Implmentasi Algoritma Minimax dan *Alpha Beta Pruning* pada AI dalam Permainan *Connect 4*

Rexy Gamaliel Rumahorbo–13519010
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: 13519010@sd.stei.itb.ac.id

***Abstract*—Kecerdasan buatan telah banyak diimplementasikan di banyak bidang, salah satunya adalah pencarian solusi terbaik dalam suatu permainan. Pada permainan Connect 4, terdapat banyak kemungkinan posisi yang dapat ditelusuri–salah satunya–dengan DFS. AI dapat memanfaatkan algoritma *minimax* dan *alpha beta pruning* dalam pembuatan keputusan dalam permainan tersebut. Algoritma *minimax* menentukan bagaimana pengambilan keputusan dibuat, sementara algoritma *alpha beta pruning* meningkatkan efisiensi algoritma *minimax*. Sementara itu, karena katerbatasan komputasi, AI perlu menggunakan *iterative deepening search* agar komputasi dapat dilakukan dalam waktu terbatas**

Keywords—DFS; minimax; alpha beta pruning; iterative deepening search

I. INTRODUCTION (*HEADING 1*)

Kecerdasan buatan/*artificial intelligence* (AI) merupakan salah sata ranah keinformatikaan yang sedang berkembang pesat saat ini. Perkembangan teknologi mendukung pemanfaatan tenaga komputasional untuk melakukan suatu pekerjaan dengan bantuan komputer. Pemanfaatan ini salah satunya berkembang menjadi suatu kecerdasan buatan. AI dikenal sebagai suatu algoritma tertentu yang dapat melakukan suatu tugas atau memecahkan suatu masalah dengan performa yang mampu mengalahkan manusia. Penerapan AI pun mencakup banyak bidang serta kompleksitas yang beragam.

Pada dasarnya kecerdasan buatan/AI memanfaatkan beberapa algoritma tertentu yang dapat membantu dalam menyelesaikan persoalan secara efisien. Saat ini telah banyak algoritma dan pemodelan yang telah dikembangkan dan diteliti, serta digunakan untuk menyelesaikan suatu persoalan tertentu.

Salah satu implementasi kecerdasan buatan dalam permainan adalah pemanfaatan algoritma *minimax* dan *alpha beta pruning* dalam permainan Connect 4. Pada permainan ini, terdapat dua pemain yang bergantian untuk memasukkan koin warna masing-masing ke dalam suatu papan berukuran 6x7 hingga membentuk suatu garis dengan 4 koin berwarna sama.

Pada permainan Connect 4, AI akan menelusuri kemungkinan-kemungkinan yang ada dan memilih gerakan terbaik untuk menghasilkan posisi yang terbaik, sembari mempertimbangkan kemungkinan yang akan dipilih lawan dan memangkas kemungkinan-kemungkinan dapat dipastikan tidak akan memberikan hasil yang optimal. Hal tersebut dapat diimplementasikan dengan algoritma *minimax*, di mana terdapat pergantian giliran antara pemain yang ingin mengoptimalkan keuntungan masing-masing, serta algoritma *alpha beta pruning*, yang digunakan untuk memangkas kemungkinan-kemungkinan yang tidak perlu demi meningkatkan performa AI.

II. TEORI DASAR

A. *Connect 4*

Permainan Connect 4 adalah salah satu permainan papan di mana terdapat dua pemain yang bergiliran untuk memasukkan koin ke dalam suaut papan vertikal. Masing-masing pemain akan memasukkan koin berwarna tertentu–umumnya kuning dan merah–sepanjang permainan hingga terdapat suatu garis lurus yang dibentuk 4 koin berwarna sama atau papan permainan penuh. Pemain yang berhasil membentuk garis horizontal, vertikal, atau diagonal yang terdiri dari 4 koin miliknya memenangkan permainan.

Gambar 1 Connect 4 oleh Popperipopp di bawah lisensi

Papan yang digunakan pada permainan ini berukuran 6x7, yakni 6 baris dan 7 kolom yang masing-masing membentang secara horizontal dan vertikal. Pada setiap giliran, pemain akan memasukkan koinnya ke dalam salah satu dari 7 kolom yang tersedia, sehingga koin tersebut jatuh dan menempati posisi terbawah yang tersedia. Jika suatu kolom sudah

dipenuhi koin, maka pemain tidak dapat memasukkan koin ke dalam kolom tersebut.

Secara garis besar, strategi permainan yang umum digunakan adalah dengan menaruh koin pada kolom-kolom yang berdekatan agar memungkinkan terbentuk garis, serta memanfaatkan tumpukan koin yang sudah ada untuk membentuk garis diagonal yang memerlukan ketinggian berbeda-beda pada setiap kolomnya untuk membentuk suatu garis. Selain itu, pemain juga dapat menaruh koin pada posisi tertentu sehingga menghalangi lawan untuk mebentuk garis. Permainan ini memerlukan kejelian karena pemain dapat dengan mudahnya tidak menyadari kemungkinan membentuk suatu garis di tengah-tengah banyaknya tumpukan koin yang ada.

B. Depth-First Search

Depth-First Search (DFS) atau pencarian mendalam adalah suatu strategi algoritma dalam penelusuran graf di mana anak suatu simpul suatu graf ditelusuri terlebih dahulu sebelum menelusuri simpul lainnya yang bertetanggaan; berlawanan dengan itu adalah *Breadth-First Search* (BFS) di mana simpul tetangga ditelusuri sebelum simpul-simpul anak dari simpul yang telah ditelusuri.

Terdapat beberapa modifikasi yang dapat dilakukan untuk mengoptimalisasi DFS pada beberapa kasus tertentu, antara lain adalah *backtracking*, dan *iterative deepening search*.

Pada algoritma runut balik/*backtracking*, pencarian DFS pada suatu simpul tidak diteruskan jika memenuhi suatu kondisi tertentu. Pada penerapannya, kasus ini yang 'dipangkas' ini dipastikan tidak memberikan suatu solusi sehingga penerapan *backtracking* akan menghindari algoritma untuk menelusuri simpul yang tidak perlu.

Sementara itu, *iterative deepening search* adalah suatu metode pencarian mendalam/DFS di mana kedalaman pencarian dibatasi pada tingkatan tertentu, di mana batas tersebut dapat diiterasi menjadi lebih dalam. Metode ini dilakukan karena pada umumnya penelusuran DFS akan memerlukan kedalaman yang sangat tinggi sehingga akan memakan sangat banyak waktu untuk menelusuri semua kemungkinan. Dengan memangkas kemungkinan yang ditelusuri pada tingakatan tertentu, program dapat mempersingkat komputasi serta memperkirakan hasil evaluasi simpul hingga kedalaman tertentu.

C. Algoritma Minimax

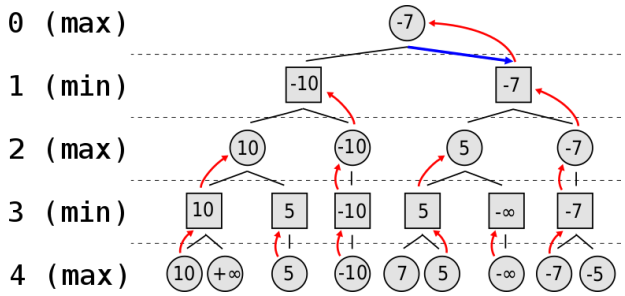
1) Definisi

Algoritma *minimax* adalah algoritma yang digunakan dalam kecerdasan buatan pada skenario di mana diperlukan untuk mengambil Tindakan yang menyebabkan kerugian sekecil mungkin dari sekumpulan skenario yang menyebabkan kerugian semaksimal mungkin[1].

2) Algoritma

Salah satu penerapan algoritma ini adalah pada permainan berbasis giliran/*turn-based game*, di mana terdapat beberapa pemain yang melakukan suatu aksi secara bergilir. Pemain akan mengusahakan gerakan terbaik dengan mempertimbangkan gerakan terbaik lawan. Dengan kata lain, pemain meminimalisir kerugian pada dirinya, dari suatu kumpulan kemungkinan gerakan yang akan dipilih lawan yang memaksimalkan kerugian bagi pemain–atau memaksimalkan keuntungan bagi lawan tersebut.

Algoritma minimax dapat divisualisasikan dengan suatu pohon ruang status dengan simpul yang menggambarkan kemungkinan-kemungkinan yang ada, di mana simpul-simpul dievaluasi menjadi sebuah angka yang menggambarkan keuntungan bagi pemain–semakin besar angkanya makan semakin menguntungkan pemain. Sementara itu, percabangan menggambarkan kemungkinan-kemungkinan yang mungkin dicapai dari suatu simpul. Simpul-simpul pada pohon ruang status dibagi ke dalam kedalamannya masing-masing. Pohon ruang status diawali dari kedalaman 0 yang merupakan status/*state* saat ini.



Gambar 2 Contoh algoritma *minimax* oleh Nuno Nogueira (Nmnogueira) di bawah lisensi

Pada umumnya–seperti pada *turn-based game* dengan dua pemain–pada setiap kedalaman, pemain akan secara bergantian untuk mencari nilai maksimum/minimum dari seluruh kemungkinan yang ada dari hasil evaluasi seluruh anak simpul. Jika pada kedalaman 3 pemain mendapat giliran, maka pemain akan mencari simpul yang memberikan evaluasi tertinggi dari semua simpul pada kedalaman 4, sementara itu lawan akan mencari simpul yang memberikan evaluasi terendah dari semua simpul pada kedalaman 5, dst. Pola pencarian nilai maksimum-minimum ini terus dilakukan hingga suatu simpul ujung yang dapat ditentukan nilai evaluasinya. Pada akhir dari alur algoritma ini adalah nilai evaluasi simpul pada akar yang merepresentasikan nilai terbaik yang dapat didapat pemain.

Pada contoh gambar di atas, simpul-simpul pada kedalaman 3 akan mencari nilai minimum dari simpul-simpul pada kedalaman 4. Dari proses tersebut dihasilkan simpul dengan hasil evaluasi 10 (dari perbandingan 10 dan +∞), 5, -10, 5 (dari hasil perbandingan 7 dan 5), -∞, -7 (dari hasil perbandingan -7 dan -5). Proses serupa dilakukan pada kedalaman 2 saat mencari nilai maksimum dari simpul-simpul yang ada pada kedalaman 3. Dari proses tersebut didapat 10 (dari hasil perbandingan 10 dan 5), -10, 5 (dari hasil perbandingan 5 dan -∞), dan -7. Hal ini dilakukan hingga didapat nilai dari simpul akar adalah -7. Hal ini berarti, dalam skenario di mana pemain dan lawan melakukan langkah terbaik, lawan akan unggul disbanding pemain sebesar 7 poin.

3) Implementasi

Algoritma *minimax* dapat diimplementasikan ke dalam

```
function minimax(node, depth, maximizingPlayer)
  if depth = 0 or node is a terminal node then
    return the heuristic value of node

  if maximizingPlayer then
    maxVal := −∞
    for each child of node do
      value = minimax(child, depth − 1, FALSE)
      maxVal := max(value, maxVal)
    return maxVal
  else (* minimizing player *)
    minVal := +∞
    for each child of node do
      value = minimax(child, depth − 1, TRUE)
      minVal := min(value, minVal)
    return minVal

{ function call example }
minimax(state, 7, TRUE)
```

fungsi rekursif yang mengembalikan nilai hasil evaluasi suatu simpul.

Pseudocode *minimax*

Parameter yang diperlukan pada fungsi minimax adalah `node`, `depth`, dan `maximizingPlayer`. Variabel `node` merepresentasikan status dari permainan, `depth` mewakili kedalaman DFS yang perlu dicari, dan `maximizingPlayer` adalah boolean yang mewakili apakah yang mewakili giliran permainan. Variabel `depth` akan bernilai 0 ketika fungsi minimax telah mencapai `node` dengan kedalaman maksimal. Variabel `maximizingPlayer` akan bernilai `TRUE` jika pemain yang memaksimalkan nilai sedang mendapat giliran.

D. ALPHA BETA PRUNING

1) Definisi

Algoritma *alpha beta pruning* merupakan implementasi *backtracking* pada algoritma *minimax*. Pada saat pemanggilan algoritma *minimax* untuk menelusuri simpul-simpul pada pohon ruang status, akan ada banyak dilakukan penelusuran simpul. Jumlah simpul akan bertambah secara eksponensial seiring betambahnya kedalaman penelusuran. Oleh karena itu, diperlukan optimisasi pada algoritma ini agar tetap efisien dan dapat memberikan solusi yang optimal.

2) Algoritma

Algoritma *alpha beta pruning* digunakan untuk ‘memangkas’ (atau melakukan *pruning* pada) simpul-simpul yang dipastikan tidak memberi solusi yang optimal. Pada algoritma ini digunakan dua buah variabel: *alpha*, yang mewakili nilai tertinggi sejauh ini yang diperoleh pemain yang memaksimalkan nilai, dan *beta*, yang mewakili nilai terendah sejauh ini yang diperoleh pemain yang meminimalkan nilai.

Pada awal pemanggilan algoritma *minimax*, kedua variabel diberi nilai ekstremum masing-masing negatif tak hingga (-∞) dan tak hingga (∞). Kedua nilai ini akan diperbarui ketika terdapat nilai evaluasi simpul yang lebih optimal, baik bagi pemain yang memaksimalkan maupun yang meminimalkan .nilai.

3) Implementasi

Algoritma *alpha beta pruning* dapat ditambahkan pada algoritma *minimax* sebelumnya menjadi sebagai berikut.

```
function minimax(node, depth, alpha, beta,
maximizingPlayer)
  if depth = 0 or node is a terminal node then
    return the heuristic value of node

  if maximizingPlayer then
    maxVal := −∞
    for each child of node do
      value = minimax(child, depth − 1, FALSE)
      maxVal := max(value, maxVal)
      alpha := max(alpha, value)
      if beta <= alpha then
        break
    return maxVal
  else (* minimizing player *)
    minVal := +∞
    for each child of node do
      value = minimax(child, depth − 1, TRUE)
      minVal := min(value, minVal)
      beta := min(beta, value)
      if beta <= alpha then
        break
    return minVal

{ function call example }
minimax(state, 7, -infinity, infinity, TRUE)
```

Pseudocode algoritma *minimax* dengan *alpha beta pruning*

Sedikit berbeda dengan algoritma *minimax* biasa, terdapat variabel `alpha` dan `beta` yang dipakai untuk melakukan *pruning*.

References

[1] http://blog.gamesolver.org/solving-connect-four/01-introduction/

[2] https://www.freecodecamp.org/news/playing-strategy-games-with-minimax-4ecb83b39b4b/

[3] https://www.oxfordreference.com/view/10.1093/oi/authority.20110803095944934

[4] Sebastian Lague. "Algorithms Explained – minimax and alpha-beta pruning". YoutTube. 5 Mei 2021, https://https://www.youtube.com/watch?v=l-hh51ncgDI

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, 11 Mei 2021

Rexy Gamaliel Rumahorbo (13519010)