

IF3170 Inteligensi Buatan

Minimax Algorithm and Alpha Beta Pruning in Adjacency Strategy Game

Tugas Besar 1



Oleh:

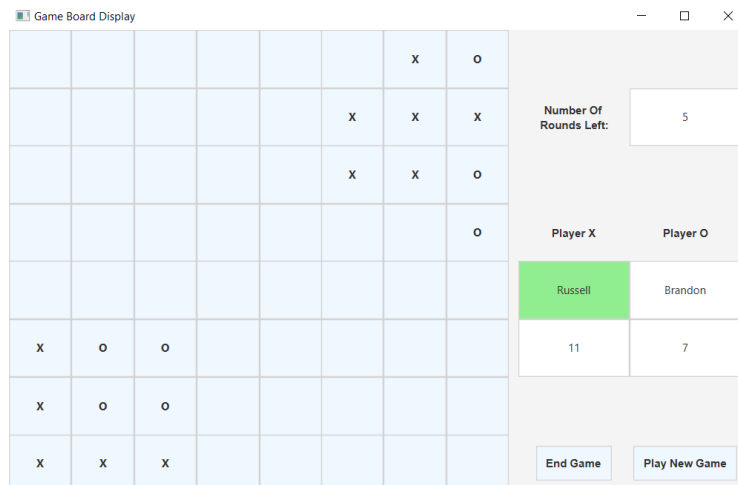
Frankie Huang	13521092
Michael Utama	13521137
Johann Christian Kandani	13521138
Dewana Gustavus Haraka Otang	13521173

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

Daftar Isi

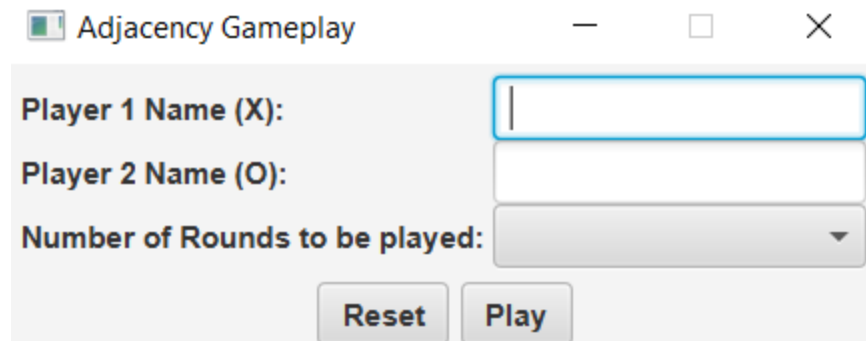
Daftar Isi	1
I. Adjacency Strategy Game	2
II. Objective Function	3
III. Minimax Algorithm dan Alpha Beta Pruning	4
IV. Local Search	5
V. Genetic Algorithm	5
VI. Hasil Pertandingan	6
5.1. Bot Minimax vs. Manusia	6
5.2. Bot Local Search vs. Manusia	8
5.3. Bot Minimax vs. Bot Local Search	9
5.4. Bot Minimax vs. Bot Genetic Algorithm	10
5.5. Bot Local Search vs. Bot Genetic Algorithm	11
VII. Pembagian Tugas	12

I. Adjacency Strategy Game



Gambar 1. Adjacency Strategy Game

Adjacency Strategy Game adalah sebuah permainan dimana pemain perlu menempatkan marka (O atau X) pada papan permainan dengan tujuan untuk memperoleh marka sebanyak mungkin pada akhir permainan. Pada awal permainan, terdapat masing-masing 4 buah marka X di bagian kiri bawah dan 4 buah marka O di bagian kanan atas. Urutan permainan akan berlangsung secara bergiliran, dimulai dari pemain X. Ketika pemain meletakkan sebuah marka, jika pada salah satu bagian atas, bawah, kiri, atau kanan kotak telah diisi oleh marka musuh, maka marka tersebut akan berubah menjadi marka pemain yang meletakkan marka barunya. Pemenang permainan ini adalah pemain yang mendapatkan jumlah marka terbanyak sebelum permainan selesai. Permainan akan selesai jika seluruh kotak pada papan permainan telah terisi penuh atau jika jumlah ronde yang tersisa telah habis.



Gambar 2. Startup Window

Sebelum permainan dimulai, pemain perlu menentukan nama pemain pertama, nama pemain kedua, dan jumlah ronde permainan. Untuk mengubah jenis pemain dari permainan ini dapat langsung diubah melalui fungsi BotFight.java yang telah dibuat.

II. Objective Function

Objective function yang digunakan untuk konfigurasi lengkap adalah banyak marka O dikurangi banyak marka X di papan permainan, atau secara formal:

$$Obj(State) = N_o - N_x$$

dengan keterangan sebagai berikut:

N_θ : Banyak petak yang berisi simbol θ

O : Simbol pemain saat ini

X : Simbol pemain musuh

Objective function tersebut berdasarkan aturan pemilihan pemenang pada permainan, yaitu pemain yang memiliki marka terbanyak di papan. Dengan fungsi objektif tersebut, *agent* akan menang jika mencapai state dengan nilai fungsi objektif positif. Namun pada kasus konfigurasi tidak lengkap, *objective function* yang digunakan adalah

$$IncObj(State) = L_o - L_x + C \times \max(|U_o - U_x| - M, 0)$$

dengan keterangan sebagai berikut:

L_θ : Banyak petak yang berisi simbol θ dan tidak memiliki petak adjacent yang kosong

U_θ : Banyak petak yang berisi simbol θ dan memiliki setidaknya satu petak adjacent yang kosong

M : Jumlah ronde tersisa

C : Konstan yang bernilai 1 jika $U_o > U_x$ atau bernilai -1 jika $U_o < U_x$

Alasan nilai petak yang tidak memiliki petak *adjacent* yang kosong (L_θ) dipisahkan dari petak yang memiliki setidaknya satu petak *adjacent* yang kosong (U_θ) adalah dikarenakan petak yang tidak memiliki petak *adjacent* yang kosong tidak mungkin berubah lagi, sehingga nilai pada petak tersebut akan statis hingga permainan selesai. Akibatnya, petak yang tidak memiliki petak *adjacent* yang kosong (L_θ) memiliki bobot objektif yang lebih besar dibandingkan petak yang

memiliki petak *adjacent* yang kosong (U_0). Kemudian, dipilih sebuah nilai konstan C yang hanya dapat bernilai 1 atau -1. Nilai ini digunakan agar hasil selisih U_0 dan U_x akan memiliki dampak terhadap nilai akhir fungsi objektif. Kemudian, jumlah ronde yang tersisa menjadi salah satu parameter dari fungsi objektif dengan asumsi jika jumlah ronde yang tersisa masih banyak, maka bobot dari selisih petak yang memiliki petak *adjacent* yang kosong belum terlalu tinggi, karena masih dapat diubah pada ronde selanjutnya.

III. Minimax Algorithm dan Alpha Beta Pruning

Algoritma Minimax dapat digunakan dalam mencari solusi optimal pada permainan Adjacency Strategy Game, dengan mempertimbangkan aksi yang memberi keuntungan maksimal pada giliran pemain, dan juga mempertimbangkan strategi lawan yang meminimalkan keuntungan pemain pada gilirannya. Algoritma Minimax secara intuitif dapat diimplementasikan dengan metode rekursi, dengan memanggil fungsi yang memilih aksi dengan keuntungan maksimal pada giliran pemain, kemudian memanggil fungsi yang memilih aksi meminimalkan yang menyimulasikan aksi lawan secara bergantian, hingga tercapai state terminal (akhir dari permainan).

Implementasi Minimax tidak akan efisien dalam lingkungan yang memiliki kandidat aksi banyak (branching factor besar) dan membutuhkan langkah yang banyak untuk mencapai tujuan akhir (depth besar). Salah satu upaya dalam mempercepat algoritma Minimax adalah dengan alpha-beta pruning, dengan “memangkas” simpul-simpul pencarian yang tidak akan memberi hasil lebih baik (tidak mungkin terpilih) oleh simpul induknya. Variasi ini tidak mengubah hasil dari algoritma Minimax dan hanya mengurangi pencarian yang dilakukan, namun pada kasus terburuk akan membutuhkan waktu dan memori yang sama tanpa pruning.

Pada permainan ini, algoritma Minimax (dengan alpha-beta pruning) diimplementasikan dalam 3 fungsi utama, yakni decide, maximize, dan minimize.

Untuk mengambil aksi pada giliran bot, jalankan method decide yang akan menghasilkan aksi optimal pada giliran tertentu. Pada fungsi decide, akan di-generate semua aksi yang mungkin pada giliran tersebut, kemudian masing-masing akan dinilai pada fungsi minimize, aksi dengan nilai terbesar adalah aksi yang dipilih. minimize akan menyimulasikan lawan yang optimal, dengan memilih aksi dengan value yang memberi keuntungan paling sedikit bagi pemain, dan sebaliknya untuk maximize.

Bot memiliki nilai alpha dan beta sebagai kriteria pruning dalam pencarian. alpha merepresentasikan lower bound (keuntungan terkecil menurut pemain yang pasti didapatkan oleh pemain) dan beta merepresentasikan upper bound (keuntungan terbesar menurut pemain yang pasti didapatkan lawan). Jika pada giliran lawan (minimize) aksi yang dipilih (oleh lawan) menghasilkan keuntungan yang lebih kecil dari nilai alpha, node minimize tersebut di-prune karena terdapat aksi lain yang memungkinkan lawan berada pada state yang tidak memungkinkan mengambil aksi dengan nilai tersebut. Hal serupa dapat diimplementasikan untuk giliran pemain.

Bot yang memanfaatkan algoritma Minimax tidak feasible pada permainan yang masih menyisakan banyak giliran hingga akhir permainan. Kompleksitas menjalankan algoritma Minimax adalah $O(N!)$, dengan N adalah sisa giliran hingga akhir permainan. Penggunaan alpha-beta pruning dapat mengurangi jumlah langkah, namun untuk N besar tetap akan menjadi sangat lambat dan boros memori. Perubahan lain dapat dilakukan dengan membatasi kedalaman Minimax, namun hal itu dapat menyebabkan algoritma tidak complete dan menghasilkan hasil yang belum tentu optimal. Meskipun demikian, hal tersebut dapat memberi hasil lebih optimal dibandingkan local search atau bahkan memilih aksi secara acak.

IV. Local Search

Local search menyelesaikan persoalan permainan Adjacency Strategy Game dengan mengiterasi seluruh neighbor state permainan dan memilih yang menghasilkan nilai objektif terbesar. Iterasi diimplementasikan dengan memeriksa “complete state” papan permainan yang merupakan keadaan papan setelah pemain memasang markanya pada kotak sudut kiri atas, kemudian mengiterasi satu per satu neighbor dengan memindahkan marka tersebut dengan urutan kiri ke kanan, atas ke bawah hingga kotak sudut kanan bawah.

Penggunaan algoritma local search feasible diimplementasikan dalam permainan Adjacency Strategy Game. Pada setiap giliran, bot hanya perlu memeriksa N^2 state (dengan N adalah dimensi papan permainan).

V. Genetic Algorithm

Pencarian dengan genetic algorithm dapat dilakukan tiap kali method move() dijalankan. Metode yang digunakan adalah genetic minimax algorithm yang diajukan Hong et al. (2001). Algoritma ini bekerja dengan mencari langkah-langkah yang paling mungkin dimainkan oleh agent yang rasional. Langkah untuk melakukan pencarian adalah

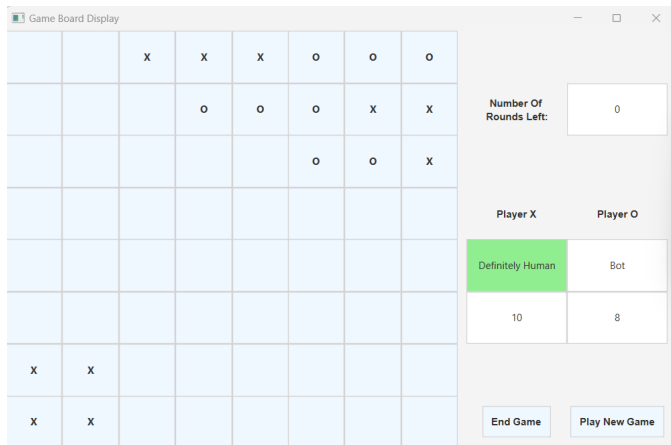
1. Definisikan batas waktu, nilai N (populasi yang dipertahankan), banyak generasi, rasio crossover dan rasio mutasi.

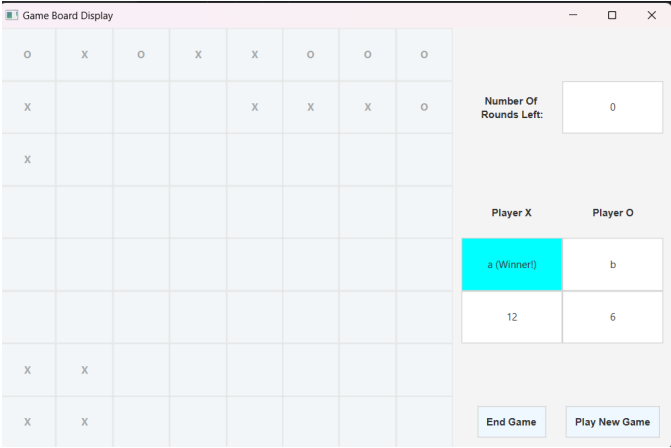
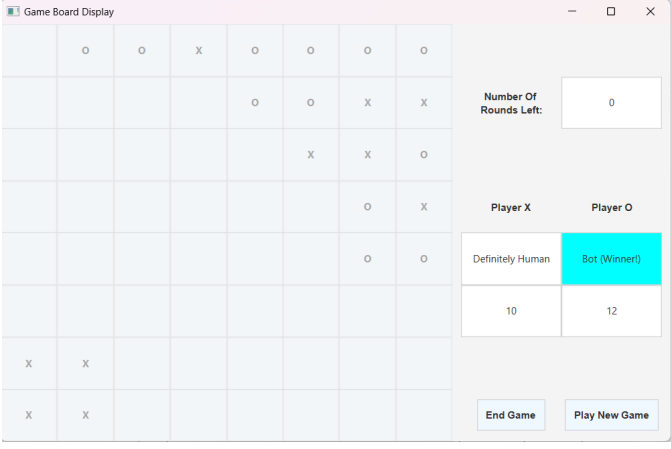
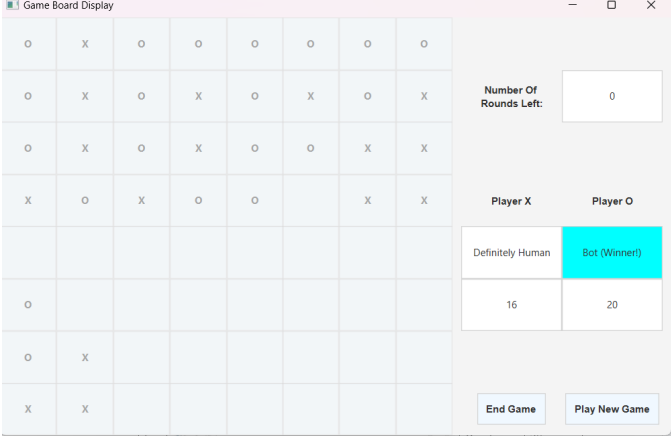
2. Buat sebuah reservation tree untuk menyimpan state dengan root yang memiliki nilai ekivalen negatif tak hingga.
3. Bangkitkan N populasi secara acak dan masukkan ke dalam reservation tree sambil mengubah nilai tree menggunakan minimax.
4. Hitung nilai fitness dari tiap state, yaitu banyak ancestor yang memiliki nilai sama dengan nilai leaf.
5. Lakukan crossover pada populasi dengan rasio yang sudah ditentukan. Crossover dapat dilakukan dengan menggunakan partially mapped crossover (PMX) atau improved PMX (Koohestani, 2020) untuk menjaga keabsahan permutasi.
6. Lakukan mutasi pada sebagian populasi menggunakan swap mutation (menukar dua bagian state).
7. Hapus kromosom duplikat.
8. Masukkan child ke reservation tree sambil mengubah nilai tree menggunakan minimax.
9. Hitung nilai fitness tiap child.
10. Pilih maksimal N individu yang memiliki nilai fitness terbaik.
11. Lakukan langkah 5-10 sampai batas yang sudah ditentukan.


Nilai fitness individu dihitung berdasarkan banyak ancestor yang memiliki nilai sama dengan nilai leaf. Semakin banyak ancestor yang memiliki nilai sama, maka semakin baik individu tersebut. Algoritma ini memiliki beberapa kelebihan, yaitu dapat menemukan langkah terbaik dalam permainan papan yang kompleks, dapat beradaptasi dengan perubahan lingkungan, dan dapat menghindari pencarian yang tidak perlu.

VI. Hasil Pertandingan

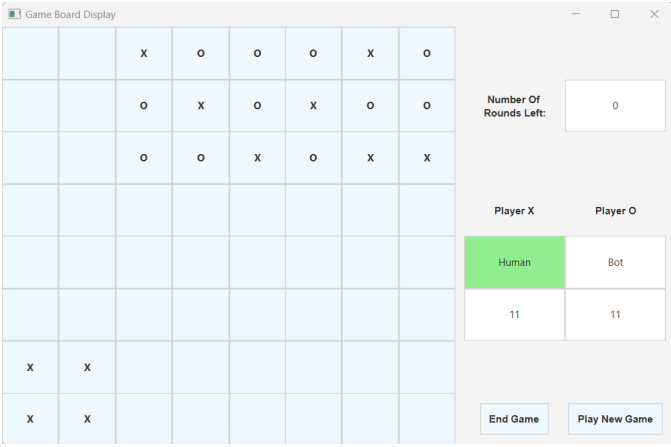
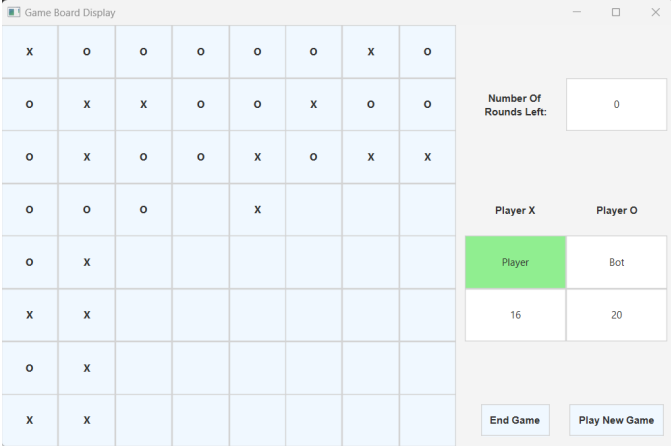
5.1. Bot Minimax vs. Manusia


No	Screenshot	Keterangan	
1		Ronde	5
		Pemenang	Manusia

2		Ronde	5
		Pemenang	Manusia
3		Ronde	7
		Pemenang	Bot Minimax
4		Ronde	14
		Pemenang	Bot Minimax

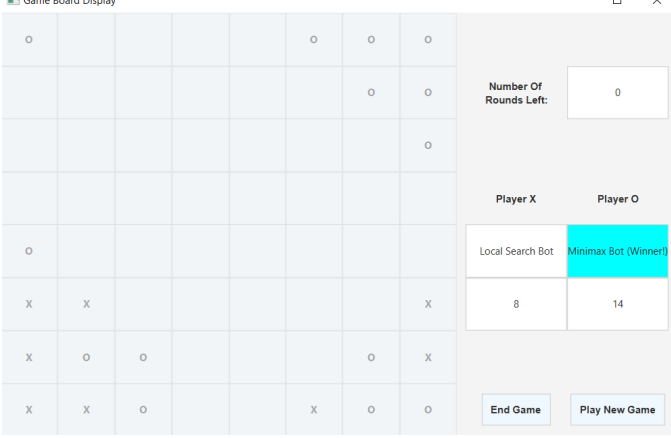
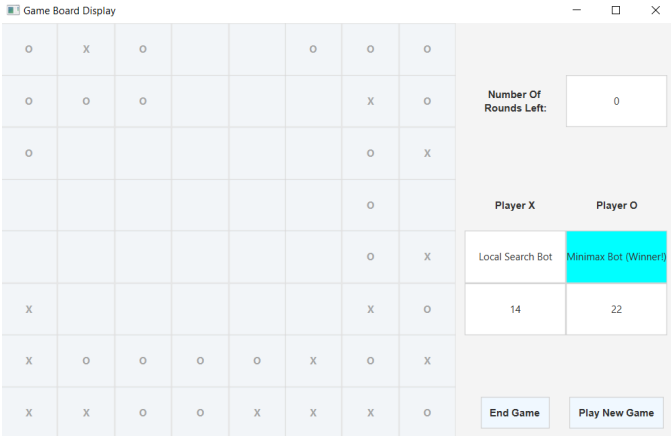
5		Ronde	28
		Pemenang	Bot Minimax

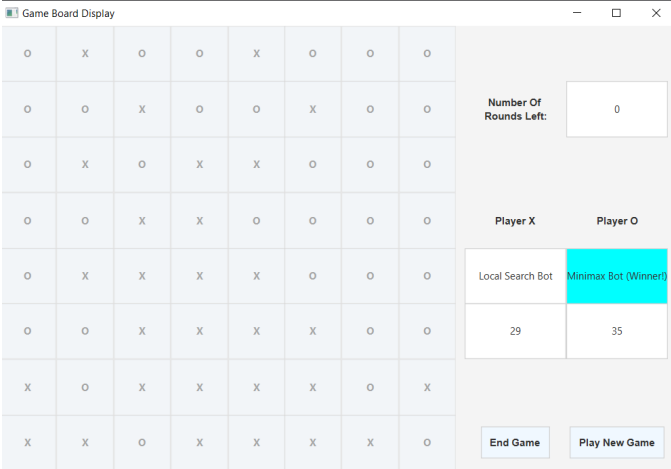
5.2. Bot Local Search vs. Manusia

No	Screenshot	Keterangan	
1		Ronde	7
		Pemenang	Seri (Bot & Manusia)
2		Ronde	14
		Pemenang	Bot Local Search

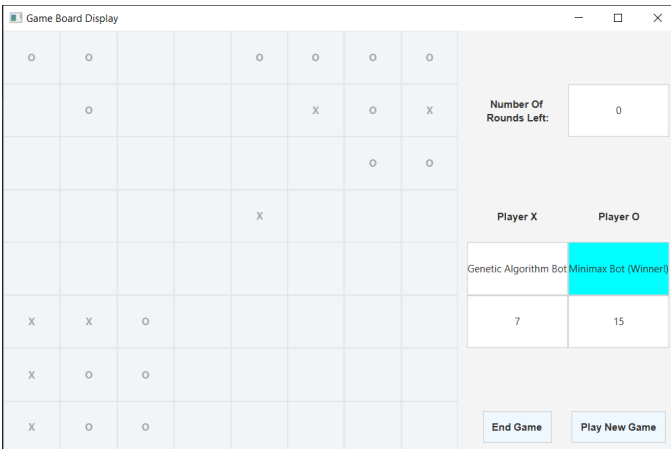
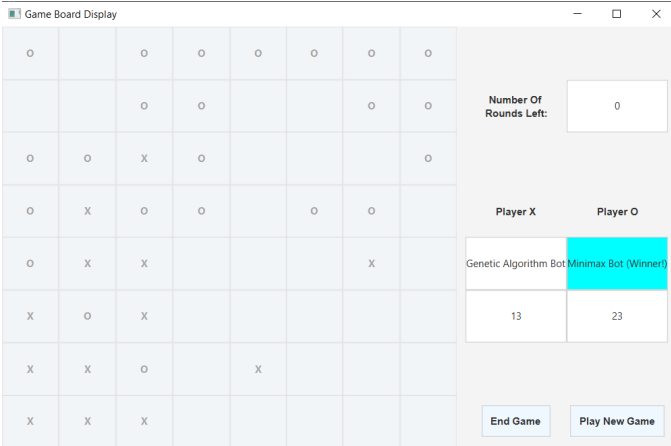
3		Ronde	28
		Pemenang	Bot Local Search

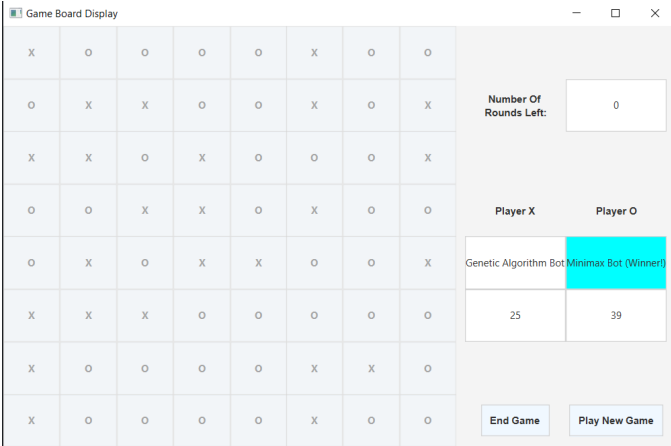
5.3. Bot Minimax vs. Bot Local Search

No	Screenshot	Keterangan	
1		Ronde	7
		Pemenang	Bot Minimax
2		Ronde	14
		Pemenang	Bot Minimax

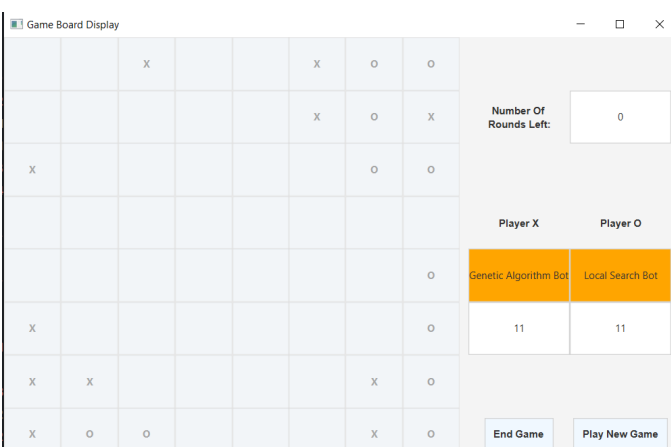
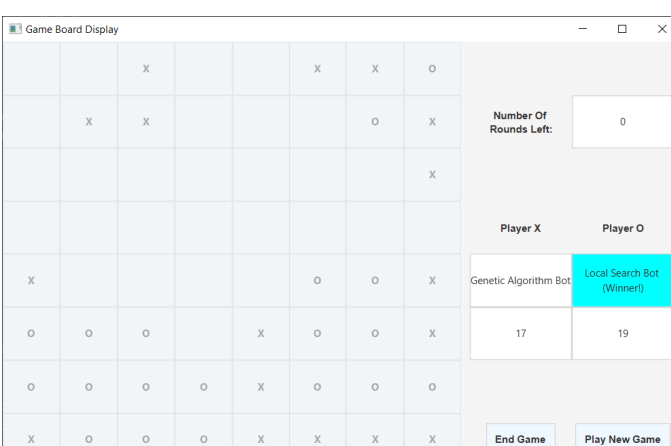
3		Ronde	28
		Pemenang	Bot Minimax

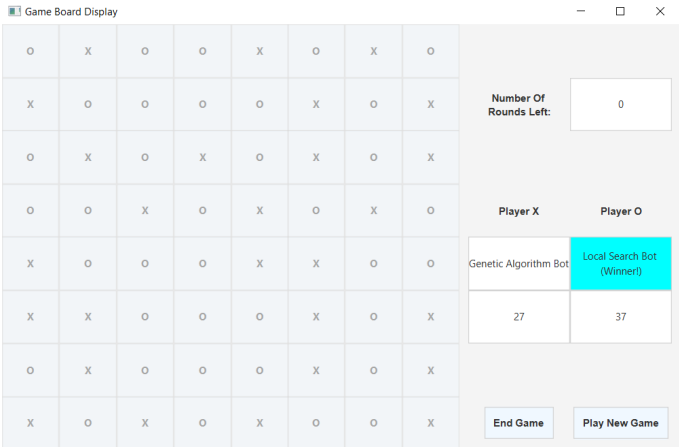
5.4. Bot Minimax vs. Bot Genetic Algorithm

No	Screenshot	Keterangan	
1		Ronde	7
		Pemenang	Bot Minimax
2		Ronde	14
		Pemenang	Bot Minimax

3		Ronde	28
		Pemenang	Bot Minimax

5.5. Bot Local Search vs. Bot Genetic Algorithm

No	Screenshot	Keterangan	
1		Ronde	7
		Pemenang	Seri (Bot Local Search & Bot Genetic Algorithm)
2		Ronde	14
		Pemenang	Bot Local Search

3		Ronde	28
		Pemenang	Bot Local Search

VII. Pembagian Tugas

NIM	Nama	Pembagian Tugas
13521092	Frankie Huang	Dokumentasi
13521137	Michael Utama	Genetic Algorithm
13521138	Johann Christian Kandani	Local Search Algorithm, Minimax Algorithm
13521173	Dewana Gustavus Haraka Otang	Local Search Algorithm, Minimax Algorithm