

TUGAS BESAR 1 IF3170 INTELIGENSI BUATAN

Minimax Algorithm and Alpha Beta Pruning in Adjacency Strategy Game



Disusun oleh:

13521059 Arleen Chrysanthia Gunardi

13521107 Jericho Russel Sebastian

13521125 Asyifa Nurul Shafira

13521133 Cetta Reswara Parahita

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG**

2023

Daftar Isi

Daftar Isi.....	i
Daftar Gambar.....	ii
1. Objective Function.....	1
2. Proses Pencarian dengan Minimax dan Alpha Beta Pruning.....	1
3. Proses Pencarian dengan Local Beam Search.....	4
4. Proses Pencarian dengan Genetic Algorithm.....	7
5. Hasil Pertandingan.....	10
5.1 Bot Minimax vs Manusia.....	10
5.1.1 Pertandingan 1.....	10
5.1.2 Pertandingan 2.....	10
5.1.3 Pertandingan 3.....	11
5.1.4 Pertandingan 4.....	12
5.1.5 Pertandingan 5.....	12
5.2 Bot Local Search vs Manusia.....	13
5.2.1 Pertandingan 1.....	13
5.2.2 Pertandingan 2.....	14
5.2.3 Pertandingan 3.....	14
5.3 Bot Minimax vs Bot Local Search.....	15
5.3.1 Pertandingan 1.....	15
5.3.2 Pertandingan 2.....	16
5.3.3 Pertandingan 3.....	16
5.4 Bot Minimax vs Bot Genetic Algorithm.....	17
5.4.1 Pertandingan 1.....	17
5.4.2 Pertandingan 2.....	18
5.4.3 Pertandingan 3.....	18
5.5 Bot Local Search vs Genetic Algorithm.....	20
5.5.1 Pertandingan 1.....	20
5.5.2 Pertandingan 2.....	20
5.5.3 Pertandingan 3.....	21
5.5.4 Pertandingan 4.....	22
5.5.5 Pertandingan 5.....	22
6. Referensi.....	23
7. Pembagian Pekerjaan.....	24
8. Lampiran.....	24

Daftar Gambar

Gambar 3.1 Contoh Local Beam Search Sederhana.....	6
Gambar 5.1.1.1 Hasil Pertandingan 1 Bot Minimax vs Manusia.....	10
Gambar 5.1.2.1 Hasil Pertandingan 2 Bot Minimax vs Manusia.....	10
Gambar 5.1.3.1 Hasil Pertandingan 3 Bot Minimax vs Manusia.....	11
Gambar 5.1.4.1 Hasil Pertandingan 4 Bot Minimax vs Manusia.....	12
Gambar 5.1.5.1 Hasil Pertandingan 5 Bot Minimax vs Manusia.....	12
Gambar 5.2.1.1 Hasil Pertandingan 1 Bot Local Search vs Manusia.....	13
Gambar 5.2.2.1 Hasil Pertandingan 2 Bot Local Search vs Manusia.....	14
Gambar 5.2.3.1 Hasil Pertandingan 3 Bot Local Search vs Manusia.....	14
Gambar 5.3.1.1 Hasil Pertandingan 1 Bot Minimax vs Bot Local Search.....	15
Gambar 5.3.2.1 Hasil Pertandingan 2 Bot Minimax vs Bot Local Search.....	16
Gambar 5.3.3.1 Hasil Pertandingan 3 Bot Minimax vs Bot Local Search.....	16
Gambar 5.4.1.1 Hasil Pertandingan 1 Bot Minimax vs Bot Genetic Algorithm.....	17
Gambar 5.4.2.1 Hasil Pertandingan 2 Bot Minimax vs Bot Genetic Algorithm.....	18
Gambar 5.4.3.1 Hasil Pertandingan 3 Bot Minimax vs Bot Genetic Algorithm.....	18
Gambar 5.5.1.1 Hasil Pertandingan 1 Bot Local Search vs Bot Genetic Algorithm.....	20
Gambar 5.5.2.1 Hasil Pertandingan 2 Bot Local Search vs Bot Genetic Algorithm.....	20
Gambar 5.5.3.1 Hasil Pertandingan 3 Bot Local Search vs Bot Genetic Algorithm.....	21
Gambar 5.5.4.1 Hasil Pertandingan 4 Bot Local Search vs Bot Genetic Algorithm.....	22
Gambar 5.5.5.1 Hasil Pertandingan 5 Bot Local Search vs Bot Genetic Algorithm.....	22

1. **Objective Function**

Objective function (f) yang digunakan pada strategi permainan ini menggambarkan seberapa menguntungkan suatu *state* bagi sebuah bot. Adapun *objective function* tersebut dapat didefinisikan sebagai $f(s) = O - X$, di mana O adalah banyak markah O di atas papan, dan X adalah banyak markah X di atas papan.

2. **Proses Pencarian dengan Minimax dan Alpha Beta Pruning**

Ide utama pencarian Minimax adalah memaksimalkan utilitas minimum yang dapat diraih secara paksa, yaitu utilitas yang diraih ketika menghadapi lawan yang bermain dengan sempurna. Dalam konteks permainan Adjacency Strategy Game, ini berarti memaksimalkan kemungkinan untuk menang dengan markah sebanyak mungkin, meskipun lawan selalu mengambil langkah yang terbaik. Hal ini dicapai dengan melakukan pergiliran antara pencarian langkah bot dan langkah lawan, sedemikian hingga pencarian untuk langkah bot selalu mengambil langkah dengan utilitas maksimum (MAX; langkah terbaik yang memaksimalkan peluang kemenangan bot), dan pencarian untuk langkah lawan selalu mengambil langkah dengan utilitas minimum (MIN; langkah terbaik bagi lawan dengan meminimalkan peluang kemenangan bot).

Definisi yang digunakan untuk pencarian Minimax dalam permainan ini adalah sebagai berikut:

1. *State*: Keadaan dari permainan pada suatu waktu, mencakup keadaan papan permainan dan ronde.
2. Aksi: Menempatkan markah pada suatu kotak kosong tertentu.
3. *Terminal test*: Sebuah *state* adalah *terminal state* jika, pada *state* tersebut, permainan sudah berakhir atau tidak ada kotak kosong yang tersisa pada papan permainan. Permainan berakhir ketika sudah mencapai ronde terakhir.
4. Fungsi utilitas: Nilai utilitas dari sebuah *state* menggambarkan seberapa baik *state* tersebut bagi bot. Definisi yang dipilih untuk fungsi utilitas ini adalah $U(s) = O - X$, di mana O adalah banyak markah O di atas papan, dan X adalah banyak markah X di atas papan.

Permainan Adjacency Strategy Game dengan 10 ronde menghasilkan pohon pencarian dengan 20 tingkat dan faktor percabangan rata-rata sebesar 37. Karena itu, teknik *alpha-beta*

pruning digunakan dalam proses pencarian untuk mengurangi banyak *state* yang harus diperiksa dan mempercepat proses pencarian.

Pada setiap awal giliran bot, pencarian dilakukan dengan algoritma rekursif berikut, dengan *state* awal mencerminkan keadaan awal permainan, dan tingkat pertama pohon pencarian merupakan tingkat MAX:

1. Jika *state* ini merupakan *terminal state*, hitung utilitas *state* ini menggunakan fungsi utilitas di atas.
2. Jika tidak, maka untuk setiap kotak kosong yang ada pada papan permainan pada *state* ini, bangkitkan sebuah *state* anak yang mencerminkan keadaan papan jika kotak tersebut diisi sebuah markah O (untuk giliran bot) atau X (untuk giliran pemain). Ulangi algoritma pada *state* tersebut.
3. Jika, saat iterasi di tahap sebelumnya, ditemukan sebuah *state* yang bernilai sangat signifikan untuk sisi lawan dari pemegang giliran sehingga, apapun langkah lanjutan yang diambil oleh sisi pemegang giliran, utilitasnya tidak akan lebih baik dari langkah lanjutan dari *state* lainnya yang sudah pernah diperiksa sebelumnya (dan dengan demikian tidak akan mempengaruhi hasil pencarian, apapun nilai utilitas dari *state* selanjutnya yang akan diperiksa), maka seluruh cabang dari *state* tersebut dapat dibuang untuk menghemat waktu pencarian.
4. Setelah semua aksi yang mungkin sudah diperiksa (atau dibuang), ambil nilai utilitas yang optimal, sesuai dengan tingkat dari *state* ini; jika *state* ini terletak pada tingkat MAX, maka ambil nilai tertinggi dari utilitas *state-state* anaknya, sedangkan jika *state* ini terletak pada tingkat MIN, ambil nilai terendah.

Hasil pencarian adalah satu atau lebih lintasan dari *state* awal menuju salah satu *terminal state*, di mana semua *state* di sepanjang lintasan ini memiliki nilai utilitas yang sama. Lintasan ini mewakili urutan kotak kosong yang akan dibubuhi markah jika bot dan pemain selalu bermain secara optimal. Urutan ini mengoptimalkan peluang kemenangan bagi bot dalam permainan.

Strategi Minimax cocok untuk diimplementasikan pada permainan dengan dua pemain, seperti Adjacency Strategy Game ini, karena strategi Minimax mempertimbangkan aksi yang mungkin dilakukan lawannya, dengan asumsi bahwa lawan selalu melakukan aksi yang menghasilkan kondisi terbaik baginya (kondisi terburuk bagi bot).

Strategi Minimax dengan *alpha-beta pruning* bisa jadi kurang efisien apabila *state* yang dibangkitkan tidak terurut dengan baik. Dengan kata lain, urutan *state* yang dibangkitkan tidak mengoptimalkan banyaknya cabang yang dapat di-*prune*. Oleh karena itu, *state-state* yang telah dibangkitkan dapat diurutkan sebelum melakukan pemilihan dengan menggunakan sebuah fungsi heuristik. Sehingga, *state* dengan nilai heuristik tertinggi akan diperiksa terlebih dulu. Adapun fungsi heuristik tersebut adalah $h(s)$, di mana nilai $h(s)$ dihitung menggunakan aturan berikut:

1. Nilai awal $h(s)$ adalah 0.
2. Utamakan langkah yang mengubah sebanyak mungkin markah lawan: untuk setiap kotak yang bertetanggaan secara horizontal atau vertikal dengan kotak s yang diisi markah lawan, tambahkan 1 ke nilai $h(s)$.
3. Utamakan langkah yang menghasilkan sesedikit mungkin kelompok markah yang dapat diubah secara sekaligus oleh lawan: untuk setiap kotak yang bertetanggaan secara diagonal dengan kotak s yang diisi markah sendiri, jika terdapat kotak kosong yang bertetanggaan secara horizontal atau vertikal dengan s dan kotak tersebut, kurangi 1 dari nilai $h(s)$.

Kekurangan terbesar dari pencarian menggunakan strategi ini adalah ukuran pohon pencarian yang sangat besar. Tanpa *alpha-beta pruning*, pohon pencarian permainan Adjacency Strategy Game untuk permainan dengan hanya 2 ronde terdiri atas lebih dari 8 juta *state* yang harus diperiksa. Jika pembatasan waktu diterapkan terhadap pencarian terhadap pohon permainan yang melibatkan banyak ronde, maka sangat mungkin bahwa pencarian terpaksa dihentikan sebelum berhasil menemukan langkah manapun yang berarti.

Solusi dari permasalahan ini adalah penerapan teknik *iterative deepening search* (IDS), yaitu teknik melakukan pencarian dengan kedalaman dangkal, dan kemudian mengulangi pencarian untuk kedalaman lebih tinggi. Secara khusus, pencarian dilakukan pada kedalaman 2, kemudian dilakukan lagi pada kedalaman 4, kemudian pada kedalaman 6, dan seterusnya hingga mencapai kedalaman maksimum, yaitu banyak *ply* yang tersisa pada permainan. Dengan demikian, jika pencarian dihentikan sebelum mencapai kedalaman maksimum, sudah ada hasil pencarian aksi yang cukup baik untuk kedalaman yang lebih dangkal yang dapat digunakan.

Kelemahan dari penggunaan teknik ini adalah pencarian pada kedalaman tinggi harus melakukan ulang banyak kerja yang sudah dilakukan oleh pencarian yang lebih dangkal. Hal ini dapat dimitigasi dengan dua teknik optimasi tambahan:

1. *Best move heuristic*, yaitu heuristik memulai pencarian lebih dalam pada urutan aksi terbaik yang ditemukan oleh pencarian sebelumnya. Ide dari teknik ini adalah bahwa urutan aksi terbaik pada pencarian dangkal seringkali juga merupakan urutan aksi terbaik pada pencarian lebih dalam, sehingga pencarian dapat melakukan lebih banyak *pruning* pada pohon pencarian yang lebih besar jika urutan aksi ini diperiksa lebih dulu.
2. *Transposition table*, yaitu sebuah tabel berisi berbagai *state* permainan yang sudah diperiksa, beserta nilai evaluasinya. Ide dibalik teknik ini adalah bahwa mayoritas dari *state* yang dibangkitkan oleh pencarian merupakan *state* yang sama yang dicapai lewat urutan aksi yang berbeda. Setiap kali algoritma Minimax menghasilkan suatu nilai evaluasi untuk sebuah *state*, *state* tersebut dan evaluasinya disimpan pada *transposition table*. Ketika *state* ini dicapai lewat urutan aksi lain, pemeriksaan tidak perlu dilakukan lebih lanjut, dan nilai evaluasi yang sudah ada pada tabel akan digunakan sebagai hasil evaluasi *state* ini. Implementasi yang dipilih pada tugas ini hanya menggunakan versi sederhana dari *transposition table*, yaitu sebuah tabel *hash* yang memetakan sebuah *state* papan permainan dan pemain yang memegang giliran saat ini (dikuncikan menggunakan *hash* Zobrist dari kedua atribut tersebut) terhadap nilai evaluasi yang ditemukan.

Menggunakan IDS dan menerapkan optimasi seperti di atas, algoritma mampu melakukan pencarian hingga kedalaman 4 secara konsisten, dan melakukan pencarian lebih dalam secara parsial untuk memeriksa kelanjutan urutan aksi terbaik sejauh ini. Perlu diperhatikan bahwa urutan aksi hasil pencarian bukanlah aksi terbaik secara mutlak, melainkan sebuah optimum lokal, karena pencarian tidak dilakukan pada pohon permainan secara lengkap.

3. Proses Pencarian dengan *Local Beam Search*

Algoritma *local search* yang digunakan dalam Adjacency Strategy Game adalah *local beam search*. Berbeda dengan algoritma *local search* lainnya yang mempertimbangkan satu buah *state*, algoritma *local beam search* mempertimbangkan k buah *states*. Prinsip *local beam search* mirip seperti *breadth first search* yang dibatasi. Pada kasus ini, k buah *states* yang dibangkitkan merupakan k buah *states* yang dapat menghasilkan nilai terbaik.

Dalam permainan ini, definisi yang digunakan adalah sebagai berikut.

1. *State*: Keadaan dari permainan pada suatu waktu, mencakup keadaan papan permainan dan ronde.
2. Aksi: Menempatkan markah pada suatu kotak kosong tertentu.
3. *Evaluation tree*: pohon evaluasi dengan *node* berupa *state* papan permainan dan *edge* berupa aksi yang dilakukan pada permainan;
4. *k*: banyaknya *state* yang dipertimbangkan pada setiap *level*;
5. Fungsi utilitas: Nilai utilitas dari sebuah *state* menggambarkan seberapa baik *state* tersebut bagi bot. Definisi yang dipilih untuk fungsi utilitas ini adalah $U(s) = O - X$, di mana *O* adalah banyak markah O di atas papan, dan *X* adalah banyak markah X di atas papan.

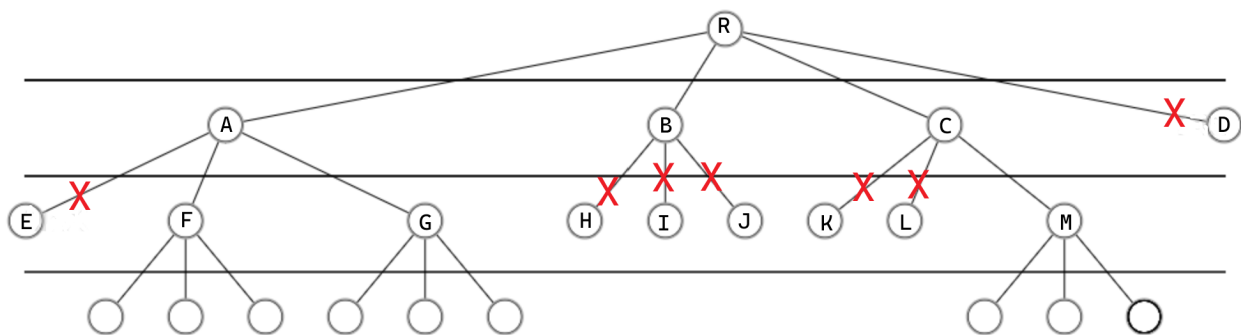
Berikut merupakan langkah-langkah strategi pencarian langkah optimum dengan menggunakan *local beam search*.

1. Tentukan *k*.
2. Inisiasi pohon evaluasi dengan *root node* berupa *state* papan permainan pada saat ini.
3. Evaluasi *root node* dengan membangkitkan semua kemungkinan aksi yang dapat dilakukan oleh *bot*.
4. Lakukan evaluasi terhadap pohon evaluasi dengan algoritma *minimax*, sehingga didapatkan nilai evaluasi pada setiap *node* pohon. Adapun nilai evaluasi didapatkan dengan menggunakan fungsi utilitas $U(s) = O - X$.
5. Dari sekian *states* yang mungkin dihasilkan tersebut, pilih *k* buah *states* dengan nilai evaluasi terbaik. *Nodes* yang tidak dipilih akan dibuang dari pohon evaluasi.
6. Untuk level berikutnya, evaluasi anak *node* pertama dari *root node* dengan membangkitkan semua kemungkinan aksi yang dapat dilakukan oleh *bot*. Lanjutkan dengan evaluasi seluruh anak *node* berikutnya dari *root node*.
7. Lakukan evaluasi terhadap pohon evaluasi dengan algoritma *minimax*, sehingga didapatkan nilai evaluasi pada setiap *node* pohon.
8. Dari semua *states* yang mungkin dihasilkan tersebut, pilih *k* buah *states* dengan nilai evaluasi yang terbaik. *Nodes* yang tidak dipilih akan dibuang dari pohon evaluasi. Lalu, cek juga *nodes* pada level sebelumnya. Jika *node* pada level sebelumnya tidak

- memiliki anak *node*, maka buang *node* tersebut dari pohon evaluasi karena *node* tersebut tidak akan mencapai seluruh langkah yang mungkin dilakukan hingga akhir permainan.
9. Lakukan hal yang sama untuk level berikutnya hingga mencapai daun.
 10. Setelah mencapai daun, lakukan evaluasi terhadap pohon evaluasi dengan algoritma *minimax*. Hasil nilai evaluasi pada *root node* merupakan nilai terbaik untuk pengambilan langkah permainan.

Sebagai contoh, Gambar 3.1 merupakan pohon evaluasi *local beam search* sederhana. *State* papan permainan saat ini disimpan pada *root node* R. Kemudian, dari *state* R, dapat dibangkitkan *state* A, B, C, dan D. Keempat *state* ini diurutkan dari nilai evaluasi yang terbesar, yaitu A, B, C, dan D. Misalkan $k = 3$, maka hanya *nodes* yang dipilih untuk dilakukan evaluasi lebih lanjut adalah *nodes* A, B, dan C. Karena tidak dipilih, maka *node* D dibuang dari pohon.

Selanjutnya, bangkitkan seluruh *state* yang mungkin dilakukan dari *state* A, *state* B, dan *state* C. Di antara seluruh *state* yang mungkin dilakukan, pilih k (3) buah *states* dengan nilai evaluasi terbaik untuk dievaluasi lebih lanjut. Pada contoh ini, *states* dengan nilai terbaik adalah F, G, dan M. *Nodes* E, H, I, J, K, dan L tidak dipilih dan dibuang dari pohon. Pada level berikutnya, lakukan kembali evaluasi dengan membangkitkan *states* yang mungkin dilakukan oleh *state* F, G, dan H. Hal ini terus diulangi hingga mencapai daun.



Gambar 3.1 Contoh *Local Beam Search* Sederhana

Hasil pencarian *local beam search* dipilih karena *local beam search* mencari sejumlah *states* yang kemungkinan dapat menghasilkan *state* terbaik. Dengan demikian, tidak semua *states* dianalisis, sehingga waktu pencarian yang dilakukan cepat. Akan tetapi, *local beam search*

mungkin mengambil langkah permainan bukan yang terbaik karena *states* yang dievaluasi dibatasi sejumlah k buah *states*.

Algoritma *local beam search* berbeda dengan algoritma *greedy* yang hanya melakukan evaluasi sebanyak satu level pada pohon evaluasi. Algoritma *greedy* hanya mempertimbangkan *state* papan permainan saat ini dan mengambil langkah terbaik saat ini, sementara algoritma *local beam search* mempertimbangkan langkah terbaik dengan memprediksi langkah-langkah yang akan diambil lawan dan *bot*.

Algoritma *local beam search* lebih cepat dibandingkan algoritma *minimax* dengan *alpha beta pruning* karena *local beam search* hanya mempertimbangkan k buah *states*, sementara *minimax* dengan *alpha beta pruning* mempertimbangkan seluruh *states* yang mungkin dibangkitkan. Akan tetapi, *minimax* dengan *alpha beta pruning* menghasilkan langkah yang lebih optimal karena mempertimbangkan seluruh kemungkinan *states* yang dapat diambil oleh lawan dan *bot*.

4. Proses Pencarian dengan *Genetic Algorithm*

Strategi *genetic algorithm* tidak dapat digunakan untuk permainan ini. Hal ini disebabkan oleh ketidaksesuaian pemakaian *genetic algorithm* untuk mencari *compound optima*. Permainan Adjacency Strategy Game merupakan permainan yang dapat dimainkan oleh dua orang. Ini berarti permainan membutuhkan hasil berupa *compound optima*, yaitu asumsi bahwa strategi menghasilkan *state* yang terbaik ketika *bot* melakukan aksi dan *state* yang terburuk ketika pemain lawan melakukan aksi. Oleh karena itu, permainan ini membutuhkan strategi *genetic algorithm* dengan pendekatan yang memungkinkan *bot* untuk memilih aksi yang terbaik.

Ide strategi *genetic algorithm* ini terinspirasi dari artikel “Adversarial Search by Evolutionary Computation” oleh Hong et al., 2001^[1]. Pada kasus ini, terdapat beberapa variabel dan fungsi sebagai berikut:

1. Banyaknya populasi (k)
2. Banyaknya generasi (n)
3. Laju mutasi (*mutation rate*)
4. Array rangkaian aksi: serangkaian aksi yang dibangkitkan dari awal hingga akhir permainan. Rangkaian aksi berupa array satu dimensi yang tersusun atas nomor-nomor kotak (01 – 64) penempatan markah yang dilakukan *bot* dan pemain

lawan. Adapun nomor kotak yang dapat dibangkitkan adalah nomor 01 sampai 64, kecuali kotak 07, 08, 15, 16, 49, 50, 57, dan 58 karena kotak-kotak tersebut telah terisi pada awal permainan. Tidak boleh terdapat nomor kotak yang sama pada satu rangkaian aksi.

5. *Reservation tree*: kumpulan *array* rangkaian aksi yang digabungkan dalam bentuk tipe data pohon (*tree*).
6. Fungsi evaluasi papan: selisih banyaknya markah bot dengan banyaknya markah pemain lawan.
7. *Fitness function*: banyaknya level suatu nilai evaluasi papan dipertahankan ketika mengoperasikan algoritma Minimax.

Berikut merupakan langkah-langkah strategi pencarian langkah optimum dengan strategi *genetic algorithm* dengan pendekatan baru:

1. Tentukan banyaknya populasi (k) dan banyaknya generasi (n).
2. Tentukan kedalaman pohon pencarian, yaitu sebanyak $2 * \text{banyak ronde permainan}$.
3. Tentukan laju mutasi.
4. Inisiasi sebuah *reservation tree*.
5. Bangkitkan k buah *array* rangkaian aksi. Rangkaian aksi dibangkitkan secara acak dengan syarat tidak ada nomor kotak yang sama pada satu rangkaian aksi.
6. Gabungkan k buah rangkaian aksi yang telah dibangkitkan ke dalam *reservation tree*. Dalam bentuk pohon, aksi direpresentasikan oleh busur. Simpul pohon merepresentasikan *state* papan permainan ketika aksi dilakukan.
7. Hitung nilai evaluasi papan setiap daun pada *reservation tree*.
8. Lakukan algoritma Minimax terhadap *reservation tree*.
9. Hitung nilai F dengan menggunakan Fitness Function.
10. Bangkitkan generasi baru dengan cara memilih dan melakukan *crossover* antara dua *parent array* rangkaian aksi yang telah dibangkitkan. Pemilihan *parent* dilakukan secara acak. Akan tetapi, *parent* dengan nilai F yang besar memiliki peluang yang lebih besar untuk terpilih. Dua buah *parent* yang terpilih di-*crossover*. Titik pemisahan pada proses *crossover* ditentukan secara acak. Setelah mengoperasikan *crossover* pada kedua *parent*, lakukan mutasi dengan titik mutasi yang ditentukan

secara acak. Perlu diingat bahwa terdapat kemungkinan rangkaian aksi menjadi tidak valid setelah dilakukan *crossover*. Rangkaian aksi yang tidak valid adalah rangkaian aksi yang memiliki aksi berulang pada koordinat yang sama. Oleh karena itu, jika terdapat aksi yang berulang, aksi tersebut digantikan dengan koordinat yang valid secara acak. Untuk mutasi, bangkitkan angka acak untuk menentukan apakah mutasi perlu dilakukan atau tidak. Jika angka acak tersebut lebih kecil dibandingkan laju mutasinya, maka lakukan mutasi dengan cara menukar (*swap*) dua aksi pada rangkaian aksi.

11. Gabungkan generasi baru yang telah dibangkitkan dengan *reservation tree*.
12. Ulangi mulai dari langkah 5 untuk membangkitkan generasi berikutnya.
13. Ambil nilai pada akar *reservation tree* untuk menentukan langkah yang harus diambil bot.

Hasil pencarian *genetic algorithm* adalah satu atau lebih lintasan pada *reservation tree* dari *state* awal menuju salah satu *terminal state*, di mana semua *state* di sepanjang lintasan ini memiliki nilai evaluasi papan yang sama. Lintasan ini mewakili urutan kotak kosong yang akan dibubuhi markah jika bot dan pemain selalu bermain secara optimal. Urutan ini mengoptimalkan peluang kemenangan bagi bot dalam permainan.

Untuk tipe permainan dengan dua pemain, strategi *genetic algorithm* kurang cocok jika banyaknya ronde sedikit karena kurang efisien jika dibandingkan dengan strategi *minimax* klasik. Namun, jika banyaknya ronde permainan cukup banyak, strategi *genetic algorithm* bisa jadi lebih efisien karena *state* akhir yang dievaluasi lebih sedikit. Hal ini disebabkan oleh kecenderungan algoritma untuk memilih lintasan yang lebih baik untuk membentuk generasi baru.

Selain pertimbangan di atas, strategi *genetic algorithm* yang diimplementasikan memiliki kekurangan, yaitu faktor pembangkitan aksi acak pada saat tahap *crossover* dan *mutation*. Tahap *crossover* pada *genetic algorithm* dipengaruhi oleh sisa ronde. Jika sisa ronde semakin banyak, maka rangkaian aksi yang dibangkitkan lebih banyak, sehingga ketika dilakukan *crossover*, peluang rangkaian aksi menjadi invalid lebih tinggi karena ada aksi berulang. Rangkaian aksi yang memiliki aksi yang berulang merupakan rangkaian aksi yang tidak valid. Aksi yang berulang harus digantikan oleh aksi valid acak lainnya. Penggantian aksi acak inilah yang

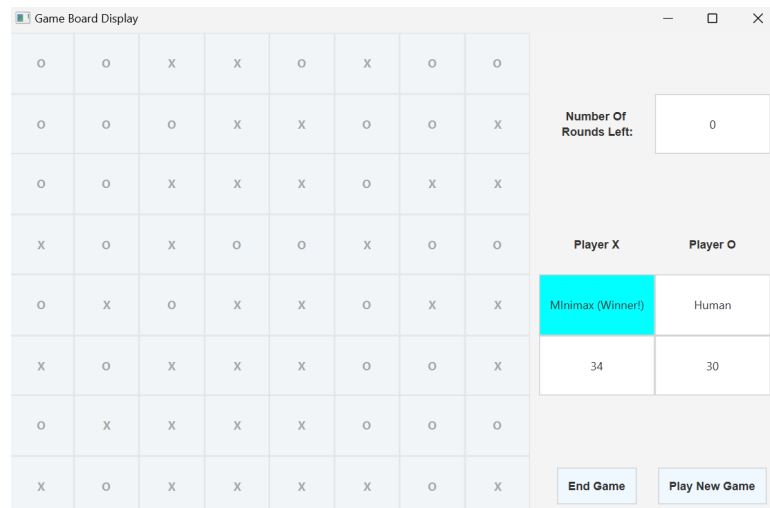
membuat tingkat keacakan menjadi lebih tinggi dan *bot* tidak dapat mengambil langkah optimal karena laju mutasi secara efektif jauh lebih tinggi dibandingkan laju mutasi yang ditentukan.

5. Hasil Pertandingan

Berikut merupakan hasil pertandingan *bot*.

5.1 Bot Minimax vs Manusia

5.1.1 Pertandingan 1



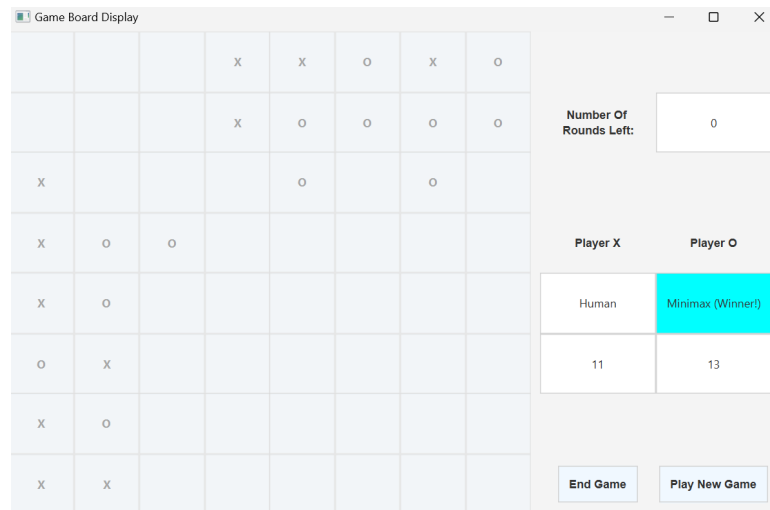
Game Board Display							
O	O	X	X	O	X	O	O
O	O	O	X	X	O	O	X
O	O	X	X	X	O	X	X
X	O	X	O	O	X	O	O
O	X	O	X	X	O	X	X
X	O	X	X	X	O	O	X
O	X	X	X	X	O	O	O
X	O	X	X	X	X	O	X

Number Of Rounds Left:	0
Player X	Player O
Minimax (Winner!)	Human
34	30
End Game	Play New Game

Gambar 5.1.1.1 Hasil Pertandingan 1 Bot Minimax vs Manusia

Pemain 1 (X) : Bot Minimax
Pemain 2 (O) : Manusia
Jumlah ronde : 28
Pemain pertama : Bot Minimax
Pemenang : Bot Minimax

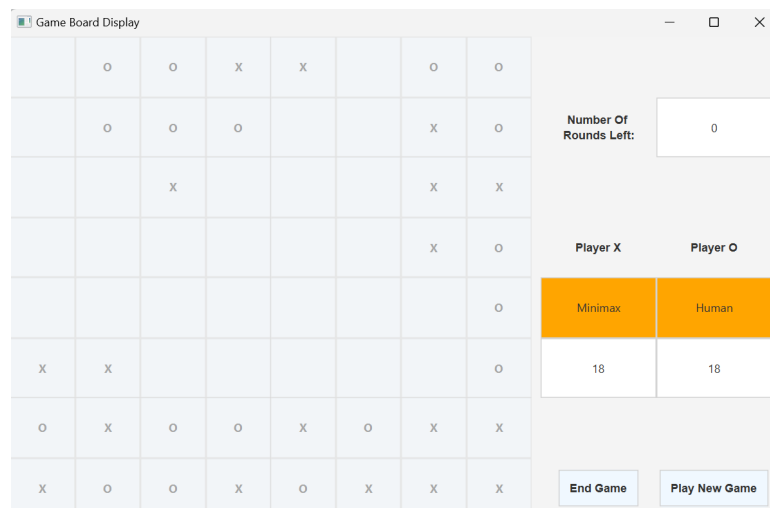
5.1.2 Pertandingan 2



Gambar 5.1.2.1 Hasil Pertandingan 2 Bot Minimax vs Manusia

Pemain 1 (X) : Manusia
Pemain 2 (O) : Bot Minimax
Jumlah ronde : 8
Pemain pertama : Manusia
Pemenang : Bot Minimax

5.1.3 Pertandingan 3

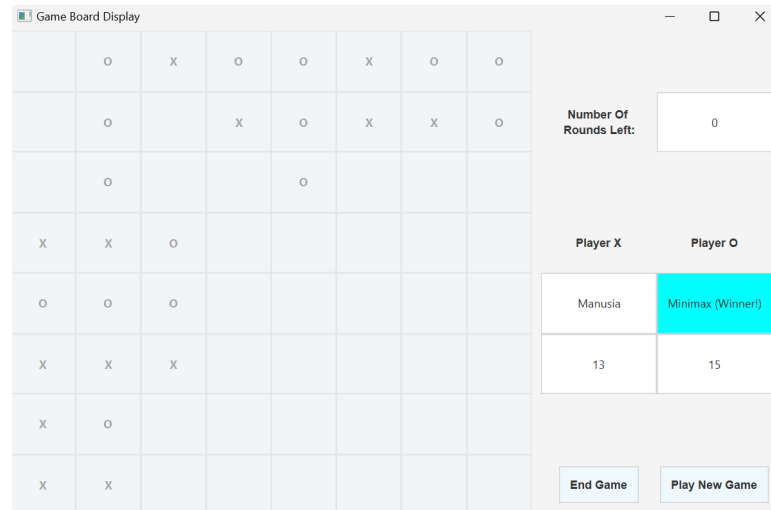


Gambar 5.1.3.1 Hasil Pertandingan 3 Bot Minimax vs Manusia

Pemain 1 (X) : Bot Minimax
Pemain 2 (O) : Manusia

Jumlah ronde : 14
 Pemain pertama : Bot Minimax
 Pemenang : Bot Minimax & Manusia

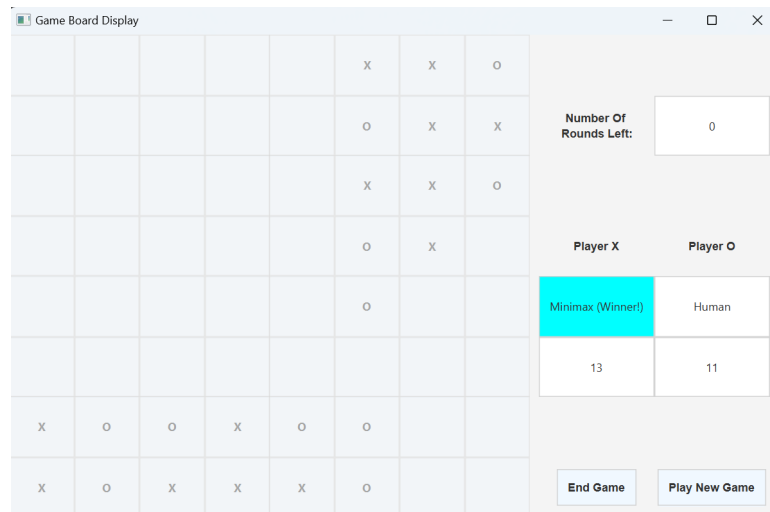
5.1.4 Pertandingan 4



Gambar 5.1.4.1 Hasil Pertandingan 4 Bot Minimax vs Manusia

Pemain 1 (X) : Manusia
 Pemain 2 (O) : Bot Minimax
 Jumlah ronde : 10
 Pemain pertama : Manusia
 Pemenang : Bot Minimax

5.1.5 Pertandingan 5



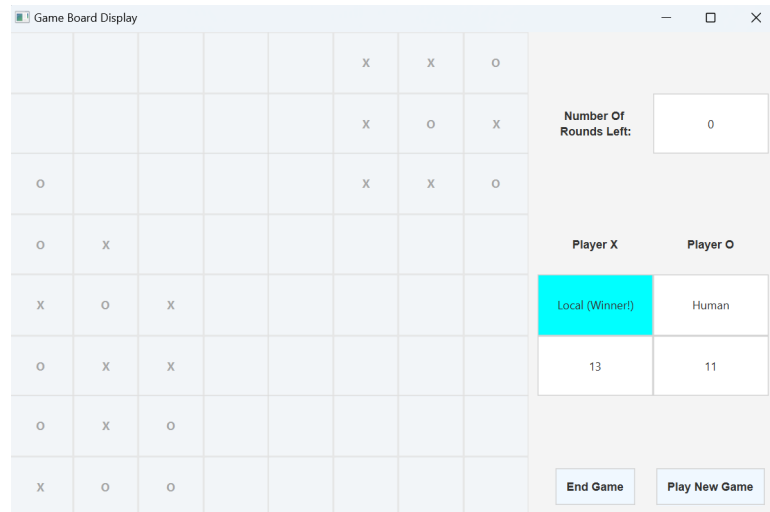
Gambar 5.1.5.1 Hasil Pertandingan 5 Bot Minimax vs Manusia

Pemain 1 (X) : Bot Minimax
Pemain 2 (O) : Manusia
Jumlah ronde : 8
Pemain pertama : Bot Minimax
Pemenang : Bot Minimax

Berdasarkan lima pertandingan Bot Minimax melawan manusia, empat kali pertandingan Bot Minimax mengungguli manusia dan satu kali pertandingan seri. Ini menunjukkan bahwa Bot Minimax sudah mampu mengambil langkah terbaik dengan memprediksi langkah-langkah yang mungkin diambil manusia ketika bermain.

5.2 Bot Local Search vs Manusia

5.2.1 Pertandingan 1



Gambar 5.2.1.1 Hasil Pertandingan 1 Bot Local Search vs Manusia

Pemain 1 (X) : Bot Local Search

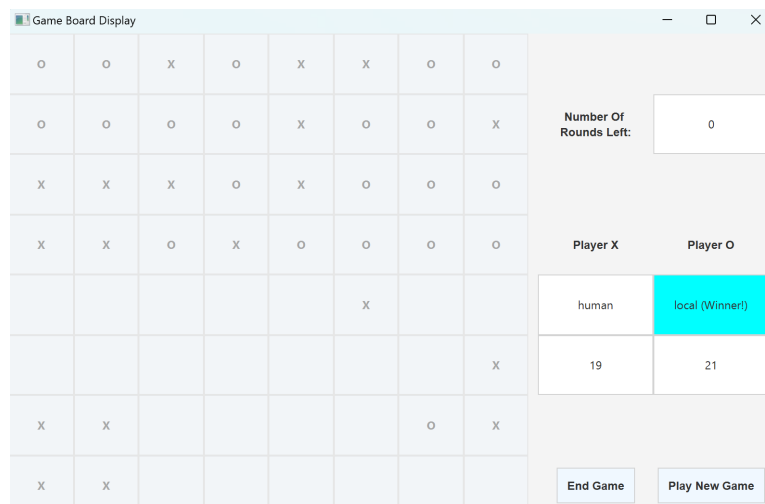
Pemain 2 (O) : Manusia

Jumlah ronde : 8

Pemain pertama : Bot Local Search

Pemenang : Bot Local Search

5.2.2 Pertandingan 2



Gambar 5.2.2.1 Hasil Pertandingan 2 Bot Local Search vs Manusia

Pemain 1 (X) : Manusia

Pemain 2 (O) : Bot Local Search

Jumlah ronde : 16
 Pemain pertama : Manusia
 Pemenang : Bot Local Search

5.2.3 Pertandingan 3



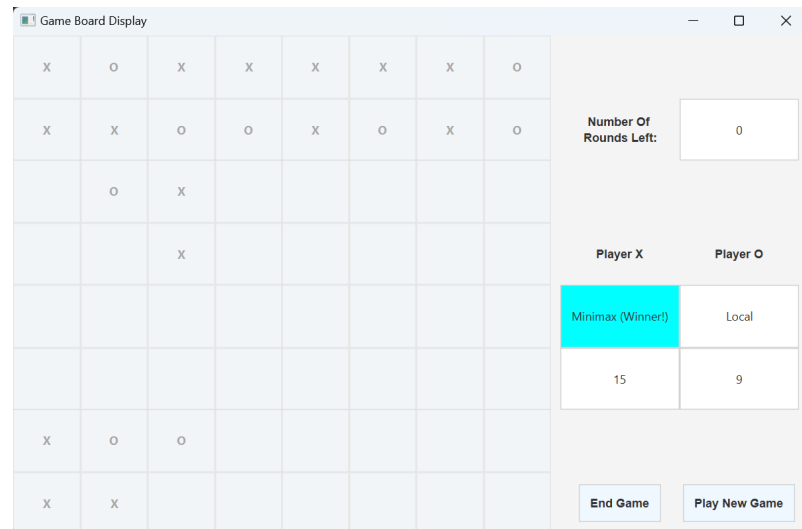
Gambar 5.2.3.1 Hasil Pertandingan 3 Bot Local Search vs Manusia

Pemain 1 (X) : Manusia
 Pemain 2 (O) : Bot Local Search
 Jumlah ronde : 28
 Pemain pertama : Manusia
 Pemenang : Bot Local Search

Pertandingan antara Bot Local Search melawan manusia tidak dapat dianalisis secara objektif karena hasil pertandingan bergantung kepada tingkat keahlian manusia dalam bermain, sehingga hasilnya cenderung subjektif. Dari tiga pertandingan di atas, Bot Local Search selalu mengungguli manusia.

5.3 Bot Minimax vs Bot Local Search

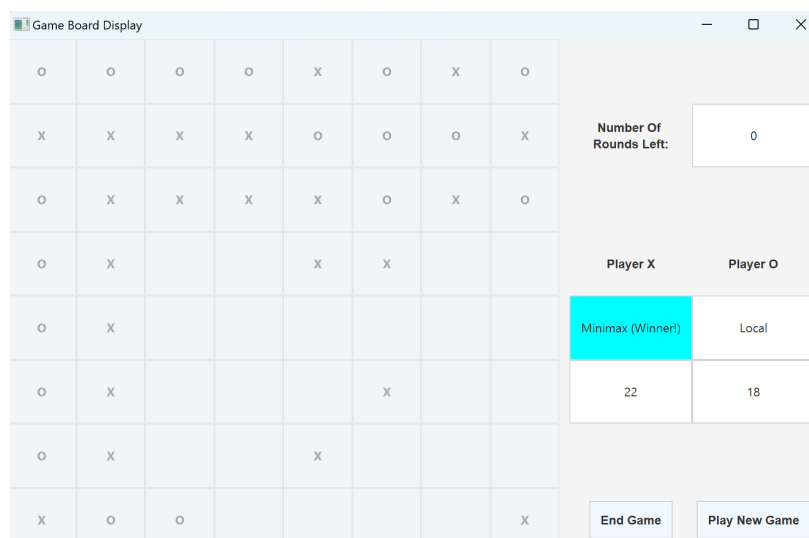
5.3.1 Pertandingan 1



Gambar 5.3.1.1 Hasil Pertandingan 1 Bot Minimax vs Bot Local Search

Pemain 1 (X) : Bot Minimax
Pemain 2 (O) : Bot Local Search
Jumlah ronde : 8
Pemain pertama : Bot Minimax
Pemenang : Bot Minimax

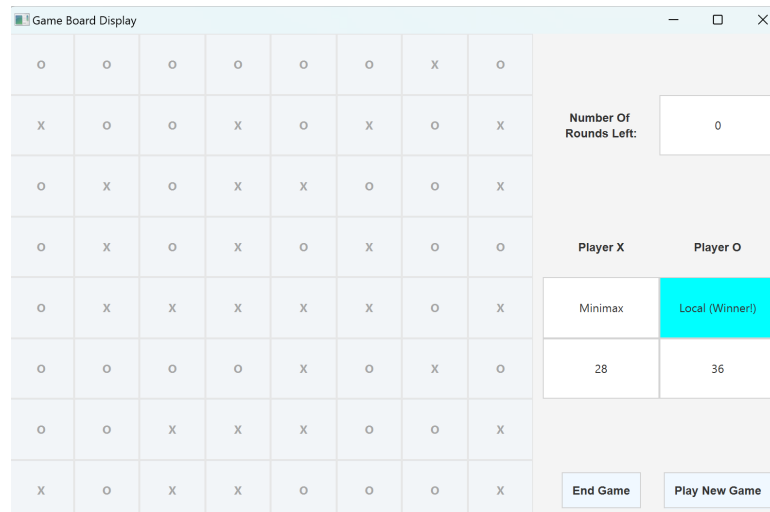
5.3.2 Pertandingan 2



Gambar 5.3.2.1 Hasil Pertandingan 2 Bot Minimax vs Bot Local Search

Pemain 1 (X) : Bot Minimax
 Pemain 2 (O) : Bot Local Search
 Jumlah ronde : 16
 Pemain pertama : Bot Local Search
 Pemenang : Bot Minimax

5.3.3 Pertandingan 3



Gambar 5.3.3.1 Hasil Pertandingan 3 Bot Minimax vs Bot Local Search

Pemain 1 (X) : Bot Minimax
 Pemain 2 (O) : Bot Local Search
 Jumlah ronde : 28
 Pemain pertama : Bot Minimax
 Pemenang : Bot Local Search

Pertandingan Bot Minimax melawan Bot Local Search berlangsung sengit. Pada ketiga pertandingan, Bot Minimax dan Bot Local Search memiliki skor yang cenderungimbang dari awal hingga menuju akhir pertandingan. Akan tetapi, Bot Local Search memiliki waktu berpikir yang cenderung lebih singkat. Hal ini terjadi karena pada Bot Minimax, pohon evaluasi tidak sempat diperiksa hingga ke level daun karena keterbatasan *timer* 5 detik, sementara Bot Local Search memeriksa hingga ke level daun tetapi dengan pohon evaluasi yang tidak lengkap.

5.4 Bot Minimax vs Bot Genetic Algorithm

5.4.1 Pertandingan 1

Game Board Display								— □ ×	
X	O	O	O	X	O	X	O	Number Of Rounds Left: 0	
X	X	O	O	X	X	O	X		
X	X	O	O	O	X	X	O		
X	O	X	O	O	X	X	O		
O	O	X	X	X	O	X	O	Player X	Player O
X	O	X	X	O	X	O	O	Minimax (Winner!)	Genetic
O	X	X	X	X	X	X	O	33	31
X	O	X	X	O	O	O	O	End Game	Play New Game

Gambar 5.4.1.1 Hasil Pertandingan 1 Bot Minimax vs Bot Genetic Algorithm

Pemain 1 (X) : Bot Minimax
Pemain 2 (O) : Bot Genetic Algorithm
Jumlah ronde : 28
Pemain pertama : Bot Minimax
Pemenang : Bot Minimax

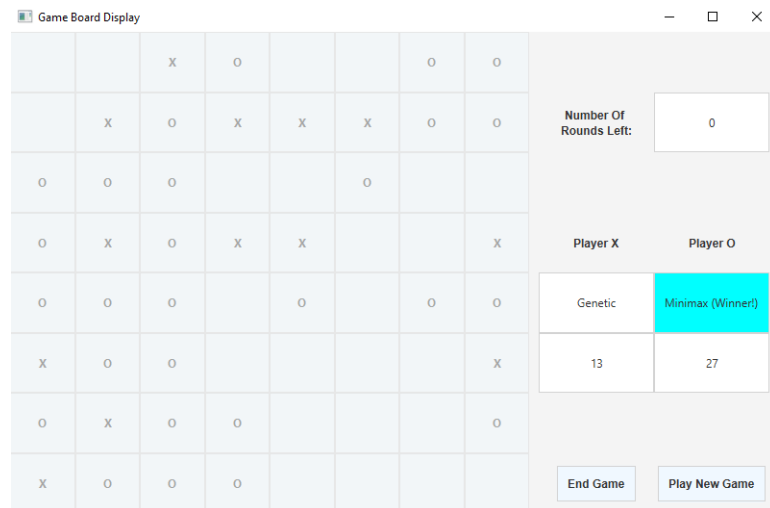
5.4.2 Pertandingan 2

Game Board Display								— □ ×	
O	X					O	O	Number Of Rounds Left: 0	
	X					O	X		
							X		
O				X	O				
O				O	O			Player X	Player O
O	X	O						Genetic	Minimax (Winner!)
X	O	O						8	16
X	O	O	O					End Game	Play New Game

Gambar 5.4.2.1 Hasil Pertandingan 2 Bot Minimax vs Bot Genetic Algorithm

Pemain 1 (X) : Bot Genetic Algorithm
 Pemain 2 (O) : Bot Minimax
 Jumlah ronde : 8
 Pemain pertama : Bot Genetic Algorithm
 Pemenang : Bot Minimax

5.4.3 Pertandingan 3



Gambar 5.4.3.1 Hasil Pertandingan 3 Bot Minimax vs Bot Genetic Algorithm

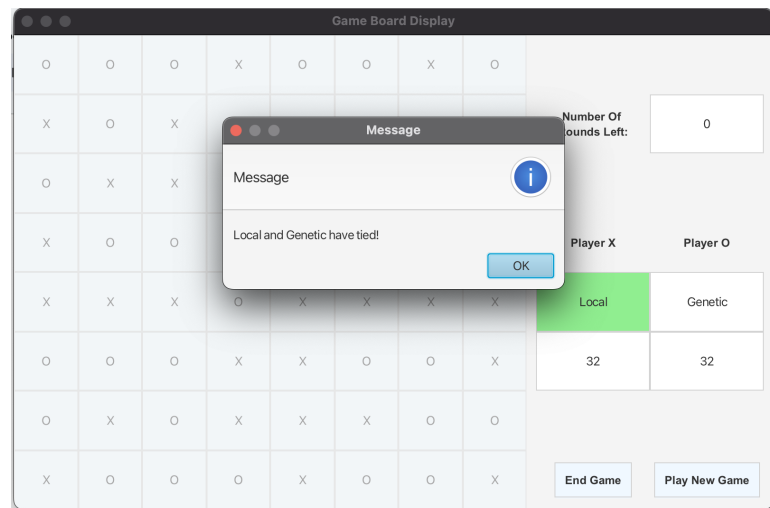
Pemain 1 (X) : Bot Genetic Algorithm
 Pemain 2 (O) : Bot Minimax
 Jumlah ronde : 16
 Pemain pertama : Bot Genetic Algorithm
 Pemenang : Bot Minimax

Berdasarkan tiga kali pertandingan antara Bot Minimax dengan Bot Genetic Algorithm, Bot Minimax jauh mengungguli Bot Genetic Algorithm. Hal ini disebabkan oleh metode Bot Minimax yang melakukan evaluasi terhadap seluruh kemungkinan *state* yang dapat dibangkitkan, sehingga menghasilkan langkah yang optimal. Bot Genetic Algorithm membangkitkan *state* inisiasi secara acak lalu memilih *states* dengan nilai Fitness yang terbaik. Ini tentu saja lebih buruk dibandingkan Bot Minimax yang memeriksa semua kemungkinan *state* karena tingkat keacakan Bot Genetic Algorithm yang cukup tinggi.

Tingkat keacakan Bot Genetic Algorithm dipengaruhi oleh banyaknya rangkaian aksi yang dibangkitkan. Tahap *crossover* pada *genetic algorithm* dipengaruhi oleh sisa ronde. Jika sisa ronde semakin banyak, maka rangkaian aksi yang dibangkitkan lebih banyak, sehingga ketika dilakukan *crossover*, peluang rangkaian aksi menjadi invalid lebih tinggi karena ada aksi berulang. Aksi yang berulang harus digantikan oleh aksi valid acak lainnya. Penggantian aksi acak inilah yang membuat tingkat keacakan menjadi lebih tinggi dan *bot* tidak dapat mengambil langkah optimal karena laju mutasi secara efektif jauh lebih tinggi dibandingkan laju mutasi yang ditentukan.

5.5 Bot Local Search vs Genetic Algorithm

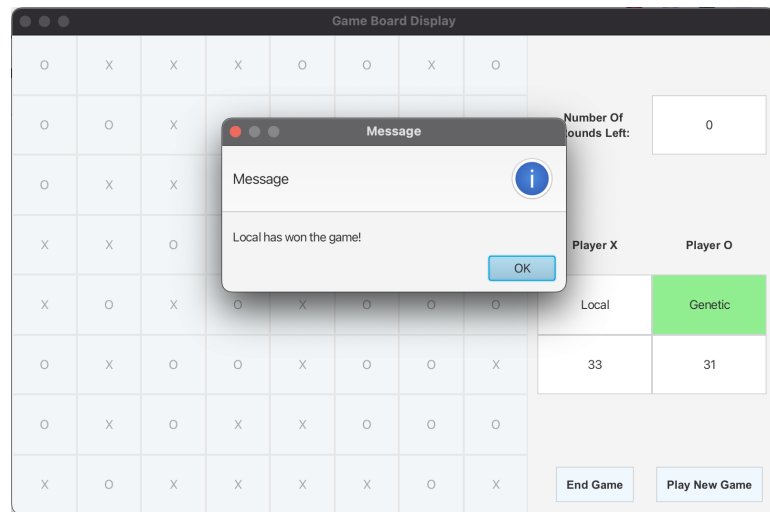
5.5.1 Pertandingan 1



Gambar 5.5.1.1 Hasil Pertandingan 1 Bot Local Search vs Bot Genetic Algorithm

Pemain 1 (X)	: Bot Local Search
Pemain 2 (O)	: Bot Genetic Algorithm
Jumlah ronde	: 28
Pemain pertama	: Bot Local Search
Pemenang	: - (Seri)

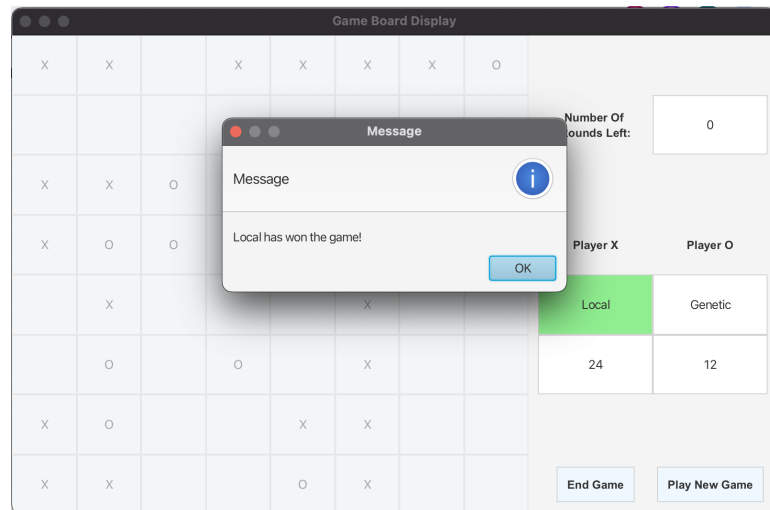
5.5.2 Pertandingan 2



Gambar 5.5.2.1 Hasil Pertandingan 2 Bot Local Search vs Bot Genetic Algorithm

Pemain 1 (X) : Bot Local Search
Pemain 2 (O) : Bot Genetic Algorithm
Jumlah ronde : 28
Pemain pertama : Bot Genetic Algorithm
Pemenang : Bot Local Search

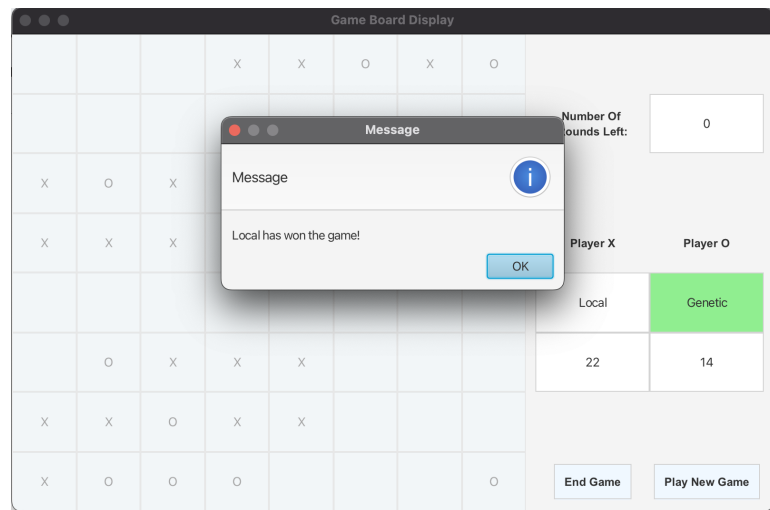
5.5.3 Pertandingan 3



Gambar 5.5.3.1 Hasil Pertandingan 2 Bot Local Search vs Bot Genetic Algorithm

Pemain 1 (X) : Bot Local Search
 Pemain 2 (O) : Bot Genetic Algorithm
 Jumlah ronde : 14
 Pemain pertama : Bot Local Search
 Pemenang : Bot Local Search

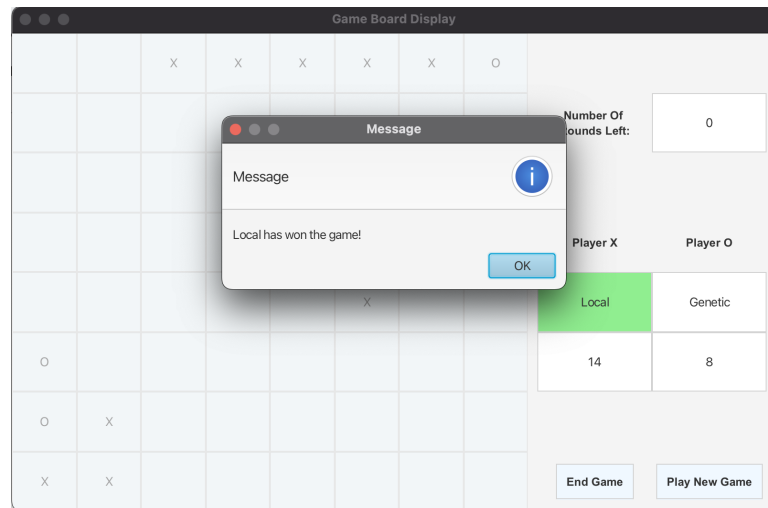
5.5.4 Pertandingan 4



Gambar 5.5.4.1 Hasil Pertandingan 2 Bot Local Search vs Bot Genetic Algorithm

Pemain 1 (X) : Bot Local Search
 Pemain 2 (O) : Bot Genetic Algorithm
 Jumlah ronde : 14
 Pemain pertama : Bot Genetic Algorithm
 Pemenang : Bot Local Search

5.5.5 Pertandingan 5



Gambar 5.5.5.1 Hasil Pertandingan 2 Bot Local Search vs Bot Genetic Algorithm

Pemain 1 (X) : Bot Local Search
Pemain 2 (O) : Bot Genetic Algorithm
Jumlah ronde : 7
Pemain pertama : Bot Local Search
Pemenang : Bot Local Search

Berdasarkan lima pertandingan antara Bot Local Search melawan Bot Genetic Algorithm, Bot Local Search cenderung selalu mengungguli Bot Genetic Algorithm (4 kali menang dan 1 kali seri). Hal ini disebabkan oleh metode evaluasi Bot Local Search yang memeriksa pohon evaluasi hingga level daun. Bot Genetic Algorithm memiliki faktor tingkat keacakan yang cukup tinggi pada awal permainan, sehingga langkah yang diambil kurang optimal. Namun menuju akhir pertandingan, tingkat keacakan Bot Genetic Algorithm mulai berkurang karena banyaknya ronde yang berkurang, sehingga tingkat mutasi dan keacakan berkurang.

Tingkat keacakan Bot Genetic Algorithm dipengaruhi oleh banyaknya rangkaian aksi yang dibangkitkan. Tahap *crossover* pada *genetic algorithm* dipengaruhi oleh sisa ronde. Jika sisa ronde semakin banyak, maka rangkaian aksi yang dibangkitkan lebih banyak, sehingga ketika dilakukan *crossover*, peluang rangkaian aksi menjadi invalid lebih tinggi karena ada aksi berulang. Aksi yang berulang harus digantikan oleh aksi valid acak lainnya. Penggantian aksi acak inilah yang membuat tingkat keacakan menjadi lebih tinggi dan *bot* tidak dapat mengambil

langkah optimal karena laju mutasi secara efektif jauh lebih tinggi dibandingkan laju mutasi yang ditentukan.

Tingkat keacakan Bot Genetic Algorithm juga cenderung lebih tinggi jika Bot Genetic Algorithm menjadi permainan pertama. Hal ini terjadi karena Bot Genetic Algorithm belum dapat memprediksi dan menentukan langkah terbaik ketika *state* awal seri dengan pemain lawan.

6. Referensi

- Alliot, J., & Durand, N. (1996). A genetic algorithm to improve an Othello program. In *Springer eBooks* (pp. 305–319). https://doi.org/10.1007/3-540-61108-8_46
- Greenblatt, R. D., Eastlake, D., & Crocker, S. (1967). The Greenblatt chess program. *Fall Joint Computer Conference*. <https://doi.org/10.1145/1465611.1465715>
- Hong, T., Huang, K., & Lin, W. (2001). Adversarial search by evolutionary computation. *Evolutionary Computation*. <https://doi.org/10.1162/106365601750406046>
- Russell, S., & Norvig, P. (2019). *Artificial intelligence: A Modern Approach* (4th ed.). Pearson Higher Education.

7. Pembagian Pekerjaan

Berikut merupakan pembagian pekerjaan pada Tugas Besar 1 IF3170 Inteligensi Buatan.

NIM	Nama	Pekerjaan
13521059	Arleen Chrysanthia Gunardi	Adversarial dan Local Search
13521107	Jericho Russel Sebastian	Minimax Algorithm
13521125	Asyifa Nurul Shafira	Minimax Algorithm
13521133	Cetta Reswara Parahita	Adversarial dan Local Search

8. Lampiran

Repositori GitHub: https://github.com/JerichoFletcher/Tubes1_AI_Adjacency