

***LOCAL SEARCH DAN ADVERSARIAL SEARCH DALAM
PEMBUATAN INTELLIGENT AGENT UNTUK
MEMENANGKAN ADJACENCY STRATEGY GAME***

**Disusun untuk memenuhi laporan tugas besar 1 mata kuliah IF3170
Inteligensi Buatan semester 5 di Institut Teknologi Bandung.**



Disusun oleh:

Athif Nirwasito	13521053
Muhammad Fadhil Amri	13521066
Ryan Samuel Chandra	13521140
Haziq Abiyyu Mahdy	13521170

**TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

**Jl. Ganesa No. 10, Lb. Siliwangi, Kecamatan Coblong,
Kota Bandung, Jawa Barat, 40132**

2023

Daftar Isi

Daftar Isi	2
1. Rencana Objective Function	3
2. Minimax dan Alpha-Beta Pruning	3
3. Local Search: Hill-Climbing Steepest Ascent	3
3.1 Definisi	3
3.2 Algoritma	4
3.3 Justifikasi Pemilihan Algoritma	5
4. Genetic Algorithm	5
5. Hasil Pertandingan	5
5.1 Bot Minimax vs Manusia	5
5.2 Bot Local Search vs Manusia	5
5.3 Bot Minimax vs Bot Local Search	5
5.4 Bot Minimax vs Bot Genetic Algorithm	5
5.5 Bot Local Search vs Bot Genetic Algorithm	5

1. Rencana *Objective Function*

$$\text{Maksimasi } h(\text{state}) = \text{BotScore} - \text{OpponentScore}$$

Nilai maksimum yang diperoleh dari BotScore - OpponentScore menunjukkan pendekatan yang dilakukan adalah menargetkan kemenangan, yaitu saat BotScore > OpponentScore. Melalui *objective function* ini, bot akan selalu mencari *movement* yang dapat membuat skornya lebih besar dari pemain lawan, atau jika skor bot < skor pemain lawan, perbedaannya menjadi minimum. Kedua hal tersebut dapat memperbesar kemungkinan bot untuk dapat mengejar skor pada giliran berikutnya.

Selain itu, hal utama yang menyebabkan *objective function* ini dipilih adalah *state* yang dapat berubah akibat pemain lain, bukan hanya akibat bot. Oleh karena itu, untuk menjamin pergerakan yang dipilih selalu mendekati kemenangan, *objective function* dirasa perlu menggunakan selisih skor terbesar, agar menghasilkan solusi terbaik.

2. *Minimax dan Alpha-Beta Pruning*

Minimax adalah algoritma yang dapat digunakan pada persoalan *Adversarial search*. *Adversarial search* adalah persoalan di mana terdapat dua atau lebih *agent* yang saling berkompetisi (misalnya dalam suatu permainan). Pada persoalan ini, aksi dilakukan oleh *agent* secara bergantian dengan *agent* lain atau pemain lawan. *Agent* bertujuan memenangkan permainan serta mencegah *agent* lain atau pemain lawan itu memenangkan permainan. Pada *adjacency strategy game*, didefinisikan beberapa unsur-unsur algoritma *minimax* sebagai berikut.

- *State* berupa matriks yang berukuran 8×8 , di mana $\text{state}[i][j] = ' '$ jika kotak pada posisi $[i, j]$ kosong, $\text{state}[i][j] = 'O'$ jika kotak berisi bidak yang dimiliki oleh *bot/agent* (pada kasus ini, bidak yang dimiliki *bot* adalah bidak 'O'), dan $\text{state}[i][j] = 'X'$ jika kotak berisi bidak yang dimiliki lawan (pada kasus ini, bidak yang dimiliki lawan adalah bidak 'X').
- *Terminal function*, yaitu sebuah fungsi yang menerima sebuah *state* dan menentukan apakah permainan sudah berakhir atau tidak. Hal ini dapat dilakukan dengan mengecek

apakah masih ada blok kosong yang tersedia pada papan permainan atau bot sudah mencapai batas pencariannya.

- *Utility function*, yaitu sebuah fungsi yang digunakan untuk menilai nilai numerik dari suatu *state* (*state value*). Nilai ini dihitung saat permainan mencapai *terminal state* untuk menentukan pemain mana yang memenangkan permainan berdasarkan *state* tersebut. Pada *adjacency strategy game*, *state value* dihitung dengan cara menjumlahkan seluruh angka pada elemen-elemen *state*. Jika *state value* > 0 , maka pemain yang dijalankan oleh *bot* memenangkan permainan. Jika *state value* < 0 , maka pemain lawan memenangkan permainan. Jika *state value* $= 0$, maka permainan berakhir seri. Karena *bot* bertujuan memaksimalkan *state value* dan lawan bertujuan meminimalkan *state value*, maka pada konteks ini, *bot* disebut sebagai *maximizing player* dan lawan disebut sebagai *minimizing player*.
- *Action*, yaitu sebuah struktur data yang menyimpan aksi yang dilakukan oleh bot. Pada permainan ini, data yang disimpan adalah koordinat peletakan bidak serta pemain yang meletakkan bidak.
- *Result function*, yaitu sebuah fungsi yang memetakan suatu *state* dan suatu *action* terhadap *state* yang dihasilkan dari *action* tersebut.
- *Actions function*, yaitu sebuah fungsi yang menerima masukan berupa *state* dan pemain yang sedang dalam gilirannya dan mengembalikan seluruh *action* yang mungkin dilakukan oleh pemain tersebut.

Sebelum membahas algoritma *minimax*, perlu juga didefinisikan komponen-komponen *game tree* dari *Adjacency Strategy Game*. Pada pohon gim ini, simpul merepresentasikan isi papan pada waktu tertentu dan *edge* merepresentasikan langkah yang diambil dari satu *state* ke *state* lainnya. Pada gim ini, kasus terminal tercapai jika ronde permainan sudah mencapai batas ronde yang ditentukan di awal permainan. Pada kasus terminal, akan dievaluasi kondisi papan pada simpul tersebut menggunakan *objective function* yang telah didefinisikan.

Pada dasarnya, pemanggilan fungsi *minimax* dilakukan secara **rekursif** dan **berselang-seling** antara *maximizing player* dan *minimizing player* hingga menemukan *terminal state*. Dengan kata lain, program **mengasumsikan lawan selalu mengambil langkah optimal**. Hal ini bertujuan

agar program dapat menemukan keputusan terbaik meskipun berada dalam skenario terburuk, yaitu lawan bermain dengan optimal.

Algoritma *minimax* bersifat *complete* dan optimal. Namun, algoritma ini membutuhkan kompleksitas waktu dan ruang yang tinggi apabila terdapat banyak pilihan aksi yang tersedia atau kedalaman pohon pencarian sangat besar. Oleh karena itu, diperlukan metode untuk mengurangi jumlah langkah yang diperlukan untuk menemukan keputusan terbaik. Metode yang digunakan adalah *alpha beta pruning*. Pada dasarnya, metode ini menambahkan dua parameter tambahan pada fungsi *minimax*, yaitu *alpha* dan *beta*.

- *Alpha* adalah *state value* terbesar yang ditemukan sejauh ini untuk *maximizing player*. Pada awalnya *alpha* bernilai $-\infty$, kemudian nilai akan diubah sering pencarian nilai terbesar *state value* oleh *maximizing player*.
- *Beta* adalah *state value* terkecil yang ditemukan sejauh ini untuk *minimizing player*. Pada awalnya *alpha* bernilai ∞ , kemudian nilai akan diubah sering pencarian nilai terkecil *state value* oleh *minimizing player*.

Terdapat kondisi di mana pencarian dapat dihentikan, yaitu jika ***alpha* \geq *beta***. Misalkan *maximizing player* pada *state* *s* ingin mendapatkan *state value* yang lebih besar dari *alpha*. Namun, pada *state* *s'* (anak dari *state* *s*), *minimizing player* ingin mendapatkan *state value* yang lebih kecil dari *beta*. Oleh karena itu, *maximizing player* pada *state* *s* dapat mengetahui bahwa tujuannya mendapatkan *state value* yang lebih besar dari *alpha* tidak akan tercapai, karena pilihan-pilihan selanjutnya akan lebih kecil dari *beta* dan *beta* \leq *alpha*. Oleh karena itu, pencarian tidak perlu dilanjutkan.

Secara umum, proses *minimax* dengan *alpha-beta pruning* pada *Adjacency Strategy Game* dapat diurai sebagai berikut:

1. Basis: Jika kasus terminal maka kembalikan nilai hasil evaluasi *state* dan koordinat gerakan terakhir.

Rekurens

2. Pada fase *maximizing*, inisiasi nilai ekstrem dengan negatif tak hingga, sedangkan jika pada fase *minimizing* inisiasi nilai ekstrim dengan tak hingga.

3. Lakukan iterasi pada setiap *state* yang langkah yang memungkinkan pada papan.
4. Setiap iterasi dimulai dengan mengisi papan agar sesuai dengan langkah yang diiterasi, kemudian lakukan rekursi pada kondisi papan yang baru. Jika simpul sekarang adalah *maximizer*, maka rekursi yang dipanggil akan *minimize*, begitu juga sebaliknya.
5. Setelah rekursi selesai, perbarui nilai ekstrem, dan nilai *alpha* jika *maximizer* atau nilai *beta* jika *minimizer*. Nilai ekstrem diperbarui dengan mengambil nilai maksimum pada *maximizer* atau nilai minimum pada *minimizer*. Perbarui langkah juga jika didapatkan nilai ekstrem baru.
6. Jika ditemukan *alpha* lebih besar dari *beta*, maka pangkas simpul tersebut.
7. Ulangi dari langkah 4, hingga semua langkah yang memungkinkan telah diiterasi.
8. Kembalikan nilai ekstrem yang didapatkan serta langkah untuk mendapatkan nilai tersebut ke simpul sebelumnya.

3. Local Search: Hill-Climbing Steepest Ascent

3.1 Definisi

Berikut adalah pendefinisian yang perlu dipahami terlebih dahulu:

1. *State*: Kondisi papan permainan saat ini.
2. *Initial state*: Jika giliran pertama adalah pemain 'X', *initial state* adalah *state* setelah pemain 'X' bergerak. Akan tetapi, jika giliran pertama adalah pemain 'O' (bot), *initial state* adalah *state* di awal permainan.
3. *Successor* : Sebuah *state* yang dapat dicapai melalui 1 (satu) *legal move* dari *current state*.

						o	o
						o	o
x	x						
x	x						

Gambar 1.1 *State Awal Permainan*

4. *Neighbor* : *Successor* yang dipilih sebagai *current state*.
5. *Objective function* : Maksimasi $h(\text{state}) = \text{BotScore} - \text{OpponentScore}$
6. *Termination Condition*: Ronde sudah mencapai *limit* yang ditetapkan, atau *state* sudah tidak dapat berubah (karena papan permainan penuh).

3.2 Algoritma

Setelah memahami beberapa definisi di atas, berikut adalah proses pencarian kotak optimal untuk ditempati berdasarkan algoritma *local search* yang digunakan:

1. Tentukan *initial state* sesuai kondisi yang sudah didefinisikan di atas.
2. Cari *neighbor*, yaitu *successor* yang memiliki nilai maksimal dari hasil evaluasi *objective function*. Apabila ada beberapa *successor* memiliki nilai maksimal yang sama, pilih secara *random* dari *successor-successor* tersebut.
3. Ganti *current state* menjadi *neighbor state* (pada kasus ini *neighbor* dapat dijamin selalu memiliki nilai yang lebih besar dari *current state* karena tipe *game*-nya hanya memungkinkan terjadi penambahan *score* dari pemain yang melakukan *move*).
4. Lakukan pembaharuan *current state* setelah pemain lawan melakukan *move*.
5. Ulangi dari langkah 3 sampai 5 hingga tercapai *termination state*.

3.3 Justifikasi Pemilihan Algoritma

Seperti yang sudah dituliskan sebelumnya, dalam *adjacency strategy game* ini, semua *successor* (dan tentunya *neighbor*) dijamin punya *state value* yang lebih tinggi dari *current state value*. Maka dari itu, diperlukan langkah optimum di setiap giliran yang dilakukan agar membawa bot menuju kemenangan. Algoritma *local search* yang paling sesuai untuk memecahkan persoalan ini adalah *Steepest Ascent Hill-Climbing*.

- *Hill-Climbing with Sideways Move* tidak dipilih karena tidak akan ada kasus *successor value = current value*.
- *Stochastic Hill-Climbing* tidak dipilih, karena dengan pemilihan *successor* yang dilakukan secara *random*, dengan jumlah ronde terbatas, dan di bawah tekanan pemain lawan, pastinya tidak akan membawa bot menuju kemenangan.
- Demikian pula *Random-Restart Hill-Climbing* tidak cocok diterapkan di sini karena jumlah ronde terbatas, serta tidak ada pengulangan pertandingan.
- *Simulated Annealing* jelas tidak mungkin dipilih karena sifatnya yang “*allow bad moves but gradually decreasing the frequency*”. Mengingat kembali, di permainan ini, tidak ada *successor* yang memiliki *state value* lebih rendah dari *current state*, sehingga tidak mungkin ada “*bad moves*”.

4. Genetic Algorithm

Genetic algorithm adalah algoritma pencarian yang terinspirasi dari proses seleksi alam. Pada proses seleksi alam, terdapat konsep *survival of the fittest*, yaitu individu yang bertahan hidup adalah individu yang terbaik (paling mampu menyesuaikan diri dengan lingkungan). Pada suatu populasi, individu-individu yang dapat bertahan hidup akan saling bereproduksi dan mewariskan sifat-sifatnya kepada keturunannya. Terjadi persilangan sifat (*crossover*) pada kromosom dua individu yang saling bereproduksi sehingga menghasilkan keturunan yang memiliki gabungan dari sifat kedua orang tuanya. Individu yang dihasilkan idealnya akan memiliki *fitness* yang lebih baik dari orang tuanya sehingga dapat mempertahankan keberlangsungan hidup spesies individu tersebut, meski hal ini tidak selalu terjadi.

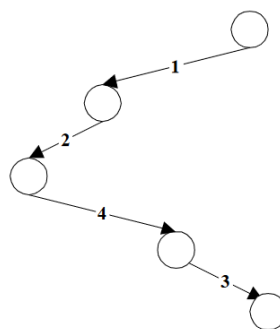
Pada *adjacency strategy game*, didefinisikan beberapa unsur-unsur *genetic algorithm* sebagai berikut.

- Representasi kromosom

Pada *adjacency strategy game*, kromosom dapat merepresentasikan urutan posisi yang diisi oleh pemain. Tiap-tiap blok pada papan permainan dapat ditandai dengan bilangan untuk menandai posisi blok tersebut seperti gambar berikut.

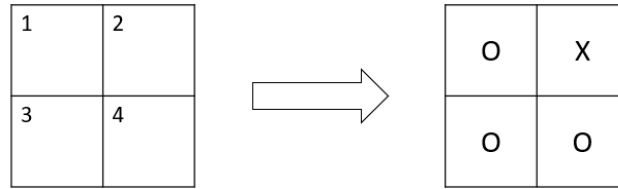
1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

Panjang kromosom sama dengan banyaknya blok yang dapat diisi, yaitu 64. Misalkan untaian kromosom adalah 1 - 4 - 21 - 64 - dst. Hal ini berarti pemain pertama mengisi blok 1, kemudian pemain kedua mengisi blok 4, kemudian pemain pertama mengisi blok 21, kemudian pemain kedua mengisi blok 64, dan seterusnya.



Pada contoh *adjacency strategy game* dengan ukuran papan 4×4 , kromosom 1-2-4-3 dapat direpresentasikan sebagai salah satu *path* pada pohon permainan seperti pada gambar di atas. Misalkan pemain pertama adalah pemain 'X' dan pemain kedua adalah

pemain ‘O’, maka *terminal state* yang dihasilkan dari kromosom 1-2-4-3 adalah sebagai berikut.



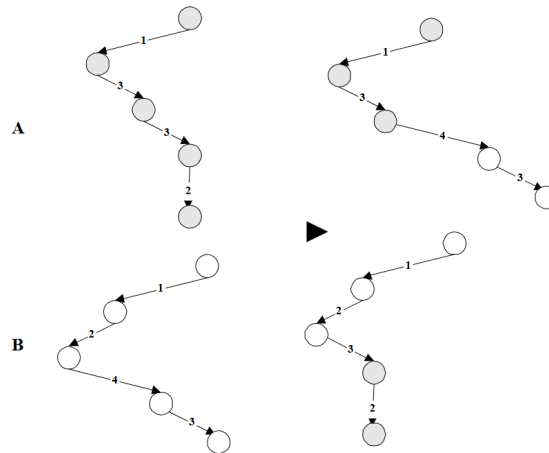
- **Populasi**

Beberapa kromosom yang tersedia disebut juga dengan populasi. Sebelum pencarian dimulai, populasi dibangkitkan secara acak. Kemudian, individu-individu pada populasi dapat bereproduksi menghasilkan keturunan dan dapat menggantikan individu lainnya.

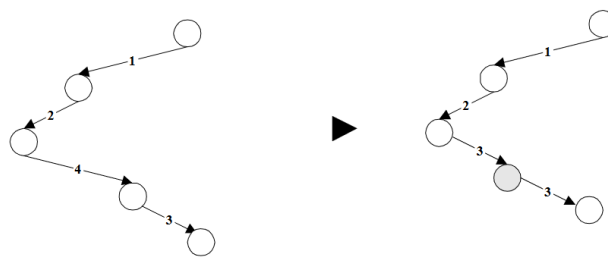
- **Crossover dan mutasi**

Pada dua individu yang saling bereproduksi, akan terjadi *crossover* untuk menghasilkan kromosom keturunannya. Pada kromosom kedua individu yang bereproduksi, dipilih suatu titik yang disebut dengan *crossover point*. Titik ini akan menentukan posisi di mana kromosom orang tua saling dipertukarkan.

Perhatikan ilustrasi berikut. Misal terdapat individu A dengan kromosom 1-3-3-2 (**Catatan:** kromosom ini bukan kromosom yang valid pada kasus *adjacency strategy game* karena blok yang sama tidak dapat diisi dua kali. Kromosom ini hanya digunakan sebagai contoh semata) dan B dengan kromosom 1-2-4-3. Jika *crossover point* terletak setelah gen kedua pada kromosom, maka kromosom individu yang dihasilkan dari proses reproduksi A dan B adalah 1-3-4-3 dan 1-2-3-2.

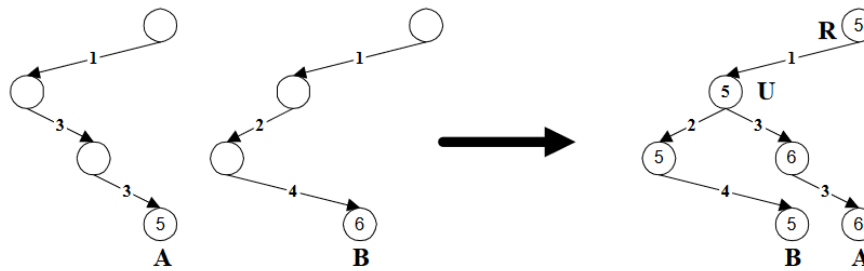


Terdapat pula kemungkinan terjadinya mutasi pada suatu individu yang menyebabkan terjadinya perubahan pada satu atau lebih gen pada kromosom individu tersebut. Contohnya adalah kromosom 1-2-4-3 berubah menjadi 1-2-3-3



- *Fitness evaluation*

Pada *genetic algorithm*, diperlukan pula suatu fungsi untuk mengukur *fitness* dari suatu kromosom. Untuk mengukur nilai *fitness* dari suatu kromosom, harus terdapat beberapa kromosom lainnya untuk melakukan perbandingan. Dari beberapa kromosom tersebut, dibentuk suatu *reservation tree*, yaitu gabungan dari pohon beberapa kromosom tersebut. Kemudian, dilakukan pencarian *state value* dari simpul akar menggunakan algoritma *minimax*, dengan asumsi aksi pertama dilakukan oleh *maximizing player*, kemudian dilanjutkan oleh *minimizing player*, dan begitu seterusnya. Apabila sudah mencapai *terminal state*, perhitungan *state value* dapat dilakukan dengan menggunakan *utility function* pada algoritma *minimax*.



Untuk menentukan *fitness* dari suatu kromosom, kita dapat melihat sejauh mana simpul mempertahankan *state value* yang sama dihitung dari simpul daun hingga atas. Pada gambar di atas, terdapat dua kromosom, A (1-3-3) dan B (1-2-4) dan dihasilkan *reservation tree* yang memiliki *least common ancestor* pada simpul U di level ke-1 pohon. Setelah menerapkan algoritma *minimax* dimulai dari simpul akar, didapat *state value* dari setiap simpul. Kemudian, kita dapat menghitung *fitness* dari kromosom A dan B sebagai berikut.

- Kromosom A mempertahankan *state value* 6 sebanyak dua kali, sehingga *fitness value* dari kromosom A adalah 2
- Kromosom B mempertahankan *state value* 5 sebanyak empat kali (hingga simpul akar), sehingga *fitness value* dari kromosom A adalah 4

Sehingga, dapat disimpulkan kromosom B memiliki *fitness* yang lebih besar dari kromosom A.

Berikut adalah langkah-langkah pemilihan aksi pada *adjacency strategy game* menggunakan *genetic algorithm*.

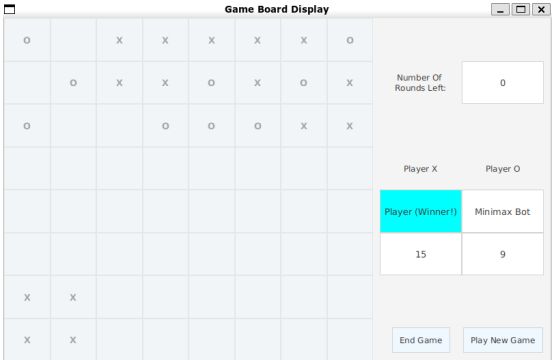
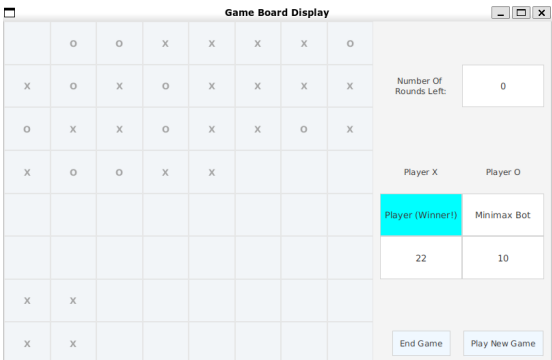
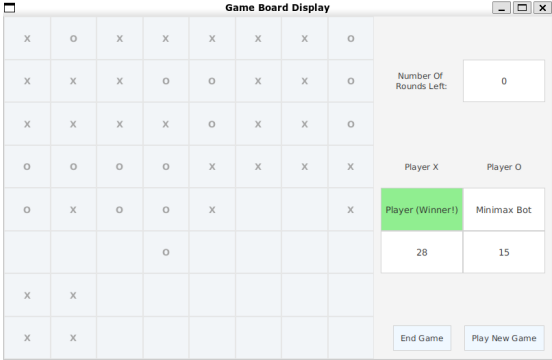
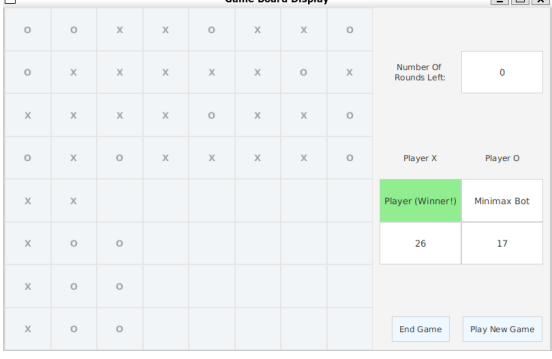
1. Tentukan banyaknya pembangkitan kromosom maksimum yang dapat dilakukan serta batas kedalaman pohon pencarian
2. Pilih N kromosom secara acak, lalu masukkan ke dalam *reservation tree*
3. Hitung *state value* dari setiap simpul pada *reservation tree* menggunakan algoritma *minimax*

4. Lakukan *crossover* dan mutasi dari kromosom-kromosom yang ada. Hitung banyaknya pembangkitan kromosom baru dari proses *crossover* dan mutasi
5. Masukkan kromosom-kromosom baru hasil *crossover* dan mutasi ke dalam *reservation tree*
6. Hitung *state value* dari setiap simpul pada *reservation tree* menggunakan algoritma *minimax*
7. Pilih N kromosom dengan *fitness value* terbaik dari seluruh kromosom yang ada pada *reservation tree*
8. Lakukan langkah 4-7 hingga banyaknya pembangkitan kromosom yang telah dilakukan sama dengan banyaknya pembangkitan kromosom maksimum yang telah didefinisikan pada langkah 1
9. Pilih aksi yang memaksimalkan *state value* menggunakan algoritma *minimax* terhadap *reservation tree* yang telah dibentuk.

5. Hasil Pertandingan

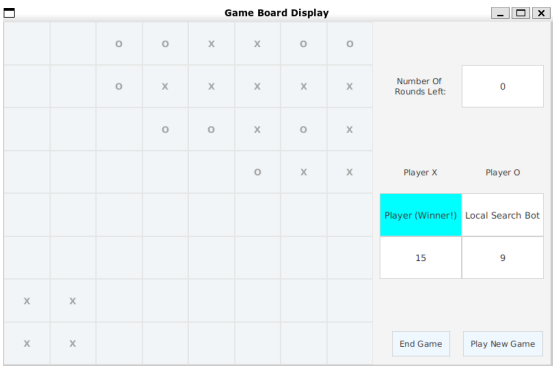
5.1 Bot *Minimax* (*Depth* = 7) vs Manusia

Pertandingan ke-	Jumlah Ronde	Pemenang	Penjelasan Singkat
1	28	Manusia	

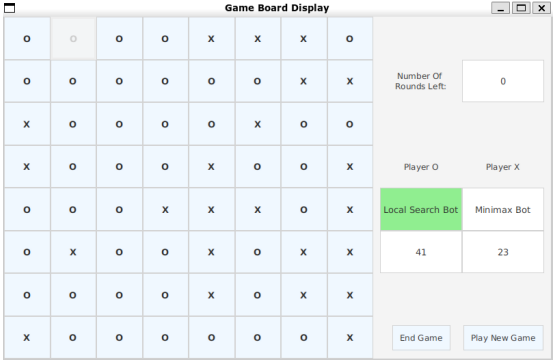
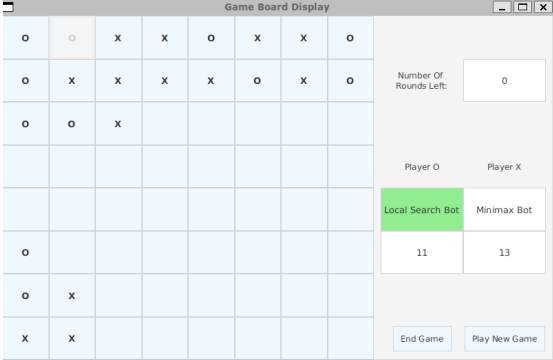
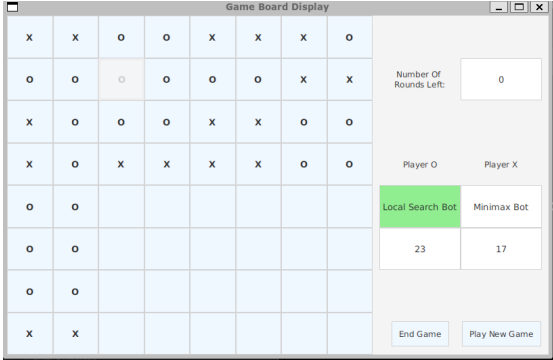
2	8	Manusia	 <p>Game Board Display</p> <p>Number Of Rounds Left: 0</p> <p>Player X: 15, Player O: 9</p> <p>Player (Winner!)</p> <p>End Game Play New Game</p>
3	12	Manusia	 <p>Game Board Display</p> <p>Number Of Rounds Left: 0</p> <p>Player X: 22, Player O: 10</p> <p>Player (Winner!)</p> <p>End Game Play New Game</p>
4	17	Manusia	<p>Bot goes first.</p>  <p>Game Board Display</p> <p>Number Of Rounds Left: 0</p> <p>Player X: 28, Player O: 15</p> <p>Player (Winner!)</p> <p>End Game Play New Game</p>
5	23	Manusia	<p>Bot goes first.</p>  <p>Game Board Display</p> <p>Number Of Rounds Left: 0</p> <p>Player X: 26, Player O: 17</p> <p>Player (Winner!)</p> <p>End Game Play New Game</p>

% Kemenangan Bot			$0/5 * 100\% = 0\%$

5.2 Bot *Local Search* vs Manusia

Pertandingan ke-	Jumlah Ronde	Pemenang	Penjelasan Singkat
1	28	Bot	
2	8	Manusia	
3	16	Manusia	
% Kemenangan Bot			$\frac{1}{3} * 100\% = 33,33\%$

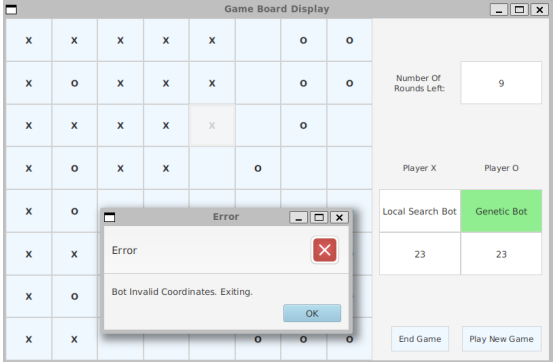
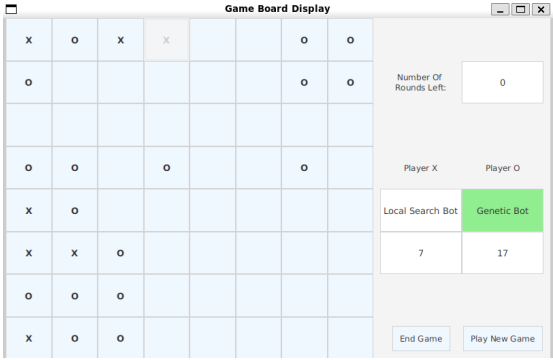
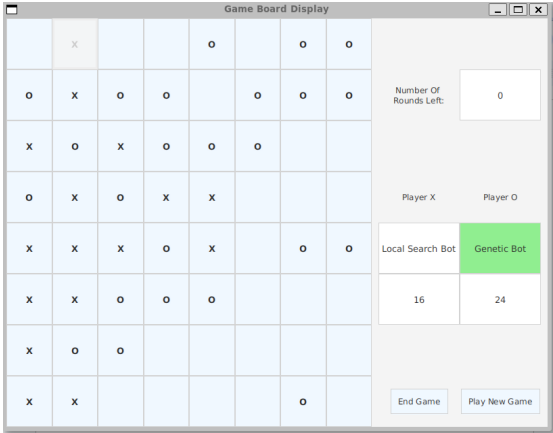
5.3 Bot *Minimax* (Depth = 7) vs Bot *Local Search*

Pertandingan ke-	Jumlah Ronde	Pemenang	Penjelasan Singkat
1	28	<i>Local Search Bot</i>	
2	8	<i>Minimax Bot</i>	
3	16	<i>Local Search Bot</i>	
% Kemenangan Local Search Bot			$\frac{2}{3} * 100\% = 66,66\%$
% Kemenangan Minimax Bot			$\frac{1}{3} * 100\% = 33,33\%$

5.4 Bot Minimax (Depth = 7) vs Bot Genetic Algorithm

Pertandingan ke-	Jumlah Ronde	Pemenang	Penjelasan Singkat
1	28	Genetic Algorithm Bot	
2	8	Genetic Algorithm Bot	
3	16	Genetic Algorithm Bot	
% Kemenangan Genetic Bot			$3/3 * 100\% = 100\%$
% Kemenangan Minimax Bot			$0/3 * 100\% = 0\%$

5.5 Bot Local Search vs Bot Genetic Algorithm

Pertandingan ke-	Jumlah Ronde	Pemenang	Penjelasan Singkat
1	28	-	 <p>Terjadi pergerakan ilegal oleh <i>Genetic Algorithm Bot</i>.</p>
2	8	<i>Genetic Bot</i>	
3	16	<i>Genetic Bot</i>	
% Kemenangan Genetic Bot			$2/2 * 100\% = 100\%$
% Kemenangan Local Search Bot			$0/2 * 100\% = 0\%$

6. Kontribusi Anggota

NIM	NAMA	KONTRIBUSI
13521053	Athif Nirwasito	Bot <i>Minimax</i> , Laporan
13521066	Muhammad Fadhil Amri	Bot <i>Local Search</i> , Laporan
13521140	Ryan Samuel Chandra	GameState, Laporan
13521170	Haziq Abiyyu Mahdy	Bot <i>Genetic Algorithm</i> , Laporan

LAMPIRAN

Pranala Github (*repository*):

<https://github.com/Onyxcodeotto/adversarial-adjacency-strategy-game.git>