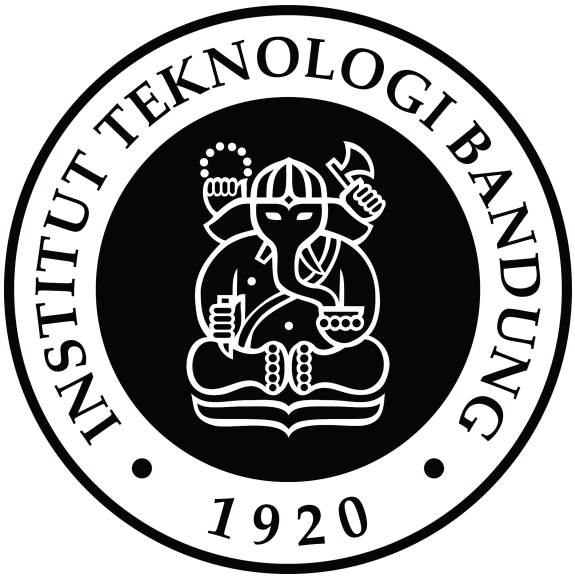
Tugas Besar 1:

Minimax Algorithm and Alpha Beta Pruning in Adjacency Strategy Game

IF3170 Inteligensi Buatan



Fatih Nararya Rashadyfa Ilhamsyah	13521060
Akbar Maulana Ridho	13521093
Noel Christoffel Simbolon	13521096
Bintang Dwi Marthen	13521144

A. *Objective Function*

Misalkan jumlah marka pemain adalah u, jumlah marka lawan adalah e, ukuran sisi kotak permainan adalah n, dan $m = u - e$ sebagai selisih jumlah marka pemain dan lawan.

Berdasarkan aturan dari permainan, fungsi objektif (f) adalah m agar pemain memiliki lebih banyak marka daripada lawan.

Fakta bahwa $u + e = n$ di akhir permainan menunjukkan dengan jelas bahwa permainan ini merupakan *zero-sum game* karena kesuksesan pemain akan selalu menghasilkan kegagalan lawan. Hal ini tampak jelas, tetapi penting untuk diingat untuk analisis yang akan dilakukan.

B. *Minimax Search Algorithm* dengan *Alpha-Beta Pruning*

Algoritma *minimax alpha beta pruning* (selanjutnya akan disebut *minimaxab* untuk keringkasan) adalah sebuah ekspansi dari algoritma *minimax*. Pada algoritma *minimax*, setiap *state* dari permainan adalah sebuah simpul pada *game tree* yang akan dievaluasi oleh algoritma tersebut secara rekursif sampai simpul daun, yakni sampai permainan selesai. Tiap simpul tersebut sebenarnya memiliki nilai yang inheren, yaitu fungsi objektif yang dihitung berdasarkan *state* permainan. Tetapi, pada algoritma ini simpul akan mendapatkan nilai dari anak-anaknya, yaitu *state* permainan yang dapat dicapai dari *state* awal melalui satu langkah.

Algoritma *minimax* dapat memperhitungkan strategi lawan dengan cara memperhitungkan aksi yang mungkin dilakukan lawan. Pada kedalaman genap (simpul akar dihitung sebagai kedalaman 0), algoritma akan bertindak sebagai *maximizing player* sehingga ia akan memilih simpul anak dengan nilai fungsi objektif tertinggi, lalu meng-*assign* nilai tersebut ke simpul yang sedang dievaluasi. Sementara pada kedalaman ganjil, algoritma akan bertindak sebagai *minimizing player* sehingga ia akan memilih simpul anak dengan nilai fungsi objektif tertinggi, lalu meng-*assign* nilai tersebut ke simpul yang sedang dievaluasi. Algoritma ini dapat digunakan pada permainan ini dengan fungsi objektif yang telah diberikan karena jelas salah satu pemain ingin memaksimalkannya, sementara pemain lawan ingin meminimalisasinya.

Tetapi, *minimax* adalah sebuah algoritma yang memiliki beban komputasi yang berat karena *game tree* sangat cepat untuk mengalami *combinatorial explosion* sehingga simpul yang perlu dievaluasi menjadi sangat banyak. Hal tersebut akan didemonstrasikan pada analisis dari *feasibility* algoritma ini. *Alpha-beta pruning* kemudian ditambahkan pada algoritma ini agar tidak semua *branch* atau anak simpul harus dievaluasi dengan melakukan *pruning* dari *branch* tersebut.

Pruning dilakukan sebagai berikut. Sebuah nilai *alpha* dan *beta* di-*pass* kepada fungsi rekursif yang digunakan untuk mengevaluasi simpul, dengan keduanya berurutan diberi nilai $-\infty$ dan ∞ pada pemanggilan pertama di simpul akar. Ketika sedang pada simpul *maximizing*, nilai dari *alpha* akan diperbarui tiap kali simpul anak dievaluasi dengan nilai simpul anak tersebut (diambil maksimum dari *alpha* dan nilai simpul anak tersebut). Sebaliknya, pada simpul *minimizing*, nilai dari *beta* akan diperbarui tiap kali simpul anak dievaluasi dengan nilai simpul anak tersebut (diambil minimum dari *beta* dan nilai simpul anak tersebut). Jika pada sebuah simpul yang dievaluasi nilai $beta \leq alpha$, maka evaluasi anak impul tersebut akan dihentikan

dan fungsi langsung mengembalikan nilai simpul tersebut yang telah dievaluasi. Untuk *maximizing player* ini adalah nilai paling besar dari simpul anak yang sudah dievaluasi dan untuk *minimizing player* ini adalah nilai paling kecil dari simpul anak yang sudah dievaluasi.

Contoh dari proses pencarian dengan *minimaxab* akan ditunjukkan sebagai berikut dengan posisi papan sudah hampir terisi semua dan sedang giliran pemain biru.

ID : A

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

u	e
31	30

Demi kemudahan demonstrasi, anak dan cucu dari *state-state* yang akan ada dari *state* tersebut langsung dipresentasikan di sini beserta ID dari mereka. Marka yang baru ditambahkan pada giliran sekarang ditandai kuning dan marka sekitar yang berubah akibat marka yang ditambahkan tersebut ditandai hijau.

ID : B

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

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

u	e
34	28

ID : C

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

u	e
32	31

ID : D

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

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

u	e	m
36	28	8

ID : E

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

u	e
33	30

ID : F

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

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

u	e	m
36	28	8

ID : G

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

u	e
34	28

ID : H

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

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

u	e
32	31

ID : I

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

u	e	m
36	28	8

ID : J

X	O	O	X	X	X	O	O

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

u	e
33	30

ID : K

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

u	e	m
36	28	8

ID : L

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

u	e
35	27

ID : M

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

u	e
33	30

ID : N

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

u	e	m
36	28	8

ID : O

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

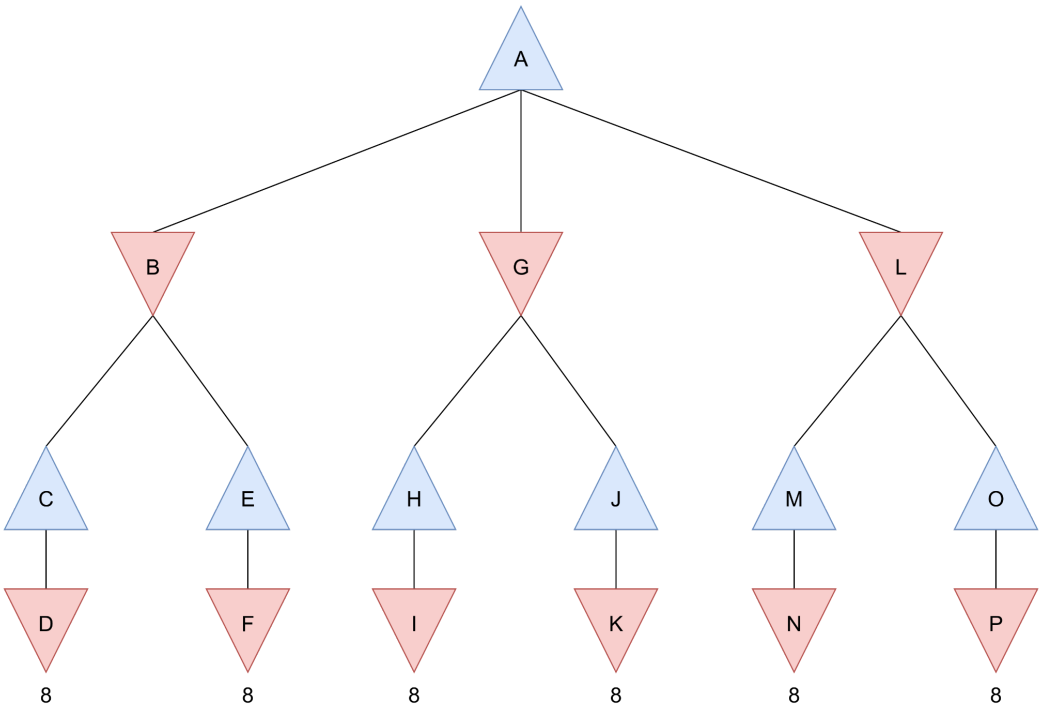
u	e
33	30

ID : P

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

u	e	m
36	28	8

Game tree dengan simpul A sebagai akar adalah sebagai berikut.



minimaxab berawal dari simpul 0 dengan nilai $\alpha = -\infty$, $\beta = \infty$. Nilai ini kemudian diturunkan ke pemanggilan fungsi yang mengevaluasi simpul B, lalu diturunkan juga ke simpul C, sampai akhirnya ke simpul D. Di sini, simpul D yang memiliki nilai 8 membuat nilai $\beta = 8$ (karena ia adalah simpul *minimizing*) di dalam fungsi evaluasi simpul D dan mengembalikan nilai 8 kepada simpul C (bukan mengembalikan β tetapi mengembalikan nilai simpul itu sendiri).

Kemudian, simpul C turut mengembalikan nilai tersebut sebagai nilai simpul itu kepada B karena D adalah anak tunggal, sehingga tidak ada anak yang perlu dievaluasi lagi. Di dalam fungsi evaluasi dari simpul C, $\alpha = 8$ (karena ia adalah simpul *maximizing*).

Lalu, simpul B menerima bahwa simpul C memiliki nilai 8. Sebagai simpul *minimizing*, fungsi evaluasi B menjadi memiliki $\beta = 8$. Proses yang terjadi pada simpul C dan D juga terjadi persis kepada simpul E dan F. Sampai akhirnya simpul B mengembalikan 8 sebagai nilai simpul tersebut kepada simpul A.

Simpul A sekarang memiliki $\alpha = 8$. Kemudian ia melanjutkan mengevaluasi anak selanjutnya yaitu simpul G, tetapi kali ini dengan $\alpha = 8$, $\beta = \infty$. Proses yang terjadi pada simpul C dan D serta E dan F persis terjadi lagi pada simpul H dan I. Akhirnya G menerima nilai simpul H sebesar 8, lalu mengubah β menjadi bernilai 8. Karena $\beta \leq \alpha$, maka G akan berhenti

mengevaluasi anaknya dan langsung mengembalikan nilai terendah yang ia terima sejauh itu, yaitu 8 dari simpul H (dan nilai satu-satunya).

Kenapa $\beta \leq \alpha$ membuat sebuah simpul berhenti mengevaluasi anak-anaknya dan langsung keluar? Karena berapapun nilai anaknya selanjutnya, bahkan jika ia ∞ atau $-\infty$, nilai tersebut tidak akan diambil oleh simpul orangtuanya. Jika nilai simpul anak selanjutnya $\geq \beta$, maka simpul G sebagai *minimizing player* tidak akan mengambil nilai tersebut. Jika nilai simpul anak selanjutnya $\leq \alpha$, maka nilai tersebut bisa jadi diambil oleh simpul G dan dikembalikan ke simpul orangtuanya (simpul A). Tetapi nilai tersebut tidak akan digunakan oleh simpul A karena yang diambil pastilah nilai α yang didapat oleh "saudara" dari simpul G yang telah dievaluasi terlebih dahulu.

Akhirnya simpul G mengembalikan nilai 8 kepada simpul A, sehingga $\alpha = 8$, tidak berubah. Nilai ini diberikan kepada fungsi yang mengevaluasi simpul L dan sama persis seperti simpul G, M dan N akan dievaluasi, lalu simpul L akan langsung mengembalikan nilai ke simpul A karena $\beta \leq \alpha$.

Sebagai catatan, *state* awal yang diambil sebagai simpul A atau simpul akar benar-benar dipilih secara acak dengan hanya dua pertimbangan :

1. Hanya sedikit kotak kosong yang masih tersisa agar demonstrasi *minimaxab* ini *practical* untuk dijabarkan.
2. m pada simpul A dibuat setipis mungkin agar *outcome* yang dihasilkan bervariasi.

Tidak ada ekspektasi sama sekali bahwa skor dari permainan akan berpusat pada nilai yang sama, yaitu 8. Perlu penelitian yang lebih dalam untuk mengetahui apakah ini hanyalah sebuah kebetulan atau jika permainan ini ternyata memiliki sebuah invariansi tertentu di mana *outcome* dari sebuah *state* dengan kondisi tertentu hanya ada sedikit saja atau bahkan hanya ada satu. Jika kasus yang ada adalah yang kedua, dapat dibuat sebuah algoritma yang lebih rumit atau melakukan augmentasi algoritma yang sudah ada dengan heuristik tersebut untuk meningkatkan kemungkinan kemenangan dengan membuat properti tersebut terjadi pada sebuah *instance* permainan.

Untuk mengukur *feasibility* dari *minimaxab*, ukur terlebih dahulu *feasibility* dari *minimax* naif. Misalkan jumlah kotak tanpa marka yang ada adalah p dengan $p = n^2 - u - e$ dan batas maksimum ronde adalah k. Pada sebuah giliran, terdapat p kotak yang dapat diisi. Setelah itu, akan terdapat $p - 1$ kotak yang dapat diisi. Jika diekstrapolasi, maka akan ada

$p \times (p-1) \times (p-2) \times \ldots \times 2 \times 1$. Karena awalnya terdapat 56 kotak kosong, maka total simpul yang ada pada $game\ tree$ permainan ini adalah 56!.

Pada setiap giliran salah satu anak simpul dipilih, dan pada giliran selanjutnya, *minimax* berjalan dari simpul yang dipilih tersebut. Selain itu, evaluasi akan berhenti pada kedalaman di mana permainan mencapai batas ronde maksimum yang telah diatur. Formulasi matematis hal tersebut adalah sebuah fungsi $g(t,\ k)$ yang memberikan jumlah simpul yang dievaluasi untuk giliran ke-t (dimulai dari 0 demi kerapian fungsi) dan batas ronde permainan k, sebagai berikut.

$$g(t,\ k)\ =\ \frac{(56-t)!}{k!}$$

Maka jika strategi *minimax* melakukan pencarian sampai ke simpul daun pada setiap giliran, jumlah total simpul yang perlu dievaluasi sampai permainan selesai untuk $k\ =\ 28$, nilai tertinggi yang mungkin.

$$\sum_{t=0}^{k} \begin{cases} 0, t\ \%\ 2 = 1 \\ g(t, 28), t\ \%\ 2 = 0 \end{cases}$$

Banyaknya jumlah dari simpul yang harus dievaluasi tersebut diilustrasikan oleh nilai dari $g(0,\ 28)\ \sim 2.33e45$. Sebagai perbandingan, terdapat 10^{22} - 10^{24} jumlah bintang di alam semesta menurut estimasi ESA[1]. Hal ini menunjukkan bagaimana strategi $minimax$ sangat tidak efisien untuk memainkan permainan ini.

Salah satu pendekatan yang dapat dilakukan untuk membuat *minimax* lebih *tractable* adalah untuk tidak mengevaluasi permainan sampai akhir permainan di setiap giliran, tetapi hanya mengevaluasi permainan sampai batas kedalaman tertentu(d), sehingga didapatkan fungsi baru $g(t,\ k,\ d)$.

$$g(t,\ k,\ d)\ =\ \frac{(56-t)!}{(max\{k,\ 56-t-d\})!}$$

$max\{k,\ 56\ -\ x\ -\ d\}$ digunakan karena terdapat dua batasan kedalaman. Pada awal permainan, *minimax* akan dibatasi kedalaman pencariannya oleh d sebab ia masih jauh dari ronde selesainya permainan, tetapi ketika mendekati akhir maka kedalaman pencarian akan mulai dibatasi oleh k.

Sayangnya, literatur yang meneliti secara *rigorous* efisiensi dari *alpha-beta pruning* dibandingkan *minimax* naif umumnya menggunakan permainan dengan *game tree* yang

[1] "How Many Stars Are There in the Universe?" n.d. ESA. ESA. Accessed September 11, 2023. https://www.esa.int/Science_Exploration/Space_Science/Herschel/How_many_stars_are_there_in_the_Universe

sederhana seperti *uniform tree*[2] ataupun *rug tree*[3], berbeda dengan permainan ini yang memiliki b konstan menurun dan nilai dari sebuah simpul (m) diskrit dengan jangkauan nilai yang jelas (lampiran B) tetapi tidak diketahui distribusi kemungkinannya.

Salah satu metode estimasi rata-rata *branching factor* yang ada telah dilakukan dan dapat dibaca pada lampiran A. Tetapi, karena hasil estimasi tersebut menghasilkan jumlah simpul yang jauh dengan nilai aktual maka estimasi berdasar hasil ini tidak akan memberi hasil yang berkualitas. Metode lain yang dapat dilakukan adalah melalui data empiris – metode yang digunakan untuk mendapatkan *branching factor* dari catur[4] – tetapi hal tersebut jelas berada di luar lingkup tugas kecil ini. Metode lain yang dapat digunakan adalah perhitungan berdasarkan *distinct states* dan *frame* dari permainan ini – sebuah metode yang telah digunakan untuk mengestimasi *branching factor* dari permainan Atari[5] – tetapi hal tersebut juga berada di luar lingkup tugas kecil ini.

Maka, untuk menghitung berapa jumlah simpul yang akan dikunjungi, pendekatan yang digunakan adalah menggunakan sebuah besaran bernama *average pruning rate* (r) yang sesuai namanya, adalah rata-rata jumlah *branch* yang di-*prune* dalam tiap kedalaman *game tree*. Jika diintegrasikan kepada g akan memberikan fungsi berikut.

$$g(t,\ k,\ d,\ r)\ = \frac{(56-t)! \times (1-r)^{56-t-max\{k,\ 56-t-d\}}}{(max\{k,\ 56-t-d\})!},\ r\ \in [0,\ 1)$$

Sebagai catatan, $56 - t - max\{k,\ 56 - t - d\}$ adalah jumlah kedalaman dari *game tree* yang dievaluasi oleh *minimaxab*.

Bagaimana hasil dari fungsi g untuk nilai $t,\ k,\ d,\ r$ berbeda-beda ditunjukkan pada grafik Desmos di tautan berikut, beserta persentase simpul yang tidak dievaluasi berkat *pruning* (R).

Didapatkan bahwa *alpha-beta pruning* menghasilkan perbedaan yang berarti pada jumlah simpul yang harus dievaluasi bahkan untuk nilai r yang moderat. Berapa sebenarnya nilai r untuk permainan ini akan dapat dianalisis secara empiris setelah *bot* diimplementasikan saat tugas besar, termasuk membandingkan pengaruh penggunaan berbagai macam heuristik terhadap

---

[2] Knuth, D. E., & Moore, R. W. (1975). An analysis of alpha-beta pruning. *Artificial Intelligence*, 6(4), 293–326. https://doi.org/10.1016/0004-3702(75)90019-3

[3] Darwish, N. M. (1983). A quantitative analysis of the alpha-beta pruning algorithm. *Artificial Intelligence*, 21(4), 405–433. https://doi.org/10.1016/s0004-3702(83)80020-4

[4] Kentdjb. 2019. "What Is the Average Number of Legal Moves per Turn?" Chess Stack Exchange. April 30, 2019. Accessed September 14, 2023. https://chess.stackexchange.com/a/24325

[5] Nelson, Mark J. 2021. "Estimates for the Branching Factors of Atari Games." *2021 IEEE Conference on Games (CoG)*, August. https://doi.org/10.1109/cog52621.2021.9619137.

urutan evaluasi simpul anak kepada nilai r. Oleh sebab itu pendekatan ini digunakan untuk menganalisis *feasibility* dari *minimaxab*.

Tetapi, sebenarnya kedalaman maksimum dari *game tree* yang dievaluasi oleh *minimaxab* menjadi faktor yang dapat dibilang lebih berpengaruh terhadap *feasibility* dari strategi ini karena ia dapat "memotong" suku dari faktorial pada pembilang fungsi g. Hal ini juga dapat disaksikan pada grafik Desmos yang diberikan, bagaimana mengurangi dan menambahkannya sebesar satu saja langsung berpengaruh pada *order of magnitude* dari jumlah simpul yang harus dievaluasi.

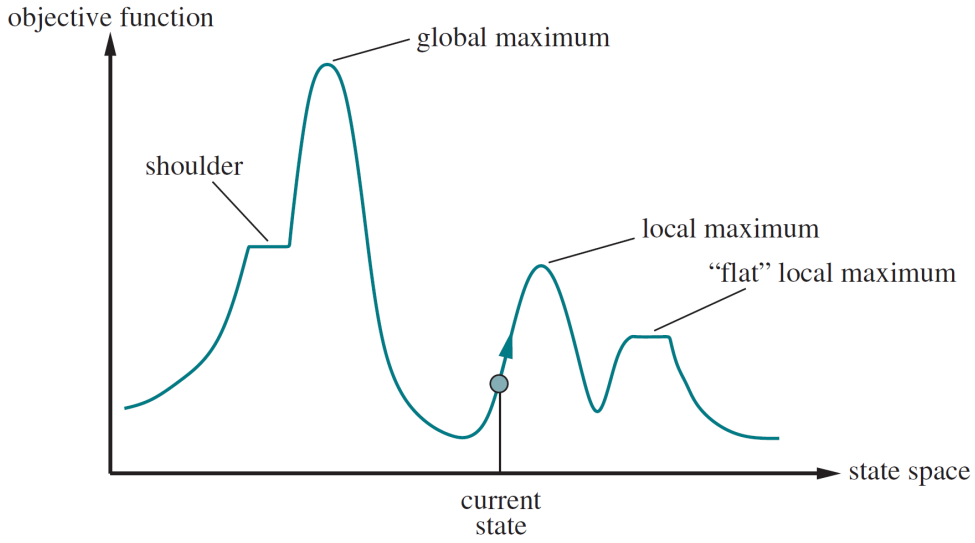
C. *Hill-Climbing Search Algorithm*

Algoritma *local search* yang kami implementasikan adalah algoritma *hill-climbing search*, bukan algoritma *simulated annealing*. Penggunaan algoritma *simulated annealing* akan menimbulkan *unnecessary complexity* dibandingkan penggunaan algoritma *hill-climbing*. *Unnecessary complexity* yang dimaksud akan dijelaskan lebih lanjut di bawah.

Pada poin-poin berikut, kami merincikan implementasi algoritma *hill-climbing* pada tugas kami.

1. *The set of candidate cells to move to* di-*generate* berdasarkan heuristik. *Candidate cells to move to* adalah semua *empty cells* yang merupakan *neighbor* dari *cell-cell* lawan. Itu adalah heuristik yang digunakan pada algoritma *hill-climbing*.
2. Jika tidak ada *possible cells to move to* yang memenuhi heuristik, maka *candidate cells to move to* adalah semua *possible cells to move to*, yaitu semua *empty cells* pada *game board*.
3. Proses '*searching for the best cell to move to*' dilakukan tiap *turn*, yaitu pencarian *cell* dengan nilai *objective function* terbesar dari *candidate cells to move to*.
4. *The best cell to move to* adalah *cell* yang dipilih sebagai langkah berikutnya.

Pada paragraf ini, akan dijelaskan mengapa kami tidak menggunakan algoritma *simulated annealing*. Algoritma *simulated annealing* didesain untuk dapat menyelesaikan salah satu masalah algoritma *hill climbing*. Masalah algoritma *hill-climbing* yang berusaha diselesaikan oleh algoritma *simulated annealing* adalah kondisi *stuck* pada *local maximum*. Hal ini dicapai dengan *randomness* dengan adanya *temperature*. Namun, usaha ini sia-sia karena tidak mungkin *stuck* pada *local maximum* pada masalah di tugas ini. Untuk ilustrasi lebih jelas, perhatikan gambar di bawah ini.



Jika permasalahan *local search* pada tugas ini dikaitkan dengan gambar di atas, *state space* merepresentasikan semua *possible cells to move to* pada satu *turn* tertentu. *Global maximum* adalah *candidate cell to move to* dengan nilai *objective function* terbesar pada satu *turn* tertentu. Algoritma *hill-climbing* yang kami desain sudah pasti menemukan *global maximum* karena adanya heuristik dan pengecekan nilai *objective function* pada tiap semua *candidate cells to move to*. Oleh karena itu, algoritma *hill-climbing* yang kami implementasikan sebenarnya kurang cocok disebut *local search*, lebih cocok disebut *greedy algorithm*.

D. *Genetic Algorithm*

Proses dari algoritma genetika yang digunakan menggunakan prinsip roda rolet dan inisialisasi acak. Kedua prinsip tersebut digunakan dengan harapan keteracakan tersebut akan membawakan variasi-variasi yang membawa ke hasil optimal pada generasi-generasi berikutnya. Selain itu, algoritma genetika juga mengandalkan mekanisme dari permainan yang akan memilih kotak acak bila lebih dari lima detik. Selain itu, tidak digunakan heuristik pada algoritma genetika karena perlu dilakukan pengecekan lagi kotak-kotak yang layak ditempati untuk setiap kromosom sehingga hanya dapat menggunakan populasi dan jumlah generasi yang kecil (lebih baik tidak menggunakan heuristik tetapi keduanya bisa cukup besar).

Pada algoritma genetika yang digunakan, gen-gen yang digunakan adalah representasi sepasang bilangan bulat x dan y yang merepresentasikan koordinat kotak yang dipilih sebagai gerakan. Setelah itu, kromosomnya adalah gerakan-gerakan yang terjadi pada permainan sehingga panjangnya sesuai dengan jumlah giliran yang tersisa. Perlu diperhatikan bahwa tidak akan terdapat gen yang muncul lebih dari sekali pada kromosom karena tidak mungkin menempatkan simbol pada kotak yang telah terisi. Fungsi kecocokan yang digunakan adalah fungsi objektif yang telah didefinisikan pada bab A.

Pada pemilihan *parents*-nya, digunakan prinsip roda rolet yang menggunakan nilai dari fungsi objektif. Perlu diperhatikan bahwa diperlukan normalisasi nilai karena nilai dari fungsi objektif dapat menjadi negatif atau total dari seluruh nilai fungsi objektif adalah nol (ketika nilai-nilai dari fungsi objektif setiap kromosom adalah nol). Bila didapati fungsi objektif yang negatif, maka dilakukan normalisasi dengan menambahkan nilai setiap fungsi objektif dengan suatu nilai k sehingga nilai minimalnya adalah 1 (satu). Ketika didapati total fungsi objektif adalah 0 (nol), maka akan dibuat setiap fungsi objektif menjadi 1 (satu). Normalisasi-normalisasi ini dilakukan untuk membuat perhitungan rasio pada roda roletnya dalam jangkauan 0 (nol) hingga 1 (satu) ketika digunakan rumus nilai fungsi objektif dibagi dengan total nilai fungsi objektif seluruh kromosom.

Setelah dipilih *parents* yang akan melakukan *crossover*, dilakukan proses *crossover* secara acak lagi. Pada tahap ini, *crossover point* yang digunakan adalah sebuah indeks acak dari kromosom (perlu diperhatikan karena sangat acak maka dapat terletak pada posisi pertama yang menyebabkan dua kromosom hanya bertukar posisi karena seluruh gennya bertukar maupun pada posisi terakhir yang menyebabkan tidak adanya pertukaran gen). Proses *crossover*

dilakukan antara dua kromosom bersebelahan yang dipilih (sehingga kromosom ke-*i* akan melakukan *crossover* dengan kromosom ke-*i+1*, kromosom ke-*i+2* akan melakukan *crossover* dengan kromosom ke-*i+3*, dan akhirnya kromosom ke-*n-1* akan melakukan *crossover* dengan kromosom ke-*n*). *Crossover point* yang dipilih akan diacak terus-menerus hingga tiga kali percobaan untuk mencoba menghasilkan kromosom yang valid, bila tidak valid maka akan digunakan *parentsnya* sebagai hasil dari *crossover* tersebut (karena satu proses *crossover* melibatkan dua *parents* dan menghasilkan dua *children*). Setelah dilakukan *crossover*, maka akan dilakukan mutasi acak dengan probabilitas 15% (lima belas persen). Apabila mutasi dilakukan, maka titik mutasi adalah indeks acak. Ketika indeks dari titik mutasi telah ditetapkan, maka akan dipilih suatu gen yang valid dari potensi-potensi gen yang ada untuk merubah gen pada indeks yang dipilih secara acak.

Proses-proses ini akan dilakukan secara berulang setiap kali *bot* algoritma genetik akan bergerak, detailnya adalah sebagai berikut: inisialisasi populasi secara acak dengan kromosom yang valid, lakukan pemilihan *parents* dengan roda rolet, lakukan *crossover*, lakukan *mutation*, dan ulangi proses pemilihan *parents* hingga *mutation* hingga jumlah generasi maksimum. Untuk parameter yang digunakan, antara lain: ukuran populasi 50 dan jumlah generasi 300.

E. Emergency Move Algorithm

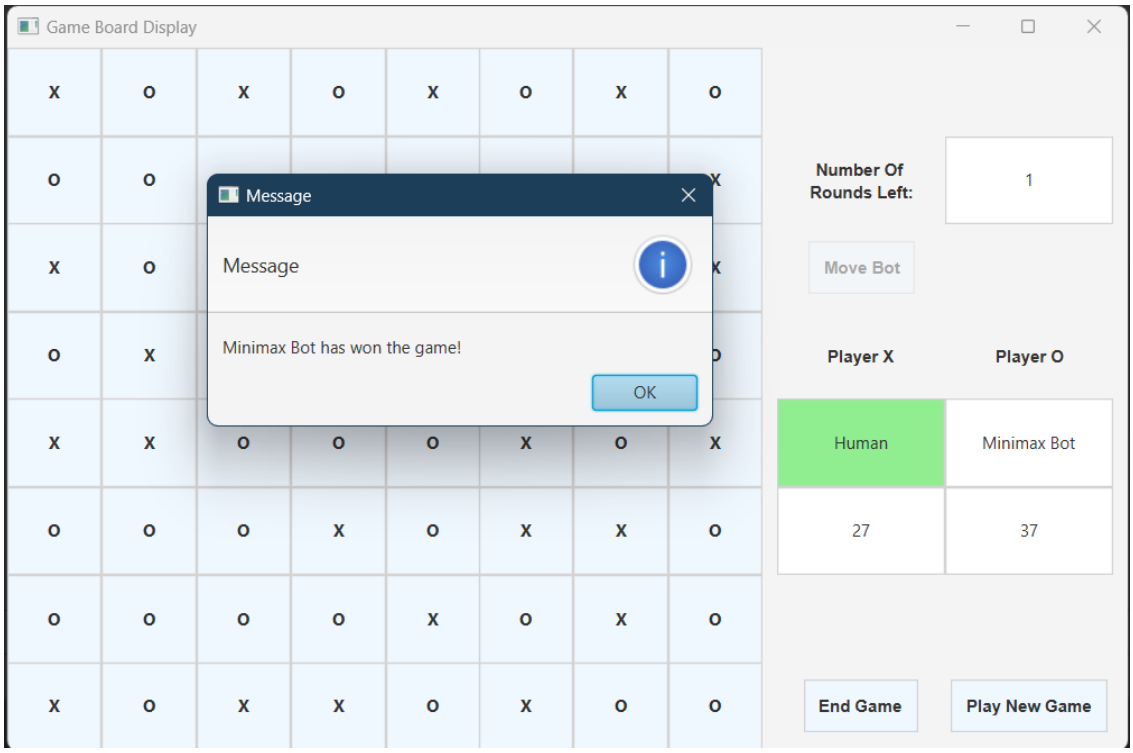
Terdapat algoritma khusus untuk menentukan langkah berikutnya ketika bot awal sudah melebihi batas waktu yang ditentukan. Ketika bot sudah melewati batas 5 detik, perhitungan akan dibatalkan dan algoritma ini akan dijalankan. Langkah-langkah algoritma ini adalah sebagai berikut.

1. Dari state sekarang, iterasi semua petak kosong
2. Dari semua petak kosong, pilih satu petak secara acak sebagai langkah berikutnya

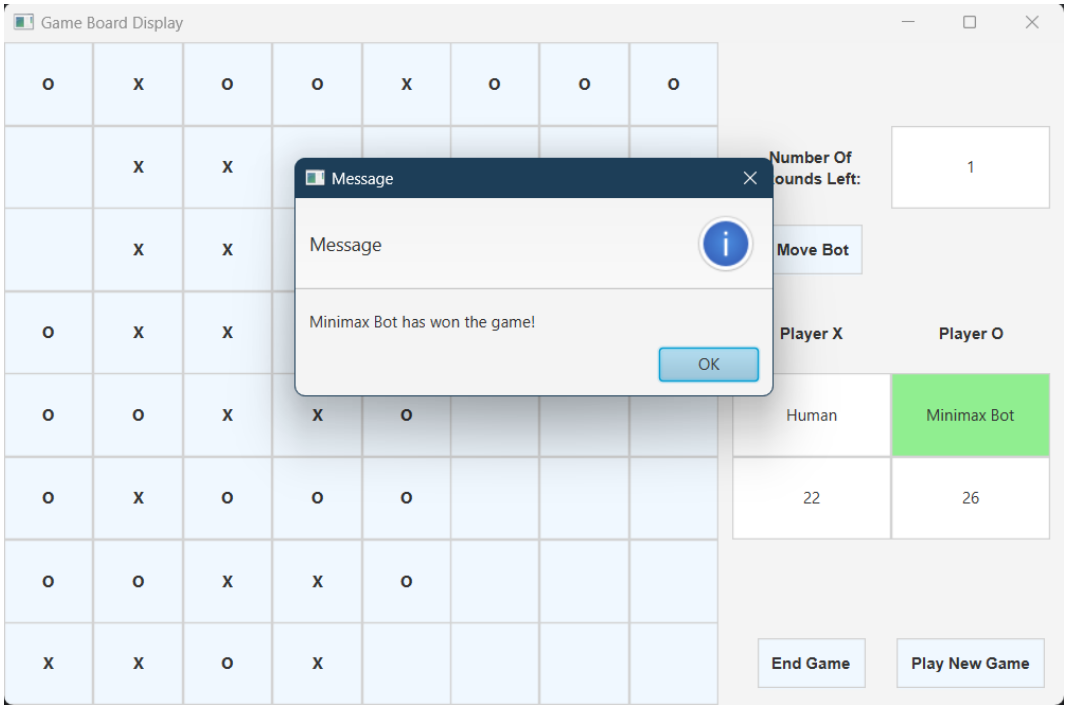
F. Hasil Percobaan

1. Bot *minimax* vs Manusia

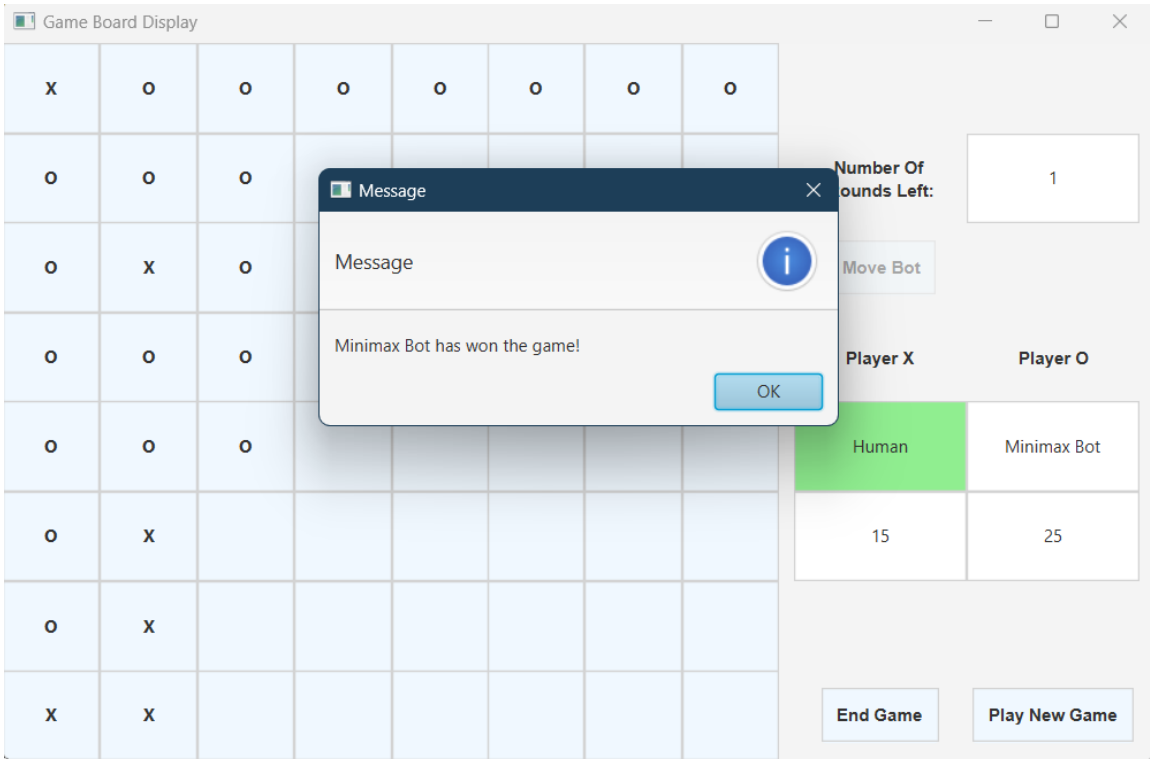
    a. Percobaan 1: 28 ronde



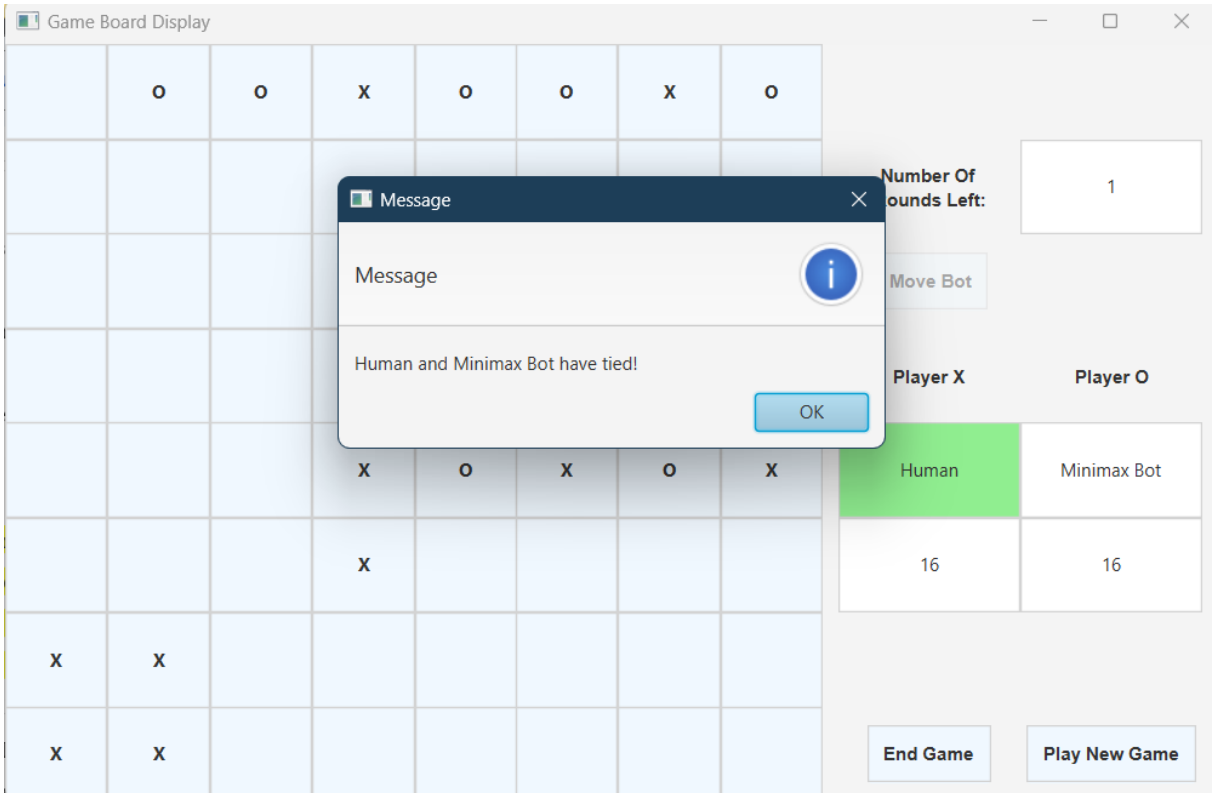
    b. Percobaan 2: 20 ronde



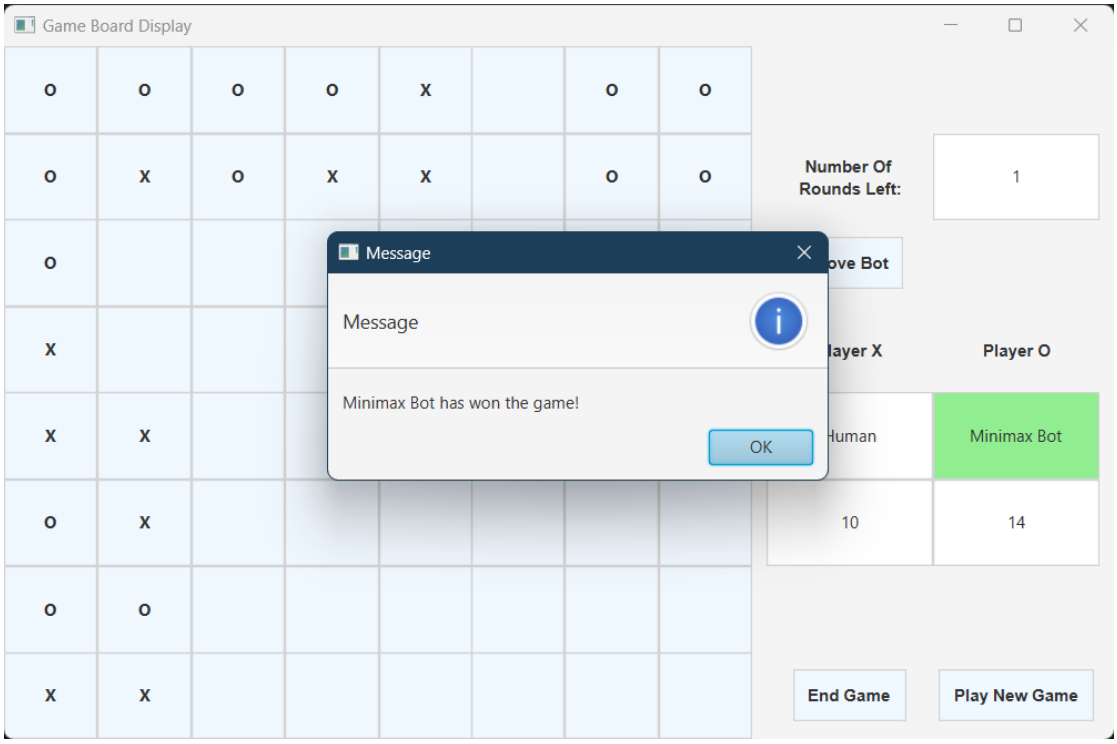
c. Percobaan 3: 16 ronde



d. Percobaan 4: 12 ronde

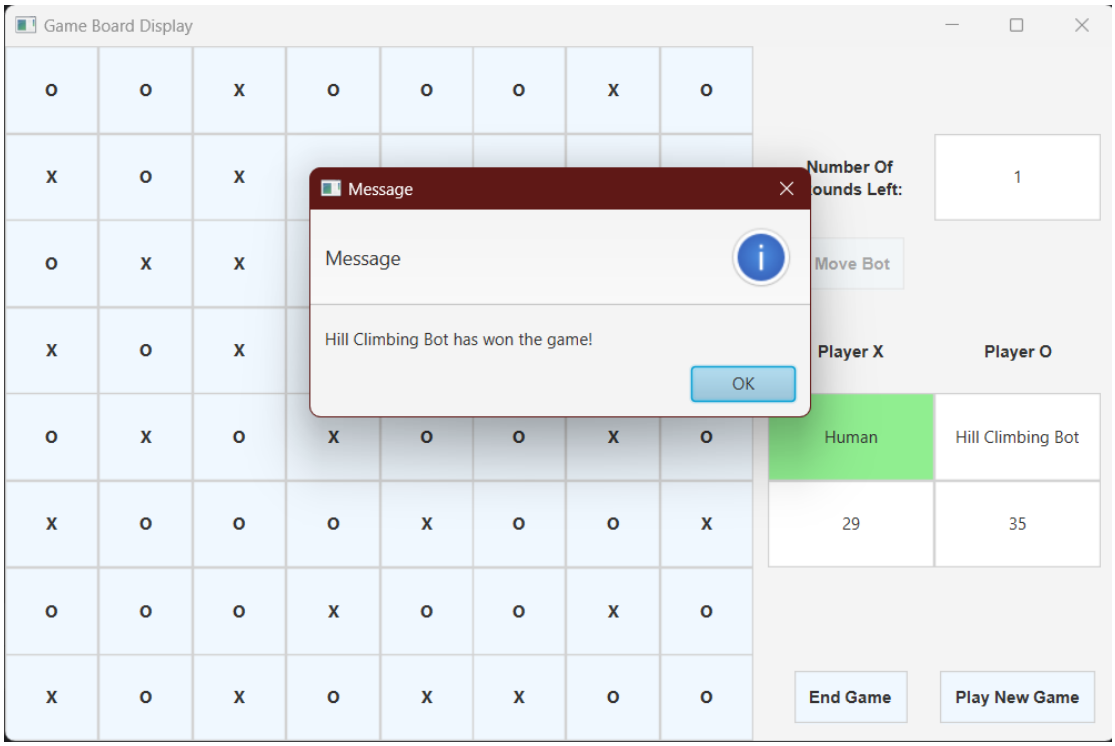


e. Percobaan 5: 8 ronde

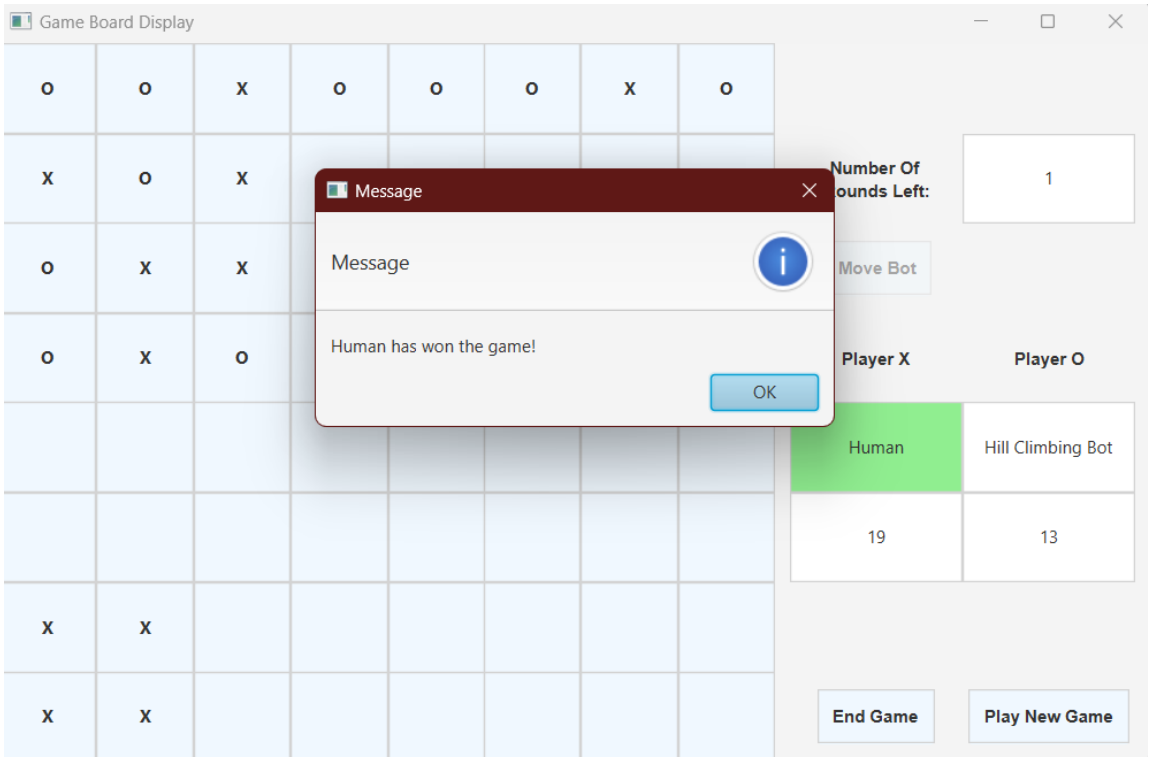


2. Bot *local search* vs manusia

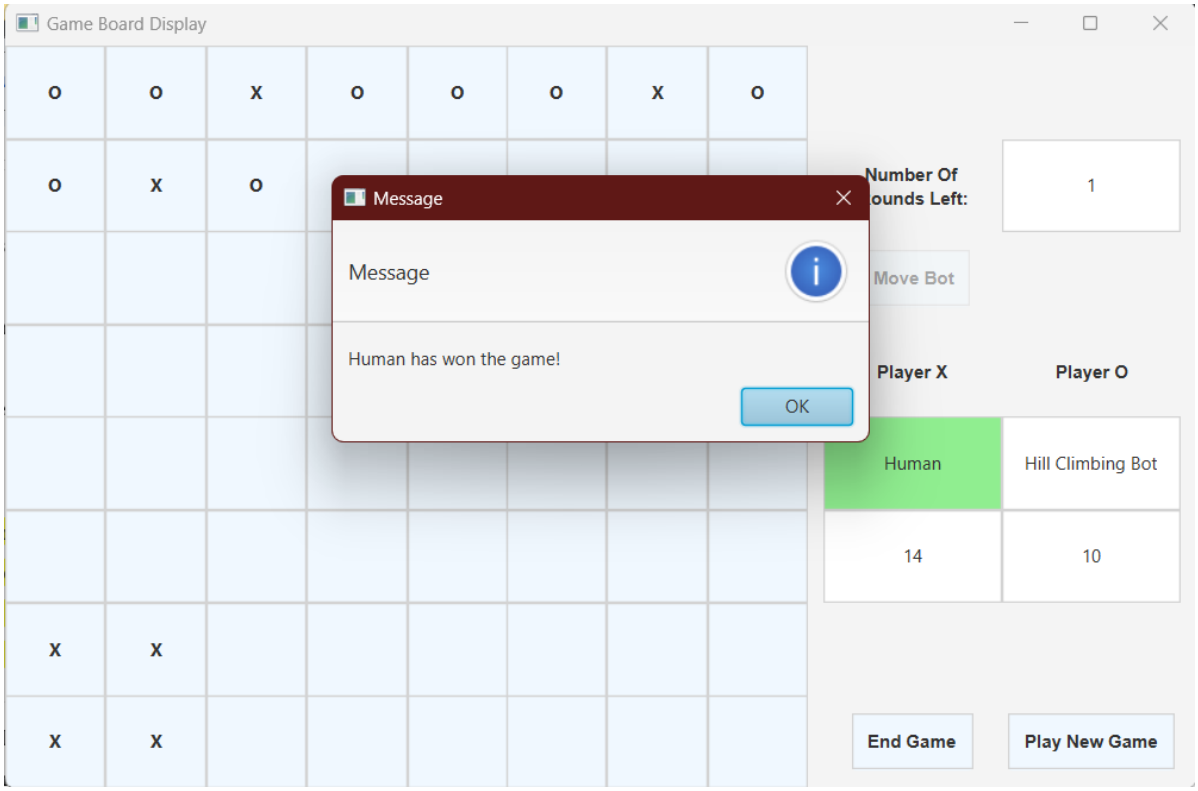
a. Percobaan 1: 28 ronde



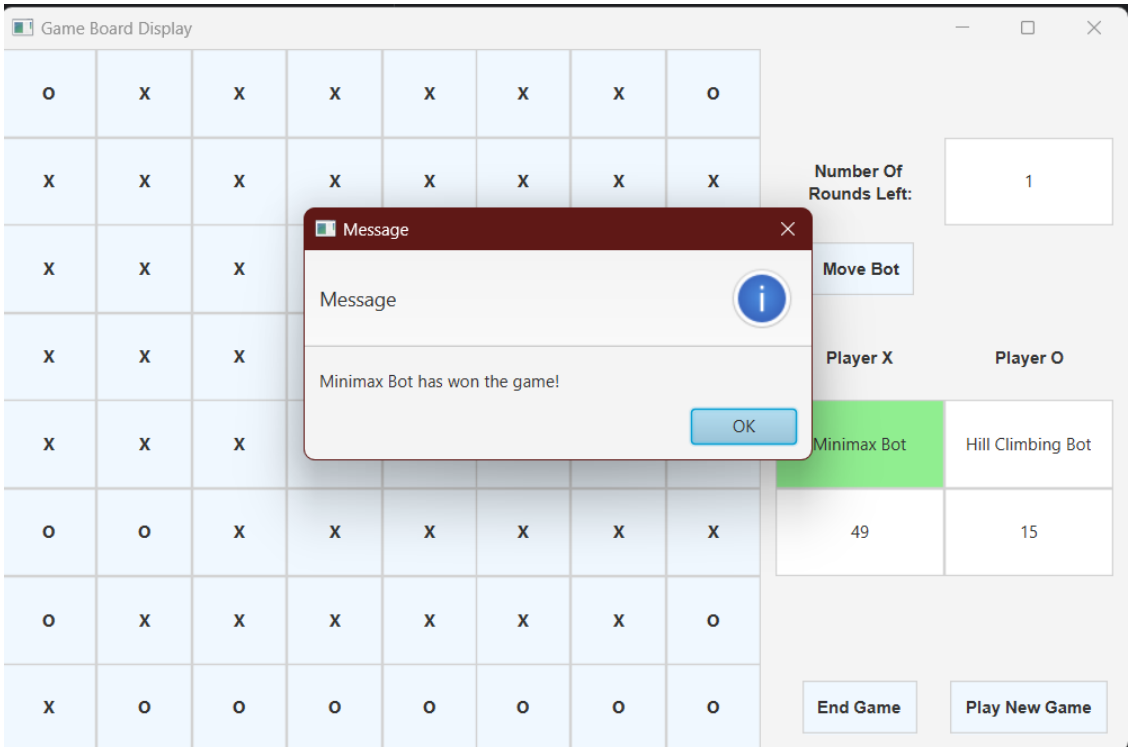
b. Percobaan 2: 12 ronde



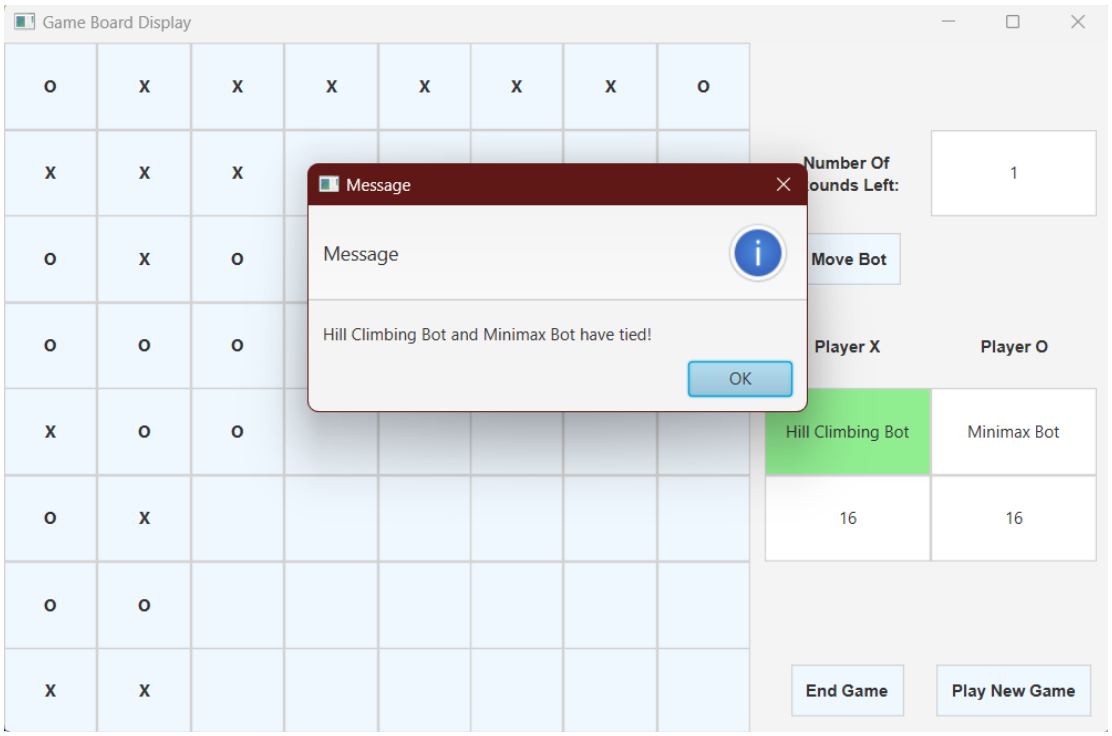
c. Percobaan 3: 8 ronde



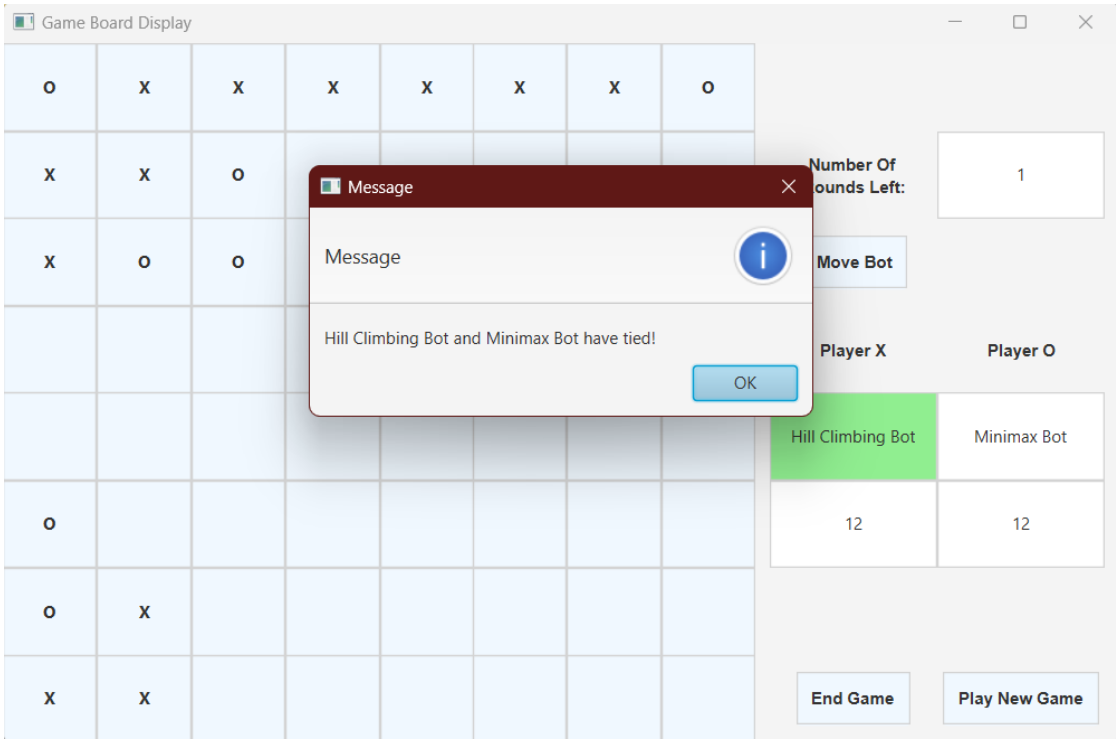
3. Bot *minimax* vs bot *local search*
    a. Percobaan 1: 28 ronde



    b. Percobaan 2: 12 ronde

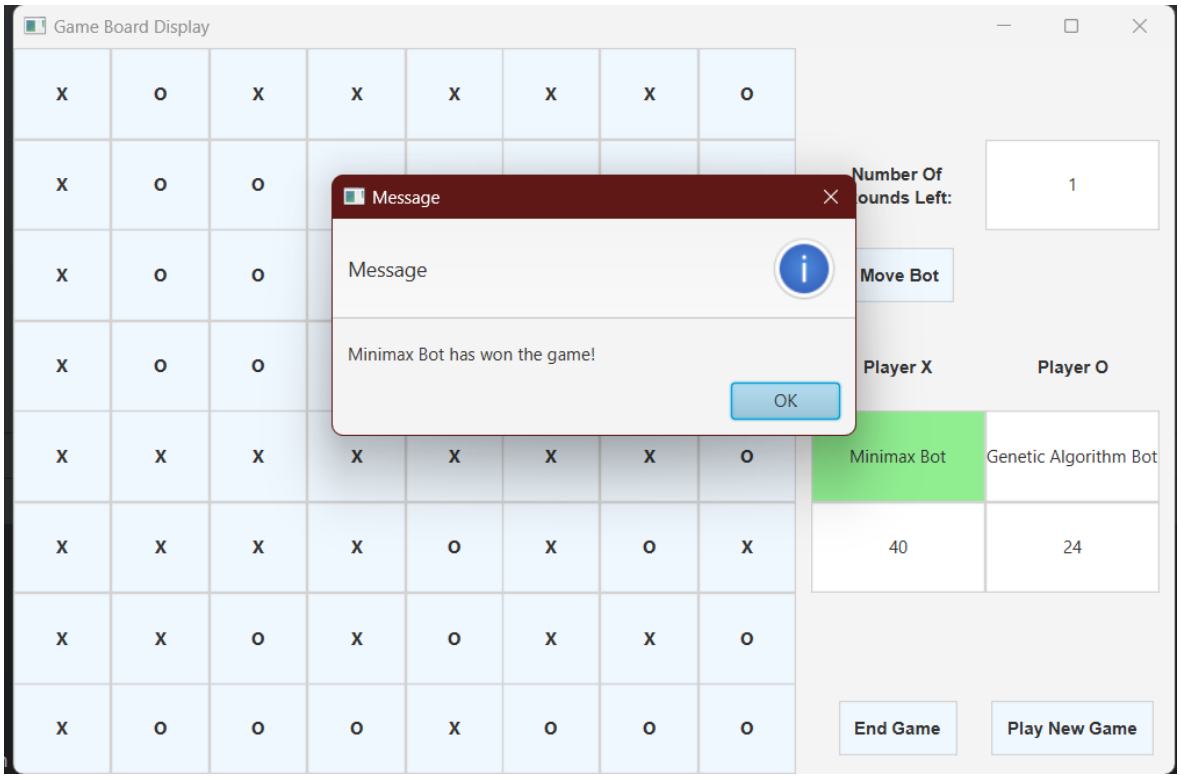


c. Percobaan 3: 8 ronde

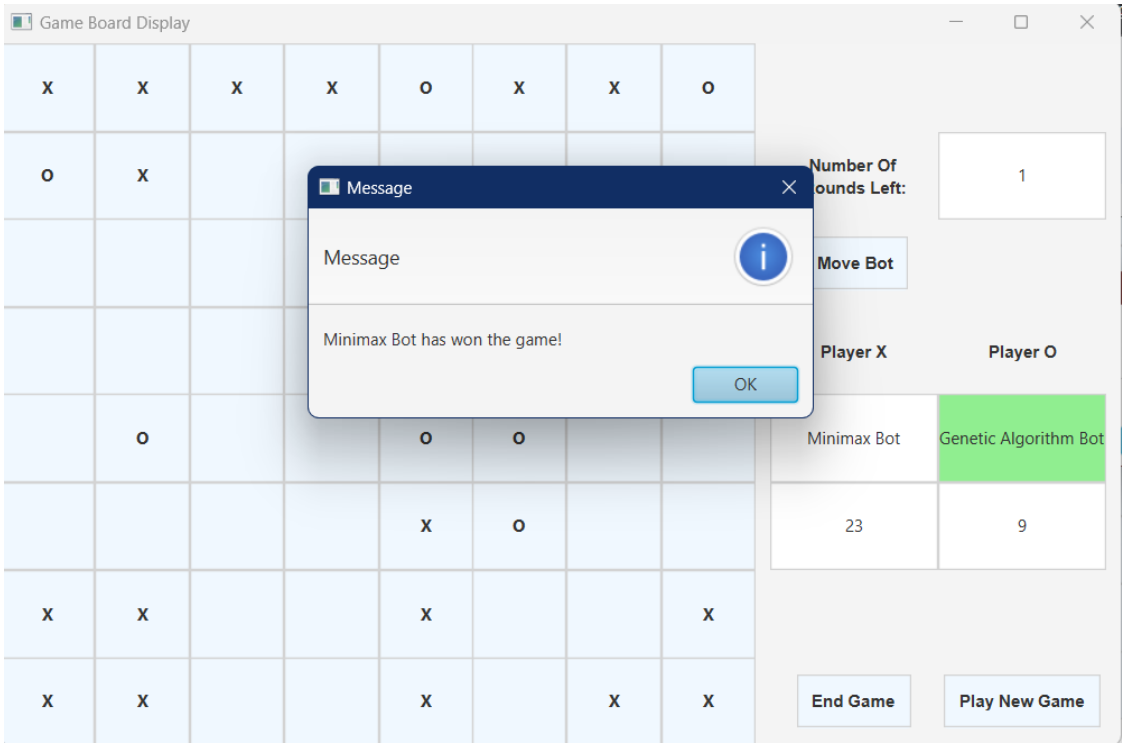


4. Bot *minimax* vs bot *genetic algorithm*

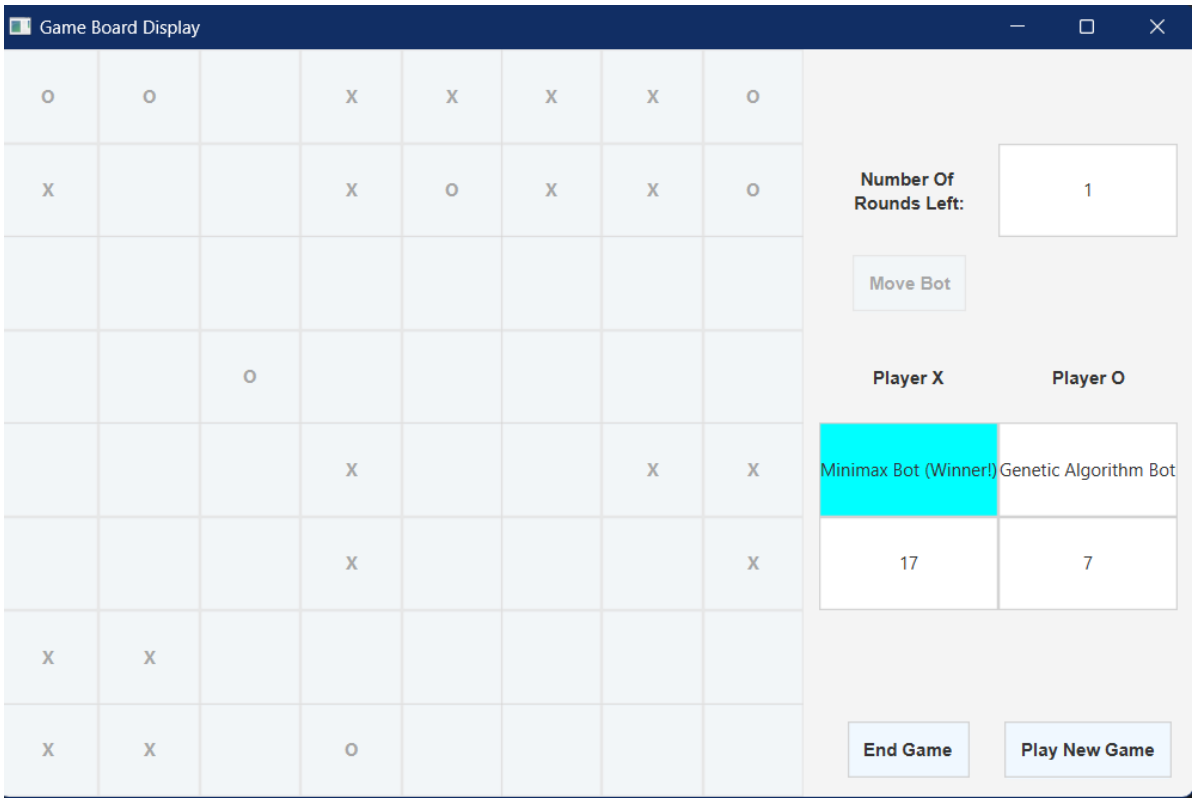
a. Percobaan 1: 28 ronde



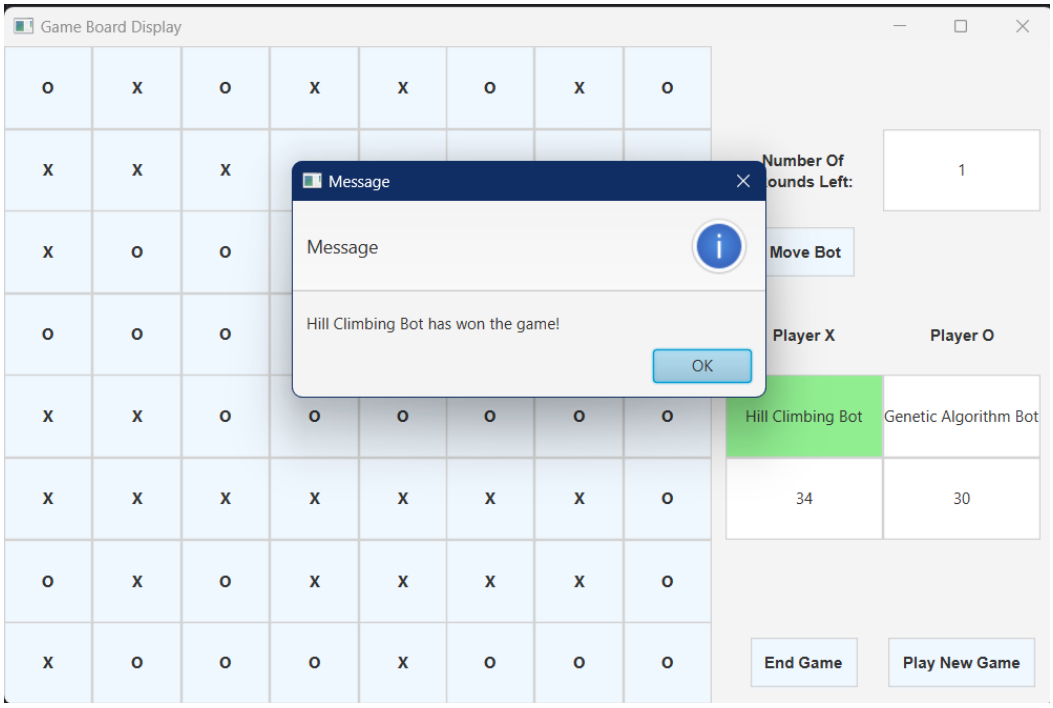
b. Percobaan 2: 12 ronde



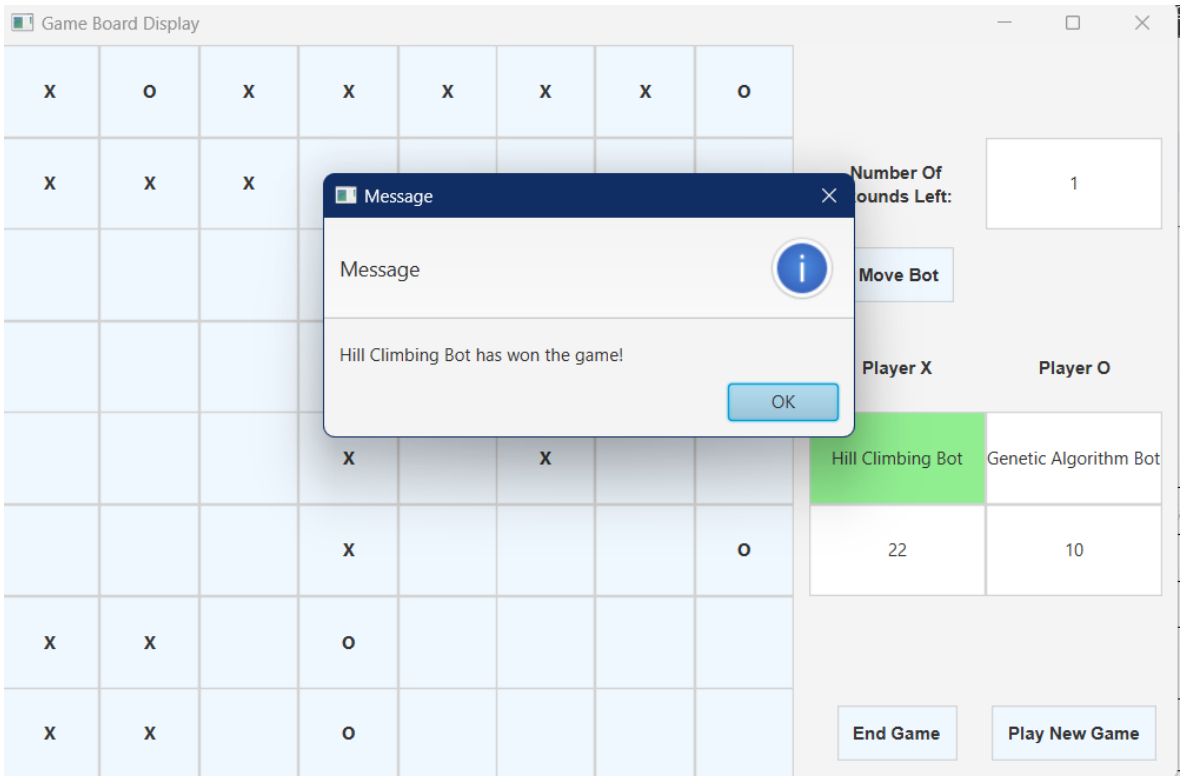
c. Percobaan 3: 8 ronde



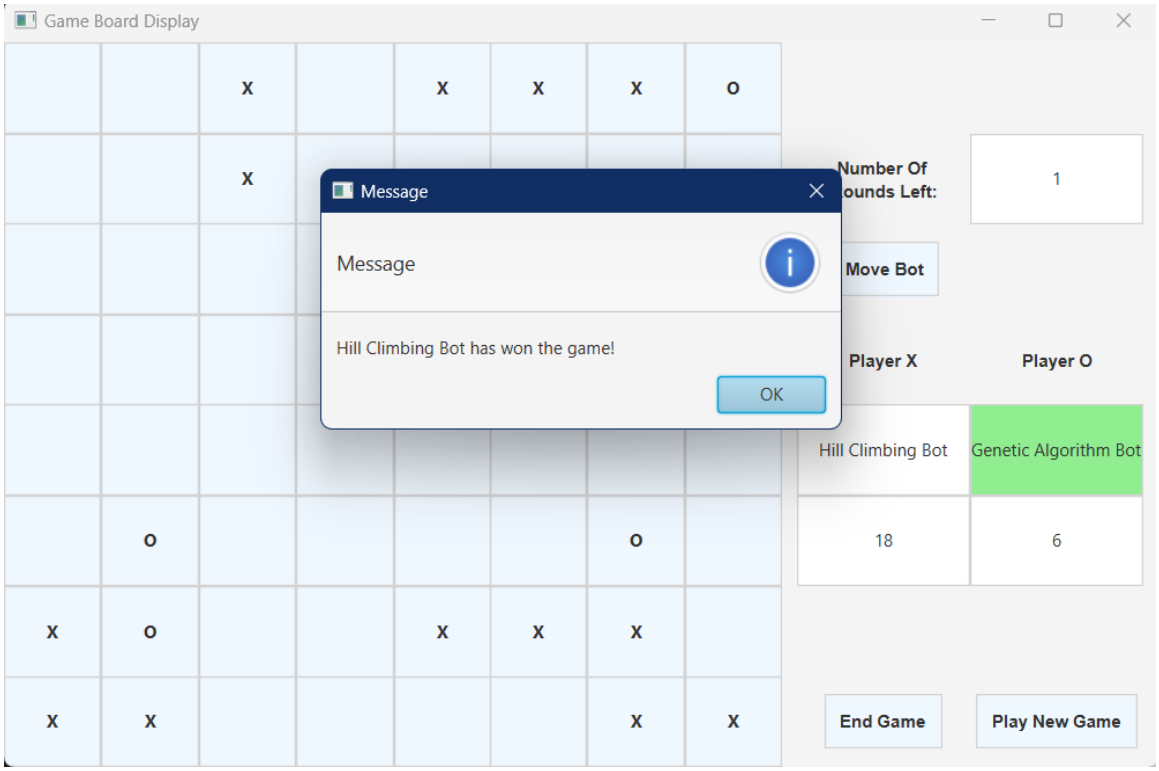
5. Bot *local search* vs bot *genetic algorithm*
    a. Percobaan 1: 28 ronde



    b. Percobaan 2: 12 ronde



c. Percobaan 3: 8 ronde



G. Ringkasan Hasil Percobaan

	Menang	Seri	Kalah	Win rate	Tie Rate	Lose rate
Manusia (noob)	2	1	5	25%	12,5%	62,5%
Bot Minimax	8	3	0	72%	28%	0%
Bot Local Search	4	2	3	44%	22%	34%
Bot Genetic Algorithm	0	0	6	0%	0%	100%

Berdasarkan hasil percobaan di atas, jelas bahwa bot minimax merupakan bot paling baik dengan win rate 72% dan tidak pernah kalah sekali pun. Bot kedua terbaik adalah bot local search dan terakhir adalah bot genetic algorithm. Pada implementasi kami, bot genetic algorithm memiliki performa yang buruk.

H. Kontribusi

NIM	Kontribusi
13521060	GameState, MiniMaxBot
13521093	Emergency move & logika trigger-nya, kelas bot utama, *generally everything*, ~~gorengan oharang~~
13521096	HillClimbingBot
13521144	Heuristik, GeneticAlgorithmBot

Lampiran

Link Repository