              8,             12,             12,
        8,              5,              -3,
        -3,              6,             13,              7,              7,
        13,             6,              -3,
         4,              5,              9,              8,              8,
        9,              5,              4,
         0,             14,             12,              5,              5,
        12,             14,             0,
        -4,             10,              6,              8,              8,
        6,              10,             -4,
        -5,              6,             10,              8,              8,
        10,             6,              -5,
        -2,             -2,              1,             -2,             -2,
        1,              -2,             -2,
        //King
        271,    327,    271,    198,    198,   271,   327,      271,
        278,    303,    234,    179,    179,    234,    303,    278,
        195,    258,    169,    120,    120,    169,    258,    195,
        164,    190,    138,    98,        98,  138,    190,    164,
        154,    179,    105,    70,        70,  105,    179,    154,
        123,    145,    81,        31,     31,  81,        145, 123,
        88,        120, 65,        33,     33,  65,     120,    88,
        59,         89, 45,        -1,     -1,  45,    89,      59
        },
        {
        //Endgame Values
        0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
        //Pawn
        0,              0,              0,              0,              0,
        0,              0,              0,      //a1...a8
        -10,    -6,             10,             0,              14,             7,
                -5,             -19,
        -9,             -10,    -10,    4,              4,              3,
        -6,             -4,
        6,              -2,             -8,             -4,             -13,    -
12,     -10,    -9,
        10,             5,              4,              -5,             -5,
```

```
-5,  14,  9,
28,  20,  21,  28,  30,
7,  6,  13,
0,  -11,  12,  21,  25,  19,
4,  7,
0,  0,  0,  0,  0,
0,  0,  0,

//Knight
-96, -65, -49, -21, -96, -65, -49, -21,
-67, -54, -18,  8, -67, -54, -18,  8,
-40, -27,  -8,  29, -40, -27,  -8,  29,
-35,  -2,  13,  28, -35,  -2,  13,  28,
-45, -16,  9,  39, -45, -16,  9,  39,
-51, -44, -16,  17, -51, -44, -16,  17,
-69, -50, -51,  12, -69, -50, -51,  12,
-100, -88, -56, -17, -100, -88, -56, -1,
//bishop
-57,  -30,  -37,  -12,  -12,  -37,  -30,  -57,
-37,  -13,  -17,  1,  1,  -17,  -13,  -
37,
-16,  -1,  -2,  10,  10,  -2,
-1,  -16,
-20,  -6,  0,  17,  17,  0,
-6,  -20,
-17,  -1,  -14,  15,  15,  -14,  -1,
-17,
-30,  6,  4,  6,  6,  4,
6,  -30,
-31,  -20,  -1,  1,  1,  -1,
-20,  -31,
-46,  -42,  -37,  -24,  -24,  -37,  -42,  -46,
//Rook
-9,  -13,  -10,  -9,  -9,  -10,  -
13,  -9,
-12,  -9,  -1,  -2,  -2,  -1,
-9,  -12,
6,  -8,  -2,  -6,  -6,
-2,  -8,  6,
-6,  1,  -9,  7,  7,
-9,  1,  -6,
-5,  8,  7,  -6,  -6,
7,  8,  -5,
6,  1,  -7,  10,  10,
-7,  1,  6,
4,  5,  20,  -5,  -5,
20,  5,  4,
18,  0,  19,  13,  13,
19,  0,  18,
//Queen
-69,  -57,  -47,  -26,  -26,  -47,  -57,  -69,
-55,  -31,  -22,  -4,  -4,  -22,  -31,  -55,
-39,  -18,  -9,  3,  3,  -9,  -18,  -39,
-23,  -3,  13,  24,  24,  13,  -3,  -23,
-29,  -6,  9,  21,  21,  9,  -6,  -29,
-38,  -18,  -12,  1,  1,  -12,  -18,  -38,
-50,  -27,  -24,  -8,  -8,  -24,  -27,  -50,
-75,  -52,  -43,  -36,  -36,  -43,  -52,  -75,
//King
1,  45,  85,  76,  76,  85,  45,  1,
```

```
	53,		100,	133,	135,	135,	133,	100,	53,
	88,		130,	169,	175,	175,	169,	130,	88,
	103,	156,	172,	172,	172,	172,	156,	103,
	96,		166,	199,	199,	199,	199,	166,	96,
	92,		175,	184,	191,	191,	184,	175,	92,
	47,		121,	116,	131,	131,	116,	121,	47,
	11,		59,	73,	78,	78,	73,	59,	11

	}
	};

// Black Positioning Evaluation Table- Dimensions:		Phase, Piece, Square
	int BTable[2][7][64] = {
{
//Midgame Values
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
//Pawn
 0,		0,	0,	0,	0,	0,	0,	0,	//a1...a8
-7,		7,	-3,	-13,	5,	-16,	10,	-8,
 5,	-12,	-7,	22,	-8,	-5,	-15,	-8,
13,	0,	-13,	1,	11,	-2,	-13,	5,
-4,	-23,	6,	20,	40,	17,	4,	-8,
-9,	-15,	11,	15,	32,	22,	5,	-22,
3,		3,	10,	19,	16,	19,	7,	-5,
0,		0,	0,	0,	0,	0,	0,	0,

//Knight
-201,	-83,	-56,	-26,	-26,	-56,	-83,	-201,
 -67,	-27,	4,	37,	37,	4,	-27,	-
67,
  -9,	22,	58,	53,	53,	58,	22,	-
9,
 -34,	13,	44,	51,	51,	44,	13,	-
34,
 -35,	8,	40,	49,	49,	40,	8,	-
35,
 -61,	-17,	6,	12,	12,	6,	-17,	-
61,
 -77,	-41,	-27,	-15,	-15,	-27,	-41,	-77,
-175,	-92,	-74,	-73,	-73,	-74,	-92,	-175,

//Bishop
-48,	1,		-14,	-23,	-23,	-14,	1,	-48,
-17,	-14,	5,		0,		0,		5,	-
14,	-17,
-16,	6,		1,		11,		11,		1,
	6,		-16,
-12,	29,		22,		31,		31,		22,
	29,		-12,
-5,		11,		25,		17,		17,		25,
		11,		-5,
-7,		21,		-5,		17,		17,		-5,
		21,		-7,
-15,	8,		19,		4,		4,		19,
	8,		-15,
-53,	-5,		-8,		-12,	-12,	-8,		-5,
	-53,
//Rook
-17,	-19,	-1,		9,		9,		-1,		-
```

```
19,    -17,
-2,          12,          16,          18,          18,          16,
             12,          -2,
-22,   -2,          6,           12,          12,          6,
       -2,          -22,
-27,   -15,  -4,          3,           3,           -4,          -
15,    -27,
-13,   -5,          -4,          -6,          -6,          -4,
       -5,          -13,
-25,   -11,  -1,          3,           3,           -1,          -
11,    -25,
-21,   -13,  -8,          6,           6,           -8,          -
13,    -21,
-31,   -20,  -14,  -5,          -5,          -14,  -20,  -31,
//Queen
-2,          -2,          1,           -2,          -2,          1,
             -2,          -2,
-5,          6,           10,          8,           8,           10,
             6,           -5,
-4,          10,          6,           8,           8,           6,
             10,          -4,
 0,          14,          12,          5,           5,           12,
             14,          0,
 4,          5,           9,           8,           8,           9,
             5,           4,
-3,       6,        13,          7,           7,           13,
       6,           -3,
-3,          5,           8,           12,          12,          8,
             5,           -3,
 3,          -5,          -5,          4,           4,           -5,
             -5,          3,
//King
59,       89,  45,      -1,      -1,  45,    89,      59,
88,      120,  65,      33,      33,  65,    120,     88,
123,   145,    81,      31,      31,  81,       145,  123,
154,   179,    105,  70,         70,  105,    179,    154,
164,   190,    138,  98,         98,  138,    190,    164,
195,   258,    169,  120,   120,     169,    258,    195,
278,   303,    234,  179,   179,     234,    303,    278,
271,   327,    271,  198,   198,  271,  327,         271
},
{
//Endgame Values
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
//Pawn
  0,     0,       0,       0,    0,       0,     0,       0,          //a1...a8
  0,  -11,        12,      21,   25,      19,    4,   7,
 28,  20,         21,      28,   30,      7,     6,       13,
 10,   5,         4,       -5,   -5,      -5,    14,  9,
  6,  -2, -8,      -4,   -13,  -12,       -10,  -9,
 -9,  -10, -10,     4,    4,       3,     -6,      -4,
-10,  -6,         10,      0,    14,      7,     -5,      -19,
  0,   0,  0,      0,    0,       0,     0,       0,

//Knight
-100,  -88,   -56,   -17,   -17,   -56,   -88,   -100,
-69,   -50,   -51,   12,           12,           -51,   -50,   -69,
-51,   -44,   -16,   17,           17,           -16,   -44,   -51,
-45,   -16,   9,           39,           39,           9,           -
```

```
16,     -45,
-35,    -2,     13,     28,     28,     13,
        -2,     -35,
-40,    -27,    -8,     29,     29,     -8,     -
27,     -40,
-67,    -54,    -18,    8,      8,      -18,    -54,    -67,
-96,    -65,    -49,    -21,    -21,    -49,    -65,    -96,

//bishop
-46,    -42,    -37,    -24,    -24,    -37,    -42,    -46,
-31,    -20,    -1,     1,      1,      -1,     -
20,     -31,
-30,    6,      4,      6,      6,      4,
        6,      -30,
-17,    -1,     -14,    15,     15,     -14,    -1,
        -17,
-20,    -6,     0,      17,     17,     0,
        -6,     -20,
-16,    -1,     2,      10,     10,     -2,
        -1,     -16,
-37,    -13,    -17,    1,      1,      -17,    -13,    -37,
-57,    -30,    -37,    -12,    -12,    -37,    -30,    -57,
//Rook
18,     0,      19,     13,     13,     19,
        0,      18,
4,      5,      20,     -5,     -5,     20,
        5,      4,
6,      1,      -7,     10,     10,     -7,
        1,      6,
-5,     8,      7,      -6,     -6,     7,
        8,      -5,
-6,     1,      -9,     7,      7,      -9,
        1,      -6,
6,      -8,     -2,     -6,     -6,     -2,
        -8,     6,
-12,    -9,     -1,     -2,     -2,     -1,
        -9,     -12,
-9,     -13,    -10,    -9,     -9,     -10,    -13,    -9,
//Queen
-75,    -52,    -43,    -36,    -36,    -43,    -52,    -75,
-50,    -27,    -24,    -8,     -8,     -24,    -27,    -50,
-38,    -18,    -12,    1,      1,      -12,    -18,    -38,
-29,    -6,     9,      21,     21,     9,      -6,     -29,
-23,    -3,     13,     24,     24,     13,     -3,     -23,
-39,    -18,    -9,     3,      3,      -9,     -18,    -39,
-55,    -31,    -22,    -4,     -4,     -22,    -31,    -55,
-69,    -57,    -47,    -26,    -26,    -47,    -57,    -69,
//King
11,     59,     73,     78,     78,     73,     59,     11,
47,     121,    116,    131,    131,    116,    121,    47,
92,     175,    184,    191,    191,    184,    175,    92,
96,     166,    199,    199,    199,    199,    166,    96,
103,    156,   172,    172,    172,    172,    156,    103,
88,     130,    169,    175,    175,    169,    130,    88,
53,     100,    133,    135,    135,    133,    100,    53,
1,      45,     85,     76,     76,     85,     45,     1
}
};
```

// Piece Value Table- Mid and Endgame

```
        int PieceValue[2][8]={0,124,781,825,1276,2538,10000,    //0,P,N,B,R,Q,K
                                        0,206,854,915,1389,2682,10000};

// Pawn Value Table- [0]Doubled Pawn,[1-6] Passed pawn per rank
        int Pawns[7]={-11,10,17,25,97,228,276};

// Mobility Value Table- Dimensions:       Piece, Phase, Quantity of squares
controlled in the Mobility Area
        int MobilityBonus[4][2][28] = {
         // Knight
  {
          {-62,   -53,    -12,    -3,             3,      12,     21,     28,
        37} ,
          {-79,   -57,    -31,    -17,    7,      13,     16,     21,     26}
        },

  {  // Bishop
   {-47, -20,     14,     29,     39,     53,     53,     60,     62,     69,     78,
        83,     91,     96},
         {-59,   -25,    -8,     12,     21,     40,     56,     58,     65,     69,
        78,     83,     88,     98}
        },

  {  // Rook
         {-60,   -24,    0,      3,      4,      14,             20,     30,     41,
        41,     41,     45,     57,     58,     67},
         {-82,   -15,    17,     43,     72,     100,    102,    122,    133,
        139,    153,    160,    165,    170,    175}
  },

        {  // Queen
   { -29,          -16,-8,  -8,     18,     25,     23,     37,     4,      54,     65,
        68,     69,     70,     70,     70,     71,     72,     74,     76,     90,
        104,105,106,112,114,114,119},
         { -49,   -29,-8,  17,     39,     54,     59,     73,     76,     95,     95,
        101,124,128,132,133,136,140,147,149,153,169,171,171,178,185,187,221}
  }

 };

        int depth=3, phase=0, move_count=0;
                        // Depth of Search ; (0=Midgame,1=Endgame) ; Number of
moves played
        int j,k,aux[9][9],aux1,h,aux3=0;
                // Auxiliaries

        int x=0,y=0;
                                        // Indexes for the current analyzed square
        int castle[4];
                                        // castle[4]= White Short Long 0 1 , Black
Short Long 2 3
        int turn=1,CPU;
                                        // Current turn analyzed ; Which side is the
Engine playing
        int rex,rey,rix,riy;
                                // Store kings' locations
        int B[9][9];
                                        // Offficial Board[Row][Column]
        int eval=0;
                                                // Stores position's evaluation
```

```
	int mov[4];
                                                    // Receive player's move
	lists list,aux_move,log;
                        // List of moves in the position; auxiliary ; log of all the
moves played during the match
	moves e_move;
                                                // Engine's move (move to be sent to main
when search is done)
	Tree* nodes[100000000];
                                                // Search Tree

	void T0(int T[9][9],lists move,int pos,int turn);
	int List_Moves(Tree* root, int color,int depth);
	int Negamax(Tree* node,int depth,int alpha,int beta, int turn);
	void InitiateBoard();

int main() {

int final_eval,three_fold;
char ch1;


	// Initiates Board
	InitiateBoard();

	// Player chooses which side to play
	printf("Choose your pieces: W or B\n");
	scanf("%c",&ch1);
	if(ch1=='W' || ch1=='w'){
	   CPU=-1;
		goto Player_Move;
	}
	if(ch1=='B' || ch1=='b')
		CPU=1;


	// Engine Calculation
	Calculate_Move:
	// Aux board updated and reset list of moves
	memcpy(aux,B,81*sizeof(int));
	memset(list.i1, 0, sizeof list.i1);
	memset(list.f1, 0, sizeof list.f1);
	memset(list.i2, 0, sizeof list.i2);
	memset(list.f2, 0, sizeof list.f2);
	memset(list.prom, 0, sizeof list.prom);

	nodes[aux3]=create(0,aux,list,0);
                                // Initial root
	aux3++;
	final_eval=Negamax(nodes[aux3-1],depth,-infinite,infinite,turn);    // Search
function

	// Threefold Repetition checking
	if(log.i1[move_count*2-4]==e_move.i1[1] && log.i2[move_count*2-
4]==e_move.i2[1] && log.f1[move_count*2-4]==e_move.f1[1] &&
log.f2[move_count*2-4]==e_move.f2[1])
		three_fold++;
	else
		three_fold=0;
	if(three_fold==6){
```

```
                printf("Draw by Threefold Repetition \n\n");
                getchar();
        }
        if(three_fold==5){                      //Neglect a draw if evaluation is >0,
increasing depth of search and oblige re-search
                if(final_eval>0){
                        depth--;
                        freetree(nodes[0]);
                        eval=0;
                        aux3=0;
                        goto Calculate_Move;
                }
        }

        // Make the Engine's move on the official board
        Engine_Move:

        if(e_move.i1[0]==0){                                                    //No more
possible moves for the Engine = Player has won!
                        printf("You won! Congrats :D \n");
                        getchar();
                        exit(0);
        }
        aux_move.i1[1]=e_move.i1[1];
        aux_move.i2[1]=e_move.i2[1];
        aux_move.f1[1]=e_move.f1[1];
        aux_move.f2[1]=e_move.f2[1];
        aux_move.prom[1]=e_move.prom[1];

        T0(B,aux_move,1,turn);                                                  //Oficially
making the chosen move

        log.i1[move_count*2]=e_move.i1[1];                          //Storing the move
on the Log
        log.i2[move_count*2]=e_move.i2[1];
        log.f1[move_count*2]=e_move.f1[1];
        log.f2[move_count*2]=e_move.f2[1];
        log.prom[move_count*2]=e_move.prom[1];

        freetree(nodes[0]);
        eval=0;
        aux3=0;
        turn=-turn;
        if(CPU==-1)
                move_count++;

                for(j=8;j>0;j--) {  //Print board
                        for(k=1;k<9;k++){
                                if(B[j][k]>=0)
                                        printf(" %d  ",B[j][k]);
                                else
                                        printf("%d  ",B[j][k]);
                        }
                        printf("\n");
                }

        // Retrieve and make the Player's move
        Player_Move:
                printf("\nInput your move\n");
```

```
	// Phase checking
		if(phase==0){
			for(j=8;j>0;j--) {					//Evaluate
non-pawn material
				for(k=1;k<9;k++){
					if(B[j][k]!=0 && B[j][k]!=pawn && B[j][k]!=-
pawn)

	eval+=PieceValue[phase][abs(B[j][k])];
				}
			}
			if(eval<=5200)
	//"We are in the Endgame now"
				phase=1;
			eval=0;
		}

	scanf("%d %d %d %d",&mov[0],&mov[1],&mov[2],&mov[3],&mov[4]);

	aux_move.i1[0]=mov[0];
	aux_move.i2[0]=mov[1];
	aux_move.f1[0]=mov[2];
	aux_move.f2[0]=mov[3];
	aux_move.prom[0]=mov[4];

	T0(B,aux_move,0,turn);					// Oficially
making the chosen move

	printf("\n");
	for(j=8;j>0;j--) {  // Print board
		for(k=1;k<9;k++){
			if(B[j][k]>=0)
				printf(" %d  ",B[j][k]);
			else
				printf("%d  ",B[j][k]);
		}
		printf("\n");
	}
	printf("\n");

	log.i1[move_count*2+1]=aux_move.i1[0];		// Storing the move
on the Log
	log.i2[move_count*2+1]=aux_move.i2[0];
	log.f1[move_count*2+1]=aux_move.f1[0];
	log.f2[move_count*2+1]=aux_move.f2[0];
	log.prom[move_count*2+1]=aux_move.prom[0];


	// Threefold Repetition checking
	if(log.i1[move_count*2-3]==log.i1[move_count*2+1] &&
log.i2[move_count*2-3]==log.i2[move_count*2+1] && log.f1[move_count*2-
3]==log.f1[move_count*2+1] && log.f2[move_count*2-3]==log.f2[move_count*2+1])
		three_fold++;
	else
		three_fold=0;
	if(three_fold==6){
		printf("Draw by Threefold Repetition \n\n");
		getchar();
	}
	eval=0;
```

```
	aux3=0;
	turn=-turn;
	if(CPU==1)
		move_count++;

	goto Calculate_Move;
}

/*  *************************                    T0- Move a piece
**********************  */

void T0(int T[9][9],lists move,int pos, int turn) {
	int a1,a2,b1,b2;

	a1=move.i1[pos];
	a2=move.i2[pos];
	b1=move.f1[pos];
	b2=move.f2[pos];

	if(T[a1][a2]==king*turn){                              //For legality check, must
update king location if moved
		rex=b1;
		rey=b2;
	}
	else if(T[a1][a2]== (-king*turn)){
		rix=b1;
		riy=b2;
	}

	if (turn>0){
		if( (T[a1][a2]==king) && abs(b2-a2)==2) {          //Castle?

			if((b2-a2)==2){

				T[a1][5]=0;
	//Short castle
				T[a1][8]=0;
				T[b1][7]=king;
				T[b1][6]=rook;
				castle[0]=1;
			}
			else{
				T[a1][5]=0;
	//Long castle
				T[a1][1]=0;
				T[b1][3]=king;
				T[b1][4]=rook;
				castle[1]=1;
			}
		return;
		}
		if(move.prom[pos]==0){
	//Common move?
			if( castle[0]==0||castle[1]==0){
				if(T[a1][a2] == king){
					castle[0]=1;
					castle[1]=1;
				}
				if(T[a1][a2] == rook){
					if(a2==1)
```

```
		//Long
                                        castle[1]=1;
                                if(a2==8)
		//Short
                                        castle[0]=1;
                        }
                }
                T[b1][b2]=T[a1][a2];
                T[a1][a2]=0;
        }
        else{
        //Promotion
                if(move.prom[pos]==-1){
//En passant
                        T[b1][b2]=T[a1][a2];
                        T[a1][a2]=0;
                        T[a1][b2]=0;
                }
                else{   //Promotion

                        T[b1][b2]=-move.prom[pos];
                        T[a1][a2]=0;
                }
        }
}
else {                  //Black
        if( (T[a1][a2]==-king) && abs(b2-a2)==2) {       //Castle?

                if((b2-a2)==2){

                        T[a1][5]=0;
        //Short castle
                        T[a1][8]=0;
                        T[b1][7]=-king;
                        T[b1][6]=-rook;
                        castle[2]=1;
                }
                else{
                        T[a1][5]=0;
        //Long castle
                        T[a1][1]=0;
                        T[b1][3]=-king;
                        T[b1][4]=-rook;
                        castle[3]=1;
                }
        return;
                }
        if(move.prom[pos]==0){
//Common move?
                if(castle[2]==0||castle[3]==0){
                        if(T[a1][a2] == -king){
                                castle[2]=1;
                                castle[3]=1;
                        }
                        if(T[a1][a2] == -rook){
                                if(a2==1)
        //Long
                                        castle[2]=1;
                                if(a2==8)
        //Short
```

```
                                                castle[3]=1;
                                        }
                                }
                                T[b1][b2]=T[a1][a2];
                                T[a1][a2]=0;
                        }
                        else{
                                if(move.prom[pos]==-1){
        //En passant
                                        T[b1][b2]=T[a1][a2];
                                        T[a1][a2]=0;
                                        T[a1][b2]=0;
                                }
                                else{   //Promotion

                                        T[b1][b2]=-move.prom[pos];
                                        T[a1][a2]=0;
                                }
                        }
                }
}

/* ********************************** T7- Controlling ********************************** */
int List_Moves(Tree* root,int turn,int depth){

int i=0,g=0;
int passed=1;           //Passed pawns counter
Tree* p;
int mat[9][9];
int castle_flag=0,mate_flag=1, val=1;

        memcpy(aux,root->board,81*sizeof(int));

        for(k=1;k<9;k++){                                       //Store king position
                for(h=1;h<9;h++){
                        if(aux[k][h]==king*turn){
                                rex=k;
                                rey=h;
                        }
                        if(aux[k][h]== -king*turn){
                                rix=k;
                                riy=h;
                        }
                }
        }

// List all the possible moves for the current position & Evaluate Piece Positioning
        for (x=1;x<9;x++){
                for (y=1;y<9;y++){
                        if(turn>0)
                                switch (aux[x][y]){
                                        case 0:
                                                break;
                                        case 1:
                /* ***************************   T1- List of moves for a pawn
***************************** */

                                                if( x==2 || x==7 ) {
                //Are the pawns on their original squares?
                                                        if(aux[x+turn][y]==0)      {
```

```
        if(aux[x+2*turn][y]==0){
                                                        //List Front &
Double Step
                                                        list.i1[i]=x;
                                                        list.i2[i]=y;
                                                        list.f1[i]=x+turn;
                                                        list.f2[i]=y;
                                                        i++;
                                                        list.i1[i]=x;
                                                        list.i2[i]=y;
                                                        list.f1[i]=x+2*turn;
                                                        list.f2[i]=y;
                                                        i++;
                                                        }
                                                        else{
                                                        //List Front
                                                        list.i1[i]=x;
                                                        list.i2[i]=y;
                                                        list.f1[i]=x+turn;
                                                        list.f2[i]=y;
                                                        i++;
                                                        }
                                                }
                                        }
                                        else{
                                                if(aux[x+turn][y]==0)      {
                                                        //List Front
                                                        list.i1[i]=x;
                                                        list.i2[i]=y;
                                                        list.f1[i]=x+turn;
                                                        list.f2[i]=y;

        if(aux[x+turn][y]==8)

        list.prom[i]=queen;
                                                        i++;

                                                }
                                        }
                                        if(aux[x+turn][y+turn]*turn<0){
        //Enemy piece on the diagonals?
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+turn;
                                                list.f2[i]=y+turn;
                                                i++;
                                        }
                                        if(aux[x+turn][y-turn]*turn<0){
        //Enemy piece on the diagonals?
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+turn;
                                                list.f2[i]=y-turn;
                                                i++;
                                        }
                                        if( aux[log.f1[move_count*2-
1]][log.f2[move_count*2-1]]==(-turn) && abs(log.f1[move_count*2-1]-
log.i1[move_count*2-1])==2 && (log.i2[move_count*2-1]==(y-
1)||log.i2[move_count*2-1]==(y+1)) && log.f1[move_count*2-1]==x ) {//En passant -
```

```
last move was from the enemy pawn, was a double step, it is in an adjacent collumn
from the analyzed square, the analyzed square's pawn is on the correct row to
capture
                                                    list.i1[i]=x;
                                                    list.i2[i]=y;

        list.f1[i]=log.f1[move_count*2-1]+turn;

        list.f2[i]=log.f2[move_count*2-1];
                                                    list.prom[i]=-1;
        //En passant coding, in order to delete the enemy pawn
                                                    i++;
                                                }
                                                break;

                                            case 2:

                    /* ***************************   T2- List of moves for a knight
**************************** */
                                                if((x+2<9) && (y+1<9) &&
(aux[x+2][y+1]*turn<=0)){
                                                    list.i1[i]=x;
                                                    list.i2[i]=y;
                                                    list.f1[i]=x+2;
                                                    list.f2[i]=y+1;
                                                    i++;
                                                }
                                                if((x+2<9)&&(y-1>0)&&(aux[x+2][y-
1]*turn<=0)){
                                                    list.i1[i]=x;
                                                    list.i2[i]=y;
                                                    list.f1[i]=x+2;
                                                    list.f2[i]=y-1;
                                                    i++;
                                                }
                                                if((x-2>0)&&(y+1<9)&&(aux[x-
2][y+1]*turn<=0)){
                                                    list.i1[i]=x;
                                                    list.i2[i]=y;
                                                    list.f1[i]=x-2;
                                                    list.f2[i]=y+1;
                                                    i++;
                                                }
                                                if((x-2>0)&&(y-1>0)&&(aux[x-2][y-
1]*turn<=0)){
                                                    list.i1[i]=x;
                                                    list.i2[i]=y;
                                                    list.f1[i]=x-2;
                                                    list.f2[i]=y-1;
                                                    i++;
                                                }

        if((x+1<9)&&(y+2<9)&&(aux[x+1][y+2]*turn<=0)){
                                                    list.i1[i]=x;
                                                    list.i2[i]=y;
                                                    list.f1[i]=x+1;
                                                    list.f2[i]=y+2;
                                                    i++;
                                                }
                                                if((x+1<9)&&(y-2>0)&&(aux[x+1][y-
```

```
2]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+1;
                                                list.f2[i]=y-2;
                                                i++;
                                        }
                                        if((x-1>0)&&(y+2<9)&&(aux[x-
1][y+2]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x-1;
                                                list.f2[i]=y+2;
                                                i++;
                                        }
                                        if((x-1>0)&&(y-2>0)&&(aux[x-1][y-
2]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x-1;
                                                list.f2[i]=y-2;
                                                i++;
                                        }
                                break;

                                case 3:

                        /* ***************************   T3- List of moves for a bishop
**************************** */
                                        j=x;
                                        k=y;
                                        while(j<8 && k<8){
                //Upper Right
                                                if(aux[j+1][k+1]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                                //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
                                                list.f1[i]=j+1;
                                                list.f2[i]=k+1;
                                                i++;
                                                if(aux[j+1][k+1]==0){
                        //Keep sweeping
                                                        j++;
                                                        k++;
                                                        continue;
                                                }
                                                else break;
                                //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j>1 && k<8){
                //Lower Right
                                                if(aux[j-1][k+1]*turn>0){
                //Ally piece found
                                                        break;
```

```
                                                }
                                                list.i1[i]=x;
                                        //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
                                                list.f1[i]=j-1;
                                                list.f2[i]=k+1;
                                                i++;
                                                if(aux[j-1][k+1]==0){
                                        //Keep sweeping
                                                        j--;
                                                        k++;
                                                        continue;
                                                }
                                                else break;
                                        //Enemy piece found
                                                }
                                                j=x;
                                                k=y;
                                                while(j<8 && k>1){
                                //Upper Left
                                                if(aux[j+1][k-1]*turn>0){
                                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                                        //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
                                                list.f1[i]=j+1;
                                                list.f2[i]=k-1;
                                                i++;
                                                if(aux[j+1][k-1]==0){
                                        //Keep sweeping
                                                        j++;
                                                        k--;
                                                        continue;
                                                }
                                                else break;
                                        //Enemy piece found
                                                }
                                                j=x;
                                                k=y;
                                                while(j>1 && k>1){
                                //Lower Left
                                                if(aux[j-1][k-1]*turn>0){
                                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                                        //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
                                                list.f1[i]=j-1;
                                                list.f2[i]=k-1;
                                                i++;
                                                if(aux[j-1][k-1]==0){
                                        //Keep sweeping
                                                        j--;
                                                        k--;
```

```
                                                continue;
                                        }
                                        else break;
                                //Enemy piece found
                                        }
                                break;

                                case 4:
                        /* ***************************   T4- List of moves for a rook
***************************** */

                                                j=x;
                                                k=y;
                                                while(k<8){
                                //Right
                                                        if(aux[j][k+1]*turn>0){
                        //Ally piece found
                                                                break;
                                                        }
                                                        list.i1[i]=x;
                                //Either empty or enemy piece, for both the move must be
listed
                                                        list.i2[i]=y;
                                                        list.f1[i]=j;
                                                        list.f2[i]=k+1;
                                                        i++;
                                                        if(aux[j][k+1]==0){
                                //Keep sweeping
                                                                k++;
                                                                continue;
                                                        }
                                                        else break;
                                        //Enemy piece found
                                                }
                                                j=x;
                                                k=y;
                                                while(k>1){
                                //Left
                                                        if(aux[j][k-1]*turn>0){
                        //Ally piece found
                                                                break;
                                                        }
                                                        list.i1[i]=x;
                                //Either empty or enemy piece, for both the move must be
listed
                                                        list.i2[i]=y;
                                                        list.f1[i]=j;
                                                        list.f2[i]=k-1;
                                                        i++;
                                                        if(aux[j][k-1]==0){
                                //Keep sweeping
                                                                k--;
                                                                continue;
                                                        }
                                                        else break;
                                        //Enemy piece found
                                                }
                                                j=x;
                                                k=y;
                                                while(j<8){
```

```
                        //Up
                                                if(aux[j+1][k]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                        //Either empty or enemy piece, for both the move must be
listed
                                                list.i2[i]=y;
                                                list.f1[i]=j+1;
                                                list.f2[i]=k;
                                                i++;
                                                if(aux[j+1][k]==0){
                        //Keep sweeping
                                                        j++;
                                                        continue;
                                                }
                                                else break;
                                //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j>1){
                        //Down
                                                if(aux[j-1][k]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                        //Either empty or enemy piece, for both the move must be
listed
                                                list.i2[i]=y;
                                                list.f1[i]=j-1;
                                                list.f2[i]=k;
                                                i++;
                                                if(aux[j-1][k]==0){
                        //Keep sweeping
                                                        j--;
                                                        continue;
                                                }
                                                else break;
                                //Enemy piece found
                                        }
                                break;

                                case 5:
                /* ****************************   T5- List of moves for a queen
**************************** */
                                        j=x;
                                        k=y;
                                        while(j<8 && k<8){
                //Upper Right
                                                if(aux[j+1][k+1]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                                //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
```

```
                                        list.f1[i]=j+1;
                                        list.f2[i]=k+1;
                                        i++;
                                        if(aux[j+1][k+1]==0){
                    //Keep sweeping
                                                j++;
                                                k++;
                                                continue;
                                        }
                                        else break;
                                //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j>1 && k<8){
            //Lower Right
                                        if(aux[j-1][k+1]*turn>0){
            //Ally piece found
                                                break;
                                        }
                                        list.i1[i]=x;
                        //Either empty or enemy piece, for both the move
must be listed
                                        list.i2[i]=y;
                                        list.f1[i]=j-1;
                                        list.f2[i]=k+1;
                                        i++;
                                        if(aux[j-1][k+1]==0){
                    //Keep sweeping
                                                j--;
                                                k++;
                                                continue;
                                        }
                                        else break;
                                //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j<8 && k>1){
            //Upper Left
                                        if(aux[j+1][k-1]*turn>0){
            //Ally piece found
                                                break;
                                        }
                                        list.i1[i]=x;
                        //Either empty or enemy piece, for both the move
must be listed
                                        list.i2[i]=y;
                                        list.f1[i]=j+1;
                                        list.f2[i]=k-1;
                                        i++;
                                        if(aux[j+1][k-1]==0){
                    //Keep sweeping
                                                j++;
                                                k--;
                                                continue;
                                        }
                                        else break;
                                //Enemy piece found
                                        }
```

```
                                        j=x;
                                        k=y;
                                        while(j>1 && k>1){
                //Lower Left
                                                if(aux[j-1][k-1]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                                //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
                                                list.f1[i]=j-1;
                                                list.f2[i]=k-1;
                                                i++;
                                                if(aux[j-1][k-1]==0){
                        //Keep sweeping
                                                        j--;
                                                        k--;
                                                        continue;
                                                }
                                                else break;
                                        //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(k<8){
                        //Right
                                                if(aux[j][k+1]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                        //Either empty or enemy piece, for both the move must be
listed
                                                list.i2[i]=y;
                                                list.f1[i]=j;
                                                list.f2[i]=k+1;
                                                i++;
                                                if(aux[j][k+1]==0){
                        //Keep sweeping
                                                        k++;
                                                        continue;
                                                }
                                                else break;
                                //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(k>1){
                        //Left
                                                if(aux[j][k-1]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                        //Either empty or enemy piece, for both the move must be
listed
                                                list.i2[i]=y;
                                                list.f1[i]=j;
```

```
                                        list.f2[i]=k-1;
                                        i++;
                                        if(aux[j][k-1]==0){
                    //Keep sweeping
                                                k--;
                                                continue;
                                        }
                                        else break;
                            //Enemy piece found
                                    }
                                    j=x;
                                    k=y;
                                    while(j<8){
                    //Up
                                        if(aux[j+1][k]*turn>0){
                //Ally piece found
                                                break;
                                        }
                                        list.i1[i]=x;
                    //Either empty or enemy piece, for both the move must be
listed
                                        list.i2[i]=y;
                                        list.f1[i]=j+1;
                                        list.f2[i]=k;
                                        i++;
                                        if(aux[j+1][k]==0){
                    //Keep sweeping
                                                j++;
                                                continue;
                                        }
                                        else break;
                            //Enemy piece found
                                    }
                                    j=x;
                                    k=y;
                                    while(j>1){
                    //Down
                                        if(aux[j-1][k]*turn>0){
                //Ally piece found
                                                break;
                                        }
                                        list.i1[i]=x;
                    //Either empty or enemy piece, for both the move must be
listed
                                        list.i2[i]=y;
                                        list.f1[i]=j-1;
                                        list.f2[i]=k;
                                        i++;
                                        if(aux[j-1][k]==0){
                    //Keep sweeping
                                                j--;
                                                continue;
                                        }
                                        else break;
                            //Enemy piece found
                                    }
                        break;

                        case 6:
```

```
                /* ****************************   T6- List of moves for a king
**************************** */
                                        for(j=-1;j<2;j++){
                                                for(k=-1;k<2;k++){

        if(aux[x+j][y+k]*turn>0||(x+j)<=0||x+j>=9||y+k<=0||y+k>=9)
                                                                continue;
                                                        list.i1[i]=x;
                                //Either empty or enemy piece, for both the move
must be listed
                                                        list.i2[i]=y;
                                                        list.f1[i]=x+j;
                                                        list.f2[i]=y+k;
                                                        i++;
                                                }
                                        }
                                        if(castle[0]==0||castle[1]==0){
                //Castle White
                                                if(turn>0){
                                                        if(castle[0]==0){
                        //Short
                                                                list.i1[i]=x;
                                                                list.i2[i]=y;
                                                                list.f1[i]=1;
                                                                list.f2[i]=7;
                                                                i++;
                                                        }
                                                        if(castle[1]==0){
                        //Long
                                                                list.i1[i]=x;
                                                                list.i2[i]=y;
                                                                list.f1[i]=1;
                                                                list.f2[i]=3;
                                                                i++;
                                                        }
                                                }
                                        }
                                        if(castle[2]==0||castle[3]==0){
                //Castle Black
                                                if(turn<0){
                                                        if(castle[2]==0){
                        //Short
                                                                list.i1[i]=x;
                                                                list.i2[i]=y;
                                                                list.f1[i]=8;
                                                                list.f2[i]=7;
                                                                i++;
                                                        }
                                                        if(castle[3]==0){
                        //Long
                                                                list.i1[i]=x;
                                                                list.i2[i]=y;
                                                                list.f1[i]=8;
                                                                list.f2[i]=3;
                                                                i++;
                                                        }
                                                }
                                        }
                                break;
                        }
```

```
                    else
                        switch(aux[x][y]){
                            case -1:

            /* ***************************   T1- List of moves for a pawn
**************************** */

                                        if( x==2 || x==7 ) {        //Are the
pawns on their original squares?
                                            if(aux[x+turn][y]==0)    {

        if(aux[x+2*turn][y]==0){
                                                //List Front &
Double Step
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+turn;
                                                list.f2[i]=y;
                                                i++;
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+2*turn;
                                                list.f2[i]=y;
                                                i++;
                                                }
                                                else{
                                                //List Front
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+turn;
                                                list.f2[i]=y;
                                                i++;
                                                }
                                            }
                                        }
                                        else{
                                            if(aux[x+turn][y]==0)    {
                                                //List Front
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+turn;
                                                list.f2[i]=y;

        if(aux[x+turn][y]==1)

        list.prom[i]=-queen;
                                                i++;

                                            }
                                        }
                                        if(aux[x+turn][y+turn]*turn<0){
        //Enemy piece on the diagonals?
                                            list.i1[i]=x;
                                            list.i2[i]=y;
                                            list.f1[i]=x+turn;
                                            list.f2[i]=y+turn;
                                            i++;
                                        }
                                        if(aux[x+turn][y-turn]*turn<0){
        //Enemy piece on the diagonals?
```

```
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+turn;
                                                list.f2[i]=y-turn;
                                                i++;
                                        }
                                        if(
aux[log.f1[move_count*2]][log.f2[move_count*2]]==(-turn) &&
abs(log.f1[move_count*2]-log.i1[move_count*2])==2 && (log.i2[move_count*2]==(y-
1)||log.i2[move_count*2]==(y+1)) && log.f1[move_count*2]==x ) {//En passant - last
move was from the enemy pawn, was a double step, it is in an adjacent collumn
from the analyzed square, the analyzed square's pawn is on the correct row to
capture
                                                list.i1[i]=x;
                                                list.i2[i]=y;

        list.f1[i]=log.f1[move_count*2]+turn;

        list.f2[i]=log.f2[move_count*2];
                                                list.prom[i]=-1;
                //En passant coding, in order to delete the enemy pawn
                                                i++;
                                        }
                                break;

                                case -2:

                /* ***************************   T2- List of moves for a knight
**************************** */
                                        if((x+2<9) && (y+1<9) &&
(aux[x+2][y+1]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+2;
                                                list.f2[i]=y+1;
                                                i++;
                                        }
                                        if((x+2<9)&&(y-1>0)&&(aux[x+2][y-
1]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+2;
                                                list.f2[i]=y-1;
                                                i++;
                                        }
                                        if((x-2>0)&&(y+1<9)&&(aux[x-
2][y+1]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x-2;
                                                list.f2[i]=y+1;
                                                i++;
                                        }
                                        if((x-2>0)&&(y-1>0)&&(aux[x-2][y-
1]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x-2;
                                                list.f2[i]=y-1;
                                                i++;
```

```
                                        }

        if((x+1<9)&&(y+2<9)&&(aux[x+1][y+2]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+1;
                                                list.f2[i]=y+2;
                                                i++;
                                        }
                                        if((x+1<9)&&(y-2>0)&&(aux[x+1][y-
2]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x+1;
                                                list.f2[i]=y-2;
                                                i++;
                                        }
                                        if((x-1>0)&&(y+2<9)&&(aux[x-
1][y+2]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x-1;
                                                list.f2[i]=y+2;
                                                i++;
                                        }
                                        if((x-1>0)&&(y-2>0)&&(aux[x-1][y-
2]*turn<=0)){
                                                list.i1[i]=x;
                                                list.i2[i]=y;
                                                list.f1[i]=x-1;
                                                list.f2[i]=y-2;
                                                i++;
                                        }
                                break;

                                case -3:

                /* ***************************   T3- List of moves for a bishop
**************************** */
                                        j=x;
                                        k=y;
                                        while(j<8 && k<8){
                //Upper Right
                                                if(aux[j+1][k+1]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                                //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
                                                list.f1[i]=j+1;
                                                list.f2[i]=k+1;
                                                i++;
                                                if(aux[j+1][k+1]==0){
                        //Keep sweeping
                                                        j++;
                                                        k++;
                                                        continue;
                                                }
```

```
                                        else break;
                                    //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j>1 && k<8){
                    //Lower Right
                                        if(aux[j-1][k+1]*turn>0){
                    //Ally piece found
                                            break;
                                        }
                                        list.i1[i]=x;
                                //Either empty or enemy piece, for both the move
must be listed
                                        list.i2[i]=y;
                                        list.f1[i]=j-1;
                                        list.f2[i]=k+1;
                                        i++;
                                        if(aux[j-1][k+1]==0){
                            //Keep sweeping
                                            j--;
                                            k++;
                                            continue;
                                        }
                                        else break;
                                    //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j<8 && k>1){
                    //Upper Left
                                        if(aux[j+1][k-1]*turn>0){
                    //Ally piece found
                                            break;
                                        }
                                        list.i1[i]=x;
                                //Either empty or enemy piece, for both the move
must be listed
                                        list.i2[i]=y;
                                        list.f1[i]=j+1;
                                        list.f2[i]=k-1;
                                        i++;
                                        if(aux[j+1][k-1]==0){
                            //Keep sweeping
                                            j++;
                                            k--;
                                            continue;
                                        }
                                        else break;
                                    //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j>1 && k>1){
                    //Lower Left
                                        if(aux[j-1][k-1]*turn>0){
                    //Ally piece found
                                            break;
                                        }
                                        list.i1[i]=x;
```

```
                                        //Either empty or enemy piece, for both the move
must be listed
                                                                list.i2[i]=y;
                                                                list.f1[i]=j-1;
                                                                list.f2[i]=k-1;
                                                                i++;
                                                                if(aux[j-1][k-1]==0){
                                //Keep sweeping
                                                                        j--;
                                                                        k--;
                                                                        continue;
                                                                }
                                                                else break;
                                                        //Enemy piece found
                                                                }
                                                        break;
                                                        case -4:

                                /* ****************************   T4- List of moves for a rook
**************************** */

                                                                j=x;
                                                                k=y;
                                                                while(k<8){
                        //Right
                                                                        if(aux[j][k+1]*turn>0){
                                //Ally piece found
                                                                                break;
                                                                }
                                                                list.i1[i]=x;
                                        //Either empty or enemy piece, for both the move must be
listed
                                                                list.i2[i]=y;
                                                                list.f1[i]=j;
                                                                list.f2[i]=k+1;
                                                                i++;
                                                                if(aux[j][k+1]==0){
                                //Keep sweeping
                                                                        k++;
                                                                        continue;
                                                                }
                                                                else break;
                                                        //Enemy piece found
                                                                }
                                                                j=x;
                                                                k=y;
                                                                while(k>1){
                        //Left
                                                                        if(aux[j][k-1]*turn>0){
                                //Ally piece found
                                                                                break;
                                                                }
                                                                list.i1[i]=x;
                                        //Either empty or enemy piece, for both the move must be
listed
                                                                list.i2[i]=y;
                                                                list.f1[i]=j;
                                                                list.f2[i]=k-1;
                                                                i++;
                                                                if(aux[j][k-1]==0){
```

```
                        //Keep sweeping
                                                k--;
                                                continue;
                                        }
                                        else break;
                                //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j<8){
        //Up
                                        if(aux[j+1][k]*turn>0){
                //Ally piece found
                                                break;
                                        }
                                        list.i1[i]=x;
                        //Either empty or enemy piece, for both the move must be
listed
                                        list.i2[i]=y;
                                        list.f1[i]=j+1;
                                        list.f2[i]=k;
                                        i++;
                                        if(aux[j+1][k]==0){
                        //Keep sweeping
                                                j++;
                                                continue;
                                        }
                                        else break;
                                //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j>1){
        //Down
                                        if(aux[j-1][k]*turn>0){
                //Ally piece found
                                                break;
                                        }
                                        list.i1[i]=x;
                        //Either empty or enemy piece, for both the move must be
listed
                                        list.i2[i]=y;
                                        list.f1[i]=j-1;
                                        list.f2[i]=k;
                                        i++;
                                        if(aux[j-1][k]==0){
                        //Keep sweeping
                                                j--;
                                                continue;
                                        }
                                        else break;
                                //Enemy piece found
                                }
                        break;

                        case -5:

                        /* ***************************   T5- List of moves for a queen
**************************** */
                                j=x;
```

```
                                        k=y;
                                        while(j<8 && k<8){
                //Upper Right
                                                if(aux[j+1][k+1]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                                //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
                                                list.f1[i]=j+1;
                                                list.f2[i]=k+1;
                                                i++;
                                                if(aux[j+1][k+1]==0){
                        //Keep sweeping
                                                        j++;
                                                        k++;
                                                        continue;
                                                }
                                                else break;
                                        //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j>1 && k<8){
                //Lower Right
                                                if(aux[j-1][k+1]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                                //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
                                                list.f1[i]=j-1;
                                                list.f2[i]=k+1;
                                                i++;
                                                if(aux[j-1][k+1]==0){
                        //Keep sweeping
                                                        j--;
                                                        k++;
                                                        continue;
                                                }
                                                else break;
                                        //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j<8 && k>1){
                //Upper Left
                                                if(aux[j+1][k-1]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                                //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
                                                list.f1[i]=j+1;
```

```
                                        list.f2[i]=k-1;
                                        i++;
                                        if(aux[j+1][k-1]==0){
                    //Keep sweeping
                                                j++;
                                                k--;
                                                continue;
                                        }
                                        else break;
                                //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j>1 && k>1){
                //Lower Left
                                        if(aux[j-1][k-1]*turn>0){
                //Ally piece found
                                                break;
                                        }
                                        list.i1[i]=x;
                            //Either empty or enemy piece, for both the move
must be listed
                                        list.i2[i]=y;
                                        list.f1[i]=j-1;
                                        list.f2[i]=k-1;
                                        i++;
                                        if(aux[j-1][k-1]==0){
                    //Keep sweeping
                                                j--;
                                                k--;
                                                continue;
                                        }
                                        else break;
                                //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(k<8){
        //Right
                                        if(aux[j][k+1]*turn>0){
                //Ally piece found
                                                break;
                                        }
                                        list.i1[i]=x;
                    //Either empty or enemy piece, for both the move must be
listed
                                        list.i2[i]=y;
                                        list.f1[i]=j;
                                        list.f2[i]=k+1;
                                        i++;
                                        if(aux[j][k+1]==0){
                    //Keep sweeping
                                                k++;
                                                continue;
                                        }
                                        else break;
                            //Enemy piece found
                                }
                                j=x;
                                k=y;
```

```
                                        while(k>1){
        //Left
                                                if(aux[j][k-1]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                        //Either empty or enemy piece, for both the move must be
listed
                                                list.i2[i]=y;
                                                list.f1[i]=j;
                                                list.f2[i]=k-1;
                                                i++;
                                                if(aux[j][k-1]==0){
                        //Keep sweeping
                                                        k--;
                                                        continue;
                                                }
                                                else break;
                                //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j<8){
        //Up
                                                if(aux[j+1][k]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                        //Either empty or enemy piece, for both the move must be
listed
                                                list.i2[i]=y;
                                                list.f1[i]=j+1;
                                                list.f2[i]=k;
                                                i++;
                                                if(aux[j+1][k]==0){
                        //Keep sweeping
                                                        j++;
                                                        continue;
                                                }
                                                else break;
                                //Enemy piece found
                                        }
                                        j=x;
                                        k=y;
                                        while(j>1){
        //Down
                                                if(aux[j-1][k]*turn>0){
                //Ally piece found
                                                        break;
                                                }
                                                list.i1[i]=x;
                        //Either empty or enemy piece, for both the move must be
listed
                                                list.i2[i]=y;
                                                list.f1[i]=j-1;
                                                list.f2[i]=k;
                                                i++;
                                                if(aux[j-1][k]==0){
```

```
                    //Keep sweeping
                                                j--;
                                                continue;
                                        }
                                        else break;
                            //Enemy piece found
                                        }
                                break;

                                case -6:

                        /* ***************************   T6- List of moves for a king
**************************** */
                                        for(j=-1;j<2;j++){
                                                for(k=-1;k<2;k++){

        if(aux[x+j][y+k]*turn>0||(x+j)<=0||x+j>=9||y+k<=0||y+k>=9)
                                                                continue;
                                                list.i1[i]=x;
                                        //Either empty or enemy piece, for both the move
must be listed
                                                list.i2[i]=y;
                                                list.f1[i]=x+j;
                                                list.f2[i]=y+k;
                                                i++;
                                        }
                                        }
                                        if(castle[0]==0||castle[1]==0){
                        //Castle White
                                                if(turn>0){
                                                        if(castle[0]==0){
                                //Short
                                                                list.i1[i]=x;
                                                                list.i2[i]=y;
                                                                list.f1[i]=1;
                                                                list.f2[i]=7;
                                                                i++;
                                                        }
                                                        if(castle[1]==0){
                                //Long
                                                                list.i1[i]=x;
                                                                list.i2[i]=y;
                                                                list.f1[i]=1;
                                                                list.f2[i]=3;
                                                                i++;
                                                        }
                                                }
                                        }
                                        if(castle[2]==0||castle[3]==0){
                        //Castle Black
                                                if(turn<0){
                                                        if(castle[2]==0){
                                //Short
                                                                list.i1[i]=x;
                                                                list.i2[i]=y;
                                                                list.f1[i]=8;
                                                                list.f2[i]=7;
                                                                i++;
                                                        }
                                                        if(castle[3]==0){
```

```
                        //Long
                                                                list.i1[i]=x;
                                                                list.i2[i]=y;
                                                                list.f1[i]=8;
                                                                list.f2[i]=3;
                                                                i++;
                                                        }
                                                }
                                        }
                                        break;
                                }
                        }

        }


        // Checking Legality of all the listed moves
        for (j=0;j<i;j++){
                        //Castle move                   Obs: As T6 checks
castle[0,1,2 or 3]==0 beforehand, premisse is king and rook did not move
                if( ((aux[list.i1[j]][list.i2[j]]==king)||(aux[list.i1[j]][list.i2[j]]==-king)) &&
abs(list.f2[j]-list.i2[j])==2){                 //Castle?
                        castle_flag=1;
                        if(list.f2[j]==7){  //Short
                                if(list.f1[j]==1){  //White
                                        k=2;
                                        while(k<9){                 //Vertical
                                                if(aux[k][7]!=0)
                                                        if(aux[k][7]>0 ||
(aux[k][7]!=(-queen)&&aux[k][7]!=(-rook)))
                                                                break;
                                                        else
                                                                goto Delete;
                                                if(aux[k][6]!=0)
                                                        if(aux[k][6]>0 ||
(aux[k][6]!=(-queen)&&aux[k][6]!=(-rook)))
                                                                break;
                                                        else
                                                                goto Delete;
                                                if(aux[k][5]!=0)
                                                        if(aux[k][5]>0 ||
(aux[k][5]!=(-queen)&&aux[k][5]!=(-rook)))
                                                                break;
                                                        else
                                                                goto Delete;
                                                k++;
                                        }
                                        if(aux[1][6]!=0||aux[1][7]!=0)
        //Horizontal
                                                goto Delete;
                                        //Diagonals
                                        k=2;
                                        h=4;
                                        while(h<7){                 //Right
                                                while((k+h)<9){
                                                        aux1=aux[k][k+h];
                                                        if(k==2){
                                                                if(aux1!=0)
                                                                        if(aux1>0 ||
(aux1!=-queen&&aux1!=-bishop&&aux1!=-pawn))
```

```
		break;
                                                                        else

		goto Delete;
                                                        }
                                                        else{
                                                                if(aux1!=0)
                                                                        if(aux1>0 ||
(aux1!=-queen&&aux1!=-bishop))

		break;
                                                                        else

		goto Delete;
                                                        }
                                                        k++;
                                                }
                                                k=2;
                                                h++;
                                        }
                                        k=2;
                                        h=8;
                                        while(h>5){             //Left
                                                while((h-k)>0){
                                                        aux1=aux[k][h-k];
                                                        if(k==2){
                                                                if(aux1!=0)
                                                                        if(aux1>0 ||
(aux1!=-queen&&aux1!=-bishop&&aux1!=-pawn))

		break;
                                                                        else

		goto Delete;
                                                        }
                                                        else{
                                                                if(aux1!=0)
                                                                        if(aux1>0 ||
(aux1!=-queen&&aux1!=-bishop))

		break;
                                                                        else

		goto Delete;
                                                        }
                                                        k++;
                                                }
                                                k=2;
                                                h--;
                                        }

                                        }
                                        else{                           //Black
                                                k=7;
                                                while(k>0){             //Vertical
                                                        if(aux[k][7]!=0)
                                                                if(aux[k][7]<0 ||
(aux[k][7]!=queen&&aux[k][7]!=rook))
                                                                        break;
```

```
                                        else
                                                goto Delete;
                                        if(aux[k][6]!=0)
                                                if(aux[k][6]<0 ||
(aux[k][6]!=queen&&aux[k][6]!=rook))
                                                        break;
                                                else
                                                        goto Delete;
                                        if(aux[k][5]!=0)
                                                if(aux[k][5]<0 ||
(aux[k][5]!=queen&&aux[k][5]!=rook))
                                                        break;
                                                else
                                                        goto Delete;

                                        k--;
                                }
                                if(aux[8][6]!=0||aux[8][7]!=0)
        //Horizontal
                                        goto Delete;
                                //Diagonals
                                k=7;
                                h=13;
                                while(h<16){            //Right
                                        while((h-k)<9){
                                                aux1=aux[k][h-k];
                                                if(k==7){
                                                        if(aux1!=0)
                                                                if(aux1<0 ||
(aux1!=queen&&aux1!=bishop&&aux1!=pawn))

        break;
                                                                else

        goto Delete;
                                                }
                                                else{
                                                        if(aux1!=0)
                                                                if(aux1<0 ||
(aux1!=queen&&aux1!=bishop))

        break;
                                                                else

        goto Delete;
                                                }
                                                k--;
                                        }
                                        k=7;
                                        h++;
                                }
                                k=7;
                                h=-1;
                                while(h>-4){            //Left
                                        while((k+h)>0){
                                                aux1=aux[k][k+h];
                                                if(k==7){
                                                        if(aux1!=0)
                                                                if(aux1<0 ||
(aux1!=queen&&aux1!=bishop&&aux1!=pawn))
```

```
                break;
                                                                        else

                goto Delete;
                                                                }
                                                                else{
                                                                        if(aux1!=0)
                                                                                if(aux1<0 ||
(aux1!=queen&&aux1!=bishop))

                break;
                                                                        else

                goto Delete;
                                                                }
                                                                k--;
                                                        }
                                                        k=7;
                                                        h--;
                                                }
                                        }
                                }
                                else{           //Long
                                        if(list.f1[j]==1){  //White
                                                k=2;
                                                while(k<9){                     //Vertical
                                                        if(aux[k][3]!=0)
                                                                if(aux[k][3]>0 ||
(aux[k][3]!=(-queen)&&aux[k][3]!=(-rook)))
                                                                        break;
                                                                else
                                                                        goto Delete;
                                                        if(aux[k][4]!=0)
                                                                if(aux[k][4]>0 ||
(aux[k][4]!=(-queen)&&aux[k][4]!=(-rook)))
                                                                        break;
                                                                else
                                                                        goto Delete;
                                                        if(aux[k][5]!=0)
                                                                if(aux[k][5]>0 ||
(aux[k][5]!=(-queen)&&aux[k][5]!=(-rook)))
                                                                        break;
                                                                else
                                                                        goto Delete;
                                                        k++;
                                                }
                                                if(aux[1][2]!=0||aux[1][3]!=0||aux[1][4]!=0)
        //Horizontal
                                                        goto Delete;
                                                //Diagonals
                                                k=2;
                                                h=4;
                                                while(h>1){             //Right
                                                        while((k+h)<9){
                                                                aux1=aux[k][k+h];
                                                                if(k==2){
                                                                        if(aux1!=0)
                                                                                if(aux1>0 ||
(aux1!=-queen&&aux1!=-bishop&&aux1!=-pawn))
```

```
		break;
                                                                        else

		goto Delete;
                                                        }
                                                        else{
                                                                if(aux1!=0)
                                                                        if(aux1>0 ||
(aux1!=-queen&&aux1!=-bishop))

		break;
                                                                        else

		goto Delete;
                                                        }
                                                        k++;
                                                }
                                                k=2;
                                                h--;
                                        }
                                        k=2;
                                        h=6;
                                        while(h>3){             //Left
                                                while((h-k)>0){
                                                        aux1=aux[k][h-k];
                                                        if(k==2){
                                                                if(aux1!=0)
                                                                        if(aux1>0 ||
(aux1!=-queen&&aux1!=-bishop&&aux1!=-pawn))

		break;
                                                                        else

		goto Delete;
                                                        }
                                                        else{
                                                                if(aux1!=0)
                                                                        if(aux1>0 ||
(aux1!=-queen && aux1!=-bishop))

		break;
                                                                        else

		goto Delete;
                                                        }
                                                        k++;
                                                }
                                                k=2;
                                                h--;
                                        }

                                        }
                                        else{                   //Black
                                                k=7;
                                                while(k>0){             //Vertical
                                                        if(aux[k][3]!=0)
                                                                if(aux[k][3]<0 ||
(aux[k][3]!=queen && aux[k][3]!=rook))
                                                                        break;
```

```
                                        else
                                                goto Delete;
                                        if(aux[k][4]!=0)
                                                if(aux[k][4]<0 ||
(aux[k][4]!=queen && aux[k][4]!=rook))
                                                        break;
                                                else
                                                        goto Delete;
                                        if(aux[k][5]!=0)
                                                if(aux[k][5]<0 ||
(aux[k][5]!=queen && aux[k][5]!=rook))
                                                        break;
                                                else
                                                        goto Delete;

                                        k--;
                                }
                                if(aux[8][2]!=0||aux[8][3]!=0||aux[8][4]!=0)
        //Horizontal
                                        goto Delete;
                                //Diagonals
                                k=7;
                                h=13;
                                while(h>10){            //Right
                                        while((h-k)<9){
                                                aux1=aux[k][h-k];
                                                if(k==7){
                                                        if(aux1!=0)
                                                                if(aux1<0 ||
(aux1!=queen && aux1!=bishop && aux1!=pawn))

        break;
                                                                else

        goto Delete;
                                                }
                                                else{
                                                        if(aux1!=0)
                                                                if(aux1<0 ||
(aux1!=queen&& aux1!=bishop))

        break;
                                                                else

        goto Delete;
                                                }
                                                k--;
                                        }
                                        k=7;
                                        h--;
                                }
                                k=7;
                                h=-3;
                                while(h>-6){            //Left
                                        while((k+h)>0){
                                                aux1=aux[k][k+h];
                                                if(k==7){
                                                        if(aux1!=0)
                                                                if(aux1<0 ||
(aux1!=queen&& aux1!=bishop&& aux1!=pawn))
```

```
		break;
                                                                        else

		goto Delete;
                                                                }
                                                                else{
                                                                        if(aux1!=0)
                                                                                if(aux1<0 ||
(aux1!=queen && aux1!=bishop))

		break;
                                                                        else

		goto Delete;
                                                                }
                                                                k--;
                                                        }
                                                        k=7;
                                                        h--;
                                                }
                                        }

                                }
                                goto Knight_check;
                        }

                        T0(aux,list,j,turn);                    // Move is artificially made in order
to check whether the king will be in check

                        k=1;
                        // Diagonals
                        while( (rex+k)<9 && (rey+k)<9){
                                aux1=aux[rex+k][rey+k]*turn;
                                if(k==1){
                                        if(aux1!=0)
                                                if(aux1>0|| (aux1!=-queen&& aux1!=-bishop
&& aux1!=-pawn && aux1!=-king))
                                                        break;
                                                else
                                                        goto Delete;
                                }
                                else{
                                        if(aux1!=0)
                                                if(aux1>0|| (aux1!=-queen && aux1!=-
bishop))
                                                        break;
                                                else
                                                        goto Delete;
                                }
                                k++;
                        }
                        k=1;
                        while( (rex+k)<9 && (rey-k)>0){
                                aux1=aux[rex+k][rey-k]*turn;
                                if(k==1){
                                        if(aux1!=0)
                                                if(aux1>0|| (aux1!=-queen && aux1!=-
bishop&& aux1!=-pawn && aux1!=-king))
                                                        break;
```

```
                                        else
                                                goto Delete;
                        }
                        else{
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop))
                                                break;
                                        else
                                                goto Delete;
                        }
                        k++;
                }
                k=1;
                while( (rex-k)>0 && (rey+k)<9){
                        aux1=aux[rex-k][rey+k]*turn;
                        if(k==1){
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop && aux1!=-pawn && aux1!=-king))
                                                break;
                                        else
                                                goto Delete;
                        }
                        else{
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop))
                                                break;
                                        else
                                                goto Delete;
                        }
                        k++;
                }
                k=1;
                while( (rex-k)>0 && (rey-k)>0){
                        aux1=aux[rex-k][rey-k]*turn;
                        if(k==1){
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop && aux1!=-pawn && aux1!=-king))
                                                break;
                                        else
                                                goto Delete;
                        }
                        else{
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop))
                                                break;
                                        else
                                                goto Delete;
                        }
                        k++;
                }
                k=1;
                // Horizontal
                while( (rey-k)>0){
                        aux1=aux[rex][rey-k]*turn;
                        if(aux1!=0){
```

```
                        if(aux1==-king && k==1)
                                goto Delete;
                        if(aux1>0 || (aux1!=-queen && aux1!=-rook))
                                break;
                        else
                                goto Delete;
                }
                k++;
        }
        k=1;
        while( (rey+k)<9){
                aux1=aux[rex][rey+k]*turn;
                if(aux1!=0){
                        if(aux1==-king && k==1)
                                goto Delete;
                        if(aux1>0|| (aux1!=-queen && aux1!=-rook))
                                break;
                        else
                                goto Delete;
                }
                k++;
        }
        k=1;
        // Vertical
        while( (rex-k)>0){
                aux1=aux[rex-k][rey];
                if(aux1!=0){
                        if(aux1==-king && k==1)
                                goto Delete;
                        if(aux1>0|| (aux1!=-queen && aux1!=-rook))
                                break;
                        else
                                goto Delete;
                }
                k++;
        }
        k=1;
        while( (rex+k)<9){
                aux1=aux[rex+k][rey]*turn;
                if(aux1!=0){
                        if(aux1==-king && k==1)
                                goto Delete;
                        if( aux1>0 || (aux1!=-queen && aux1!=-rook) )
                                break;
                        else
                                goto Delete;
                }
                k++;
        }
        // Knight
        Knight_check:
        if(aux[rex+2][rey+1]==(-knight*turn)||aux[rex+2][rey-1]==(-
knight*turn)||aux[rex-2][rey+1]==(-knight*turn)||aux[rex-2][rey-1]==(-
knight*turn)||aux[rex+1][rey+2]==(-knight*turn)||aux[rex+1][rey-2]==(-
knight*turn)||aux[rex-1][rey+2]==(-knight*turn)||aux[rex-1][rey-2]==(-knight*turn))
                goto Delete;
        else
                goto Legal_Move;
```

```
		// Delete the move from the list
		Delete:
		list.i1[j]=0;				//Deletable, mere illustrative as the
node wont be created nevertheless
		list.i2[j]=0;
		list.f1[j]=0;
		list.f2[j]=0;
		list.prom[j]=0;

		// Undo the move and store the king's previous square
		for(k=1;k<9;k++){
			for(h=1;h<9;h++){
				if(root->board[k][h]==king*turn){
					rex=k;
					rey=h;
				}
				if(root->board[k][h]==-king*turn){
					rix=k;
					riy=h;
				}
				aux[k][h]=root->board[k][h];
			}
		}
		castle_flag=0;
		continue;

		Legal_Move:
		mate_flag=0;
							// One legal move found= not a
mate
		memcpy(mat,aux,81*sizeof(int));

		// Evaluate the position

		if(move_count<8 && aux[list.i1[j]][list.i2[j]]==queen*turn)		//
Discourage early Queen moves
			eval-=50;
		if(move_count<20 && aux[list.i1[j]][list.i2[j]]==rook*turn)		//
Discourage early rook moves
			eval-=50;
		if(castle_flag==0 && aux[list.i1[j]][list.i2[j]]==king*turn)	//
Discourage early rook moves
			eval-=80;
		if(phase==0 && castle_flag==1)
				// Encourage castling
			eval+=80;

			for(k=1;k<9;k++){
				for(h=1;h<9;h++){
					if(root->board[k][h]==king*turn){
			// Store the king previous location
						rex=k;
						rey=h;
					}
					if(root->board[k][h]==-king*turn){
						rix=k;
						riy=h;
					}
					if(CPU==1){			// Engine is White
						if(aux[k][h]>0){
```

```
                                                        eval=eval +
PieceValue[phase][aux[k][h]] + WTable[phase][aux[k][h]][8*k+h-9];
                    // Obs:Position on the Eval tables[64]= 8*(x-1)+(y-1)

                                                        if(aux[k][h]==pawn){

          if(aux[k+1][h]==pawn)

          eval+=Pawns[0];

          for(g=k+1;g<8;g++){                 //Passed Pawns - Black pawns cant go to
rank 8
                                                                        if(h!=8 &&
h!=1)

          if(aux[g][h]==-pawn || aux[g][h+1]==-pawn || aux[g][h-1]==-pawn){

          passed=0;

          break;
                                                                                }
                                                                        if(h==1)

          if(aux[g][h]==-pawn || aux[g][h+1]==-pawn){

          passed=0;

          break;
                                                                                }

                                                                        if(h==8)

          if(aux[g][h]==-pawn || aux[g][h-1]==-pawn){

          passed=0;

          break;
                                                                                }

                                                                }
                                                                eval+=Pawns[h-
1]*passed;          //Higher bonus for higher rank
                                                        }
                                                        else
                                                                if(aux[k][h]!=king)
                              //Mobility not evaluated for pawns and kings

          eval+=MobilityBonus[aux[h][k]-2][phase][Mobility(k,h,aux[k][h],turn)];
                                                    }
                                                    if(aux[k][h]<0){
                                                                eval=eval-
PieceValue[phase][-aux[k][h]]-BTable[phase][-aux[k][h]][8*k+h-9];
                                                                if(aux[k][h]==-pawn){
                                                                        if(aux[k-1][h]==-
pawn)   //Doubled Pawns
                                                                                eval-
=Pawns[0];
                                                                for(g=k-1;g>1;g--){
                    //Passed Pawns - White pawns cant go to rank 1
```

```
                                                                        if(h!=8 &&
h!=1)
        if(aux[g][h]==-pawn || aux[g][h+1]==-pawn || aux[g][h-1]==-pawn){
        passed=0;
        break;
                                                                                }
                                                                        if(h==1)
        if(aux[g][h]==-pawn || aux[g][h+1]==-pawn){
        passed=0;
        break;
                                                                                }
                                                                        if(h==8)
        if(aux[g][h]==-pawn || aux[g][h-1]==-pawn){
        passed=0;
        break;
                                                                                }
                                                                }
                                                                eval-=Pawns[8-
h]*passed;
                                                        }
                                                        else
                                                                if(aux[k][h]!=-king)
                                                                        eval-
=MobilityBonus[-aux[h][k]-2][phase][Mobility(k,h,aux[k][h],turn)];
                                                        }
                                                }
                                                if(CPU==-1){ //Engine is Black
                                                        if(aux[k][h]>0){
                                                                eval= eval-
PieceValue[phase][aux[k][h]]-WTable[phase][aux[k][h]][8*k+h-9];
                        // Position on the Eval tables[64]= 8*(x-1)+(y-1)
                                                                if(aux[k][h]==pawn){
        if(aux[k+1][h]==pawn)
                                                                        eval-
=Pawns[0];                      // Doubled Pawns
        for(g=k+1;g<8;g++){                     // Passed Pawns - Black pawns cant go to
rank 8
                                                                        if(h!=8 &&
h!=1)
        if(aux[g][h]==-pawn || aux[g][h+1]==-pawn || aux[g][h-1]==-pawn){
        passed=0;
```

```
        break;
                                                                    }
                                                                if(h==1)
        if(aux[g][h]==-pawn || aux[g][h+1]==-pawn){
        passed=0;
        break;
                                                                    }
                                                                if(h==8)
        if(aux[g][h]==-pawn || aux[g][h-1]==-pawn){
        passed=0;
        break;
                                                                    }
                                                            }
                                                            eval-=Pawns[h-
1]*passed;
                                                    }
                                                    else
                                                            if(aux[k][h]!=king)
                                                                    eval-
=MobilityBonus[aux[h][k]-2][phase][Mobility(k,h,aux[k][h],turn)];
                                                }
                                                if(aux[k][h]<0){
                                                        eval=eval +
PieceValue[phase][-aux[k][h]] + BTable[phase][-aux[k][h]][8*k+h-9];
                                                        if(aux[k][h]==-pawn){
        // Doubled Pawns
                                                                if(aux[k-1][h]==-
pawn)
        eval+=Pawns[0];
                                                                for(g=k-1;g>1;g--){
                // Passed Pawns - White pawns cant go to rank 1
                                                                if(h!=8 &&
h!=1)
        if(aux[g][h]==-pawn || aux[g][h+1]==-pawn || aux[g][h-1]==-pawn){
        passed=0;
        break;
                                                                    }
                                                                if(h==1)
        if(aux[g][h]==-pawn || aux[g][h+1]==-pawn){
        passed=0;
        break;
                                                                    }
```

```
                                                                        if(h==8)

        if(aux[g][h]==-pawn || aux[g][h-1]==-pawn){

        passed=0;

        break;
                                                                                }

                                                                }
                                                                eval+=Pawns[8-
h]*passed;
                                                        }

                                                        else
                                                                if(aux[k][h]!=-king)

        eval+=MobilityBonus[-aux[h][k]-2][phase][Mobility(k,h,aux[k][h],turn)-1];
                                                        }
                                                }
                                        }
                                        }
                        passed=1;


                memcpy(aux,root->board,81*sizeof(int));

                //Create node
                nodes[aux3]=create(eval,mat,list,j);
                insert(root,nodes[aux3]);
                if(root->move.i1[0]==0){
                //Obs: As it is the first ever root, it wont have a parent move stored,
so these nodes shall receive the current analyzed move as they will be parent nodes
for the following ones
                        nodes[aux3]->move.i1[1]=list.i1[j];
                        nodes[aux3]->move.i2[1]=list.i2[j];
                        nodes[aux3]->move.f1[1]=list.f1[j];
                        nodes[aux3]->move.f2[1]=list.f2[j];
                        nodes[aux3]->move.prom[1]=list.prom[j];
                }
                else{
                        nodes[aux3]->move.i1[1]=root->move.i1[1];
                        nodes[aux3]->move.i2[1]=root->move.i2[1];
                        nodes[aux3]->move.f1[1]=root->move.f1[1];
                        nodes[aux3]->move.f2[1]=root->move.f2[1];
                        nodes[aux3]->move.prom[1]=root->move.prom[1];
                }
                eval=0;
                aux3++;
                castle_flag=0;
}
        if(mate_flag==1){          // Checks whether the enemy king is in check, as
check + no legal moves = Checkmate. If there's no check, it would be a Stalemate

                k=1;
                //Diagonals
                while( (rex+k)<9 && (rey+k)<9){
                        aux1=aux[rex+k][rey+k]*turn;
                        if(k==1){
                                if(aux1!=0)
```

```
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop && aux1!=-pawn && aux1!=-king))
                                                break;
                                        else
                                                goto Check;
                        }
                        else{
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop))
                                                break;
                                        else
                                                goto Check;
                        }
                        k++;
                }
                k=1;
                while( (rex+k)<9 && (rey-k)>0){
                        aux1=aux[rex+k][rey-k]*turn;
                        if(k==1){
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop&& aux1!=-pawn && aux1!=-king))
                                                break;
                                        else
                                                goto Check;
                        }
                        else{
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop))
                                                break;
                                        else
                                                goto Check;
                        }
                        k++;
                }
                k=1;
                while( (rex-k)>0 && (rey+k)<9){
                        aux1=aux[rex-k][rey+k]*turn;
                        if(k==1){
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop && aux1!=-pawn && aux1!=-king))
                                                break;
                                        else
                                                goto Check;
                        }
                        else{
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop))
                                                break;
                                        else
                                                goto Check;
                        }
                        k++;
                }
                k=1;
                while( (rex-k)>0 && (rey-k)>0){
```

```
                        aux1=aux[rex-k][rey-k]*turn;
                        if(k==1){
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop && aux1!=-pawn && aux1!=-king))
                                                break;
                                        else
                                                goto Check;
                        }
                        else{
                                if(aux1!=0)
                                        if(aux1>0|| (aux1!=-queen && aux1!=-
bishop))
                                                break;
                                        else
                                                goto Check;
                        }
                        k++;
                }
                k=1;
                //Horizontal
                while( (rey-k)>0){
                        aux1=aux[rex][rey-k]*turn;
                        if(aux1!=0){

                                if(aux1==-king && k==1)
                                        goto Check;
                                if(aux1>0 || (aux1!=-queen && aux1!=-rook))
                                        break;
                                else
                                        goto Check;
                        }
                        k++;
                }
                k=1;
                while( (rey+k)<9){
                        aux1=aux[rex][rey+k]*turn;
                        if(aux1!=0){
                                if(aux1==-king && k==1)
                                        goto Check;
                                if(aux1>0|| (aux1!=-queen && aux1!=-rook))
                                        break;
                                else
                                        goto Check;
                        }
                        k++;
                }
                k=1;
                //Vertical
                while( (rex-k)>0){
                        aux1=aux[rex-k][rey];
                        if(aux1!=0){
                                if(aux1==-king && k==1)
                                        goto Check;
                                if(aux1>0|| (aux1!=-queen && aux1!=-rook))
                                        break;
                                else
                                        goto Check;
                        }
                        k++;
```

```
                }
                k=1;
                while( (rex+k)<9){
                        aux1=aux[rex+k][rey]*turn;
                        if(aux1!=0){
                                if(aux1==-king && k==1)
                                        goto Check;
                                if( aux1>0 || (aux1!=-queen && aux1!=-rook) )
                                        break;
                                else
                                        goto Check;
                        }
                        k++;
                }
                //Knight
                if(aux[rex+2][rey+1]==(-knight*turn)||aux[rex+2][rey-1]==(-
knight*turn)||aux[rex-2][rey+1]==(-knight*turn)||aux[rex-2][rey-1]==(-
knight*turn)||aux[rex+1][rey+2]==(-knight*turn)||aux[rex+1][rey-2]==(-
knight*turn)||aux[rex-1][rey+2]==(-knight*turn)||aux[rex-1][rey-2]==(-knight*turn))
                        goto Check;
                else
                        goto Stalemate;

Check:
val=(infinite+depth);       //Mate found
goto Erase;
Stalemate:
val=0;                              //Stalemate found

}
Erase:
memset(list.i1, 0, sizeof list.i1);
memset(list.f1, 0, sizeof list.f1);
memset(list.i2, 0, sizeof list.i2);
memset(list.f2, 0, sizeof list.f2);
memset(list.prom, 0, sizeof list.prom);
return val;
}

// Search Function - Negamax applied
int Negamax(Tree* node,int dept,int alpha,int beta, int color){
        int auxi=0,value,aux;
        Tree *p;
        //|| (node->first==NULL && node->next==NULL && dept!=depth)
   if (dept == 0){
        auxi=CPU*color*(node->eval);
      return auxi;
        }
   aux=List_Moves(node,color,dept);

   if(aux!=1){
        aux=CPU*color*aux;
      return aux;
        }
//      value=-infinite;
//   for (p=node->first; p!=NULL; p=p->next){
//              auxi=-Negamax(p, dept-1, -beta, -alpha, -color);
//      if(auxi>value){
//              value=auxi;
//              if(dept==depth)
```

```
//                              e_move=p->move;
//          }
//    }
//     return value;
    for (p=node->first; p!=NULL; p=p->next){
                    auxi=-Negamax(p, dept-1, -beta, -alpha, -color);
          if(auxi>beta)
                    return beta;
                    if(auxi>alpha){
                    alpha=auxi;
                    if(dept==depth)
                              e_move=p->move;
          }
          }
    return alpha;
}

// Initiates the Board on its initial position
void InitiateBoard(){

// Initialize Board (Actual and aux auxiliares)
int j,k;

          memset(B, 0, sizeof B);             //Reset all elements

          for(j=1;j<9;j++) {

                    B[2][j]=1;
                    B[7][j]=-1;
          }
//B[8][8]=0; //Mate in 2 test
//B[1][1]=0;
//B[1][2]=0;
//B[1][3]=0;
//B[1][4]=0;
//B[4][4]=5;
//B[1][7]=-6;
//B[3][5]=6;
//B[1][8]=0;
//
//B[8][1]=0;
//B[8][2]=0;
//B[8][3]=0;
//B[8][4]=0;
//B[8][5]=0;
//B[8][6]=0;
//B[8][7]=0;
//B[8][8]=0;
//rex=3;
//rey=5;
//rix=1;
//riy=7;

//B[8][8]=0; //Mate in 3 test
//B[1][1]=0;
//B[1][2]=0;
//B[1][3]=0;
//B[1][4]=0;
//B[4][4]=5;
//B[1][6]=-6;
```

```
//B[4][5]=6;
//B[1][8]=0;
//
//B[8][1]=0;
//B[8][2]=0;
//B[8][3]=0;
//B[8][4]=0;
//B[8][5]=0;
//B[8][6]=0;
//B[8][7]=0;
//B[8][8]=0;
//rex=4;
//rey=5;
//rix=1;
//riy=6;
	B[1][1]=4;			//Classical board
	B[1][2]=2;
	B[1][3]=3;
	B[1][4]=5;
	B[1][5]=6;
	B[1][6]=3;
	B[1][7]=2;
	B[1][8]=4;

	B[8][1]=-4;
	B[8][2]=-2;
	B[8][3]=-3;
	B[8][4]=-5;
	B[8][5]=-6;
	B[8][6]=-3;
	B[8][7]=-2;
	B[8][8]=-4;
	rex=1;
	rey=5;
	rix=8;
	riy=5;

	memcpy(aux,B,81*sizeof(int));

	printf("\n");
	for(j=8;j>0;j--) {  //Print board
		for(k=1;k<9;k++){
			if(B[j][k]>=0)
				printf(" %d  ",B[j][k]);
			else
				printf("%d  ",B[j][k]);
		}
		printf("\n");
	}
	printf("\n");
}


// Function Mobility: Create Mobility Area and evaluates the Mobility of the Piece
analyzed
// Returns the number of squares attacked by the Piece in the Mobility Area
// Mobility Area: Whole Board excepting Pawns on their original squares, squares
attacked by Pawns and Queen and King current square
int Mobility(x,y,piece,turn){
	int v=0,k,h,MobArea[9][9];
```

```
	int j;

	memset(MobArea, 0, sizeof MobArea);

	for (k=1;k<9;k++){			//Generate the Mobility Area
		for (h=1;h<9;h++){
			if(aux[k][h]==pawn && x==2)
				continue;
			if(aux[k][h]==-pawn && x==7)
				continue;
			if(aux[k+turn][h+turn]!=(-pawn*turn)&&aux[k+turn][h-turn]!=(-
pawn*turn)&&aux[k][h]!=king*turn)	//Do not include in mobility area squares
protected by enemy pawns,occupied by our king
				MobArea[k][h]=1;
		}
	}

	switch(piece) {
			case 2:
				if( (x+2<9) && (y+1<9) && (MobArea[x+2][y+1]==1)
)
					v++;
				if( (x+2<9)&&(y-1>0)&&(MobArea[x+2][y-1]==1) )
					v++;
				if( (x-2>0)&&(y+1<9)&&(MobArea[x-2][y+1]==1))
					v++;
				if((x-2>0)&&(y-1>0)&&(MobArea[x-2][y-1]==1))
					v++;
				if((x+1<9)&&(y+2<9)&&(MobArea[x+1][y+2]==1))
					v++;
				if((x+1<9)&&(y-2>0)&&(MobArea[x+1][y-2]==1))
					v++;
				if((x-1>0)&&(y+2<9)&&(MobArea[x-1][y+2]==1))
					v++;
				if((x-1>0)&&(y-2>0)&&(MobArea[x-1][y-2]==1))
					v++;
				break;

			case -2:
				if( (x+2<9) && (y+1<9) && (MobArea[x+2][y+1]==1)
)
					v++;
				if( (x+2<9)&&(y-1>0)&&(MobArea[x+2][y-1]==1) )
					v++;
				if( (x-2>0)&&(y+1<9)&&(MobArea[x-2][y+1]==1))
					v++;
				if((x-2>0)&&(y-1>0)&&(MobArea[x-2][y-1]==1))
					v++;
				if((x+1<9)&&(y+2<9)&&(MobArea[x+1][y+2]==1))
					v++;
				if((x+1<9)&&(y-2>0)&&(MobArea[x+1][y-2]==1))
					v++;
				if((x-1>0)&&(y+2<9)&&(MobArea[x-1][y+2]==1))
					v++;
				if((x-1>0)&&(y-2>0)&&(MobArea[x-1][y-2]==1))
					v++;
				break;

			case 3:
				j=x;
```

```
                                k=y;
                                while(j<8 && k<8){
        //Upper Right
                                        if(aux[j+1][k+1]*turn>0 &&
aux[j+1][k+1]!=queen*turn){             //Ally piece found & not a queen to
x-ray
                                        break;
                                        }
                                        if(MobArea[j+1][k+1]==1)
                                                v++;
                                        if(aux[j+1][k+1]==0){
                                                                //Keep sweeping
                                                j++;
                                                k++;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j>1 && k<8){
        //Lower Right
                                        if(aux[j-1][k+1]*turn>0 && aux[j-
1][k+1]!=queen*turn){                   //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j-1][k+1]==1)
                                                v++;
                                        if(aux[j-1][k+1]==0){
                                                                //Keep sweeping
                                                j--;
                                                k++;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j<8 && k>1){
        //Upper Left
                                        if(aux[j+1][k-1]*turn>0 && aux[j+1][k-
1]!=queen*turn){                        //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j+1][k-1]==1)
                                                v++;
                                        if(aux[j+1][k-1]==0){
                                                                //Keep sweeping
                                                j++;
                                                k--;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
```

```
                                j=x;
                                k=y;
                                while(j>1 && k>1){
        //Lower Left
                                        if(aux[j-1][k-1]*turn>0 && aux[j-1][k-
1]!=queen*turn){                        //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j-1][k-1]==1)
                                                v++;
                                        if(aux[j-1][k-1]==0){
                                                                //Keep sweeping
                                                j--;
                                                k--;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
                                break;
                        case -3:
                                j=x;
                                k=y;
                                while(j<8 && k<8){
        //Upper Right
                                        if(aux[j+1][k+1]*turn>0 &&
aux[j+1][k+1]!=queen*turn){             //Ally piece found & not a queen to
x-ray
                                                break;
                                        }
                                        if(MobArea[j+1][k+1]==1)
                                                v++;
                                        if(aux[j+1][k+1]==0){
                                                                //Keep sweeping
                                                j++;
                                                k++;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j>1 && k<8){
        //Lower Right
                                        if(aux[j-1][k+1]*turn>0 && aux[j-
1][k+1]!=queen*turn){                   //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j-1][k+1]==1)
                                                v++;
                                        if(aux[j-1][k+1]==0){
                                                                //Keep sweeping
                                                j--;
                                                k++;
                                                continue;
                                        }
                                        else break;
```

```
        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j<8 && k>1){
        //Upper Left
                                        if(aux[j+1][k-1]*turn>0 && aux[j+1][k-
1]!=queen*turn){                        //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j+1][k-1]==1)
                                                v++;
                                        if(aux[j+1][k-1]==0){
                                                                //Keep sweeping
                                                j++;
                                                k--;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j>1 && k>1){
        //Lower Left
                                        if(aux[j-1][k-1]*turn>0 && aux[j-1][k-
1]!=queen*turn){                        //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j-1][k-1]==1)
                                                v++;
                                        if(aux[j-1][k-1]==0){
                                                                //Keep sweeping
                                                j--;
                                                k--;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }

                                break;
                        case 4:

                                j=x;
                                k=y;
                                while(k<8){                             //Right
                                        if(aux[j][k+1]*turn>0 &&
aux[j][k+1]!=queen*turn && aux[j][k+1]!=rook*turn ){                     //Ally piece
found
                                                break;
                                        }
                                        if(MobArea[j][k+1]==1)
                                                v++;
                                        if(aux[j][k+1]==0){

                                        //Keep sweeping
```

```
                                        k++;
                                        continue;
                                }
                                else break;

                                        //Enemy piece found
                        }
                        j=x;
                        k=y;
                        while(k>1){                                     //Left
                                if(aux[j][k-1]*turn>0 && aux[j][k-
1]!=queen*turn && aux[j][k-1]!=rook*turn){                              //Ally piece found
                                        break;
                                }
                                if(MobArea[j][k-1]==1)
                                        v++;
                                if(aux[j][k-1]==0){

                                //Keep sweeping
                                        k--;
                                        continue;
                                }
                                else break;

                                        //Enemy piece found
                        }
                        j=x;
                        k=y;
                        while(j<8){                                     //Up
                                if(aux[j+1][k]*turn>0 &&
aux[j+1][k]!=queen*turn && aux[j+1][k]!=rook*turn){                      //Ally piece
found
                                        break;
                                }
                                if(MobArea[j+1][k]==1)
                                        v++;
                                if(aux[j+1][k]==0){

                                //Keep sweeping
                                        j++;
                                        continue;
                                }
                                else break;

                                        //Enemy piece found
                        }
                        j=x;
                        k=y;
                        while(j>1){                                     //Down
                                if(aux[j-1][k]*turn>0 && aux[j-
1][k]!=queen*turn && aux[j-1][k]!=rook*turn){                           //Ally piece found
                                        break;
                                }
                                if(MobArea[j-1][k]==1)
                                        v++;
                                if(aux[j-1][k]==0){

                                //Keep sweeping
                                        j--;
                                        continue;
```

```
                                        }
                                        else break;

                                                //Enemy piece found
                                }
                                break;
                        case -4:

                                j=x;
                                k=y;
                                while(k<8){                                     //Right
                                        if(aux[j][k+1]*turn>0 &&
aux[j][k+1]!=queen*turn && aux[j][k+1]!=rook*turn ){                     //Ally piece
found
                                                break;
                                        }
                                        if(MobArea[j][k+1]==1)
                                                v++;
                                        if(aux[j][k+1]==0){

                                        //Keep sweeping
                                                k++;
                                                continue;
                                        }
                                        else break;

                                                //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(k>1){                                     //Left
                                        if(aux[j][k-1]*turn>0 && aux[j][k-
1]!=queen*turn && aux[j][k-1]!=rook*turn){                              //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j][k-1]==1)
                                                v++;
                                        if(aux[j][k-1]==0){

                                        //Keep sweeping
                                                k--;
                                                continue;
                                        }
                                        else break;

                                                //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j<8){                                     //Up
                                        if(aux[j+1][k]*turn>0 &&
aux[j+1][k]!=queen*turn && aux[j+1][k]!=rook*turn){                      //Ally piece
found
                                                break;
                                        }
                                        if(MobArea[j+1][k]==1)
                                                v++;
                                        if(aux[j+1][k]==0){

                                        //Keep sweeping
```

```
                                        j++;
                                        continue;
                                }
                                else break;

                                        //Enemy piece found
                        }
                        j=x;
                        k=y;
                        while(j>1){                                     //Down
                                if(aux[j-1][k]*turn>0 && aux[j-
1][k]!=queen*turn && aux[j-1][k]!=rook*turn){                   //Ally piece found
                                        break;
                                }
                                if(MobArea[j-1][k]==1)
                                        v++;
                                if(aux[j-1][k]==0){

                                //Keep sweeping
                                        j--;
                                        continue;
                                }
                                else break;

                                        //Enemy piece found
                        }
                        break;
                        break;
                case 5:
                        j=x;
                        k=y;
                        while(j<8 && k<8){
        //Upper Right
                                if(aux[j+1][k+1]*turn>0 &&
aux[j+1][k+1]!=queen*turn){                     //Ally piece found & not a queen to
x-ray

                                        break;
                                }
                                if(MobArea[j+1][k+1]==1)
                                        v++;
                                if(aux[j+1][k+1]==0){
                                                        //Keep sweeping
                                        j++;
                                        k++;
                                        continue;
                                }
                                else break;

        //Enemy piece found
                        }
                        j=x;
                        k=y;
                        while(j>1 && k<8){
        //Lower Right
                                if(aux[j-1][k+1]*turn>0 && aux[j-
1][k+1]!=queen*turn){                   //Ally piece found
                                        break;
                                }
                                if(MobArea[j-1][k+1]==1)
                                        v++;
```

```
                                        if(aux[j-1][k+1]==0){
                                                                        //Keep sweeping
                                                j--;
                                                k++;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j<8 && k>1){
        //Upper Left
                                        if(aux[j+1][k-1]*turn>0 && aux[j+1][k-
1]!=queen*turn){                        //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j+1][k-1]==1)
                                                v++;
                                        if(aux[j+1][k-1]==0){
                                                                        //Keep sweeping
                                                j++;
                                                k--;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j>1 && k>1){
        //Lower Left
                                        if(aux[j-1][k-1]*turn>0 && aux[j-1][k-
1]!=queen*turn){                        //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j-1][k-1]==1)
                                                v++;
                                        if(aux[j-1][k-1]==0){
                                                                        //Keep sweeping
                                                j--;
                                                k--;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }

                                j=x;
                                k=y;
                                while(k<8){                             //Right
                                        if(aux[j][k+1]*turn>0 &&
aux[j][k+1]!=queen*turn && aux[j][k+1]!=rook*turn ){                     //Ally piece
found
                                                break;
                                        }
                                        if(MobArea[j][k+1]==1)
```

```
                                        v++;
                                    if(aux[j][k+1]==0){

                                    //Keep sweeping
                                        k++;
                                        continue;
                                    }
                                    else break;

                                        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(k>1){                                     //Left
                                    if(aux[j][k-1]*turn>0 && aux[j][k-
1]!=queen*turn && aux[j][k-1]!=rook*turn){                              //Ally piece found
                                        break;
                                    }
                                    if(MobArea[j][k-1]==1)
                                        v++;
                                    if(aux[j][k-1]==0){

                                    //Keep sweeping
                                        k--;
                                        continue;
                                    }
                                    else break;

                                        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j<8){                                     //Up
                                    if(aux[j+1][k]*turn>0 &&
aux[j+1][k]!=queen*turn && aux[j+1][k]!=rook*turn){                      //Ally piece
found
                                        break;
                                    }
                                    if(MobArea[j+1][k]==1)
                                        v++;
                                    if(aux[j+1][k]==0){

                                    //Keep sweeping
                                        j++;
                                        continue;
                                    }
                                    else break;

                                        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j>1){                                     //Down
                                    if(aux[j-1][k]*turn>0 && aux[j-
1][k]!=queen*turn && aux[j-1][k]!=rook*turn){                           //Ally piece found
                                        break;
                                    }
                                    if(MobArea[j-1][k]==1)
                                        v++;
                                    if(aux[j-1][k]==0){
```

```
                                        //Keep sweeping
                                                j--;
                                                continue;
                                        }
                                        else break;

                                                //Enemy piece found
                                }
                                break;
                        case -5:
                                j=x;
                                k=y;
                                while(j<8 && k<8){
        //Upper Right
                                        if(aux[j+1][k+1]*turn>0 &&
aux[j+1][k+1]!=queen*turn){             //Ally piece found & not a queen to
x-ray
                                                break;
                                        }
                                        if(MobArea[j+1][k+1]==1)
                                                v++;
                                        if(aux[j+1][k+1]==0){
                                                                //Keep sweeping
                                                j++;
                                                k++;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j>1 && k<8){
        //Lower Right
                                        if(aux[j-1][k+1]*turn>0 && aux[j-
1][k+1]!=queen*turn){           //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j-1][k+1]==1)
                                                v++;
                                        if(aux[j-1][k+1]==0){
                                                                //Keep sweeping
                                                j--;
                                                k++;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j<8 && k>1){
        //Upper Left
                                        if(aux[j+1][k-1]*turn>0 && aux[j+1][k-
1]!=queen*turn){                 //Ally piece found
                                                break;
                                        }
```

```
                                        if(MobArea[j+1][k-1]==1)
                                                v++;
                                        if(aux[j+1][k-1]==0){
                                                                        //Keep sweeping
                                                j++;
                                                k--;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j>1 && k>1){
        //Lower Left
                                        if(aux[j-1][k-1]*turn>0 && aux[j-1][k-
1]!=queen*turn){                        //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j-1][k-1]==1)
                                                v++;
                                        if(aux[j-1][k-1]==0){
                                                                        //Keep sweeping
                                                j--;
                                                k--;
                                                continue;
                                        }
                                        else break;

        //Enemy piece found
                                }

                                j=x;
                                k=y;
                                while(k<8){                                     //Right
                                        if(aux[j][k+1]*turn>0 &&
aux[j][k+1]!=queen*turn && aux[j][k+1]!=rook*turn ){                     //Ally piece
found
                                                break;
                                        }
                                        if(MobArea[j][k+1]==1)
                                                v++;
                                        if(aux[j][k+1]==0){

                                        //Keep sweeping
                                                k++;
                                                continue;
                                        }
                                        else break;

                                                //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(k>1){                                     //Left
                                        if(aux[j][k-1]*turn>0 && aux[j][k-
1]!=queen*turn && aux[j][k-1]!=rook*turn){                              //Ally piece found
                                                break;
                                        }
```

```
                                        if(MobArea[j][k-1]==1)
                                                v++;
                                        if(aux[j][k-1]==0){

                                        //Keep sweeping
                                                k--;
                                                continue;
                                        }
                                        else break;

                                                //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j<8){                                     //Up
                                        if(aux[j+1][k]*turn>0 &&
aux[j+1][k]!=queen*turn && aux[j+1][k]!=rook*turn){                      //Ally piece
found
                                                break;
                                        }
                                        if(MobArea[j+1][k]==1)
                                                v++;
                                        if(aux[j+1][k]==0){

                                        //Keep sweeping
                                                j++;
                                                continue;
                                        }
                                        else break;

                                                //Enemy piece found
                                }
                                j=x;
                                k=y;
                                while(j>1){                                     //Down
                                        if(aux[j-1][k]*turn>0 && aux[j-
1][k]!=queen*turn && aux[j-1][k]!=rook*turn){                           //Ally piece found
                                                break;
                                        }
                                        if(MobArea[j-1][k]==1)
                                                v++;
                                        if(aux[j-1][k]==0){

                                        //Keep sweeping
                                                j--;
                                                continue;
                                        }
                                        else break;

                                                //Enemy piece found
                                }
                                break;
                        }
        return v;
}
```