# Penerapan *Dynamic Programming* pada Permainan Catur

Muhammad Rifat Abiwardani 13519205 (*Author*)
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
13519205@std.stei.itb.ac.id

***Abstract*—Sejak terjadinya pandemi Covid-19, permainan catur telah kian banyak menjadi aktivitas yang digiati masyarakat. Catur merupakan permainan papan dua pemain yang menggunakan strategi untuk memenangkan pasukan pemain lawan. Penentuan strategi terbaik menggunakan analisis teoretis dan analisis posisi yang dalam, sehingga sulit bagi masyarakat umum untuk menentukan langkah terbaik dari suatu posisi. Dalam bidang informatika, persoalan langkah terbaik dapat dimodelkan sebagai persoalan optimasi, dan terdapat banyak algoritma optimasi yang bisa digunakan untuk mencari langkah terbaik dalam catur. Salah satu algoritma yang bisa digunakan adalah *dynamic programming*. Makalah ini membahas efektivitas *dynamic programming* untuk mencari langkah terbaik dari suatu posisi dalam catur.**

***Keywords—catur; algoritma; dynamic programming***

## I. PENDAHULUAN

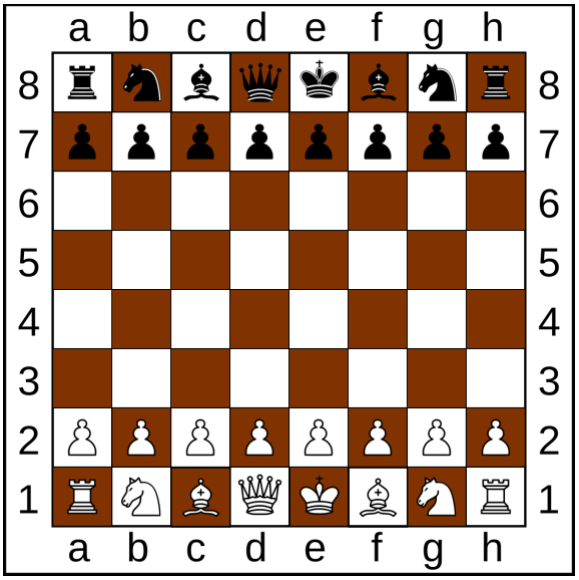
Pada Maret 11 2020, WHO (World Health Organization) resmi menyatakan Covid-19 sebagai pandemi. Pandemi ini menyebabkan pemerintah di seluruh dunia untuk menerapkan *lockdown* dan protokol kesehatan masa pandemi untuk mencegah penyebaran virus secara massal. Salah satu imbas dari kebijakan tersebut adalah banyaknya masyarakat yang terpaksa berdiam di rumah.

Pada 2020, akibat pandemi, mayoritas kegiatan catur berpindah ke daring. Dengan itu pula mulai diadakan turnamen untuk *YouTuber* dan *streamer* seperti *PogChamps* dan *BlockChamps*. Turnamen-turnamen tersebut membuka catur ke khalayak umum sehingga lebih banyak masyarakat terpapar ke dunia catur. Selain itu, 2020 menghadirkan rilis serial TV "Queen's Gambit" yang memikat perhatian banyak orang. Selain itu, masyarakat juga memiliki lebih banyak waktu di rumah menggunakan internet, dan salah satu kegiatan yang menjadi pilihan orang adalah catur. Ketiga faktor tersebut menyebabkan komunitas catur untuk berkembang pesat pada masa pandemi Covid-19.

Gambar 1: Serial TV "Queen's Gambit"

Sumber: www.netflix.com

Catur sendiri adalah permainan papan dua pemain berbasis strategi. Dalam catur, terdapat dua pemain, putih dan hitam, dimana pemain bergantian memainkan langkah-langkahnya dengan putih memulai duluan. Terdapat 32 buah catur pada awal papan, 16 berwarna putih dan 16 berwarna hitam. Untuk masing pemain terdapat 1 raja, 1 menteri, 2 benteng, 2 gajah, 2 kuda, dan 8 pion. Objektif permainan adalah untuk memberi skakmat ke raja lawan dengan cara memainkan langkah-langkah terbaik agar memiliki kondisi dan posisi pasukan yang lebih baik daripada lawan.



Gambar 2: Papan catur

Sumber: https://commons.wikimedia.org/wiki/File:AAA_SVG_Chessboard_and_chess_pieces_02.svg

Pencarian langkah terbaik tidak selalu mudah, dan catur dimainkan pada tingkat tinggi secara global oleh pemain-pemain terbaik. Namun evaluasi langkah terbaik dapat dikuantifikasi sehingga algoritma optimasi dapat digunakan untuk mencarinya. Satu cara adalah dengan menggunakan *dynamic programming*. *Dynamic programming* atau program dinamis adalah metode penyelesaian persoalan menggunakan penguraian solusi per tahap sedemikian sehingga solusi persoalan dapat dilihat sebagai serangkaian transisi simpul status persoalan yang mengalir satu ke berikutnya. Dalam program dinamis digunakan tabulasi atau penghafalan yang isinya dikembangkan secara dinamis, untuk digunakan dalam evaluasi harga optimal.

Dalam makalah ini, penulis menganalisis efektivitas dan kegunaan *dynamic programming* untuk menyelesaikan persoalan pencarian langkah terbaik dalam catur.

## II. LANDASAN TEORI

### A. *Dynamic Programming*

*Dynamic programming* atau program dinamis adalah metode penyelesaian persoalan menggunakan penguraian solusi per tahap sedemikian sehingga solusi persoalan dapat dilihat sebagai serangkaian transisi simpul status persoalan yang mengalir satu ke berikutnya. Program dinamis menggunakan prinsip optimalitas, yang menyatakan bahwa jika solusi total optimal, maka bagian solusi sampai tahap ke-k juga optimal. Akibat dari prinsip ini adalah untuk mencari solusi optimal dari tahap ke-k ke tahap ke-k+1, solusi optimal pada tahap ke-k dapat digunakan tanpa harus menjalankan pencarian dari tahap pertama.

Dalam program dinamis, terdapat dua metode perangkaian tahap: program dinamis maju dan program dinamis mundur. Dalam program dinamis maju, pencarian menggunakan penghafalan atau tabulasi untuk menghitung ongkos pada tahap ke-1, ke-2, hingga tahap ke-n lalu membangun solusi terbaik setelah solusi optimal setiap tahap diperoleh. Sementara itu, program dinamis mundur menghitung terlebih dahulu ongkos pada tahap ke-n, ke-n-1, hingga tahap ke-1, lalu membangun solusi terbaik dari nilai-nilai yang diperoleh.

Untuk membangun algoritma program dinamis, harus didefinisikan beberapa aspek. Pertama, pemodelan persoalan dan karakteristik struktur solusi terbaik. Kedua, mendefinisikan secara rekursif nilai solusi terbaik. Ketiga, nilai solusi terbaik dihitung, baik secara maju maupun secara mundur, menggunakan penghafalan atau tabulasi. Lalu terakhir, solusi terbaik direkonstruksi, baik dengan fungsi maksimasi maupun penentu optimalitas lainnya.

### B. *Catur*

Catur merupakan salah satu permainan papan tertua, pertama dikenal di India pada abad ke-6, lalu menyebar dari Asia ke Timur Tengah dan Eropa pada abad ke-10. Catur merupakan permainan strategi dua pemain pada papan 8x8, dimana masing pemain berusaha memberi skakmat pada raja lawannya menggunakan buah-buah catur yang mengikuti peraturan tertentu. Pada awal permainan, dalam satu papan terdapat 32 buah catur, 16 berwarna putih dan 16 berwarna hitam. Untuk masing pemain terdapat 1 raja, 1 menteri, 2 benteng, 2 gajah, 2 kuda, dan 8 pion. Raja dapat bergerak sebanyak 1 petak ke petak sekitarnya; menteri dapat bergerak sebanyak mungkin petak ke arah horizontal, vertikal, dan diagonal; benteng dapat bergerak sebanyak mungkin petak ke arah horizontal dan vertikal; gajah dapat bergerak sebanyak mungkin petak ke arah diagonal; kuda dapat bergerak sebanyak 2 petak pada horizontal dan 1 petak pada vertikal, atau 2 petak pada vertikal dan 2 petak pada horizontal; sementara pion hanya dapat bergerak 1 petak maju. Namun jika pion belum pernah bergerak, maka pion tersebut dapat bergerak 2 petak maju. Selain itu, pion hanya bisa memakan buah catur lawan pada diagonal maju sejauh 1 petak. Dalam kasus dimana suatu pion dimajukan 2 petak dalam satu langkah lalu menempati petak yang bersebalahan dengan pion musuh, pion musuh tersebut dapat memakan pion yang maju seolah-olah pion hanya maju 1 petak. Pion yang maju lalu mencapai baris terjauh dari baris pemain dapat ditukar untuk buah catur yang bernilai lebih tinggi, kecuali raja. Selain itu, jika raja dan benteng pada baris pertama pemain belum pernah bergerak dan petak di antara kedua buah catur tidak ditempati buah catur lain, serta raja tidak diancam dengan skak atau terkena skak sepanjang 2 petak ke arah benteng, maka raja dapat bertukar sisi dengan benteng dengan bergerak 2 petak ke arah benteng dan benteng berpindah ke 1 petak sebelah raja.

Permainan berakhir jika salah satu pemain memberi skakmat, atau salah satu pemain tidak memiliki langkah yang mungkin, atau kedua pemain mengulangi langkah terakhirnya sebanyak tiga kali.

### C. *Analisis Statis*

Untuk menganalisis suatu posisi, evaluasi statik dapat digunakan. Evaluasi ini terdiri dari dua komponen, evaluasi nilai buah catur dan evaluasi posisi buah catur. Dari www.chessprogramming.org, satu sistem evaluasi yang dapat digunakan sebagai berikut.

Masing buah catur memiliki harga sebagai berikut: raja bernilai 20000, menteri bernilai 900, benteng bernilai 500, gajah bernilai 330, kuda bernilai 320, dan pion bernilai 100.

Posisi pion dievaluasi sebagai berikut:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 50 | 50 | 50 | 50 | 50 | 50 | 50 | 50 |
| 10 | 10 | 20 | 30 | 30 | 20 | 10 | 10 |
| 5 | 5 | 10 | 25 | 25 | 10 | 5 | 5 |
| 0 | 0 | 0 | 20 | 20 | 0 | 0 | 0 |
| 5 | -5 | -10 | 0 | 0 | -10 | -5 | 5 |
| 5 | 10 | 10 | -20 | -20 | 10 | 10 | 5 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Posisi kuda dievaluasi sebagai berikut:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -50 | -40 | -30 | -30 | -30 | -30 | -40 | -50 |
| -40 | -20 | 0 | 0 | 0 | 0 | -20 | -40 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -30 | 0 | 10 | 15 | 15 | 10 | 0 | -30 |
| -30 | 5 | 15 | 20 | 20 | 15 | 5 | -30 |
| -30 | 0 | 15 | 20 | 20 | 15 | 0 | -30 |
| -30 | 5 | 10 | 15 | 15 | 10 | 5 | -30 |
| -40 | -20 | 0 | 5 | 5 | 0 | -20 | -40 |
| -50 | -40 | -30 | -30 | -30 | -30 | -40 | -50 |

Posisi gajah dievaluasi sebagai berikut:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -20 | -10 | -10 | -10 | -10 | -10 | -10 | -20 |
| -10 | 0 | 0 | 0 | 0 | 0 | 0 | -10 |
| -10 | 0 | 5 | 10 | 10 | 5 | 0 | -10 |
| -10 | 5 | 5 | 10 | 10 | 5 | 5 | -10 |
| -10 | 0 | 10 | 10 | 10 | 10 | 0 | -10 |
| -10 | 10 | 10 | 10 | 10 | 10 | 10 | -10 |
| -10 | 5 | 0 | 0 | 0 | 0 | 5 | -10 |
| -20 | -10 | -10 | -10 | -10 | -10 | -10 | -20 |

Posisi benteng dievaluasi sebagai berikut:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 10 | 10 | 10 | 10 | 10 | 10 | 5 |
| -5 | 0 | 0 | 0 | 0 | 0 | 0 | -5 |
| -5 | 0 | 0 | 0 | 0 | 0 | 0 | -5 |
| -5 | 0 | 0 | 0 | 0 | 0 | 0 | -5 |
| -5 | 0 | 0 | 0 | 0 | 0 | 0 | -5 |
| -5 | 0 | 0 | 0 | 0 | 0 | 0 | -5 |
| 0 | 0 | 0 | 5 | 5 | 0 | 0 | 0 |

Posisi menteri dievaluasi sebagai berikut:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -20 | -10 | -10 | -5 | -5 | -10 | -10 | -20 |
| -10 | 0 | 0 | 0 | 0 | 0 | 0 | -10 |
| -10 | 0 | 5 | 5 | 5 | 5 | 0 | -10 |
| -5 | 0 | 5 | 5 | 5 | 5 | 0 | -5 |
| 0 | 0 | 5 | 5 | 5 | 5 | 0 | -5 |
| -10 | 5 | 5 | 5 | 5 | 5 | 0 | -10 |
| -10 | 0 | 5 | 0 | 0 | 0 | 0 | -10 |
| -20 | -10 | -10 | -5 | -5 | -10 | -10 | -20 |

Posisi raja pada tahap tengah permainan dievaluasi sebagai berikut:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -30 | -40 | -40 | -50 | -50 | -40 | -40 | -30 |
| -30 | -40 | -40 | -50 | -50 | -40 | -40 | -30 |
| -30 | -40 | -40 | -50 | -50 | -40 | -40 | -30 |
| -30 | -40 | -40 | -50 | -50 | -40 | -40 | -30 |
| -20 | -30 | -30 | -40 | -40 | -30 | -30 | -20 |
| -10 | -20 | -20 | -20 | -20 | -20 | -20 | -10 |
| 20 | 20 | 0 | 0 | 0 | 0 | 20 | 20 |
| 20 | 30 | 10 | 0 | 0 | 10 | 30 | 20 |

Posisi raja pada tahap akhir permainan dievaluasi sebagai berikut:

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| -50 | -40 | -30 | -20 | -20 | -30 | -40 | -50 |
| -30 | -20 | -10 | 0 | 0 | -10 | -20 | -30 |
| -30 | -10 | 20 | 30 | 30 | 20 | -10 | -30 |
| -30 | -10 | 30 | 40 | 40 | 30 | -10 | -30 |
| -30 | -10 | 30 | 40 | 40 | 30 | -10 | -30 |
| -30 | -10 | 20 | 30 | 30 | 20 | -10 | -30 |
| -30 | -30 | 0 | 0 | 0 | 0 | -30 | -30 |
| -50 | -30 | -30 | -30 | -30 | -30 | -30 | -50 |

Nilai evaluasi suatu posisi adalah total nilai bagi buah catur putih dikurangi total nilai bagi buah catur hitam.

## III. Analisis dan Pembahasan

### A. *Model Permasalahan*

Untuk memodelkan persoalan catur, setiap posisi dianggap sebagai simpul. Untuk membantu enumerasi, setiap posisi memiliki kode FEN yang unik yang mengkodekan posisi setiap buah catur dan warna setiap buah catur serta kondisi giliran dan legalitas *castling*. Untuk berpindah dari satu simpul ke simpul lain, langkah digunakan sebagai transisi. Langkah inilah yang dianalisis untuk dicari keputusan terbaik dari semua langkah yang mungkin. Untuk mengevaluasi ongkos akhir, digunakan analisis statik dimana pemain putih berusaha memaksimalkan nilai evaluasi dan pemain hitam berusaha meminimalkan nilai evaluasi. Sementara *dynamic programming* digunakan untuk menyimpan nilai evaluasi setiap kode FEN agar program tidak harus menghitung nilai suatu posisi dua kali atau lebih.

### B. *Implementasi Dynamic Programming*

Berikut implementasi *dynamic programming* untuk pencarian langkah terbaik. Untuk menghindari pencarian mendalam yang berat secara *resource*, program mengadakan kedalaman maksimal, karena jumlah node bertambah secara eksponensial:

```
import chess
import chess.engine
import datetime

def eval_board(cboard):
    #hardcode nilai masing buah catur
    #raja bernilai 20000, menteri bernilai 900, bento
bernilai 500, gajah bernilai 330, kuda bernilai 320, p
bernilai 100
    piece_values = {"K": 20000, "Q": 900, "R": 500, "B": 3
"N": 320, "P": 100}

    #nilai          bobot          posisi          berdasar
https://www.chessprogramming.org/Simplified_Evaluation_Funct
    pawn_weights = [[0, 0, 0, 0, 0, 0, 0, 0], [50, 50, 50,
50, 50, 50, 50, 50], [10, 10, 20, 30, 30, 20, 10, 10], [5, 5,
25, 25, 10, 5, 5], [0, 0, 0, 20, 20, 0, 0, 0], [5, -5, -10,
0,-10, -5, 5], [5, 10, 10,-20,-20, 10, 10, 5], [0, 0, 0, 0,
0, 0, 0]]
    knight_weights = [[-50,-40,-30,-30,-30,-30,-40,-50], [-4
20, 0, 0, 0, 0,-20,-40], [-30, 0, 10, 15, 15, 10, 0,-30], [-
5, 15, 15, 15, 15, 5,-30], [-30, 0, 15, 15, 15, 15, 0,-30],
30, 5, 10, 15, 15, 10, 5,-30], [-40,-20, 0, 0, 0, 0,-20,-4
[-50,-40,-30,-30,-30,-30,-40,-50]]
    bishop_weights = [[-20,-10,-10,-10,-10,-10,-10,-20], [-
0, 0, 0, 0, 0, 0,-10], [-10, 0, 5, 10, 10, 5, 0,-10], [-10,
5, 10, 10, 5, 5,-10], [-10, 0, 10, 10, 10, 10, 0,-10], [-
10, 10, 10, 10, 10, 10,-10], [-10, 5, 0, 0, 0, 0, 5,-10],
```

```
20,-10,-10,-10,-10,-10,-10,-20]]
    rook_weights = [[0, 0, 0, 0, 0, 0, 0, 0], [5, 10, 10, 10,
10, 10, 10,  5], [-5, 0, 0, 0, 0, 0, 0, -5], [-5, 0, 0, 0, 0,
0, 0, -5], [-5, 0, 0, 0, 0, 0, 0, -5], [-5, 0, 0, 0, 0, 0, 0, -
5], [-5, 0, 0, 0, 0, 0, 0, -5], [0, 0, 0, 5, 5, 0, 0, 0]]
    queen_weights = [[-20,-10,-10, -5, -5,-10,-10,-20], [-10,
0, 0, 0, 0, 0, 0,-10], [-10, 0, 5, 5, 5, 5, 0,-10], [-5, 0, 5,
5, 5, 5, 0, -5], [0, 0, 5, 5, 5, 5, 0, -5], [-10, 5, 5, 5, 5,
5, 0,-10], [-10, 0, 5, 0, 0, 0, 0,-10], [-20,-10,-10, -5, -5,-
10,-10,-20]]
    king_mid_weights = [[-30,-40,-40,-50,-50,-40,-40,-30], [-
30,-40,-40,-50,-50,-40,-40,-30], [-30,-40,-40,-50,-50,-40,-40,-
30], [-30,-40,-40,-50,-50,-40,-40,-30], [-20,-30,-30,-40,-40,-
30,-30,-20], [-10,-20,-20,-20,-20,-20,-20,-10], [20, 20, 0, 0,
0, 0, 20, 20], [20, 30, 10, 0, 0, 10, 30, 20]]
    king_end_weights = [[-50,-40,-30,-20,-20,-30,-40,-50], [-
30,-20,-10,  0,  0,-10,-20,-30], [-30,-10, 20, 30, 30, 20,-10,-
30], [-30,-10, 30, 40, 40, 30,-10,-30], [-30,-10, 30, 40, 40,
30,-10,-30], [-30,-10, 20, 30, 30, 20,-10,-30], [-30,-30, 0, 0,
0, 0,-30,-30], [-50,-30,-30,-30,-30,-30,-30,-50]]

    #inisialisasi variabel
    minor_pc_count = 0
    queen_count = 0
    score = 0

    #hitung jumlah ratu dan jumlah kuda atau gajah untuk
penentuan tahap permainan
    for i in range(64):
        piece = str(cboard.piece_at(i))
        if (piece != "None"):
            if (piece.upper() == "Q"):
                queen_count += 1
            elif (piece.upper() == "B" or piece.upper() ==
"N"):
                minor_pc_count += 1

    #jika satu ratu sudah hilang dan jumlah kuda atau gajah
kurang dari sama dengan 2
    endgame = queen_count <= 1 and minor_pc_count <= 2

    #hitung nilai evaluasi setiap petak
    for i in range(8):
        for j in range(8):
            k = i + j*8
            piece = str(cboard.piece_at(k))

            #jika ada buah catur di petak tersebut
            if (piece != "None"):
                #hitung nilai buah catur
                if (piece == piece.upper()):
                    score += piece_values[piece]
                else:
                    score -= piece_values[piece.upper()]

                #hitung nilai posisi yang ditempati buah catur
tersebut
                if (piece == "K"):
                    if (endgame):
                        score += king_end_weights[7-i][j]
                    else:
                        score += king_mid_weights[7-i][j]
                elif (piece == "k"):
                    if (endgame):
                        score -= king_end_weights[i][j]
                    else:
                        score -= king_mid_weights[i][j]
                elif (piece == "Q"):
                    score += queen_weights[7-i][j]
                elif (piece == "q"):
                    score -= queen_weights[i][j]
                elif (piece == "R"):
                    score += rook_weights[7-i][j]
                elif (piece == "r"):
                    score -= rook_weights[i][j]
                elif (piece == "B"):
                    score += bishop_weights[7-i][j]
                elif (piece == "b"):
                    score -= bishop_weights[i][j]
                elif (piece == "N"):
                    score += knight_weights[7-i][j]
                elif (piece == "n"):
                    score -= knight_weights[i][j]
                elif (piece == "P"):
                    score += pawn_weights[7-i][j]
                elif (piece == "p"):
                    score -= pawn_weights[i][j]

    return score
class chess_engine_DP:
    def __init__(self, board, player, max_depth):
        self.board = board
        self.player = player
        self.max_depth = max_depth
        self.costs = {}

    def move(self, move_san):
        self.board.push_san(move_san)

    def best_move(self):
        #inisialisasi variabel
        self.costs = {}
        fens = []
        depth = 0

        #transisi simpul yang mungkin
        moves = [self.board.san(move) for move
self.board.legal_moves]

        #evaluasi rekursif untuk setiap kemungkinan simpul
        for move in moves:
            self.board.push_san(move)
            fens.append(self.board.fen())
            self.dynamic_programming_eval(self.board,
depth+0.5, not self.player)
            self.board.pop()

        #cari nilai evaluasi terbesar jika permain berma
putih, terkecil jika hitam
        evals = [self.costs[str(not self.player)[0]+fen]
fen in fens]
        best_score = 0

        if (self.player):
            best_score = max(evals)
        else:
            best_score = min(evals)

        self.costs[str(self.player)[0]+self.board.fen()]
best_score

        #kembalikan langkah yang menghasilkan nilai evalu
terbaik tersebut
        for i in range(len(moves)):
            if (evals[i] == best_score):
                return moves[i]

    def dynamic_programming_eval(self, cboard, depth, turn
chess.WHITE):
        #jika sudah di ujung kedalaman, tetapkan cost seba
nilai evaluasi posisi kini
        if (depth == self.max_depth):
            self.costs[str(turn)[0]+cboard.fen()]
eval_board(cboard)
        #jika posisi belum dihitung costnya, evaluasi rekur
```

```
untuk setiap kemungkinan simpul
        elif (str(turn)[0]+cboard.fen() not in self.costs):
            moves    =    [cboard.san(move)    for    move    in
cboard.legal_moves]

            fens = []

            for move in moves:
                cboard.push_san(move)
                fens.append(cboard.fen())
                self.dynamic_programming_eval(cboard,
depth+0.5, not turn)
                cboard.pop()

            evals  =  [self.costs[str(not  turn)[0]+fen]  for  fen
in fens]

            #jika skakmat
            if (board.is_checkmate()):
                if (turn):
                    self.costs[str(turn)[0]+cboard.fen()]  =
999999
                else:
                    self.costs[str(turn)[0]+cboard.fen()]  =
999999
            else: #jika tidak
                if (turn):
                    self.costs[str(turn)[0]+cboard.fen()]  =
max(evals)
                else:
                    self.costs[str(turn)[0]+cboard.fen()]  =
min(evals)
```

*C. Metode Analisis*

Variabel dalam analisis adalah kedalaman pencarian *dynamic programming*, kedalaman mesin Stockfish, dan jenis *opening*/pembuka. Parameter performa algoritma adalah rata-rata waktu mencari langkah dan jumlah langkah total permainan.

Berikut contoh eksperimen untuk permainan dengan pembuka “1. e4 2. c5”:

```
board = chess.Board()

#opening
board.push_san("d4")
board.push_san("e6")

#inisialisasi engine dynamic programming (pemain) dan
engine stockfish (lawan)
my_engine = chess_engine_DP(board, chess.WHITE, 1.5)
engine = chess.engine.SimpleEngine.popen_uci("stockfish")
limit = chess.engine.Limit(time=1)

nmoves = 0
time = 0
#selama belum skakmat
while (not my_engine.board.is_checkmate()):
  #mulai
  nmoves += 1
  start = datetime.datetime.now()

  #cari langkah terbaik dengan dynamic programming
  my_engine.board.push_san(my_engine.best_move())
  print(eval_board(my_engine.board))

  #cetak waktu yang dibutuhkan
  end = datetime.datetime.now()
  diff = (end-start).seconds
  time += diff
  print(diff)

  #stockfish memainkan langkah terbaik
  playres = engine.play(my_engine.board, limit)
  my_engine.move(my_engine.board.san(playres.move))
  print(eval_board(my_engine.board))

  #cetak papan
  print(board)
  print("----------")
print(nmoves)
print(time/ nmoves)
```

*D. Hasil Analisis*

1. Percobaan ke-1

Pertama dicoba kedalaman pencarian *dynamic* programming bernilai 7, yang memakan waktu lebih dari 30 menit per langkah, sehingga kedalaman ini tidak cocok untuk digunakan sebagai eksperimen.

2. Percobaan ke-2

Kedua dicoba kedalaman 2 yang rata-rata waktunya adalah 485 detik per langkah. Ini juga terlalu lama untuk digunakan sebagai eksperimen, sehingga nilai kedalaman harus dikurangkan.

3. Percobaan ke-3

Ketiga dicoba kedalaman 1,5, yang rata-rata waktunya adalah 20,351 detik per langkah. Ini cukup pendek dan dapat digunakan sebagai eksperimen

4. Percobaan ke-4

Pada kedalaman 1,5, dicoba pembuka “1. e4 e5” sebagai putih. Dari eksperimen, diperoleh rata-rata waktu 24,238 detik per langkah. Solusi program dinamis bertahan sebanyak 21 langkah melawan Stockfish dengan waktu pemrosesan 1 detik, dan pada ujungnya, Stockfish memberi skakmat.

5. Percobaan ke-5

Pada kedalaman 1,5, dicoba pembuka “1. d4 e6” sebagai putih. Dari eksperimen, diperoleh rata-rata waktu 18,535 detik per langkah. Solusi program dinamis bertahan sebanyak 28 langkah melawan Stockfish dengan waktu pemrosesan 1 detik, dan pada ujungnya, Stockfish kembali memberikan skakmat.

## IV. Kesimpulan

Dari hasil analisis, dapat disimpulkan bahwa program dinamis tidak cocok untuk menyelesaikan persoalan catur. Ini dapat dilihat dari nilai kedalaman yang harus ditetapkan sebagai cukup rendah untuk menghemat waktu, dan hasil permainan yang cukup buruk, yaitu kalah dalam rentang 21 hingga 28 langkah. Oleh karena itu, program dinamis tidak cocok digunakan untuk mencari langkah terbaik dalam catur.

## V. Saran

### *A. Saran Pengembangan*

Untuk pengembangan kedepan, algoritma lain dapat dicoba untuk memperoleh hasil yang lebih baik, dengan kedalaman yang lebih dalam.

## VI. Referensi


[1] Bostock, B. (2021, March 18). *'The Queen's gambit' and the pandemic injected new life into the multimillion-dollar chess industry, with esports teams and sponsors rushing to snap up the game's Twitch stars*. Insider. https://www.insider.com/chess-boom-stars-get-new-esports-contracts-sponsorships-2021-3

[2] *Chess*. (n.d.). Encyclopedia Britannica. https://www.britannica.com/topic/chess

[3] (n.d.). Informatika. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Program-Dinamis-2020-Bagian1.pdf

[4] *Python-chess: A chess library for Python — Python-chess 1.5.0 documentation*. (n.d.). python-chess: a chess library for Python — python-chess 1.5.0 documentation. https://python-chess.readthedocs.io/en/latest/

[5] *Simplified evaluation function*. (n.d.). Chessprogramming wiki. Retrieved May 11, 2021, from https://www.chessprogramming.org/Simplified_Evaluation_Function

[6] *WHO director-general's opening remarks at the media briefing on COVID-19 - 11 March 2020*. (n.d.). WHO | World Health Organization. https://www.who.int/director-general/speeches/detail/who-director-general-s-opening-remarks-at-the-media-briefing-on-covid-19---11-march-2020


## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Depok, 11 Mei 2021

Muhammad Rifat Abiwardani 13519205