Implementasi Minimax Algorithm dan Local Search pada Permainan Dots and Boxes

Tugas Besar I IF3170 Inteligensi Buatan

Disusun oleh:

Marcellus Michael H.K.	**13520057**
Dimas Shidqi Parikesit	**13520087**
Saul Sayers	**13520094**
Maria Khelli	**13520115**

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

1. Fungsi objektif

Ide dari fungsi objektif kita adalah menghitung nilai dari tiap grid untuk state yang sedang diperiksa. State yang menguntungkan player 2 akan diberikan nilai yang lebih positif, sementara state yang menguntungkan player 1 diberikan nilai yang lebih negatif. Ketentuan nilai dari grid adalah sebagai berikut :

a. Sebuah grid yang memiliki sisi sebanyak 0, 1, atau 2 bernilai 0
b. Sebuah grid yang memiliki empat sisi (lengkap) dan dimiliki oleh player 2 (grid merah) bernilai 100
c. Sebuah grid yang memiliki empat sisi (lengkap) dan dimiliki oleh musuh bernilai -100
d. Sebuah grid yang memiliki tiga sisi dan state tersebut merupakan turn dari player 2 bernilai 10
e. Sebuah grid yang memiliki tiga sisi dan state tersebut merupakan turn dari player 1 bernilai -10

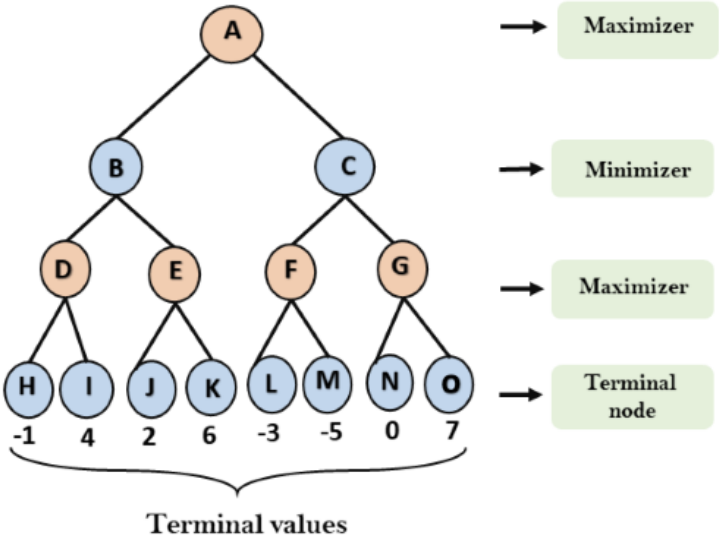
Sehingga apabila diformulasikan maka fungsi objektifnya adalah sebagai berikut:

$$f(s1, s2, a, p) = 100 * (s2 - s1) + 10 * a * (-1)^p$$

Dengan s1 adalah banyaknya kotak lengkap player 1, s2 adalah banyaknya kotak lengkap player 2, a adalah almost square yaitu grid yang memiliki sisi sebanyak 3, dan p adalah giliran player pada state itu (1 untuk giliran player 1 dan 2 untuk giliran player 2).

Dalam program, fungsi objektif kami di diimplementasikan dengan memeriksa objek gamestate yang diberikan. Variabel s1, s2, dan a diperoleh dari memeriksa atribut board_status menggunakan looping. Cara memperoleh s2 adalah banyaknya angka 4, s1 adalah banyaknya angka -4, dan a adalah banyaknya angka -3 atau 3. p diperoleh dari atribut isPlayer1Turn, apabila true maka p adalah 1 dan apabila false maka p adalah 2.

2. Algoritma minimax dengan *alpha-beta pruning*



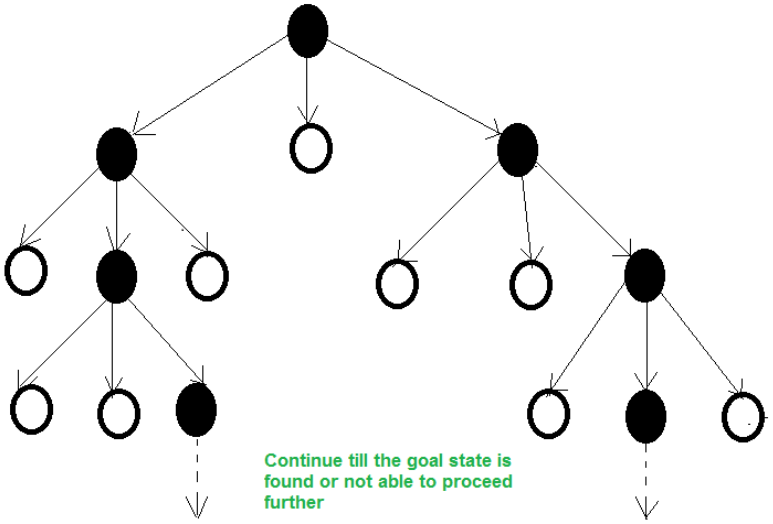
Gambar 2.1 Ilustrasi algoritma minimax

Algoritma minimax merupakan metode pencarian dengan prinsip memaksimalkan nilai objective function bagi 1 pemain dan meminimumkan nilai bagi pemain lain. Untuk melakukan hal tersebut, algoritma ini perlu mencari seluruh ruang kemungkinan gerakan yang dapat diambil beserta nilai objective function nya.

Mencari seluruh ruang kemungkinan gerakan ini memerlukan komputasi yang berat, karena pada tiap pencarian gerakan akan dilakukan perhitungan nilai objective function serta deteksi terbentuknya persegi baru pada permainan. Setelah dicari semua kemungkinan gerakan, akan diterapkan algoritma minimax menggunakan nilai objective function pada semua ruang gerakan yang telah dicari. Algoritma minimax diterapkan dengan menggunakan alpha beta pruning untuk mengurangi kemungkinan gerakan yang perlu dicek lebih jauh.

Karena terdapat constraint maksimal lima detik komputasi untuk tiap gerakan, maka perlu diatur seberapa dalam ruang algoritma yang akan dicari oleh algoritma. Berdasarkan tuning yang dilakukan, didapatkan bahwa kedalaman empat merupakan kedalaman maksimal yang dapat dilakukan, dengan total kemungkinan gerakan berjumlah 24x23x22x21 yaitu 255.024 gerakan.

3. Algoritma *local search*.



Gambar 3.1 Ilustrasi algoritma Beam Search

Algoritma Local search merupakan metode pencarian solusi berdasarkan neighborhood dari solusi awal. Pada umumnya, algoritma Local Search merupakan algoritma yang path irrelevant dan solusi nya berupa final state.

Algoritma Local Search yang digunakan oleh kelompok kami adalah algoritma Beam Search. Cara kerja algoritma Beam Search cukup mirip dengan algoritma Minimax, yaitu dengan memaksimalkan jumlah skor yang dimiliki oleh Bot dan meminimalkan jumlah skor yang dimiliki oleh lawan. Perhitungan skor menggunakan fungsi objektif yang telah didefinisikan sebelumnya.

Akan tetapi, perbedaan antara algoritma Beam Search dan algoritma Minimax adalah pada algoritma Beam Search, jumlah anak yang akan dilakukan pengecekan dibatasi, yaitu sebanyak W. Untuk memaksimalkan, jumlah anak akan diurutkan berdasarkan skor yang tertinggi, lalu diambil W terbaik. Untuk meminimalkan, jumlah anak akan diurutkan berdasarkan skor terendah, lalu diambil W terburuk.

Alasan kenapa kelompok kami memilih algoritma ini adalah algoritma Beam Search turut serta mempertimbangkan pergerakan lawan hingga kedalaman H. Kami berasumsi bahwa jika mempertimbangkan hingga ke kedalaman H, algoritma Beam Search akan bekerja dengan lebih baik karena semakin banyak hal yang dipertimbangkan dalam pengambilan keputusan oleh Bot.

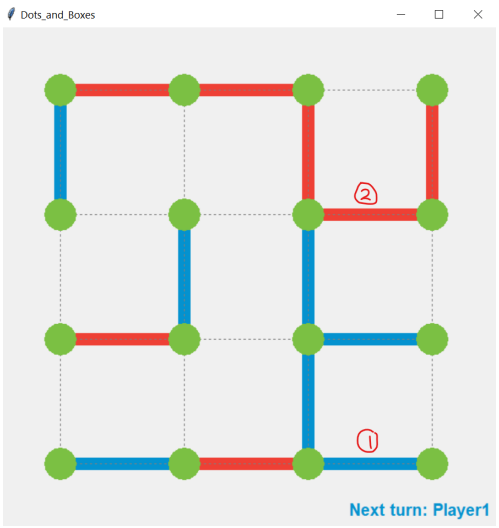
4. Analisis hasil pertandingan

4.1 Minimax vs manusia

Skor

Minimax	Manusia	Pemenang
6	3	Minimax
6	3	Minimax
7	2	Minimax
5	4	Minimax
4	5	Manusia

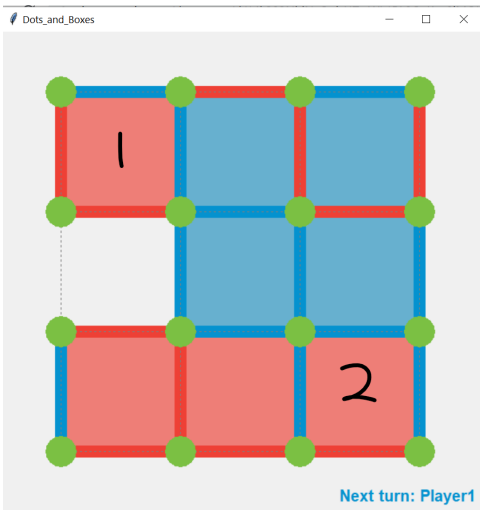
Analisis:



Perhatikan bahwa langkah nomor 1 adalah langkah yang manusia lakukan terlebih dahulu untuk membuat 3 garis sehingga memancing bot untuk membuat kotak. Strateginya adalah agar membuat bot "membuka jalan" untuk manusia membuat 6 kotak di samping kiri.

Namun, bot cukup pintar untuk melakukan langkah nomor 2 daripada menutup 3 kotak yang dibuat manusia. Hal ini berarti bot sudah memprediksi lebih jauh state-state di masa depan sehingga dia menghindarinya.

Gambar 4.1 Minimax 1



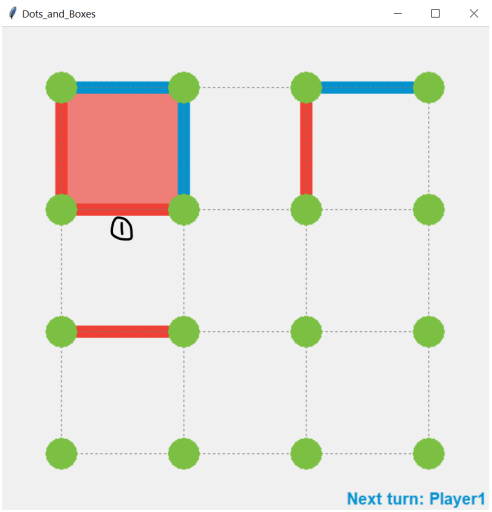
Pada permainan ini, bot sudah bermain dengan baik karena jika dilihat, box 1 dan 2 memiliki banyak garis biru (bot membuat kotak ketika manusia melakukan kesalahan), tetapi karena dalam fungsi objektif bot tidak menghitung chaining, kemungkinan bot kalah masih ada.

Gambar 4.2 Minimax 2

4.2 *Local search* vs manusia

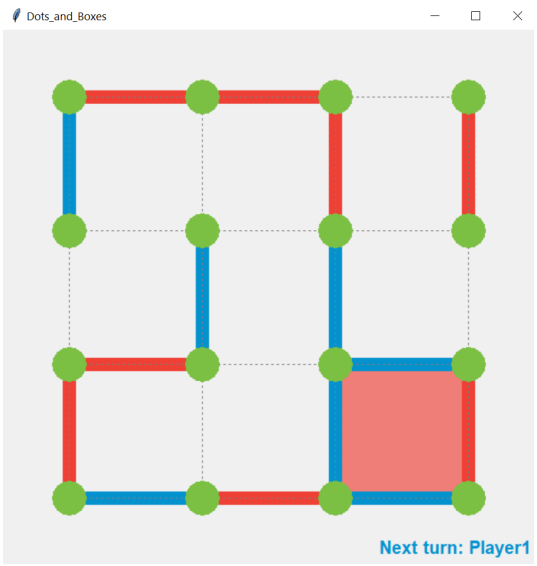
Skor

Local Search (n anak = 8)	Manusia	Pemenang
4	5	Manusia
1	8	Manusia
2	7	Manusia
4	5	Manusia
9	0	Local search



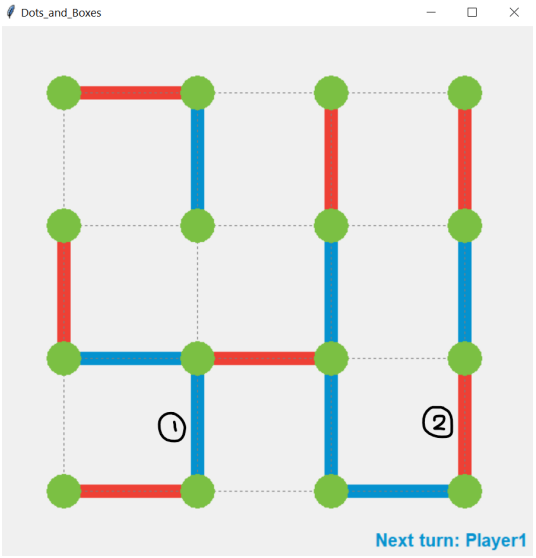
Pada gambar di samping, bot memilih langkah nomor 1. Hal ini bagus karena walaupun bot bersifat lokal, ketika ada kesempatan, bot tetap mengambil untuk membuat kotaknya.

Gambar 4.3 Local search 1



Gambar ini mirip dengan Gambar 4.1, tetapi berbeda dengan bot Minimax, bot Local Search mengambil kesempatan untuk membuat kotak. Akibatnya, bot membuka jalan bagi manusia untuk "menyerang" dan membuat kotak yang berantai.

Gambar 4.4 Local search 2



Perhatikan bahwa manusia mencoba untuk memberikan kesempatan kepada bot dengan melakukan langkah nomor 1, tetapi bot tidak mengambil kesempatan itu dan mengambil langkah nomor 2. Gerakan ini tidak optimal.

Gambar 4.5 Local search 3

4.3 Minimax vs *local search*

Skor

Minimax	*Local Search* (n)	Pemenang
5	4 (n = 1)	Minimax
5	4 (n = 3)	Minimax
5	4 (n = 8)	Minimax
5	4 (n = 16)	Minimax
5	4 (n = 24)	Minimax

Pada pertandingan Minimax vs *local search*, kemenangan didominasi oleh Minimax karena *local search* tidak seoptimal Minimax. Kemudian, perhatikan bahwa skornya konstan. Hal ini terjadi karena pola permainan bot cukup kaku (tidak ada random, hanya berdasarkan fungsi objektif) sehingga gerakan-gerakannya tidak terlalu bervariasi.

5. Saran perbaikan terhadap kode program yang telah dibuat

Pada pertandingan manusia melawan bot Minimax ataupun *local search,* beberapa kasus terjadi bot memilih langkah yang kurang optimal karena dapat di-*comeback* oleh manusia di akhir - akhir pertandingan. Hal tersebut dikarenakan kedalaman pencarian pada bot yang cukup rendah. Akan tetapi, apabila kedalaman ditambah, waktu pencarian program akan menjadi cukup lama sehingga tidak *feasible* untuk dilakukan. Kompleksitas algoritma dalam membangkitkan seluruh anak pada pohon pencarian adalah $O(b^d)$ dengan b adalah branching factor dan d adalah kedalaman sehingga meningkatkan kedalaman dapat memberikan dampak yang cukup signifikan.

Untuk mengatasi permasalahan tersebut, kita dapat memanfaatkan library external yang sudah tersedia demi mengoptimasi waktu pengecekan. Sebagai contoh, kita dapat memanfaatkan fungsi argwhere pada numpy untuk langsung mendapatkan banyaknya grid lengkap di state, atau bisa langsung menggunakan numpy untuk menghitung banyaknya *occurence* dari sebuah grid yang memiliki sisi 3. Dengan waktu pencarian yang lebih optimal, penambahan kedalaman bisa menjadi *feasible*.

Selain itu, kami membuat class yang berbeda untuk tiap algoritma pencarian. Apabila diperhatikan, minimax dan *local search* menggunakan *beam search* memiliki algoritma pencarian yang cukup mirip, hanya berbeda dari banyak anak yang dibangkitkan. Dengan demikian, seharusnya kedua algoritma dapat diimplementasikan dalam class yang sama, hanya ditambahkan sedikit parameter untuk memilih apakah banyak anak yang dibangkitkan perlu dibatasi atau tidak.

Terakhir, kami menggunakan fungsi objektif berupa selisih banyak grid yang sudah diperoleh player 2 dengan player 1. Fungsi objektif tersebut cukup optimal dengan minimax apabila kita melakukan pencarian hingga paling dalam (terminal state). Untuk kedalaman yang dibatasi, fungsi objektif tersebut dapat melakukan kesalahan sehingga dapat di-*comeback* manusia seperti yang sudah disebutkan sebelumnya. Pada kode program dapat digunakan alternatif fungsi objektif yakni dengan menghitung banyak chain pada state yang sedang diperiksa. Nilai yang dikembalikan dari fungsi objektif itu cukup unik karena memeriksa apakah banyak chain genap atau ganjil saja. Dengan demikian, untuk kedalaman yang cukup terbatas maka memungkinkan bagi bot untuk memberikan luaran *game action* yang lebih baik.

6. Kontribusi setiap anggota dalam kelompok

NIM - Nama	Kontribusi
13520057 - Marcellus Michael H.K.	Local search
13520087 - Dimas Shidqi Parikesit	Minimax
13520094 - Saul Sayers	Minimax, objective function
13520115 - Maria Khelli	Supporting modules (node, load config)