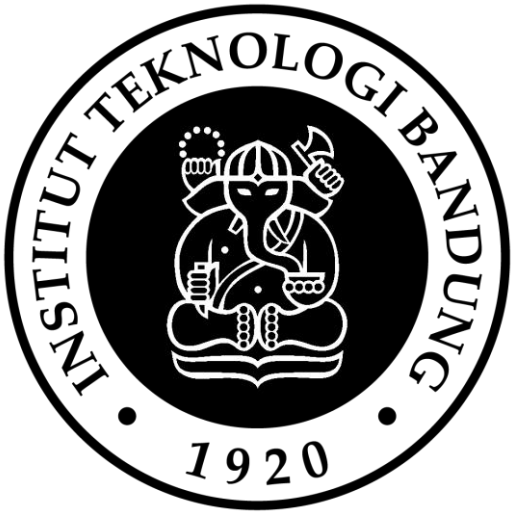
Tugas Besar 1 IF3170 Intelegensi Buatan

Implementasi *Minimax Algorithm* dan *Local Search* pada Permainan *Dots and Boxes*



Disusun oleh:

Ahmad Alfani Handoyo	13520023
Aditya Prawira Nugroho	13520049
Felicia Sutandijo	13520050
Putri Nurhaliza	13520066

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

PROGRAM STUDI TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

A. Objective Function

Objective function dibuat sebisa mungkin untuk merepresentasikan keadaan sebenarnya dari permainan, dengan melihat potensi menang/kalah suatu *player*. Di sini, *objective function* kami melihat seberapa banyak garis yang mengelilingi setiap kotak.

Variabel:

- L : Jumlah kotak (*box*) pada *board* permainan. Pada kasus ini L = 9.
- B_i : Kotak (*box*) 1x1 ke-i pada *board* permainan
- $v(B_i)$: Nilai (*value*) dari kotak 1x1 ke-i pada *board* permainan
- t : Giliran (*turn*) pada *state* tersebut. Bila giliran kita bernilai -1, bila giliran lawan bernilai 1

Nilai setiap kotak ($v(B_i)$):

1. Kotak dengan 0 garis : 0
2. Kotak dengan 1 garis : 5
3. Kotak dengan 2 garis : 10
4. Kotak dengan 3 garis : $25 \cdot t$
5. Kotak yang berhasil dibuat oleh kita : 100
6. Kotak yang berhasil dibuat oleh musuh : -100

Objective function:

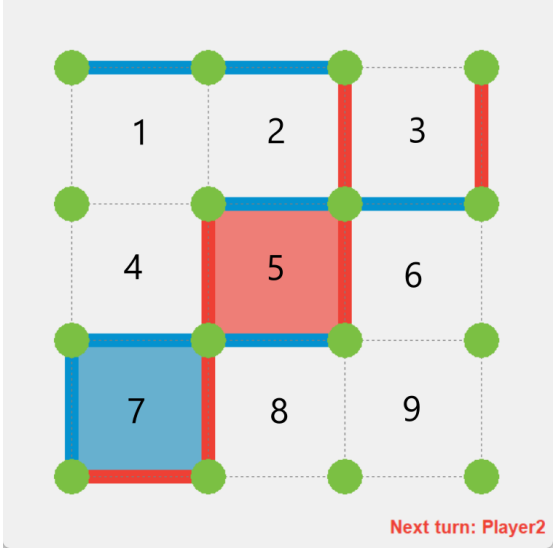
$$\sum_{i=1}^{L} v(B_i)$$

Objective function dihitung ketika membangkitkan suatu *state* setelah salah satu pemain meletakkan satu garis. Digunakan heuristik untuk menghitung *objective function* dengan melihat keuntungan berdasarkan garis pada setiap kotak. Pada kasus kotak dengan 3 garis, diberikan parameter tambahan yaitu gilian saat ini. Jika giliran saat ini adalah giliran kita, maka kotak dengan 3 garis akan memberikan keuntungan pada lawan di giliran setelahnya. Sebaliknya, jika saat ini adalah giliran lawan, maka kita memiliki kesempatan besar untuk memenuhi kotak pada giliran setelahnya.

Nilai untuk kotak dengan jumlah garis yang lain ditentukan menimbang bahwa ketika suatu kotak berisi 0 garis, masih tidak memberi keuntungan bagi kedua pemain. Ketika kotak berisi 1 garis tentunya lebih menguntungkan dibanding 0 garis karena artinya hanya perlu 3 garis lagi untuk memenuhi kotak, dan selanjutnya juga untuk kotak dengan 2 garis.

Penalti atau pengaruh terbesar secara logis harusnya ketika kotak berhasil dibuat oleh kita atau musuh. Tentunya saat kotak dibuat oleh kita berdampak positif karena membawa kita lebih dekat ke kemenangan sehingga direpresentasikan dengan nilai positif yang besar. Namun, bila kotak dibuat oleh musuh tentu berdampak negatif karena membawa kita semakin dekat ke kekalahan dan direpresentasikan dengan nilai-nilai negatif yang besar.

Contoh perhitungan *objective function* terdapat pada konfigurasi di bawah ini.



Gambar 1. Contoh Konfigurasi

Misal baru saja adalah giliran kita dan kita meletakkan satu garis sehingga *t* bernilai -1.

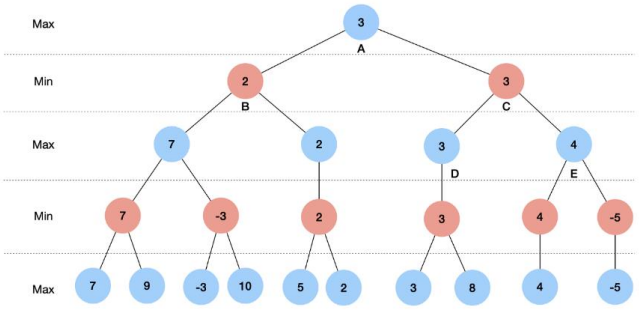
- Kotak 1 bernilai 5 karena diisi oleh 1 garis. ($v(B_1) = 5$)
- Kotak 2 bernilai -20 karena diisi oleh 3 garis. ($v(B_2) = 25 \cdot t = 25 \cdot (-1) = -25$)
- Kotak 3 bernilai -20 karena diisi oleh 3 garis. ($v(B_3) = 25 * t = 25 \cdot (-1) = -25$)
- Kotak 4 bernilai 10 karena diisi oleh 2 garis. ($v(B_4) = 10$)
- Kotak 5 bernilai -100 karena berhasil dibuat oleh musuh. ($v(B_5) = -100$)
- Kotak 6 bernilai 10 karena diisi oleh 2 garis. ($v(B_6) = 10$)
- Kotak 7 bernilai 100 karena berhasil dibuat oleh kita. ($v(B_7) = 100$)
- Kotak 8 bernilai 10 karena diisi oleh 2 garis. ($v(B_8) = 10$)
- Kotak 9 bernilai 0 karena belum diisi oleh garis. ($v(B_9) = 0$)

Sehingga total skor untuk *objective function*:

$$\sum_{i=1}^{L} v(B_i) = 5 + (-25) + (-25) + 10 + (-100) + 10 + 100 + 10 + 0 = -15$$

B. Minimax dengan Alpha-Beta Pruning

Algoritma Minimax merupakan salah satu algoritma yang dapat digunakan dalam pembangunan *bot* pemain *Dots and Boxes*. Algoritma Minimax adalah salah satu algoritma yang dikembangkan untuk menyelesaikan permasalahan permainan *multiplayer*. Algoritma ini bekerja dengan cara *cross-recursive*, dengan salah satu pemain berperan sebagai pemain yang memaksimalkan, dan pemain yang lainnya berperan sebagai pemain yang meminimalkan. Pemain yang memaksimalkan akan berusaha untuk mendapatkan skor *utility* yang terbesar, sedangkan pemain yang meminimalkan akan berusaha untuk mendapatkan skor *utility* yang terkecil. Pemain akan bermain secara bergantian, sehingga fungsi yang memaksimalkan akan memanggil fungsi yang meminimalkan, dan begitu pula sebaliknya. Skor *utility* ini dihitung untuk setiap *game state* menggunakan fungsi objektif (*objective function*) yang telah dijelaskan pada bab sebelumnya.



Gambar 2. Ilustrasi Algoritma Minimax

Bot Minimax pada tugas besar ini dirancang menggunakan beberapa fungsi utama, yaitu fungsi get_action, fungsi minimax, fungsi min_value, fungsi max_value, dan fungsi get_utility. Selain fungsi-fungsi utama tersebut, terdapat pula beberapa fungsi pembantu, yaitu fungsi is_terminal, fungsi update_board, dan fungsi get_actions.

Fungsi get_action merupakan fungsi bawaan dari kelas bot yang akan mengembalikan aksi yang dipilih oleh *bot* untuk dimainkan. Fungsi ini memanggil fungsi minimax yang menghitung dan menganalisis aksi-aksi yang mungkin dan memilih yang paling baik.

Fungsi minimax sendiri terdiri atas tiga bagian, yaitu bila *state* sudah terminal (kotak sudah penuh), bila sedang giliran pemain yang memaksimalkan, dan bila sedang giliran pemain yang meminimalkan. Pada penjelasan ini, diasumsikan player1 merupakan pemain yang meminimalkan, dan player2 merupakan pemain yang memaksimalkan.

Proses perhitungan value minimax adalah sebagai berikut:

1. Bila *state* merupakan *state* terminal (diperiksa menggunakan fungsi is_terminal yang mengembalikan *boolean true* atau *false*), fungsi minimax mengembalikan skor *utility state* tersebut.
2. Bila sedang giliran player1, fungsi minimax mengembalikan nilai minimum dari *value* minimax seluruh *state* yang mungkin dihasilkan melalui kemungkinan-kemungkinan aksi saat itu. Pada setiap iterasi, nilai dari *alpha* diperbarui dengan membandingkan nilai *alpha* sebelumnya dan *value* yang didapat pada *state* saat itu, serta mengambil nilai yang lebih besar. Bila *alpha* sudah lebih dari beta, iterasi tidak perlu dilanjutkan (*state-state* yang tidak mungkin dipilih tidak perlu dibangkitkan).
3. Bila sedang giliran player2, fungsi minimax mengembalikan nilai maksimum dari *value minimax* seluruh *state* yang mungkin dihasilkan melalui kemungkinan-kemungkinan aksi saat itu. Pada setiap iterasi, nilai dari beta diperbarui dengan membandingkan nilai beta sebelumnya dan *value* yang didapat pada *state* saat itu, serta mengambil nilai yang lebih kecil. Bila *alpha* sudah lebih dari beta, iterasi tidak perlu dilanjutkan (*state-state* yang tidak mungkin dipilih tidak perlu dibangkitkan).

Pada tugas besar ini, pohon yang dibangkitkan hanya sampai kedalaman 3 agar *bot* dapat melakukan komputasi yang cepat dikarenakan *branching factor* yang sangat besar, terutama di awal permainan.

C. Local Search

Local search untuk penyelesaian permainan *Dots and Boxes* dapat dilakukan dengan metode *hill-climbing search* dengan pendekatan *greedy* yang memilih gerakan untuk menghasilkan *state* baru dengan *objective function*-nya yang paling tinggi. Ini berbeda dengan variasi-variasi *hill-climbing* yang diajarkan di kelas dengan pemilihan *state* baru selalu yang lebih baik (atau bernilai sama pada variasi *sideways move*) dari *state* saat ini.

Alasan pendekatan *greedy* dipilih agar bot selalu melakukan aksi terbaik yang mungkin namun tetap memastikan bot berjalan apa bila tidak ada *state* selanjutnya yang lebih baik dari *state* saat ini. Tidak memungkinkan untuk melakukan terminasi *state* apa bila tidak ada pilihan yang lebih baik, karena bot harus memilih suatu aksi pada setiap gilirannya.

Proses penentuan langkah optimal menggunakan algoritma *greedy hill-climbing search* adalah sebagai berikut:

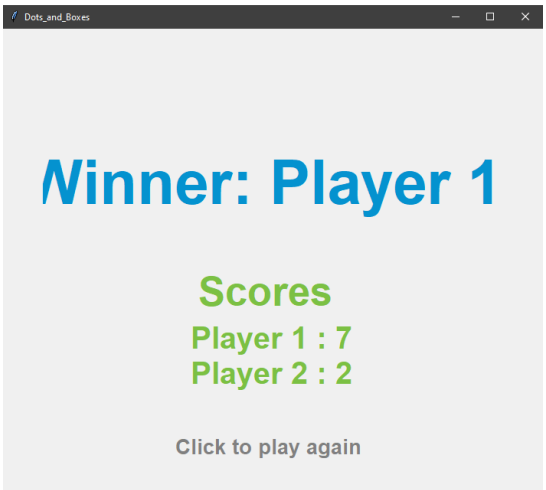
1. *Current state* adalah kondisi papan permainan saat pemain akan meletakkan garis.
2. Pada pemilihan garis pertama oleh pemain (saat papan permainan masih kosong), maka garis diletakkan secara *random* di bagian dalam papan permainan (garis yang melibatkan 2 kotak agar total *objective value*-nya lebih tinggi).
3. Pada langkah selanjutnya, semua *neighbor state* yang mungkin dibangkitkan. Pada program, tahap ini diimplementasikan dengan fungsi *get_all_possible_actions()*.
4. Pilih *neighbor state* dengan *objective value* tertinggi yang dihitung dengan fungsi *get_utility()* berdasarkan *objective function* di atas. Pada program, pemilihan *neighbor* terbaik ini diimplementasikan dengan fungsi *getNeighbor()*.
5. *Return action* untuk meletakkan garis tersebut.

D. Hasil Pertandingan

1. Bot Minimax vs Manusia

Dalam permainan kali ini, bot minimax adalah *player* 2 dan manusia adalah *player* 1. Berikut adalah hasil pertandingan dan jumlah skor masing-masing pemain.

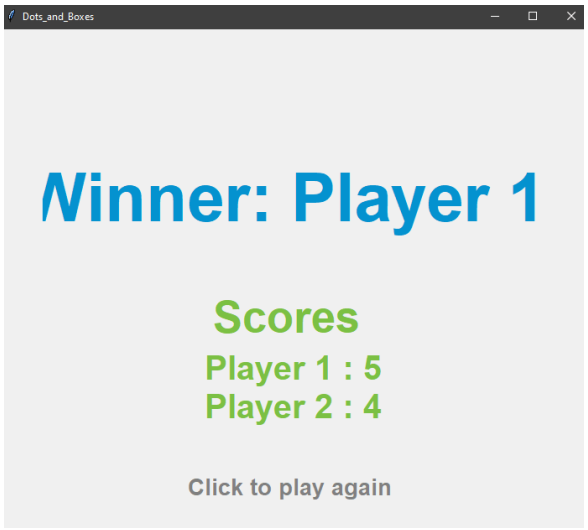
Gambar 1 Hasil pertandingan 1 bot minimax vs manusia



Gambar 2 Hasil pertandingan 2 bot minimax vs manusia

Gambar 3 Hasil pertandingan 3 bot minimax vs manusia

Gambar 4 Hasil pertandingan 4 bot minimax vs manusia

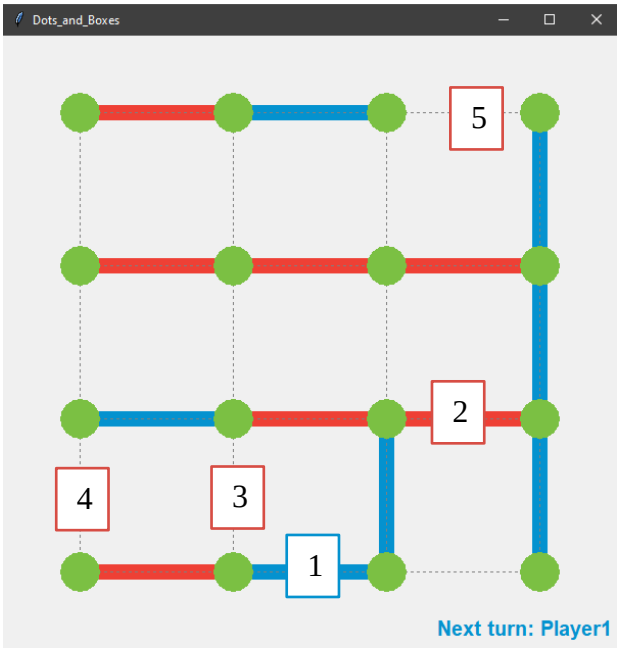


Gambar 5 Hasil pertandingan 5 bot minimax vs manusia

Statistik pertandingan:

- Jumlah bot minimax menang: 0
- Jumlah manusia menang: 5

Dari 5 permainan, terlihat bahwa manusia menang sebanyak 5 kali dan bot minimax tidak menang sama sekali. Dari beberapa permainan tersebut, bot sering melakukan kesalahan yang hampir mirip. Salah satu contohnya adalah pada gambar 6, *player* 1 meletakkan garis pada nomor 1. Kemudian, bot minimax meletakkan garis pada nomor 2. Seharusnya, bot meletakkan garis pada nomor 3 untuk mendapatkan satu *box* dan nomor 4 untuk mendapatkan tambahan satu box lagi. Setelah itu, bot menggunakan giliran tambahan untuk meletakkan garis pada nomor 5 sehingga *player* 1 hanya mendapatkan 3 *box* dan terpaksa untuk memberikan sisa empat *box* kepada bot.



Gambar 6 Ilustrasi permainan

Kesalahan ini dapat disebabkan oleh pendefinisian fungsi objektif yang mengurangi poin ketika pemain meletakkan satu sisi sehingga *box* memiliki 3 sisi. Selain itu, bot minimax kali ini hanya dapat membangkitkan simpul hingga kedalaman 3. Sehingga, bot tidak dapat mendapatkan solusi yang paling optimal. Selain itu, bot menghitung minimal dengan fungsi objektif yang sama, padahal manusia bisa saja berpikir bahwa tidak apa-apa membuat 3 sisi pada sebuah *box* untuk mendapatkan tambahan *box* lain.

2. Bot *Local Search* vs Manusia

Dalam permainan kali ini, bot *local search* adalah *player* 2 dan manusia adalah *player* 1. Berikut adalah hasil pertandingan dan jumlah skor masing-masing pemain.

Gambar 1 Hasil pertandingan 1 bot minimax vs manusia

Gambar 2 Hasil pertandingan 2 bot minimax vs manusia

Gambar 3 Hasil pertandingan 3 bot minimax vs manusia

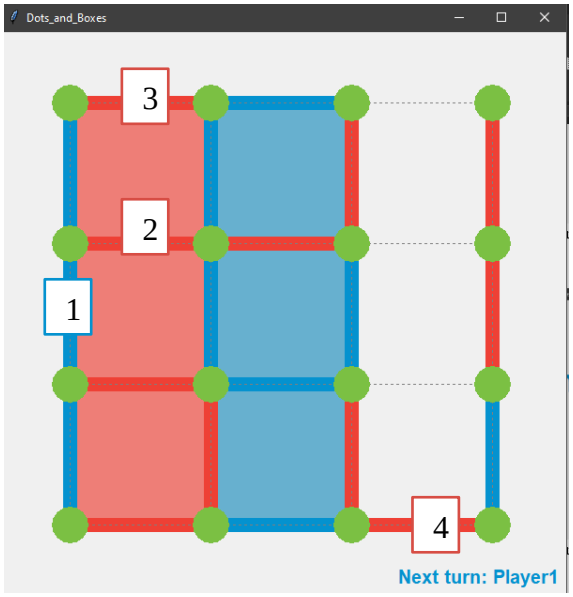
Gambar 4 Hasil pertandingan 4 bot minimax vs manusia

Gambar 5 Hasil pertandingan 5 bot minimax vs manusia

Statistik pertandingan:

- Jumlah bot *local search* menang: 2
- Jumlah manusia menang: 3

Dari 5 permainan, terlihat bahwa manusia menang sebanyak 3 kali dan bot *local search* menang 2 kali. Dari beberapa permainan tersebut, bot sering melakukan kesalahan yang hampir mirip. Salah satu contohnya adalah pada gambar 12, *player* 1 meletakkan garis pada nomor 1. Kemudian, bot minimax meletakkan garis pada nomor 2, seharusnya, bot meletakkan garis pada nomor 3 untuk menjebak *player* 1. Kesalahan pemilihan ini disebabkan oleh fungsi objektif yang mengurangi poin ketika garis yang ditambahkan membuat 3 sisi. Jika bot meletakkan pada sisi 3, maka bot tidak akan mendapatkan poin membuat box sehingga nilai objektif dari *state* tersebut lebih jelek dibandingkan meletakkan pada sisi 2. Namun, hal itu justru menyebabkan *player* 1 menang karena bot harus menggunakan giliran tambahan untuk mengisi 1 garis sehingga bot meletakkan garis pada nomor 4. Dengan begitu, *player* 1 akan mendapatkan sisa *box* sebanyak 3.

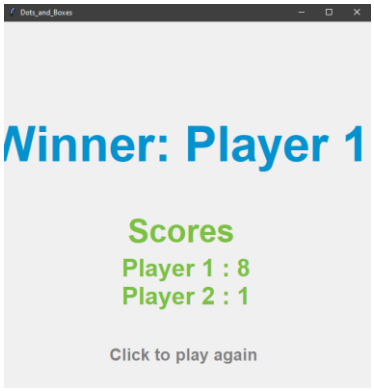


Gambar 12 Ilustrasi permainan

3. Bot *Local Search* vs Bot Minimax

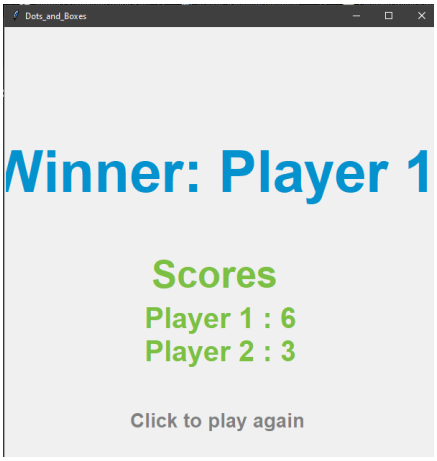
Dalam permainan kali ini, bot *local search* adalah *player* 1 dan bot minimax adalah *player* 2. Berikut adalah hasil pertandingan dan jumlah skor masing-masing pemain.

Gambar 13 Hasil pertandingan 1 bot *local search* vs bot minimax

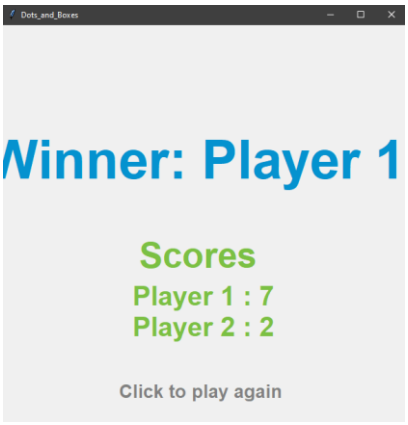


Gambar 14 Hasil pertandingan 2 bot *local search* vs bot minimax

Gambar 15 Hasil pertandingan 3 bot *local search* vs bot minimax



Gambar 16 Hasil pertandingan 4 bot *local search* vs bot minimax



Gambar 17 Hasil pertandingan 5 bot *local search* vs bot minimax

Statistik pertandingan:

- Jumlah bot *local search* menang: 5
- Jumlah bot minimax menang: 0

Dari statistik pertandingan, didapatkan bahwa bot *local search* menang 5 kali dari 5 pertandingan. Kemenangan bot *local search* dapat disebabkan oleh sifat algoritma yang lebih *straightforward*, yaitu memilih gerakan yang menyebabkan *state* terbaik. Selain itu, bot minimax yang kami gunakan hanya mencari hingga kedalaman 3 sehingga pengambilan keputusan kurang optimal. Selain itu, setelah bot minimax membuat keputusan gerakan, bot *local search* akan melakukan hitung ulang dari *state* sehingga prediksi dari bot minimax bisa saja salah.

E. Saran Perbaikan

Dalam menentukan pilihan apa yang terbaik untuk dijalankan oleh bot, dibutuhkan suatu *objective function* yang merepresentasikan *state* permainan sedekat-dekatnya dengan kondisi di dunia nyatanya. Artinya, suatu *state* permainan yang dianggap bagus seharusnya mempunyai nilai *objective function* yang tinggi. Begitu pula *state* permainan yang dianggap buruk seharusnya berkebalikan yaitu mempunyai nilai *objective function* yang buruk. *Objective function* yang baik penting bagi bot agar bot memilih *state* dengan *objective function* tertinggi yang seharusnya ekivalen dengan keadaan dunia nyatanya yang berarti pilihan yang paling baik.

Saat ini *objective function* yang diimplementasikan sudah memperhitungan pengaruh giliran dari pemain terutama untuk kotak yang sudah dikeliling oleh 3 garis. Perhitungan ini penting karena dengan meletakkan garis agar kotak bergaris 3 tidak diinginkan sehingga musuh dapat meletakkan garis ke-4 agar mendapatkan kotak. Namun, *objective function* ini masih dapat dioptimasi lagi agar lebih merepresentasikan keadaan dunia nyatanya. Kekurangan ini dapat dilihat khususnya pada bot minimax yang memperhitungkan hingga 3 langkah ke depan yang terkadang masih mengambil pilihan yang ketika dianalisis kurang optimal. Optimasi *objective function* ini dapat dilakukan dengan mengutak-atik *multiplier*

pada setiap kondisi atau juga memperhitungkan kondisi atau metode dan strategi permainan lainnya.

Bot minimax yang diimplementasikan saat ini masih menggunakan struktur data yang kurang efisien. Hal tersebut menyebabkan waktu yang dibutuhkan bot untuk menentukan keputusan dan melakukan komputasi bertambah. Beberapa perbaikan yang dapat dilakukan untuk mengoptimasi bot minimax adalah membuat sebuah kelas *node* yang menyimpan data yang diperlukan untuk algoritma minimax. Sehingga, pembangkitan simpul anak bisa lebih efisien.

Algoritma yang dipilih untuk bot *local search* saat ini adalah *greedy hill climbing*. Algoritma tersebut dapat diganti menjadi algoritma *local search* lain yang lebih efektif, yaitu *beam search*. Jika menggunakan algoritma *beam search*, bot dapat melakukan prediksi alur permainan hingga akhir sehingga dapat dikatakan keputusan yang dibuat oleh bot bisa lebih baik karena algoritma akan meninjak keputusan-keputusan terbaik berdasarkan nilai objektifnya dibandingkan dengan algoritma *greedy hill climbing* yang hanya mencari keputusan terbaik saat ini.

F. Kontribusi Anggota

Nama - NIM	**Kontribusi**
Ahmad Alfani Handoyo – 13520023	Pembuatan *objective function, debugging*, pembuatan laporan
Aditya Prawira Nugroho – 13520049	Pembuatan bot *local search*¸ pembuatan laporan
Felicia Sutandijo – 13520050	Pembuatan bot minimax¸ pembuatan laporan
Putri Nurhaliza – 13520066	Pembuatan bot *local search*¸ pembuatan laporan

REFERENSI

GeeksforGeeks. 2021. *Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning)*. Diakses pada 12 September 2022, dari https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/

Khodra, Masayu Leylia. 2021. “Beyond Classical Search: Classical Search vs Local Search”. Bandung: Institut Teknologi Bandung.

Russell, Stuart. Peter Norvig. 2010. *Artificial Intelligence a Modern Approach Third Edition*. New Jersey: Pearson Education, Inc.