ANALISIS KOMPLEKSITAS 2 ALGORITMA PENYELESAIAN PERMAINAN *TIC TAC TOE* SECARA ITERATIF & REKURSIF (*RANDOM-ITERATIVE ALGORITHM* & MINIMAX ALGORITHM)

Hilmi Musyafa
hilmimusyafa@student.telkomuniversity.ac.id
Fakultas Informatika, Universitas Telkom
Bandung

Adika Ghivan Pratama
adikaghh@student.telkomuniversity.ac.id
Fakultas Informatika, Universitas Telkom
Bandung

ABSTRAK

Permainan *Tic Tac Toe* merupakan sebuah permainan klasik yang sering dimainkan oleh dua orang, permainan yang hanya menggunakan logika dan strategi sederhana yang banyak dimainkan oleh dua orang. *Tic Tac Toe* juga sering menjadi studi kasus dalam anaiisis algoritma karena ruang lingkup solusi yang cukup sedikit namun menantang dalam menyusun strategi penyelesaian. Penelitian ini bertujuan untuk menganalisis kompleksitas dua algoritma penyelesaian permainan *Tic Tac Toe*, yaitu algoritma Iteratif dan algoritma rekursif dengan pendekatan Minimax. Dengan pilihan papan antara 3 x 3, 4 x 4, atau 5 x 5, studi ini memberikan wawasan mengenai kekuatan dan kelemahan masing-masing pendekatan serta relevansinya dalam pengembangan algoritma untuk permainan strategi yang lebih kompleks.

KATA KUNCI

Algoritma, Analisis Kompleksitas Algoritma, Iteratif, Rekursif, Minimax, Kompleksitas Waktu.

1 PERKENALAN

Tic Tac Toe adalah permainan sederhana untuk dua pemain yang dimainkan di atas papan berukuran 3 x 3 atau N x N. Setiap pemain menggunakan salah satu dari dua simbol pembeda, yaitu simbol silang (X) dan simbol bulat (O). Tujuan permainan ini adalah menyusun tiga atau lebih (tergantung papan) simbol yang sama dalam satu baris secara horizontal, vertikal, atau diagonal. Permainan dimulai dengan papan kosong, dan kedua pemain bergantian untuk mengisi salah satu baki (kotak) pada papan yang masih kosong. Pemain pertama biasanya menggunakan simbol X, diikuti pemain kedua dengan simbol O. Pemain hanya boleh mengisi simbolnya di baki yang belum ditempati.

Strategi bermain menjadi kunci kemenangan dalam *Tic Tac Toe*. Pemain harus merencanakan dengan cermat peletakan simbol untuk membentuk barisan yang diinginkan sekaligus mencegah lawan menyelesaikan barisan tiga atau lebih simbol terlebih dahulu. Giliran bermain berlangsung secara bergantian hingga salah satu pemain berhasil membuat tiga atau lebih simbol sejajar dalam satu baris atau hingga semua baki pada papan terisi tanpa ada pemenang, yang menghasilkan situasi seri. Situasi ini menuntut pemain untuk terus berpikir cepat dan strategis, memanfaatkan peluang sambil memblokir lawan.

Karena adanya permaian *Tic Tac Toe* yang menggunakan cara teknis dan cara main yang terlihat menarik dan penuh tantangan, *Tic Tac Toe* merupakan salah satu permainan yang menjadi topik untuk di teliti di ranah Ilmu Komputer. *Tic Tac Toe* sering digunakan sebagai studi kasus dalam ilmu komputer karena ruang solusinya terbatas, memungkinkan analisis mendalam terhadap berbagai strategi penyelesaian tanpa memerlukan sumber daya komputasi yang besar.

Penelitian terhadap algoritma yang mampu menyelesaikan *Tic Tac Toe* memiliki beberapa alasan utama. Pertama, permainan ini menyediakan lingkungan yang ideal untuk menguji dan membandingkan performa berbagai pendekatan algoritmik, seperti metode sekuensial dan rekursif. Kedua, algoritma yang dikembangkan untuk *Tic Tac Toe* dapat menjadi dasar dalam memahami algoritma untuk permainan strategi lain yang lebih kompleks, seperti catur atau Go. Ketiga, melalui analisis algoritma, kita dapat mengevaluasi efisiensi dan efektivitas berbagai metode penyelesaian dalam menghadapi berbagai situasi permainan, baik dari segi waktu eksekusi maupun penggunaan memori, hal tersebut juga

Dalam penelitian ini, fokus diberikan pada dua pendekatan penyelesaian *Tic Tac Toe* : algoritma iteratif dan algoritma Minimax yang bersifat rekursif. Penelitian ini tidak hanya bertujuan untuk memahami kelebihan dan kekurangan masing-masing pendekatan, tetapi juga untuk memberikan wawasan dalam pengembangan algoritma permainan yang lebih efisien dan optimal. Penelitian ini juga melihat banyak aspek, salah satunya ukuran input.

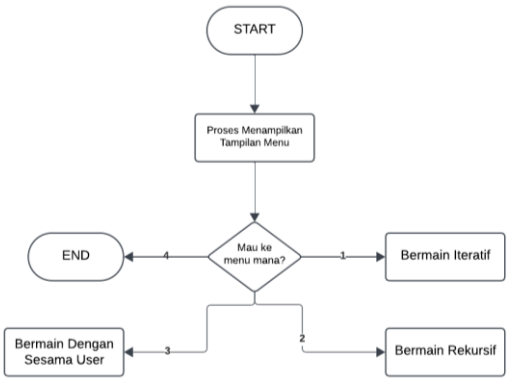
2 BATASAN MASALAH

Batasan Masalah untuk Kompleksitas Algoritma pada penelitian ini adalah membahas bagaiamana cara algoritma mengisi baki baki pada permainan *Tic Tac Toe*, bukan kompleksitas secara keseluruhan permainan hingga selesai. Sehingga dengan pembatasan materi ini, menjadi pelurusan topik dalam penelitian *Tic Tac Toe ini.*

3 STRUKTUR APLIKASI

Dalam penelitian ini, dikembangkan juga aplikasi sederhana menggunakan bahasa C++ yang sudah di *compile* dengan GCC berbentuk aplikasi konsol (*console application*) dengan interaksi menggunakan Windows Terminal tanpa tambahan GUI.

Aplikasi pada penelitian ini dibentuk untuk mampu membawa *Tic Tac Toe* pada komputer secara sederhana dengan terminal, dengan pembentuk papan "*3 x 3, 4 x 4, atau 5 x 5 tic tac toe board*" menggunakan *array* 2 dimensi yang mewakili panjang, lebar, dan lokasi elemen simbol berada tergantung dari inputan pengguna dan di tambah dengan inputan manual tanpa sentuh. Aplikasi pengembangan dapat di akses github di (https://github.com/hilmimusyafa/tictactoe) dan dapat di unduh dengan langsung melalui GitHub atau *direct download*. Sudah disiapkan kode dan aplikasi, jika ingin terdapat perubahan.



Gambar 1 : Alur sederhana aplikasi pengatur pilihan data

Pada aplikasi yang di kembangkan, sudah di beri interkasi antarmuka pengguna yang sederhana, yang mampu melayani pilihan pengguna antara bermain dengan sistem iteratif, bermain dengan sistem rekursif, atau bermain dengan sesama pengguna. Aplikasi juga di desain agar dapat mengatur siapa yang bermain, siapa yang pertama bermain, siapa yang menang, apakah papan sudah penuh atau belum, dan masih banyak penyesuaian untuk menjalankan permainan *Tic Tac Toe* secara sempurna. Sehingga aplikasi yang dikembangkan dinilai sudah siap sebagai aplikasi permainan *Tic Tac Toe* yang sempurna dan siap menjadi aplikasi percobaan dalam penelitian ini.

4 STRATEGI ALGORITMA

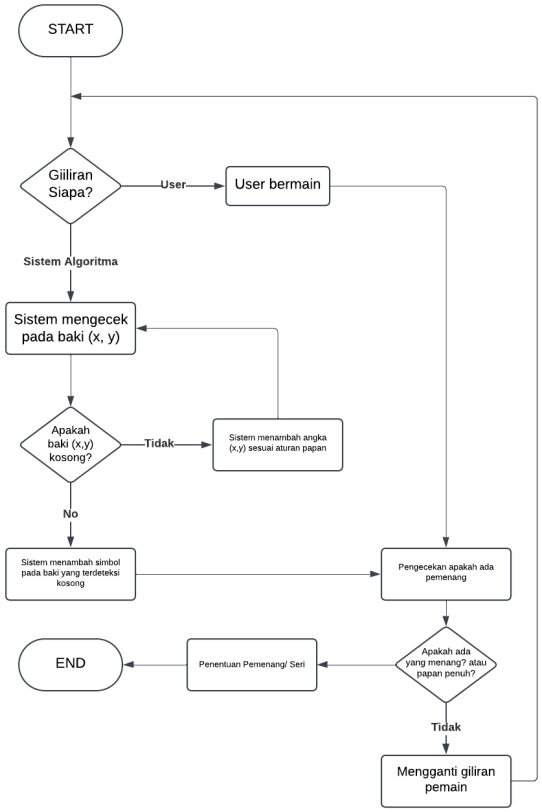
4.1 ALGORITMA ITERATIF

Berdasar pada buku "*Introduction to the Design and Analysis of Algorithm"* oleh Levitin. Algoritma Iteratif meruapakan sebuah algortima yang diimplementasikan dengan menggunakan struktur perulangan seperti *for* atau *while* untuk mengulang langkah-langkah tertentu hingga kondisi tertrntu terpenuhi. Algoritma iteratif untuk menghitung faktorial akan menggunakan perulangan untuk mengalikan bilangan dari 1 hingga *n* secara berurutan.

Dalam meneliti studi kasus yang lebih dalam terkait Algoritma Iteratif dalam *Tic Tac Toe*, telah di buatkan algoritma iteratif sederhana yang dapat meneyelesaikan permainan *Tic Tac Toe*, yang dinamakan "*Random-Iterative Tic Tac Toe Algorithm*", yang dimana menggabungkan konsep algoritma iteratif dengan dibantu fungsi acak / random. Untuk algoritma iteratif yang dibuat berpola seperti berikut :

1. Sistem akan membaca mana baki yang kosong, baki yang terdeteksi kosong tersebut akan disimpan kedalam array yang berisi posisi baki yang kosong.
2. Jika baki sudah terisi, sistem akan melanjutkan iterasi dan meneruskan pengecekan per baki papan tanpa menyimpan lokasi yang terisi tersebut.
3. Setelah baki sudah disimpan, sistem akan memilih acak dua buah angka dari angka angka array yang kosong
4. Setelah mendapat lokasi yang ingin diisi dari isi acak, sistem akan menaruh simbol di lokasi tersbut.
5. Ulangi hingga sistem menyatakan ada yang menang atau papan sudah penuh.

Algoritma ini bersifat dikarenakan dia perlu mengulang dengan perulangan/ loop (for-do) sebanyak jumlah papan. Dengan pendefinisian alur algoritma tersebut, sehingga sitem permainan iteratif dapat digambarkan dalam *flowchart* seperti pada lampiran **Gambar 2**.



Gambar 2 : *Flowchart* penjelasan jalannya permainan iteratif berlangsung.

Sajian algoritma iteratif ini akan di bentuk dalam satu prosedur pemrograman yang bermana "iterativeEngine" yang berfungsi menerapkan algoritma iteratif untuk giliran sistem dalam permainan *Tic Tac Toe*. Prosedur tersebut dapat dibentuk dalam bentuk kode *pseuducode* seperti di lampiran **Kode 1.**

```
1  procedure iterrativeEngine (in/out b : board, in : stringValue string)
2     chunkEmptyLocationArray[25][2] : integer
3     arrayCounter : integer
4
5     for (int cX = 0; cX < ukuranPapan; cX++)
6        for (int cY = 0; cY < ukuranPapan; cY)
7           if (b.board.chart[chunkLocationX][chunkLocationY] == " ") then
8              chunkEmptyLocationArray[arrayCounter][0] = chunkLocationX;
9              chunkEmptyLocationArray[arrayCounter][1] = chunkLocationY;
10             arrayCounter++;
11          endif
12       endfor
13    endfor
14
15    if (arrayCounter > 0) then
16       int randomIndex = randomFunction() % arrayCounter;
17       int x = chunkEmptyLocationArray[randomIndex][0];
18       int y = chunkEmptyLocationArray[randomIndex][1];
19
20       b.boardSize3.chart[x][y] = stringValue;
21    endif
22 endprocedure
```

Kode 1 : Kode *pesudocode* prosedur "iterativeEngine" implementasi dari algoritma iteratif

Dengan di bentuknya prosedure "iterativeEngine", sistem mampu mengisi baki baki papan *Tic Tac Toe* dengan algoritma Iteratif hingga permainan dinyatakan selesai, dan di bantu oleh

fungsi aplikasi yang mengendalikan jalannya permainan *Tic Tac Toe* secara bergantian dan tersusun.

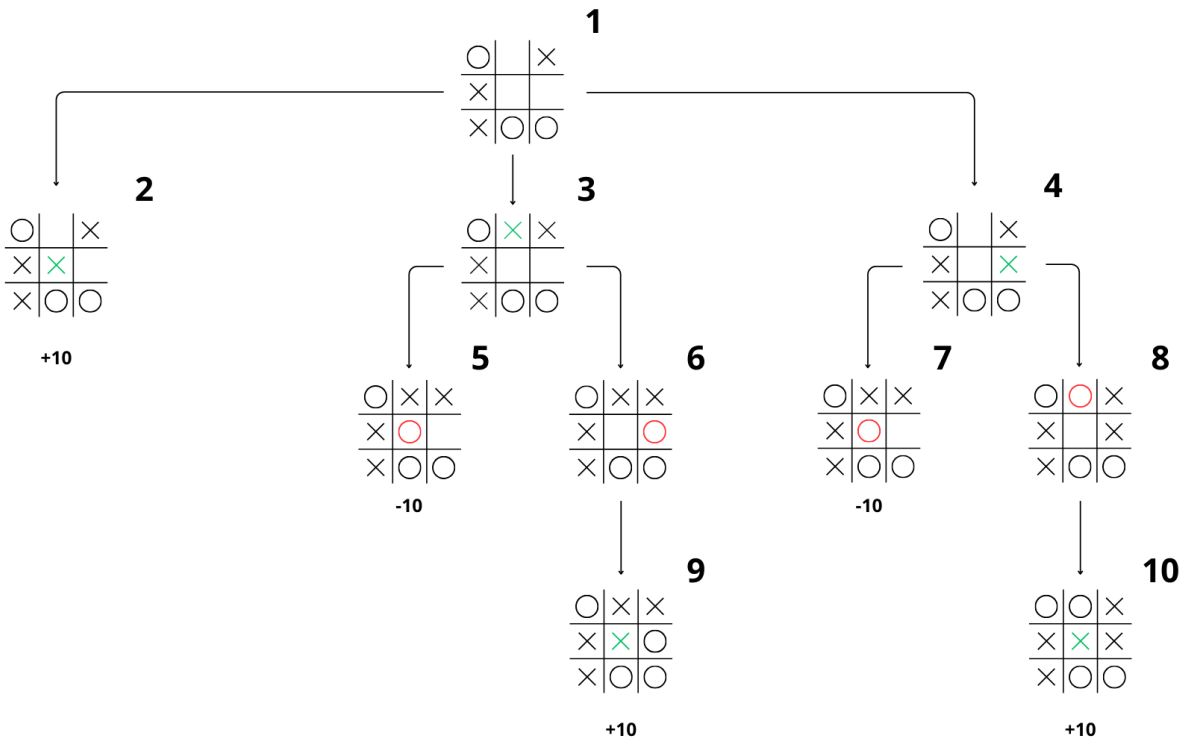
Alasan terpilihnya algoritma Iteratif pada *Tic Tac Toe* adalah penggunaan yang dinilai mudah dikarenakan prosedur algoritma sederhana dan tak memakan banyak waktu untuk komputasi. Hanya saja cara kerja pada algoritma iteratif ini dinilai tidak prediktif dikarena menggunakan eksekusi acak.

4.2 ALGORITMA REKURSIF (MINIMAX)

Algoritma rekursif merupakan algoritma yang menyelesaikan suatu masalah dengan cara memecahnya menjadi submasalah yang lebih kecil dan serupa, kemudian memanggil dirinya sendiri secara berulang hingga mencapai kondisi dasar (base case).

Dalam penelusuran dan penelitian algoritma rekursif, terdapat salah satu pilihan algoritma yang dapat di aplikasikan dalam permainan *Tic Tac Toe*, yaitu algoritma Minimax. Algoritma Minimax sendiri menggunakan model algoritma bersifat rekursif dengan cara di setiap giliran sistem bermain, algoritma akan membaca kemungkinan yang ada berdasarkan dari pola papan permainan, dalam penelitian ini, pengaturan langkah algoritma di atur secara lengkap seperti ini :

1. Pada tahap awal, sistem akan membaca dan mengidentifikasi pola papan permainan yang sedang berlangsung. Sistem memetakan setiap baki atau kotak pada papan permainan untuk menentukan mana yang sudah diisi, baik oleh sistem sendiri maupun oleh lawan, serta baki yang masih kosong. Informasi ini akan digunakan sebagai dasar untuk melakukan langkah-langkah selanjutnya.
2. Berdasarkan pola papan saat ini, sistem akan melakukan konstruksi terhadap semua kemungkinan konfigurasi papan yang dapat terjadi setelah sistem atau lawan mengambil langkah. Untuk setiap kemungkinan langkah, sistem menyimulasikan pengisian simbol pada baki yang masih kosong. Proses ini menghasilkan berbagai skenario yang mencerminkan semua langkah potensial dalam permainan. Seperti pada penggambaran pada **Gambar 3**.
3. Setelah semua kemungkinan pola papan terbentuk, sistem akan mengevaluasi setiap konfigurasi menggunakan fungsi evaluasi yang telah dirancang sebelumnya. Fungsi ini bertugas memberikan nilai terhadap suatu pola berdasarkan kemungkinan kemenangan atau keuntungan bagi sistem. Pola yang membawa sistem lebih dekat ke kemenangan diberi nilai tinggi, sementara pola yang mendekatkan lawan ke kemenangan diberi nilai rendah.
4. Dari hasil evaluasi pola, sistem menggunakan pendekatan algoritma Minimax untuk memilih langkah terbaik. Minimax bekerja dengan asumsi bahwa kedua pemain akan bermain secara optimal: sistem mencoba memaksimalkan keuntungannya (nilai tinggi), sementara lawan berusaha meminimalkan keuntungan sistem. Oleh karena itu, sistem akan mencari langkah dengan nilai evaluasi tertinggi, sambil mempertimbangkan potensi langkah balasan dari lawan di masa mendatang.
5. Setelah langkah optimal dipilih, sistem akan mengisi simbolnya pada baki yang sesuai berdasarkan langkah tersebut. Proses ini mengubah keadaan papan permainan ke konfigurasi baru, yang kemudian digunakan untuk langkah berikutnya.
6. Setelah sistem mengambil langkah, giliran berpindah ke lawan. Proses ini diulang secara bergantian hingga salah satu pemain mencapai kondisi akhir, seperti kemenangan, kekalahan, atau permainan berakhir seri. Pada setiap giliran, algoritma Minimax diaktifkan untuk memastikan bahwa sistem selalu memilih langkah yang paling optimal



Gambar 3 : Gambaran pola algoritma Minimax yang diimplementasikan pada permainan *Tic Tac Toe* dengan penggambaran Tree

Algoritma Minimax menggunakan 3 skenario pengolahan dan pengaliran data yang di masukkan dalam 3 fungsi/ prosedur dengan keterangan seperti berikut :

a. Fase/ Fungsi Evaluasi Board
   Fungsi evaluateBoardScoring digunakan untuk mengevaluasi keadaan papan permainan dan memberikan skor berdasarkan siapa yang menang. Fungsi ini menerima tiga parameter: b yang merupakan papan permainan, sistemValue yang merupakan nilai yang digunakan oleh sistem (misalnya "X" atau "O"), dan humanValue yang merupakan nilai yang digunakan oleh pemain manusia.
   - Jika sistem adalah pemenang, fungsi ini mengembalikan skor 10.
   - Jika pemain manusia adalah pemenang, fungsi ini mengembalikan skor -10.
   - Jika tidak ada pemenang dan papan penuh, fungsi ini mengembalikan skor 0.

```
function evaluateBoardScoring (in/out b : selectedBoard, in  botValue : string,
humanValue string)
    if  (isWinner(b, botValue)) then
        return 10
    else if (isWinner(b, humanValue)) then
        return -10
    else
        return 0
    endif
endfunction
```

Kode 2 : Pseuducode fungsi evaluateBoardScoring pada Recursive

b. Fase/ Fungsi Minimax
   Fungsi minimax adalah implementasi dari algoritma Minimax yang digunakan dalam permainan dua pemain seperti *Tic Tac Toe* untuk menentukan langkah terbaik. Fungsi ini menerima lima parameter: papan permainan, kedalaman rekursi, boolean yang menunjukkan apakah giliran sistem atau pemain manusia, nilai yang digunakan oleh sistem, dan nilai yang digunakan oleh pemain manusia.
   - Fungsi ini pertama-tama mengevaluasi skor papan permainan menggunakan evaluateBoardScoring.
   - Jika ada pemenang atau papan penuh, fungsi ini mengembalikan skor yang sesuai.
   - Jika giliran sistem (isMaximizing adalah true), fungsi ini mencoba memaksimalkan skor dengan mengevaluasi semua langkah yang mungkin.
   - Jika giliran pemain manusia (isMaximizing adalah false), fungsi ini mencoba meminimalkan skor dengan mengevaluasi semua langkah yang mungkin.

```
function minimax (in/out b : selectedBoard, in depth : integer, bool : isMaximizing,
botValue : string, humanValue string)
    currentScore : integer
    currentScore = evaluateBoardScoring(b, botValue, humanValue)

    // Base Case
    if  (isWinner(b, botValue)) then
        return 10 - depth
    endif
    if (isWinner(b, humanValue)) then
        return depth - 10
    endif
    if (isFull((b)) then
        return 0
    endif
    // Recuresive Case
    if (isMaximizing) then
        bestScore : integer
        bestScore = -1000

        for (integer i = 0 ; i < cX; i++) do
            for (integer = 0 ; j < cY; j++) do
                if (b.chart[i][j] == " ") then
                    b.chart[i][j] = botValue
                    int score = minimax(b, depth + 1, false, botValue, humanValue)
                    b.chart[i][j] = " "
                    bestScore = max(bestScore, score)
            endfor
        endfor
        return bestScore
    else
        bestScore : integer
        bestScore = 1000

        for (integer i = 0 ; i < cX; i++) do
            for (integer = 0 ; j < cY; j++) do
                if (b.chart[i][j] == " ") then
                    b.chart[i][j] = humanValue
                    int score = minimax(b, depth + 1, true, botValue, humanValue)
                    b.chart[i][j] = " "
                    bestScore = min(bestScore, score)
            endfor
        endfor
        return bestScore
endprocedure
```

Kode 3 : Pseuducode fungsi minima yang menjalankan algoritma minimax *Tic Tac Toe*

c. Fase/ Fungsi recursiveEngine
   Fungsi recursiveEngine adalah fungsi utama yang menggerakkan algoritma Minimax. Fungsi ini memutuskan langkah terbaik untuk bot dengan memanggil fungsi minimax dan mengupdate papan permainan berdasarkan hasil dari minim.
   - Fungsi ini memeriksa semua langkah yang mungkin untuk bot.
   - Untuk setiap langkah, fungsi ini memanggil minimax untuk mengevaluasi skor langkah tersebut.
   - Fungsi ini memilih langkah dengan skor terbaik dan mengupdate papan permainan.

5 ANALISIS HASIL

Dengan melakukan penelitian ini, kami mendapat hasil penelitian yaitu :

5.1 ANALISIS KOMPLEKSITAS ALGORITMA

a. Algoritma Iteratif (*Random-Iterative Algorithm)*
   Untuk melakukan analisa kompleksitas algoritma pada *Tic Tac Toe* dengan Iteratif. Pertama, hitung berapa kali sistem melakukan loop untuk mencari data yang kosong, bisa dilihat pada fungsi algoritma ini :

```
procedure iterrativeEngine (in/out b : board, in : stringValue string)
    chunkEmptyLocationArray[25][2] : integer
    arrayCounter : integer

    for (int cX = 0; cX < ukuranPapan; cX++)
        for (int cY = 0; cY < ukuranPapan; cY)
            if (b.board.chart[chunkLocationX][chunkLocationY] == " ") then
                chunkEmptyLocationArray[arrayCounter][0] = chunkLocationX;
                chunkEmptyLocationArray[arrayCounter][1] = chunkLocationY;
                arrayCounter++;
            endif
        endfor
    endfor

    if (arrayCounter > 0) then
      int randomIndex = randomFunction() % arrayCounter;
      int x = chunkEmptyLocationArray[randomIndex][0];
      int y = chunkEmptyLocationArray[randomIndex][1];

```

```
20          b.boardSize3.chart[x][y] = stringValue;
21        endif
22    endprocedure
```

Pada algoritma fungsi di atas, dapat di simpulkan memiliki beberapa operasi, yaitu :

Operasi	Tipe	Kesesuaian Eksekusi
If (b.board.chart[chunkLocationX][chunkLocationY] == " ")	Perbandingan	cX * cY kali

Dengan data yang ada di atas, eksekusi berdasarkan dari panjang/ lebar papan (cX, cY), dan dapat disimpulkan kompleksitas waktu dari algoritma diatas adalah :

$$T(n) = cX * cY$$
$$T(n) = n * n$$
$$T(n) = n^2$$
$$n = Panjang\ atau\ Lebar\ papan$$

Dengan begitu kita mendapatkan untuk analisis Kompleksitas sistem mencari yang kosong adalah $O \in (n^2)$. Setiap iterasi pada algoritma iteratif dilakukan untuk mencari posisi kolom yang kosong, sebagai berikut:

1. Pada langkah pertama, seluruh papan dengan ukuran $n\ x\ n$ diperiksa, menghasilkan n^2 operasi.
2. Setelah setiap langkah pengisian, jumlah kolom kosong berkurang, tetapi algoritma tetap memeriksa semua elemen papan.

Sebagai contoh :
- Langkah pertama: n^2 operasi.
- Langkah kedua: n^2 operasi.
- Langkah ke-k: n^2, dan seterusnya

Jadi operasi pada langkah berapa pun akan tetap sama, Ini disebabkan oleh Algoritma Iteratif yang tidak akan mengurangi jumlah elemen yang diperiksa pada setiap langkah, jumlah operasi untuk k langkah pengisian baki dapat dihitung sebagai

$$k * n^2$$
$$atau$$
$$T(k,n) = n^2 + n^2 + n^2 + \cdots + n^2$$

Dan diperoleh bentuk rumus geometri nya :

$$T(k,n) = \sum_{i=1}^{k} n^2$$

Dari hasil Analisis diatas, kita dapat meninjau Best Case (Kasus Terbaik), Average Case (Kasus Rata-Rata), dan Worst Case (Kasus Terburuk) dari Algoritma ini.

- Best Case :
  Sistem hanya perlu melakukan iterasi minimal untuk mendaapatkan baki kosong pertama. Kompleksitas waktu untuk Case ini berada dalam kategori $O(n^2)$, karena iterasi bergantung pada ukuran papan,
- Average Case :
  Sistem akan melakukan iterasi ke Sebagian besar baki pada papan sebelum menemukan baki kosong yang dipilih secara acak. Kompleksitas waktu untuk Case ini masih berada dalam kategori $O(n^2)$, karena iterasi masih mencakup Sebagian besar bagian dari papan.
- Worst Case :
  Sistem harus memeriksa seluruh kolom pada papan untuk mencari kolom terakhir yang masih kosong, hal ini akan terjadi apabila semua kolom hampir terisi seluruhnya. Kompleksitas waktu untuk Case ini masih berada dalam kategori $O(n^2)$.

Kompleksitas waktu untuk Algoritma Iteratif ini konsisten berada dalam kategori $O(n^2)$, hal ini disebabkan oleh penggunaan dua Nested Loop yang bergantung pada ukuran papan.` Kelebihan dari Algoritma Iteratif ini adalah Implementasi nya yang sederhana, namun Algoritma ini juga memiliki kekurangan. Apabila ukuran papan mengalami peningkatan, maka efisiensi nya menurun secara signifikan, ini mengakibatkan Algoritma ini kurang optimal untuk permainan *Tic Tac Toe* yang lebih besar.

b. Algoritma Rekursif (Minimax *Algorithm*)
Untuk Melakukan analisa kompleksitas algoritma pada skenario *Tic Tac Toe* dengan algoirtma rekursif minimax, akan di angkat algoritma minimax dengan *pseudocode* seperti ini :

```
1   function minimax (in/out b : selectedBoard, in depth : integer, bool :
2   isMaximizing,  botValue : string, humanValue string)
3       currentScore : integer
4       currentScore = evaluateBoardScoring(b, botValue, humanValue)
5
6       // Base Case
7       if  (isWinner(b, botValue)) then
8           return 10 - depth
9       endif
10      if (isWinner(b, humanValue)) then
11          return depth - 10
12      endif
13      if (isFull((b)) then
14          return 0
15      endif
16      // Recuresive Case
17      if (isMaximizing) then
18          bestScore : integer
19          bestScore = -1000
20
21          for (integer i = 0 ; i < cX; i++) do
22              for (integer = 0 ; j < cY; j++) do
23                  if (b.chart[i][j] == " ") then
24                      b.chart[i][j] = botValue
25                      int score = minimax(b, depth + 1, false, botValue, humanValue)
26                      b.chart[i][j] = " "
27                      bestScore = max(bestScore, score)
28              endfor
29          endfor
30          return bestScore
31      else
32          bestScore : integer
33          bestScore = 1000
34
35          for (integer i = 0 ; i < cX; i++) do
36              for (integer = 0 ; j < cY; j++) do
37                  if (b.chart[i][j] == " ") then
38                      b.chart[i][j] = humanValue
39                      int score = minimax(b, depth + 1, true, botValue, humanValue)
40                      b.chart[i][j] = " "
41                      bestScore = min(bestScore, score)
42              endfor
43          endfor
44          return bestScore
45  endprocedure
```

Dengan kode *pseuducode* di atas, akan di coba untuk mendapatkan analisis kompleksitas dengan cara mengecek mana base case dan mana recursive case :

Operasi	Algoritma

Base Case	if (isWinner(b, botValue)) then
	return 10 - depth
	endif
	if (isWinner(b, humanValue)) then
	return depth - 10
	endif
	if (isFull((b)) then
	return 0
	endif

Recursive Case	if (isMaximizing) then
	bestScore : integer
	bestScore = -1000
	
	for (integer i = 0 ; i < cX; i++) do
	for (integer = 0 ; j < cY; j++) do
	if (b.chart[i][j] == " ") then
	b.chart[i][j] = botValue
	int score = minimax(b, depth + 1, false, botValue, humanValue)
	b.chart[i][j] = " "
	bestScore = max(bestScore, score)
	endfor
	endfor
	return bestScore
	else
	bestScore : integer
	bestScore = 1000
	
	for (integer i = 0 ; i < cX; i++) do
	for (integer = 0 ; j < cY; j++) do
	if (b.chart[i][j] == " ") then
	b.chart[i][j] = humanValue
	int score = minimax(b, depth + 1, true, botValue, humanValue)
	b.chart[i][j] = " "
	bestScore = min(bestScore, score)
	endfor
	endfor
	return bestScore

Dengan adanya tabel di atas dapat di simpulkan dengan cara pendekatan algoritma rekursif yang dimana mendapatkan hasil ukuran input :

$$ukuran\ input = cX * cY$$
$$ukuran\ input = b$$

Dan mendapatkan operasi dasar berupa :
- Mendapatkan evaluateScore
- Pengecekan kondisi kemenangan
- Iterasi melalui semua kemungkinan langkah kosong pada papan.

Sehingga bisa di simpulkan untuk mendapat rumus rekursifnya adalah :

$$T(m,b) = \begin{cases} O(1), & jika\ m = 0 \\ \mathrm{b} * \mathrm{T(m-1)} + O(1), & jika\ m > 0 \end{cases}$$

Bisa di rubah menjadi :

$$T(m,b) = \begin{cases} 1, & jika\ m = 0 \\ \mathrm{b} * \mathrm{T(m-1)} + 1, & jika\ m > 0 \end{cases}$$

Dengan m adalah *depth* atau kedalaman tree dari banyaknya pembuatan prediksi *tic tac toe*. Dan bisa didapatkan dari $T(m)$, yaitu :

$$T(m,b) = b * T(m-1) + 1$$

Dengan 1 adalah operasi dasar seperti evaluasi skor atau pengecekan kondisi menang. Setelah itu akan di coba di hitung :

$$T(m) = b * T(m-1) + O(1)$$
$$T(m) = b * T(m-1) + 1$$

Misal m = 0

$$T(0,b) = 1$$

Subtitusikan untuk m > 0, untuk m = 1

$$T(1,b) = b * T(0,b) + 1$$
$$b * 1 + 1$$
$$b + 1$$

Untuk m = 2

$$T(2,b) = b * T(1,b) + 1$$
$$b * (b+1) + 1$$
$$b^2 + b + 1$$

Untuk m = 3

$$T(3,b) = b * T(2,b) + 1$$
$$b * (b^2 + b + 1) + 1$$
$$b^3 + b^2 + b + 1$$

Dan pola yang terbentuk

$$T(m,b) = b^m + b^{m-1} + b^{m-2} + \cdots + b + 1$$

Sehingga bisa dibentuk dalam bentuk geometri

$$T(m,b) = \sum_{i=0}^{m} b^i$$

Dengan rumus deret geometri, dapat disederhanakan menjadi :

$$\sum_{i=0}^{m} b^i = \frac{b^{m+1} - 1}{b - 1}, untuk\ b \neq 1$$

Dan mendapat hasil akhir

$$T(m) = b^m + m$$
$$\mathrm{T(m)} = b^m$$

Dengan melakukan perhitungan dari rumus rekursif tersebut dengan pola, didapatkan kompleksitas waktu pada algoritma rekursif adalah b^m. Untuk analisis ini juga dapat disimpulkan dan di kembangkan dalam skenario yang berbeda :

- Best Case :
  Dalam skenario best case, algoritma dapat berhenti lebih awal jika salah satu pemain menang di langkah-langkah awal. Misalnya Salah satu pemain menang di langkah ke-5 atau ke-6 (kedalaman lebih dangkal dari $m = 9$). Kompleksitasnya bergantung pada kedalaman sebenarnya dari pohon yang perlu dieksplorasi sebelum pemenang ditemukan. Jika permainan selesai di langkah ke-5, maka

kedalaman yang dicapai adalah $m = 5$ sehingga kompleksitas best case nya adalah :

$$b^5$$

- Worst Case
  Dalam skenario worst case, algoritma akan mengeksplorasi seluruh pohon permainan hingga kedalaman maksimum m. Ini terjadi ketika Permainan berlangsung hingga langkah terakhir tanpa pemenang sampai langkah ke-9 (misalnya, semua kotak terisi), sehingga worse case nya adalah :

$$9^9 = 387.420.489\ skenario$$

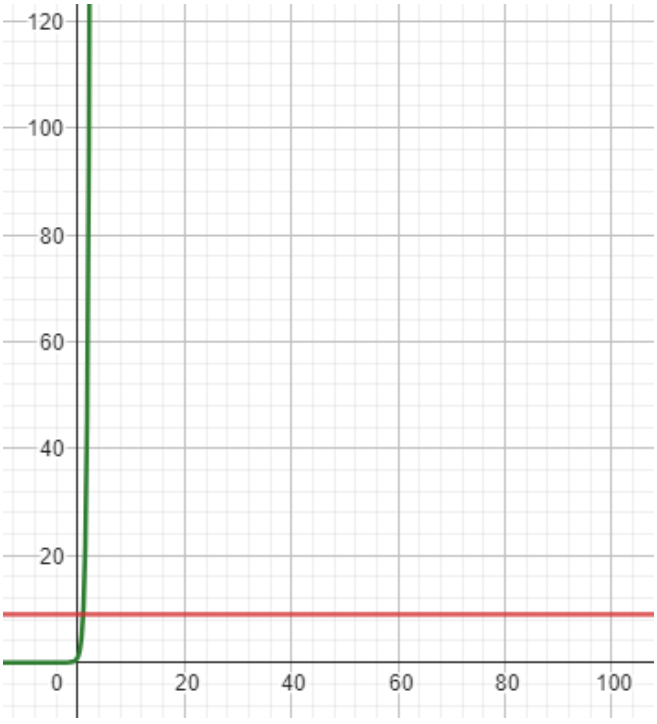
- Average Case :
  Dalam skenario average case, permainan biasanya berlangsung hingga langkah tengah atau mendekati langkah akhir, tetapi tidak semua cabang dieksplorasi. Kompleksitasnya bergantung pada kedalaman rata-rata permainan (biasanya mmm antara 6 hingga 8 dalam Tic-Tac-Toe), yang bisa di tulis dengan :

$$b^{rata\ rata\ main}$$

Tiga skenario di atas adalah skenario yang optimal digunakan untuk papan umum *Tic Tac Toe* yaitu 3 x 3. Dengan hasil tersebut, untuk meninjau ulan lagi untuk papan 4 x 4, 5 x 5, atau lebih dalam *Worse Case, Best Case,* dan *Average Case*.

5.2 RUNNING TIME

Berdasarkan penelitian di atas dengan dua algoritma, dapat disimpulkan dengan kompleksitas algoritma yang di dapatkan yaitu iteratif sebesar b^2 dan rekursif minimax sebesar b^m dan dapat digamabarkan dengan grafik seperti di bawah ini :



Grafik 1 : Perbandingan Running Time Iteratif dan Rekursif

Dengan grafik, yang ada di atas dengan keterangan merah yaitu algoritma iteratif dan hijau untuk algortima rekursif, dengan uji coba papan 3x3 dapat disimpulkan bahwa :

1. Algoritma Iteratif, memiliki grafik lurus yang berarti nilai konstan dan linier tergantung papan, diuji dengan 3 x 3 memiliki nilai yang konstan, tidak berubah, karena memang sifat perulangan yang sebesar papan.

2. Algortima Rekursif, memiliki grafik menaik menunjukkan bahwa running time algoritma rekursif bertambah seiring dengan pertambahan input. Pada papan 3x3, running time meningkat lebih cepat dibandingkan algoritma iteratif, karena rekursi cenderung menghasilkan overhead yang lebih besar.

6 KESIMPULAN

Dengan melakukan penelitian di atas, dapat disimpulkan hasil penelitian dan kesimpulan yaitu :

1. Algoritma Iteratif (Random-Iterative**)** memiliki kompleksitas waktu $O(b^2)$, di mana b adalah ukuran papan. Algoritma ini mudah diimplementasikan, tetapi kurang optimal untuk papan *Tic Tac Toe* yang lebih besar karena efisiensinya menurun secara signifikan dengan meningkatnya ukuran papan.
2. Algoritma Rekursif (Minimax) memiliki kompleksitas waktu $O(b^d)$, di mana b adalah *branching factor* yang berasal dari kotak kosong dan d adalah kedalaman pohon rekursi. Meskipun algoritma ini menjamin langkah optimal, kompleksitas waktunya dapat menjadi eksponensial terhadap ukuran ruang solusi, sehingga jika kasusnya pada papan yang lebih besar maka waktu eksekusi makin besar.

Penelitian ini memberikan wawasan tentang kekuatan dan kelemahan masing-masing pendekatan dalam pengembangan algoritma untuk permainan strategi. Algoritma iteratif lebih efisien untuk permainan dengan ruang solusi kecil, sedangkan algoritma rekursif lebih optimal untuk permainan dengan ruang solusi yang lebih besar, tetapi memerlukan optimasi untuk meningkatkan efisiensinya.

7 SARAN

Berdasarkan penelitian dan analisis yang dilakukan, berikut beberapa yang bermunculan dari penelitian ini :

a. Percobaan *Tic Tac Toe dengan* peningkatan algoritma, melihat kekurangan dari algoritma Minimax yang semakin besar papan, disarankan untuk menerapkan tektik optimasi seperti *alpha-beta pruning*.
b. Pengembangan Algoritma untuk skenario yang lebih kompleks.
c. Pengembangan algoritma Integrasi dengan *Machine Learning*, yang dapat memungkinkan sistem belajar dari permainan sebelumnya, lalu sistem dapat meningkatkan kemampuan algoritma dalam menghadapi pola permainan yang tidak terduga.

d. Peningkatan fungsi Evaluasi pada Memori, terutama pada papan besar. Implementasi struktur data yang lebih efisien, seperti *hash table* untuk menyimpan pola permainan yang telah dievaluasi, dapat membantu mengurangi penggunaan memori.
e. Mencari Algoritma Alternatif, selain algoritma Minimax, masih terdapat algoritma lain seperti *Monte Carlo Tree Search* (MCTS).