

Tugas Besar I IF3170 Inteligensi Buatan

Minimax Algorithm and Alpha Beta Pruning in Adjacency Strategy Game



Disusun oleh :

Bernardus Willson	13521021
Raditya Naufal Abiyu	13521022
Kenny Benaya Nathan	13521023
Kartini Copa	13521026

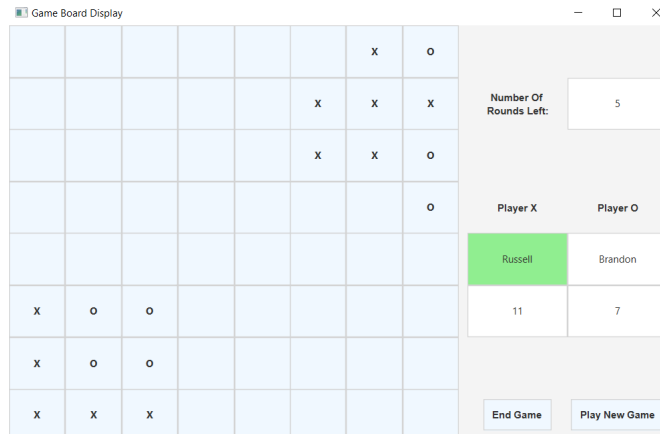
**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023**

Daftar Isi

Daftar Isi	2
Bab 1	3
Deskripsi Tugas	3
Bab 2	5
Landasan Teori	5
2.1 Algoritma Minimax Alpha Beta Pruning	5
2.2 Algoritma Local Search Simulated Annealing	7
2.3 Algoritma Genetik	8
Bab 3	10
Analisis dan Pembahasan	10
3.1 Objective Function	10
3.2 Strategi Bot dengan Minimax Alpha Beta Pruning	11
3.3 Strategi Bot dengan Local Search Simulated Annealing	16
3.4 Strategi Bot dengan Genetic Algorithm	19
Bab 4	21
Evaluasi dan Pengujian	21
4.1 Bot Minimax VS Manusia	21
4.2 Bot Local Search VS Manusia	24
4.3 Bot Minimax VS Bot Local Search	26
4.4 Bot Minimax VS Bot Genetic Algorithm	28
4.5 Bot Local Search VS Bot Genetic Algorithm	30
Bab 5	32
Kesimpulan dan Saran	32
4.1 Kesimpulan	32
4.2 Saran	32
Lampiran	33

Bab 1

Deskripsi Tugas



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 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). 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: terisi marka musuh akan berubah menjadi marka pemain.

	O	
X		
X	X	O
X	X	X

Lalu O mengisi board[1][1]

	O	
O	O	
X	O	O
X	X	X

Perhatikan bahwa kotak pada arah diagonal tidak 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.

Implementasikan bot, yaitu:

- Bot dengan algoritma *minimax alpha beta pruning*.
- Bot dengan salah satu algoritma *local search*.
- Bot dengan Genetic Algorithm *search*.

Bab 2

Landasan Teori

2.1 Algoritma Minimax Alpha Beta Pruning

Algoritma Minimax Alpha Beta Pruning adalah salah satu teknik terpenting dalam teori permainan dan kecerdasan buatan yang digunakan untuk mengambil keputusan dalam permainan strategi dengan dua pemain, seperti catur atau permainan papan lainnya. Prinsip dasar dari algoritma ini adalah untuk menghasilkan langkah terbaik yang mungkin untuk seorang pemain dengan meminimalkan kerugian potensial dalam skenario terburuk dan memaksimalkan keuntungan dalam skenario terbaik.

Algoritma Minimax Alpha Beta Pruning memiliki aplikasi luas dalam permainan dan kecerdasan buatan, terutama dalam permainan berbasis giliran di mana pemain harus membuat keputusan yang strategis. Dengan menggabungkan pemikiran strategis, pemahaman tentang preferensi lawan, dan pengoptimalan melalui Alpha Beta Pruning, algoritma ini telah berhasil digunakan dalam permainan catur, permainan video, perencanaan pergerakan robot, dan bahkan dalam keputusan bisnis yang kompleks. Meskipun memiliki keterbatasan dalam permainan dengan banyak pemain atau permainan dengan banyak aksi simultan, Algoritma Minimax Alpha Beta Pruning tetap menjadi salah satu alat yang sangat berharga dalam analisis permainan dan pengambilan keputusan.

Langkah pertama adalah membangun pohon permainan yang merepresentasikan kemungkinan langkah dan keadaan permainan. Bagi pemain, bot akan mencari langkah terbaik yang memaksimalkan nilai, sementara bagi lawannya akan dicari langkah yang meminimalkan nilai. Selama pencarian, algoritma menghitung nilai evaluasi untuk setiap keadaan permainan dan menggunakan strategi minimax untuk memutuskan langkah terbaik. Pruning alpha-beta memanfaatkan dua parameter, alpha dan beta, yang merepresentasikan nilai terbaik yang sudah diketahui untuk pemain maksimal dan minimal saat ini. Selama pencarian, algoritma secara dinamis memperbarui nilai alpha dan beta dan menghilangkan cabang-cabang yang tidak relevan, memungkinkan penghematan waktu dan ruang memori yang signifikan.

Kompleksitas waktu dari algoritma Minimax Alpha-Beta Pruning tergantung pada kedalaman pohon permainan dan faktor percabangan. Dengan asumsi bahwa pohon permainan memiliki kedalaman h , dan faktor percabangan rata-rata adalah b , kompleksitas waktu algoritma ini dapat diekspresikan sebagai $O(b^{h/2})$. Dengan kata lain, algoritma ini dapat secara signifikan mengurangi jumlah simpul yang harus dianalisis dibandingkan dengan Minimax tanpa pruning alpha-beta. Oleh karena itu, algoritma ini adalah langkah penting dalam mengoptimalkan perhitungan dalam pemrograman permainan, memungkinkan komputer untuk mengambil keputusan yang lebih cerdas dalam permainan yang melibatkan banyak kemungkinan langkah.

```
function ALPHA-BETA-SEARCH(state) returns an action
   $v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$ 
  return the action in ACTIONS(state) with value  $v$ 
```

```
function MAX-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow -\infty$ 
  for each  $a$  in ACTIONS(state) do
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    if  $v \geq \beta$  then return  $v$ 
     $\alpha \leftarrow \text{MAX}(\alpha, v)$ 
  return  $v$ 
```

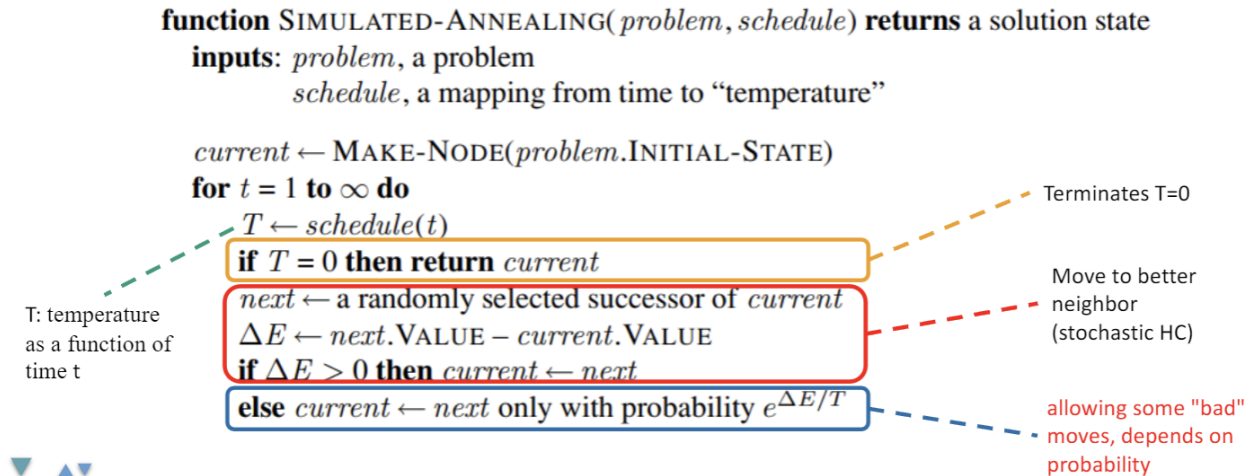
```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for each  $a$  in ACTIONS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a), \alpha, \beta))$ 
    if  $v \leq \alpha$  then return  $v$ 
     $\beta \leftarrow \text{MIN}(\beta, v)$ 
  return  $v$ 
```

2.2 Algoritma Local Search Simulated Annealing

Algoritma Simulated Annealing (SA) adalah algoritma Local Search yang terinspirasi oleh konsep pendinginan logam dalam proses manufaktur yang disebut "annealing." Tujuan utama dari SA adalah menemukan solusi yang mendekati optimum global dalam ruang pencarian masalah, terutama ketika ruang pencarian ini besar dan kompleks. Konsep utama di balik SA adalah kemampuan untuk menerima solusi yang lebih buruk tergantung pada suhu saat pencarian berlangsung, yang kemudian perlahan-lahan "mendingin" seiring berjalannya waktu. Pertama-tama, SA memulai pencarian dengan solusi awal yang mungkin suboptimal. Selanjutnya, algoritma ini mengikuti pendekatan iteratif untuk meningkatkan solusi tersebut.

SA menjaga sebuah parameter yang disebut "suhu" yang menandakan tingkat penerimaan solusi yang lebih buruk. Pada awalnya, suhu tinggi, sehingga SA cenderung menerima perubahan yang mengarah ke solusi yang lebih buruk. Namun, seiring berjalannya waktu, suhu ini turun secara eksponensial, mengurangi kemungkinan menerima solusi yang lebih buruk seiring konvergensi menuju optimum global. Konsep "pendinginan" ini adalah salah satu aspek paling krusial dari SA, karena memungkinkan algoritma ini untuk keluar dari local optima dan menjelajahi ruang pencarian secara lebih luas. SA juga memerlukan fungsi evaluasi (objective function) yang digunakan untuk mengukur kualitas solusi. Selain itu, SA melibatkan operator perpindahan yang menghasilkan solusi baru dengan memodifikasi solusi saat ini. Solusi ini kemudian dibandingkan dengan solusi sebelumnya, dan berdasarkan perbedaan nilai fungsi objektif dan suhu saat ini, solusi baru dapat diterima atau ditolak.

Algoritma Simulated Annealing telah diterapkan secara luas pada berbagai masalah optimisasi seperti penjadwalan, perencanaan, dan desain. Keunggulan utama dari SA adalah kemampuannya untuk mengatasi masalah dengan berbagai tingkat kompleksitas dan ketidaklinieran, serta kemampuannya untuk menghindari local optima. Keunggulan tersebut menjadi alasan utama pemilihan algoritma Simulated Annealing untuk dijadikan algoritma local search terbaik untuk game ini, mengingat ada banyak sekali kemungkinan yang dapat dihasilkan di setiap langkah. Pada bab-bab berikutnya, kita akan menjelajahi lebih lanjut konsep-konsep penting dalam SA, teknik tuning, dan studi kasus aplikasinya dalam berbagai konteks.



2.3 Algoritma Genetik

Algoritma Genetik adalah salah satu teknik untuk menyelesaikan masalah optimisasi dan pencarian yang kompleks dengan mengadopsi konsep dasar dari proses evolusi genetik dalam populasi organisme. Algoritma ini telah terbukti efektif dalam berbagai aplikasi seperti optimasi parameter, desain mesin, pengelompokan, dan bahkan pembelajaran mesin. Dalam Algoritma Genetik, setiap solusi masalah direpresentasikan sebagai individu dalam populasi dan diperlakukan sebagai kromosom. Populasi ini kemudian dievolusi melalui beberapa generasi untuk menghasilkan solusi yang semakin baik.

Satu aspek kunci dalam Algoritma Genetik adalah representasi kromosom dan pemilihan fungsi evaluasi. Representasi kromosom harus sesuai dengan jenis masalah yang dihadapi, dan fungsi evaluasi digunakan untuk mengukur kualitas setiap solusi. Populasi solusi pertama kali diinisialisasi dengan cara acak atau berdasarkan pengetahuan awal. Kemudian, iteratif, algoritma beroperasi melalui serangkaian generasi, dengan setiap generasi menghasilkan keturunan baru melalui operasi genetik seperti seleksi, *crossover* (persilangan), dan mutasi. Prinsip dasar di balik Algoritma Genetik adalah memberikan peluang lebih besar bagi solusi yang lebih baik untuk bertahan hidup dan mewariskan ciri-ciri unggul mereka kepada generasi berikutnya.

Salah satu keuntungan utama Algoritma Genetik adalah kemampuannya untuk menjelajahi ruang pencarian secara global dan menemukan solusi yang mendekati optimum global. Namun, Algoritma Genetik juga memiliki beberapa tantangan, seperti parameter yang

sensitif, kemungkinan terjebak dalam local optima, dan kebutuhan komputasi yang tinggi untuk masalah yang kompleks. Pemilihan operator genetik, ukuran populasi, dan kriteria berhenti adalah faktor-faktor penting dalam perancangan Algoritma Genetik yang mempengaruhi kinerjanya.

function GENETIC-ALGORITHM(*population*, FITNESS-FN) **returns** an individual

inputs: *population*, a set of individuals

FITNESS-FN, a function that measures the fitness of an individual

repeat

new_population \leftarrow empty set

for $i = 1$ **to** SIZE(*population*) **do**

$x \leftarrow$ RANDOM-SELECTION(*population*, FITNESS-FN)

$y \leftarrow$ RANDOM-SELECTION(*population*, FITNESS-FN)

$child \leftarrow$ REPRODUCE(x, y)

if (small random probability) **then** $child \leftarrow$ MUTATE($child$)

add *child* to *new_population*

population \leftarrow *new_population*

until some individual is fit enough, or enough time has elapsed

return the best individual in *population*, according to FITNESS-FN

Bab 3

Analisis dan Pembahasan

3.1 Objective Function

Objective function yang digunakan dalam kode adalah fungsi `evaluateBoard(int[][] board, int player, int opponent)`. Fungsi ini merupakan fungsi abstrak dimana akan diimplementasikan secara berbeda sesuai dengan karakteristik dari masing-masing bot. Fungsi ini bertanggung jawab untuk mengevaluasi papan permainan, dengan kata lain fungsi ini dapat menunjukkan keunggulan suatu posisi atau bisa digunakan untuk memperhitungkan langkah-langkah yang akan dilakukan setelahnya. Fungsi ini merupakan bagian penting dari algoritma yang digunakan dalam permainan.

Berikut adalah alasan pembuatan fungsi `evaluateBoard` dan penggunaannya:

- Mengukur keunggulan pemain: Fungsi ini digunakan untuk mengukur seberapa baik posisi pemain dalam permainan. Dengan menghitung selisih antara jumlah tanda yang dimiliki oleh pemain dengan jumlah tanda lawan, kita dapat mengukur sejauh mana pemain unggul dalam permainan. Jika selisihnya positif, berarti pemain memiliki keunggulan; jika negatif, berarti pemain tertinggal.
- Pengaruh terhadap evaluasi Minimax: Dalam algoritma Minimax, fungsi ini digunakan untuk menilai hasil dari setiap cabang gerakan yang mungkin dilakukan oleh pemain atau lawan. Nilai yang dihasilkan dari fungsi ini adalah selisih tanda milik pemain dan milik lawan. Hasil dari fungsi ini akan digunakan dalam pemilihan langkah terbaik dalam algoritma Minimax. Jika hasil positif, pemain cenderung memilih langkah tersebut; jika negatif, pemain akan mencoba menghindari langkah tersebut.
- Pengaruh terhadap evaluasi Local Search: Dalam algoritma local search, fungsi ini digunakan untuk menghitung tanda kita dimana jika hasilnya lebih besar dibanding nilai sebelumnya maka pemain memiliki keunggulan dan sebaliknya.

- Pengaruh terhadap evaluasi Genetic: Dalam algoritma genetic, fungsi `evaluateBoard` digunakan dalam perhitungan move terbaik dimana fungsi `evaluateBoard` sendiri mengembalikan total poin yang dimiliki oleh player

Dengan demikian, fungsi `evaluateBoard` memiliki peran penting dalam mengevaluasi keadaan permainan, membantu pemain membuat keputusan, dan menentukan hasil akhir permainan. Hal ini adalah salah satu komponen kunci dalam algoritma untuk permainan strategi seperti yang digunakan dalam kode program ini.

3.2 Strategi Bot dengan Minimax Alpha Beta Pruning

Langkah 1: Inisialisasi

Permainan akan memanggil fungsi `move` pada Bot dengan kedalaman yang sudah ditentukan. Kedalaman akan sejumlah dengan *round* yang sudah ditentukan oleh *user*. Jika *round* yang dipilih lebih dari 4, maka kedalaman akan hanya sejumlah 4 karena waktu *response* dari bot akan menjadi cukup lama. Fungsi ini akan memulai fungsi utama dari algoritma minimax. Kedalaman di sini akan berfungsi sebagai basis dari algoritma rekursif minimax. `move(int[][] board, int player)`: metode utama yang digunakan untuk memilih langkah oleh bot. Ini akan mengevaluasi semua langkah yang mungkin dan memilih langkah terbaik dengan memanggil metode `minimax`.

Langkah 2: Mengiterasi setiap sel di papan

Bot melakukan iterasi melalui setiap sel di papan permainan yang masih kosong (dilambangkan oleh 0) untuk mencari langkah yang mungkin untuk dijalankan. Bot akan berhenti melakukan iterasi jika sudah mencapai seluruh sel atau *response time* dari bot sudah lama.

Langkah 3: Pengambilan Salinan Papan

Sebelum mencoba langkah berikutnya, bot akan membuat salinan papan saat ini. Ini dilakukan untuk menjaga keadaan papan saat ini tidak berubah selama proses pencarian.

Langkah 4: Rekursi

Jika permainan belum mencapai kondisi berhenti, algoritma akan masuk ke tahap rekursi. Untuk setiap langkah yang mungkin, bot akan memanggil fungsi minimax (dirinya sendiri) dengan papan yang diperbarui (setelah langkah tersebut diambil) dan kedalaman pencarian yang berkurang. Fungsi minimax adalah metode rekursif yang mengimplementasikan algoritma Minimax dengan pemotongan alpha-beta. Ini akan menghitung skor yang sesuai dengan nilai objective function pada papan dan menghasilkan skor terbaik yang mungkin. Rekursi ini dibedakan berdasarkan giliran pemain saat ini. Jika giliran pemain adalah pemain pertama (player 1), algoritma akan mencari poin yang paling maksimum dari simpul anak. Sebaliknya, jika giliran pemain adalah pemain kedua (player 2), algoritma akan mencari poin yang paling minimum. Pencarian simpul anak akan memanggil kembali fungsi Minimax dengan parameter kedalaman yang dikurangi 1 dari sebelumnya. Selama pencarian ini, algoritma juga akan memperhitungkan bahwa setiap penempatan simbol akan mengubah simbol lawan di kotak sekitarnya.

```

public int[] move(int[][] board, int player) {
    int[] move = new int[2];
    int bestScore = Integer.MIN_VALUE;
    int opponent = (player == 1) ? 2 : 1;

    this.timeLimitReached = false;
    this.startTime = System.nanoTime();
    for (int i = 0; i < board.length && !this.timeLimitReached; i++) {
        for (int j = 0; j < board[i].length; j++) {
            if (board[i][j] == 0) {
                int[][] tempBoard = copyBoard(board);
                tempBoard[i][j] = player;

                captureMarks(tempBoard, i, j, player, opponent);

                int score = minimax(tempBoard, 0, false, Integer.MIN_VALUE, Integer.MAX_VALUE, player, opponent);
                if (this.timeLimitReached) {
                    break;
                }
                if (score > bestScore) {
                    bestScore = score;
                    move[0] = i;
                    move[1] = j;
                }
            }
        }
    }
}

```

```

1 private int minimax(int[][] board, int depth, boolean isMaximizing, int alpha, int beta, int player, int opponent) {
2     /*
3      * If the time limit has been reached, it will break out of tree
4      * If not, it will check if the time elapsed is greater than 5 seconds
5      */
6     if (this.timeLimitReached) {
7         return -1;
8     }
9     else {
10         if (getTimeElapsed() / 1000000 > 5000) {
11             this.timeLimitReached = true;
12             return -1;
13         }
14     }
15
16     int result = evaluateBoard(board, player, opponent);
17
18     if (depth == MAX_DEPTH) {
19         return result;
20     }
21
22     if (isMaximizing) {
23         int bestScore = Integer.MIN_VALUE;
24         for (int i = 0; i < board.length; i++) {
25             for (int j = 0; j < board[i].length; j++) {
26                 if (board[i][j] == 0) {
27                     int[][] tempBoard = copyBoard(board);
28                     tempBoard[i][j] = player;
29
30                     // Capture opponent's marks
31                     captureMarks(tempBoard, i, j, player, opponent);
32
33                     int score = minimax(tempBoard, depth + 1, false, alpha, beta, player, opponent);
34                     bestScore = Math.max(score, bestScore);
35                     alpha = Math.max(alpha, bestScore);
36
37                     if (beta <= alpha) {
38                         break; // Beta cutoff
39                     }
40                 }
41             }
42         }
43         return bestScore;
44     } else {
45         int bestScore = Integer.MAX_VALUE;
46         for (int i = 0; i < board.length; i++) {
47             for (int j = 0; j < board[i].length; j++) {
48                 if (board[i][j] == 0) {
49                     int[][] tempBoard = copyBoard(board);
50                     tempBoard[i][j] = opponent;
51
52                     captureMarks(tempBoard, i, j, opponent, player);
53
54                     int score = minimax(tempBoard, depth + 1, true, alpha, beta, player, opponent);
55                     bestScore = Math.min(score, bestScore);
56                     beta = Math.min(beta, bestScore);
57
58                     if (beta <= alpha) {
59                         break;
60                     }
61                 }
62             }
63         }
64         return bestScore;
65     }
66 }

```

Langkah 5: Kondisi Berhenti

Saat kedalaman pencarian mencapai batas maksimal kedalaman, proses pencarian akan berhenti, dan skor yang dihasilkan akan digunakan sebagai skor heuristik. Ini bertujuan untuk menghindari pencarian yang terlalu dalam.

Ketika sedang mengunjungi *children* baru, program akan mengecek apabila waktu algoritma bot sudah lebih dari 5 detik. Jika sudah lebih dari 5 detik, maka bot akan keluar dari seluruh pencarian algoritma *minimax alpha-beta pruning* dan akan melakukan algoritma *greedy* untuk menentukan langkah selanjutnya. Algoritma *greedy* pada program ini akan memberikan langkah selanjutnya (*neighbour*) dengan objective function tertinggi.

```

1  /*
2   * fallbackPlan() is called when the time limit has been reached
3   * It will use the Greedy algorithm to find the best move
4   * because it takes less than 1 ms.
5   */
6  private void fallbackPlan(int[][] board, int player, int[] move) {
7      int currentScore = countMarks(board, player);
8      int bestScore = currentScore;
9      int opponent = (player == 1) ? 2 : 1;
10
11     for (int i = 0; i < board.length; i++) {
12         for (int j = 0; j < board[i].length; j++) {
13             if (board[i][j] == 0) {
14                 int[][] tempBoard = copyBoard(board);
15                 tempBoard[i][j] = player;
16
17                 captureMarks(tempBoard, i, j, player, opponent);
18
19                 int newScore = countMarks(board, player);
20
21                 if (newScore > bestScore) {
22                     bestScore = newScore;
23                     move[0] = i;
24                     move[1] = j;
25                 }
26             }
27         }
28     }
29 }

```

Langkah 6: Maksimisasi dan Minimisasi

Proses pencarian dijalankan dalam dua mode yaitu, mode maksimisasi dan mode minimisasi. Saat di mode maksimisasi, bot mencoba langkah-langkah yang paling menguntungkan untuk pemain yang sedang dievaluasi (pemain 1), sementara dalam mode minimisasi, bot mencoba langkah-langkah yang paling merugikan untuk lawan (pemain 2).

Langkah 7: Alpha-Beta Prunning

Saat dalam mode maksimisasi, bot menghitung skor terbaik (maksimum) yang dapat dicapai, dan saat dalam mode minimisasi, bot menghitung skor terburuk (minimum) yang dapat dicapai. Pemotongan alpha-beta terjadi saat bot menemukan bahwa ada langkah yang lebih buruk daripada langkah sebelumnya ($\beta \leq \alpha$ dalam mode maksimisasi atau $\alpha \geq \beta$ dalam mode minimisasi). Dalam hal ini, pencarian dicabut (*prunning*) karena tidak ada keuntungan lebih lanjut yang dapat diperoleh. Jika giliran pemain adalah pemain pertama (player 1), algoritma akan memperbaharui nilai "alfa" dengan nilai evaluasi untuk setiap simpul anak jika nilai tersebut lebih besar daripada "alfa" saat ini. Sebaliknya, jika giliran pemain adalah pemain kedua (player 2), algoritma akan memperbaharui nilai "beta" dengan nilai evaluasi untuk setiap simpul anak jika nilai tersebut lebih kecil daripada "beta" saat ini. Selanjutnya, algoritma akan memeriksa apakah terdapat kondisi di mana "beta" lebih kecil atau sama dengan "alfa." Jika kondisi ini terpenuhi, pencarian akan dihentikan untuk simpul anak lain dari simpul tersebut.

Langkah 8: Penentuan Langkah Terbaik

Setelah pencarian dilakukan untuk semua langkah yang mungkin, bot memilih langkah yang memiliki skor terbaik sesuai dengan mode (maksimisasi atau minimisasi). Langkah ini akan dianggap sebagai langkah terbaik untuk bot.

Langkah 9: Evaluasi Kondisi Akhir

`evaluateBoard(int[][] board, int player, int opponent):` Metode ini digunakan untuk menghitung skor objective function berdasarkan papan saat ini. Dalam kasus ini, skor dihitung sebagai selisih antara jumlah *marks* pemain dan lawan.

Langkah 10: Perulangan

Bot akan terus mengulangi langkah-langkah ini pada setiap giliran permainan hingga permainan selesai atau mencapai batas ronde yang telah ditetapkan.

3.3 Strategi Bot dengan Local Search Simulated Annealing

Proses pencarian dilakukan dengan salah satu algoritma Local Search yaitu, algoritma Simulated Annealing. Simulated Annealing merupakan algoritma local search yang cocok untuk strategi pencarian dalam Adversial Adjacency Strategy Game karena kemampuannya untuk menjelajahi ruang pencarian yang luas, mengatasi local optima, dan membuat keputusan adaptif.

Langkah 1: Inisialisasi

- Inisialisasi papan permainan (*game board*) dengan konfigurasi awal, termasuk peletakan awal bidak X dan O sesuai dengan aturan permainan.
- Penentuan pemain yang akan mulai (misalnya, pemain X).
- Inisialisasi parameter, seperti temperatur awal (INITIAL_TEMPERATURE), temperatur akhir (FINAL_TEMPERATURE), tingkat pendinginan (COOLING_RATE), dan jumlah iterasi maksimum (MAX_ITERATIONS).

Langkah 2: Evaluasi Fungsi Tujuan (*Objective Function*)

Implementasi fungsi evaluasi yang akan mengukur kualitas setiap konfigurasi papan. Ini adalah fungsi yang akan digunakan untuk menilai apakah langkah yang diambil oleh algoritma merupakan perbaikan atau tidak. Dalam permainan *Adjacency Strategy Game*, evaluasi fungsi bisa mempertimbangkan jumlah bidak yang dimiliki oleh pemain saat ini, seberapa dekat bidak-bidak tersebut dengan bidak lawan, atau aturan khusus lain yang sesuai dengan permainan.

Langkah 3: Generate Tetangga

Bot memilih langkah acak dengan memilih satu sel acak di papan yang kosong (nilai 0) dan mencoba menempatkan tanda (*marks*) pemain di sana. Bot kemudian mengevaluasi bagaimana ini mempengaruhi skor.

- Setiap iterasi, memilih acak satu langkah yang mungkin (i, j) pada papan (board).

- Membuat salinan papan (tempBoard) dengan melakukan langkah tersebut.
- Evaluasi papan yang telah diubah (tempBoard) dengan fungsi evaluasi untuk menghitung skor baru (newScore).

Langkah 4: Perulangan dan Temperatur:

Pada tahap "Perulangan dan Temperatur", algoritma Simulated Annealing menjalankan iterasi untuk mencari solusi terbaik dalam ruang pencarian. Ini dilakukan dengan membatasi jumlah iterasi maksimum sesuai dengan nilai MAX_ITERATIONS yang telah ditetapkan atau hingga suhu (temperature) menurun di bawah nilai FINAL_TEMPERATURE. Suhu awal (INITIAL_TEMPERATURE) menggambarkan seberapa besar perubahan drastis diterima pada awal pencarian.

Selama setiap iterasi, temperatur dikurangi dengan faktor COOLING_RATE. Penurunan temperatur bertujuan untuk mengendalikan probabilitas penerimaan solusi yang lebih buruk seiring berjalannya waktu. Semakin rendah temperatur, semakin ketat kriteria penerimaan, sehingga algoritma akan lebih cenderung menerima solusi yang lebih baik. Dengan demikian, proses Simulated Annealing memungkinkan eksplorasi solusi di awal (saat temperatur masih tinggi) dan konvergensi ke solusi yang lebih baik seiring berjalannya waktu (saat temperatur semakin mendekati FINAL_TEMPERATURE). Hal ini membuat algoritma mampu menemukan solusi yang mungkin tidak ditemukan oleh pencarian sederhana.

Langkah 5: Menerima atau Menolak Tetangga

Bot memutuskan apakah akan menerima atau menolak langkah berdasarkan perbandingan antara skor evaluasi langkah saat ini dan langkah sebelumnya, serta temperatur saat ini.

- Jika skor baru (newScore) lebih baik (lebih tinggi) daripada skor saat ini (currentScore), maka solusi baru diterima sebagai solusi saat ini.
- Jika skor baru lebih buruk daripada skor saat ini, maka masih ada peluang solusi baru diterima berdasarkan probabilitas.
- Probabilitas menerima solusi baru dihitung menggunakan fungsi probabilitas Boltzmann: $\text{probability} = \exp((\text{newScore} - \text{currentScore}) / \text{temperature})$.
- Solusi baru diterima jika $\text{random.nextDouble()} < \text{probability}$.

```
private boolean acceptNewSolution(int currentScore, int newScore, double temperature) {  
    if (newScore > currentScore) {  
        return true;  
    }  
    double probability = Math.exp((newScore - currentScore) / temperature);  
    return random.nextDouble() < probability;  
}
```

Langkah 6: Kondisi Berhenti

Algoritma Simulated Annealing memiliki dua kriteria berhenti. Pertama, pencarian akan berhenti jika jumlah iterasi maksimum (MAX_ITERATIONS) yang telah ditentukan telah tercapai. Ini bertujuan untuk memastikan bahwa pencarian tidak berlanjut secara tak terbatas dan memiliki batasan waktu tertentu. Kedua, pencarian akan berhenti jika suhu (temperature) turun di bawah nilai FINAL_TEMPERATURE yang telah ditentukan. Hal ini mengindikasikan bahwa algoritma telah mencapai konvergensi, dan karena suhu sangat rendah, solusi yang lebih buruk kemungkinan besar tidak akan diterima lagi. Kriteria berhenti ini membantu algoritma Simulated Annealing menghasilkan solusi terbaik yang memadai dalam waktu yang masuk akal, dengan mengkombinasikan eksplorasi awal dan konvergensi ke solusi yang lebih baik seiring berjalannya waktu.

Langkah 7: Solusi Terbaik

Solusi Terbaik adalah output akhir dari algoritma Simulated Annealing setelah menjalani serangkaian iterasi dan mengikuti aturan probabilitas penerimaan solusi yang mungkin lebih buruk. Solusi ini merujuk pada langkah yang dianggap optimal atau paling baik yang ditemukan oleh algoritma dalam konteks permainan. Solusi ini akan digunakan sebagai langkah berikutnya dalam permainan Adjacency Strategy Game, dengan harapan bahwa solusi ini akan mengoptimalkan posisi pemain sesuai dengan strategi yang telah diterapkan dalam algoritma Simulated Annealing. Solusi terbaik ini mencerminkan hasil pencarian berdasarkan pengkombinasian eksplorasi awal dengan konvergensi ke solusi yang lebih baik seiring berjalannya waktu dan pendinginan suhu.

3.4 Strategi Bot dengan Genetic Algorithm

Genetic Algorithm (GA) bukanlah pendekatan yang umum digunakan untuk pencarian langkah optimum dalam permainan *adjacency strategy game*. Representasi setiap langkah akan sangat besar dan kompleks. Selain itu, evaluasi perlu memperhitungkan banyak faktor, seperti kontrol sudut, mobilitas, dan stabilitas bidak, yang tidak dapat direpresentasikan secara efektif dalam sebuah fungsi tujuan dalam GA. Jumlah kemungkinan langkah yang dapat diambil dalam setiap posisi bisa sangat besar, sehingga GA akan membutuhkan waktu komputasi yang sangat besar untuk mengevaluasi semua kemungkinan. Ini akan menghasilkan algoritma yang sangat lambat. Terakhir, *Genetic Algorithm* dapat digunakan untuk mengoptimalkan parameter dalam model pemain AI, misalnya, dalam algoritma evaluasi permainan yang digunakan oleh bot. Namun, itu berbeda dari penggunaan GA untuk mencari langkah-langkah optimal dalam permainan secara langsung.

Langkah 1: Inisialisasi

Pada tahap awal ini, akan ditetapkan jumlah maksimum generasi dan pohon reservasi dengan nilai simpul akar $-\infty$.

- Secara acak, bangunkan sebuah populasi awal dari game state.
- Setiap game state mewakili pengaturan yang mungkin dari X dan O pada papan.
- Evaluasi skor setiap keadaan berdasarkan jumlah kotak yang dimiliki.

```
private List<int[]> generateOffspring(List<int[]> population) {
    List<int[]> offspring = new ArrayList<>();

    for (int i = 0; i < population.size(); i++) {
        int[] parent1 = selectParent(population);
        int[] parent2 = selectParent(population);
        int[] child = crossover(parent1, parent2);
        mutate(child);
        offspring.add(child);
    }

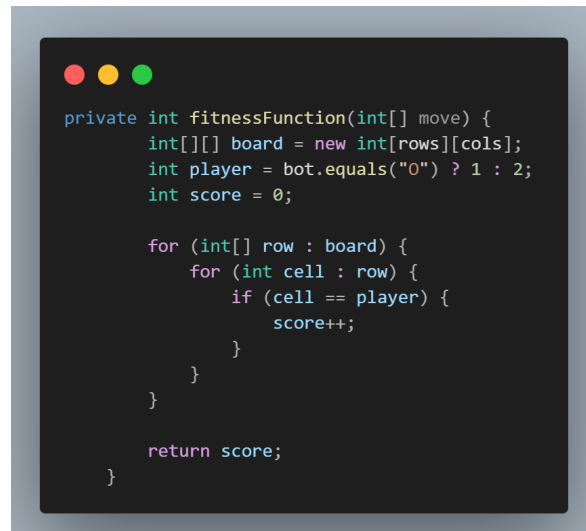
    return offspring;
}
```

Langkah2: *Offspring Evaluation*

Algoritma memilih dua orangtua (parent) dari populasi saat ini secara acak dan kemudian membuat keturunan (offspring) dengan melakukan crossover (percampuran) dan mutasi. Crossover menggabungkan gen-gen dari dua orangtua untuk menciptakan anak yang baru, sementara mutasi menghasilkan perubahan acak dalam anak.

Langkah 3: *Fitness Evaluation*

Setiap langkah dalam populasi, termasuk keturunan yang baru, dievaluasi berdasarkan seberapa baik langkah tersebut dalam mencapai tujuan permainan. Dalam kode ini, evaluasi dilakukan dengan menggunakan fungsi `evaluateBoard` yang menghitung skor pemain berdasarkan langkah-langkah yang diambil.



```
private int fitnessFunction(int[] move) {
    int[][] board = new int[rows][cols];
    int player = bot.equals("O") ? 1 : 2;
    int score = 0;

    for (int[] row : board) {
        for (int cell : row) {
            if (cell == player) {
                score++;
            }
        }
    }

    return score;
}
```

Langkah 4: *Selection*

Populasi yang termasuk langkah-langkah terbaik dipilih untuk menjadi populasi berikutnya. Langkah-langkah yang memiliki nilai fungsi objektif (*fitness*) tertinggi dipertahankan dalam populasi berikutnya, sementara yang lainnya dihapus.

Langkah 6: Pemilihan Langkah Terbaik

Setelah beberapa generasi, algoritma memilih langkah terbaik dari populasi saat ini untuk digunakan sebagai langkah berikutnya dalam permainan. Langkah terbaik ini dipilih berdasarkan evaluasi fitness terhadap situasi permainan yang ada.

Bab 4

Evaluasi dan Pengujian

4.1 Bot Minimax VS Manusia

Pada ronde 8, 10, dan 12, Bot Minimax berhasil memenangkan permainan melawan manusia. Ini menunjukkan bahwa Bot Minimax mampu menghasilkan langkah-langkah yang lebih baik daripada manusia pada ronde-rondeanya. Bot Minimax mungkin menggunakan algoritma Minimax, yang adalah salah satu pendekatan yang digunakan dalam permainan strategi untuk menghitung langkah terbaik yang dapat diambil dengan mempertimbangkan berbagai skenario permainan yang mungkin dan mencari langkah yang memberikan hasil terbaik. Namun, pada ronde 28, manusia memenangkan permainan, menunjukkan bahwa keputusan manusia lebih baik daripada Bot Minimax dalam situasi ini. Ini bisa terjadi karena kecerdasan manusia yang lebih adaptif dan kemampuan untuk membuat strategi yang lebih kompleks yang tidak dapat diantisipasi oleh Bot Minimax. Terkadang, manusia dapat menggunakan kejutan atau taktik yang tidak terduga, yang sulit diprediksi oleh program komputer.

4.1.1 8 Ronde

		o	o	x	o	x	o	<div style="text-align: right; padding-right: 10px;"> Number Of Rounds Left: 0 </div> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> Player X Player O </div> <table border="1" style="width: 100%; text-align: center;"> <tr> <td style="background-color: #00FFFF;">Minimax (Winner!)</td> <td>Human</td> </tr> <tr> <td>13</td> <td>10</td> </tr> </table> <div style="display: flex; justify-content: space-around; margin-top: 10px;"> End Game Play New Game </div>	Minimax (Winner!)	Human	13	10
Minimax (Winner!)	Human											
13	10											
		x	o	x	o	x	x					
		o	x	o	x	x	x					
x	x											
x	o	o										

4.1.2 9 Ronde

		O	O	X	O	X	O
		X	O	X	O	X	O
		X	X	O	X	O	X
		X					X
X	O	O					
X	X						

Number Of Rounds Left:

Player X Player O

Minimax (Winner!) Human

14 11

End Game Play New Game

4.1.3 10 Ronde

		O	O	X	O	X	O
		X	O	X	O	X	X
		O	X	O	X	X	X
O	X	X					
O	X	X					
X	O	O					

Number Of Rounds Left:

Player X Player O

Minimax (Winner!) Human

15 12

End Game Play New Game

4.1.4 12 Ronde

				X	X	X	O	Number Of Rounds Left: <input type="text" value="0"/>
				X	O	O	X	
				X	O	X	X	
				X	X	X	X	
O	X	O	O					Player X Player O
X	O	X	O					Minimax (Winner!) Human
O	X	X						18 13
X	O	O	O					End Game Play New Game

4.1.5 28 Ronde

X	X	X	X	O	X	X	O	Number Of Rounds Left: <input type="text" value="0"/>
X	O	X	O	O	O	O	X	
X	O	O	O	O	O	O	O	
X	X	O	O	O	X	O	X	
X	O	O	O	O	O	X	X	Player X Player O
X	O	X	O	O	O		X	Minimax Human (Winner!)
O	X	O	X	O	X	X	X	29 34
X	O	X	X	O	O	O	X	End Game Play New Game

4.2 Bot Local Search VS Manusia

Dalam permainan Adversial Adjacency Strategy Game, Bot Simulated Annealing dan manusia bersaing dalam ronde-rondenya. Manusia berhasil memenangkan permainan pada ronde 8 dan 10, menunjukkan kemampuan adaptif dan strategisnya dalam mengatasi bot. Namun, pada ronde 28, Bot Simulated Annealing berhasil keluar sebagai pemenang, menunjukkan kemampuannya untuk mengeksplorasi dan mengoptimalkan langkah-langkahnya seiring berjalannya waktu dengan pendekatan yang terinspirasi oleh proses pendinginan logam (annealing), sehingga memenangkan permainan tersebut.

4.2.1 8 Ronde

					O	X	O	<div>Number Of Rounds Left: 0</div> <div> <div>Player X</div> <div>Player O</div> </div> <div> <div>Local</div> <div>Human (Winner!)</div> </div> <div> <div>11</div> <div>12</div> </div> <div> <div>End Game</div> <div>Play New Game</div> </div>
				X	X	O	O	
		X						
O	X	X	O					
O	X	O	O	X	X			
X	O	O	X	O				

4.2.2 10 Ronde

						O	O	Number Of Rounds Left:	0
						X	X		
	X					X	X		
O	O	X	X					Player X	Player O
O	O	X						Local	Human (Winner!)
X	O	O	O					12	15
O	X	O	O					End Game	Play New Game
X	O	O	O	X					

4.2.3 28 Ronde

X	X	X	X	X	X	X	O	Number Of Rounds Left:	0
O	X	O	X	X	O	X	X		
X	O	O	O	O	O	X	X		
O	O	X	O	O	O	X	X	Player X	Player O
O	X	O	O	X	O	X	X	Local (Winner!)	Human
X	O	O	X	X	O	X		34	29
O	X	O	O	O	X	X	X	End Game	Play New Game
X	O	O	X	X	O	O	X		

4.3 Bot Minimax VS Bot Local Search

Dalam serangkaian pertandingan Adversarial Adjacency Strategy Game antara bot Minimax dan bot yang menggunakan algoritma Local Search Simulated Annealing selama 8 ronde, bot Minimax berhasil meraih kemenangan dalam 8 pertandingan tersebut. Namun, ketika pertandingan diperpanjang menjadi 10 ronde dan bahkan hingga 28 ronde, kedua bot akhirnya bermain dengan sangat kompetitif dan mampu menyamakan posisi, sehingga tidak ada yang meraih kemenangan dalam jangka waktu tersebut, menghasilkan kedudukan seri. Ini menunjukkan bahwa dalam pertandingan yang lebih lama, bot yang menggunakan algoritma Simulated Annealing mampu mengembangkan strateginya dan meningkatkan kemampuan adaptasinya, sehingga menjadi lawan yang semakin tangguh bagi bot Minimax yang memiliki strategi yang lebih konsisten. Hal ini mencerminkan pentingnya ketahanan, adaptabilitas, dan kesesuaian strategi dengan waktu dalam permainan yang berlanjut.

4.3.1 8 Ronde

				X	O	X	O	Number Of Rounds Left: <input type="text" value="0"/>
				O	X	O	X	
				X	O	X	O	
				X	X	O	O	
O	O							Player X Player O
O	O							Local Mini (Winner!)
O	O							10 14
X	X							End Game Play New Game

4.3.2 12 Ronde

O	O	X	O	O	X	X	O	Number Of Rounds Left:	0
			O	X	X	X	X		
				X	O	X	O		
				X	X	O	O		
X	O	O						Player X	Player O
O	X							Local	Mini
O	O							16	16
X	O	X	X					End Game	Play New Game

4.3.3 28 Ronde

X	O	X	O	X	X	X	O	Number Of Rounds Left:	0
X	X	O	O	O	O	X	X		
X	X	O	X	O	X	O	O		
X	O	X	O	O	X	X	X		
X	X	O	O	X	O	X	O	Player X	Player O
O	O	X	X	O	O	O	X	Local	Mini
O	X	O	X	O	X	O	O	32	32
X	O	X	O	O	O	X	X	End Game	Play New Game

4.4 Bot Minimax VS Bot Genetic Algorithm

Dalam serangkaian pertandingan Adversarial Adjacency Strategy Game antara bot minimax dan bot yang menggunakan algoritma genetik (genetic algorithm) selama 8, 10, dan bahkan 28 ronde, bot minimax berhasil meraih kemenangan dalam setiap ronde tersebut. Bot Minimax beroperasi dengan metode yang kuat, yakni algoritma Minimax dengan pemotongan alpha-beta, yang memungkinkan pemilihan langkah yang optimal dengan mempertimbangkan pergerakan pemain dan lawan. Kemenangan dalam 8, 10, dan 28 pertandingan berturut-turut menunjukkan superioritas strategi bot Minimax dalam menghadapi bot yang menggunakan algoritma Genetic. Ini menegaskan kemampuan bot Minimax dalam mengevaluasi dan merencanakan pergerakan dengan cermat untuk mencapai hasil terbaik dalam permainan Adversarial Adjacency Strategy Game, serta mengukuhkan statusnya sebagai strategi yang paling kuat dan efektif dalam konteks ini.

4.4.1 8 Ronde

				O		O	O	Number Of Rounds Left:	0
				O		O	O		
O									
O	O	O		O	O	X			
X						O		Player X	Player O
								Genetic	Mini (Winner!)
O	O		X					5	19
O	O								
X	O	O			X			End Game	Play New Game

4.4.2 10 Ronde

		O	X			O	O	<div>Number Of Rounds Left: 0</div> <div> <div>Player X</div> <div>Player O</div> </div> <div> <div>Genetic</div> <div>Mini (Winner!)</div> </div> <div> <div>9</div> <div>19</div> </div> <div> <div>End Game</div> <div>Play New Game</div> </div>
	X		X		O	O	O	
X						O		
O						X	O	
O						O	O	
O	O					X	O	
O	O		O					
X	X		O				X	

4.4.3 28 Ronde

O	O	O	X	O	O	O	O	<div>Number Of Rounds Left: 0</div> <div> <div>Player X</div> <div>Player O</div> </div> <div> <div>Genetic</div> <div>Mini (Winner!)</div> </div> <div> <div>27</div> <div>37</div> </div> <div> <div>End Game</div> <div>Play New Game</div> </div>
O	O	X	O	X	O	O	X	
X	O	X	X	X	O	O	O	
O	O	X	X	X	X	O	X	
O	O	X	X	O	O	O	O	
O	X	X	O	O	O	O	X	
O	X	X	O	O	O	X	X	
X	O	X	X	O	O	X	X	

4.5 Bot Local Search VS Bot Genetic Algorithm

Selama 8 ronde, bot genetik berhasil meraih kemenangan dalam 8 pertandingan tersebut. Namun, ketika pertandingan diperpanjang menjadi 10 ronde dan bahkan hingga 28 ronde, bot Local Search Simulated Annealing mampu menunjukkan daya saing yang semakin kuat dan berhasil meraih kemenangan dalam pertandingan tersebut. Kemenangan bot Local Search Simulated Annealing dalam pertandingan yang lebih lama mencerminkan kemampuannya untuk terus-menerus meningkatkan strategi dan penyesuaian selama permainan berlangsung. Ini menunjukkan bahwa, dalam jangka panjang, bot Local Search Simulated Annealing mampu mengungguli bot Genetic dalam hal adaptabilitas dan peningkatan kinerja seiring berjalannya waktu, menggambarkan pentingnya strategi berbasis pencarian lokal dalam permainan yang berlanjut.

4.5.1 8 Ronde

			X		X	X	O	Number Of Rounds Left: <input type="text" value="0"/>
		O	X	X	X	X	X	
		O	O			X	X	
X	X							
	O							Player X
	X	X						Player O
X	O							<input type="button" value="Local"/>
X	X					O		<input checked="" type="button" value="Genetic"/>
								<input type="text" value="17"/>
								<input type="text" value="7"/>
								<input type="button" value="End Game"/>
								<input type="button" value="Play New Game"/>

4.5.2 10 Ronde

				O		O	O	<div>Number Of Rounds Left: 0</div> <div> <div>Player X</div> <div>Player O</div> </div> <div> <div>Genetic</div> <div>Local</div> </div> <div> <div>6</div> <div>22</div> </div> <div> <div>End Game</div> <div>Play New Game</div> </div>
O	O			O		O	O	
O			O	O	X			
X				O			O	
X				O	O	O	O	
X	O							
O	O				X			
X	O	O						

4.5.3 28 Ronde

O	O	X	O	O	O	X	O	<div>Number Of Rounds Left: 0</div> <div> <div>Player X</div> <div>Player O</div> </div> <div> <div>Genetic</div> <div>Local (Winner!)</div> </div> <div> <div>24</div> <div>40</div> </div> <div> <div>End Game</div> <div>Play New Game</div> </div>
O	X	O	O	O	O	O	O	
X	O	O	X	X	O	O	O	
O	X	X	O	O	X	X	O	
O	O	X	O	O	X	X	X	
O	X	X	O	X	X	O	O	
O	X	O	O	O	X	X	O	
X	O	X	O	O	O	O	X	

Bab 5

Kesimpulan dan Saran

4.1 Kesimpulan

Dalam tugas ini, kami telah mengevaluasi beberapa strategi bot dalam permainan Adjacency Strategy Game. Hasil evaluasi kami menunjukkan bahwa bot minimax alpha-beta adalah strategi yang paling efektif dalam permainan ini. Bot ini menggunakan algoritma Minimax dengan pemotongan alpha-beta untuk memilih langkah terbaik, mempertimbangkan pemain dan lawan, serta mencapai hasil yang optimal dalam waktu yang relatif singkat. Kinerja bot ini didasarkan pada evaluasi papan saat ini dan memiliki pemahaman yang lebih baik tentang potensi pergerakan pemain dan lawan.

Bot local search simulated annealing, meskipun menawarkan pendekatan yang berbeda dengan pendekatan pencarian lokal dan proses pengoptimalan temperatur (T) terhadap langkah-langkah yang mungkin, tidak dapat bersaing dengan keunggulan algoritma Minimax dalam permainan ini. Bot ini mungkin lebih cocok untuk permainan yang kurang kompleks atau dengan aturan yang berbeda. Bot genetik, meskipun menarik dalam penggunaan teknik evolusi untuk menghasilkan solusi yang kuat, mungkin memerlukan penyesuaian yang lebih baik terhadap permainan Adjacency Strategy Game untuk meningkatkan kinerjanya.

4.2 Saran

Untuk membuat bot AI yang cerdas dan efisien untuk permainan adversarial adjacency strategy game berukuran 8x8, Minimax dengan pemotongan alpha-beta adalah pilihan terbaik. Ini akan memberikan keseimbangan antara kualitas langkah-langkah yang dihasilkan dan waktu komputasi yang wajar. Selain itu, bot genetik dan bot local search simulated annealing dapat ditingkatkan dengan lebih mendalam memahami permainan dan menerapkan strategi yang lebih canggih yang sesuai dengan metode pencarian dan optimasi yang digunakan oleh bot tersebut.

Lampiran

Repository Github: https://github.com/bernarduswillson/Tubes1_13521021.git

Kontribusi setiap anggota kelompok

NIM	Nama	Kontribusi
13521021	Bernardus Willson	Engine, Minimax, Laporan
13521022	Raditya Naufal Abiyu	Genetic, Laporan
13521023	Kenny Benaya Nathan	Minimax, Laporan, Fallback Plan
13521026	Kartini Copa	Local, Laporan