Penerapan Algoritma Backtracking pada Permainan TicTacToe Berbagai Variasi Ukuran

Ziyad Dhia Rafi - 13520064
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: ziyaddhrf21@gmail.com

***Abstract*—Permainan TicTacToe merupakan sebuah permainan papan untuk dua pemain yang dimainkan secara bergiliran. Pemain yang mendapat giliran mengisikan X atau O sesuai bagiannya pada kotak yang kosong. Pemain yang berhasil menempatkan 3 *mark* dalam garis horizontal, vertical, atau diagonal menjadi pemenangnya. (*Abstract*)**

Keywords—TicTacToe; minimax; Tic; Tac; Toe; Alpha-Beta; Pruning; backtracking; algorithm;

I. PENDAHULUAN

Menurut penelusuran pada zaan mesir Kuno tahun 1300SM, papan permainan 3x3 untuk TicTacToe ditemukan di genteng. Pada masa kekaisaran Romawi, Tic Tac Toe dikenal sebagai "terni lapilli" (tiga kerikil sekaligus). Tic Tac Toe pertama kali dipublikasikan di Inggris dengan nama "nought and crosses" dalam sebuah yang terbit per tiap bulan, yaitu *Notes and Queries*.

Bukan hanya Tic Tac Toe ini menjadi sebuah permainan kertas dan pensil, Tic Tac Toe juga merupakan salah satu dari *video game* yang paling pertama. Video game ini dikembangkan oleh seorang ahli komputer, Alexander S. Doughlas pada tahun 1952. Pada tahun 1975, mahasiswa MIT mengembangkan permainan Tic Tac Toe yang sempurna mengguankan sebuah komputer *Tinketoy*, yang saat ini telah dimuseumkan di *Museum of Science* di Boston.

Pada permainan TicTacToe ini ada pemain yang kalah, menang, atau keduanya berakhir imbang. Namun, bagaimana caranya agar kita selalu memenangkan pertandingan. Tentunya dengan bantuan komputer, kita akan sulit untuk dikalahkan dalam permainan ini . Pada makalah ini akan dibahas program dan algoritma untuk mencari gerakan terbaik agar bisa memenangkan permainan.

II. LANDASAN TEORI

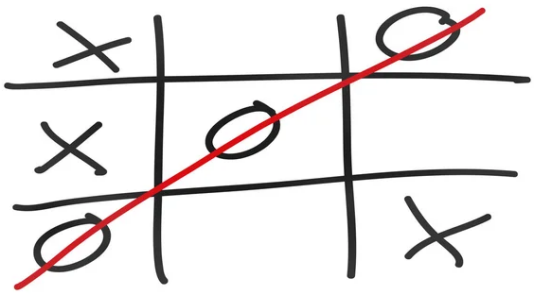
Untuk dapat memahami konsep algoritma pemilihan langkah terbaik pada permaianan TicTacToe dengan menggunakan algoritma backtracking, diperlukan beberapa pemahaman mengenai bidang berikut:

A. *TicTacToe*

TicTacToe merupakan sebuah permainan papan yang dimainkan oleh dua orang pada sebuah papan kotak yang terdiri dari 9 (3x3) kotak. Permainan ini sangat sederhana, pemain yang pertama kali menggambarkan 3 O atau X pada satu garis lurus, menjadi pemenangnya. Permainan ini hanya memerlukan sebuah kertas dan pensil, saat ini tic tac toe dapat dimainkan dengan mudah melalui perangkat seluler ataupun komputer.

1. TicTacToe 3x3

   TicTacToe 3x3 merupakan variasi klasik dari Tic Tac Toe.Permainan terdiri dari 3 baris, 4 kolom, dan memerlukan 3 *mark* dalam satu garis sebagai syarat menang. Apabila seluruh kotak sudah terisi, dan tidak ada yang berhasil menempatkan 3 X atau 3 O dalam satu garis, maka permainan berakhir imbang.



Gambar 2.1 Permainan *TicTacToe* 3x3
Sumber: https://st2.depositphotos.com/3591429/7819/i/600/depositphotos_78191464-stock-photo-tic-tac-toe-game.jpg

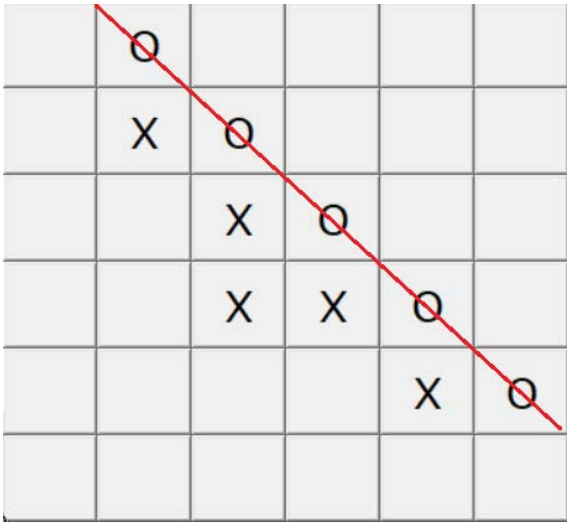
Dalam sebuah permainan Tic Tac Toe yang terdiri dari 9 kotak, terdapat 9! kemungkinan posisi (urutan Gerakan berpengaruh) yaitu sebesar 362880. Tentunya solusi dari permainan ini bisa dipecahkan dengan mudah dengan cara *brute force*.

Telah dibuktikan menggunakan algoritma *brute force*, permainan ini bila dimainkan secara sempurna, akan selalu memberikan hasil imbang, tidak peduli pemain, bermain sebagai X atau O (pertama

atau kedua). Kemenangan dapat diraih jika salah satu sisi melakukan Gerakan yang tidak akurat.

2. TicTacToe variasi lain (m,n,k) game

Pada makalah ini, akan dibahas juga variasi lain dari TicTacToe, yaitu m, n, k games. Permainan ini memiliki m baris, n kolom, serta k *mark* dibutuhkan dalam satu garis lurus untuk memenangkan permainan.



Gambar 2.1 Permainan m, n, k game (6,6,5)

Sumber: dokumen penulis

B. *Algoritma Backtracking*

Algoritma *backtracking* atau algoritma runut balik dapat dipandang dalam dua hal. Yang pertama, algoritma backtracking merupakan sebuah fase di dalam algoritma traversal DFS. Algoritma *backtracking* juga dapat dipandang sebagia sebuah metode pemecahan masalah yang mangkus , terstruktur, sistematis, baik untuk persoalan optimasi maupun non optimasi. Pada makalah ini, akan digunakan definisi kedua algoritma *backtracking*, yaitu sebagai metode.

Backtracking sebagai sebuah metode

- Merupakan perbaikan dari *exhaustive search*, dimana semua kemungkinan solusi dieksplorasi dan dievaluasi satu per satu
- Pada algoritma *backtracking*. Hanya pilihan yang mengarah ke solusi yang dieksplorasi
- Algoritma ini memangkas (*pruning*) simpul-simpul yang tidak mengarah ke solusi
- Algoritma *backtracking* pertama kali diperkenalkan oleh D. H. Lehmer tahun 1950.

Properti Umum Algoritma Backtracking

1. Solusi persoalan
   Solusi dinyatakan sebagai vector dengan n-tuple: X = (x_1, x_2, …, x_n), $x_i \in S_i$. Umumnya $S_1 = S_2 = S_n$
2. Fungsi pembangkit nilai x_k
   - Dinyatakan sebagai predikat T()
   - T(x[1], x[2], …, x[k – 1]) membangkitkan nilai untuk xk, yang merupakan komponen vektor solusi.
3. Fungsi pembatas
   - Dinyatakan sebagai predikat B($x_1, x_2, …, x_3$)
   - B bernilai true jika (x1, x2, …, xk) mengarah ke solusi. Mengarah ke solusi artinya tidak melanggar kendala (constraints)
   - Jika nilai true, maka pembangkitan untuk nilai x_{k+1} dilanjutkan, tetapi jika false, maka ($x_1, x_2, …, x_k$) dibuang

III. PEMBAHASAN DAN ANALISIS

Sebelum menggunakan algoritma *backtracking* dan menentukan langkah terbaik permainan pada setiap posisi, diperlukan berbagai properti yang diperlilan untuk menemukan solusi.

A. *Cost dan Fungsi Evaluasi*

Diperlukan sebuah nilai yang menyatakan kekuatan sebuah posisi, relatif terhadap pemain X maupun pemain O pada permainan TicTacToe atau m,n,k game. Rumus untuk menentukan nilai evaluasi tiap posisi dapat berbeda-beda sesuai dengan kebutuhan untuk meningkatkan keakuratan pencarian solusi.

Pada kali ini, akan digunakan sebuah algoritma evaluasi untuk permainan m,n,k game sebagai berikut

Nilai cost positif menyatakan X kuat relatif terhadap O. Sementara cost negatif menyatakan O kuat relatif terhadap X. Nilai cost maksimum adalah 10^{k-1}, sementara nilai cost minimum adalah -10^{k-1}.

m = baris

n = kolom

k = *mark* bertetangga yang dibutuhkan

Untuk setiap garis yang ada dengan panjang k, pada kotak berukuran m x n, akan dihitung jumlah X dan jumlah O dengan langkah-langkah sebagai berikut:

- Poin_O dan Poin_O diinisiasikan 0
- Apabila seluruh kotak pada garis seluruhnya merupakan X maka cost = nilai maksimum, perhitungan selesai
- Apabila seluruh kotak pada garis seluruhnya merupakan O maka cost = nilai minimum, perhitungan selesai
- Apabila hanya terdapat X dan tidak terdapat O, maka poin_X ditambah sebesar $10^{\text{jumlahX-1}}$
- Apabila hanya terdapat O dan tidak terdapat O, maka poin_O ditambah sebesar $-10^{\text{jumlah\_O-1}}$

- Apabila terdapat X dan O, atau tidak ada X atau O sama sekali, maka tidak akan ditambahkan poin

cost = poin_X – poin_O

Snippet code perhitungan evaluasi:

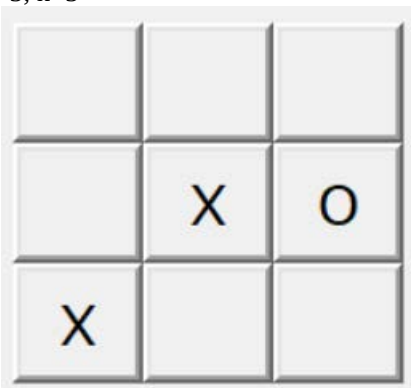
```
for line in indexes:
    XCount = 0
    OCount = 0
    for idx in line:
        char = self.buffer[idx]
        if char == 'X':
            XCount += 1
        elif char == 'O':
            OCount += 1
    if (XCount == self.wincount):
        return MAXPOINTS
    elif (OCount == self.wincount):
        return MINPOINTS
    elif (XCount != 0 and OCount == 0):
        X_points += 10**(XCount-1)

    elif (XCount == 0 and OCount != 0):
        O_points += 10**(OCount-1)
return X_points - O_points
```

Gambar 3.1 *Snippet* code perhitungan evaluasi
Sumber: dokumen penulis, program lengkap dapat dilihat di: https://github.com/ziyaddr/TicTacToeMinimax

contoh perhitungan evaluasi:

1. m=3; n=3; k=3

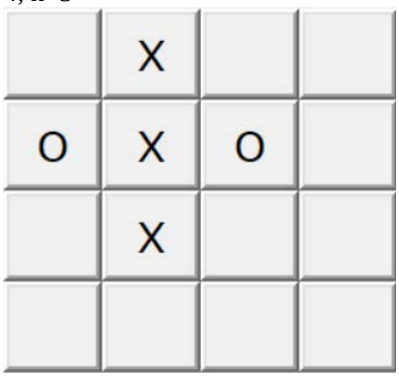


Gambar 3.2 Contoh kasus evaluasi TicTacToe (3x3)
Sumber: dokumen penulis

- Vertikal kiri = 1
- Vertikal tengah = 1
- Vertikan kanan = -1
- Horizontal atas = 0
- Horizontal tengah = 0
- Horizontal bawah = 1
- Diagonal atas-kiri – bawah-kanan = 1
- Diagonal atas-kanan – bawah-kiri = 10

total cost = 13

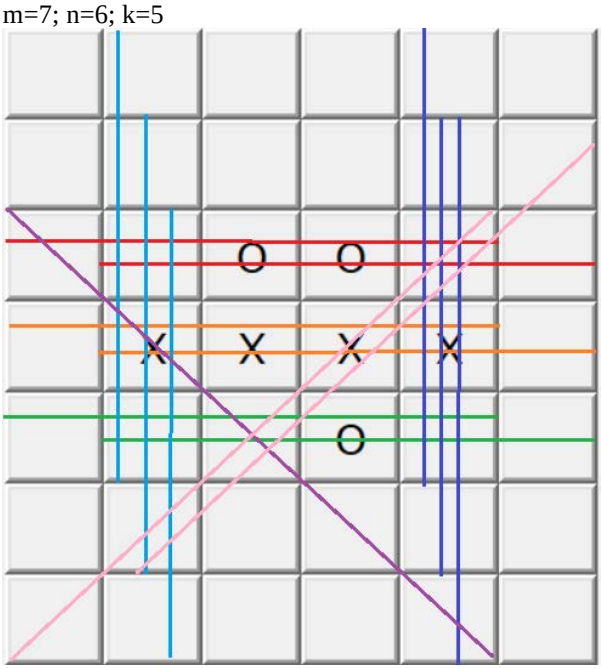
2. m=4; n=4; k=3



Gambar 3.3 Contoh kasus evaluasi m,n,k game (4, 4, 3)
Sumber: dokumen penulis

Vertikal 1-4 (kiri-kanan)
Horizontal 1-4(atas-bawah)
Vertikal 1 = -1
Vertikal 2 = 100, selesai
total cost = 100

3. m=7; n=6; k=5



Gambar 3.4 Contoh kasus evaluasi m,n,k game (7,6,5)
Sumber: dokumen penulis

Garis yang menghasilkan poin:
biru muda = 1 + 1 + 1
biru tua = 1 + 1 + 1
merah = -(10 + 10)
hijau = -(1 + 1)

oranye = 1000 + 1000
pink = 1 + 1
ungu = 1
total = 1987

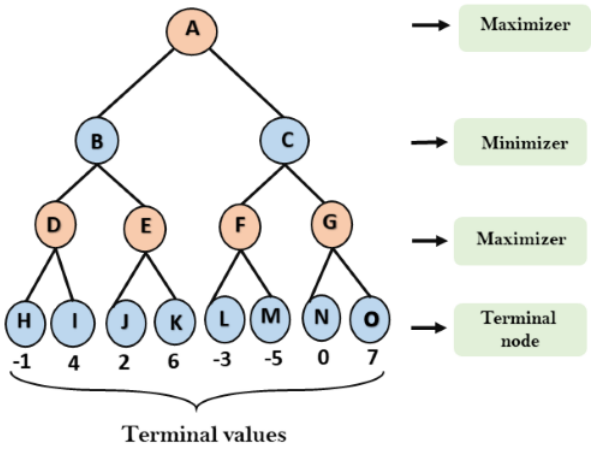
Dari perhitungan cost yang dihasilkan, akan dibentuk seluruh anak dari posisi tersebut yaitu seluruh kemungkinan posisi yang dipilih pemain selanjutnya. Setiap posisi permainan berubah, akan selalu dipanggil fungsi pembangkit, yang membangkitkan node anak hingga kedalaman akhir. Tentunya kedalaman atau *depth* dari pohon solusi ini harus dibatasi karena akan mengakibatkan perhitungan yang sangat lama.

Algoritma perhitungan cost dari sebuah state game tersebut tentunya tidak selalu merepresentasikan kekuatan sebenarnya, misalnya pada kasus 3 di atas, pin untuk evaluasi tersebut adalah 1987, akan tetapi, sebenarnya pemain X sudah pasti memenangkan permainan, terlepas dari apapun pilihan yang diberikan oleh pemain O. Oleh karena itu, nilai cost atau evaluasi dari game tersebut harusnya merupakan nilai maksimal, atau 10000 poin. Hal inilah yang mendasari algoritma Minimax yang akan dibahas selanjutnya.

B. *Algoritma Minimax*

Algoritma minimax merupakan sebuah algoritma *backtracking* yang digunakan untuk pengambilan sebuah keputusan sebuah permainan giliran, misalnya TicTacToe, catur, checkers, dll. Algoritma ini menyajikan Gerakan yang optimal untuk pemain, dengan asumsi lawan juga melakukan Gerakan yang optimal.

Dalam algoritma ini terdapat dua peran, yaitu *maximizer* yang selalu memilih cost maksimum, serta *minimizer* yang selalu memilih cost minimum. Algoritma ini menerapkan *backtracking* dengan urutan pencarian *depth-first-search*



Gambar 3.5 Algoritma minimax

Sumber: https://static.javatpoint.com/tutorial/ai/images/mini-max-algorithm-in-ai-step1.png

Terminal node atau daun merupakan state akhir sebuah permainan atau kedalaman maksimum yang dicapai

Untuk menghitung cost dari node A, digunakan langkah sebagai berikut:

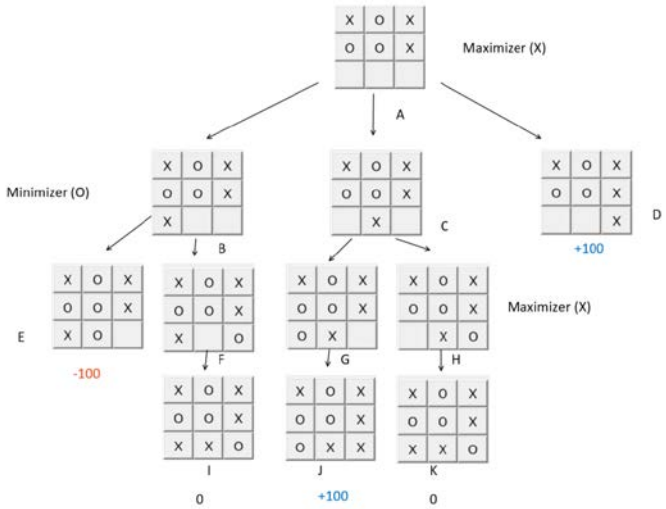
- Buat pohon keputusan untuk setiap kemungkinan gerakan hingga mencapai state akhir yang menghasilkan *terminal node*
- Hitung cost tiap *terminal node*
- Node D, E, F, G Maximizer:
  D = max(-1. 4) = 4
  E = max(2, 6) = 6
  F = max(-3, -5) = -3
  G = max(0, 7) = 7
- Node B, C Minimizer:
  B = min (4, 6) = 4
  C = min(-3, 7) = -3
- Node A Maximizer:
  A = max(4, -3) = 4

Dari hasil perhitungan, didapat gerakan yang optimal adalah ke kiri (ke Node B).

Minimax pada TicTacToe

Minimax pada TicTacToe memiliki prinsip yang sama dengan Minimax pada umumnya. Pemain X, atau pemain yang jalan pertama berperan sebagai maximizer, sementara pemain O berperan sebagai minimizer. Mula-mula pohon keputusan dibuat hingga mencapai *terminal node*.

contoh:



Gambar 3.6 Demonstrasi algoritma *minimax TicTacToe*
sumber: dokumen penulis

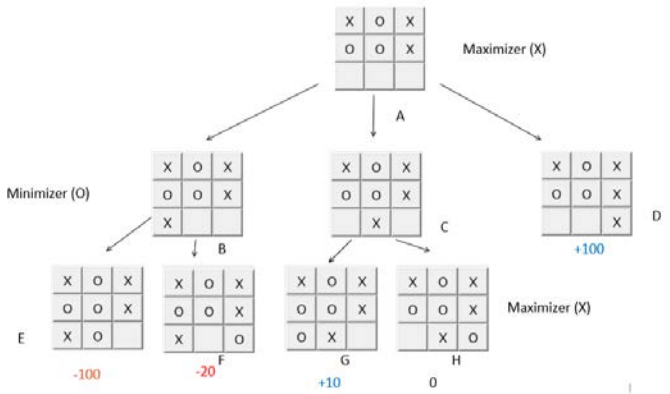
Dengan menggunakan langkah-langkah algoritma minimax dan fungsi perhitungan cost yang sudah ada sebelumnya, didapat:

F = 0
G = 100
H = 0
B = -100
C = 0
D = 100
A = 100
oleh karena itu, didapat langkah paling optimal adalah X(3,3)

Minimax pada TicTacToe menggunakan depth maksimum

Dengan menggunakan kedalaman pohon maksimum, terminal node tidak selalu merupakan state akhir dari permainan. Pencarian diakhiri hingga mencapai kedalaman tertentu. Dengan menerapkan *maximum depth* ini, pemilihan gerakan optimal tidak akan memakan waktu yang lama. Akan tetapi cara ini mungkin menghasilkan solusi yang kurang optimal dibanding tanpa batas kedalaman. Semakin besar kedalaman, akan semakin optimal hasilnya, namun semakin lama waktu dibutuhkan untuk kalkulasi. Algoritma minimax dengan kedalaman terenda atau 1, adalah sebuah algoritma Greedy yang hanya memperhatikan cost maksimum atau minimum lokal.

Contoh kasus yang sama pada bagian sebelumnya, menggunakan kedalaman maksimum 2:



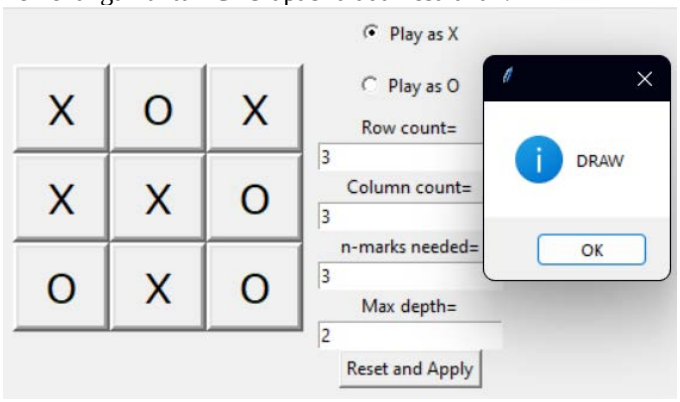
Gambar 3.7 Demonstrasi algoritma *minimax* pada *TicTacToe* dengan kedalaman 2
sumber: dokumen penulis

Dengan menggunakan minimax dengan kedalaman maksimum 2 dan perhitungan cost yang sudah ada sebelumnya, didapat hasil sebagai berikut:

- E = -100 (Endstate)
- F = -20
- G = 10
- H = 0
- B = -100
- C = 0
- D = 100
- A = 100

Dari kalkulasi di atas, didapat cost yang berbeda untuk node yang ada, tetapi tetap memberikan keputusan yang sama untuk setiap posisi permainan.

Penggunaan minimax dengan kedalaman maksimum 2 pada TicTacToe, selalu menghasilkan gerakan yang paling optimal. Dengan menggunakan program yang telah dibuat, dengan kedalaman 2, CPU tidak akan bisa dikalahkan dan hasilnya akan selalu imbang dengan *best play* atau kemenangan untuk CPU apabila ada kesalahan.



Gambar 3.8 AI *TicTacToe* 3x3
sumber: dokumen penunlis

```
def setCostAndExpand(self, currentDepth):
    MAXPOINTS = 10**(self.matrix.wincount-1)
    MINPOINTS = -MAXPOINTS
    points = self.matrix.calculate()
    # self.cost = points
    # one side win
    if (points == MAXPOINTS or points == MINPOINTS):
        self.noExpand = True
        self.cost = points
        return

    # draw end
    if (self.turns == self.matrix.row*self.matrix.col):
        self.noExpand = True
        self.cost = 0
        return


    if (currentDepth < self.maxdepth):
    ## keep expanding
        self.generateMoves()
```

Gambar 3.9 *Snippet* kode penggunaan algoritma minimax pada *TicTacToe*
Sumber: dokumen penulis, program lengkap dapat dilihat di: https://github.com/ziyaddr/TicTacToeMinimax

Minimax pada m, n, k

Berbeda dengan TicTacToe yang menghasilkan hasil imbang untuk setiap permainan dengan permainan terbaik, m,n,k game tidak diketahui secara pasti apakah dengan permainan terbaik, permainan menghasilkan *forced win* atau *forced draw*.

Algoritma *minimax* pada m, n, k game tidak berbeda dengan *minimax* pada TicTacToe. Akan tetapi, pada m, n, k game waktu yang dibutuhkan untuk kalulasi akan jauh lebih

besar. Semakin besar m, n, serta k akan waktu yang dibutuhkan untuk kalkulasi tiap gerakannya akan meningkat secara signifikan. Oleh karena itu, pemilihan depth akna sangat berpengaruh dalam permainan ini.

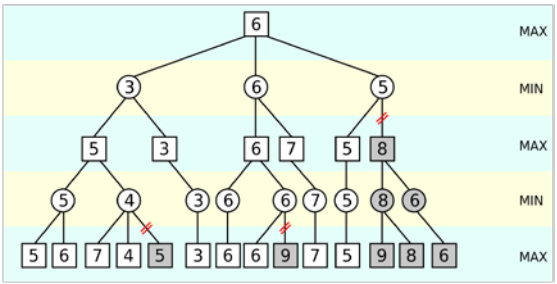
Pada program yang telah dibuat, TicTacToe (3x3) dapat menggunakan kedalaman 2 hingga 5. Untuk m, n, k game dengan ukuran lebih besar dari (5x5), disarankan untuk menggunakan kedalaman 2 atau 3 agar waktu perhitungan tidak terlalu lama.

Saat ini belum ada berbagai riset mengenai hasil akhir dari m, n, k game. Hasil akhir dari game tersebut cukup sulit untuk dibuktikan disebabkan banyaknya kombinasi gerakan yang dapat dihasilkan. Meskipun demikian, CPU yang menggunaan algoritma *minimax* meski hanya dengan kedalaman 2, sangat sulit atau bahkan terlihat tidak bisa untuk dikalahkan.

C. *Alpha-Beta Pruning*

Alpha-Beta Pruning merupakan sebuah metode atau fungsi pembatas, untuk mengoptimasi algoritma backtracking (*minimax*). Dengan cara ini, program membatasi dan "membunuh" node yang dianggap tidak menuju ke solusi, dan menyisakan node yang berpotensi menuju sokusi.

Alpha-Beta Pruning dipastikan memberikan hasil yang sama dengan algoritma minimax biasa. Dengan menggunakan metode ini, sebuah program dapat melakukan pencarian solusi jauh lebih cepat dibanding tanpa menggunakan *Alpha-Beta Pruning*. Dengan demikian, program dapat melakukan pencarian untuk kedalama yang lebih jauh lagi.



Gambar 3.10 Demonstrasi *Alpha-Beta Pruning*
sumber: https://thelinuxos.com/wp-content/uploads/2020/04/Alpha-Beta-Pruning-Artificial-Intelligence.png

Untuk setiap anak dari sebuah node. Apabila ditemukan node dengan cost yang lebih baik, maka node tersebut akan langusng "dibunuh", hingga menyisakan node anak terbaik. Baik disini berarti lebih besar pada fase *Maximizer* atau lebih kecil pada fase Minimizer

```
for i in range(len(self.children)):
    child = self.children[i]
    child.setCostAndExpand(currentDepth+1)
    value = child.costR
    child.noExpand = True
    if (value > maxVal):
        maxVal = value
        if maxNode != None:
            maxNode.noExpand = True
        maxNode = child
        child.noExpand = False
    if (value < minVal):
        minVal = value
        if minNode != None:
            minNode.noExpand = True
        minNode = child
        child.noExpand = False
self.children.sort(key=lambda x: x.cost, reverse=True) ## always sorted
```

Gambar 3.11 *Snippet* kode penggunaan *pruning* pada algoritma minimax *TicTacToe*
Sumber: dokumen penulis, program lengkap dapat dilihat di: https://github.com/ziyaddr/TicTacToeMinimax

IV. Kesimpulan dan Saran

A. *Kesimpulan*

Penerapan algoritma *backtracking* dalam permainan TicTacToe akan memberikan pilihan/solusi terbaik secara lokal .dengan kedalaman tertentu. Solusi yang dihasilkan tidak selalu memberikan solusi optimum global karena keterbatasan komputer, sehingga pencarian perlu dibatasi hingga kedalaman tertentu. Penggunaan algoritma *backtracking* dengan *minimax* dan *Alpha-Beta Pruning* banyak digunakan untuk AI permainan giliran 2 pemain karena dapat memberikan solusi yang baik, dengan waktu pencaraian yang jauh lebih singkat dibanding menggunakan algoritma *Brute* Force

B. *Saran*

Dalam makalah ini, penulis sadari, masih banyak terdapat kekurangan, oleh karena itu, terdapat beberapa saran yang dapat membuat makalah ini menjadi lebih baik lagi:

1) Program dapat dibuat lebih efektif dan efisien
2) Lebih banyak memberikan gambar dan visualisasi agar lebih mudah dipahami.

Video Link at Youtube *(Heading 5)*

https://youtu.be/Q4Vafn88iIc

Ucapan Terima Kasih *(Heading 5)*

Puji Syukur Penulis sampaikan kepada Tuhan Yang Maha Esa, karena berkat rahmat-Nya, penulis dapat meyelesaikan makalah ini. Penulisan makalah ini dilakukan dalam rangka memenuhi salah satu tugas mata kuliah IF2211 Strategi Algoritma semester 4 tahun 2021/2022.

Penulis mengucapkan terima kasih kepada:

1. Dosen-dosen pengajar mata kuliah IF2211 Strategi Algoritma
2. Orang tua
3. Teman dan sahabat

karena berkat merekalah, saya dapat menyelesaikan makalah ini dengan baik.

Penulis menyyadari, dalam penulisan makalah ini, masih terdapat banyak kekurangan. Oleh karena itu, diharapkan kritik, serta saran dapat disampaikan kepada penulis, untuk menyempurnakan karya tulis ilmiah ini.

Akhi kaya, penulis mengucapkan terimakasih, dan semoga makalah ini dapat bermanfaat bagi segala pihak yang membutuhkan.

REFERENSI

[1]R. Hariadi, I. Kuswardayan, I. Arieshanti and I. Alfi T.Z, "Penerapan Algoritma Alphabeta Pruning Sebagai Kecerdasan Buatan pada Game Pawn Battle", *JURNAL INFOTEL*, vol. 9, no. 2, p. 185, 2017. Available: 10.20895/infotel.v9i2.166.

[2]A. CHOI, "TIC-TAC-TOE", *Momath.org*. [Online]. Available: https://momath.org/wp-content/uploads/2021/08/Alyssa-Choi-Tic-Tac-Toe.pdf.

[3]A. Nugraha, "Membuat AI TicTacToe dengan Algoritma Alpha–beta pruning —Javascript", *Medium*, 2020. [Online]. Available: https://viandwi24.medium.com/membuat-ai-tictactoe-dengan-algoritma-alpha-beta-pruning-javascript-2508006fd090.

[4]"Artificial Intelligence | Alpha-Beta Pruning - Javatpoint", *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/ai-alpha-beta-pruning.

[5]"Artificial Intelligence | Mini-Max Algorithm - Javatpoint", *www.javatpoint.com*. [Online]. Available: https://www.javatpoint.com/mini-max-algorithm-in-ai.

[6]"Tic-Tac-Toe (Noughts & Crosses): 20+ Interesting Facts (Trivia, History,…)", *Gamesver*. [Online]. Available: https://www.gamesver.com/tic-tac-toe-noughts-crosses-interesting-facts-trivia-history/.

[7]R. Munir, *Informatika.stei.itb.ac.id*, 2022. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-backtracking-2021-Bagian1.pdf.

Source code: https://github.com/ziyaddr/TicTacToeMinimax

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 21 Mei 2022

Ttd

Ziyad Dhia Rafi 13520064