Laporan Tugas Besar 1 IF3170 Inteligensi Buatan
Semester I Tahun 2023/2024

Minimax Algorithm and Alpha Beta Pruning in Adjacency Strategy Game

Disusun oleh:

Kelvin Rayhan Alkarim	13521005
Azmi Hasna Zahrani	13521006
Ditra Rizqa Amadia	13521019
Ahmad Nadil	13521024

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

DAFTAR ISI

I. Objective Function

Berikut adalah link *github* berisi *source code* implementasi program:
https://github.com/IceTeaXXD/Tubes1_13521005

1.1 Class HCSidewaysBot

1.1.1 *public int[] move(Button[][] board, int roundsLeft)*

Fungsi ini bertujuan untuk melakukan iterasi sepanjang board dan melakukan simulasi penempatan / gerakan pada sebuah kotak yang kosong. Setelah mensimulasi pergerakan tersebut, nantinya fungsi ini akan memanggil fungsi *evaluate* untuk melakukan kalkulasi gerakan yang mengembalikan nilai dari gerakan tersebut (akan dijelaskan lebih lanjut). Pada fungsi move ini akan juga ditentukan gerakan manakah yang memiliki nilai terbaik. Gerakan yang memiliki nilai terbaik akan dikembalikan dalam bentuk tuple yang berisi indeks baris dan kolom gerakan terbaik. Program akan diterminasi ketika neighbor yang akan diassign ke state current memiliki skor evaluasi lebih kecil daripada state saat ini.

1.2. Class StochasticHCBot

1.2.1. public int[] move(Button[][] board, int roundsLeft)

Fungsi ini bertujuan untuk melakukan iterasi sebanyak N_MAX di dalam board dan melakukan simulasi penempatan / gerakan pada sebuah kotak yang kosong secara *random*. Setelah mensimulasi pergerakan tersebut, nantinya fungsi ini akan memanggil fungsi *evaluate* untuk melakukan kalkulasi gerakan yang mengembalikan nilai dari gerakan tersebut (akan dijelaskan lebih lanjut). Pada fungsi move ini akan juga ditentukan gerakan manakah yang memiliki nilai terbaik. Gerakan yang memiliki nilai terbaik akan dikembalikan dalam bentuk tuple yang berisi indeks baris dan kolom gerakan terbaik.

1.2.2. public int evaluate(Button[][] board)

Fungsi ini digunakan untuk menghitung jumlah skor pemain dengan menjumlahkan total banyaknya bidak “O” dikurangi banyaknya bidak “X”. Fungsi ini juga dipanggil pada fungsi move.

1.2.3. public Button[][] updateVirtualBoard(int x, int y, Button[][] board, boolean bot)

Fungsi ini digunakan untuk menyimpan evaluasi dari papan permainan untuk beberapa kemungkinan solusi yang akan digunakan oleh pemain untuk langkah selanjutnya.

1.2.4. private int[] getRandomMove(Button[][] board)

Fungsi ini digunakan untuk mendapatkan populasi random dan akan dipanggil ketika fungsi move melakukan generate populasi.

1.3. Class MinimaxBot

1.3.1 public int[] move(Button[][] board, int roundsLeft)

Fungsi ini bekerja dengan memanggil fungsi minimax untuk melakukan kalkulasi gerakan yang mengembalikan nilai dari gerakan tersebut. Hasil akhir pada fungsi ini merupakan koordinat (move) yang akan menjadi koordinat peletakan bidak pemain.

1.3.2 public int minimaxEvaluation(Button[][] board)

Fungsi ini merupakan fungsi yang digunakan untuk mengevaluasi skor pemain "O" dan pemain "X". Fungsi ini bekerja dengan cara menghitung jumlah bidak pemain "O" dikurangi dengan jumlah bidak pemain "X". Selanjutnya, fungsi akan mengembalikan hasil skor yang didapat.

1.3.3. public Button[][] updateVirtualBoard(int x, int y, Button[][] board, boolean bot)

Fungsi ini digunakan untuk menyimpan evaluasi dari papan permainan untuk beberapa kemungkinan solusi yang akan digunakan oleh pemain untuk langkah selanjutnya.

1.3.4. public int[] minimax(Button[][] board, int depth, int roundsLeft, int alpha, int beta, boolean bot)

Fungsi ini merupakan fungsi utama yang digunakan dalam penyelesaian permainan Adjacency Strategy Game. Pada fungsi ini dilakukan evaluasi langkah yang mungkin menjadi solusi oleh pemain maupun lawan dengan cara rekursif. Di dalam evaluasi, dilakukan pemanggilan fungsi updateVirtualBoard sebagai penyimpan hasil evaluasi sementara. Kemudian, board sementara akan dievaluasi skornya menggunakan algoritma minimax. Selain itu, terdapat algoritma pruning untuk mengoptimalkan jalannya algoritma.

1.3.5. public int[] fallbackHC(Button[][] board, int roundsLeft)

Fungsi ini merupakan fungsi fallback ketika algoritma mengevaluasi plan lebih dari 5 detik maka akan otomatis langkah akan diambil menggunakan algoritma Hill Climbing.

1.4. Class GeneticBot

1.4.1. public int[] move(Button[][] board, int roundsLeft)

Fungsi ini digunakan untuk melakukan operasi algoritma genetic, yaitu inisialisasi populasi, crossover, selection, dan mutation. Pada fungsi ini dilakukan generate populasi secara random sebanyak POPULATION_SIZE, yaitu yang akan digunakan sebagai parent. Kemudian, parent-parent tersebut akan dihitung fitness functionnya. Selanjutnya, akan dipilih parent dari pemanggilan fungsi

roulleteWheelSelection(fittness), atau merandom fitness function masing-masing parent menggunakan roullete wheel. Setelah didapatkan dua parent terbaik, mereka akan disilangkan (crossover) sehingga membentuk dua child. Kemudian, dua child tersebut akan dilakukan mutation sebagai pelengkap dari algoritma genetic. Selanjutnya, untuk memilih solusi terbaik dari kedua child yang didapat, akan dilakukan pemilihan menggunakan algoritma minimax dan ketika telah didapatkan solusi, akan direturn koordinat untuk penempatan bidak pemain.

1.4.2. private int[] getRandomMove(Button[][][] board)

Fungsi ini digunakan untuk mendapatkan populasi random dan akan dipanggil ketika fungsi move melakukan generate populasi.

1.4.3. private int rouletteWheelSelection(ArrayList<Integer> fitness)

Fungsi ini dipanggil pada fungsi move dan digunakan untuk mendapatkan fitness function pada populasi secara random.

1.4.4. public int evaluate(Button[][][] board, int row, int col)

Fungsi ini digunakan untuk menghitung jumlah skor pemain dengan menjumlahkan total banyaknya bidak “O” dikurangi banyaknya bidak “X”. Fungsi ini juga dipanggil pada fungsi move.

1.4.5. public Button[][][] updateVirtualBoard(int x, int y, Button[][][] board, boolean bot)

Fungsi ini digunakan untuk menyimpan evaluasi dari papan permainan untuk beberapa kemungkinan solusi yang akan digunakan oleh pemain untuk langkah selanjutnya.

1.4.6. private int minimax(Button[][][] board, int row, int col, int depth, int roundsLeft, boolean maximizing)

Fungsi ini bekerja sama seperti fungsi minimax pada class HillClimbingBot.

1.4.7. public int[] fallbackHC(Button[][][] board, int roundsLeft)

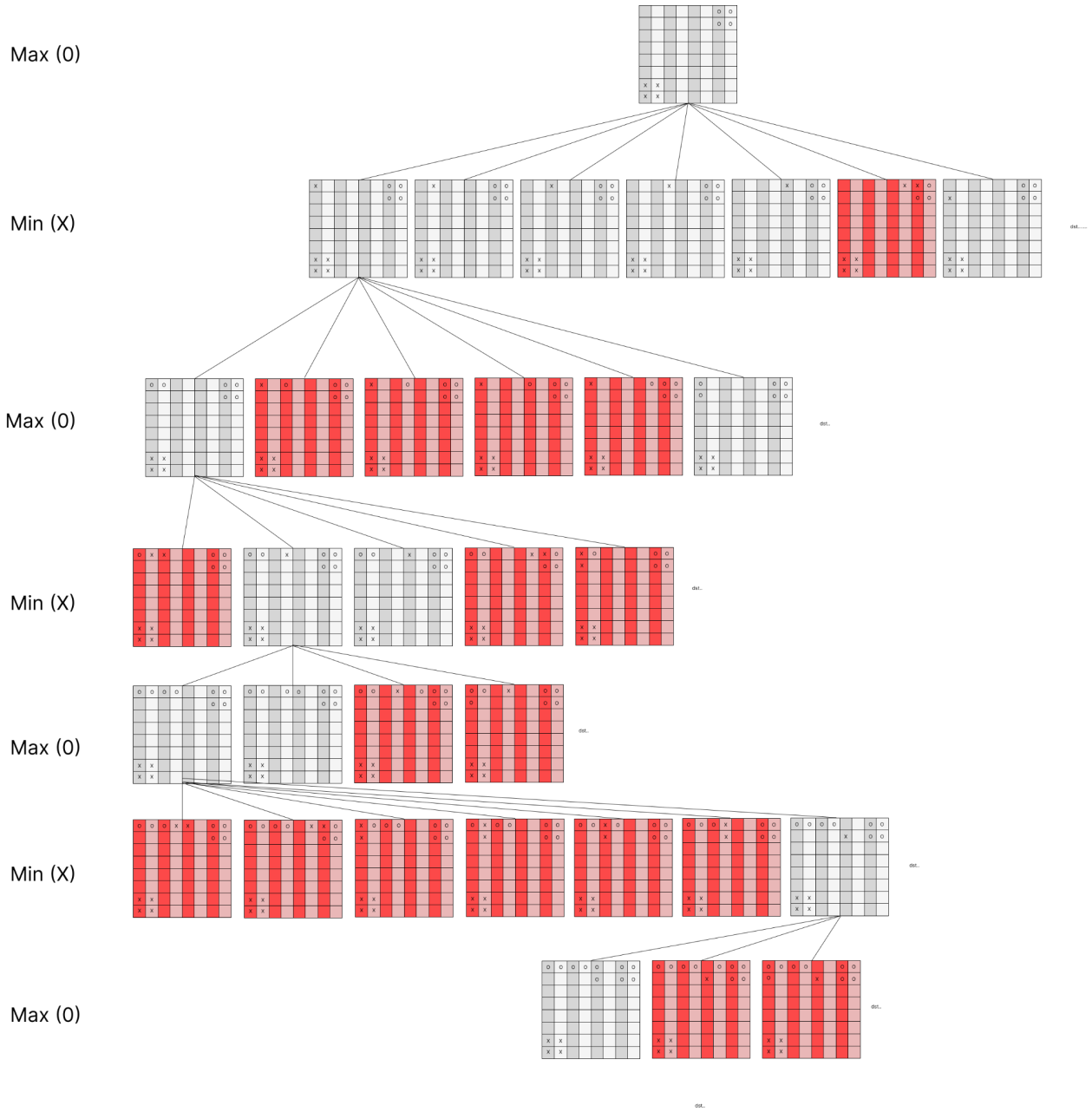
Fungsi ini merupakan fungsi fallback ketika algoritma mengevaluasi plan lebih dari 5 detik maka akan otomatis langkah akan diambil menggunakan algoritma Hill Climbing.

II. Proses Pencarian dengan Minimax dan Alpha Beta Pruning pada Permainan Adjacency Strategy Game

Permainan Adjacency Strategy Game dapat diselesaikan menggunakan beberapa algoritma pencarian. Pada bab ini akan dijelaskan proses pencarian permainan Adjacency Strategy Game menggunakan algoritma minimax dan alpha beta pruning. Pencarian solusi permainan Adjacency Strategy Game menggunakan minimax dan alpha beta pruning dilakukan dengan memaksimalkan semua kemungkinan langkah oleh pemain dan meminimalkan semua langkah yang diambil oleh lawan. Pada setiap tahap pencarian, algoritma akan mengevaluasi seluruh kemungkinan gerakan yang dilakukan oleh pemain. Saat giliran pemain melakukan langkah, algoritma akan mencari gerakan yang dapat dilakukan sehingga menghasilkan nilai yang maksimal. Sedangkan, ketika giliran bermain berada pada lawan, algoritma akan mencari gerakan yang mungkin dilakukan oleh lawan sehingga menghasilkan nilai yang minimal. Algoritma ini akan berlangsung selama permainan masih berjalan hingga ditemukan kondisi akhir dari permainan.

Kemungkinan evaluasi yang dilakukan oleh algoritma minimax sangat banyak, sehingga hal ini dapat memengaruhi banyaknya waktu untuk evaluasi setiap langkahnya. Untuk mengatasi hal tersebut, dapat digunakan alpha beta pruning dalam pencarian dengan algoritma minimax. Alpha beta pruning akan mengurangi jumlah node yang akan dievaluasi sehingga pencarian minimax akan jauh lebih optimal. Ketika permainan berlangsung, akan disimpan nilai dengan variabel alpha dan beta. Alpha merepresentasikan skor terbaik yang telah ditemukan dari langkah pemain saat ini. Sedangkan, beta merupakan representasi dari skor terbaik yang telah ditemukan dari langkah lawan saat ini. Algoritma minimax dengan alpha beta pruning akan mengevaluasi setiap langkah pemain dan lawan. Ketika ditemukan langkah pemain dengan skor yang lebih rendah dari nilai alpha saat ini, maka pencarian pada cabang tersebut akan dihentikan pada saat itu sehingga tidak perlu mengevaluasi cabang tersebut lebih lanjut. Begitu pula apabila terjadi kondisi sebaliknya, ketika ditemukan langkah lawan dengan skor yang lebih tinggi daripada nilai beta saat ini, maka evaluasi pada cabang tersebut dipotong. Pemotongan cabang tersebut akan mempercepat evaluasi pada algoritma tanpa memengaruhi hasil akhir dari algoritma minimax.

Contoh langkah permainan algoritma minimax dengan alpha beta pruning pada permainan Adjacency Strategy Game direpresentasikan dengan ilustrasi berikut.



Pada ilustrasi di atas, digambarkan bagaimana algoritma minimax dengan alpha beta pruning bekerja. Setiap langkah yang diambil direpresentasikan dengan satu tingkat tree, dengan Max(O) merupakan langkah pemain dan Min(X) merupakan langkah lawan. Sesuai dengan algoritma alpha beta pruning, pada langkah pemain Max(O), cabang yang memiliki nilai lebih kecil dari alpha akan dipotong (dalam tree ditandai dengan warna merah). Begitu pula pada langkah lawan Min(X), ketika ditemukan langkah lawan dengan nilai yang lebih besar dari alpha akan dipotong.

III. Proses Pencarian dengan Local Search pada Permainan Adjacency Strategy Game

Algoritma Local Search dapat digunakan pada proses pencarian permainan Adjacency Strategy Game. Pada tugas besar ini, kelompok mengaplikasikan algoritma Hill-Climbing with Sideways move dan Stochastic Hill-climb search sebagai contoh dari algoritma Local Search. Pencarian permainan Adjacency Strategy Game menggunakan algoritma hill climbing dilakukan dengan mencari neighbor state pemain yang memiliki nilai skor tertinggi. Hal ini sesuai dengan konsep hill climbing, dimana algoritma akan terus mengambil kemungkinan terbaik seolah-olah seperti sedang mendaki gunung. Pada algoritma ini, pemain akan memaksimalkan skornya tanpa menghiraukan skor lawan.

Stochastic hill-climb akan merandom langkah sebanyak N_MAX kali, kemudian meng-*assign random successor* ke neighbor (kemungkinan langkah yang dapat diambil). Di saat neighbor memiliki nilai skor yang lebih besar daripada state saat ini, maka neighbor akan diambil sebagai solusi. Sedangkan, pada Hill Climbing dengan sideways move akan membolehkan langkah yang menghasilkan "flat" atau skor yang sama dengan skor saat ini. Jadi, ketika hasil skor tidak "naik", langkah tersebut masih mungkin untuk masuk ke solusi.

Berikut adalah *pseudocode* untuk Stochastic Hill-climb:

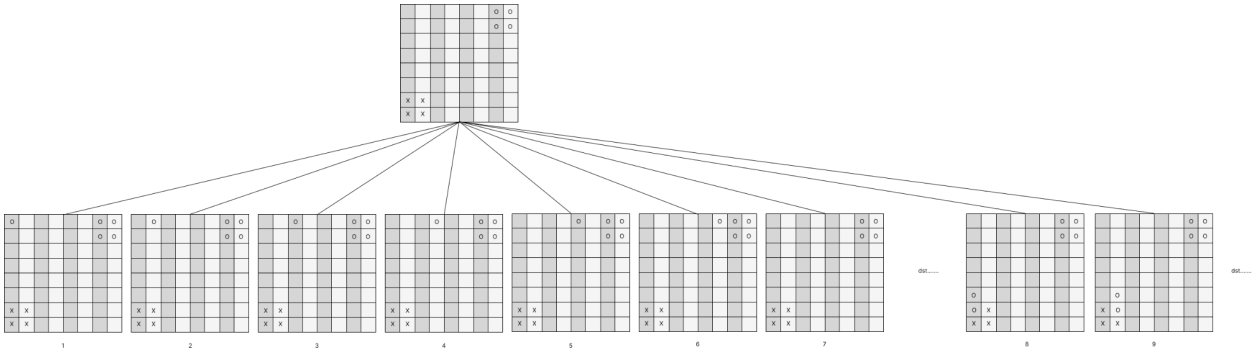
```
current <- Make-Node(problem.Initial-state)
repeat nmax times
   neighbor <- a random successor of current
   if neighbor.value > current.value THEN current <- neighbor
```

Hill-climbing with Sideways Move akan melakukan sebuah loop. Pada loop tersebut, akan diambil sebuah *successor* dari *initial state*. Setelah itu, akan dilakukan evaluasi nilai terhadap *successor* tersebut. Jika nilai yang dihasilkan lebih kecil dari *state* saat ini, maka akan diterminasi dan *return* saat ini. Akan tetapi, jika nilai yang dihasilkan lebih baik dari *state* saat ini, maka akan diambil *successor* tersebut menjadi *current state*.

Berikut adalah *pseudocode* untuk Hill-climb with Sideways Move:

```
current <- Make-Node(problem.Initial-state)
loop do
   neighbor <- a highest-valued successor of current
   if neighbor.value < current.value THEN return current.State
   current <- neighbor
```

Contoh permainan Adjacency Strategy Game menggunakan algoritma hill climbing direpresentasikan pada ilustrasi di bawah ini.



Pada ilustrasi di atas, kemungkinan langkah yang diambil pemain adalah langkah nomor 8 dan 9. Kedua langkah tersebut memiliki skor yang lebih tinggi daripada langkah lain di ronde yang sama sehingga memiliki kemungkinan untuk diambil sebagai solusi.

Meskipun algoritma ini bekerja untuk terus “naik” ke skor tertinggi, tetapi pada permainan ini, algoritma hill-climbing search hanya dapat diterapkan untuk mendapatkan Local Optimum. Hal ini disebabkan tidak ada depth atau kedalaman pada tree, atau dalam kata lain hanya mencari gerakan dengan nilai terbaik pada state saat ini dan tidak melakukan kalkulasi lebih lanjut untuk state selanjutnya.

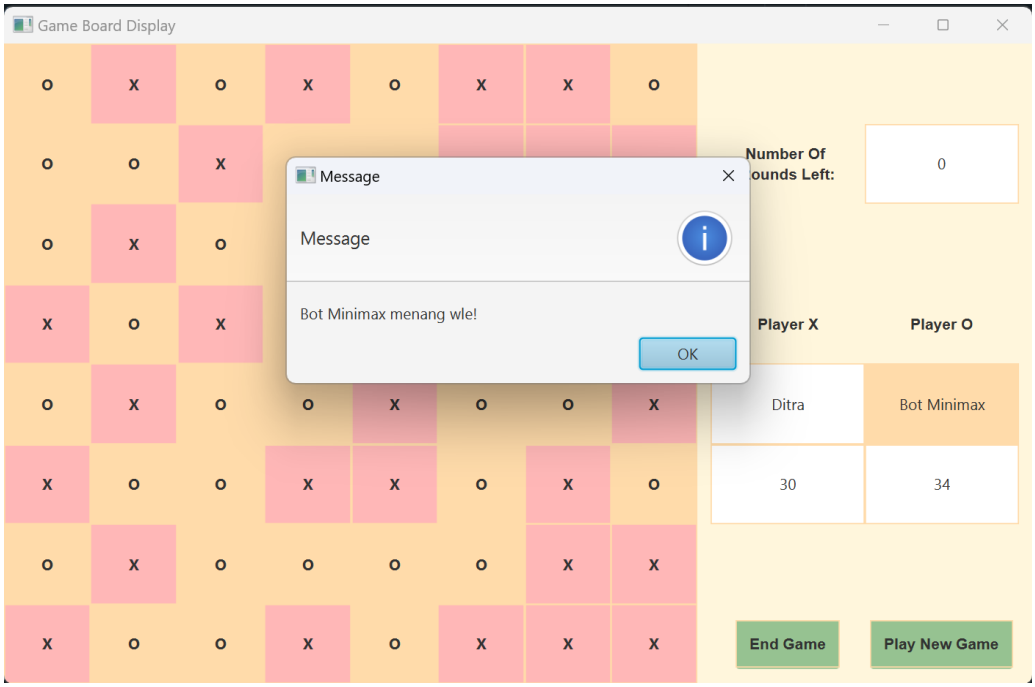
IV. Proses Pencarian dengan Genetic pada Permainan Adjacency Strategy Game

Penerapan algoritma *genetic* pada permainan Adjacency Strategy Game hanya dapat dilakukan dengan mengabaikan semua bidak yang telah diletakkan pada papan permainan. Algoritma *genetic* bekerja dengan cara merandom populasi sebanyak POPULATION_SIZE kali untuk menjadi parent. Kemudian, akan dievaluasi fitness function dari parent-parent tersebut dengan menghitung selisih jumlah bidak pemain dan lawan. Kemudian, parent tersebut akan disilangkan sebanyak MAX_GENERATION kali yang nantinya hasil persilangan ini akan dipilih melalui roulette yang menghasilkan sebanyak 2 parent untuk crossover. Setelah dilakukan crossover, setiap hasil crossover akan dilakukan mutation melalui random. Kemudian, hasil akhir child dari mutation akan dipilih yang memiliki skor paling baik menggunakan minimax.

V. Hasil Pengujian

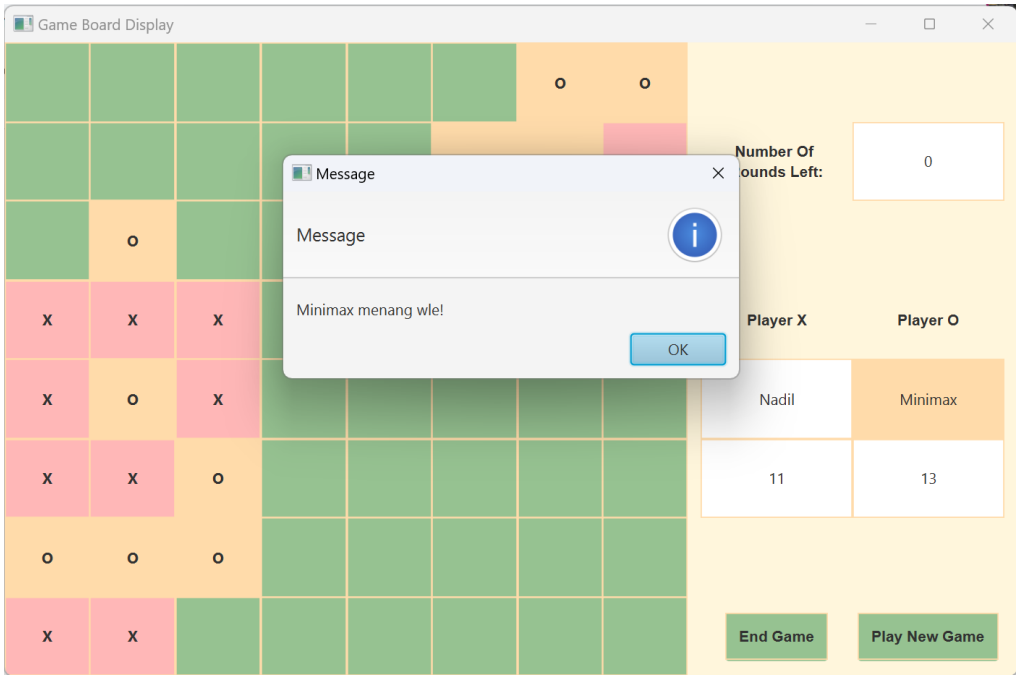
5.1. Bot Minimax vs Manusia

5.1.1. Pengujian Pertama (28 Ronde)



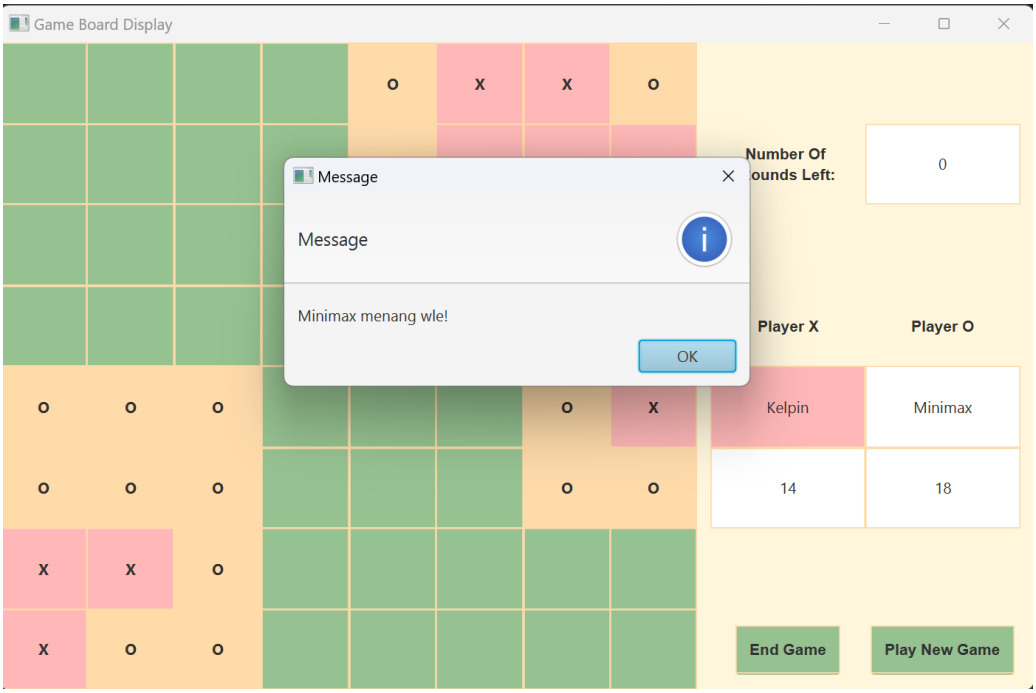
Gambar 5.1.1. Hasil Pengujian 1 antara Manusia vs Bot Minimax Alpha Beta Pruning

5.1.2. Pengujian Kedua (8 Ronde)



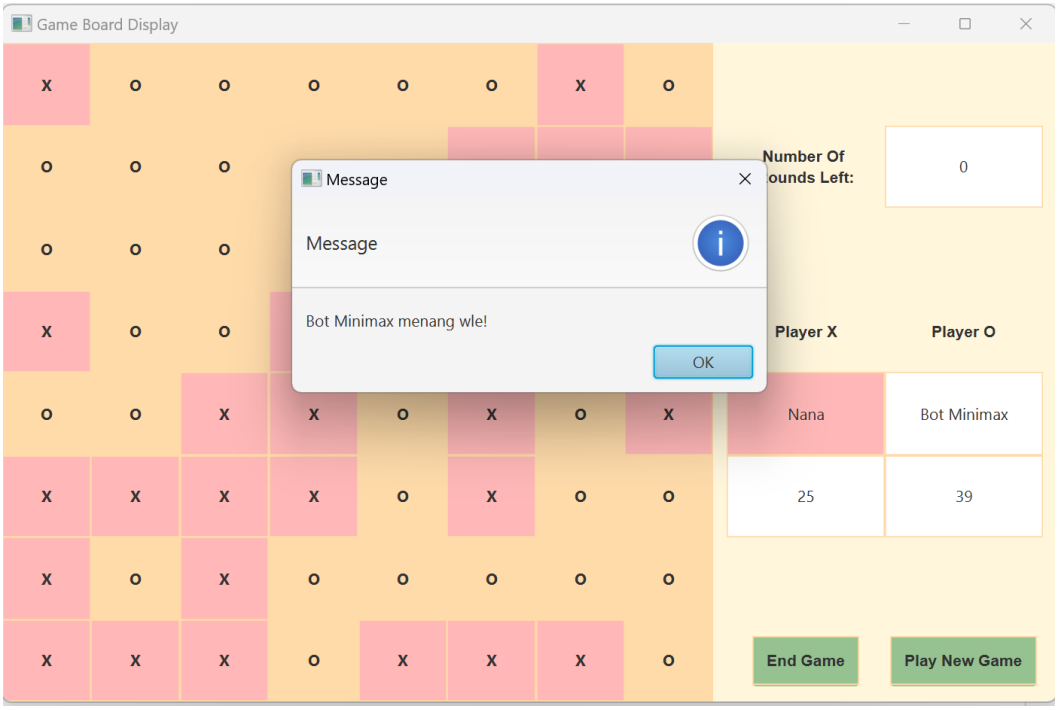
Gambar 5.1.2. Hasil Pengujian 2 antara Manusia vs Bot Minimax Alpha Beta Pruning

5.1.3. Pengujian Ketiga (12 Ronde)



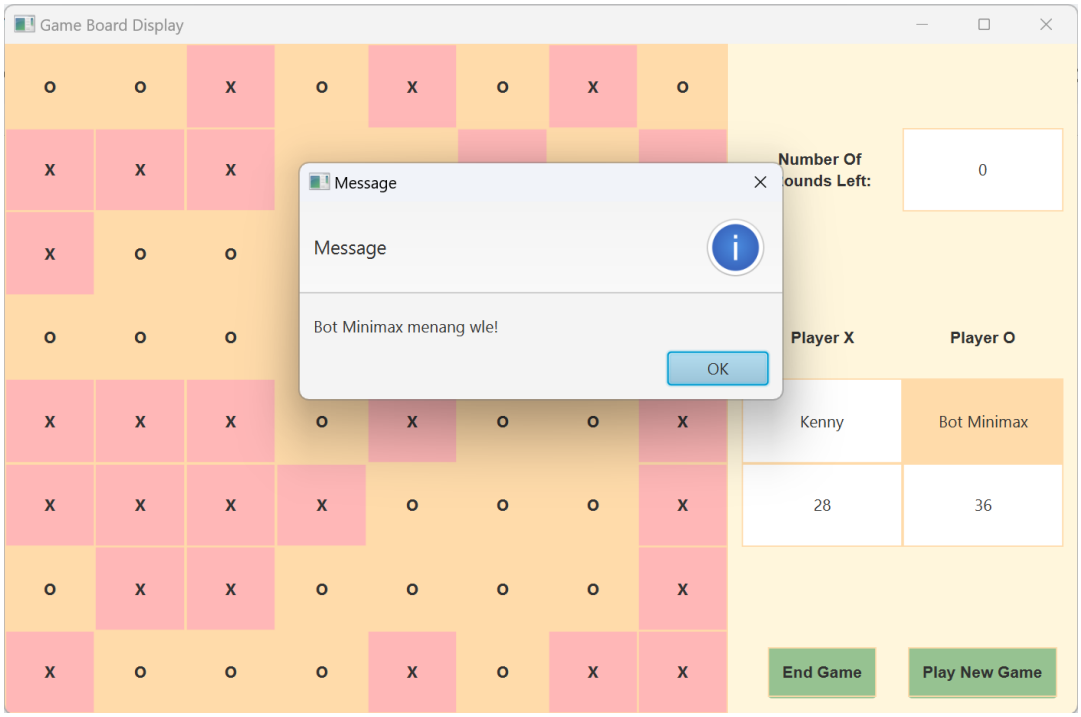
Gambar 5.1.3. Hasil Pengujian 3 antara Manusia vs Bot Minimax Alpha Beta Pruning

5.1.4. Pengujian Keempat (28 Ronde)



Gambar 5.1.4. Hasil Pengujian 4 antara Manusia vs Bot Minimax Alpha Beta Pruning

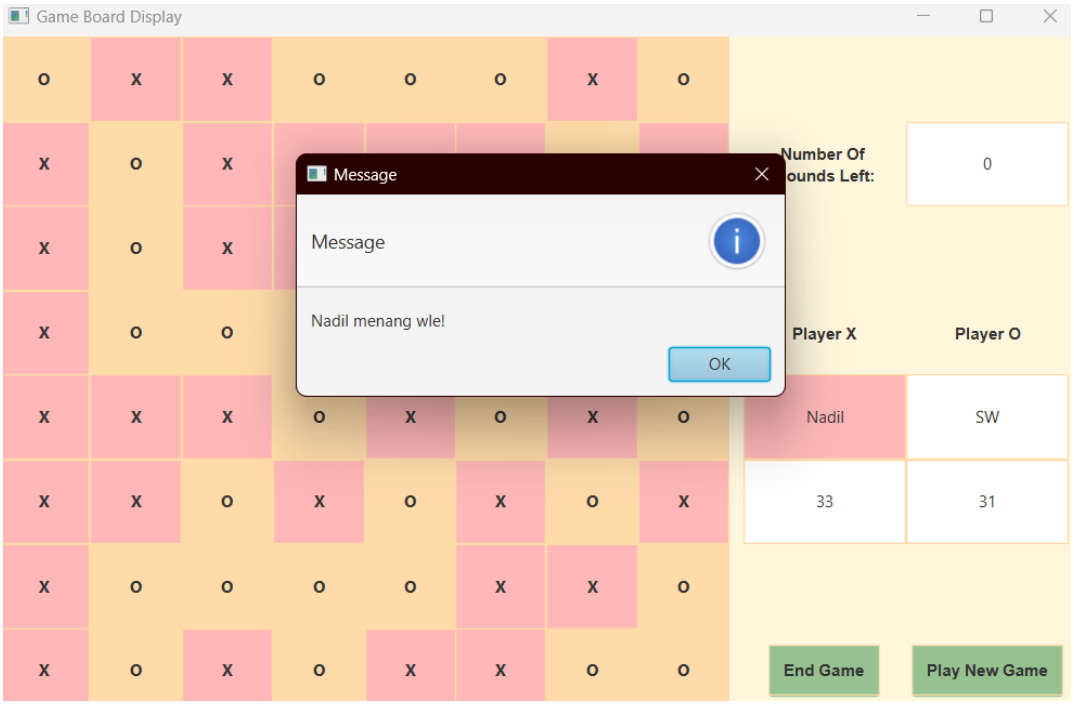
5.1.5. Pengujian Kelima (28 Ronde)



Gambar 5.1.5. Hasil Pengujian 5 antara Manusia vs Bot Minimax Alpha Beta Pruning

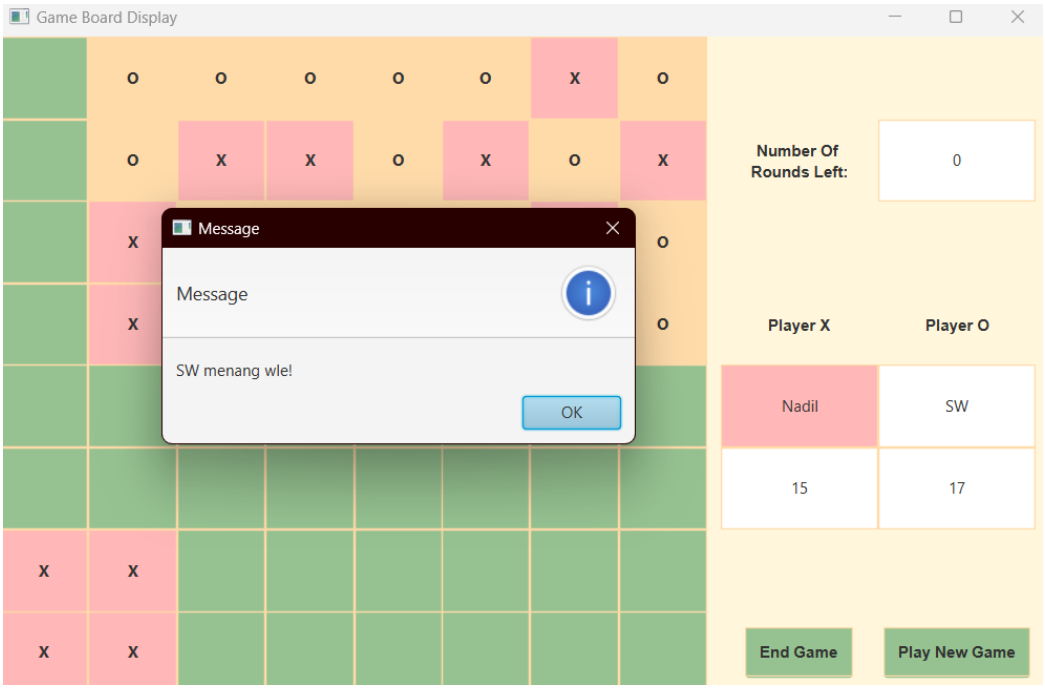
5.2. Bot Hill-climbing with Sideways Move vs Manusia

5.2.1. Pengujian Pertama (28 ronde)



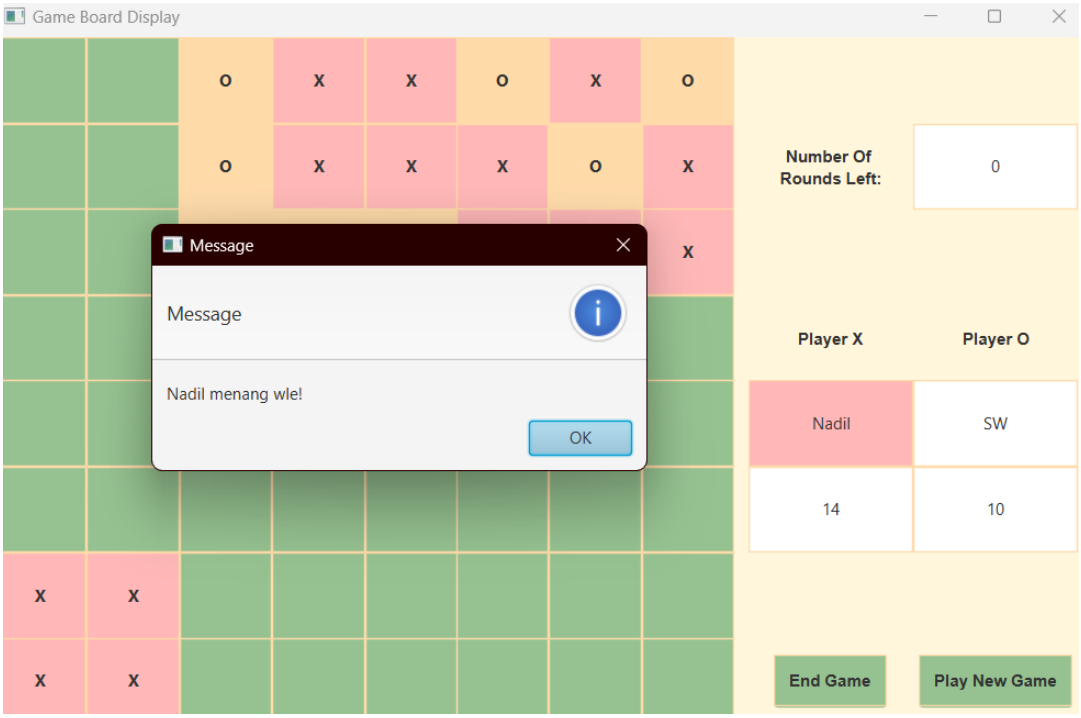
Gambar 5.2.1. Hasil Pengujian 1 antara Manusia vs Bot Hill Climbing with Sideways Move

5.2.2. Pengujian Kedua (12 ronde)



Gambar 5.2.2. Hasil Pengujian 1 antara Manusia vs Bot Hill Climbing with Sideways Move

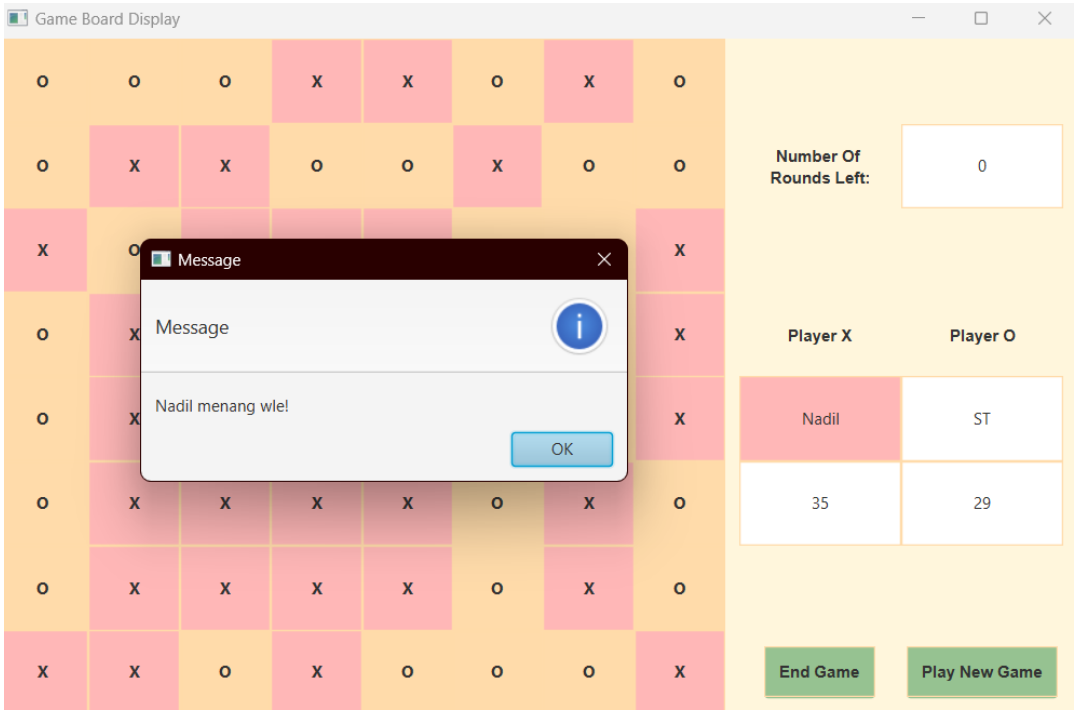
5.2.3. Pengujian Ketiga (8 ronde)



Gambar 5.2.3. Hasil Pengujian 1 antara Manusia vs Bot Hill Climbing with Sideways Move

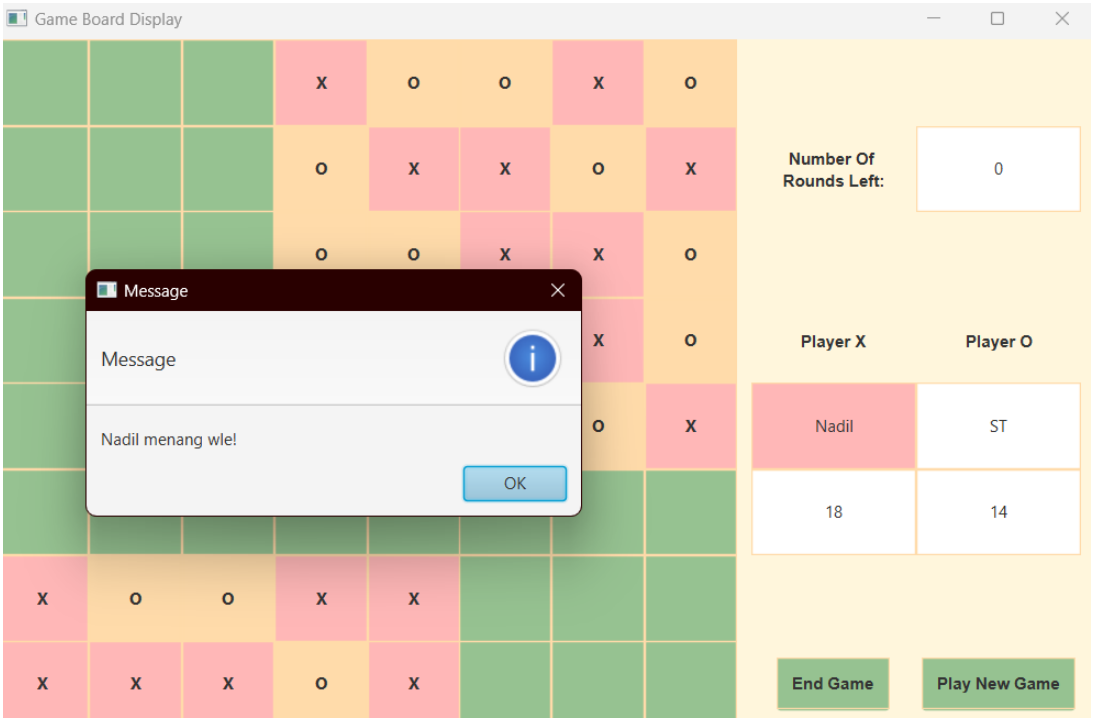
5.3. Bot Stochastic Hill-climb vs Manusia

5.3.1. Pengujian Pertama (28 ronde)



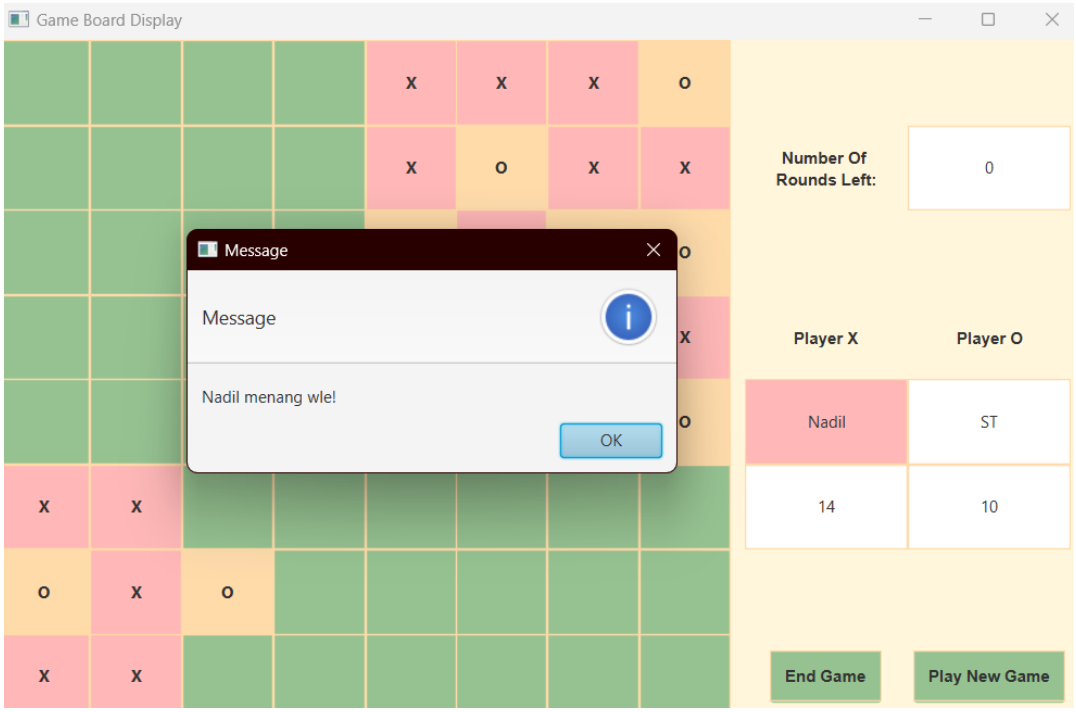
Gambar 5.3.1. Hasil Pengujian 1 antara Manusia vs Bot Stochastic Hill Climbing

5.3.2. Pengujian Kedua (12 ronde)



Gambar 5.3.2. Hasil Pengujian 2 antara Manusia vs Bot Stochastic Hill Climbing

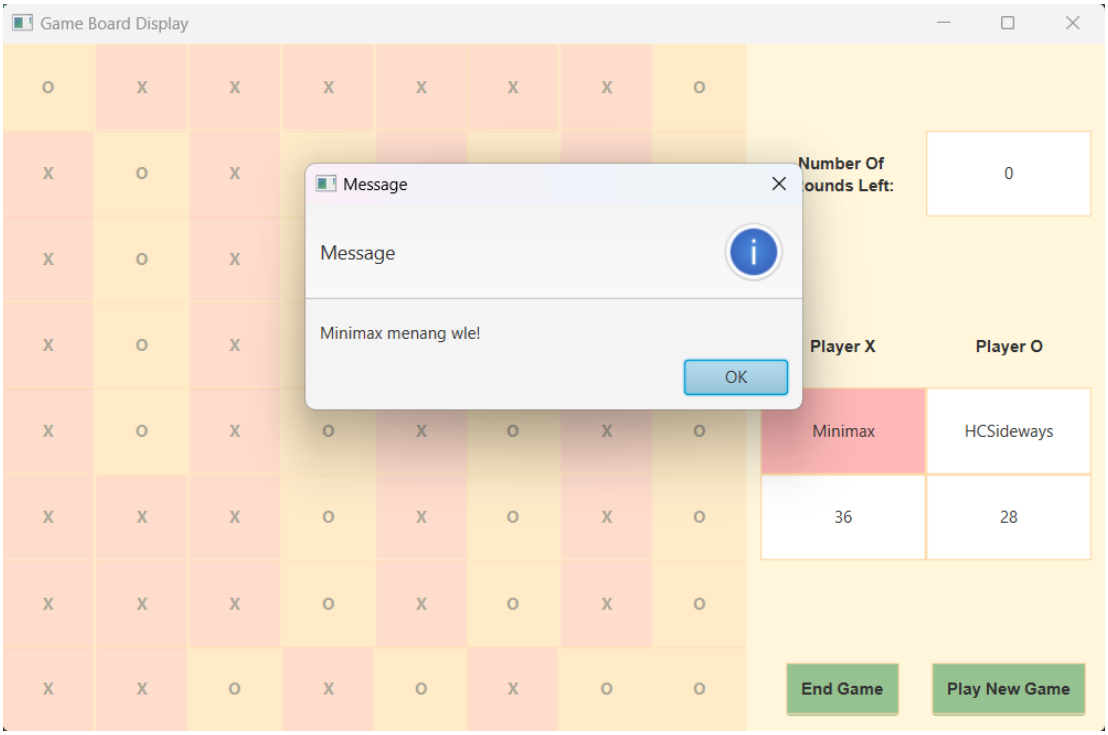
5.3.3. Pengujian Ketiga (8 ronde)



Gambar 5.3.3. Hasil Pengujian 3 antara Manusia vs Bot Stochastic Hill Climbing

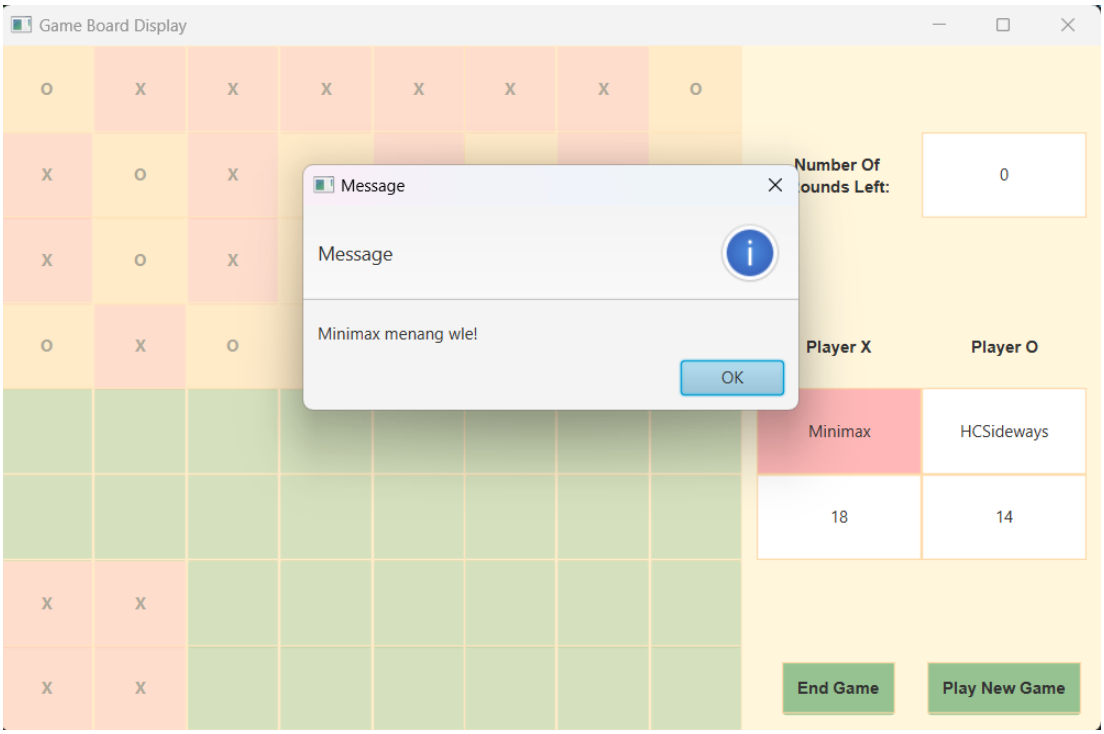
5.4. Bot Minimax vs Bot Hill-climbing with sideways move

5.4.1. Pengujian Pertama (28 ronde)



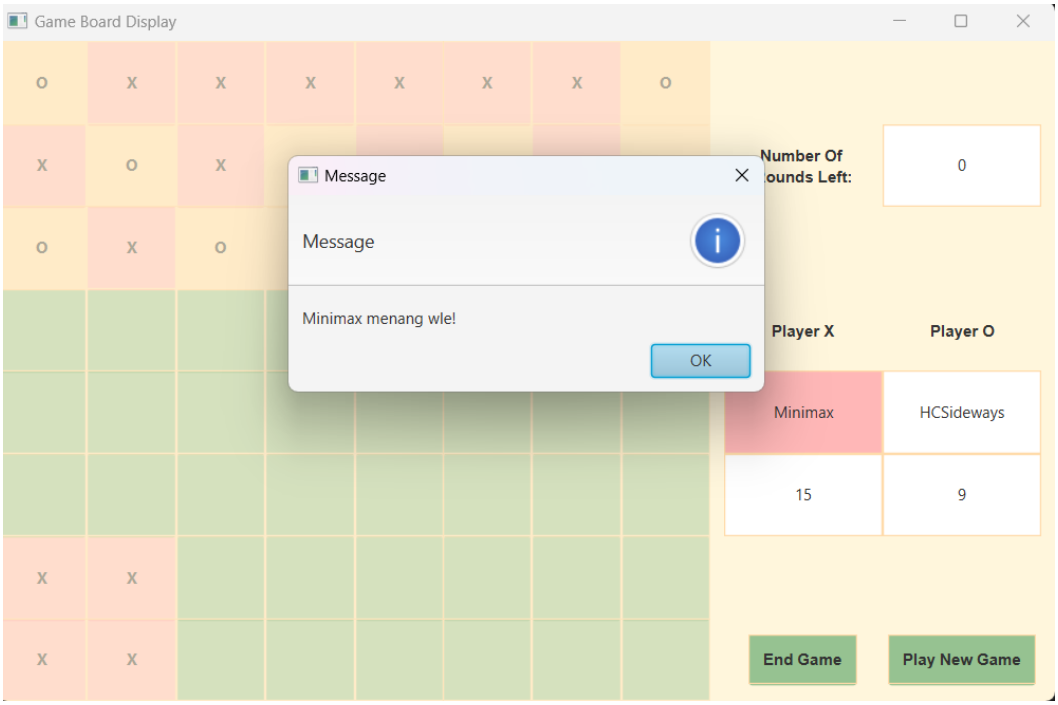
Gambar 5.4.1. Hasil Pengujian 1 antara Bot Minimax vs Bot Hill climbing with sideways move

5.4.2. Pengujian Kedua (12 Ronde)



Gambar 5.4.2. Hasil Pengujian 2 antara Bot Minimax vs Bot Hill climbing with sideways move

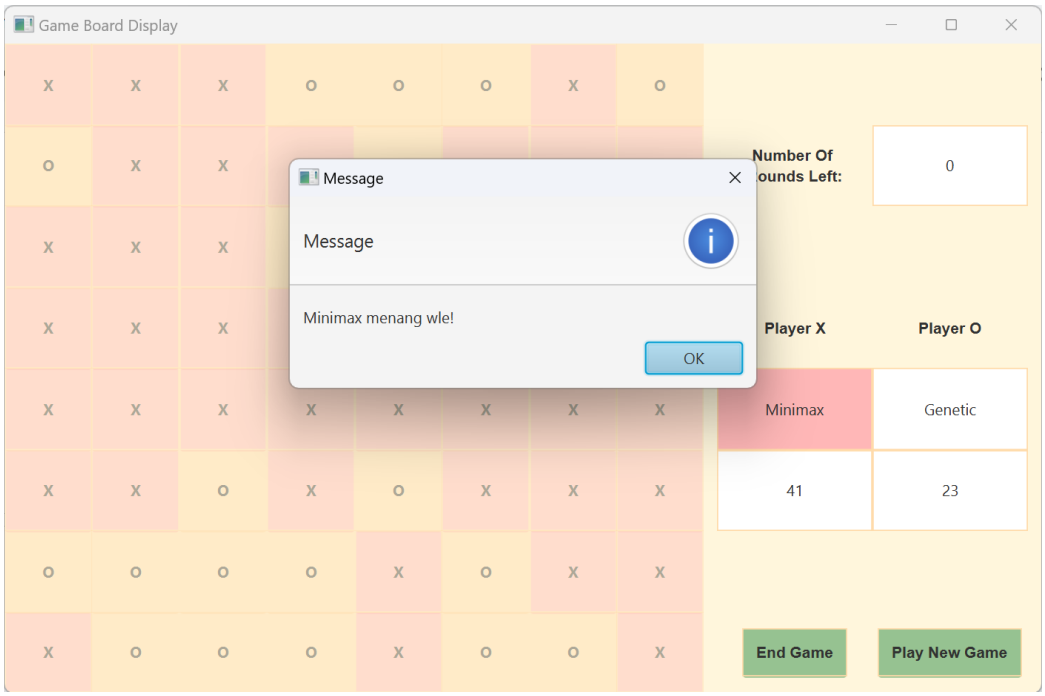
5.4.3. Pengujian Ketiga (8 Ronde)



Gambar 5.4.3. Hasil Pengujian 3 antara Bot Minimax vs Bot Local Search (Stochastic Hill Climbing)

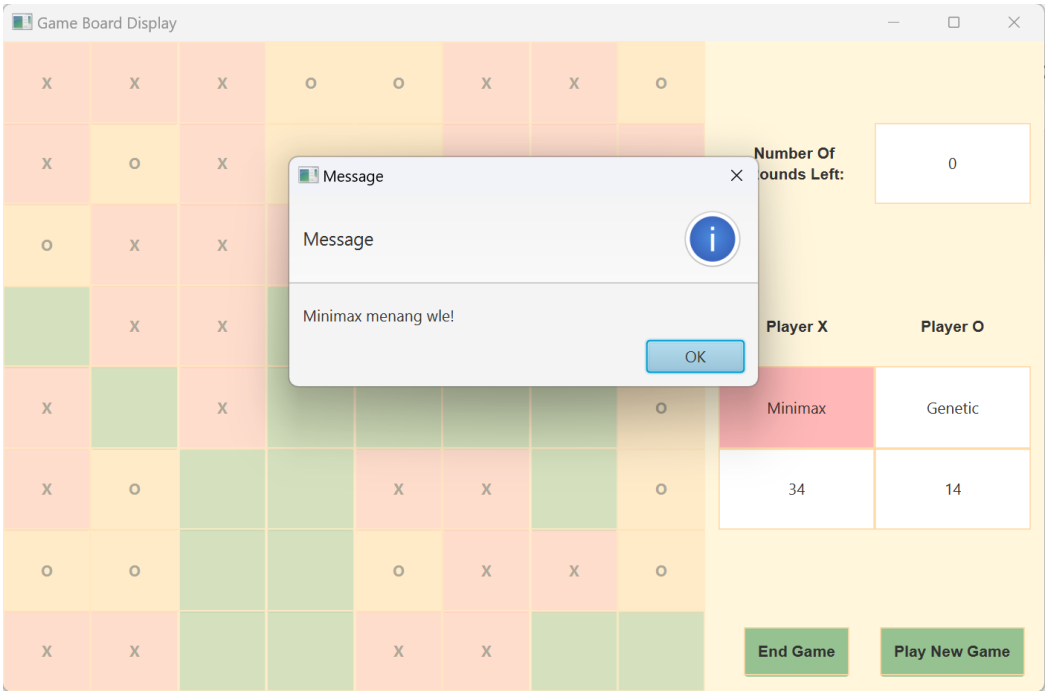
5.5. Bot Minimax vs Bot Genetic Algorithm

5.5.1. Pengujian Pertama (28 Ronde)



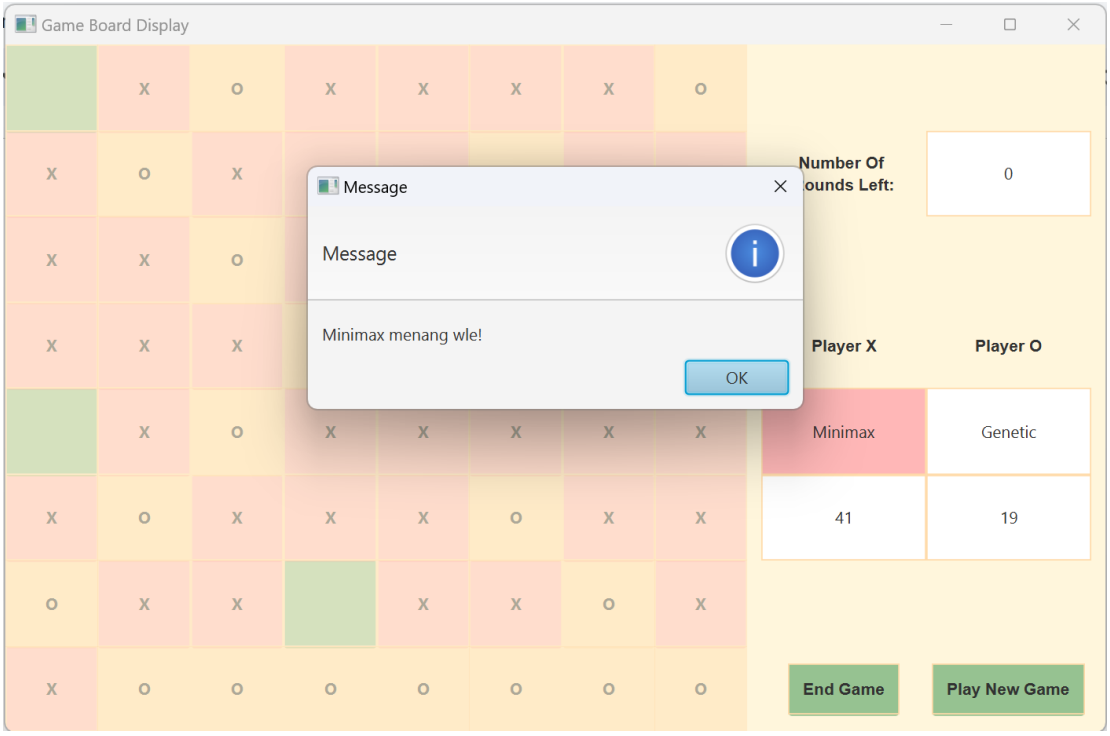
Gambar 5.5.1. Hasil Pengujian 1 antara Bot Minimax vs Bot Genetic Algorithm

5.5.2. Pengujian Kedua (12 Ronde)



Gambar 5.5.2. Hasil Pengujian 2 antara Bot Minimax vs Bot Genetic Algorithm

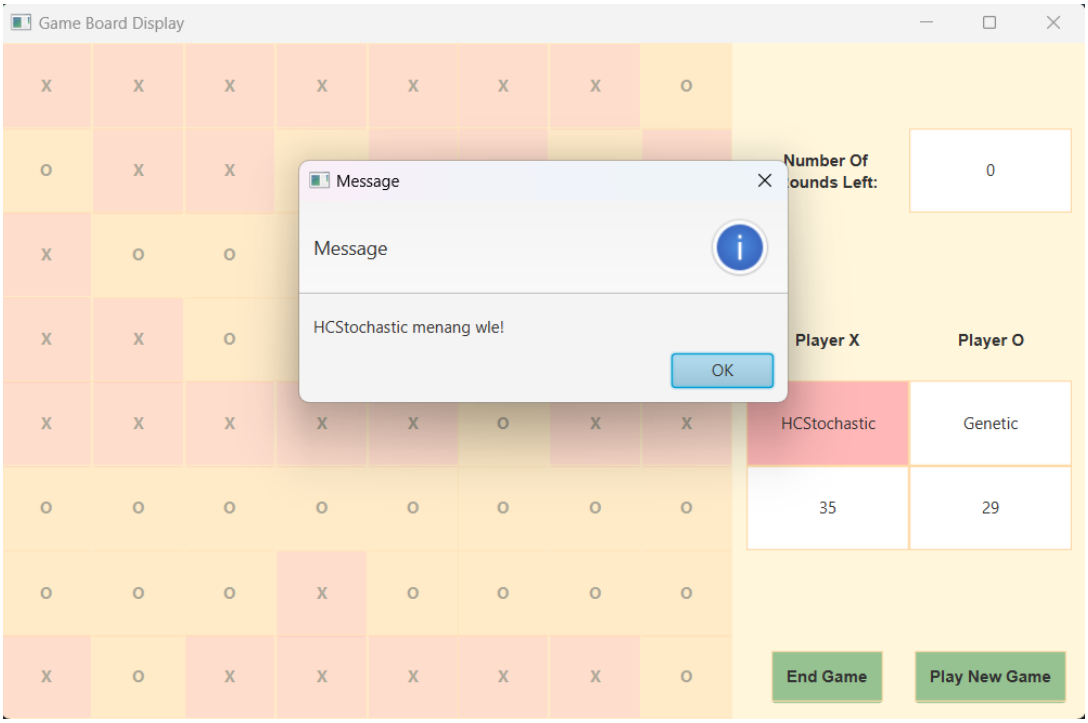
5.5.3. Pengujian Ketiga (8 Ronde)



Gambar 5.5.3. Hasil Pengujian 3 antara Bot Minimax vs Bot Genetic Algorithm

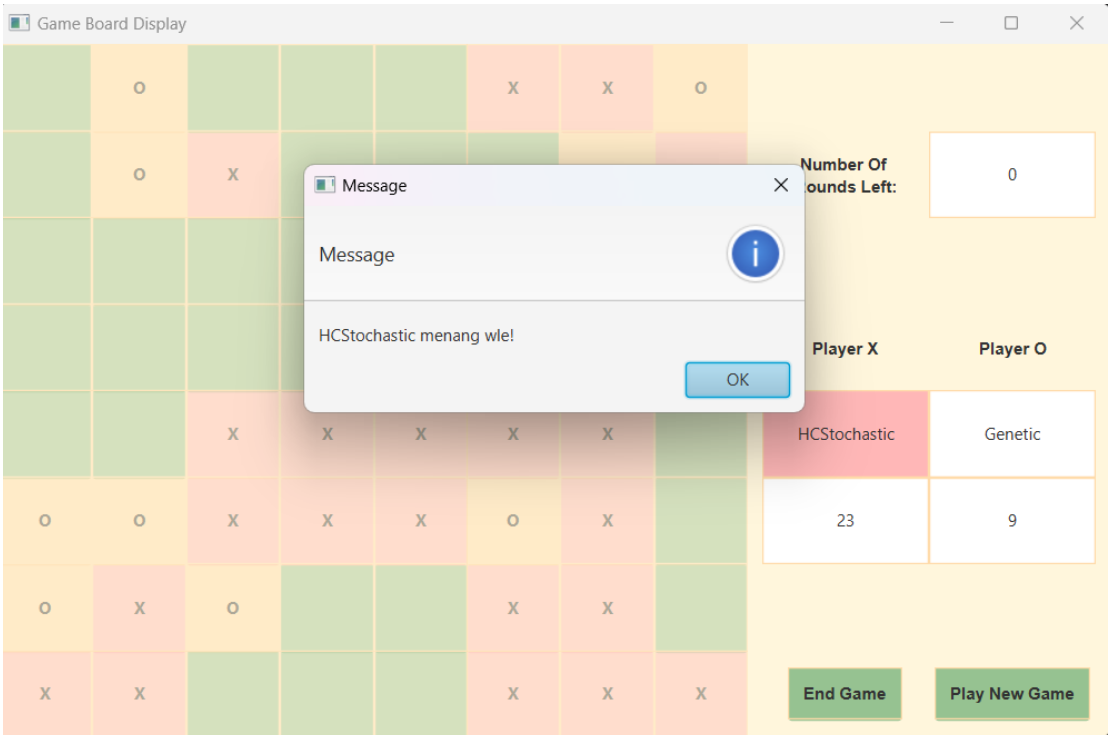
5.6. Bot Stochastic Hill Climbing vs Bot Genetic Algorithm

5.6.1. Pengujian Pertama (28 Ronde)



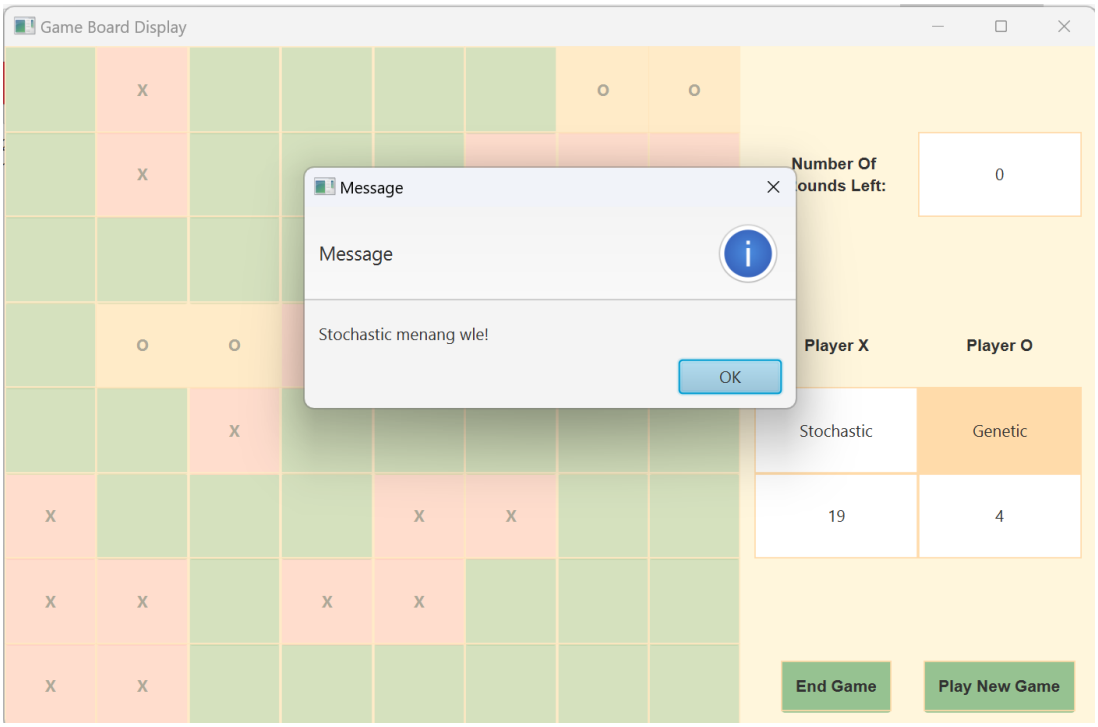
Gambar 5.6.1. Hasil Pengujian 1 antara Bot Local Search (Stochastic Hill Climbing) vs Bot Genetic Algorithm

5.6.2. Pengujian Kedua (12 Ronde)



Gambar 5.6.2. Hasil Pengujian 2 antara Bot Local Search (Stochastic Hill Climbing) vs Bot Genetic Algorithm

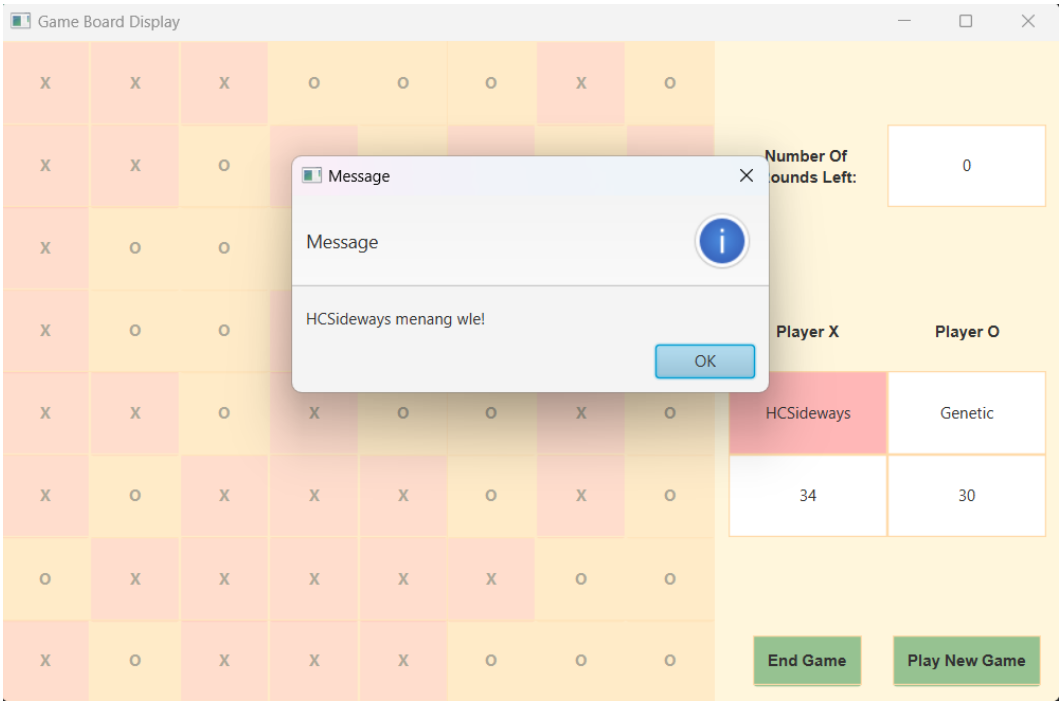
5.6.3. Pengujian Ketiga (8 Ronde)



Gambar 5.6.3. Hasil Pengujian 3 antara Bot Local Search (Stochastic Hill Climbing) vs Bot Genetic Algorithm

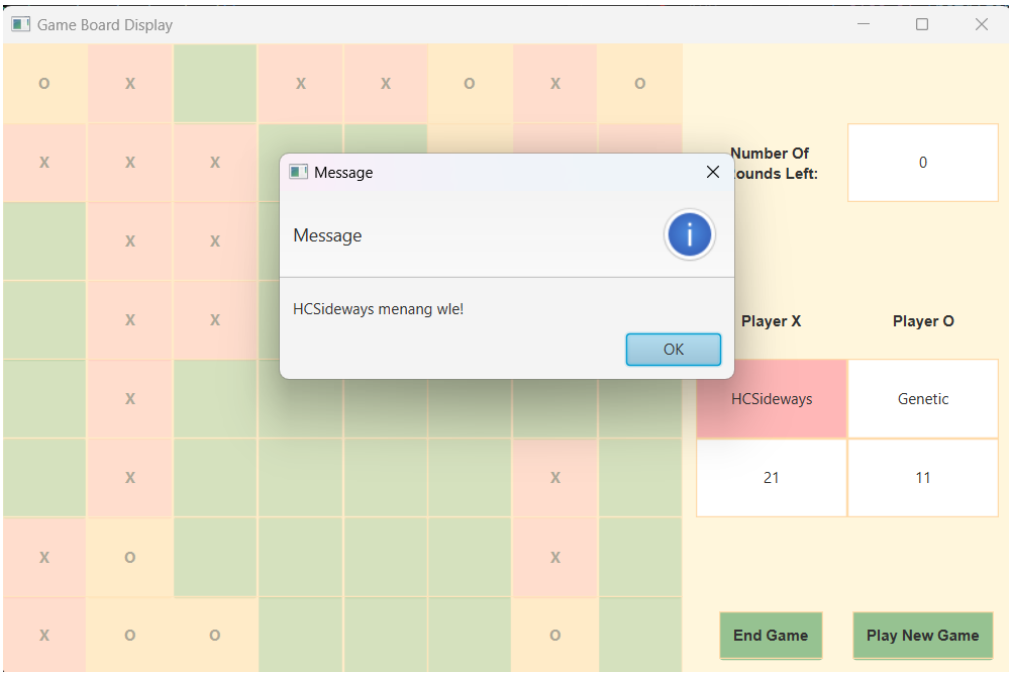
5.7. Bot Hill Climbing with Sideways Move vs Bot Genetic Algorithm

5.7.1. Pengujian Pertama (28 Ronde)



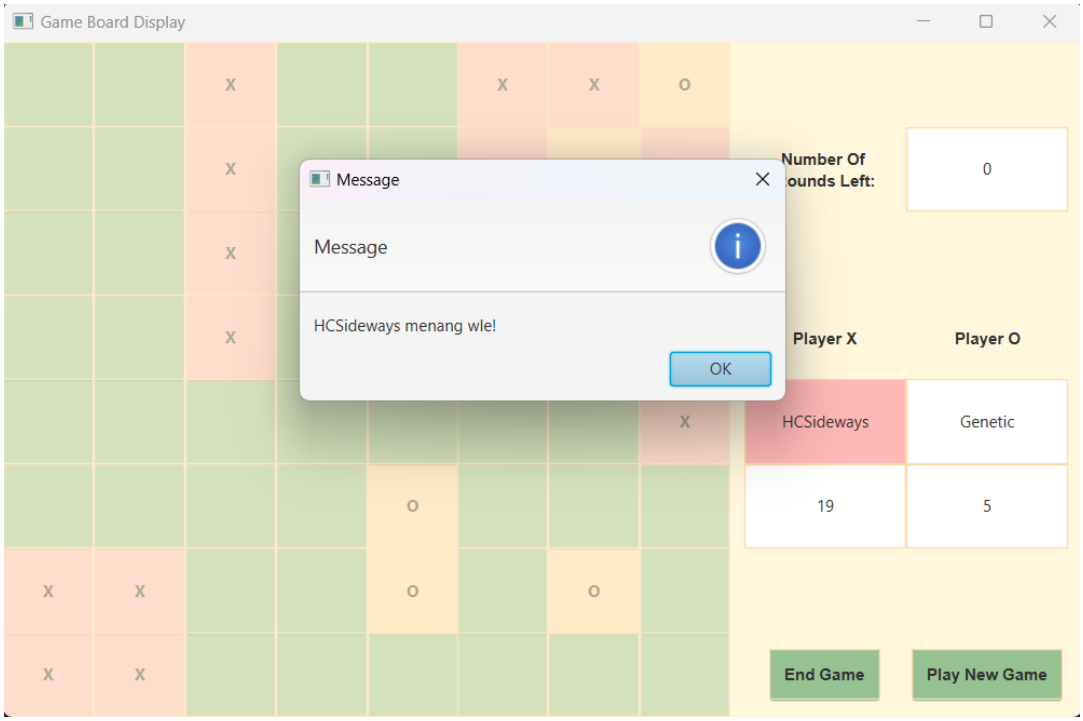
Gambar 5.7.1. Hasil Pengujian 1 antara Bot Hill Climbing with sideways move vs Bot Genetic Algorithm

5.7.2. Pengujian Kedua (12 Ronde)



Gambar 5.7.2. Hasil Pengujian 2 antara Bot Hill Climbing with sideways move vs Bot Genetic Algorithm

5.7.3. Pengujian Ketiga (8 Ronde)



Gambar 5.7.3. Hasil Pengujian 3 antara Bot Hill Climbing with sideways move vs Bot Genetic Algorithm

5.8. Tabel hasil jumlah kemenangan

Tabel 5.1. Hasil Jumlah Kemenangan

Minimax	Stochastic Hill Climbing	Hill Climbing with Sideways Move	Genetic Algorithm	Manusia
11	3	4	0	5

Tabel 5.2. Presentase Kemanangan

Minimax	Stochastic Hill Climbing	Hill Climbing with Sideways Move	Genetic Algorithm	Manusia
47.8%	13.04%	17.39%	0%	21.73%

VI. Kontribusi Anggota

NIM	Tugas
13521005	Perubahan *game engine* untuk *game mode*, *hill-climbing with sideways move*
13521006	*stochastic hill-climb*, UI *game engine*, laporan
13521019	*Minimax with alpha beta pruning*
13521024	Perubahan *game engine* untuk *game mode*, *genetic-minimax algorithm, stochastic hill-climb*