LAPORAN TUGAS BESAR I
IF3170 INTELEGENSI BUATAN

Minimax Algorithm and Alpha Beta Pruning in Adjacency Strategy Game

Disusun oleh:

13521099	Vieri Fajar Firdaus
13521106	Mohammad Farhan Fahrezy
13521121	Saddam Annais Shaquille
13521160	M. Dimas Sakti Widyatmaja

PROGRAM STUDI
TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

2023

Daftar Isi

Penjelasan Objective Function

```
private int getScore(Map map){
    int score=0;
    for(int row=0;row<8;row++){
        for(int column=0;column<8;column++){
            if(map.getPosition(row,column).equals("O")){
                score+=1;
            }
            if(map.getPosition(row,column).equals("X")){
                score-=1;
            }
        }
    }
    return score;
}
```

Fungsi objektif yang kami gunakan yaitu perhitungan skor berdasarkan heuristik yang telah kami tentukan. Heuristik penentuan skor *state* saat ini yaitu dengan menambahkan 1 poin untuk setiap bidak yang dimiliki dan pengurangan 1 poin untuk setiap bidak lawan yang ada pada papan.

Alasan pembuatan *objective function* tersebut adalah untuk mengetahui nilai dari suatu *move* untuk dilakukan agar mendekati kemenangan dengan cara memperbanyak tanda bot pada papan. Tergantung dari algoritma yang akan digunakan, fungsi ini akan digunakan untuk menguji kelayakan dari beberapa kemungkinan yang bisa dilakukan.

Implementasi Algoritma Minimax dan Alpha Beta Pruning

Algoritma Minimax Alpha Beta Pruning memiliki kompleksitas eksponensial. Hal tersebut membuat algoritma minimax alpha beta pruning tidak dapat secara penuh diterapkan pada permainan ini dikarenakan ukuran permainannya yang terlalu besar. Tujuan dari algoritma ini adalah untuk menentukan langkah optimal yang harus diambil oleh pemain dalam permainan tersebut, dengan mempertimbangkan kemungkinan langkah lawan.

```
    private Pair<Integer,Pair<Integer,Integer>> minimax(Map map,int
depth, int alpha, int beta, boolean turn){
                                                Pair<Integer,Integer>[]
validLocation=getValidLocations(map);


            int alphaNow=alpha;
            int betaNow=beta;

            if(depth==0){
                return new Pair<>(getScore(map),null);
            }

            Pair<Integer,Integer> locationNow=validLocation[0];
            int score;

            if(turn){
                score=-1000;
                for(int i=0;i<validLocation.length;i++){
                    Map mapCopy = new Map(map);

mapCopy.addPosition(validLocation[i].getKey(),validLocation[i].getValue(
),"O");
                                                        int newScore =
minimax(mapCopy,depth-1,alphaNow,betaNow,false).getKey();
                    if(newScore>score){
                        score=newScore;
                        locationNow=validLocation[i];
                    }
```

```
                    if(alpha<score){
                        alphaNow=score;
                    }

                    if(alpha>=beta){
                        break;
                    }
                }
                return new Pair(score,locationNow);
            }else{
                score=1000;
                for(int i=0;i<validLocation.length;i++){
                    Map mapCopy = new Map(map);

mapCopy.addPosition(validLocation[i].getKey(),validLocation[i].getValue(
),"X");
                                                        int   newScore   =
minimax(mapCopy,depth-1,alphaNow,betaNow,true).getKey();
                    if(newScore<score){
                        score=newScore;
                        locationNow=validLocation[i];
                    }
                    if(beta>score){
                        betaNow=score;
                    }

                    if(alpha>=beta){
                        break;
                    }
                }
                return new Pair(score,locationNow);
            }

        }
```

Berikut adalah langkah-langkah dalam proses pencarian dengan Minimax Alpha-Beta Pruning:

1. Fungsi minimax menerima beberapa parameter yaitu peta permainan saat ini, kedalaman pencarian (*depth*), nilai alpha dan beta, serta giliran siapa yang

bermain. Pada awalnya, fungsi akan mendapatkan semua lokasi yang valid untuk ditempati di peta saat ini.

2. Untuk mengurangi kompleksitas, tidak semua petak kosong pada peta akan dibangkitkan rekursifnya. Namun, akan dicek apakah petak tersebut bertetanggaan dengan petak yang terisi. Jika bertetangga, petak tersebut akan dibangkitkan rekursifnya.
3. Jika kedalaman pencarian (*depth*) sudah mencapai 0, maka fungsi akan mengembalikan skor yang didapat dari peta saat ini. Jika belum, maka akan melakukan pencarian secara rekursif.
4. Jika giliran pemain sendiri atau *turn* bernilai true, rekursif akan dibangkitkan di semua lokasi yang valid. Pertama-tama dengan cara membuat *copy* dari peta saat ini, menambahkan posisi yang diisi, lalu memanggil rekursif fungsi minimax dengan giliran lawan, dan mengambil skor yang didapat. Skor terbaik akan disimpan dan dikembalikan bersama lokasi terpilih.
5. Sedangkan jika giliran lawan, caranya juga sama seperti pemain sendiri, akan tetapi dengan tujuan meminimalkan skor. Fungsi akan mengembalikan skor terbaik dan lokasi yang harus dipilih agar skor maksimal bagi pemain sendiri. Dengan cara ini algoritma dapat memprediksi langkah terbaik berdasarkan analisis cabang-cabang pilihan selanjutnya.

Implementasi Algoritma Local Search

Pada persoalan ini, rute yang dipilih digunakan pada pemilihan posisi petak bot, sehingga persoalan ini tidak cocok untuk dikerjakan dengan metode *Local Search*, namun dengan sedikit modifikasi persoalan diatas diselesaikan dengan *local search* dengan variabel sebagai berikut :

- State : penyusunan posisi bidak bot dalam waktu tertentu
- Initial State : posisi yang paling dekat dengan koordinat (0,0) atau posisi yang terdekat pada baris 0 kolom 0
- Goal State : banyak bidak bot lebih dari bidak player lain
- Action : penempatan bidak bot pada papan
- Solution : langkah pemilihan peletakan posisi bot

Pada persoalan kali ini dipilih metode Hill-Climbing untuk menyelesaikannya, dengan algoritma sebagai berikut:

```
private Pair<Integer,Integer> localSearch(Button[][] buttons){
    Pair<Integer,Integer> location = null;
    int score=-10000;
    for(int row=0;row<8;row++){
        for(int column=0;column<8;column++){
            if(buttons[row][column].getText().isEmpty()){

                Map mapCopy = new Map(buttons);
                mapCopy.addPosition(row,column,"O");
                if(score<getScore(mapCopy)){
                    score=getScore(mapCopy);
                    location=new Pair<>(row,column);
                }
            }

        }
    }
    return  location;
}
```

Kode yang diberikan adalah implementasi dari algoritma Local Search untuk mencari lokasi optimal di dalam permainan Adjacency Strategy Game yang direpresentasikan oleh buttons. Tujuan utamanya adalah untuk menemukan posisi

yang optimal (baris dan kolom) untuk menambahkan elemen "O" di papan permainan tersebut sehingga akan menghasilkan skor tertinggi.

Langkah yang dilakukan adalah melakukan iterasi dari baris nol kolom nol sampai baris tujuh kolom tujuh untuk mendapatkan score terbaik, score terbaik diukur berdasarkan fungsi objektif getScore.

Pada persoalan ini tidak digunakan metode Simulated Annealing karena bekerja dengan cara mengizinkan memilih langkah yang tidak optimal dengan probabilitas tertentu yang kecil. Pada permainan ini, bot berjalan dengan maksimal langkah sebanyak 28 dan angka ini terlalu sedikit untuk mendapat hasil terbaik jika permainan Adjacency Strategy Game menggunakan metode Simulated Annealing.

Implementasi Genetic Algorithm

Genetic Algorithm diimplementasikan pada file GeneticAlgorithm.java. Saat pertama kali dipanggil, constructor akan menyimpan seluruh atribut yang akan digunakan selama perhitungan untuk menentukan langkah yang akan diambil bot. Atribut tersebut terdiri dari ukuran populasi, panjang kromosom, jumlah maksimal generasi dan rate mutasi.

```
public GeneticAlgorithm(int population, int chromosomeLength, int
max_generations, double mutationRate, Button[][] buttons, String firstPlayer,
String secondPlayer) {
    this.POPULATION_SIZE = population;
    this.CHROMOSOME_LENGTH = chromosomeLength;
    this.MAX_GENERATIONS = max_generations;
    this.MUTATION_RATE = mutationRate;
    Map tempMap = new Map(buttons);
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (!tempMap.getPosition(i, j).isEmpty()) {
                pointMap.add(i * 8 + j);
                playerMap[i * 8 + j] = tempMap.getPosition(i, j);
            }
        }
    }
    this.firstPlayer = firstPlayer;
    this.secondPlayer = secondPlayer;
}
```

Berikut adalah langkah-langkah dalam proses pencarian dengan Genetic Algorithm:

1. Trigger fungsi execute untuk menjalankan algoritma

```
public int execute() {
    int[][] population = initializePopulation();
    int generation = 1;
    int[] fitnessScores;
    int fitnessMax=-9999999;
    int idxMax=0;
    while (generation <= MAX_GENERATIONS) {
        fitnessScores = calculateFitness(population);

        int[][] newPopulation = new int[POPULATION_SIZE][CHROMOSOME_LENGTH];

        for (int i = 0; i < POPULATION_SIZE; i++) {
            int[] parent1 = selectParent(population, fitnessScores);
            int[] parent2 = selectParent(population, fitnessScores);
            int[] offspring = crossover(parent1, parent2);
            offspring = mutate(offspring);
```

```
            newPopulation[i] = offspring;
        }

        population = newPopulation;
        generation++;
        if(generation==MAX_GENERATIONS){
            for(int i=0;i<POPULATION_SIZE;i++){
                if(CHROMOSOME_LENGTH%2==0){
                    if(fitnessScores[i]<fitnessMax){
                        fitnessMax=fitnessScores[i];
                        idxMax=i;
                    }
                }
                else{
                    if(fitnessScores[i]>fitnessMax){
                        fitnessMax=fitnessScores[i];
                        idxMax=i;
                    }
                }
            }

        }
    }


    return population[idxMax][0];
}
```

2. Initialize populasi awal

```
private int[][] initializePopulation() {
    int[][] population = new int[POPULATION_SIZE][CHROMOSOME_LENGTH];
    Random random = new Random();

    for (int i = 0; i < POPULATION_SIZE; i++) {
        for (int j = 0; j < CHROMOSOME_LENGTH; j++) {
            ArrayList<Integer> randomize= new ArrayList<>();
            for(int idxRandom=0;idxRandom<64;idxRandom++){
                randomize.add(idxRandom);
            }
            randomize.removeAll(pointMap);
            //amann
            for(int idx=0;idx<j;idx++){
                randomize.remove(randomize.indexOf(population[i][idx]));
            }
            population[i][j]=randomize.get(random.nextInt(randomize.size()));
        }
    }

    return population;
}
```

3. Menghitung fitness score dari populasi

```
private int[] calculateFitness(int[][] population) {
    int[] fitnessScores = new int[POPULATION_SIZE];

    for (int i = 0; i < POPULATION_SIZE; i++) {
        int chromosomeSum = 0;

        Map tempMap = new Map();
        for(int idx=0;idx<pointMap.size();idx++){

tempMap.addPosition(pointMap.get(idx)/8,pointMap.get(idx)%8,playerMap[pointMap
.get(idx)]);
        }
        for(int idx=0;idx<CHROMOSOME_LENGTH;idx++){
            if(idx%2==0){

tempMap.addPosition(population[i][idx]/8,population[i][idx]%8,firstPlayer);
            }
            else{

tempMap.addPosition(population[i][idx]/8,population[i][idx]%8,secondPlayer);
            }
        }

        fitnessScores[i] = tempMap.getScore(firstPlayer,secondPlayer);
    }

    return fitnessScores;
}
```

4. Memilih parent untuk dijadikan offspring

```
private int[] selectParent(int[][] population, int[] fitnessScores) {
    Random random = new Random();
    int totalFitness = Arrays.stream(fitnessScores).sum();
    double rouletteWheelPosition = random.nextDouble() * totalFitness;
    double spinWheel = 0;

    for (int i = 0; i < POPULATION_SIZE; i++) {
        spinWheel += fitnessScores[i];

        if (spinWheel >= rouletteWheelPosition) {
            return population[i];
        }
    }

    return population[POPULATION_SIZE - 1];
}
```

5. Melakukan crossover untuk mendapatkan offspring

```
private int[] crossover(int[] parent1, int[] parent2) {
    int[] offspring = new int[CHROMOSOME_LENGTH];
    Random random = new Random();
    int crossoverPoint = random.nextInt(CHROMOSOME_LENGTH);

    for (int i = 0; i < CHROMOSOME_LENGTH; i++) {

        if (i < crossoverPoint) {
            offspring[i] = parent1[i];
        } else {
            offspring[i] = parent2[i];
        }

    }

    for(int i=0;i<CHROMOSOME_LENGTH;i++){
        for(int j=i+1;j<CHROMOSOME_LENGTH;j++){
            if(offspring[i]==offspring[j]){
                offspring=parent1;
                break;
            }
        }
    }

    return offspring;
}
```

6. Melakukan mutasi pada offspring

```
private int[] mutate(int[] chromosome) {
    Random random = new Random();

    for (int i = 0; i < CHROMOSOME_LENGTH; i++) {
        if (random.nextDouble() <= MUTATION_RATE) {
            ArrayList<Integer> randomize= new ArrayList<>();
            for(int idxRandom=0;idxRandom<64;idxRandom++){
                randomize.add(idxRandom);
            }
            randomize.removeAll(pointMap);
            for(int idx=0;idx<i;idx++){
               for(int idxRandom=0;idxRandom<randomize.size();idxRandom++){
                   if(randomize.get(idxRandom)==chromosome[idx]){
                       randomize.remove(idxRandom);
                       break;
                   }
               }
            }
            chromosome[i]=randomize.get(random.nextInt(randomize.size()));
            break;
        }

    }
```

```
    return chromosome;
}
```

7. Memasukkan seluruh offspring menjadi populasi baru
8. Melakukan iterasi step 3–7 hingga mencapai MAX_GENERATIONS
9. Setelah seluruh iterasi selesai, maka akan dipilih salah satu offspring generasi terakhir yang memiliki fitness score paling tinggi.

Eksperimen

a. Bot *minimax* vs. manusia

No. Pertandingan	Jumlah Ronde	Skor Pemain	
		Bot *minimax*	Manusia
1	28	35	29
2	24	32	24
3	20	24	24
4	12	16	16
5	8	14	10
Kemenangan(Bot *minimax* : Manusia) = 3 : 0			

Dari eksperimen pertama, bot dengan algoritma *minimax* mendapatkan 3 kali kemenangan dan 2 seri melawan manusia.

b. Bot *local search* vs. manusia

No. Pertandingan	Jumlah Ronde	Skor Pemain	
		Bot *local search*	Manusia
1	24	29	27
2	12	16	17
3	8	12	12
Kemenangan(Bot *local search*: Manusia) = 1 : 1			

Dari eksperimen ke-2, bot dengan algoritma *local search* mendapatkan 1 kali kemenangan, 1 kali seri, dan 1 kali kalah dengan pertandingan melawan manusia.

c. Bot *minimax* vs. bot *local search*

No. Pertandingan	Jumlah Ronde	Skor Pemain	
		Bot *minimax*	Bot *local search*
1	24	30	26
2	12	18	14
3	8	13	11
Kemenangan(Bot *minimax* : *local search*) = 3 : 0			

Dari eksperimen ketiga, bot dengan algoritma *minimax* mendapatkan 3 kali kemenangan atas bot *local search*.

d. Bot *minimax* vs Bot *genetic algorithm*

No. Pertandingan	Jumlah Ronde	Skor Pemain	
		Bot *minimax*	Bot *genetic algorithm*
1	24	40	16
2	12	23	9
3	8	19	5
Kemenangan(Bot *minimax* : *genetic algorithm*) = 3 : 0			

Dari eksperimen ke-4, bot dengan algoritma *minimax* mendapatkan 3 kali kemenangan atas bot *genetic algorithm*.

e. Bot *local search* vs bot *genetic algorithm*

No. Pertandingan	Jumlah Ronde	Skor Pemain	
		Bot *local search*	Bot *genetic algorithm*
1	24	39	17
2	12	24	8
3	8	20	4
Kemenangan (Bot *local search*: *genetic algorithm*) = 3 : 0			

Dari eksperimen ke-5, bot dengan algoritma *local search* mendapatkan 3 kali kemenangan atas bot *genetic algorithm*.

Kesimpulan

Dalam konteks eksperimen yang dilakukan, hasil menunjukkan bahwa algoritma minimax menunjukkan performa yang sangat baik dibandingkan dengan algoritma genetic algorithm dan local search. Kemampuan minimax untuk memperhitungkan konsekuensi jangka panjang dari setiap langkah dalam permainan dan memilih langkah terbaik memberikan keunggulan kompetitif yang signifikan. Meskipun genetic algorithm dan local search juga mampu mengimbangi ketika melawan manusia, dalam permainan dengan banyak kemungkinan langkah dan skenario, minimax tetap menjadi pilihan terbaik.

Kontribusi Anggota Kelompok

No.	NIM	Nama	Kontribusi
1	13521099	Vieri Fajar Firdaus	Implementasi, minimax, local search, dokumen, Implementasi Genetic Algorithm
2	13521106	Saddam Annais Shaquille	Implementasi minimax, dokumen, Implementasi Genetic Algorithm
3	13521121	Mohammad Farhan Fahrezy	Implementasi minimax, dokumen, Implementasi Genetic Algorithm
4	13521160	Muchammad Dimas Sakti Widyatmaja	Implementasi minimax, dokumen, Implementasi Genetic Algorithm

Lampiran

Repository github : https://github.com/vierifirdaus/TUBES1_AI
Video eksperimen : https://bit.ly/TUCIL1AIsakti