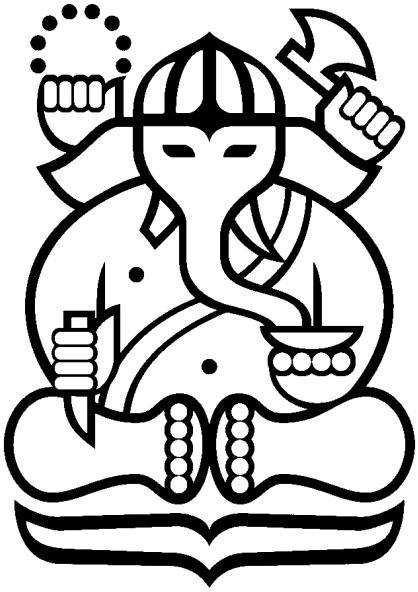
Laporan Tugas Besar 1 IF3170 Intelegensi Buatan
Semester I 2023-2024

ALGORITMA MINIMAX ALPHA BETA PRUNING, LOCAL SEARCH, DAN GENETIC ALGORITHM PADA ADJACENCY STRATEGY GAME

Dipersiapkan oleh :

Kelompok 17

Muhammad Equilibrie Fajria	13521047
Addin Munawwar Yusuf	13521085
Edia Zaki Naufal Ilman	13521141
Reza Pahlevi Ubaidillah	13521165



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

2023

Jawaban Soal

1. Objective Function

Fungsi objektif/utilitas yang kami gunakan untuk mengevaluasi nilai state dari papan adalah sebagai berikut.

$$g(n) = N\_B - N\_M$$

N_B = jumlah petak yang dimiliki bot
N_M = jumlah petak yang dimiliki musuh

Total jumlah petak pada permainan adalah 64 petak, sehingga nilai objektif maksimal yang dapat dicapai adalah 64, sedangkan nilai minimumnya adalah -64. Pada persoalan ini, bot perlu melakukan maksimasi terhadap fungsi objektif untuk memenangkan permainan. Fungsi objektif ini cukup *straightforward*, karena sejalan dengan *win condition* dari game.

Untuk membuat bot yang lebih pintar, maka kita bisa menambahkan heuristik tambahan yang dapat mengarahkan bot ke arah yang lebih baik. Heuristik tersebut harus admissible, tidak boleh overestimate fungsi objektif utama (1 poin setiap penambahan petak), sehingga nilai heuristik harus berada pada range (0, 1) — konsepnya mirip dengan A* search.

Berdasarkan beberapa kali percobaan bermain, kami mendapati bahwa strategi yang baik adalah **meletakkan simbol di area sudut papan** permainan. Hal ini karena petak di area sudut lebih sulit diambil alih. Maka, ada heuristik tambahan yaitu:

$$h(petak\_n) = abs(length((i/3{,}5, j/3{,}5) - (1,1)))/16$$

i = index baris i dari petak yang dimiliki
j = index baris j dari petak yang dimiliki

Algoritma diatas memetakan kembali koordinat dari [0, 7] menjadi [-1, 1], kemudian dicari panjangnya dan dimutlakkan agar mendapatkan panjang radius dari tengah papan ke index tersebut (tengah papan adalah koordinat (3.5, 3.5)). Nilai maksimum dari h(n) diatas adalah $\sqrt{2}$ (didapat dari length vector (1,1)). Agar nilai tetap berada pada range (0, 1), maka h(n) kita bagi lagi dengan $\sqrt{2}$. Dalam tabel, nilai heuristiknya adalah sebagai berikut.

Heuristic							
0.063	0.054	0.048	0.045	0.045	0.048	0.054	0.063
0.054	0.045	0.037	0.032	0.032	0.037	0.045	0.054
0.048	0.037	0.027	0.020	0.020	0.027	0.037	0.048
0.045	0.032	0.020	0.009	0.009	0.020	0.032	0.045
0.045	0.032	0.020	0.009	0.009	0.020	0.032	0.045
0.048	0.037	0.027	0.020	0.020	0.027	0.037	0.048
0.054	0.045	0.037	0.032	0.032	0.037	0.045	0.054
0.063	0.054	0.048	0.045	0.045	0.048	0.054	0.063

Hasil akhir dari fungsi objektif yang kita miliki adalah:

```
g(n) = N_B - N_M
h(n) {
   res = 0
   for (i,j : index_petak_dimiliki) {
      res += abs(length((i/3,5,j/3.5) - (1,1)))
   }
   ⟶ res
}

f(n) = g(n) + h(n)
```

N_B = jumlah petak yang dimiliki bot
N_M = jumlah petak yang dimiliki musuh
i = index baris i dari petak yang dimiliki
j = index baris j dari petak yang dimiliki

Untuk setiap petak yang dimiliki, tambahkan objective function dengan nilai heuristik untuk petak pada indeks tersebut.

2. Algoritma *Minimax* dan *Alpha Beta Pruning*

Bot akan menerapkan algoritma Minimax Alpha Beta Pruning untuk menentukan aksi yang optimal (atau suboptimal). Sebelum dapat menentukan aksi awal yang akan dilakukan, bot perlu mengeksplorasi game tree. Initial statenya adalah kondisi permainan pada turn pertama bot, jika bot mulai duluan berarti 4 X di pojok kiri bawah dan 4 O di pojok kanan atas, jika player mulai duluan berarti kondisi awal permainan yang sudah dipengaruhi sebuah aksi dari player.

```
function Minimax-Decision(state) returns an action
  α <- -∞
  β <- ∞

  for each a in ACTIONS do
    temp <- Min-Value(Move(state, a), α, β)
    if α <= temp then
      α <- temp
  end
  return optimalAction
```

```
function Min-Value(state, α, β) returns a utility value
  if Terminal-Test(state) then
    return Utility(state), []
  else
    for each a in ACTIONS do
      temp <- Max-Value(Move(state, a), α, β)
      if β >= temp then
        β <- temp
      if β<=α then return α/*pruning*/
    end
    return β
```

```
function Max-Value(state, α, β) returns a utility value
  if Terminal-Test(state) then
    return Utility(state), []
  else
    for each a in ACTIONS do
      temp <- Min-Value(Move(state, a), α, β)
      if α <= temp then
        α <- temp
      if β<=α then return β /*pruning*/
    end
    return α
```

Sesuai algoritma, bot akan menjelajahi state-state yang ada dengan cara mirip DFS dengan depth maksimum sebesar $2 \times \text{Number-of-Round}$ jika bot mulai duluan dan $2 \times \text{Number-of-Round} - 1$ jika player mulai duluan. Berarti setelah mendapat *initial state*-nya, bot akan mengambil *successor state* untuk kemudian dijelajahi dan akan dilakukan rekursif hingga mencapai basisnya yaitu *terminal state*. Bedanya dengan DFS, bot akan selang-seling memanggil Min-Value dan Max-Value sampai terminal state, Min-Value akan dipanggil jika state saat itu adalah turn player, Max-Value akan dipanggil jika state saat itu adalah turn bot. Terlebih dari itu, bot juga akan memangkas "*pruning*" state yang tidak perlu dikunjungi dengan cara membandingkan nilai α state ini dengan β parent jika state ini adalah *turn* bot (MAX) dan membandingkan nilai β state ini dengan α parent jika state ini adalah *turn* player (MIN). Pada *terminal state*, *utility function* akan memberikan *value* pada *state.* Begitulah seterusnya sampai diperoleh aksi-aksi yang optimal untuk setiap state (yang tidak kena pangkas) di akhir Minimax-Decision.

Berbeda dengan algoritma lain, bot ini tidak memiliki fungsi heuristik sehingga hanya akan memanfaatkan fungsi utilitas sepenuhnya.

3. Algoritma Local Search

Local search adalah algoritma pencarian solusi dengan cara peningkatan kualitas state solusi secara iteratif. Peningkatan kualitas state dilakukan dengan membuat perubahan inkremental menuju state tetangga terhadap mengevaluasi nilainya terhadap fungsi objektif. Local search tidak menjamin solusi yang optimum secara global, akan tetapi sering digunakan untuk masalah-masalah yang pencariannya secara menyeluruh tidak praktikal.

Local search memiliki perbedaan dengan classical search. Pada classical search, eksplorasi dilakukan secara traversal dari state awal yang belum *complete configuration* (konfigurasi state bukan merupakan konfigurasi state solusi yang valid) ke state goal secara inkremental. Path to goal pada pencarian relevan, dan biasanya merupakan solusi dari classical search.

Sementara itu, local search bermulai dari state *complete configuration* yang random, atau dipilih dengan algoritma tertentu. Setelah itu, berdasarkan teknik local search yang

dipilih, pencarian akan dilakukan dari current state melalui state tetangganya sampai dicapai state yang optimum, baik secara lokal maupun global (tergantung teknik yang dipakai). Path to goal pada persoalan local search tidak relevan.

Persoalan adjacency strategy game adalah persoalan yang path to goal-nya relevan dan tidak bisa dianalisis secara langsung *complete configuration*-nya, sehingga kurang cocok digunakan untuk pencarian solusi pada permainan ini. Akan tetapi kita bisa menggunakan salah satu karakteristiknya, yaitu teknik pemilihan next statenya.

Sebagai contoh pada persoalan adjacency-strategy game ini, kita akan menggunakan pendekatan *hill climbing with sideway moves*. Pendekatan ini akan memilih *state* selanjutnya yang memiliki nilai *state* lebih besar atau sama dengan *state* saat ini. Sebenarnya, pada persoalan ini, setiap gerakan pasti akan memperbaiki state saat ini, karena penilaian pasti akan menjadi lebih baik ketika bot menambahkan marka baru di papan. Sehingga, sebenarnya pendekatan ini sama saja dengan *steepest-ascent hill climbing.*

Secara garis besar, *pseudocode* dari program adalah sebagai berikut.

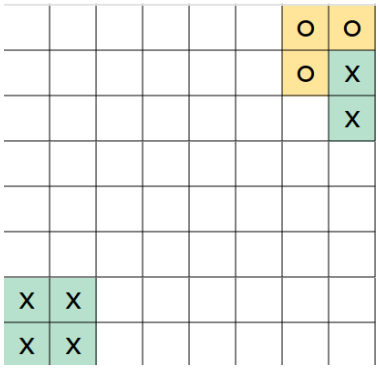
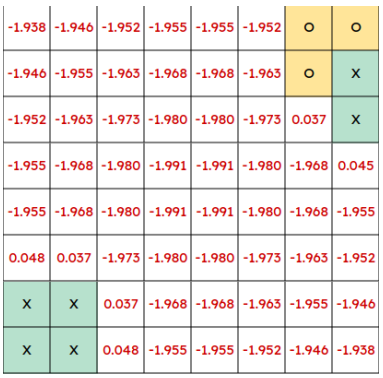
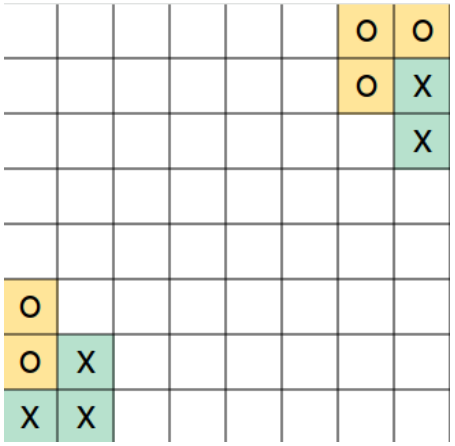
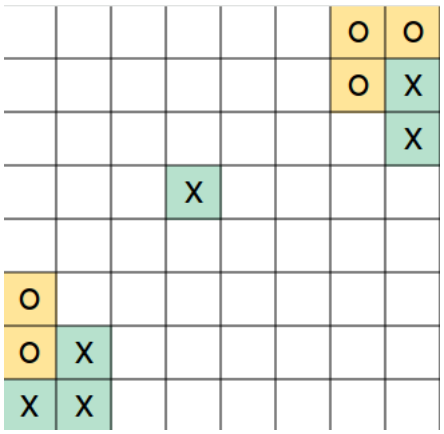
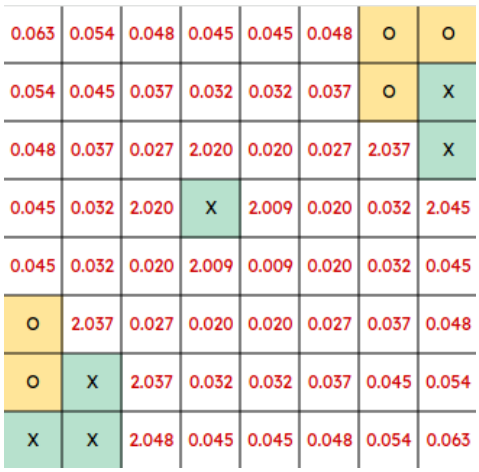
```
public int[] move(Board board) {
        int[][] validMoves ← board.getValidMoves();
        int[][] bestMoves ← new int[validMoves.length][2];
        int i ← -1;
        float bestScore ← -999;

        for (int[] move : validMoves) {
        Board newBoard ← new Board(board);
        newBoard.addMove(move[0], move[1], Tile.BOT);

        float score ← objectiveFunction(newBoard);
        if (score >= bestScore) {
        if (score = bestScore) {
                i++;
                bestMoves[i] ← move;
        } else {
                i ← 0;
```

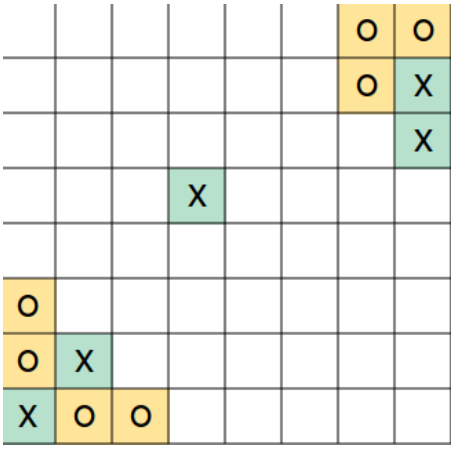
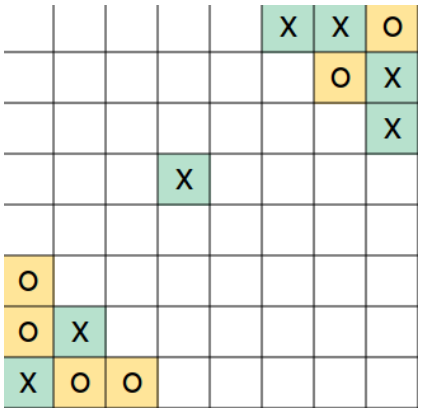
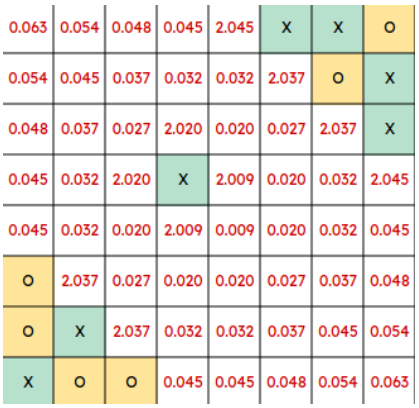
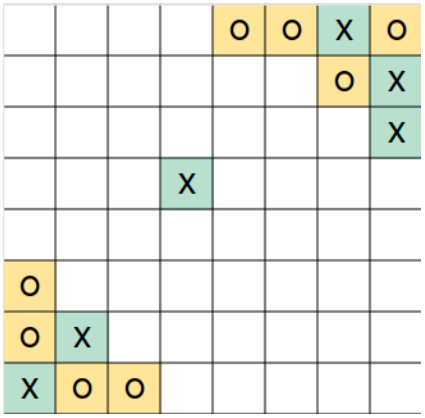
```
                    bestScore ← score;
                    bestMoves ← new int[validMoves.length][2];
                    bestMoves[i] ← move;
            }
            }
            }
            if (i > 0) {
            → bestMoves[random.nextInt(i)];
            }
            if (i == 0) {
            → bestMoves[0];
            }
             → null;
}
```

Move dipilih berdasarkan suksesor tertinggi selanjutnya untuk mendapatkan kenaikan nilai objektif tertinggi. Pendekatan ini sangat mirip dengan greedy best first pada classical search. Pada permainan ini, move apapun yang kita buat **pasti memperbaiki** fungsi objektif state kita, sehingga kita tidak perlu melakukan pengecekan lagi jika nilai state lebih buruk.

Contoh permainan dengan 3 ronde sebagai ilustrasi adalah sebagai berikut. Player adalah X dan Bot adalah O.

Player(1) Move Player	Bot(1) Hasil Evaluasi Fungsi Objektif

	
	Pilih Highest Successor
	
Player(2) Move Player	Bot(2) Hasil Evaluasi Fungsi Objektif
	

	Pilih Highest Successor
	
Player(3) Move Player	Bot(3) Hasil Evaluasi Fungsi Objektif
	
	Pilih Highest Successor
	

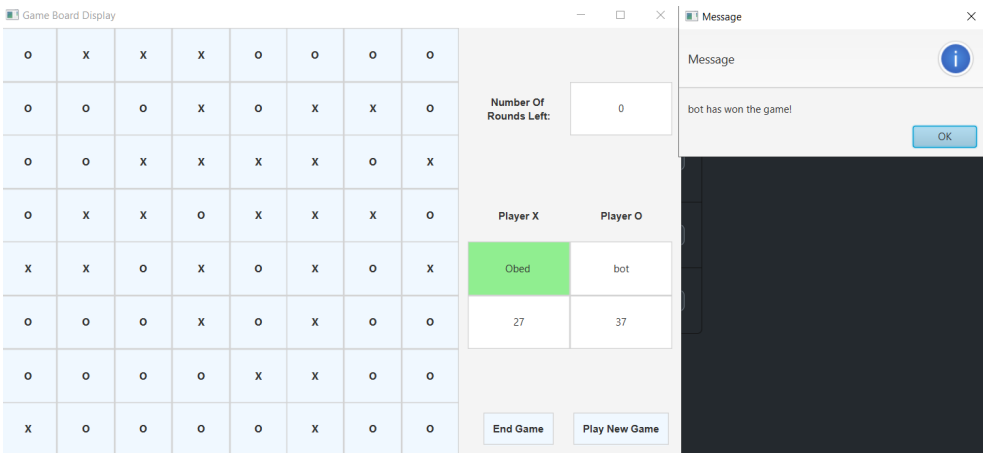
Hasil Akhir: Bot menang dengan keunggulan 2 poin

4. Algoritma Genetic

Pada algoritma genetic, bot pertama-tama melakukan inisialisasi populasi awal. *Complete configuration* pada algoritma ini adalah move berikutnya yang valid dari bot diwakili dengan koordinat x dan koordinat y. Pada bot ini terdapat empat individu yang menjadi populasi awal. Bot lalu menghitung nilai fitness dari setiap individu. Setelah itu, bot melakukan selection menggunakan *roulette wheel*. *Parents* yang terpilih akan dilakukan *crossover* pada koordinat y nya. Jika tidak bisa (karena move tidak valid) maka dilakukan *cross-over* pada koordinat x. Jika tidak bisa juga maka tidak dilakukan *cross-over*. Setelah itu, bot melakukan *mutation* pada koordinat y dari setiap anak. Percobaan dilakukan paling banyak tiga kali dan jika dari ketiga hasil mutasi individu tersebut tidak ada move yang valid, maka tidak dilakukan *mutation*. Terakhir bot memilih anak dengan nilai fitness terbesar lalu mengembalikannya sebagai next move dari bot.

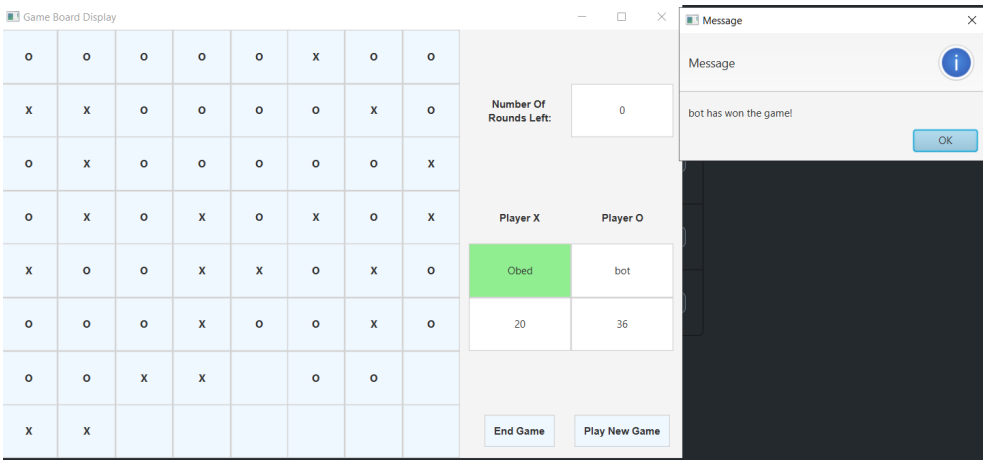
5. Penjelasan hasil pertandingan yang dilakukan antara:

Bot *minimax* vs. manusia (sebanyak 5 kali)

Sampai Board Penuh

Poin Bot : 37 Poin Player : 27

Bot menang dengan perbedaan 10 poin

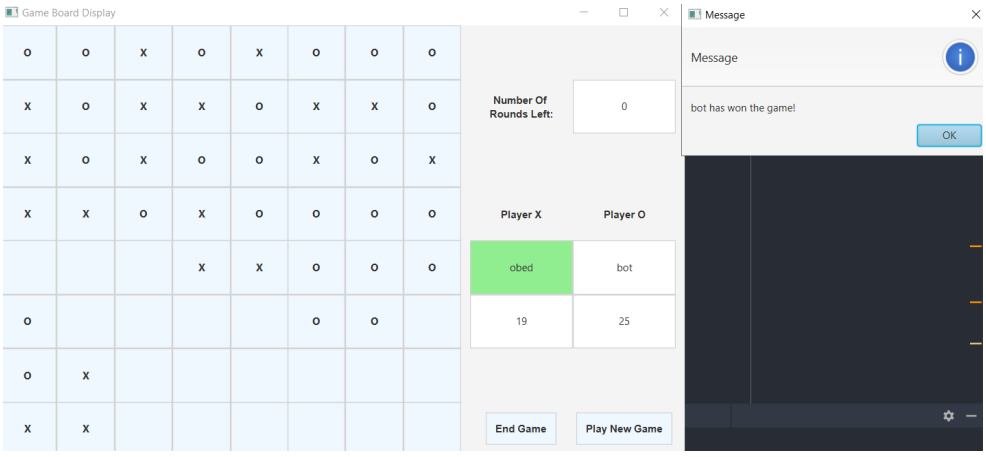
24 Gerakan



Poin Bot : 36
Poin Player : 20

Bot menang dengan perbedaan 16 poin

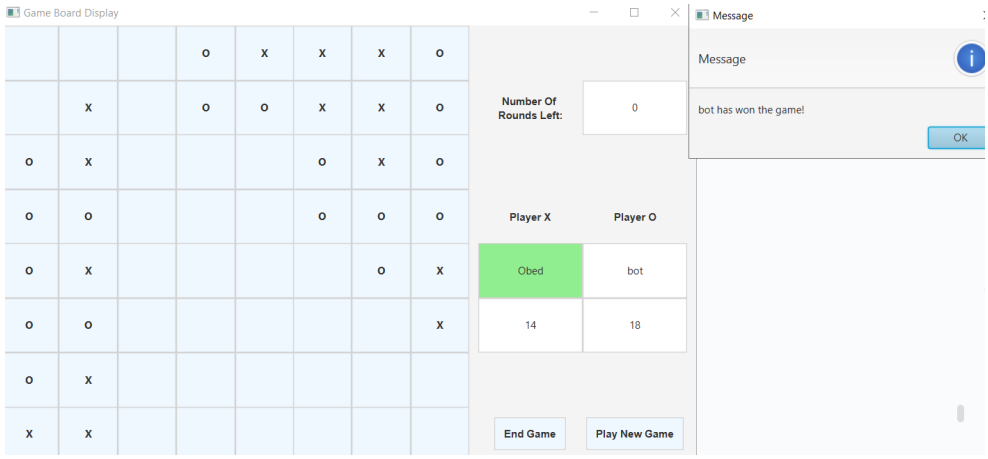
18 Gerakan



Poin Bot : 25
Poin Player : 19

Bot menang dengan perbedaan 6 poin

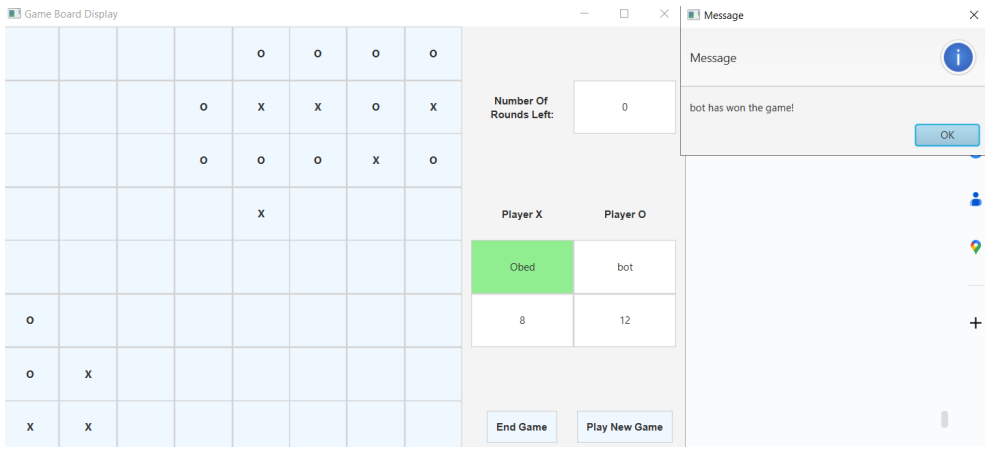
12 Gerakan



Poin Bot : 18
Poin Player : 14

Bot menang dengan perbedaan 4 poin

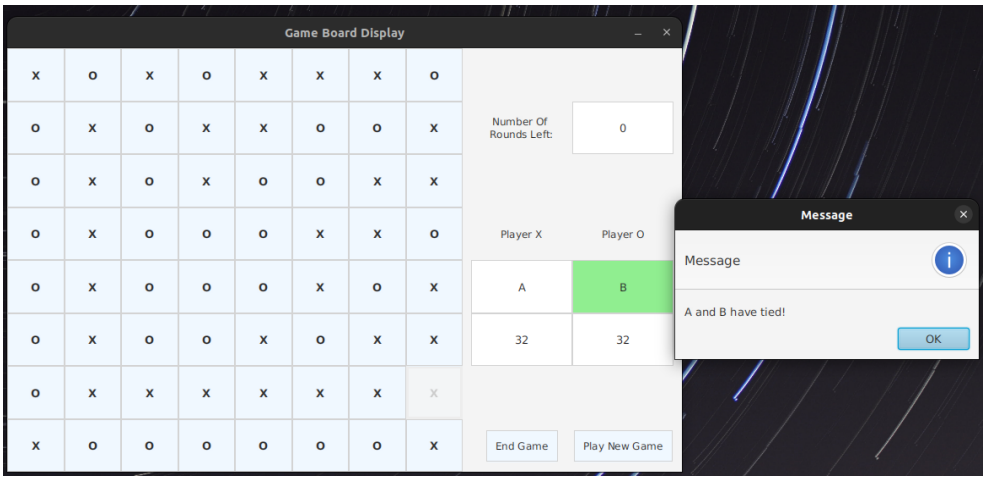
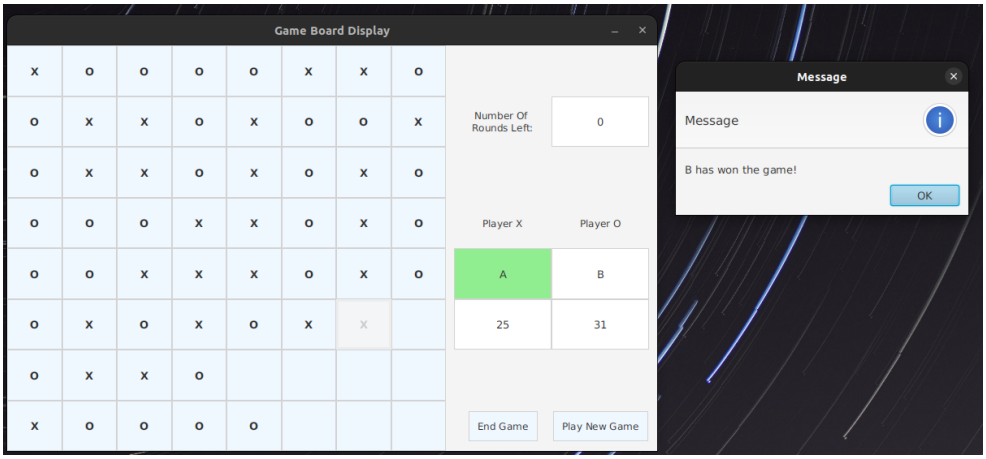
6 Gerakan

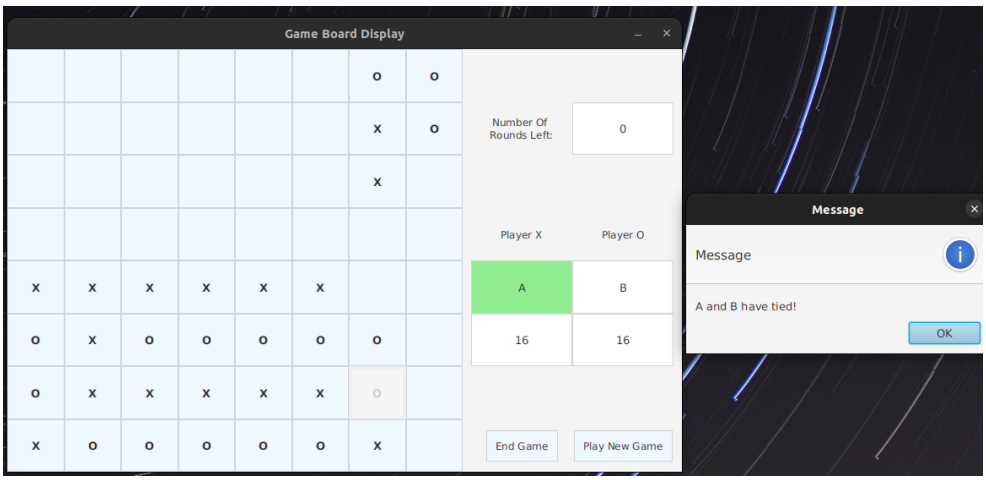


Poin Bot : 12
Poin Player : 8

Bot menang dengan perbedaan 4 poin

Jika mengimplementasikan, Bot *local search* vs. manusia (sebanyak 3 kali)

Sampai Board Penuh
 Poin Bot : 32 Poin Player : 32 Bot dan Player Seri
24 Gerakan
 Poin Bot : 31 Poin Player : 25 Bot Menang dengan perbedaan 6 Poin
12 Gerakan

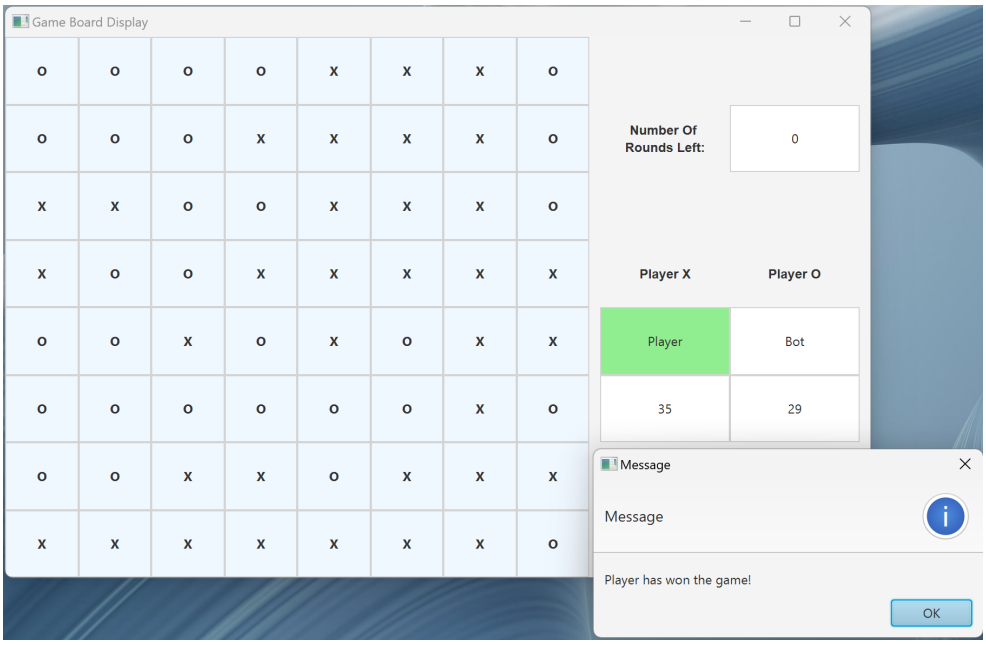


Poin Bot : 16
Poin Player : 16

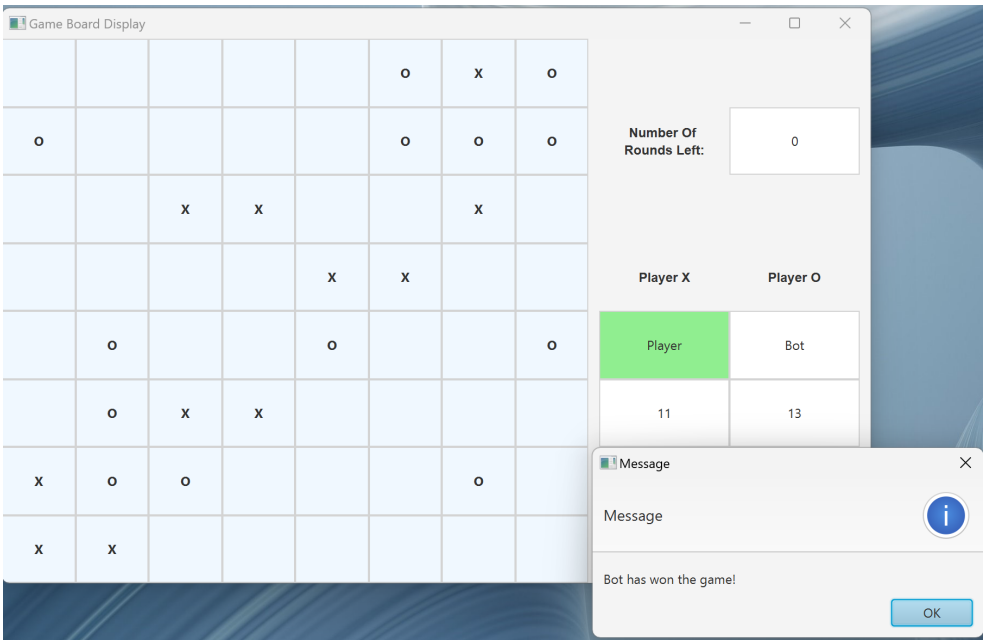
Bot dan Player Seri

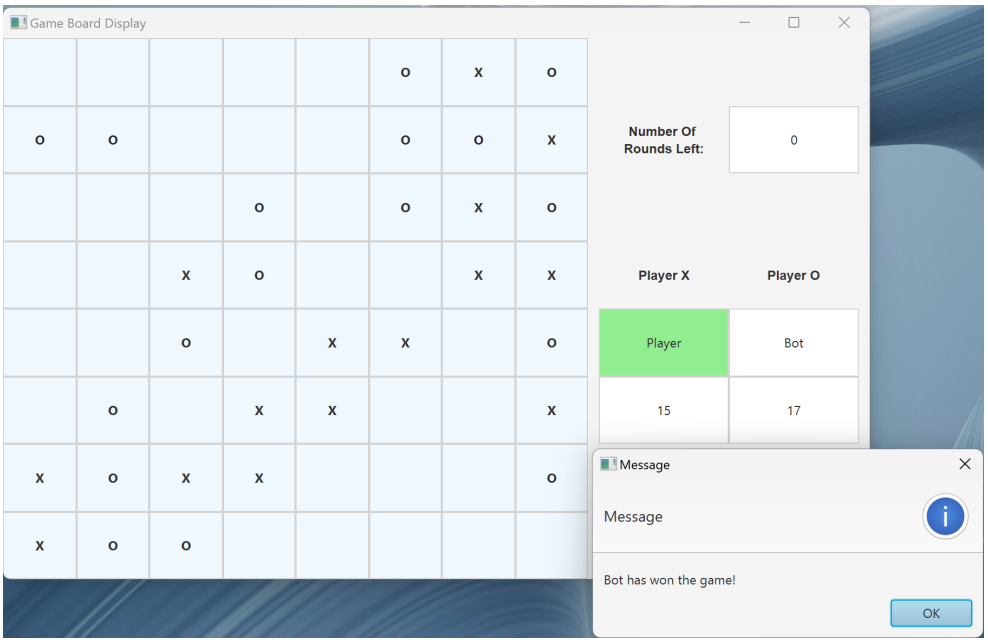
Jika mengimplementasikan, Bot *genetic* vs. manusia (sebanyak 3 kali)

Sampai Board Penuh



Poin Bot : 29
Poin Player : 35

Player Menang dengan perbedaan 6 poin
8 Gerakan
 Poin Bot : 13 Poin Player : 11 Bot Menang dengan perbedaan 2 Poin
12 Gerakan

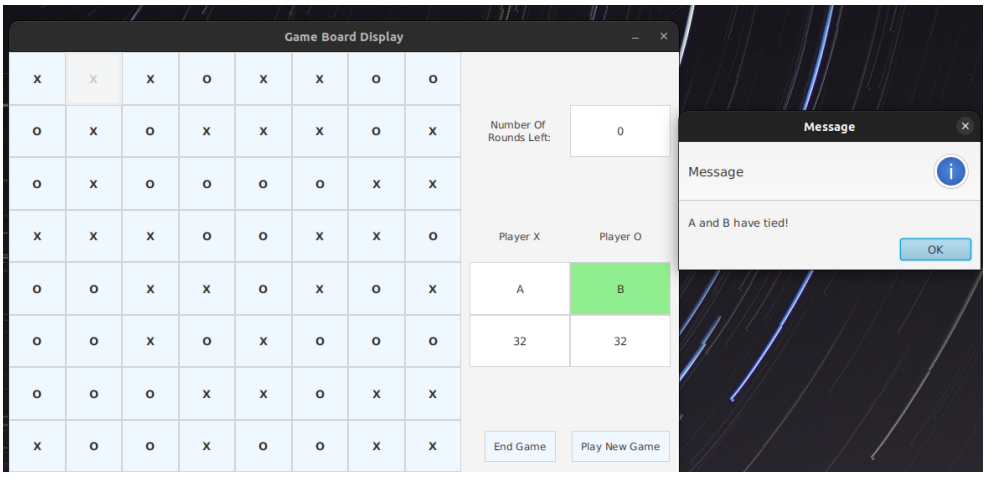


Poin Bot : 17
Poin Player : 15

Bot menang dengan perbedaan 2 poin

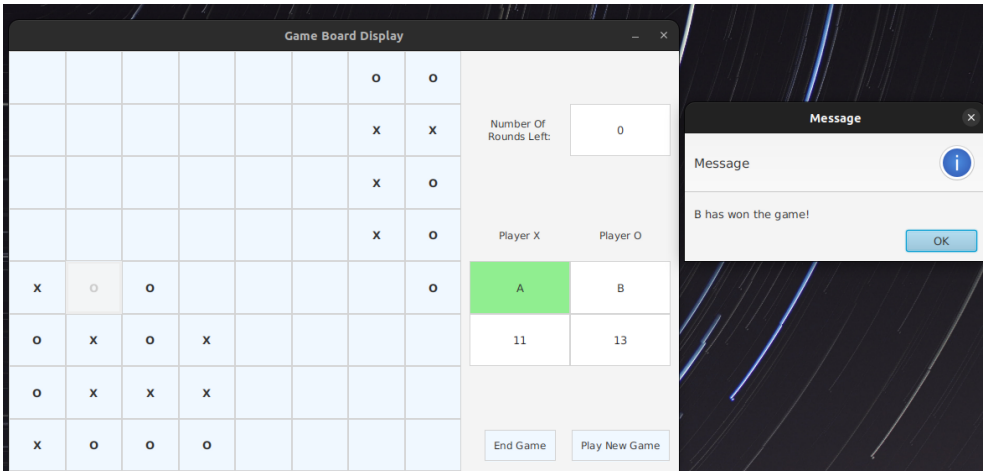
Jika mengimplementasikan, Bot *minimax* vs bot *local search* (sebanyak 3 kali)

Sampai Board Penuh



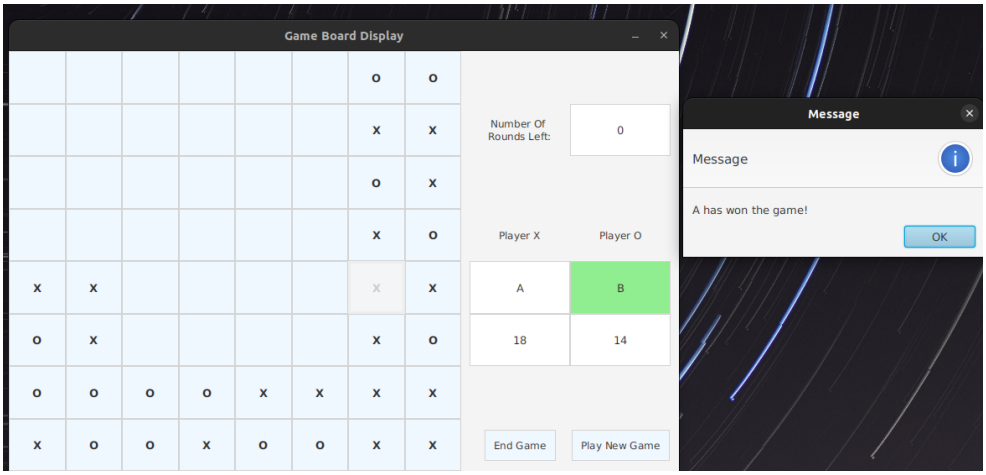
Poin Local Search : 32
Poin Minimax : 32

Bot Local Search dan Minimax Imbang

8 Gerakan


Poin Local Search : 13
Poin Minimax : 11

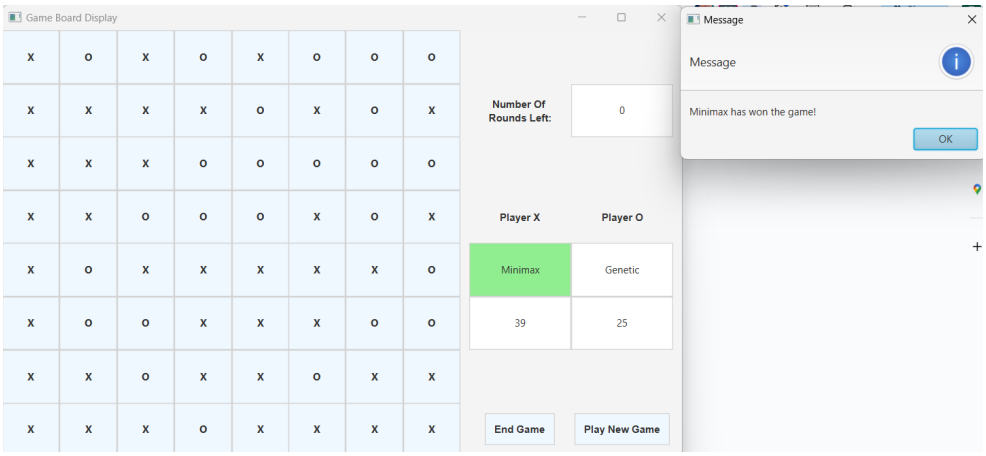
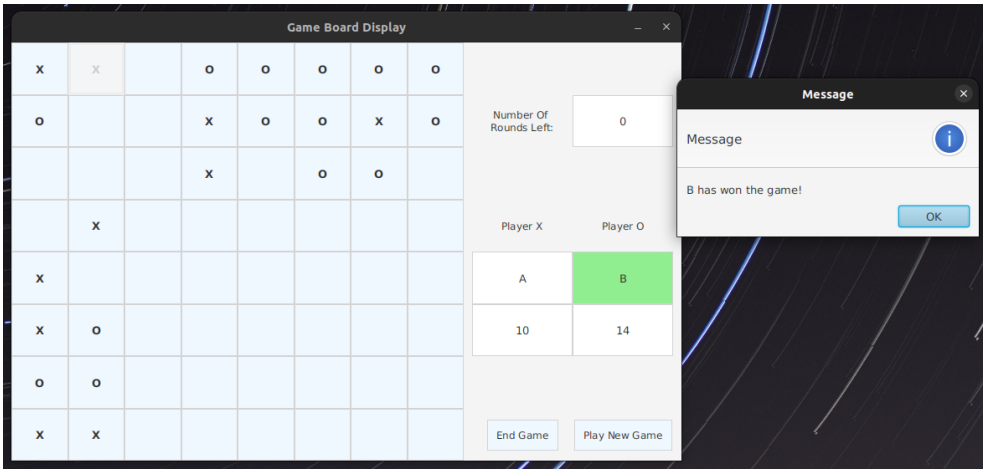
Bot Local Search Menang dengan perbedaan 2 Poin

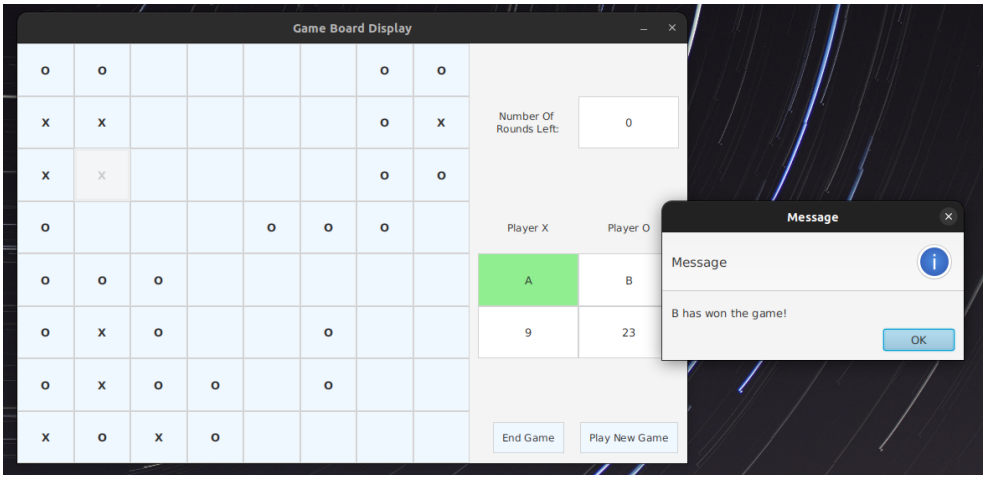
12 Gerakan


Poin Local Search : 14
Poin Minimax : 18

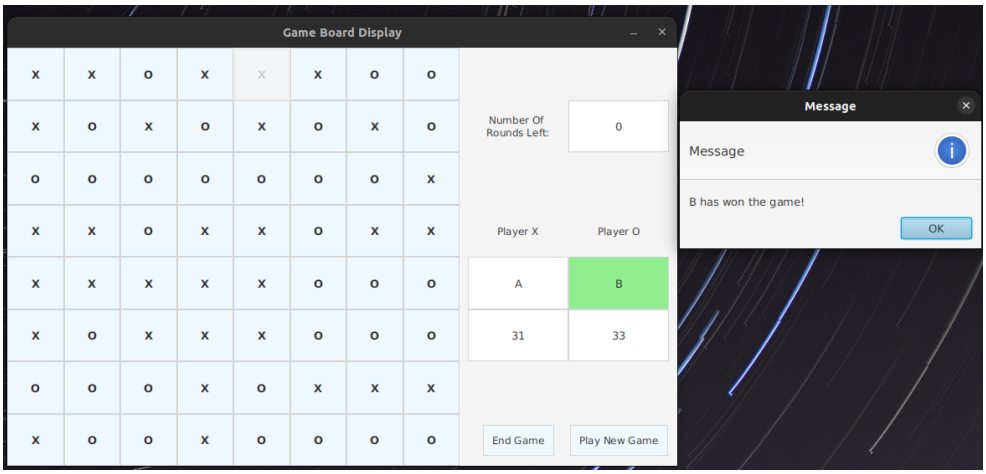
Bot Minimax Menang dengan perbedaan 18 Poin

Jika mengimplementasikan, Bot *minimax* vs bot *genetic algorithm* (sebanyak 3 kali)

Sampai Board Penuh

Poin Genetic : 39 Poin Minimax : 25 Bot Genetic Menang dengan perbedaan 14 Poin
8 Gerakan

Poin Genetic : 10 Poin Minimax : 14 Bot Minimax Menang dengan perbedaan 4 Poin

12 Gerakan
 Poin Genetic : 9 Poin Minimax : 23 Bot Minimax Menang dengan perbedaan 14 Poin

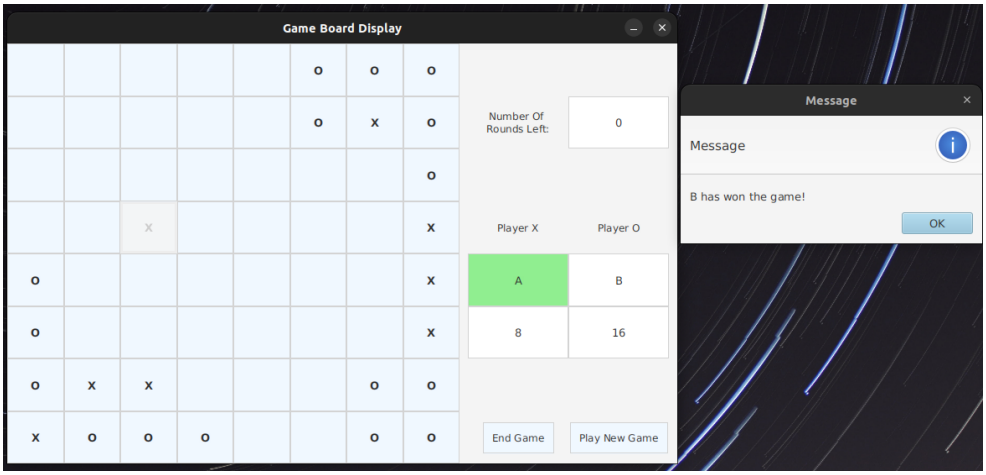
Jika mengimplementasikan, Bot *local search* vs bot *genetic algorithm* (sebanyak 3 kali)

Sampai Board Penuh


Poin Local Search : 33
Poin Player : 31

Bot Local Search Menang dengan perbedaan 2 poin

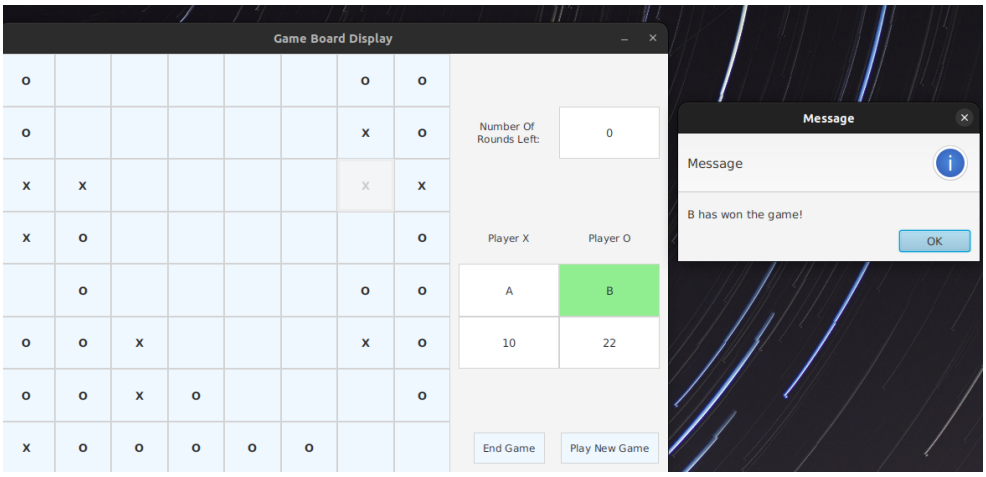
8 Gerakan



Poin Bot : 16
Poin Player : 8

Bot Local Search Menang dengan perbedaan 8 poin

12 Gerakan



Poin Bot : 22
Poin Player : 10

Bot Local Search Menang dengan perbedaan 12 poin

Lampiran

Kontribusi Anggota

Muhammad Equilibrie Fajria (13521047)	Addin Munawwar Yusuf (13521085)	Edia Zaki Naufal Ilman (13521141)	Reza Pahlevi Ubaidillah (13521165)
Mengimplementasikan genetic algorithm search	Mengimplementasikan algoritma local search	Mengimplementasikan algoritma minimax alpha beta pruning.	Mengimplementasikan algoritma minimax alpha beta pruning.
Menyusun dokumen laporan tugas besar 1	Menyusun dokumen laporan tugas besar 1	Menyusun dokumen laporan tugas besar 1	Menyusun dokumen laporan tugas besar 1
	Membuat heuristik fungsi objektif		

Link Repository: https://github.com/Ezaaan/adversarial-adjacency-strategy-game