TUGAS BESAR I

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

LAPORAN

Diajukan sebagai salah satu tugas mata kuliah IF3170 Intelegensi Buatan Semester I

Tahun Akademik 2022-2023

Oleh

Kelompok 27

Dzaky Fattan Rizqullah	13520003
Bariza Haqi	13520018
Rozan Fadhil Al Hafidz	13520039
Jeremy Rionaldo Pasaribu	13520082

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

BAB 1: Pembahasan

1.1. Penjelasan *Objective Function* yang Digunakan

1.1.1. *Objective Function* Algoritma *Minimax* dengan *Alpha Beta Pruning*

Berikut merupakan *objective function* dari algoritma *Minimax* dengan *Alpha Beta Pruning*:

$$U = P - E$$

dengan keterangan:

$U = fungsi\ obyektif$

$P = jumlah\ kotak\ pemain$

$E = jumlah\ kotak\ musuh$

Implementasi *objective function* tersebut sebagai berikut:

```
if depth == 0 or len(node.listMove) == 0 or end_time > 4.5:
            return [move, node.score[True] - node.score[False]]

# objective function = node.score[True] - node.score[False]
```

Dalam implementasi *objective function* ini, **`node.score[True]`** merupakan jumlah kotak 1x1 yang dimiliki oleh bot pemain dalam papan sedangkan **`node.score[False]`** merupakan jumlah kotak 1x1 yang dimiliki oleh bot musuh dalam papan. Rumus *objective function* ini digunakan untuk memudahkan bot pemain dalam mengambil langkah yang optimal terhadap *terminal state* pada batas kedalaman pohon (*depth*) yang telah ditentukan. Jika nilai *objective function* yang dihasilkan positif (bot pemain memiliki jumlah kotak 1x1 yang lebih banyak daripada musuh), maka pemain memiliki kemungkinan yang besar dalam memenangkan permainan. Jika nilai *objective function* yang dihasilkan negatif (bot musuh memiliki jumlah kotak 1x1 yang lebih banyak daripada pemain), maka musuh memiliki kemungkinan yang besar untuk memenangkan permainan.

1.1.2. *Objective Function* Algoritma *Local Search* (*Hill-Climbing Sideways Move*)

Algoritma *Local Search* memiliki *objective function* yang menghitung nilai berdasarkan banyaknya jumlah garis yang terdapat pada kotak 1x1 dalam papan permainan. Implementasi *objective function* tersebut sebagai berikut:

```
def utility_local_search(self) -> int:
     if (self.count_box() - self.current_box) > 0:
         return 1
     if (np.any(self.board_status == 3)):
         return -1
     return 0
```

Dalam implementasi *objective function* ini, **`self.current_box`** merupakan jumlah kotak 1x1 yang terbentuk pada *state* awal permainan, **`self.count_box()`** merupakan jumlah kotak 1x1 yang terbentuk pada *state* permainan setelah dilakukan aksi, **`np.any(self.board_status == 3)`** untuk mengecek apakah terdapat suatu kotak 1x1 yang memiliki jumlah garis tiga. *Objective function* ini akan menghasilkan tiga nilai yaitu -1, 0, atau 1. *Objective function* akan menghasilkan nilai -1 jika aksi penambahan garis oleh pemain mengakibatkan suatu kotak 1x1 memiliki jumlah 3 garis. Hal ini memberikan kesempatan musuh untuk mengambil kotak 1x1 tersebut dan musuh mendapatkan giliran kembali untuk menambahkan garis. *Objective function* akan menghasilkan nilai 0 jika aksi penambahan garis oleh pemain tidak membentuk kotak 1x1 yang baru (**`(self.count_box() - self.current_box) == 0`**). *Objective function* akan menghasilkan nilai 1 jika aksi penambahan garis oleh pemain membentuk kotak 1x1 yang baru (**`(self.count_box() - self.current_box) > 0`**). Hal ini dikarenakan pemain mendapatkan kotak 1x1 dan pemain akan mendapatkan giliran kembali.

1.2. Proses Pencarian dengan *Minimax* dan *Alpha Beta Pruning*

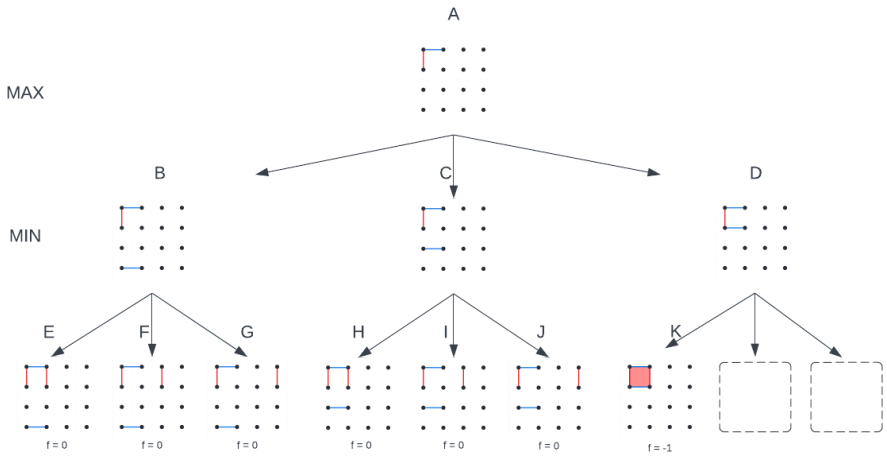
Pada awal permainan terdapat 24 kemungkinan pemasangan garis dan setiap langkah akan mengurangi 1 kemungkinan sehingga terdapat total 24! node yang dapat dilalui. Pada permainan *Dots and Boxes* ini langkah awal yang dilakukan bot adalah membuat node seluruh *state* mungkin terjadi pada beberapa step ke depan. Banyaknya *state node* yang dibuat bergantung pada waktu yang diberikan. Jika waktu cukup, akan

tercipta proses pencarian minimax untuk keseluruhan node. Namun jika waktu yang diberikan tidak cukup maka proses pencarian minimax akan dibatasi kedalamannya. Karena waktu pencarian algoritma maksimal 5 detik, batas kedalaman akan bergantung kepada jumlah node yang akan dibuat. Sebelum memulai pencarian, bot akan menghitung jumlah node yang akan dibuat terlebih dahulu beserta menghitung kedalamannya. Perhitungan akan berhenti jika jumlah node melebihi 4.000.000 dan bot akan mengambil kedalaman terbesar dengan jumlah node yang kurang dari 4.000.000. Bot juga memperhatikan batas waktu. jika waktu pencarian sudah lebih dari 4.5 detik, bot akan menghentikan pencarian dan mengambil *state* terbaik yang didapat saat itu.

Setelah membuat node, bot juga menghitung nilai dari *terminal node* menggunakan *objective function*. *Terminal node* adalah node paling bawah yang berhasil di-*generate* berdasarkan batasan jumlah node yang telah ditentukan. Dilakukan juga pruning untuk mengurangi jumlah node yang tidak perlu dengan metode *alpha-beta pruning*. Alpha (**`bestScore`**) adalah nilai terbaik yang memaksimalkan pada tingkat tersebut atau di atasnya. Beta (**`worstScore`**) adalah nilai yang meminimalkan pada tingkat tersebut atau di atasnya. Selain itu, dilakukan juga pemberian nilai pada setiap node. Pemberian nilai dilakukan dengan menggunakan metode minimax berdasarkan *children*-nya.

Dimulai dari bawah, akan digunakan nilai alpha beta dari *children*-nya untuk menilai sebuah node, dengan maksimum/minimum tergantung giliran pemain. Jika *state* tersebut merupakan giliran bot ini (**`playerTurn == True`**), maka akan dicari nilai maksimumnya. Jika *state* tersebut merupakan giliran lawan (**`playerTurn == False`**), maka akan diambil nilai minimumnya dengan asumsi lawan akan mengambil langkah terburuk bagi bot ini. Setelah mengetahui seluruh nilai minimax-nya, bot akan memilih langkah terbaik, yaitu node yang memiliki nilai paling besar. Proses tersebut dilakukan hingga permainan berakhir. Jika bot ini membuat kotak 1x1 pada papan permainan, maka bot ini akan mendatpkan giliran kembali. Jika bot musuh membuat kotak 1x1 pada papan permainan, maka bot musuh akan mendapatkan giliran kembali.

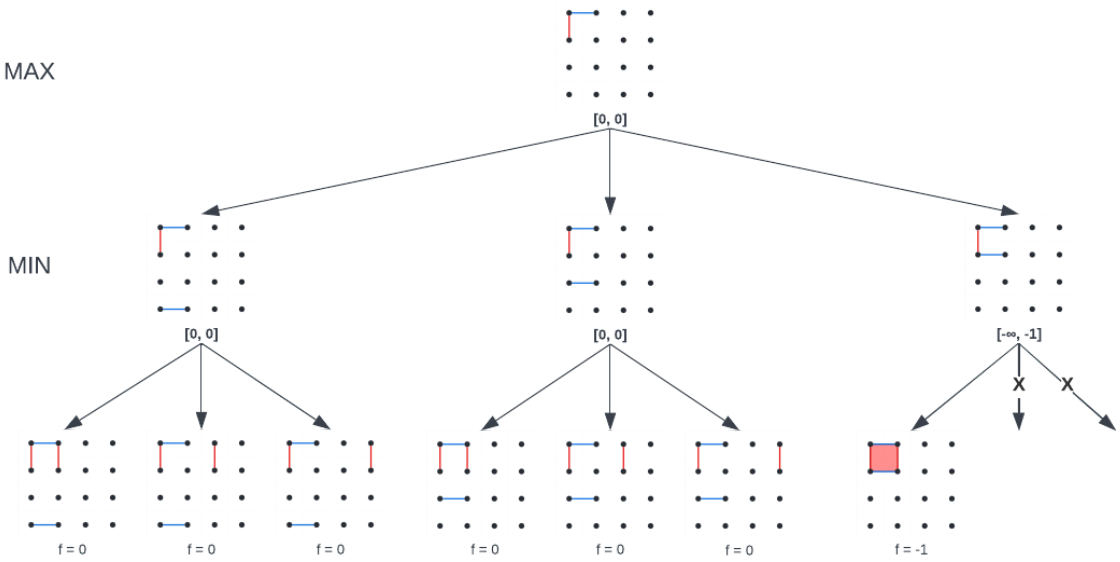
Berikut ini adalah contoh penerapan algoritma pada bot. Untuk mempersingkat penjelasan, seluruh *state* tidak digambar.



Sebelum melakukan pencarian akan ditentukan nilai alpha dan beta terlebih dahulu yaitu alpha adalah -INFINITY dan beta adalah +INFINITY. Pencarian dimulai dari A. Pada A akan dicari nilai maksimum dari B, C dan D, maka A akan memanggil B terlebih dahulu. Pada B, akan dicari nilai minimum dengan dimulai dari node paling kiri. Node E mengembalikan nilai 0, maka didapat beta-nya yaitu min (+INFINITY, 0) adalah 0. Cek apakah alpha>=beta untuk menentukan lanjutnya pencarian. Karena -INFINITY>=0 salah maka pencarian dilanjutkan ke node selanjutnya. Node F mengembalikan nilai 0 maka beta= min(0,0) yaitu 0. Karena -INFINITY>=0 salah maka pencarian node dilanjutkan. Untuk kasus node G juga akan sama yaitu beta=0 maka B mengembalikan nilai 0 ke A.

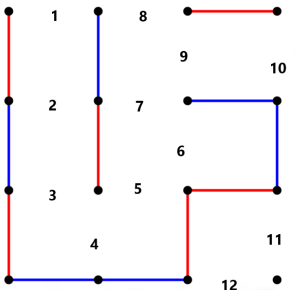
Pada node A, nilai alpha berubah yaitu max(-INFINITY, 0) = 0. A akan memanggil C dan mengembalikan nilai 0 juga maka nilai alpha tidak berubah. Pencarian akan dilanjut dengan A memanggil D. Pada D, alpha=0 dan beta=+INFINITY. D akan memanggil K dan mengembalikan nilai -1 maka beta= min(+INFINITY, -1) yaitu -1. Kemudian dicek apakah alpha>=beta. 0>=-1 bernilai benar maka pencarian pun dihentikan. Berapapun nilai node di kanannya tidak akan berpengaruh karena nilai maksimum sudah dijamin 0 atau lebih sehingga saat terdapat nilai -1, nilai maksimum tidak akan didapat lebih dari itu pada bagian node tersebut. D akan mengembalikan nilai -1 maka nilai optimal untuk A adalah max(0,-1) yaitu 0.

Pada program ini, nilai -INFINITY diganti menjadi -10 dan +INFINITY diganti menjadi 10 untuk mempermudah perhitungan dalam *python*. Berikut ini merupakan contoh hasil dari perhitungan tersebut.

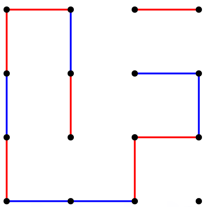


1.3. Proses Pencarian dengan *Local Search* (*Hill-Climbing Sideways Move*)

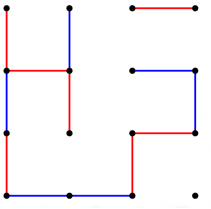
Pada pencarian menggunakan algoritma *Local Search* akan ditentukan berdasarkan value yaitu -1 untuk *state* yang dapat menghasilkan kotak untuk lawan, 0 untuk tidak ada kotak yang terbentuk dan +1 untuk bot ini yang membuat kotak. Pada saat menggunakan algoritma *Local Search*, pencarian akan berhenti saat didapat value yang bernilai +1. Algoritma *Local Search* yang digunakan adalah *Hill-Climbing Sideways Move***.** Algoritma *Local Search* tidak diberikan fungsi batasan waktu karena proses pencariannya selalu di bawah 5 detik. Berikut ini adalah contoh *state* dan proses pencarian menggunakan local search. Nomor menunjukkan posisi garis baru yang akan dibuat oleh bot ini.



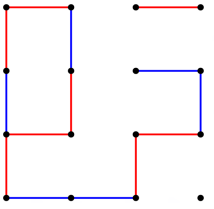
Pada gambar di atas terdapat 12 *state* pemasangan garis yaitu :

1. 

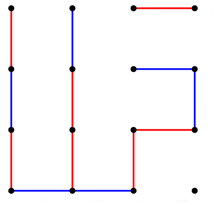
Pada *state* ini bernilai -1 karena lawan dapat membuat kotak dengan membuat garis pada nomor 2 dan 6

2. 

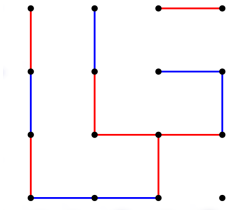
Pada *state* ini bernilai -1 karena lawan dapat membuat kotak dengan membuat garis pada nomor 1, 3 dan 6

3. 

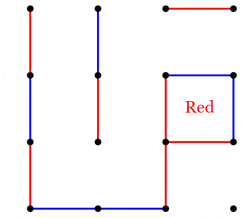
Pada *state* ini bernilai -1 karena lawan dapat membuat kotak dengan membuat garis pada nomor 2, 4 dan 6

4. 

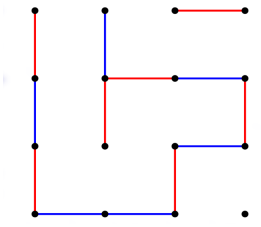
Pada *state* ini bernilai -1 karena lawan dapat membuat kotak dengan membuat garis pada nomor 3, 5 dan 6

5. 

Pada *state* ini bernilai -1 karena lawan dapat membuat kotak dengan membuat garis pada nomor 4 dan 6

6. 

Pada step ini bernilai +1 karena bot ini berhasil membuat kotak

7. 

Pada *state* ini bernilai -1 karena kotak lawan dapat membuat kotak dengan membuat garis pada nomor 6. Pada *state* ini, pencarian berhenti karena *state* ini bernilai lebih kecil dibandingkan dengan *state* sebelumnya.

Pada pencarian di atas, nomor 6 akan menjadi step yang dipilih untuk langkah selanjutnya. Jika bot tidak menemukan value yang lebih kecil, maka bot akan memilih step pada *state* terakhir. Algoritma *Hill-Climbing Sideways Move* dipilih dibandingkan

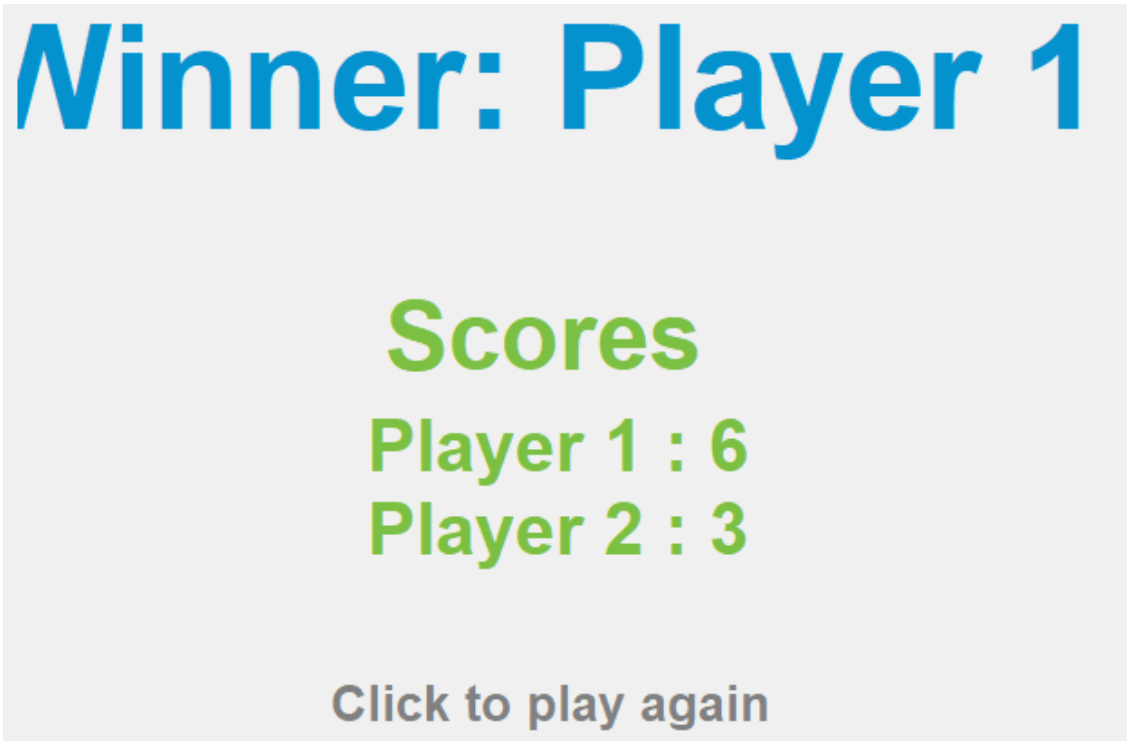
dengan algoritma-algoritma lain karena algoritma ini mengambil *state* yang terbaik pada saat itu meskipun *state* tersebut merupakan lokal maksimum dibandingkan dengan algoritma lain. Algoritma ini juga dipilih karena semua tetangga node bisa mendapatkan giliran dibandingkan dengan algoritma lain yang memilih tetangga node secara *random* seperti *Stochastic Hill-Climbing* yang kemungkinan ada beberapa tetangga node yang belum dilakukan pengecekan.

1.4. Hasil Pertandingan

Berikut terlampir hasil pertandingan antara bot *minimax* melawan manusia, bot *local search* melawan manusia, dan bot *minimax* melawan bot *local search*.

1.4.1. Bot *minimax* vs Manusia

Bot *minimax* (Player 1) vs Manusia (Player 2)



Winner: Player 1

Scores

Player 1 : 7
Player 2 : 2

Click to play again

Winner: Player 1

Scores

Player 1 : 7
Player 2 : 2

Click to play again

Winner: Player 2

Scores

Player 1 : 3
Player 2 : 6

Click to play again

Winner: Player 1

Scores

Player 1 : 6
Player 2 : 3

Click to play again

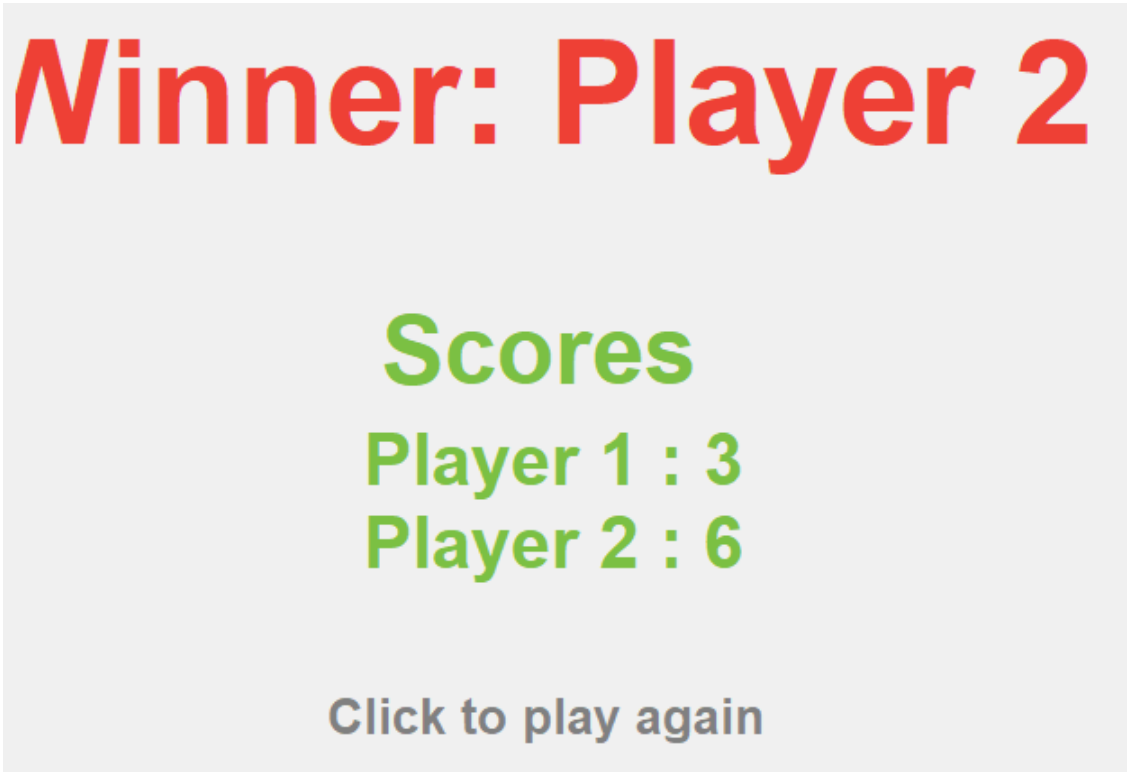
Berikut merupakan jumlah skor statistik kemenangan pertandingan:

	Jumlah Kemenangan	**Jumlah Kekalahan**	**Persentase Kemenangan**
Bot *minimax*	4	1	80%
Manusia (Penulis)	1	4	20%

Berdasarkan data pertandingan bot *minimax* vs penulis yang dilakukan 5 kali, terlihat bahwa bot *minimax* memiliki persentase kemenangan yang lebih tinggi daripada penulis sehingga bot *minimax* memiliki performa yang lebih baik ketika melawan penulis.

1.4.2. Bot *local search* vs Manusia

Bot *minimax* (Player 1) vs Manusia (Player 2)



Winner: Player 2

Scores

Player 1 : 3
Player 2 : 6

Click to play again

Winner: Player 1

Scores

Player 1 : 5
Player 2 : 4

Click to play again

Winner: Player 2

Scores

Player 1 : 2
Player 2 : 7

Click to play again

Winner: Player 2

Scores

Player 1 : 3
Player 2 : 6

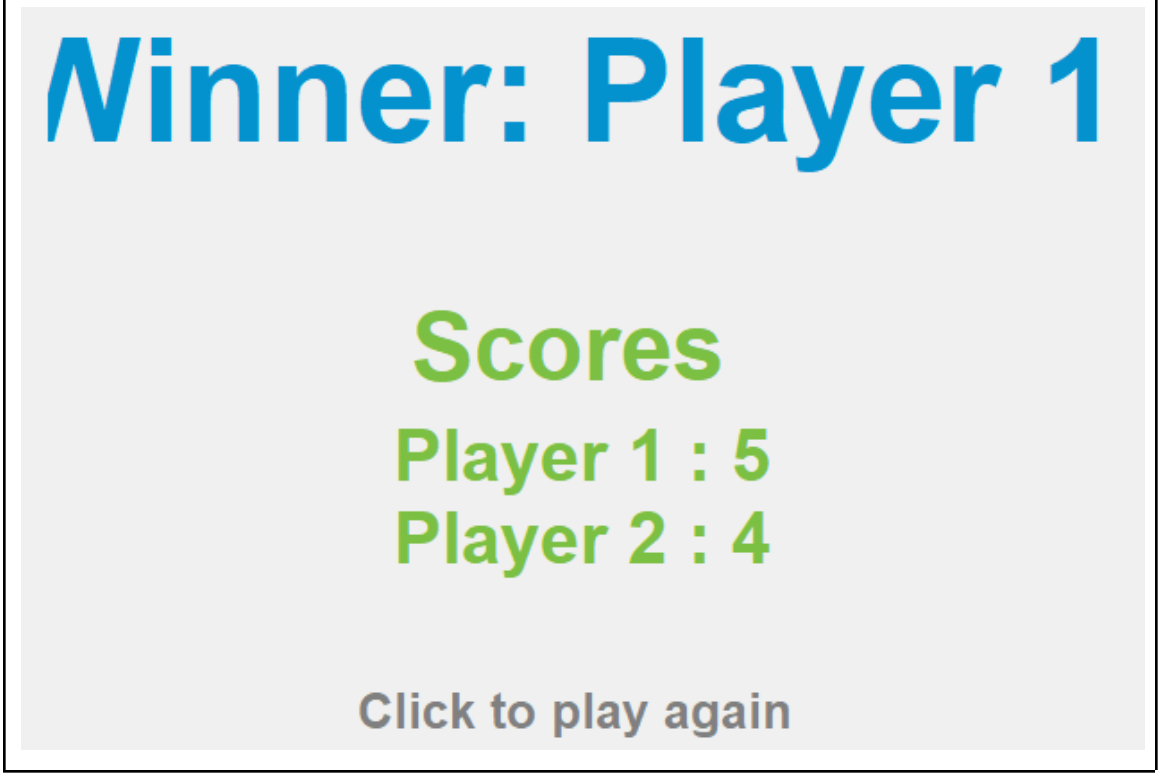
Click to play again

Berikut merupakan jumlah skor statistik kemenangan pertandingan:

	Jumlah Kemenangan	**Jumlah Kekalahan**	**Persentase Kemenangan**
Bot *local search*	2	3	40%
Manusia (Penulis)	3	2	60%

Berdasarkan data pertandingan bot *local search* vs manusia yang dilakukan 5 kali, terlihat bahwa penulis memiliki persentase kemenangan yang lebih tinggi daripada bot *local search* sehingga bot *local search* memiliki performa yang sedikit buruk ketika melawan penulis.

1.4.3. Bot *minimax* vs Bot *local search*

Bot *minimax* (Player 1) vs Bot *local search* (Player 2)


Winner: Player 1

Scores

Player 1 : 8
Player 2 : 1

Click to play again

Winner: Player 2

Scores

Player 1 : 2
Player 2 : 7

Click to play again

Winner: Player 1

Scores

Player 1 : 5
Player 2 : 4

Click to play again

Winner: Player 1

Scores

Player 1 : 6
Player 2 : 3

Click to play again

Berikut merupakan jumlah skor statistik kemenangan pertandingan:

	Jumlah Kemenangan	**Jumlah Kekalahan**	**Persentase Kemenangan**
Bot *minimax*	4	1	80%
Bot *local search*	1	4	20%

Berdasarkan data pertandingan bot *minimax* vs bot *local search* yang dilakukan 5 kali, terlihat bahwa bot *minimax* memiliki persentase kemenangan yang lebih tinggi daripada bot *local search* sehingga bot *minimax* memiliki performa yang lebih baik dibandingkan dengan bot *local search*.

1.4.4. Hasil Pertandingan Keseluruhan

Berikut merupakan jumlah skor statistik kemenangan pertandingan keseluruhan untuk bot *minimax* dan bot *local search*:

	Total Jumlah Kemenangan	**Total Jumlah Kekalahan**	**Persentase Total Kemenangan**
Bot *minimax*	8	2	80%
Bot *local search*	3	7	30%

Berdasarkan data pertandingan bot *minimax* vs bot *local search* yang dilakukan 5 kali, terlihat bahwa bot *minimax* memiliki persentase kemenangan yang lebih tinggi daripada bot *local search* sehingga bot *minimax* memiliki performa yang lebih baik dibandingkan dengan bot *local search*.

Terlihat dari performa yang ditunjukkan oleh kedua bot, bot *minimax* memiliki performa lebih baik dibandingkan bot *local search*. Tentu saja hal ini karena keterbatasan yang dimiliki oleh *local search* yang diimplementasikan pada bot, yaitu berpotensi akan terjebak dalam *state local maximum*, ketika tidak ada tetangga yang bernilai lebih besar dari *current state* tetapi *state* tersebut tidak bernilai paling besar. Terlihat pula jika giliran pemain ditukar, bot *minimax* tetap mengungguli bot *local search*.

Ketika diadu dengan manusia, secara umum manusia harus bermain dengan baik untuk memenangkan permainan atau untuk mengimbangi skor lawan bila kalah. Giliran pemain yang mengawali permainan juga memengaruhi hasil permainan. Pemain yang mendapat giliran pertama relatif lebih sulit untuk menang dibandingkan pemain yang mendapat giliran kedua.

1.5. Saran Perbaikan Kode Program

Beberapa saran yang dapat penulis berikan terkait kode program adalah sebagai berikut.

1. Menggunakan suatu mekanisme yang lebih baik untuk membatasi waktu bot "berpikir", dalam hal ini mencari langkah selanjutnya yang diambil.
2. Penggunaan *objective function* yang lebih baik seperti menghitung jumlah *long chain* (rantai kotak 1x1 yang berjumlah 3 atau lebih), menghitung jumlah *short chain* (rantai kotak 1x1 yang berjumlah di bawah 3), dan lain-lain.
3. Terkait penggunaan *Local Search*, mungkin dapat menggunakan alternatif seperti *Stochastic Hill Climbing* ataupun *Random-restart Hill Climbing*.
4. Menggunakan algoritma alternatif selain algoritma *Local Search* ataupun algoritma *Minimax* dengan *Alpha Beta Pruning* yang mungkin lebih baik untuk permainan ini.

BAB 2: Kontribusi Anggota dalam Kelompok

NIM	Nama	Kontribusi
13520003	Dzaky Fattan Rizqullah	Laporan
13520018	Bariza Haqi	Minimax with Alpha-Beta Pruning Bot
13520039	Rozan Fadhil Al Hafidz	Local Search (Hill Climbing Sideways Move) Bot
13520082	Jeremy Rionaldo Pasaribu	Laporan