LAPORAN TUGAS BESAR 1

Minimax Algorithm and Alpha Beta Pruning in Adjacency Strategy Game

Anggota Kelompok:
Ezra Maringan Christian Mastra Hutagaol - 13521073
Christian Albert Hasiholan - 13521078
Tobias Natalio Sianipar - 13521090
Zidane Firzatullah - 13521163

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
BANDUNG
Oktober 2023

1. Penjelasan Objective Function

Objective Function yang digunakan adalah menghitung *score point* dengan cara banyak simbol agen dikurang simbol lawan. Perhitungan *score point* dilakukan karena tujuan akhir dari game adalah memaksimalkan *score* agen dan meminimalkan *score* lawan.

2. Implementasi *Minimax Alpha Beta Pruning*

Proses pencarian dengan menggunakan *Minimax Alpha Beta Pruning* dapat dilakukan dengan membandingkan nilai pada tiap kemungkinan langkah yang dilakukan pemain dan bot. Nilai ini diperoleh dari hasil pengurangan jumlah kotak yang terisi ‘X’ milik pemain dengan jumlah kotak yang terisi ‘O’ milik bot. Pencarian nilai tersebut dilakukan secara rekursif sampai pada beberapa *endgame state* pertama. Selanjutnya nilai dari *node* pertama yang mencapai *leaf* akan digunakan sebagai *alpha* atau *beta* untuk memangkas pencarian pada child *node* lain dari *parent node* tersebut. Nilai *alpha* didapatkan dengan menyimpan nilai maksimum suatu *node* yang telah ditelusuri, sedangkan nilai *beta* didapatkan dengan menyimpan nilai minimum *node* yang telah ditelusuri. Ketika permainan dalam giliran pemain, maka pencarian dilakukan bergiliran secara *Max-Min* dengan *root node* akan menyimpan *alpha* atau nilai maksimum, sedangkan ketika giliran bot, pencarian bergiliran secara *Min-Max* dengan *root node* menyimpan *beta* atau nilai minimum.

Method AlphaBetaBot ( )

```java
public int[] move() {
    long startTime = System.nanoTime();
    Integer[] nextMove = minmax( isMax: true, depth: 3, new Pair<Integer[], Integer[][]>(new Integer[]{0,0}, boardState), a: -10000, b: 10000)
    System.out.println("Next move is:" + nextMove[0] + nextMove[1]);
    long endTime = System.nanoTime();
    long elapsedTime = (endTime - startTime) / 1_000_000; // Convert nanoseconds to milliseconds
    System.out.println("Execution time: " + elapsedTime + " ms");
    return new int[]{nextMove[0], nextMove[1]};
}
```

Method minmax ( )

```java
public Pair<Integer[], Integer[][]> minmax(boolean isMax, Integer depth, Pair<Integer[], Integer[][]> state, Integer a, Integer b) {
    List<Pair<Integer[], Integer[][]>> neighbors = createSuccessor(isMax, getEmptyBlock(state.getValue()), state.getValue());
      Comparator<Pair<Integer[], Integer[][]>> pairComparator = new Comparator<Pair<Integer[], Integer[][]>>() {
          @Override
          public int compare(Pair<Integer[], Integer[][]> pair1, Pair<Integer[], Integer[][]> pair2) {
              if(isMax){
                  return Integer.compare(getBoardScore(pair2.getValue()), getBoardScore(pair1.getValue()));
              } else {
                  return Integer.compare(getBoardScore(pair1.getValue()), getBoardScore(pair2.getValue()));
              }
          }
      };
      Collections.sort(neighbors, pairComparator);
    if(depth == 0 || neighbors.size() == 0){ return state; }
    Pair<Integer[], Integer[][]> chosenState = neighbors.get(0);
    if (Thread.currentThread().isInterrupted()) {
        return null;

    }
```

Method createSuccessor

```java
public List<Pair<Integer[], Integer[][]>> createSuccessor(boolean isBot, List<Integer[]> emptyBlock, Integer[][] state) {
    List<Pair<Integer[], Integer[][]>> successor = new ArrayList<>();
    int value = -1;
    if(!isBot) { value = -value; }
    for (Integer[] move: emptyBlock) {
        Integer[][] nextBoardState = Arrays.stream(state).map(Integer[]::clone).toArray(Integer[][]::new);

        nextBoardState[move[0]][move[1]] = -1;
        if(move[1] - 1 >= 0) {
            if (nextBoardState[move[0]][move[1] - 1] == -value) {
                nextBoardState[move[0]][move[1] - 1] = value;
            }
        }
        if(move[1] + 1 < 8) {
            if (nextBoardState[move[0]][move[1] + 1] == -value) {
                nextBoardState[move[0]][move[1] + 1] = value;
            }
        }
        if(move[0] - 1 >= 0) {
            if (nextBoardState[move[0] - 1][move[1]] == -value) {
                nextBoardState[move[0] - 1][move[1]] = value;
            }
        }
```

```
        if(move[0] + 1 < 8) {
            if (nextBoardState[move[0] + 1][move[1]] == -value) {
                nextBoardState[move[0] + 1][move[1]] = value;
            }
        }

        successor.add(new Pair<>(move, nextBoardState));
    }

    return successor;
}
```

Method getEmptyBlock ( )

```
public List<Integer[]> getEmptyBlock(Integer[][] state) {
    List<Integer[]> emptyBlock = new ArrayList<>();

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (state[i][j] == 0) {
                emptyBlock.add(new Integer[]{i, j});
            }
        }
    }

    return emptyBlock;
}
```

Method getBoardScore ( )

```java
public int getBoardScore(Integer[][] board) {
    int score = 0;

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (board[i][j] == -1) {
                score += 1;
            }
            if (board[i][j] == 1) {
                score -= 1;
            }
        }
    }

    return score;
}
```

Method setBoardState ( )

```java
public void setBoardState(Button[][] buttons) {
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (buttons[i][j].getText().equals("")) {
                boardState[i][j] = 0;
            } else if (buttons[i][j].getText().equals("X")) {
                boardState[i][j] = 1;
            }
            else {
                boardState[i][j] = -1;
            }
        }
    }
}
```

3. Implementasi *Local Search*

Algoritma *Local Search* yang akan digunakan pada permainan ini adalah *hill climbing*.
Berikut merupakan langkah langkah dari algoritma ini :

1. State awal berupa keadaan permainan saat itu dengan papan yang sudah terbentuk
2. Bot akan membuat *successor* berupa semua kemungkinan gerakan yang bisa dilakukan oleh bot saat itu
3. Akan dilakukan loop sebanyak semua kemungkinan gerakan, dan pada tiap loop agen akan mensimulasikan state baru
4. Tiap kemungkinan state akan dihitung nilainya dengan fungsi objektif dan kemudian dibandingkan.
5. Agen menemukan *neighbor* dengan nilai fungsi objektif tertinggi lalu akan mengembalikan aksi yang akan dilakukan sesuai state game pada *neighbor* tersebut
6. Pencarian selesai lalu giliran lawan. Local search akan dilakukan lagi dari langkah 1

Adapun beberapa method yang digunakan untuk bot ini sebagai berikut :

Method move ( )

```
public int[] move() {
    Integer[] nextMove = hillclimbing();
    System.out.println("Next move is:" + nextMove[0] + nextMove[1]);
    return new int[]{nextMove[0], nextMove[1]};
}
```

Method setBoardState ( )

```java
public void setBoardState(Button[][] buttons) {
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (buttons[i][j].getText().equals("")) {
                this.boardState[i][j] = 0;
            } else if (buttons[i][j].getText().equals("X")) {
                this.boardState[i][j] = 1;
            }
            else {
                this.boardState[i][j] = -1;
            }
        }
    }
}
```

Method getEmptyBlock ( )

```java
public List<Integer[]> getEmptyBlock() {
    List<Integer[]> emptyBlock = new ArrayList<>();

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (boardState[i][j] == 0) {
                emptyBlock.add(new Integer[]{i, j});
            }
        }
    }

    return emptyBlock;
}
```

Method hillclimbing ( )

```java
public Integer[] hillclimbing() {
    List<Integer[]> emptyBlock = new ArrayList<>(getEmptyBlock());
    List<Pair<Integer[], Integer[][]>> successorList = new ArrayList<>(createSuccessor(emptyBlock));
    Integer[][] nextBoard = boardState;
    Integer[] nextMove = new Integer[]{(int) (Math.random()*8), (int) (Math.random()*8)};
    int limit = 10;

    while(true) {
        Pair<Integer[], Integer[][]> neighbor = getNeighbor(successorList);
        if (getBoardScore(neighbor.getValue()) < getBoardScore(nextBoard)) {
            return nextMove;
        }
        else if (getBoardScore(neighbor.getValue()) == getBoardScore(nextBoard)) {
            limit -= 1;
        }
        else {
            limit = 10;
            nextBoard = neighbor.getValue();
            nextMove = neighbor.getKey();
        }

        if (limit == 0) {
            return nextMove;
        }
    }
}
```

Method createSuccessor ( )

```java
public List<Pair<Integer[], Integer[][]>> createSuccessor(List<Integer[]> emptyBlock) {
    List<Pair<Integer[], Integer[][]>> successor = new ArrayList<>();

    for (Integer[] move: emptyBlock) {
        Integer[][] nextBoardState = Arrays.stream(boardState).map(Integer[]::clone).toArray(Integer[][]::new);

        nextBoardState[move[0]][move[1]] = -1;
        if(move[1] - 1 >= 0) {
            if (nextBoardState[move[0]][move[1] - 1] == 1) {
                nextBoardState[move[0]][move[1] - 1] = -1;
            }
        }
        if(move[1] + 1 < 8) {
            if (nextBoardState[move[0]][move[1] + 1] == 1) {
                nextBoardState[move[0]][move[1] + 1] = -1;
            }
        }
        if(move[0] - 1 >= 0) {
            if (nextBoardState[move[0] - 1][move[1]] == 1) {
                nextBoardState[move[0] - 1][move[1]] = -1;
            }
        }
        if(move[0] + 1 < 8) {
            if (nextBoardState[move[0] + 1][move[1]] == 1) {
                nextBoardState[move[0] + 1][move[1]] = -1;
            }
        }

        successor.add(new Pair<>(move, nextBoardState));
    }

    return successor;
```

Method getBoardScore ( )

```java
public int getBoardScore(Integer[][] board) {
    int score = 0;

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (board[i][j] == -1) {
                score += 1;
            }
            if (board[i][j] == 1) {
                score -= 1;
            }
        }
    }

    return score;
}
```

Method getNeighbor ( )

```java
    public Pair<Integer[], Integer[][]> getNeighbor(List<Pair<Integer[], Integer[][]>> successorList) {
        Collections.shuffle(successorList);

        Pair<Integer[], Integer[][]> neighbor = successorList.get(0);
        int currentScore = getBoardScore(neighbor.getValue());
        int successorScore;

        for (Pair<Integer[], Integer[][]> successor: successorList) {
            successorScore = getBoardScore(successor.getValue());
            if (currentScore < successorScore) {
                neighbor = successor;
                currentScore = successorScore;
            }
        }

        return neighbor;
    }
}
```

Adapun algoritma local search yang dipilih adalah *hill-climbing Steepest Ascent*. Pemilihan ini karena *hill-climbing steepest ascent* lebih efisien digunakan dan state yang dipilih merupakan state dengan *objective function* tertinggi.

4. Implementasi *Genetic Algorithm*

Penggunaan genetic algorithm dalam menyelesaikan permasalahan *adversarial* bisa dilakukan dengan memanfaatkan konsep game tree, genetic algorithm, dan minimax secara bersamaan. Proses genetic algorithm dilakukan untuk mendapatkan langkah selanjutnya yang harus diambil, bukan untuk menyelesaikan permasalahan dari awal hingga akhir

Beberapa penjelasan terkait konsep yang digunakan adalah sebagai berikut:

1. Game Tree

   Game tree di-inisialisasi di awal untuk seluruh kemungkinan langkah seperti di algoritma minimax. Tambahannya, seluruh edge akan diberi sebuah nomor untuk proses encoding kromosom. Penomoran diberikan secara *incremental* dari angka 1 untuk *direct children* dari *parent node*, duplikasi penomoran untuk node dari parent yang berbeda sengaja dilakukan.
2. Encoding Kromosom

   Sebuah kromosom didefinisikan sebagai urutan bilangan yang merepresentasikan edge yang dipilih untuk mencapai leaf. Contoh: 145 artinya memilih edge 1 (depth 0 ke depth 1) -> 4 (depth 1 ke depth 2) -> 5 (depth 2 ke depth 3).
3. Nilai Fitness

   Nilai fitness didefinisikan sebagai seberapa tinggi nilai minimax sebuah *leaf* (terminal node) bisa dipilih oleh parent node. Artinya, leaf terbaik adalah leaf dengan nilai minimax yang dipilih hingga ke root node. Rumusan fungsi fitness adalah H - R + 1. H menyatakan tinggi game tree dan R menyatakan ketinggian yang diraih nilai minimax leaf.
4. Crossover

   Crossover adalah proses penggabungan 2 kromosom menjadi generasi baru dengan cara memilih 1 index dimulainya penukaran, dalam permasalahan ini index diasumsikan dimulai dari 1. Sebagai contoh 2 kromosom 12345 dan 67891 dilakukan crossover dengan index 3, maka kromosom hasil crossover adalah 12391 dan 67845.
5. Mutasi

   Mutasi adalah proses random untuk menukar sebuah gen dengan gen baru dengan harapan mendapatkan kromosom yang lebih baik. Proses mutasi dilakukan dengan probabilitas yang sangat rendah, misalnya 0.5% atau 0.1%. Contoh: kromosom 12345 terjadi mutasi di gen dengan index 3 (random), maka mungkin saja kromosom tersebut menjadi 12945.
6. Reservation Tree

   Reservation tree menggambarkan populasi kromosom. Seluruh populasi kromosom, initial population atau hasil crossover & mutation, akan digambarkan di sebuah reservation tree dengan cara menggambar node dan edge yang dipilih oleh

kromosom. Reservation tree akan digunakan untuk kepentingan evaluasi populasi, artinya minimax value dan fitness value akan hanya dihitung untuk node yang berada di dalam reservation tree.

7. Objective Function dan Minimax Function

   Konsep ini sama seperti konsep yang digunakan di algoritma adversarial search minimax. Adapun objective function yang digunakan sama seperti yang digunakan pada persoalan minimax, obj(s, p): total_mark(p) - total_mark(enemy(p)).

Berikut merupakan step yang perlu dilakukan:

1. **Tentukan banyaknya total langkah player dan musuh (ply) yang ingin dievaluasi dan banyaknya generasi yang diinginkan**. Pendekatan ini digunakan karena genetic algorithm hanya bisa menentukan 1 langkah selanjutnya, sehingga evaluasi hingga game selesai tidak terlalu dibutuhkan dan akan memakan waktu yang jauh lebih lama.
2. **Populasikan game tree**.
3. **Tentukan nilai probabilitas mutasi dan crossover.** Nilai ini akan digunakan dalam proses pembentukan generasi selanjutnya.
4. **Generate populasi awal dengan jumlah N kromosom**.
5. **Buat reservation tree kosong**.
6. **Lakukan proses mutasi**.
7. **Gambarkan seluruh populasi ke dalam reservation tree**. Perlu diperhatikan setiap kromosom tidak menggambar seluruh edge dan node yang dimiliki, tetapi hanya menambahkan edge dan node yang belum ada di dalam reservation tree.
8. **Evaluasi fitness value seluruh populasi di dalam reservation tree**.
9. **Lakukan proses crossover untuk menghasilkan generasi berikutnya**.
10. **Jika sudah mendapatkan angka generasi yang diinginkan maka berhenti. Jika belum, ulang proses dari langkah 5**.

Adapun method yang digunakan untuk bot ini yaitu :

Method setBoardState( )

```
public void setBoardState(Button[][] buttons) {
    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (buttons[i][j].getText().equals("")) {
                this.boardState[i][j] = 0;
            } else if (buttons[i][j].getText().equals("X")) {
                this.boardState[i][j] = 1;
            }
            else {
                this.boardState[i][j] = -1;
            }
        }
    }
}
```

Method move ( )

```java
public int[] move() {
    int turnCount = Math.min(8, GeneticUtil.countEmptyBlock(boardState)); // should be compared (min function) with emptyBlocks
    List<String> population = new ArrayList<>();

    for (int i = 0; i < turnCount; ++i) {
        population.add(go.generateRandomChromosome(turnCount));
    }

    Node root = null;
    int generation = 2;
    for (int i = 0; i < generation; ++i) {
        root = go.formTree(population.get(0));
        for (int j = 1; j < population.size(); ++j) {
            go.mergeTree(population.get(j), root);
        }
        dfs(root, boardState);
        max(root);
        initFitness(root, turnCount);
        population = reproduction(root.getLeafNodes(), population.size());
    }
    List<Node> leafs = root.getLeafNodes();
    List<Node> sortedLeaf = leafs.stream()
            .sorted((n1, n2) -> Integer.compare(n2.getFitness(), n1.getFitness()))
            .limit( maxSize: 1)
            .toList();
    List<Pair<Integer, Integer[]>> topStates = GeneticUtil.getTopStates(boardState);
    Integer[] res = topStates.get(sortedLeaf.get(0).getChromosome().charAt(0)-48).getValue();

    return new int[]{res[0], res[1]};
}
```

Method reproduction ( )

```java
public static List<String> reproduction(List<Node> leafs, int populationNumber) {
    int chosen = (int) Math.ceil((double) leafs.size() /4);
    List<Node> sortedLeaf = leafs.stream()
            .sorted((n1, n2) -> Integer.compare(n2.getFitness(), n1.getFitness()))
            .limit(chosen)
            .toList();

    List<String> population = new ArrayList<>();
    while (population.size() < populationNumber) {
        Node leaf1 = sortedLeaf.get(go.getRandomPoint(chosen)),
                leaf2 = sortedLeaf.get(go.getRandomPoint(chosen));
        String[] children = go.crossover(leaf1.getChromosome(), leaf2.getChromosome(),
                point: leaf2.getChromosome().length()/2);
        for (String child: children) {
            if (child != null) {
                population.add(child);
            }
        }
    }

    return population;
}
```

Method initFitness ( )

```
public static int max(Node root) {
    if (root.getChildren().isEmpty()) {
        return root.getValue();
    }
    int max = -99999999;
    for (int index: root.getChildren().keySet()) {
        max = Math.max(max, min(root.getChild(index)));
    }
    root.setValue(max);
    return max;
}
```

Method max( )

```
public static int max(Node root) {
    if (root.getChildren().isEmpty()) {
        return root.getValue();
    }
    int max = -99999999;
    for (int index: root.getChildren().keySet()) {
        max = Math.max(max, min(root.getChild(index)));
    }
    root.setValue(max);
    return max;
}
```

Method min ( )

```java
public static int min(Node root){
    if (root.getChildren().isEmpty()) {
        return root.getValue();
    }
    int min = 99999999;
    for (int index: root.getChildren().keySet()) {
        min = Math.min(min, max(root.getChild(index)));
    }
    root.setValue(min);
    return min;
}
```

Method dfs ( )

```java
public static void dfs(Node root, Integer[][] board) {
    if (root.getChildren().isEmpty()) {
        root.setValue(getBoardScore(board));
        return;
    }
    for (Integer i: root.getChildren().keySet()) {
        Integer[][] board_Copy = copyArr(board);
        List<Pair<Integer, Integer[]>> topStates = GeneticUtil.getTopStates(board_Copy);
        Integer[] top = topStates.get(0).getValue();
        System.out.println("top pair: " + top[0] + " " + top[1]);
        fill(board_Copy, top);
        dfs(root.getChild(i), board_Copy);
    }
}
```

Method getBoardScore ( )

```java
public static int getBoardScore(Integer[][] board) {
    int score = 0;

    for (int i = 0; i < 8; i++) {
        for (int j = 0; j < 8; j++) {
            if (board[i][j] == -1) {
                score += 1;
            }
            if (board[i][j] == 1) {
                score -= 1;
            }
        }
    }

    return score;
}
```

Method fill ( )

```java
public static void fill(Integer[][] board, Integer[] move) {
    board[move[0]][move[1]] = -1;
    if(move[1] - 1 >= 0) {
        if (board[move[0]][move[1] - 1] == 1) {
            board[move[0]][move[1] - 1] = -1;
        }
    }
    if(move[1] + 1 < 8) {
        if (board[move[0]][move[1] + 1] == 1) {
            board[move[0]][move[1] + 1] = -1;
        }
    }
    if(move[0] - 1 >= 0) {
        if (board[move[0] - 1][move[1]] == 1) {
            board[move[0] - 1][move[1]] = -1;
        }
    }
    if(move[0] + 1 < 8) {
        if (board[move[0] + 1][move[1]] == 1) {
            board[move[0] + 1][move[1]] = -1;
        }
    }
}
```

Method copyArr ( )

```java
public static Integer[][] copyArr(Integer[][] arr) {
    int numRows = arr.length;
    int numCols = arr[0].length;

    // Create a new 2D integer array of the same dimensions.
    Integer[][] copy = new Integer[numRows][numCols];

    // Copy elements from the original array to the new array.
    for (int i = 0; i < numRows; i++) {
        for (int j = 0; j < numCols; j++) {
            copy[i][j] = arr[i][j];
        }
    }

    return copy;
}
```

Method formTree ( ) dan Method mergeTree ( )

```java
public Node formTree(String chromosome) {
    Node parent = new Node(), current = parent;
    for (int i = 0; i < chromosome.length(); ++i) {
        Node next = new Node();
        current.addChild(Character.getNumericValue(chromosome.charAt(i)), next);
        next.setParent(current);
        current = next;
    }
    parent.initLeafNodes();
    current.setChromosome(chromosome);
    parent.addLeaf(current);
    return parent;
}

1 usage  zidane-itb
public Node mergeTree(String chromosome, Node treeRoot) {
    Node current = treeRoot;
    for (int i = 0; i < chromosome.length(); ++i) {
        int index = Character.getNumericValue(chromosome.charAt(i));
        if (!current.hasChild(index)) {
            Node child = new Node();
            child.setParent(current);
            current.addChild(index, child);
        }
        current = current.getChild(index);
    }
    current.setChromosome(chromosome);
    treeRoot.addLeaf(current);
    return treeRoot;
}
```

Method getElevationCount ( )

```java
public int getElevationCount(Node leaf) {
    Node current = leaf;
    int peak = 0;
    while (current.getParent() != null && leaf.getValue() == current.getValue()) {
        peak += 1;
        current = current.getParent();
    }
    return peak;
}
```

Method crossover ( )

```java
public String[] crossover(String parent1, String parent2, int point) {
    if (parent1.length() != parent2.length()) {
        throw new RuntimeException("Error, crossover point is larger than parent length");
    }
    if (point > parent1.length()) {
        throw new RuntimeException("Error, crossover point is larger than parent length");
    }
    String[] arr = new String[4];
    StringBuilder child1 = new StringBuilder(), child2 = new StringBuilder();
    for (int i = 0; i < point; ++i) {
        child1.append(parent1.charAt(i));
        child2.append(parent2.charAt(i));
    }
    for (int i = point; i < parent1.length(); ++i) {
        child1.append(parent2.charAt(i));
        child2.append(parent1.charAt(i));
    }
    arr[0] = child1.toString();
    arr[1] = child2.toString();

    // mutate
    if (mutate()) {
        int mutationPoint = getRandomPoint(parent1.length());
        child1.setCharAt(mutationPoint, child2.charAt(mutationPoint));
        arr[2] = child1.toString();

        mutationPoint = getRandomPoint(parent1.length());
        child2.setCharAt(mutationPoint, child1.charAt(mutationPoint));
        arr[3] = child2.toString();
    }

    return arr;
```

Method gerRandomPoint ( ), Method mutate( ), Method generateRandomChromosome ( )

```
    public int getRandomPoint(int length) {
        Random random = new Random();
        return random.nextInt(length);
    }

    1 usage  zidane-itb
    private boolean mutate() {
        Random random = new Random();
        double probability = random.nextDouble();

        return probability <= 0.03; // 3% mutation probability
    }

    1 usage  zidane-itb
    public String generateRandomChromosome(int length) {
        StringBuilder res = new StringBuilder();
        for (int i = length; i > 1; --i) {
            res.append((char) ('0' + getRandomPoint(i)));
        }
        return res.toString();
    }

}
```

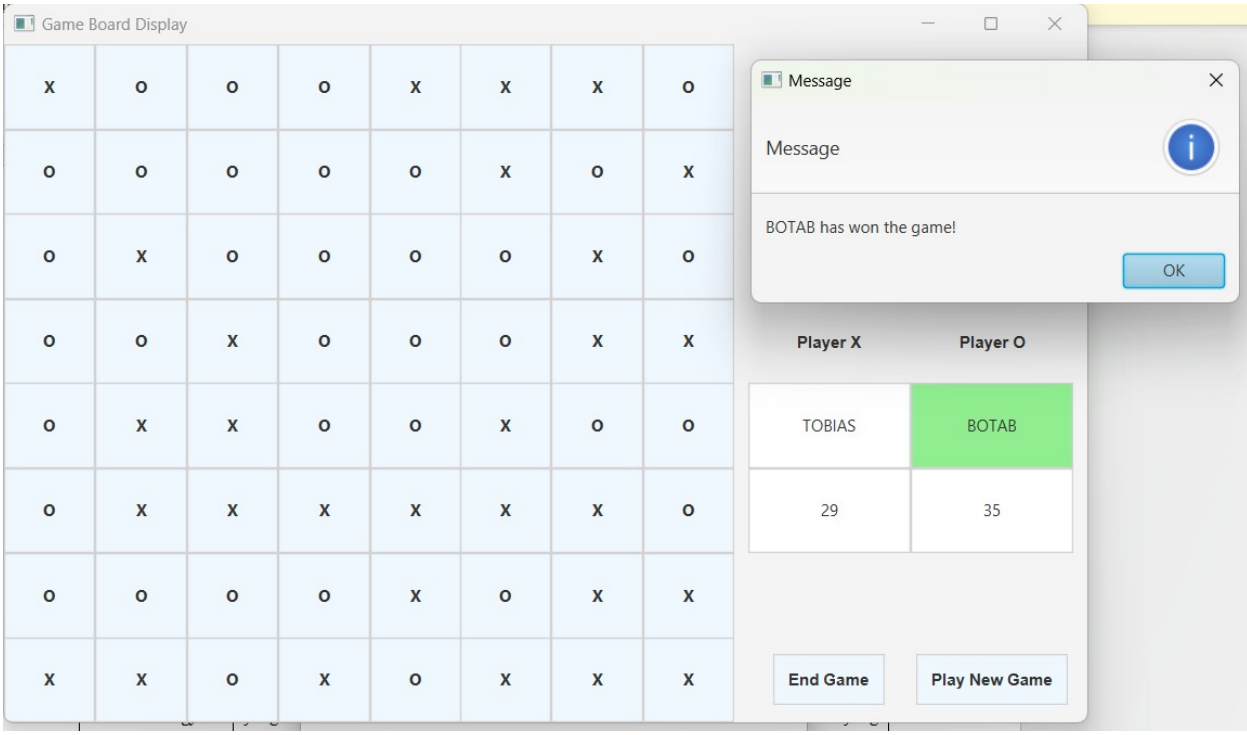
5. Hasil Pertandingan

a. Bot *Minimax* vs manusia

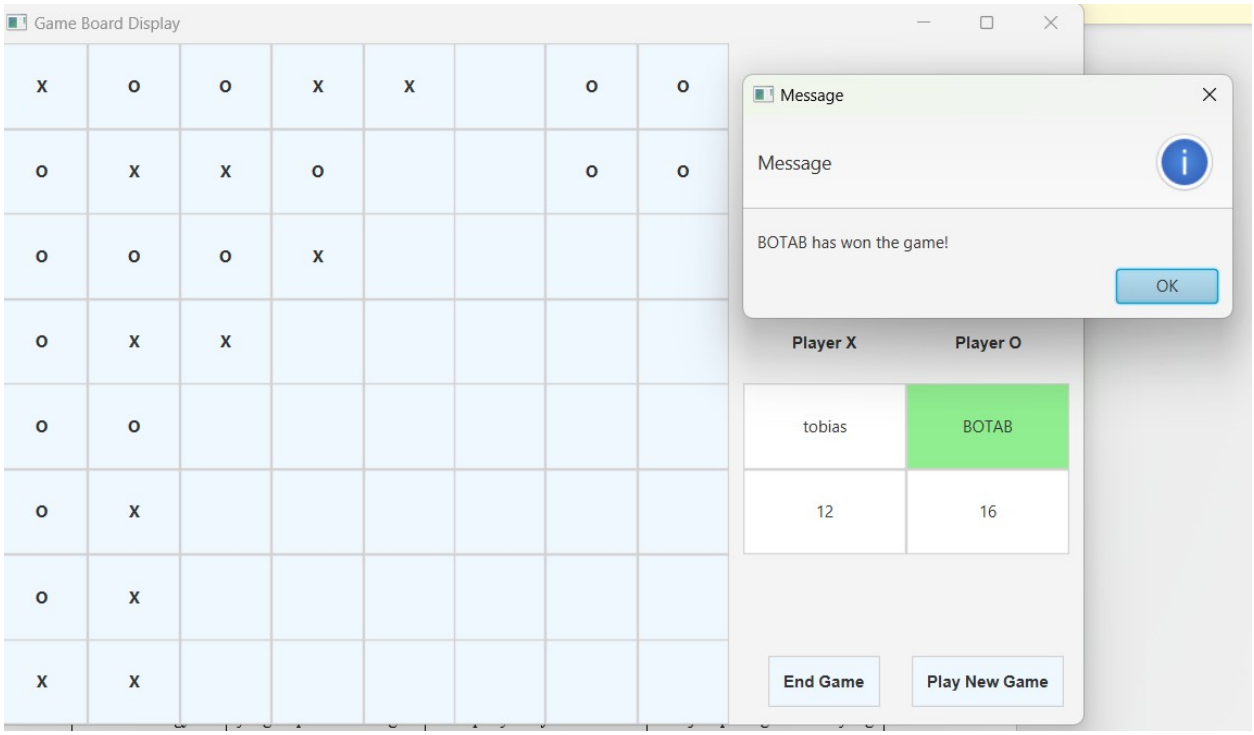
Match Desc		Player	Score
Match 1	Player 1	Bot *Minimax*	35

	Player 2	Manusia	29
Match 2	Player 1	Bot *Minimax*	16
	Player 2	Manusia	12
Match 3	Player 1	Bot *Minimax*	21
	Player 2	Manusia	15
Match 4	Player 1	Bot *Minimax*	21
	Player 2	Manusia	23
Match 5	Player 1	Bot *Minimax*	14
	Player 2	Manusia	10
Wins (Minimax vs Manusia) = 4 : 1			

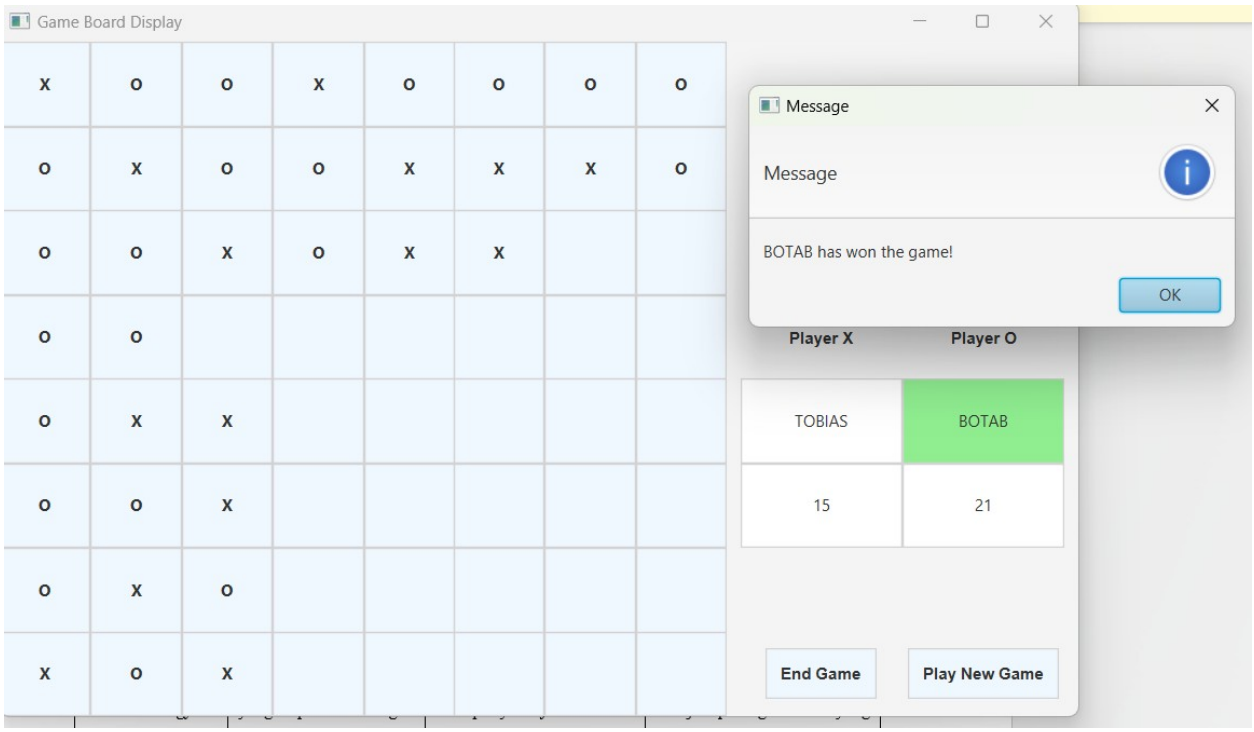
1. Maksimal Ronde



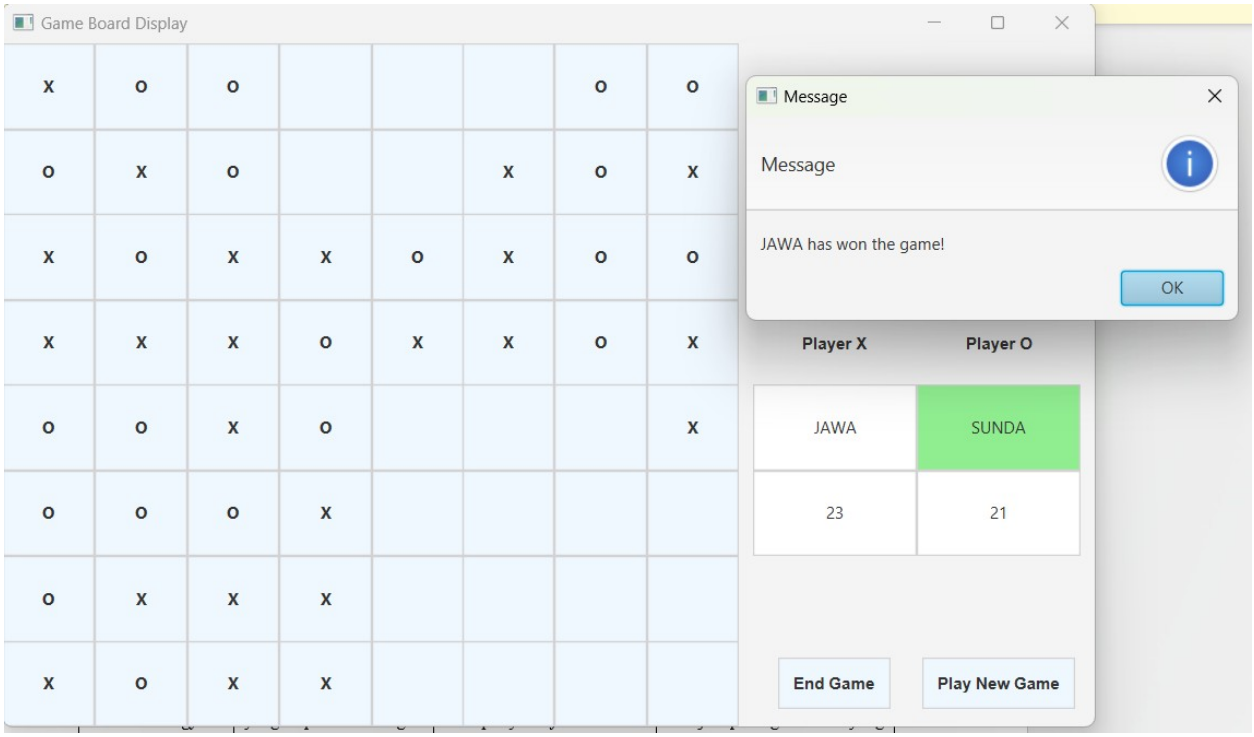
2. 10 Ronde



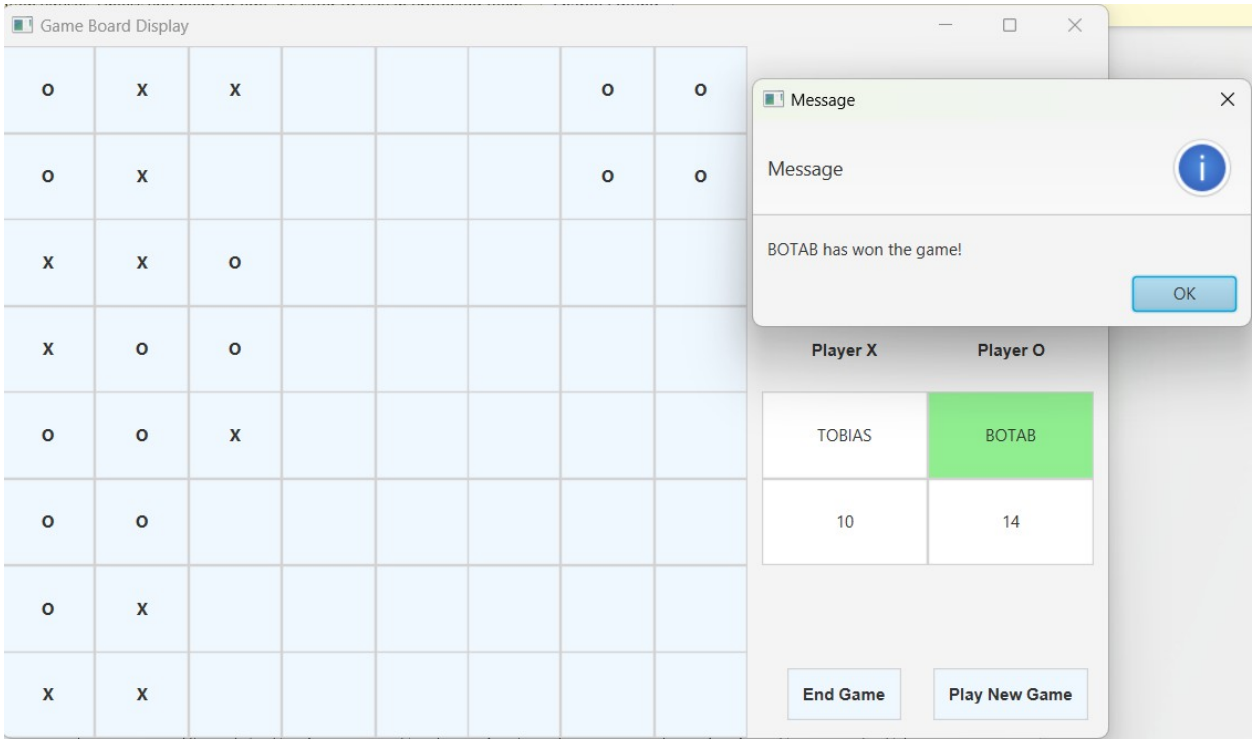
3. 14 Ronde



4. 18 Ronde



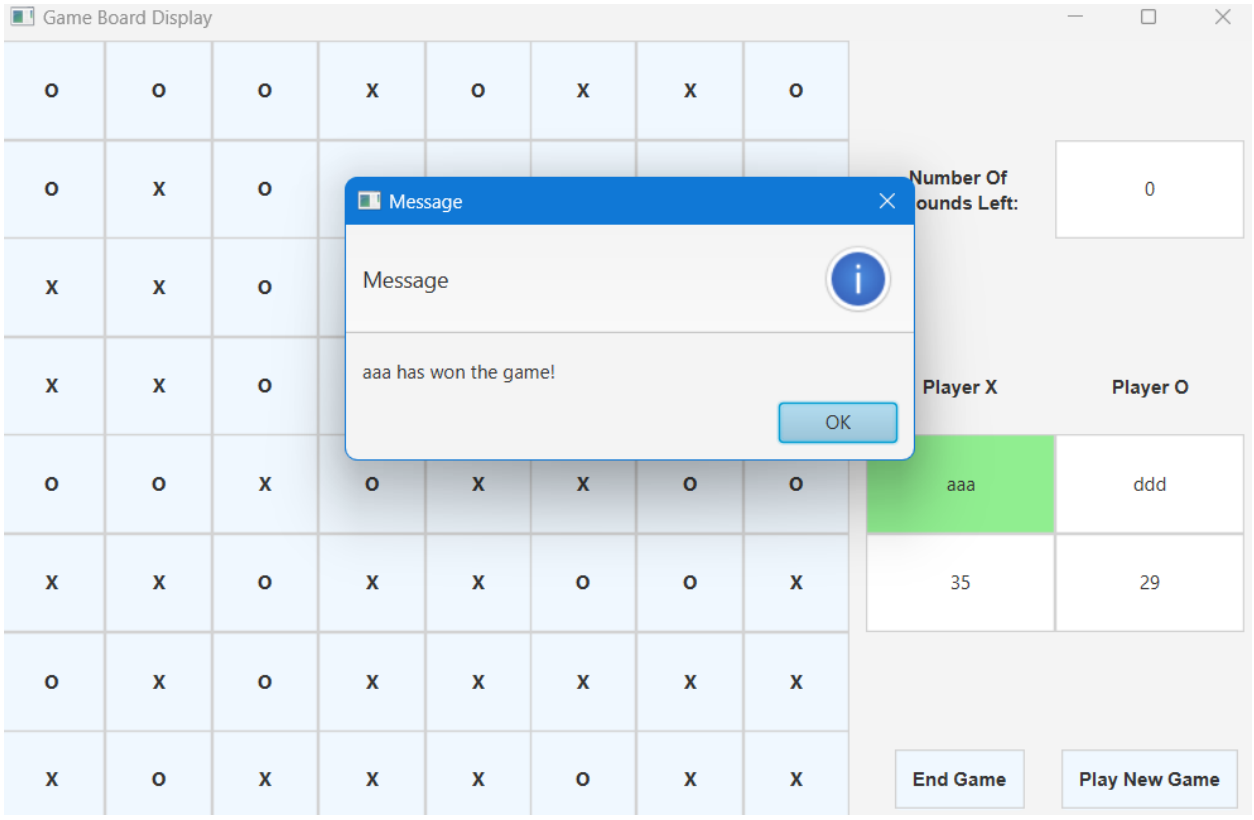
5. 8 Ronde



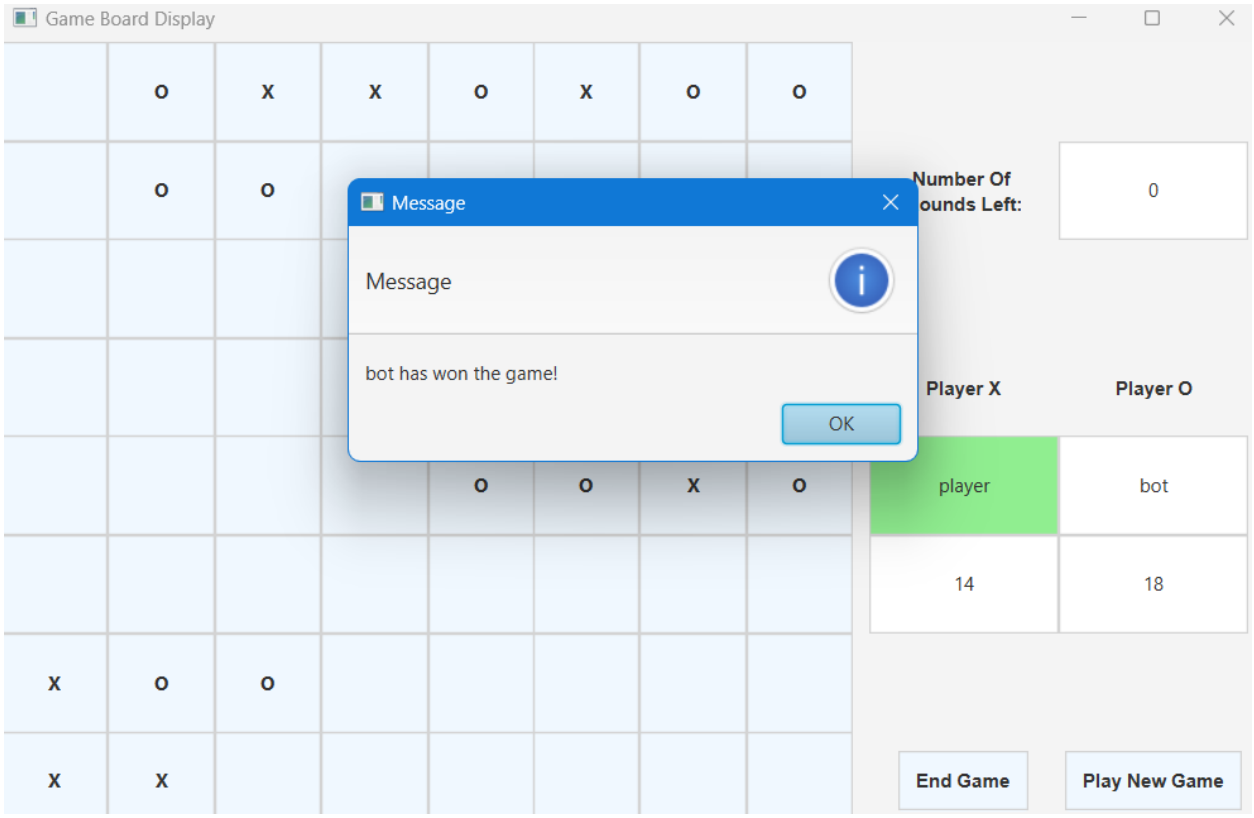
b. Bot *Local Search* vs manusia

Match Desc		Player	Score
Match 1	Player 1	Bot *Local Search*	35
	Player 2	Bot *Genetic Algorithm*	29
Match 2	Player 1	Bot *Local Search*	14
	Player 2	Bot *Genetic Algorithm*	18
Match 3	Player 1	Bot *Local Search*	12
	Player 2	Bot *Genetic Algorithm*	12
Wins (Local Search vs Genetic Algorithm) = (1 : 1)			

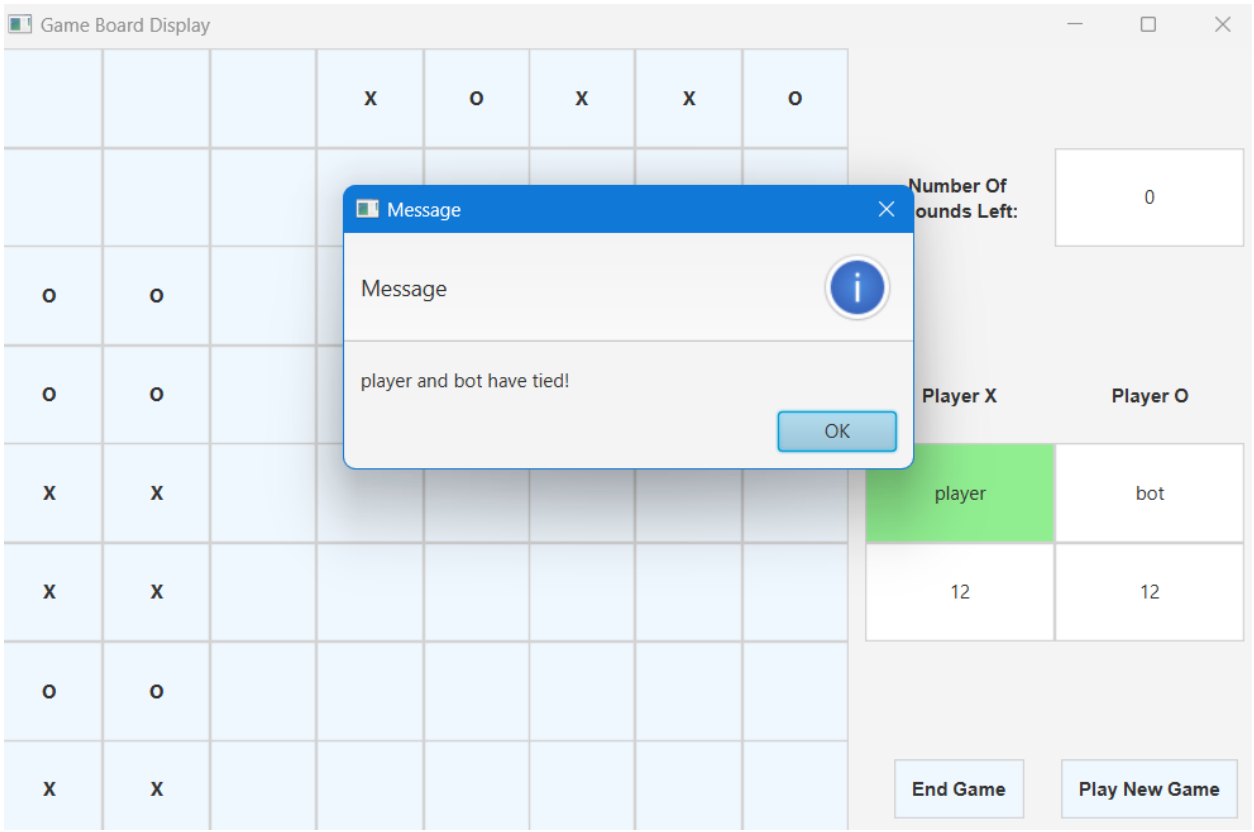
1. Maksimal Ronde



2. 12 ronde


Game Board Display
O X X O X O O
O O
O O X O
X O O
X X
Message
Message
bot has won the game!
OK
Number Of ounds Left:
0
Player X
Player O
player
bot
14
18
End Game
Play New Game

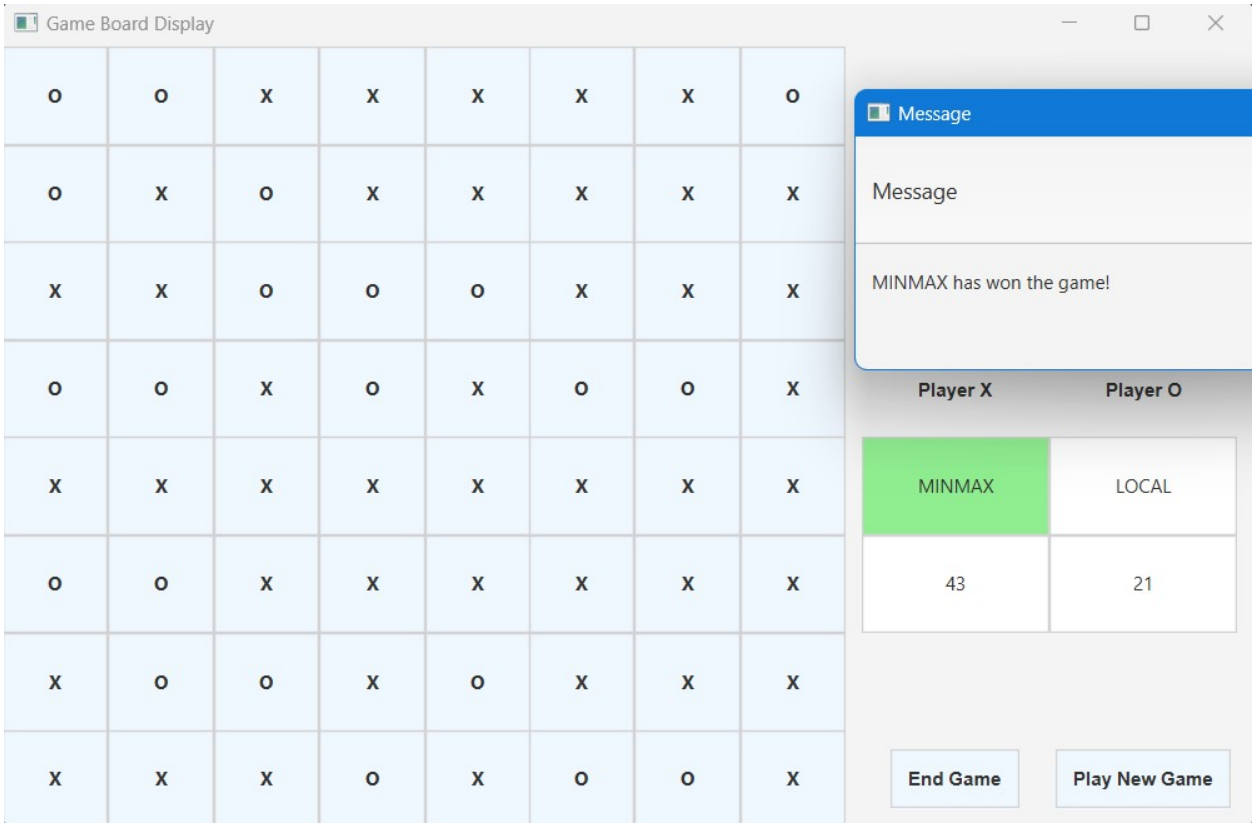
3. 8 ronde



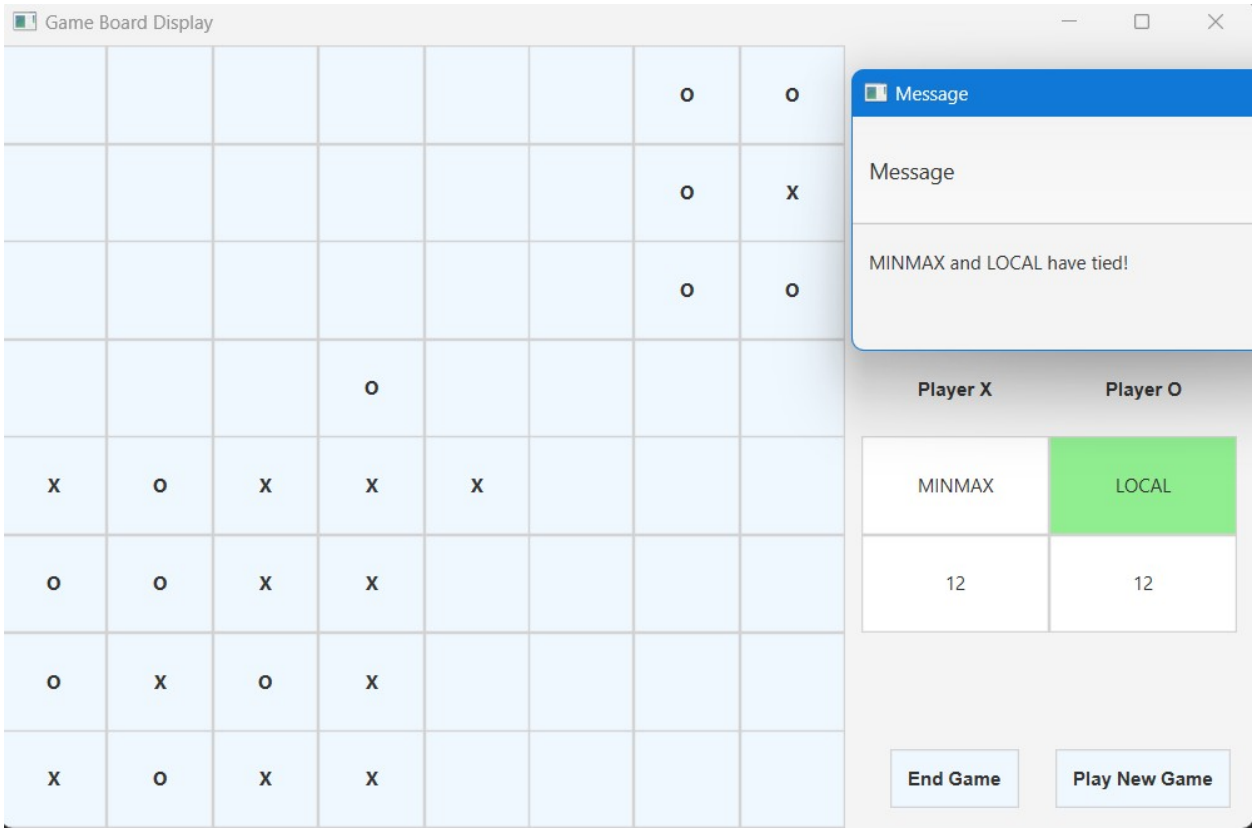
c. Bot *Minimax* vs *local search*

Match Desc		Player	Score
Match 1	Player 1	Bot *Minimax*	43
	Player 2	Bot *Local Search*	21
Match 2	Player 1	Bot *Minimax*	12
	Player 2	Bot *Local Search*	12
Match 3	Player 1	Bot *Minimax*	14
	Player 2	Bot *Local Search*	14
Wins (Minimax vs Local) = 1 : 1			

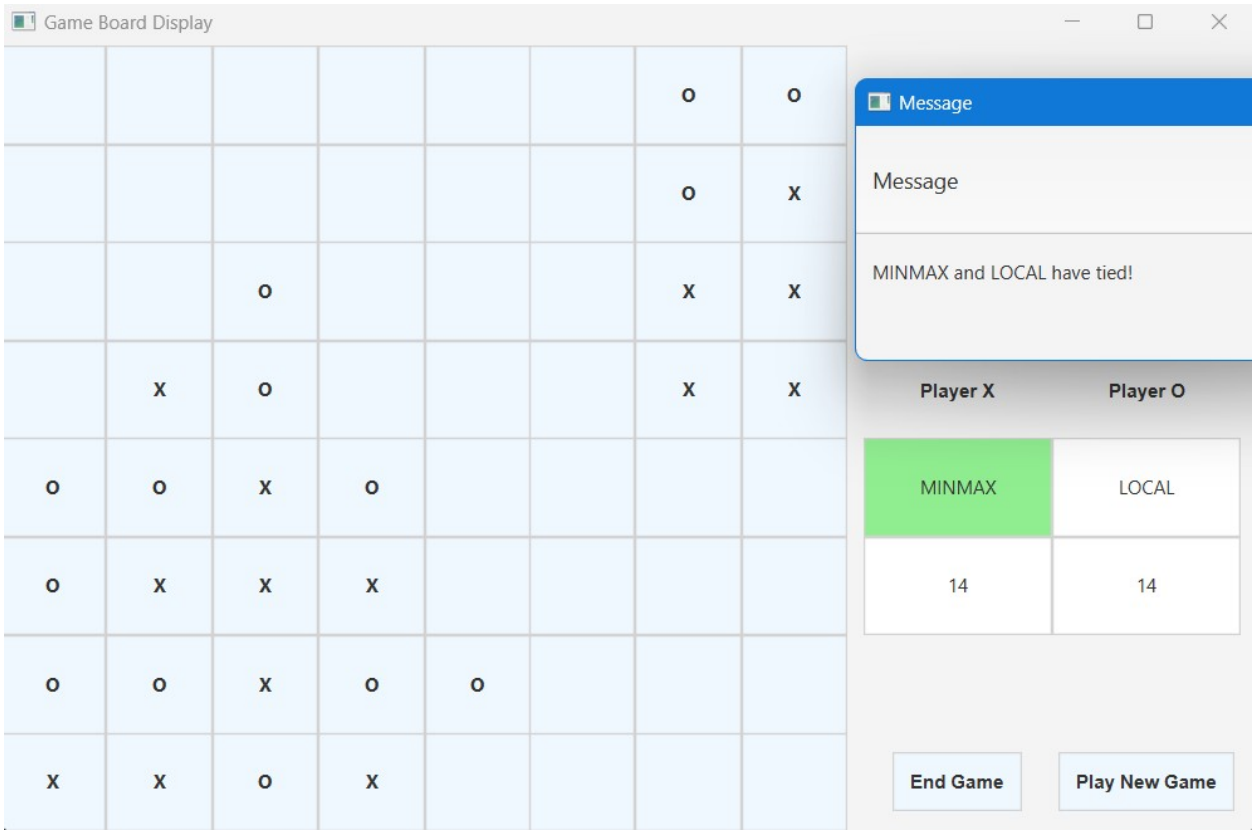
1. 28 Ronde



2. 8 Ronde


Game Board Display
Message
Message
MINMAX and LOCAL have tied!
Player X
Player O
MINMAX
LOCAL
12
12
End Game
Play New Game

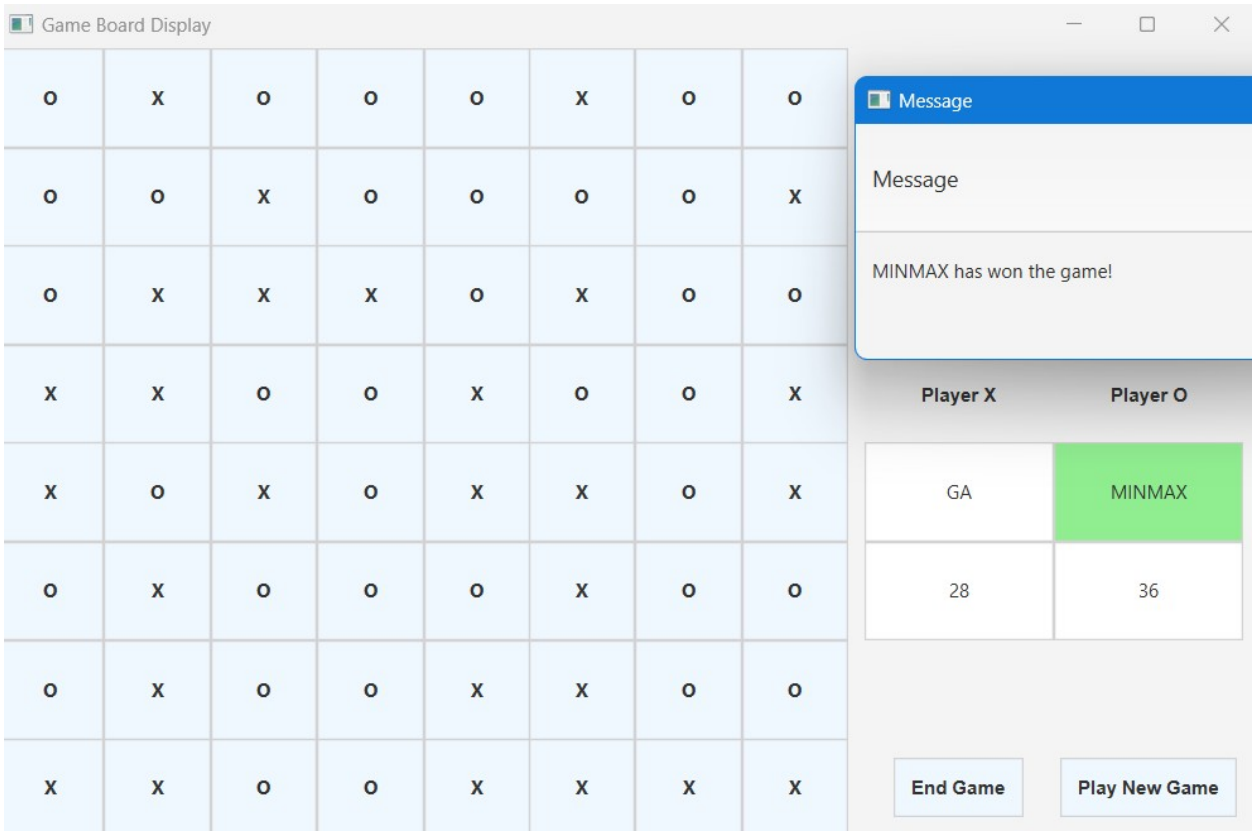
3. 10 Ronde



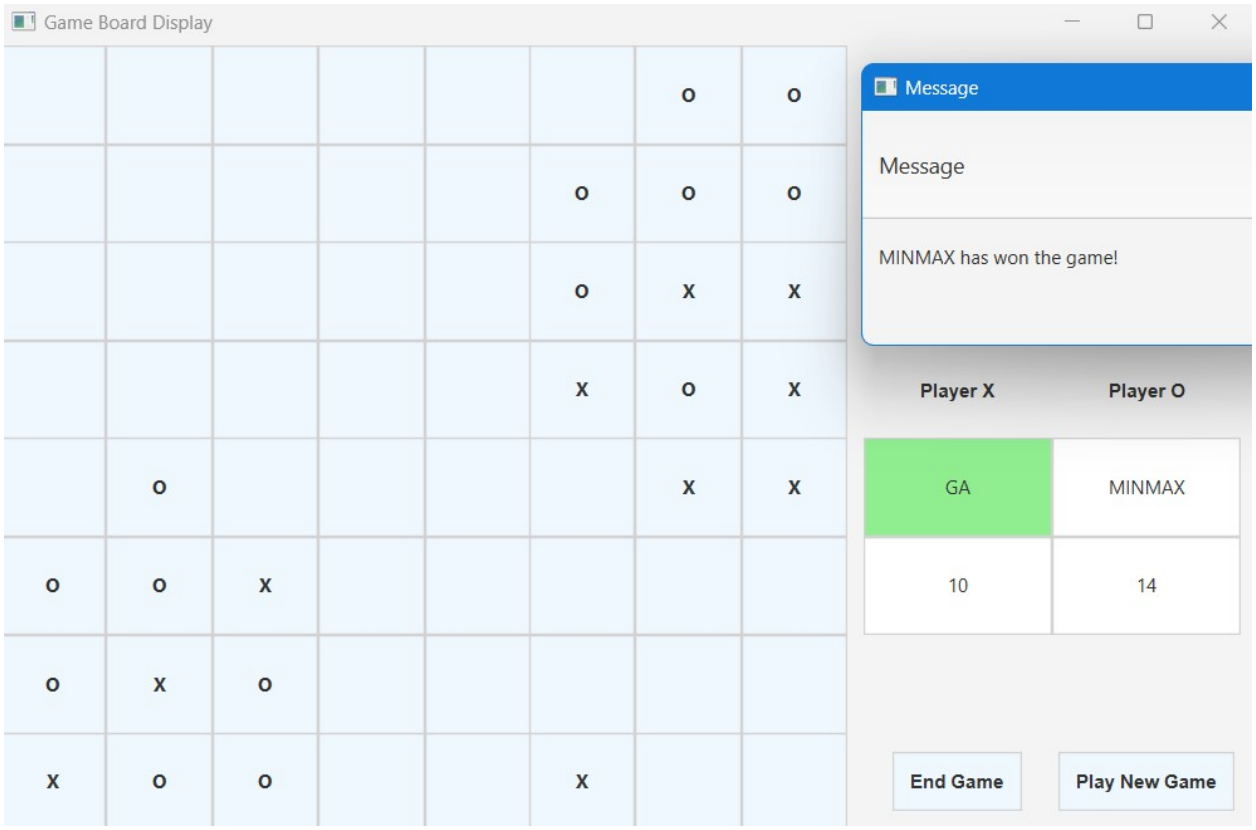
d. Bot *Minimax* vs *Genetic Algorithm*

Match Desc		Player	Score
Match 1	Player 1	Bot *Minimax*	36
	Player 2	Bot *Genetic Algorithm*	28
Match 2	Player 1	Bot *Minimax*	14
	Player 2	Bot *Genetic Algorithm*	10
Match 3	Player 1	Bot *Minimax*	19
	Player 2	Bot *Genetic Algorithm*	9
Wins (Minimax vs Genetic Algorithm) = 3 : 0			

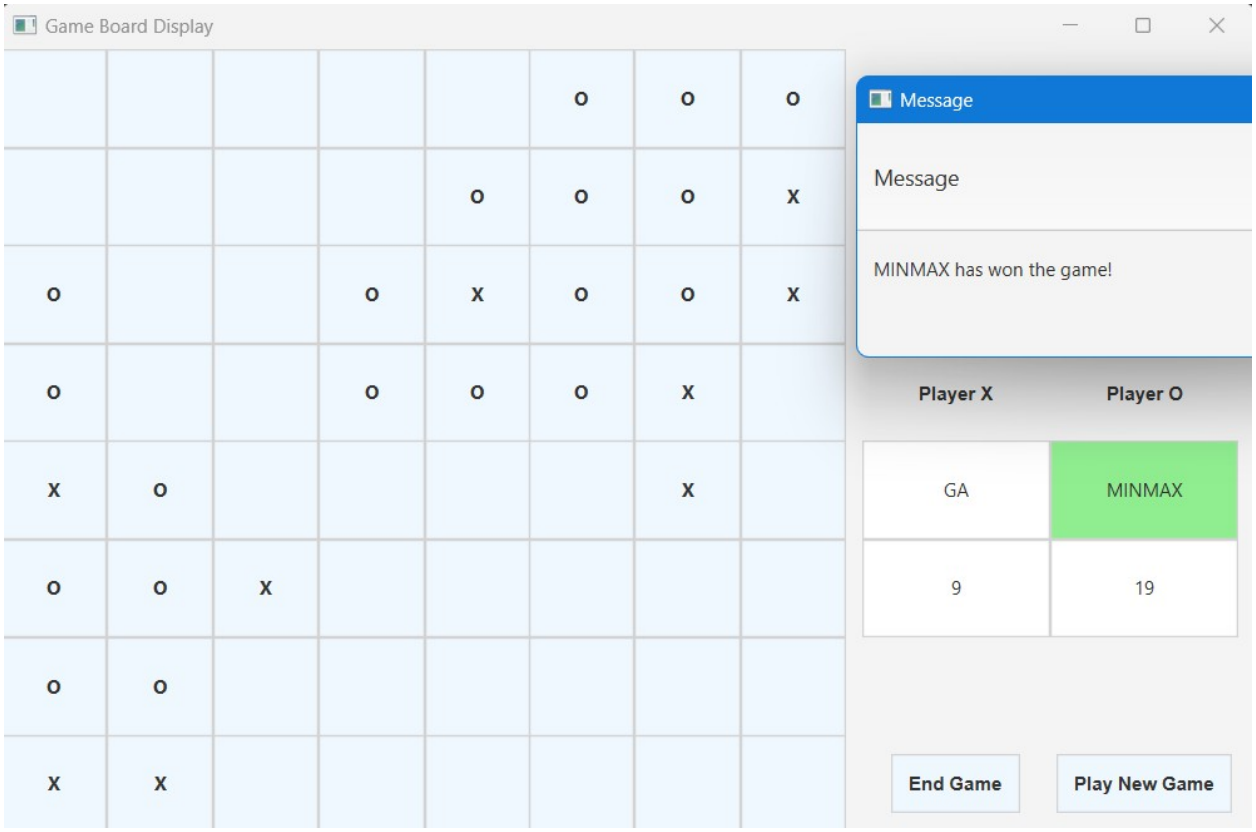
1. 28 Ronde



2. 8 Ronde


Game Board Display
Message
Message
MINMAX has won the game!
Player X
Player O
GA
MINMAX
10
14
End Game
Play New Game

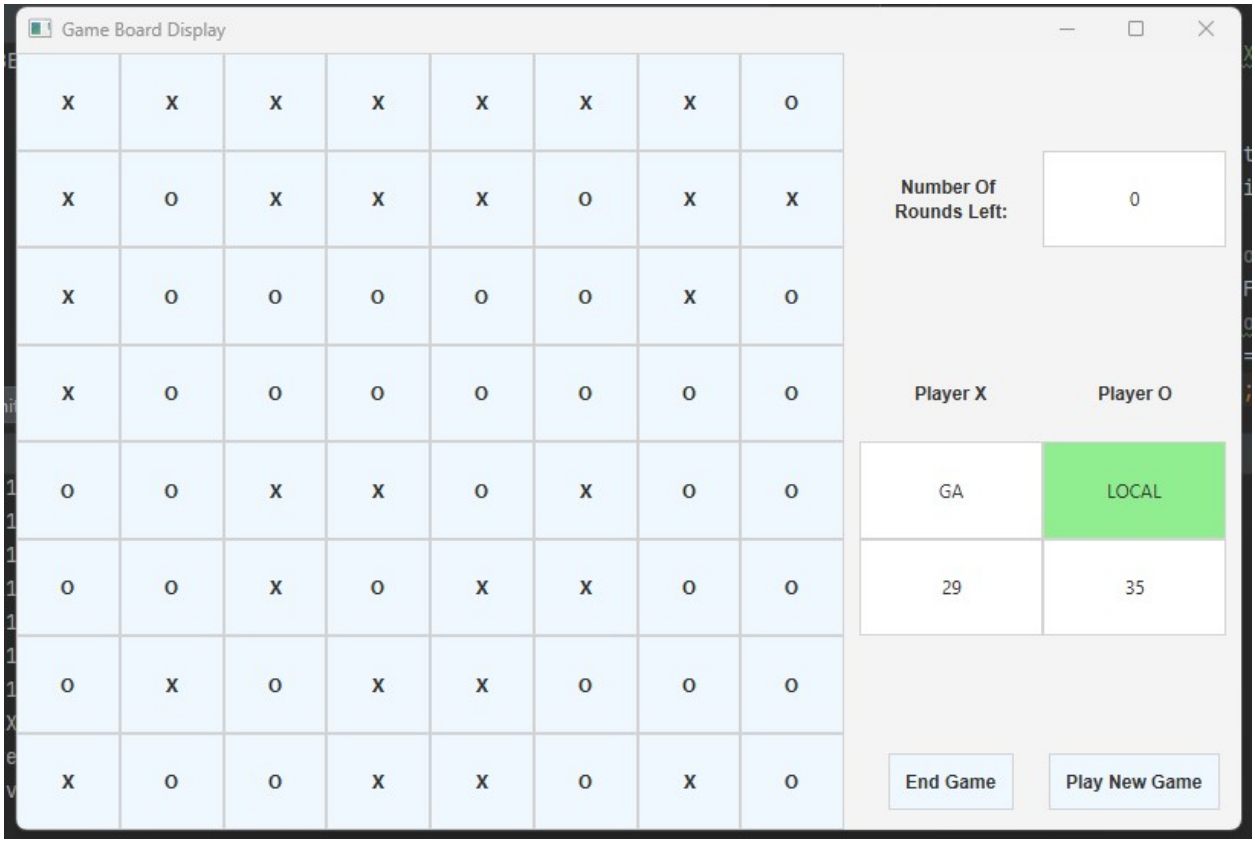
3. 10 Ronde



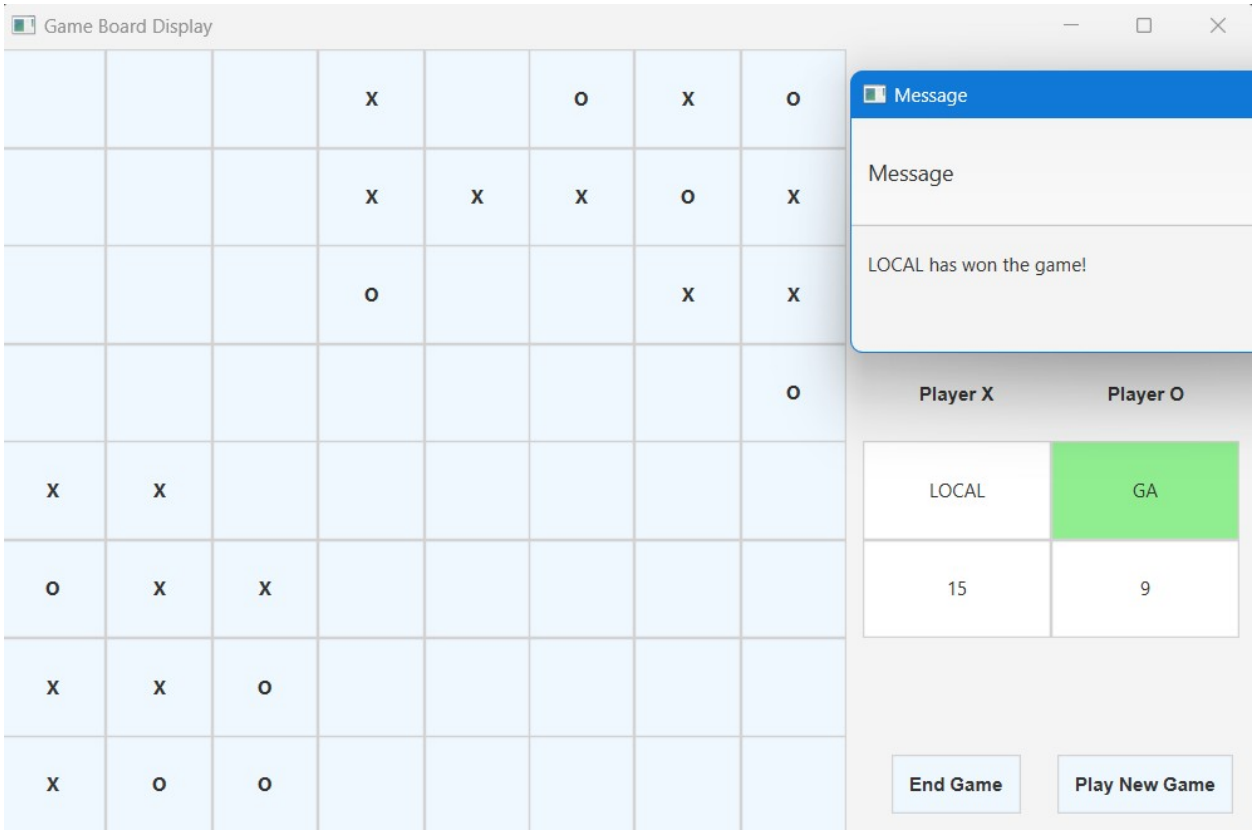
e. Bot *Local Search* vs *Genetic Algorithm*

Match Desc		Player	Score
Match 1	Player 1	Bot *Local Search*	35
	Player 2	Bot *Genetic Algorithm*	29
Match 2	Player 1	Bot *Local Search*	
	Player 2	Bot *Genetic Algorithm*	
Match 3	Player 1	Bot *Local Search*	
	Player 2	Bot *Genetic Algorithm*	
Wins (Local Search vs Genetic Algorithm) =			

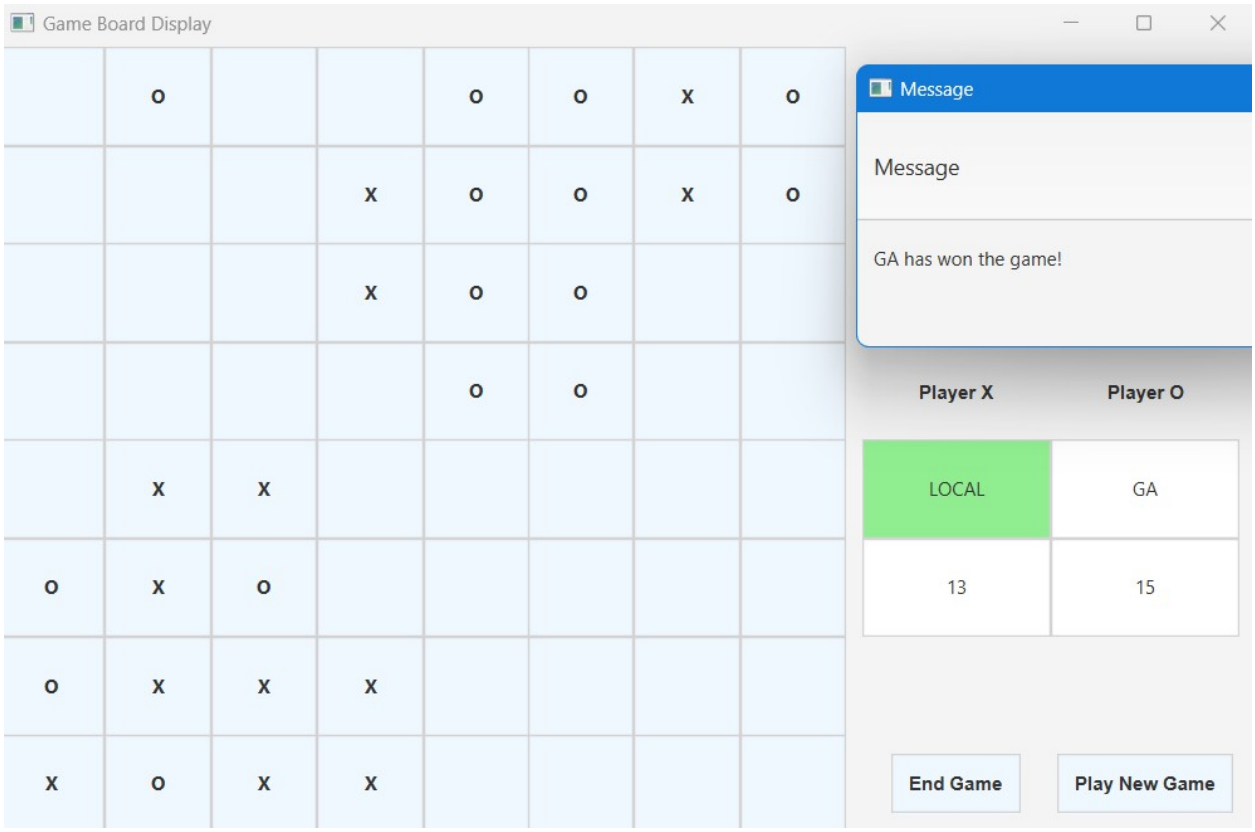
1. 28 Ronde



2. 8 Ronde


Game Board Display
Message
Message
LOCAL has won the game!
Player X
Player O
LOCAL
GA
15
9
End Game
Play New Game

3. 10 Ronde


Game Board Display
Message
Message
GA has won the game!
Player X
Player O
LOCAL
GA
13
15
End Game
Play New Game

Pembagian Tugas

Ezra M C M H / 13521073	**Laporan**
Christian Albert Hasiholan / 13521078	**Bot Local Search**
Tobias Natalio Sianipar / 13521090	**Bot MinMax**
Zidane Firzatullah / 13521163	**Bot Genetic Algorithm**

Link Github : https://github.com/tobisns/Tubes1_13521073