

# Tugas Besar I

IF3170 Inteligensi Buatan

Minimax Algorithm and Alpha Beta Pruning in Adjacency Strategy Game



Disusun oleh:

13521087 Razzan Daksana Yoni

13521089 Kenneth Ezekiel Supranton

13521095 Muhamad Aji Wibisono

13521101 Arsa Izdiyar Islam

**Sekolah Teknik Elektro dan Informatika**

**Institut Teknologi Bandung**

**2023**

## DAFTAR ISI

<b>BAB I PENDAHULUAN</b>	<b>3</b>
1.1. Latar belakang	3
1.2. Aturan Main	3
1.3. Implementasi	4
1.4. Spesifikasi Pengerjaan Tugas	4
<b>BAB II ISI</b>	<b>6</b>
2.1. Fungsi Objektif	6
2.1.1. Penjelasan	6
2.2. Penjelasan Algoritma	6
2.2.1. Minimax Alpha-Beta Pruning	6
2.2.2. Local Search	9
2.2.3. Genetic Algorithm	10
2.3. Implementasi	12
2.3.1. Struktur Folder	12
2.3.2. Objective Function	12
2.3.3. Minimax Alpha-Beta Pruning	13
2.3.4. Local Search	15
2.3.5. Genetic Algorithm	16
2.3.6. Pendukung	20
<b>BAB III EXPERIMENT</b>	<b>34</b>
3.1. Eksperimen	34
3.1.1. Bot Minimax vs. Human	34
3.1.2. Bot Local Search vs. Human	35
3.1.3. Bot Minimax vs. Bot Local Search	37
3.1.4. Bot Minimax vs. Bot Genetic Algorithm	38
3.1.5. Bot Local Search vs. Bot Genetic Algorithm	39
3.2. Analisis	40
3.2.1. Bot Minimax	40
3.2.2. Bot Local Search	41
3.2.3. Bot Genetic Algorithm	42
<b>BAB IV PEMBAGIAN TUGAS</b>	<b>43</b>
<b>BAB V LAMPIRAN</b>	<b>43</b>

# BAB I

## PENDAHULUAN

### 1.1. Latar belakang

Tugas Besar I pada kuliah IF3170 Inteligensi Buatan bertujuan agar peserta kuliah mendapatkan wawasan tentang implementasi algoritma MiniMax pada suatu bentuk permainan yang memanfaatkan adversarial search. Pada tugas kecil kali ini, permainan yang akan digunakan adalah Adjacency Strategy Game. Secara singkat, Adjacency Strategy Game adalah suatu permainan dimana pemain perlu menempatkan marka (O atau X) pada papan permainan dengan tujuan memperoleh marka sebanyak mungkin pada akhir permainan (dengan jumlah ronde yang telah ditetapkan).

### 1.2. Aturan Main

Aturan permainan Adjacency Strategy Game yang perlu diikuti adalah:

- Permainan dimainkan pada papan 8 x 8, dengan dua jenis pemain O dan X.
- Pada awal permainan, terdapat 4 X di pojok kiri bawah, dan 4 O di pojok kanan atas.
- Secara bergantian pemain X dan pemain O akan menaruh markanya di kotak kosong. Ketika sebuah kotak kosong diisi, seluruh kotak di sekitar yang sudah terisi marka musuh akan berubah menjadi marka pemain. Misal:

	O	
X		
X	X	O
X	X	X

- Lalu O mengisi board[1][1]

	O	
O	O	
O	O	O
X	X	X

Perhatikan bahwa kotak pada arah diagonal berubah ketika kotak kosong diisi O.

- Permainan selesai ketika papan penuh atau mencapai batas ronde yang telah ditetapkan.
- Pemenang adalah yang pemain yang memiliki marka terbanyak pada papan.

### 1.3. Implementasi

Anda diminta untuk mengimplementasikan bot, yaitu:

- a. [Poin 80] Bot dengan algoritma minimax alpha beta pruning.
- b. [Poin 20] Bot dengan salah satu algoritma local search.
- c. [Poin 10] Bot dengan Genetic Algorithm search.

### 1.4. Spesifikasi Pengerjaan Tugas

- Tugas dikerjakan berkelompok, dan 1 kelompok terdiri atas 4 mahasiswa (gabungan 2 kelompok tugas kecil).
- Tugas dikumpulkan seperti repository yang sudah ada. Hanya saja, ditambah dengan folder docs yang berisi laporan.
- Laporan dalam format .pdf berisi informasi sbb:
  - Penjelasan mengenai objective function yang digunakan
  - Penjelasan proses pencarian dengan minimax dan alpha beta pruning yang dilakukan pada permainan Adjacency Strategy Game.
  - Jika mengimplementasikan, penjelasan algoritma local search yang digunakan. Berikan juga justifikasi mengapa algoritma tersebut dipilih dibandingkan dengan algoritma-algoritma yang lain.
  - Jika mengimplementasikan, penjelasan Genetic Algorithm yang digunakan.
  - Penjelasan hasil pertandingan yang dilakukan antara:
    - Bot minimax vs. manusia (sebanyak 5 kali)
    - Jika mengimplementasikan, Bot local search vs. manusia (sebanyak 3 kali)
    - Jika mengimplementasikan, Bot minimax vs bot local search (sebanyak 3 kali)
    - Jika mengimplementasikan, Bot minimax vs bot genetic algorithm (sebanyak 3 kali)

- Jika mengimplementasikan, Bot local search vs bot genetic algorithm (sebanyak 3 kali)
- Catat jumlah kemenangan dan kekalahan untuk setiap jenis pertandingan untuk mengetahui persentase kemenangan setiap jenis bot.
- Untuk percobaan, minimal 1 percobaan dengan jumlah ronde maksimal atau sampai board penuh. Kemudian, untuk percobaan lainnya minimal 8 ronde. Contoh: Dalam 3 kali percobaan, direkam: 24 ronde, 8 ronde, 12 ronde.
  - Kontribusi setiap anggota dalam kelompok
- Penamaan file yang dikumpulkan: Tubes1\_[NIM anggota terkecil].pdf/zip (misal: Tubes1\_13521001.pdf/zip). Pengumpulan hanya dilakukan oleh NIM terkecil.
- Pengumpulan yang terlambat tidak diperbolehkan, batas akhir adalah hari Rabu, 18 Oktober 2023 pukul 23.00 WIB. Pengumpulan dilakukan oleh NIM terkecil.
- Dilarang bekerja sama antar kelompok, kecurangan akan berakibat nilai E pada mata kuliah IF3170.

## **BAB II**

### **ISI**

#### **2.1. Fungsi Objektif**

##### **2.1.1. Penjelasan**

Fungsi objektif merupakan fungsi yang nilainya akan dioptimalkan, bisa bernilai maksimum atau bisa juga bernilai minimum, tergantung dari kasusnya. Dalam konteks Inteligensi Buatan, fungsi objektif merujuk pada tujuan atau target

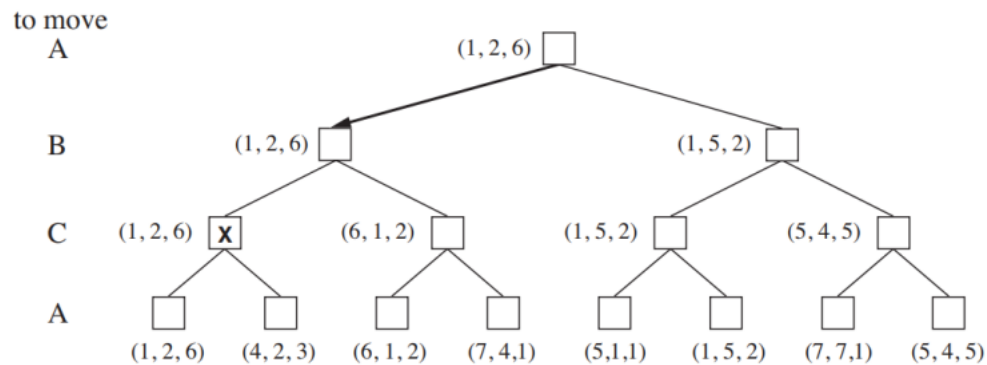
yang ingin dicapai atau dioptimalkan. Fungsi objektif ini juga digunakan untuk mengukur kinerja model atau algoritma yang dikembangkan dalam mencapai tujuan tertentu dalam tugas yang diberikan.

## 2.2. Penjelasan Algoritma

### 2.2.1. Minimax Alpha-Beta Pruning

Minimal Alpha-Beta Pruning adalah algoritma pencarian yang mencari nilai berdasarkan Algoritma Minimax dan mengurangi jumlah node menggunakan Alpha-Beta Pruning. Algoritma Minimax adalah algoritma yang memperhitungkan setiap langkah dengan berselingan mencari minimum dan maksimum dari gerakan state yang dapat diraih secara rekursif untuk mendapatkan gerakan optimal yang dapat dilakukan saat ini.

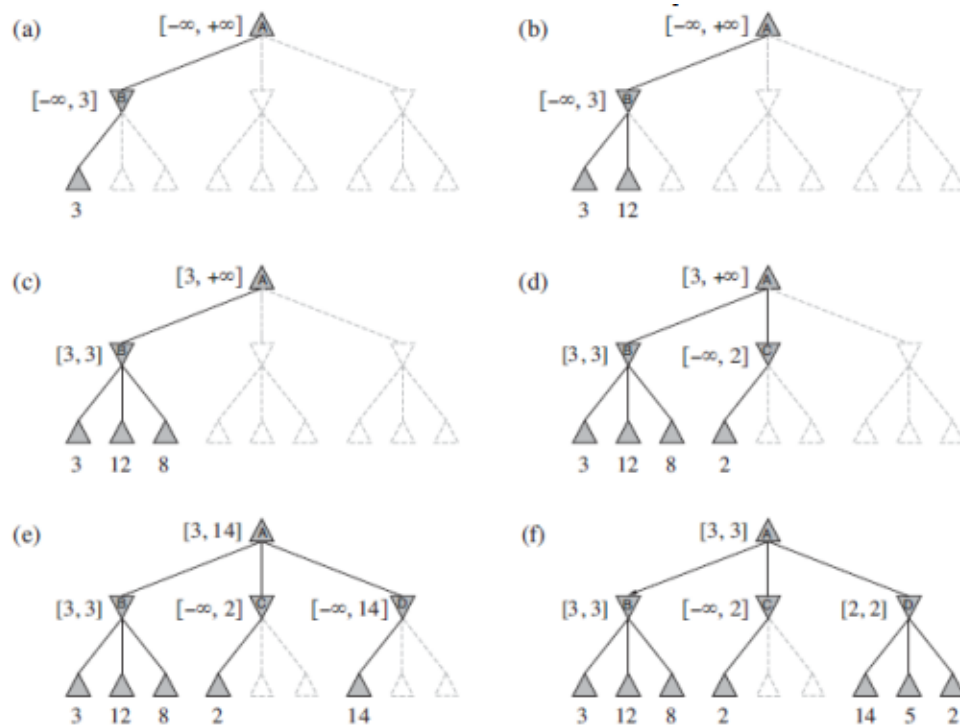
Dalam permainan multi pemain, Algoritma Minimax digunakan untuk memperhitungkan keputusan terbaik yang diambil oleh pemain dalam permainan yang menggunakan gerakan berselingan secara diskrit. Perhitungan max memperhitungkan gerakan yang pemain tersebut lakukan, sementara gerakan min akan memperhitungkan gerakan yang pemain lawan lakukan.



Gambar 2.2.11. Ilustrasi Algoritma Minimax

Masalah dari Algoritma Minimax adalah perkembangan jumlah *node* dalam tiap pergerakan yang berkembang secara eksponensial. Hal ini menyebabkan pencarian menjadi lama. Masalah eksponen ini tidak dapat dihilangkan sepenuhnya, namun terdapat teknik yang memperhitungkan keputusan minimax untuk semua *node* yang terdapat pada pencarian. Cara yang dilakukan adalah dengan *pruning*, yaitu dengan memotong *node* yang sudah tidak mungkin mengubah keputusan akhir dari perhitungan yang telah dilakukan sejauh itu. Tekniknya adalah Alpha-Beta Pruning, yaitu dengan menyimpan nilai alpha

sebagai nilai terbaik (umumnya tertinggi) yang ditemukan sejauh itu dalam jalur max, dan menyimpan nilai beta sebagai nilai terbaik (umumnya terendah) yang ditemukan sejauh itu dalam jalur min. Jika dalam suatu cabang *node* didapatkan nilai beta sudah lebih besar sama dengan alpha maka cabang tersebut dapat dipotong (*prune*) dan bercabangan berikutnya tidak diperhitungkan lagi.



Gambar 2.21.2. Ilustrasi Algoritma Minimax dengan Alpha-Beta Pruning

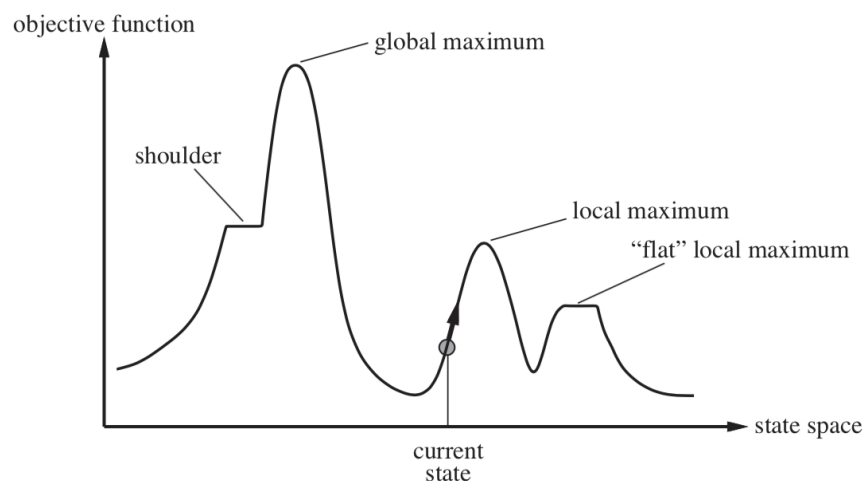
Alpha-Beta Pruning dapat dilakukan ke sebuah *tree* dengan kedalaman berapapun dan dapat mengurangi node dengan memotong *subtree* secara keseluruhan dan tidak hanya daun dari *tree* tersebut. Efektivitas dari Alpha-Beta Pruning sangat bergantung kepada urutan node yang diperiksa. Jika urutan node yang diperiksa baik, Alpha-Beta Pruning dapat mengurangi pencarian dengan signifikan, sebaliknya jika urutan node yang diperiksa tidak baik.

Meskipun sudah mengimplementasikan Alpha-Beta Pruning, secara *performance*, Algoritma Minimax masih memiliki permasalahan apabila *depth* dari *backtracking* algoritma ini besar. Contohnya pada kasus permainan ini, apabila *board* berukuran 8x8 dan *depth* hingga 10, maka worst case bisa hingga  $64^{10}$  yang akan memakan waktu komputasi yang lama. Maka dari itu, perlu diberlakukan *fallback* apabila maksimal waktu yang diwajibkan adalah 5 detik.

Metode *fallback* yang kami lakukan adalah dengan menggunakan default *depth* 8. Apabila selama 4,5 detik komputasi belum selesai, maka akan *fallback* dengan menggunakan *depth* 4 dan *depth* untuk *round* selanjutnya akan dikurangi 1. Namun, jika pada suatu waktu komputasi algoritma utama selesai dalam waktu kurang dari 3 detik, *depth* untuk *round* selanjutnya akan ditambahkan 1.

### 2.2.2. Local Search

Local Search adalah algoritma pencarian yang mencari nilai berdasarkan konfigurasi yang ada di sekitarnya pada saat ini dengan membandingkan nilai pada saat ini dengan nilai tetangganya tanpa menyimpan jalur yang telah dilaluinya. Yang berarti pencarian tidak sistematis karena mungkin ada jalur yang tidak ditempuh/dicari yang padahal mungkin jalur tersebut memiliki solusi. Algoritma Local Search dapat membantu menyelesaikan permasalahan optimasi yang bertujuan untuk mencari kondisi terbaik saat ini berdasarkan *objective function*. Ruang penyelesaian dengan Algoritma Local Search dapat digambarkan dengan gambar berikut.



Gambar 2.2.2.1 Ruang masalah di mana ketinggian sesuai dengan fungsi obyektifnya

Dapat dilihat bahwa Algoritma Local Search bertujuan untuk mencari titik tertinggi berdasarkan fungsi objektifnya yang biasanya proses ini disebut dengan *hill climbing*. Algoritma Pencarian Hill Climbing akan melacak keadaan saat ini dengan mendaki bukit dan di setiap iterasinya akan pindah ke *neighbor state* dengan nilai tertinggi yaitu yang memberikan pendakian paling curam. Pencarian akan berakhir ketika sudah mencapai “puncak” pendakian yang paling curam(*steepest ascent*) yaitu ketika tidak ada tetangga yang memiliki nilai



objektif yang lebih tinggi dari state saat ini. Algoritma Pencarian Hill Climbing juga tidak lanjut mencari pencarian setelah mendapatkan tetangga dari *current state*.

Pemilihan Algoritma Pencarian Hill Climbing dengan metode *steepest ascent* untuk permainan Adjacency Strategy Game dipilih dikarenakan algoritma ini lebih masuk akal dibandingkan algoritma lain yaitu memilih objective function tetangga yang lebih tinggi dibanding objective function state saat ini karena berdasarkan objective function yang dipilih akan selalu menghasilkan nilai objective yang lebih tinggi daripada nilai objective saat ini. Algoritma lain seperti Hill Climbing Sideways Move tidak dipilih karena berdasarkan objective function dipilih tidak mungkin menghasilkan nilai objective yang sama dengan berdasarkan *neighbor state* dan *current state*. Algoritma Pencarian Random Restart Hill Climbing juga tidak mungkin karena tidak memungkinkannya kembali ke *state* sebelumnya. Algoritma Pencarian Stochastic Hill Climbing juga dapat memungkinkan *state* tidak bergerak kemanapun jika ketika random hingga ke maksimum iterasi tidak pernah menemukan nilai objective tetangga yang lebih tinggi daripada nilai objective saat ini. Oleh karena itu, algoritma yang dipilih adalah Algoritma Pencarian Hill Climbing dengan metode *steepest ascent*.

### 2.2.3. Genetic Algorithm

Genetic Algorithm adalah algoritma pencarian yang mendapatkan solusi menggunakan prinsip evolusi, dimana prinsipnya adalah jika solusi yang baik diperanakan dengan solusi yang baik juga, dengan sedikit mutasi, akan menghasilkan solusi yang terbaik (global optimum) diberikan generasi yang cukup. Genetic Algorithm dimulai dari populasi awal, kemudian secara iteratif memilih, menggabungkan, dan memodifikasi solusi-solusi untuk mencapai solusi yang lebih baik.

Genetic Algorithm biasa digunakan untuk mencari global optimum, tetapi tidak biasa untuk nilai minimax yang iteratif. Sehingga, oleh paper riset oleh T.P. Hong et al, Adversarial Search by Evolutionary Computation, perlu dirumuskan sebuah Genetic Algorithm untuk nilai minimax yang iteratif, dinamakan Genetic Minimax.

Perumusan Genetic Minimax yang digunakan adalah sebagai berikut:

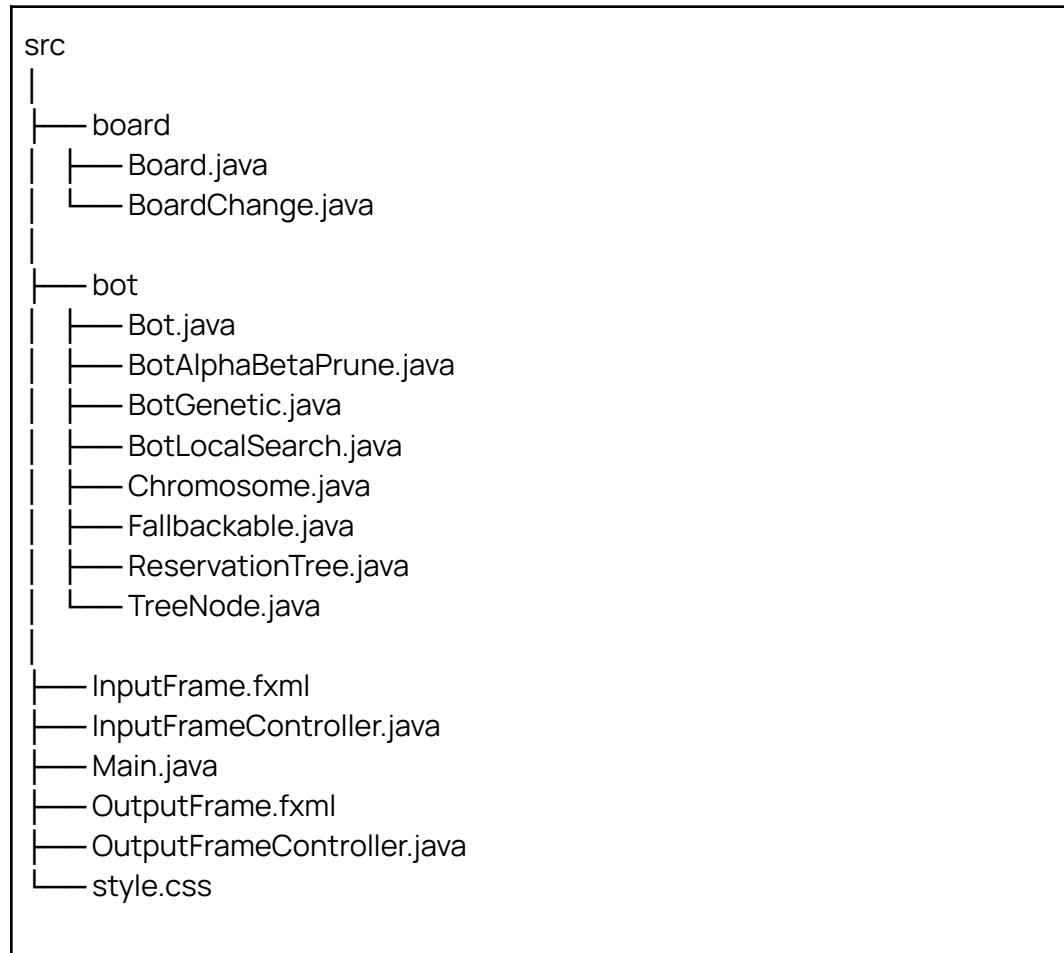
- *Encode* permainan ke dalam *solution state* yang sesuai dengan karakteristik permainan

- Kromosom menandakan langkah-langkah yang sudah diambil
- Panjang dari kromosom ditentukan dari kedalaman search tree
- *Crossover* dapat mengadopsi banyak cara, misalkan simple *one-point crossover*, dimana *parents* dapat saling bertukar *substring* dari kromosom
- *Mutation* dapat terjadi secara acak untuk mengubah elemen di dalam kromosom pada individu acak yang akan membantu keluar dari local optima
- Sebuah mekanisme untuk mempertahankan semua fitness value dari semua kromosom, yaitu reservation tree, yang akan dievaluasi dengan prinsip minimax
- *Reservation tree* akan sekaligus menjadi mekanisme untuk *breeding* dari *parents*, dengan menggabungkan *Least Common Ancestor* dari kedua kromosom, lalu memberikan dua *child* yang akan menjadi titik perbedaan
- *Fitness evaluation* dilakukan dengan melihat simpul daun mana yang nilainya dapat dipropagasikan dengan prinsip minimax paling panjang
- *Fitness function* harus memperhitungkan prinsip minimax, sehingga suatu kromosom perlu dibandingkan dengan kromosom lainnya di dalam populasi menggunakan *reservation tree* untuk mendapatkan nilai *fitness* yang akurat.
- *Fitness function* yang digunakan adalah ketinggian yang dapat dicapai dalam *reservation tree* oleh nilai dari sebuah simpul daun

## 2.3. Implementasi

### 2.3.1. Struktur Folder

Berikut adalah struktur dari folder src



### 2.3.2. Objective Function

Objective Function yang digunakan dalam implementasi adalah dengan memperhitungkan selisih dari simbol yang dimiliki oleh agen atau pemain dengan simbol yang dimiliki oleh lawan agen atau pemain.

Dalam Minimax Alpha-Beta Pruning, Objective Function diperhitungkan dengan mendapatkan nilai awal dari Oscore dan Xscore yang diperhitungkan oleh user interface, lalu melakukan passing ke state yang di dalamnya. Dengan demikian, perhitungan Objective Function pada state hanya dilakukan dengan menambahkan perubahan nilai Objective Function pada setiap gerakan yang

menuju state tersebut ke Objective Function awal yang didapatkan. Hal ini dilakukan agar tidak perlu menghitung ulang jumlah simbol pada papan sehingga mempercepat perhitungan dan penggunaan memori.

### 2.3.3. Minimax Alpha-Beta Pruning

Implementasi Minimax Alpha-Beta Pruning terdapat pada file BotAlphaBetaPrune.java di fungsi move dan minimax. Karena pada tugas ini terdapat batasan waktu dalam bergerak, yaitu 5 detik, terdapat fallback yang dilakukan oleh implementasi ini yaitu dengan melakukan move dengan depth yang lebih rendah. Jika fallback terjadi, depth akan berkurang, sementara itu jika fungsi diselesaikan dengan waktu cepat, depth akan bertambah

```
public int[] move(int Xscore, int Oscore, int maxDepth) {
    int diff;
    if (ownSymbol == 'X')
        diff = Xscore - Oscore;
    else
        diff = Oscore - Xscore;

    // limiting depth to maxDepth
    int startDepth = Math.max(currentDepth, nRounds * 2 -
maxDepth);
    System.out.println("START DEPTH " + startDepth);
    System.out.println(nRounds);
    int[] result = minimax(getBoard(), Integer.MIN_VALUE,
Integer.MAX_VALUE, diff, -1, -1, startDepth, true);
    if (!Thread.currentThread().isInterrupted()) {
        currentDepth += 2;
    }

    return new int[]{result[2], result[1]};
}

public int[] minimax(
    Board board,
    int alpha,
    int beta,
    int diff,
    int xloc,
    int yloc,
    int depth,
    boolean ismax
) {
    if (Thread.currentThread().isInterrupted() || depth == nRounds
* 2) {
        return new int[]{diff, xloc, yloc};
    }
}
```

```

    }
    int coorx = xloc;
    int coory = yloc;

    if (ismax) {
        int maxval = Integer.MIN_VALUE;
        outerLoop:
        for (int y = 0; y < board.getRowCount(); y++) {
            for (int x = 0; x < board.getColCount(); x++) {
                if (Thread.currentThread().isInterrupted()) {
                    break outerLoop;
                }
                if (board.getCol(x, y) != 'n') continue;
                BoardChange boardChange = new BoardChange();
                int value = board.addSymbol(ownSymbol, x, y,
boardChange);
                int[] result = minimax(board, alpha, beta, diff +
value, x, y, depth + 1, false);
                boardChange.rollback(board);
                maxval = Math.max(result[0], maxval);
                if (alpha < maxval || coorx == -1) {
                    alpha = maxval;
                    coorx = x;
                    coory = y;
                }
                if (beta <= alpha) break;
            }
            if (beta <= alpha) break;
        }

        return new int[]{maxval, coorx, coory};
    } else {
        int minval = Integer.MAX_VALUE;
        outerLoop:
        for (int y = 0; y < board.getRowCount(); y++) {
            for (int x = 0; x < board.getColCount(); x++) {
                if (Thread.currentThread().isInterrupted()) {
                    break outerLoop;
                }
                if (board.getCol(x, y) != 'n') continue;
                BoardChange boardChange = new BoardChange();
                int value = board.addSymbol(enemySymbol, x, y,
boardChange);
                int[] result = minimax(board, alpha, beta, diff -
value, x, y, depth + 1, true);
                boardChange.rollback(board);

                minval = Math.min(result[0], minval);
                if (beta > minval || coorx == -1) {
                    beta = minval;
                    coorx = x;
                    coory = y;
                }
            }
        }
    }
}

```

```

        if (beta <= alpha) break;
    }
    if (beta <= alpha) break;
}
return new int[]{minval, coorx, coory};
}
}

```

#### 2.3.4. Local Search

Implementasi Local Search terdapat pada file BotLocalSearch.java di fungsi move. Local Search relatif cepat untuk dilakukan sehingga tidak memerlukan fungsi fallback.

```

public int[] move(int Xscore, int Oscore) {
    int obj = Integer.MIN_VALUE;
    ArrayList<int []> locations = new ArrayList<>();
    Board board = this.getBoard();
    for (int y = 0; y < getBoard().getRowCount(); y++) {
        for (int x = 0; x < getBoard().getColCount(); x++) {
            if (this.getBoard().getCol(x, y) != 'n') continue;
            BoardChange boardChange = new BoardChange();
            /* Mengecek value yang dihasilkan akibat penambahan
            di posisi x,y */
            int val = board.addSymbol(
                                ownSymbol, x,
                                y, boardChange
                            );
            boardChange.rollback(board);
            if (val > obj) {
                obj = val;
                locations.clear();
                locations.add(new int[]{y, x});
            } else if (val == obj) {
                locations.add(new int[]{y, x});
            }
        }
    }

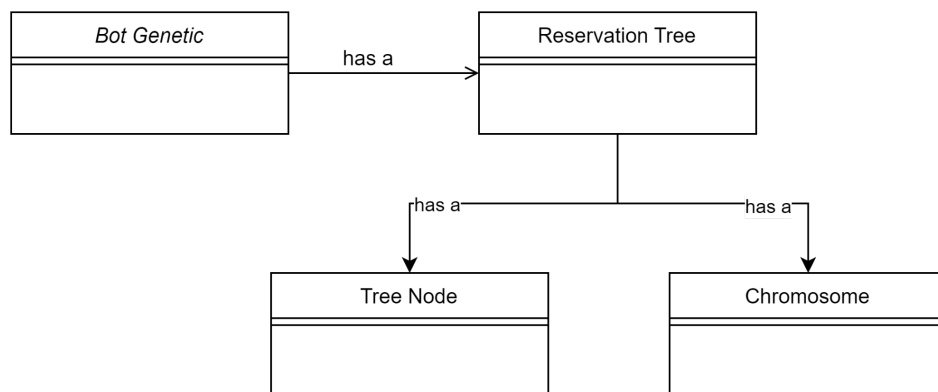
    return locations.get(new Random().nextInt(locations.size()));
}

```

### 2.3.5. Genetic Algorithm

Implementasi Genetic Algorithm terdapat pada file BotGenetic.java di fungsi move dan generate. Karena pada tugas ini terdapat batasan waktu dalam bergerak, yaitu 5 detik, terdapat fallback yang dilakukan oleh implementasi ini yaitu dengan melakukan move dengan kromosom terbaik yang diraih saat ini.

Bot Genetic dapat berperforma dengan baik dengan hanya 200 generasi, dengan waktu eksekusi yang lebih cepat dari minimax, dan dapat secara konsisten memenangkan pertandingan melawan Bot Local Search.



Gambar 1. Struktur Pembangun Bot Genetic Algorithm dan Elemen Pendukungnya

```
public int[] move(int Xscore, int Oscore) {
    this.numRoundPlayed += 1;
    return generate();
}

protected int[] generate() {
    // From the current state, generate N chromosome
    // Need the first part (history), then the second part (future)
    // Concat (for processing) then divide again

    // Coding scheme is indexing the board square (x = 0, y = 0) ->
    0

    // remove all filled indexes
    for (int i = 0; i < this.getBoard().getRowCount(); i++) {
        for (int j = 0; j < this.getBoard().getColCount(); j++) {
            if (this.getBoard().getCol(j, i) != 'n') {
                int finalI = i;
                int finalJ = j;
                availIdx.removeIf(e1 -> e1.equals(finalJ + finalI *
this.getBoard().getColCount()));
                System.out.print(finalJ + finalI *
```

```

this.getBoard().getColCount() + " ");
    }
}
}
if (availIdx.size() == 1) {
    int i = availIdx.get(0);
    int[] move = new int[2];
    move[1] = i % this.getBoard().getColCount();
    move[0] = i / this.getBoard().getColCount();
    return move;
}
System.out.println(" ");

// history
List<Integer> history = new ArrayList<>();
int gene;
for (int[] pair : boardChange.getLocations()) {
    gene = pair[0] + pair[1] * getBoard().getColCount();
    history.add(gene);
    int finalGene = gene;
    availIdx.removeIf(el -> el.equals(finalGene));
}

// Generate random genes
List<List<Integer>> nRandom = new ArrayList<>();

for (int i = 0; i < nGenerate; i++) {
    List<Integer> generated = new ArrayList<>();
    Random random = new Random();
    for (int j = 0; j < gameTreeDepth - numRoundPlayed + 1;
j++) {
        int chosenIdx = Math.min(availIdx.isEmpty()? 0 :
(availIdx.size() - 1), random.nextInt(availIdx.isEmpty()? 1 :
availIdx.size()));
        int element = availIdx.get(chosenIdx);
        generated.add(element);
        availIdx.removeIf(e -> e.equals(element));
    }
    availIdx.addAll(generated);
    nRandom.add(generated);
}

// remove all filled indexes
for (int i = 0; i < this.getBoard().getRowCount(); i++) {
    for (int j = 0; j < this.getBoard().getColCount(); j++) {
        if (this.getBoard().getCol(j, i) != 'n') {
            int finalI = i;
            int finalJ = j;
            availIdx.removeIf(el -> el.equals(finalJ + finalI *
this.getBoard().getColCount()));
        }
    }
}
}

```



```

// Concatenate history with current
List<List<Integer>> genes = new ArrayList<>();

for (List<Integer> i : nRandom) {
    List<Integer> concatenated = new ArrayList<>();
    concatenated.addAll(history);
    concatenated.addAll(i);
    genes.add(concatenated);
}

// Reservation tree

for (List<Integer> i : genes) {
    rt.insertChromosome(new Chromosome(i, this.ownSymbol));
}

for (int x = 0; x < this.limitGeneration; x++) {
    if (Thread.currentThread().isInterrupted()) {
        return new int[2];
    }
    // Crossover and mutation
    List<Chromosome> selected = new ArrayList<>();
    List<List<Integer>> subGenes = new ArrayList<>();
    List<List<Integer>> newGenes = new ArrayList<>();
    for (int i = 0; i < 4; i++) {
        // Select 4 random chromosomes with probability
selected.add(rt.selectRandomWithProbability(this.nThreshold));
        // Crossover and mutation index is n history + some
number, max n_rounds
        // Select back-side of crossover
        subGenes.add(new
ArrayList<>(selected.get(i).gene.subList(Math.min(history.size() - 1 +
crossoverRate, gameTreeDepth-1), selected.get(i).gene.size())));
        // Crossover
        newGenes.add(new
ArrayList<>(selected.get(i).gene.subList(0, Math.min(history.size() - 1
+ crossoverRate, gameTreeDepth-1))));
    }

    // Crossover swap
    newGenes.get(0).addAll(subGenes.get(1));
    newGenes.get(1).addAll(subGenes.get(0));
    newGenes.get(2).addAll(subGenes.get(3));
    newGenes.get(3).addAll(subGenes.get(2));

    Random random = new Random();
    Random mutation = new Random();
    for (int i = 0; i < 4; i++) {

```

```

        // Mutation
        int a = mutation.nextInt(10);
        if (a < this.mutationRate) {
//            System.out.println(a + " MUTATION");
            availIdx.removeAll(newGenes.get(i));
            List<Integer> removedIdx = new
ArrayList<>(newGenes.get(i));
            newGenes.get(i).set(Math.min(history.size() - 1 +
mutationIdx + random.nextInt(mutationIdx), newGenes.get(i).size()-1),
availIdx.get(Math.min(availIdx.isEmpty()? 0 : (availIdx.size() - 1),
random.nextInt(availIdx.isEmpty()? 1 : availIdx.size()))));
            availIdx.addAll(removedIdx);
        }
    }

    // Offspring
    List<Chromosome> offsprings = new ArrayList<>();
    for (int i = 0; i < 4; i++) {
        offsprings.add(new Chromosome(newGenes.get(i),
this.ownSymbol));
        rt.insertChromosome(offsprings.get(i));
    }

}

// Returning move
int[] move = new int[2];
int i = 0;
do {
    List<Map.Entry<Chromosome, Integer>> fin =
rt.getNTopValues(15);
    for (int j = 0; j < 15; j++) {
        System.out.println("Tree height: " +
fin.get(j).getValue() + " Chromosome value: " +
fin.get(j).getKey().value + " " + fin.get(j).getKey().gene);
    }
    Chromosome result = fin.get(i).getKey();
    System.out.println("PICKED: " + result.value + " " +
result.gene + " " + result.gene.get(i));

    move[1] = result.gene.get(i) %
this.getBoard().getColCount();
    move[0] = result.gene.get(i) /
this.getBoard().getColCount();
    i += 1;
} while (!availIdx.contains(move[1] + move[0] *
this.getBoard().getColCount()));

    rt = new ReservationTree();
    return move;
}

```

## 2.3.6. Pendukung

### 2.3.6.1. Bot

File ini berisikan kelas Bot yang merupakan abstract class yang diturunkan menjadi kelas bot - bot yang berisikan algoritma yang telah dijelaskan sebelumnya.

```
public abstract class Bot {
    private final Board board;

    protected Bot(Board board) {
        this.board = board;
    }

    public Board getBoard() {
        return board;
    }

    public abstract int[] move(int Xscore, int Oscore);

    protected int getObjectivFunc() {
        return 0;
    }
}
```

### 2.3.6.2. Board

File ini berisikan kelas Board yang merupakan class yang digunakan untuk merepresentasikan papan permainan.

```
public class Board {
    private final int colCount;
    private final int rowCount;
    private final Button[][] buttons;
    private final char[][] matrix;

    public Board(Button[][] buttons) {
        this.buttons = buttons;
        this.rowCount = buttons.length;
        this.colCount = buttons[0].length;
        matrix = new char[this.rowCount][this.colCount];

        for (int i = 0; i < this.rowCount; i++) {
            for (int j = 0; j < this.colCount; j++) {
                if (buttons[i][j].getText().length() > 0) {
                    matrix[i][j] =
buttons[i][j].getText().charAt(0);
                } else {
```

```

        matrix[i][j] = 'n';
    }
}

}

public int getColCount() {
    return colCount;
}

public int getRowCount() {
    return rowCount;
}

public char[] getRow(int y) {
    return matrix[y];
}

public char getCol(int x, int y) {
    return matrix[y][x];
}

public void setButton(int x, int y, char c) {
    buttons[y][x].setText(String.valueOf(c));
    setCol(x, y, c);
}

public void setCol(int x, int y, char value) {
    matrix[y][x] = value;
}

public void setCol(int x, int y, char value, BoardChange
boardChange) {
    if (boardChange != null) boardChange.addChange(x, y,
matrix[y][x]);
    setCol(x, y, value);
}

public int addSymbol(char symbol, int x, int y, BoardChange
boardChange) {
    if (matrix[y][x] != 'n') return -1;

    int value = 1;
    if (boardChange != null) {
        boardChange.reset();
    }
    setCol(x, y, symbol, boardChange);

    if (symbol == 'X') {
        if (x > 0) {
            if (matrix[y][x - 1] == 'O') {
                value += 2;
                setCol(x - 1, y, 'X', boardChange);
            }

            if (y > 0) {
                if (matrix[y - 1][x - 1] == 'O') {
                    value += 2;
                    setCol(x - 1, y - 1, 'X', boardChange);
                }
            }
        }
    }
}

```

```

        if (y < this.getRowCount() - 1) {
            if (matrix[y + 1][x - 1] == 'O') {
                value += 2;
                setCol(x - 1, y + 1, 'X', boardChange);
            }
        }
    }

    if (x < this.getColCount() - 1) {
        if (matrix[y][x + 1] == 'O') {
            value += 2;
            setCol(x + 1, y, 'X', boardChange);
        }

        if (y > 0) {
            if (matrix[y - 1][x + 1] == 'O') {
                value += 2;
                setCol(x + 1, y - 1, 'X', boardChange);
            }
        }

        if (y < this.getRowCount() - 1) {
            if (matrix[y + 1][x + 1] == 'O') {
                value += 2;
                setCol(x + 1, y + 1, 'X', boardChange);
            }
        }
    }

    if (y > 0) {
        if (matrix[y - 1][x] == 'O') {
            value += 2;
            setCol(x, y - 1, 'X', boardChange);
        }
    }

    if (y < this.getRowCount() - 1) {
        if (matrix[y + 1][x] == 'O') {
            value += 2;
            setCol(x, y + 1, 'X', boardChange);
        }
    }
} else if (symbol == 'O') {
    if (x > 0) {
        if (matrix[y][x - 1] == 'X') {
            value += 2;
            setCol(x - 1, y, 'O', boardChange);
        }

        if (y > 0) {
            if (matrix[y - 1][x - 1] == 'X') {
                value += 2;
                setCol(x - 1, y - 1, 'O', boardChange);
            }
        }

        if (y < this.getRowCount() - 1) {
            if (matrix[y + 1][x - 1] == 'X') {
                value += 2;
                setCol(x - 1, y + 1, 'O', boardChange);
            }
        }
    }
}

```

```

    }
}

if (x < this.getColCount() - 1) {
    if (matrix[y][x + 1] == 'X') {
        value += 2;
        setCol(x + 1, y, 'O', boardChange);
    }

    if (y > 0) {
        if (matrix[y - 1][x + 1] == 'X') {
            value += 2;
            setCol(x + 1, y - 1, 'O', boardChange);
        }
    }

    if (y < this.getRowCount() - 1) {
        if (matrix[y + 1][x + 1] == 'X') {
            value += 2;
            setCol(x + 1, y + 1, 'O', boardChange);
        }
    }
}

if (y > 0) {
    if (matrix[y - 1][x] == 'X') {
        value += 2;
        setCol(x, y - 1, 'O', boardChange);
    }
}

if (y < this.getRowCount() - 1) {
    if (matrix[y + 1][x] == 'X') {
        value += 2;
        setCol(x, y + 1, 'O', boardChange);
    }
}

return value;
}
}

```

#### 2.3.6.3. BoardChange

File ini berisikan kelas BoardChange yang merupakan kelas yang digunakan untuk menyimpan perubahan yang terjadi dalam suatu gerakan. Kelas ini berfungsi untuk melakukan backtracking pada papan.

```

public class BoardChange {
    private final ArrayList<int[]> locations;
    private final ArrayList<Character> symbols;

    public BoardChange() {
        this.locations = new ArrayList<>();
        this.symbols = new ArrayList<>();
    }
}

```

```

    public void rollback(Board board) {
        for (int i = 0; i < locations.size(); i++) {
            int[] location = locations.get(i);
            board.setCol(location[0], location[1],
symbols.get(i));
        }

    public void reset() {
        locations.clear();
        symbols.clear();
    }

    public void addChange(int x, int y, char symbol) {
        locations.add(new int[]{x, y});
        symbols.add(symbol);
    }

    public ArrayList<int[]> getLocations() {
        return this.locations;
    }
}

```

#### 2.3.6.4. Fallbackable

File ini berisikan interface Fallbackable yang merupakan interface untuk kelas yang memerlukan fungsi fallback ketika waktu melebihi waktu yang ditentukan.

```

public interface Fallbackable {
    int[] fallback(int Xscore, int Oscore);

    void onFastSuccess();
}

```

#### 2.3.6.5. Chromosome

File ini berisikan kelas Chromosome yang merupakan kelas yang digunakan untuk Genetic Algorithm seperti yang sudah dijelaskan sebelumnya.

```

public class Chromosome {
    public int value;
    public List<Integer> gene;

    protected char[][] board;

    protected int rowCount = 8;
    protected int colCount = 8;

    private final char ownSymbol;

    public Chromosome(char ownSymbol) {

```

```

        this.value = 0;
        this.ownSymbol = ownSymbol;
        this.gene = null;
        this.board = new char[8][8];
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                this.board[i][j] = 'n';
            }
        }

        // Removed pre-filled tiles
        this.setCol(6, 0, 'O');
        this.setCol(7, 0, 'O');
        this.setCol(6, 1, 'O');
        this.setCol(7, 1, 'O');

        this.setCol(0, 6, 'X');
        this.setCol(0, 7, 'X');
        this.setCol(1, 6, 'X');
        this.setCol(1, 7, 'X');
        this.evaluateAll();
    }

    public Chromosome(List<Integer> gene, char ownSymbol) {
        this.gene = gene;
        this.ownSymbol = ownSymbol;
        this.board = new char[8][8];
        for (int i = 0; i < 8; i++) {
            for (int j = 0; j < 8; j++) {
                this.board[i][j] = 'n';
            }
        }

        // Removed pre-filled tiles
        this.setCol(6, 0, 'O');
        this.setCol(7, 0, 'O');
        this.setCol(6, 1, 'O');
        this.setCol(7, 1, 'O');

        this.setCol(0, 6, 'X');
        this.setCol(0, 7, 'X');
        this.setCol(1, 6, 'X');
        this.setCol(1, 7, 'X');

        this.evaluateAll();
    }

    public int getRowCount() {
        return rowCount;
    }

    public int getColCount() {
        return colCount;
    }

    protected Pair<Integer, Integer> translate(int iGene) {
        // Translating index into coordinate
        Integer x = iGene % 8;
        Integer y = iGene / 8;

        return new Pair<>(x, y);
    }

```



```

public void updateValue() {
    // Update chromosome value
    int value = 0;
    char symbol;
    if (this.ownSymbol == 'X') {
        symbol = 'O';
    } else {
        symbol = 'X';
    }
    for (char[] row : this.board) {
        for (char el : row) {
            if (el == this.ownSymbol) {
                value += 1;
            } else if (el == symbol) {
                value -= 1;
            }
        }
    }

    this.value = value;
}

protected void setCol(int x, int y, char symbol) {
    // Set row and col value of board
    this.board[y][x] = symbol;
}

public void evaluateAll() {
    // Evaluates all the gene in chromosome
    boolean alt = ('X' == this.ownSymbol);
    for (Integer i : this.gene) {
        Pair<Integer, Integer> coordinate = translate(i);
        if (alt) {
            this.addSymbol(coordinate.getKey(),
coordinate.getValue(), 'X');
        } else {
            this.addSymbol(coordinate.getKey(),
coordinate.getValue(), 'O');
        }
        alt = !alt;
    }
    this.updateValue();
    // System.out.println("Value chromosome: " +
this.value);
}

public void insertGene(Integer i) {
    // Insert a new gene
    this.gene.add(i);
    this.evaluateAll();
}

public int addSymbol(int x, int y, char symbol) {
    // Board logic when adding a symbol
    if (this.board[y][x] != 'n') return -1;

    int value = 1;
    this.setCol(x, y, symbol);

    if (symbol == 'X') {
        if (x > 0) {
            if (this.board[y][x - 1] == 'O') {

```

```

        value += 2;
        this.setCol(x - 1, y, 'X');
    }

    if (y > 0) {
        if (this.board[y - 1][x - 1] == 'O') {
            value += 2;
            this.setCol(x - 1, y - 1, 'X');
        }
    }

    if (y < this.getRowCount() - 1) {
        if (this.board[y + 1][x - 1] == 'O') {
            value += 2;
            this.setCol(x - 1, y + 1, 'X');
        }
    }
}

if (x < this.getColCount() - 1) {
    if (this.board[y][x + 1] == 'O') {
        value += 2;
        this.setCol(x + 1, y, 'X');
    }

    if (y > 0) {
        if (this.board[y - 1][x + 1] == 'O') {
            value += 2;
            this.setCol(x + 1, y - 1, 'X');
        }
    }

    if (y < this.getRowCount() - 1) {
        if (this.board[y + 1][x + 1] == 'O') {
            value += 2;
            this.setCol(x + 1, y + 1, 'X');
        }
    }
}

if (y > 0) {
    if (this.board[y - 1][x] == 'O') {
        value += 2;
        this.setCol(x, y - 1, 'X');
    }
}

if (y < this.getRowCount() - 1) {
    if (this.board[y + 1][x] == 'O') {
        value += 2;
        this.setCol(x, y + 1, 'X');
    }
}
} else if (symbol == 'O') {
    if (x > 0) {
        if (this.board[y][x - 1] == 'X') {
            value += 2;
            this.setCol(x - 1, y, 'O');
        }

        if (y > 0) {
            if (this.board[y - 1][x - 1] == 'X') {

```

```

        value += 2;
        this.setCol(x - 1, y - 1, 'O');
    }
}

if (y < this.getRowCount() - 1) {
    if (this.board[y + 1][x - 1] == 'X') {
        value += 2;
        this.setCol(x - 1, y + 1, 'O');
    }
}

if (x < this.getColCount() - 1) {
    if (this.board[y][x + 1] == 'X') {
        value += 2;
        this.setCol(x + 1, y, 'O');
    }

    if (y > 0) {
        if (this.board[y - 1][x + 1] == 'X') {
            value += 2;
            this.setCol(x + 1, y - 1, 'O');
        }
    }

    if (y < this.getRowCount() - 1) {
        if (this.board[y + 1][x + 1] == 'X') {
            value += 2;
            this.setCol(x + 1, y + 1, 'O');
        }
    }
}

if (y > 0) {
    if (this.board[y - 1][x] == 'X') {
        value += 2;
        this.setCol(x, y - 1, 'O');
    }
}

if (y < this.getRowCount() - 1) {
    if (this.board[y + 1][x] == 'X') {
        value += 2;
        this.setCol(x, y + 1, 'O');
    }
}

return value;
}
}

```

#### 2.3.6.6. Reservation Tree

File ini berisikan kelas Reservation Tree yang merupakan kelas yang digunakan untuk Genetic Algorithm seperti yang sudah dijelaskan sebelumnya.

```

public class ReservationTree {
    int height;
    TreeNode root;
    Map<Chromosome, Integer> Chromosomes = new HashMap<>();
    // bot.Chromosome

    public ReservationTree() {
        this.root = new TreeNode(null, 0, true, false);
    }

    public void insertChromosome(Chromosome chromosome) {
        // Insert chromosome into the tree
        TreeNode node = this.root;
        int count = 0;
        for (Integer i : chromosome.gene) {
            // If node has a child with the index, then insert
            // at that node and only update the node's value
            // If no, then create a node child with the index
            // and value then insert at that node
            if (node.children.containsKey(i)) {
                // System.out.print("Have\t" + i + "\t");
                // System.out.println(node.value);
                node.isLeaf = false;
            } else {
                if (count == chromosome.gene.size() - 1) {
                    node.addChildren(i, new
TreeNode(chromosome.value, node.level+1, !node.isMax, true));
                    node.isLeaf = false;
                } else {
                    node.addChildren(i, new TreeNode(null,
node.level+1, !node.isMax, true));
                    node.isLeaf = false;
                }
                this.height = Math.max(node.level+1,
this.height);
                // System.out.print("Not have\t" + i + "\t");
                //System.out.println(node.value);
            }
            count += 1;
            node = node.children.get(i);
            // System.out.println("node " + i + " value " +
node.value);
        }
        updateTreeValue(root);
        this.Chromosomes.put(chromosome, 1);
    }

    public Integer updateTreeValue(TreeNode node) {
        // To update / propagate the value from the leaf upwards
        List<Integer> values = new ArrayList<>();
        if (node.isLeaf) {
            values.add(node.value);
        }
        for (TreeNode child : node.children.values()) {
            values.add(this.updateTreeValue(child));
        }
        // System.out.println(values);
        node.updateValue(values);
        return node.value;
    }
}

```

```

        private void evaluateFitness() {
            // Evaluate the fitness of each chromosome in the
reservation tree
            for (Map.Entry<Chromosome, Integer> entry :
Chromosomes.entrySet()) {
                Chromosome chromosome = entry.getKey();

                // Update the value
                entry.setValue(getChromosomeValue(root, chromosome,
0));
            }
        }

        private int getChromosomeValue(TreeNode node, Chromosome
chromosome, int i) {
            // Recursive exploration of node, bottom-up, according
to chromosome
            // If node is the same value as the chromosome, then +1
            int val = 1;
            if (i < chromosome.gene.size()) {
                if (node.value == chromosome.value) {
                    val = 1 +
this.getChromosomeValue(node.children.get(chromosome.gene.get(i)
), chromosome, i + 1);
                } else {
                    val =
this.getChromosomeValue(node.children.get(chromosome.gene.get(i)
), chromosome, i + 1);
                }
            }
            return val;
        }

        public List<Map.Entry<Chromosome, Integer>>
getNTopValues(int n) {
            // Return the top valued N chromosome
            this.evaluateFitness();

            if (n > this.Chromosomes.size()) {
                n = this.Chromosomes.size();
            }

            // Custom comparator
            Comparator<Map.Entry<Chromosome, Integer>>
valueComparator = (entry1, entry2) ->
entry2.getValue().compareTo(entry1.getValue());

            // Convert map entries to a list
            List<Map.Entry<Chromosome, Integer>> entryList = new
ArrayList<>(Chromosomes.entrySet());

            // Sort list using custom comparator
            entryList.sort(valueComparator);

            // Take top n entries
            List<Map.Entry<Chromosome, Integer>> topNEntries =
entryList.subList(0, Math.min(n, entryList.size()));

            // Extract keys (objects) from top N entries
            List<Map.Entry<Chromosome, Integer>> topNObjects = new
ArrayList<>();
            for (Map.Entry<Chromosome, Integer> entry : topNEntries)
{

```

```

        topNObjects.add(entry);
    }
    return topNObjects;
}

public Chromosome selectRandomWithProbability(int n) {
    // Select a random chromosome with probability
    double total = 0;
    Map<Chromosome, Double> probabilityMap = new
HashMap<>();
    for (Map.Entry<Chromosome, Integer> entry :
this.getNTopValues(n)) {
        total += entry.getValue();
    }
    // Value/Probability mapping
    for (Map.Entry<Chromosome, Integer> entry :
this.Chromosomes.entrySet()) {
        probabilityMap.put(entry.getKey(), ((double)
entry.getValue()) / total);
    }

    double totalProbability = 0.0;
    double randomValue = new Random().nextDouble();

    for (Map.Entry<Chromosome, Double> entry :
probabilityMap.entrySet()) {
        totalProbability += entry.getValue();
        if (randomValue <= totalProbability) {
            return entry.getKey();
        }
    }

    // If no element selected (due to rounding errors),
return last element
    return null;
}

public void print(TreeNode node) {
    System.out.println(node.level + " " + node.value);
    for (TreeNode child : node.children.values()) {
        this.print(child);
    }
}

public static void main(String[] args) {
    ReservationTree rt = new ReservationTree();
    List<Integer> genes = new ArrayList<>();
    genes.add(1);
    genes.add(2);
    genes.add(3);
    Chromosome c = new Chromosome(genes, 'X');
    rt.insertChromosome(c);
    genes.remove(2);
    genes.add(10);
    Chromosome d = new Chromosome(genes, 'X');
    rt.insertChromosome(d);
    List<Integer> genes2 = new ArrayList<>();
    genes2.add(1);
    genes2.add(4);
    genes2.add(5);
    Chromosome e = new Chromosome(genes2, 'X');
}

```

```

        rt.insertChromosome(e);

        rt.print(rt.root);
        System.out.println(rt.getNTopValues(3));
        System.out.println(rt.selectRandomWithProbability(3));
    }

}

```

#### 2.3.6.7. TreeNode

File ini berisikan kelas Reservation Tree yang merupakan kelas yang digunakan untuk Genetic Algorithm seperti yang sudah dijelaskan sebelumnya.

```

public class TreeNode {
    Integer value;
    Integer level;

    boolean isMax;

    Map<Integer, TreeNode> children;
    TreeNode parent;

    public boolean isLeaf = true;

    public TreeNode(Integer value, Integer level, boolean isMax,
        boolean isLeaf) {
        this.value = value;
        this.level = level;
        this.children = new HashMap<>();
        this.parent = null;
        this.isMax = isMax;
        this.isLeaf = isLeaf;
    }

    public void addChildren(Integer index, TreeNode child) {
        this.children.put(index, child);
    }

    public void updateValue(Integer value) {
        // Update node value, according to isMax
        if (this.value == null) {
            this.value = value;
            return;
        }
        if (this.isMax) {
            this.value = Math.max(this.value, value);
        } else {
            this.value = Math.min(this.value, value);
        }
    }

    public void updateValue(List<Integer> values) {
        // Update node value, according to isMax, get the
        min/max of a list
        if (this.isMax) {

```

```
        this.value = Collections.max(values);  
    } else {  
        this.value = Collections.min(values);  
    }  
}  
}
```



## BAB III EXPERIMENT

### 3.1. Eksperimen

#### 3.1.1. Bot Minimax vs. Human

Tabel 3.1.1.1 Hasil percobaan *bot minimax* vs. manusia

No	Screenshot Hasil	Pemenang
1		Minimax
2		Minimax

3	<div><div>Game Board Display</div><table><tr><td>X</td><td>O</td><td>X</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr><tr><td>X</td><td>O</td><td>X</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr><tr><td>X</td><td>X</td><td>X</td><td>O</td><td>X</td><td>X</td><td>O</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr><tr><td>X</td><td>X</td><td>X</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td></tr><tr><td>X</td><td>X</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td></tr><tr><td>X</td><td>X</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td></tr></table><div><div>Number Of Rounds Left:</div><div>0</div></div><div><div>Player X</div><div>Player O</div></div><div><div>Human</div><div>Minimax (Winner!)</div></div><div><div>21</div><div>43</div></div><div>&gt;</div><div><div>End Game</div><div>Play New Game</div></div></div>	X	O	X	O	O	O	O	O	X	O	X	O	O	O	O	O	X	X	X	O	X	X	O	O	O	O	X	O	O	O	O	O	X	X	X	O	O	O	O	O	O	O	X	O	O	O	X	O	X	X	O	O	O	X	O	O	X	X	O	O	O	X	O	O	Minimax
X	O	X	O	O	O	O	O																																																											
X	O	X	O	O	O	O	O																																																											
X	X	X	O	X	X	O	O																																																											
O	O	X	O	O	O	O	O																																																											
X	X	X	O	O	O	O	O																																																											
O	O	X	O	O	O	X	O																																																											
X	X	O	O	O	X	O	O																																																											
X	X	O	O	O	X	O	O																																																											
4	<div><div>Game Board Display</div><table><tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td></tr><tr><td>X</td><td>X</td><td>O</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td></tr><tr><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>X</td><td>X</td></tr><tr><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td><td>O</td><td>X</td><td>X</td></tr><tr><td>O</td><td>O</td><td>X</td><td>X</td><td>O</td><td>X</td><td>O</td><td>O</td></tr><tr><td>X</td><td>O</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td><td>O</td></tr></table><div><div>Number Of Rounds Left:</div><div>0</div></div><div><div>Player X</div><div>Player O</div></div><div><div>manusia</div><div>minimax (Winner!)</div></div><div><div>19</div><div>45</div></div><div>&gt;</div><div><div>End Game</div><div>Play New Game</div></div></div>	O	O	O	O	O	X	O	O	O	O	X	O	O	O	X	O	X	X	O	O	X	O	X	O	O	O	O	O	O	O	X	X	O	O	X	O	O	O	O	O	O	O	X	O	O	O	X	X	O	O	X	X	O	X	O	O	X	O	O	X	O	X	O	O	Minimax
O	O	O	O	O	X	O	O																																																											
O	O	X	O	O	O	X	O																																																											
X	X	O	O	X	O	X	O																																																											
O	O	O	O	O	O	X	X																																																											
O	O	X	O	O	O	O	O																																																											
O	O	X	O	O	O	X	X																																																											
O	O	X	X	O	X	O	O																																																											
X	O	O	X	O	X	O	O																																																											
5	<div><div>Game Board Display</div><table><tr><td>O</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td><td>O</td><td>O</td></tr><tr><td>O</td><td>O</td><td>X</td><td>O</td><td>X</td><td>O</td><td>O</td><td>O</td></tr><tr><td>O</td><td>X</td><td>O</td><td>X</td><td>X</td><td>O</td><td>X</td><td>X</td></tr><tr><td>X</td><td>X</td><td>O</td><td>O</td><td>X</td><td>O</td><td>X</td><td>X</td></tr><tr><td>X</td><td>X</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td><td>O</td></tr><tr><td>X</td><td>X</td><td>O</td><td>O</td><td>X</td><td>O</td><td>O</td><td>O</td></tr><tr><td>X</td><td>X</td><td>O</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td></tr><tr><td>X</td><td>X</td><td>O</td><td>O</td><td>O</td><td>O</td><td>X</td><td>O</td></tr></table><div><div>Number Of Rounds Left:</div><div>0</div></div><div><div>Player X</div><div>Player O</div></div><div><div>Human</div><div>Minimax (Winner!)</div></div><div><div>25</div><div>39</div></div><div>&gt;</div><div><div>End Game</div><div>Play New Game</div></div></div>	O	O	X	O	X	O	O	O	O	O	X	O	X	O	O	O	O	X	O	X	X	O	X	X	X	X	O	O	X	O	X	X	X	X	O	O	O	O	O	O	X	X	O	O	X	O	O	O	X	X	O	O	O	O	X	O	X	X	O	O	O	O	X	O	Minimax
O	O	X	O	X	O	O	O																																																											
O	O	X	O	X	O	O	O																																																											
O	X	O	X	X	O	X	X																																																											
X	X	O	O	X	O	X	X																																																											
X	X	O	O	O	O	O	O																																																											
X	X	O	O	X	O	O	O																																																											
X	X	O	O	O	O	X	O																																																											
X	X	O	O	O	O	X	O																																																											

### 3.1.2. Bot Local Search vs. Human

Tabel 3.1.2.1 Hasil percobaan *bot local search* vs. manusia

No	Screenshot Hasil	Pemenang
1		Human
2		Human
3		Local Search

### 3.1.3. Bot Minimax vs. Bot Local Search

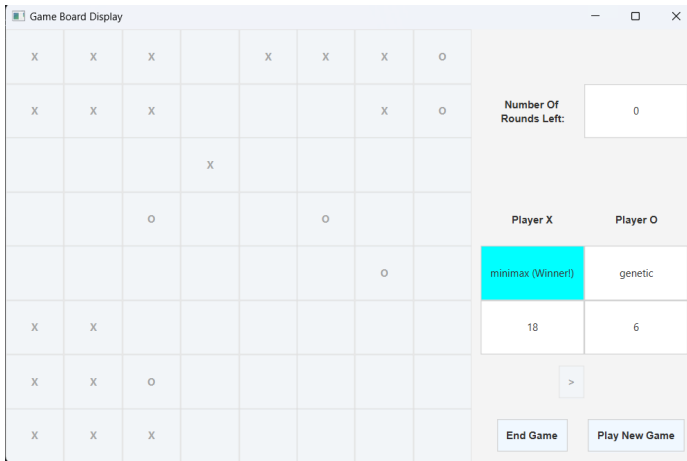
Tabel 3.1.3.1 Hasil percobaan *bot minimax vs. bot local search*

No	Screenshot Hasil	Pemenang
1	 <p>Note : Ronde Maksimum</p>	Minimax
2		Minimax
3		Minimax

3.1.4. Bot Minimax vs. Bot Genetic Algorithm

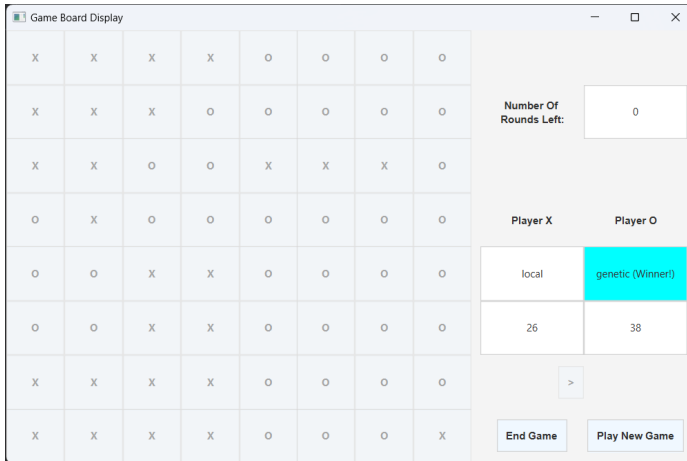
Tabel 3.1.4.1 Hasil percobaan *bot minimax* vs. bot genetic algorithm

No	Screenshot Hasil	Pemenang
1	<div><p>Note : Ronde Maksimum</p></div>	Minimax
2	<div></div>	Minimax

3		Minimax
---	--	---------

### 3.1.5. Bot Local Search vs. Bot Genetic Algorithm

Tabel 3.1.5.1 Hasil percobaan *bot local search vs. bot genetic algorithm*

No	Screenshot Hasil	Pemenang
1	 <p>Note : Ronde Maksimum</p>	Genetic

2	 <p>Game Board Display</p> <p>Number Of Rounds Left: 0</p> <p>Player X: local (29)</p> <p>Player O: genetic (Winner!) (35)</p> <p>End Game Play New Game</p>	Genetic
3	 <p>Game Board Display</p> <p>Number Of Rounds Left: 0</p> <p>Player X: local (25)</p> <p>Player O: genetic (Winner!) (39)</p> <p>End Game Play New Game</p>	Genetic

## 3.2. Analisis

### 3.2.1. Bot Minimax

Tabel 3.2.1.1 Analisis kemenangan *bot minimax*

Jumlah Permainan	11
Jumlah Kemenangan	11
Persentase Kemenangan	100%

Bot minimax memenangi 100% dari pertandingan yang dimainkannya, yaitu 11 kemenangan dalam 11 pertandingan.

Pada 5 permainan melawan manusia, minimax mengalahkan manusia dengan cukup telak dan konsisten. Hal ini disebabkan oleh pemikiran beberapa langkah ke depan yang cukup sulit dilakukan untuk manusia yang tidak terlatih.

Hal yang sama terjadi untuk 3 pertandingan melawan local search. Hal ini disebabkan oleh local search yang hanya mencari titik optimal pada satu gerakan, sedangkan minimax mencari titik optimal dalam beberapa langkah ke depan.

Pada 3 pertandingan permainan melawan genetik, minimax masih memenangkan permainan secara relatif konsisten. Hal ini disebabkan genetik relatif memiliki kemungkinan penyelesaian yang lebih random dan untuk kemungkinan besar dan generasi yang rendah, minimax masih lebih baik.

### 3.2.2. Bot Local Search

Tabel 3.2.1.1 Analisis kemenangan *bot local search*

Jumlah Permainan	9
Jumlah Kemenangan	1
Persentase Kemenangan	11%

Bot minimax memenangi 100% dari pertandingan yang dimainkannya, yaitu 1 kemenangan dalam 9 pertandingan.

Pada 3 permainan melawan manusia, minimax kalah tipis dalam 2 pertandingan dan memenangkan 1 pertandingan melawan manusia. Hal ini disebabkan manusia secara umum dapat berpikir lebih dari satu langkah kedepan, namun manusia juga dapat melakukan perhitungan yang salah dalam permainannya.

Pada 3 permainan melawan minimax, local search kalah telak dalam permainannya. Hal ini disebabkan oleh local search yang hanya mencari titik optimal pada satu gerakan, sedangkan minimax mencari titik optimal dalam beberapa langkah ke depan.

Pada 3 permainan melawan genetik, local search masih kalah dalam permainannya, namun relatif lebih tipis dibandingkan melawan minimax. Hal ini



disebabkan genetika dapat memperhitungkan gerakan lebih dari satu langkah ke depan walaupun memiliki faktor random yang lebih besar.

Local search kurang dapat mendapatkan performa yang baik karena local search akan selalu mengambil keputusan yang terbaik untuk saat itu saja, tanpa memperhitungkan efeknya pada masa depan. Hal tersebut (selalu mengambil keputusan optimal pada saat itu saja) tidak bisa mendapatkan solusi yang optimal dalam sebuah permainan dengan *multi-agent*.

### 3.2.3. Bot Genetic Algorithm

Tabel 3.3.1.1 Analisis kemenangan *bot genetic algorithm*

Jumlah Permainan	6
Jumlah Kemenangan	3
Persentase Kemenangan	50%

Bot minimax memenangi 50% dari pertandingan yang dimainkannya, yaitu 3 kemenangan dalam 6 pertandingan.

Pada 3 pertandingan melawan local search, genetika menang tipis dalam permainannya. Hal ini disebabkan genetika dapat memperhitungkan gerakan lebih dari satu langkah ke depan walaupun memiliki faktor random yang lebih besar. Sedangkan local search hanya mengambil gerakan paling optimal saat itu saja, tanpa memperhitungkan kemungkinan gerakan lawan. Genetika juga dapat lebih optimal dibandingkan local search dalam minimax karena genetika melakukan perbandingan yang lebih baik dan dalam secara *game-search tree* dibandingkan local search.

Pada 3 permainan melawan minimax, genetika kalah telak dalam permainannya. Hal ini disebabkan genetika relatif memiliki kemungkinan penyelesaian yang lebih random dan untuk kemungkinan besar dan generasi yang rendah, minimax masih lebih baik. Tetapi untuk jumlah generasi yang dapat mendapatkan solusi optimal (~200 generasi), algoritma genetika dapat selalu berjalan lebih cepat dibandingkan minimax with *alpha beta pruning*.

## **BAB IV**

### **PEMBAGIAN TUGAS**

NIM	Nama	Pekerjaan
13521087	Razzan Daksana Yoni	Local Search
13521089	Kenneth Ezekiel Supranton	Genetic Algorithm for Minimax
13521095	Muhamad Aji Wibisono	Minimax Alpha-Beta Pruning, Logic Board
13521101	Arsa Izdihar Islam	Minimax Alpha-Beta Pruning, Mekanisme Fallback

## **BAB V**

### **LAMPIRAN**

Github : <https://github.com/arsaizdihar/Tubes1-AI-2023.git>