TUGAS BESAR I

IMPLEMENTASI MINIMAX ALGORITHM DAN LOCAL SEARCH PADA PERMAINAN DOTS AND BOXES

Laporan dibuat untuk memenuhi salah satu tugas besar mata kuliah

IF3170 Inteligensi Buatan

Disusun oleh:

13520036	**I Gede Arya Raditya P.**
13520048	**Arik Rayi Arkananta**
13520114	**Kevin Roni**
13520141	**Yoseph Alexander Siregar**

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2022

DAFTAR ISI

I. Latar Belakang

1. Dots and Boxes

Dots and Boxes merupakan permainan *turn-based* yang biasanya dimainkan oleh dua pemain. Permainan Dots and Boxes dimulai dengan suatu *grid* kosong dengan ukuran titik persegi. Ukuran *grid* dapat beragam. Pada tugas kali ini akan digunakan grid berukuran 3 x 3 (4 x 4 titik). Pemain akan saling bergantian menambahkan garis *horizontal* atau *vertikal* dengan tujuan mendapatkan *box* terbanyak. Pemain yang berhasil membentuk suatu *box* dapat menambahkan garis lagi. Permainan akan berakhir apabila semua *box* sudah berhasil terbentuk dan pemain dengan jumlah *box* terbanyak memenangkan permainan.

2. Local Search (*Hill-Climbing* Search)

Algoritma *Hill-Climbing Search* adalah salah satu contoh dari sekian algoritma *Local Search*. Konsep *Hill-Climbing Search* mirip seperti mendaki gunung dengan amnesia. Proses pencarian dimulai dari *initial state* lalu melangkah ke *state* berikutnya dengan nilai yang terus meninggi atau menurun, sesuai dengan kebutuhan. Proses pencarian akan berhenti ketika program mencapai puncak atau tidak ada lagi *state* berikutnya yang lebih baik daripada *initial state*.

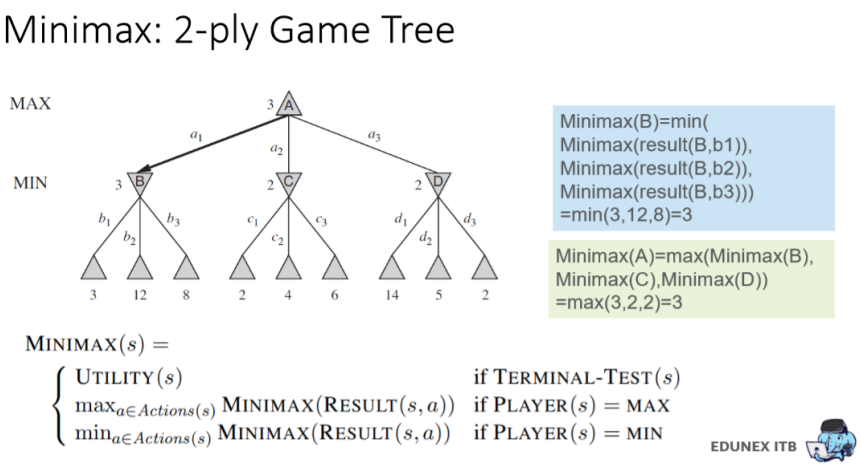
Algoritma *Hill-Climbing Search* memiliki beberapa komponen utama, yaitu:

- *Initial State* : Kondisi permainan yang sedang diamati
- *Successor* : Kumpulan kemungkinan untuk langkah berikutnya
- *Neighbor* : *Successor* dengan nilai tertinggi
- *Action* : Langkah yang diambil (pindah ke *Neighbor* atau terminasi pencarian)
- *State Value* : fungsi objektif untuk menghitung nilai pada setiap *state*

3. Adversarial Search (Minimax with Alpha-Beta Pruning)

Algoritma *minimax* termasuk salah satu dari penerapan *Adversarial Search.* Pada *adversarial search,* terdapat 6 komponen pada *Game Search Problem,* yaitu *initial state,* pemain, aksi, fungsi *result* yang didasarkan pada aksi dan state yang dikenakan aksi tersebut, fungsi terminal untuk mengecek apakah suatu state merupakan state terminal atau bukan, dan fungsi utilitas yang akan mengembalikan nilai utilitas dari suatu state. Pada *adversarial search,* kondisi permainan biasanya direpresentasikan dalam sebuah tree dengan *node*(simpul) yang merepresentasikan suatu state dan *edge*(sisi) yang merepresentasikan aksi atau gerakan pemain.

Pada algoritma *minimax*, biasanya terdapat 2 pemain saling berlawanan (sifat dasar *adversarial search* yang *competitive)* yang nantinya akan dilihat sebagai *maximizing player* dan juga *minimizing player*. Dari suatu state awal (*initial state*), yang pada umumnya akan dimiliki oleh *maximizing player* dengan tujuan untuk memaksimalkan hasil akhir yang ada. Dari state tersebut, akan diproyeksikan semua kemungkinan state yang dihasilkan apabila pemain melakukan suatu aksi. Pada setiap kemungkinan state, akan dicek apakah state tersebut merupakan terminal state, jika ya maka akan dikembalikan nilai utilitas state tersebut dengan *utility function*, jika tidak maka akan dilakukan projeksi kembali kepada state tersebut untuk semua kemungkinan state berikutnya. Pada setiap perpindahan *depth* (representasi permainan dengan *tree*), maka posisi player akan berubah (antara *max* dan *min*), perubahan posisi ini yang akan mempengaruhi fungsi *result* pada setiap state untuk pemain tersebut. Apabila suatu depth adalah untuk pemain *max*, maka suatu state akan mengembalikan nilai maksimal dari segala kemungkinan state di bawahnya dan sebaliknya untuk pemain *min.* Terkecuali untuk terminal state yang akan langsung mengembalikan nilai utilitas state tersebut.



Gambar 1.1

Sumber : Slide Perkuliahan IF3170 *Artificial Intelligence - Beyond Classical Search Part 2*

Algoritma minimax akan berjalan secara rekursif dimulai dari *initial state* hingga terminal state untuk mendapatkan hasil maksimal (asumsi *initial state* adalah milik *maximizing player*). Untuk melakukan optimalisasi pada algoritma *minimax*, diterapkan metode *alpha-beta pruning* yang akan memangkas cabang pada *tree* yang dianggap tidak akan mempengaruhi hasil akhir dari algoritma *minimax* untuk mempersingkat waktu.

II. Algoritma Program

1. Objective Function

a. Local Search (*Hill-Climbing* Search)

Fungsi Objektif akan dihitung sebagai berikut:

- Akan dilihat box yang dipisahkan oleh garis, jika box sudah memiliki 0 dan 1 garis lainnya, maka h = +0. Hal ini karena box yang belum memiliki garis atau hanya memiliki 1 garis pada semua sisinya, jika ditambahkan 1 garis tidak akan berefek apa-apa dan tidak akan menambahkan poin bagi siapapun
- Jika box sudah memiliki 2 garis, maka h = -1. Alasannya adalah jika ditambahkan 1 garis, maka pada giliran musuh berikutnya musuh dapat menambahkan 1 garis yang menambah poin bagi musuh
- Jika box sudah memiliki 3 garis, maka h = +1. Alasannya adalah jika ditambahkan 1 garis, box akan menjadi milik pemain dan menambahkan poin bagi pemain
- Jika box yang dipisahkan memiliki 3 dan 2 garis, maka h = +2. Alasannya adalah setelah menambahkan garis pada box yang memiliki 3 garis, pemain dapat melakukan aksi lagi dan menambahkan garis pada box yang sebelumnya memiliki 2 garis (sekarang sudah 3 garis).
- *Initial state* akan selalu bernilai 0

b. Minimax with Alpha-Beta Pruning

Fungsi objektif atau pada minimax lebih dikenal dengan fungsi utilitas akan mengevaluasi terminal state.

Untuk bagian ini akan dijelaskan dengan sudut pandang dari bot. Pertama-tama akan dihitung jumlah box yang dimiliki bot dikurang dengan jumlah box yang dimiliki oleh pemain lawan.

Selain itu fungsi ini juga akan menghitung *chain* yang terbentuk pada state tersebut. *Chain* yang dimaksudkan adalah kondisi dimana pemain dapat mengambil banyak box sekaligus.

Terdapat 2 jenis *chain*, yaitu *open chain* dan *closed chain. Open chain* adalah kondisi dimana *chain* yang terbentuk tidak tertutup sehingga jika bot

menutup *chain* tersebut pada salah satu ujungnya (dengan asumsi tidak membentuk suatu box sehingga giliran berikutnya adalah pemain lawan) maka pemain lawan dapat mengambil banyak *box* yang ada pada chain tersebut.

Sedangkan *closed chain* adalah kondisi dimana *chain* tertutup pada salah satu ujungnya atau keduanya, sehingga bot dapat mengambil seluruh *box* yang ada pada chain tersebut.

Fungsi objektif akan mengembalikan nilai selisih box yang dimiliki ditambahkan nilai *closed chain* dikurangi dengan *open chain* yang terbentuk.

2. Implementasi Local Search (*Hill-Climbing* Search) untuk Permainan Dots and Boxes

Untuk algoritma local search, akan digunakan *Hill-Climbing*. Dari sekian banyak jenis algoritma *local search*, kami memutuskan untuk menggunakan *Hill-Climbing* karena hal-hal berikut:

- *State* yang perlu dibangkitkan tidak banyak sehingga proses pembangkitan tidak memakan waktu yang lama
- Algoritma ini menelusuri semua suksesor sehingga kemungkinan mendapatkan suksesor dengan nilai tertinggi akan lebih baik

Untuk algoritma *Simulated Annealing*, ada temperatur sehingga ketika temperatur mencapai 0 dan tidak ditemukan suksesor dengan nilai yang lebih baik harus dilakukan proses terminasi. Selain itu, menemukan suksesor dengan nilai yang lebih baik tetapi tidak yang terbaik akan menghentikan proses pencarian begitu saja.

Selain itu, *genetic algorithm* tidak bisa diterapkan pada permainan ini. Penerapan *genetic algorithm* pada game *Dots And Boxes* memerlukan langkah *crossover* dan *mutation* yang akan melanggar aturan dari permainan ini. Pemberlakuan langkah *crossover* perlu menambahkan atau menghilangkan giliran yang jelas tidak dapat dilakukan oleh player, mengingat permainan interaktif dan

dilakukan oleh 2 player. Begitu juga dengan *mutation* akan melanggar aturan dasar permainan.

Dari state yang ada, akan dicari *neighbor* yang berupa *highest-value successor.* Untuk komponen dalam algoritma *Hill-Climbing Search* adalah sebagai berikut:

- *Initial State* : Kondisi permainan yang sedang berjalan
- *Successor* : Kumpulan kemungkinan jalan yang bisa dilakukan oleh bot (garis yang belum terhubung)
- *Neighbor* : *Successor* dengan nilai *state value* tertinggi, bot akan memilih *neighbor* sebagai *action*
- *Action* : Langkah yang diambil (pindah ke *neighbor*). Tidak dilakukan *action* terminasi pencarian karena *bot* harus selalu memilih jalan
- *State Value* : fungsi objektif untuk menghitung nilai pada setiap *state* yang telah dijelaskan II.1.a

Algoritma akan dimulai pada initial state dan akan meng-*generate* semua suksesor beserta *state value*-nya untuk giliran pemain. *State* akan berpindah ke suksesor yang memiliki *state value* paling tinggi. Algoritma tidak diiterasi dan langsung diambil *action* ketika pencarian selesai. Hal ini disebabkan oleh permainan yang bersifat *turn-taking*. *State* berikutnya ditentukan oleh pemain lawan sehingga tidak bisa meneruskan pencarian.

Pada *local search* umumnya ada proses terminasi ketika proses pencarian dilakukan. Pada implementasi kami, tidak dilakukan terminasi ketika tidak ada *successor* yang bernilai lebih baik daripada *initial state*. Hal ini disebabkan oleh keharusan *bot* untuk jalan di setiap giliran. Maka jika semua *successor* memiliki *state value* < 0, tetap dilakukan pergerakan ke *successor* dengan *state value* tertinggi.

3. Implementasi Minimax with Alpha-Beta Pruning untuk Permainan Dots and Boxes

Pada permainan Dots and Boxes, salah satu hal penting yang harus diperhatikan adalah meskipun permainan ini merupakan permainan *turn taking*, suatu pemain dapat memperpanjang gilirannya jika pemain tersebut berhasil membentuk sebuah kotak. Hal ini akan mempengaruhi proses rekursif pada algoritma minimax.

Pada algoritma *minimax* dengan *alpha-beta pruning,* parameter yang akan diterima adalah state permainan saat itu, *depth* atau kedalaman untuk pengecekan, nilai *alpha* dan *beta* untuk metode *pruning* dan juga parameter giliran pemain. Pada implementasi algoritma minimax pada tugas ini, *maximizing player nya* adalah untuk sisi bot minimax dan nilai *alpha* pada awal diinisialisasikan dengan nilai yang sangat kecil dan nilai *beta* dengan nilai yang sangat besar . Pada algoritma minimax ini, pertama-tama akan dilakukan pengecekan apakah state yang diterima sebagai parameter merupakan terminal state atau tidak. Pengecekan terminal state dilakukan dengan memanfaatkan parameter *depth* yang diterima dan juga kondisi permainan. Apabila *depth* yang diterima sudah bernilai 0 atau permainan sudah berakhir maka state tersebut akan dianggap sebagai sebuah terminal state dan akan mengembalikan nilai utility dari state tersebut dengan memanfaatkan *objective function* yang sudah dijelaskan sebelumnya.

Jika state yang diterima sebagai parameter bukanlah terminal state, maka akan dilakukan pengecekan terlebih dahulu apakah sekarang merupakan giliran dari bot atau tidak. Proses ini dilakukan untuk membedakan perilaku yang akan dilakukan pada state permainan nantinya. Pada umumnya perlakuan yang dilakukan pada state sama, yaitu dengan memproyeksikan semua ‘anak’ dari state tersebut dengan suatu fungsi (tiap anak merepresentasikan state terbaru setelah pemain menambahkan garis), lalu kepada tiap ‘anak’ tersebut akan dilakukan proses rekursif minimax dengan parameter state adalah anak tersebut, *depth* yang sudah berkurang satu, nilai *alpha* dan *beta,* dan juga giliran pemain (yang sudah dijelaskan pada paragraf pertama).

Pada permainan dots and boxes ini, tiap anak dari suatu state didapatkan dengan mengambil semua kemungkinan yang ada. Kemungkinan ini didapatkan dengan mengecek pada status *row* dan *column* pada state tersebut. Jika terdapat *row* atau *column* yang tidak terisi, maka akan didapatkan 'anak' dengan state saat *row* atau *column* tersebut terisi. Untuk tiap 'anak' itu, akan di-*update* nilai dari *board status* yang merepresentasikan banyaknya garis pada tiap box untuk state tersebut. Apabila setelah nilai *board status* di-*update* dan terdapat box yang bernilai 4, maka dapat disimpulkan bahwa pemain tersebut berhasil membentuk sebuah kotak dan giliran berikutnya masih merupakan giliran pemain tersebut. Metode inilah yang akan dimanfaatkan saat proses rekursif nantinya pada penerapan minimax pada tiap 'anak'.

Hasil yang dikembalikan dari algoritma minimax pada tahap ini serta nilai *alpha* dan *beta*, akan diproses berdasarkan giliran pemain saat itu. Apabila saat itu merupakan giliran dari *maximizing player* (bot) maka sebelum proses pengaplikasian minimax pada tiap 'anak' akan diinisialisasikan terlebih dahulu nilai max eval dengan nilai yang sangat kecil, lalu hasil minimax pada tiap state 'anak' misal akan disimpan, misal pada variabel eval, dan dibandingkan dengan nilai max eval dan diambil nilai terbesar lalu di-*assign* pada max eval dan juga perbandingan antara eval dengan nilai *alpha* saat itu dengan nilai yang lebih besar di-*assign* pada *alpha.*

Sebaliknya terjadi jika saat itu merupakan giliran pemain lawan (bukan bot minimax), sebelum proses pengaplikasian minimax pada tiap 'anak' akan diinisialisasikan terlebih dahulu nilai min eval dengan nilai yang sangat besar, lalu hasil minimax pada tiap state 'anak' misal akan disimpan, misal pada variabel eval, dan dibandingkan dengan nilai min eval dan diambil nilai terkecil lalu di-*assign* pada min eval dan juga perbandingan antara eval dengan nilai *beta* saat itu dengan nilai yang lebih kecil di-*assign* pada *beta.*

Nilai *alpha* dan *beta* ini akan diteruskan kepada proyeksi 'anak' dari state tersebut dan dapat di-*update* berdasarkan proses rekursif minimax yang sudah dijelaskan sebelumnya. Pruning akan dilakukan jika nilai *alpha* lebih besar atau sama dengan *beta*.

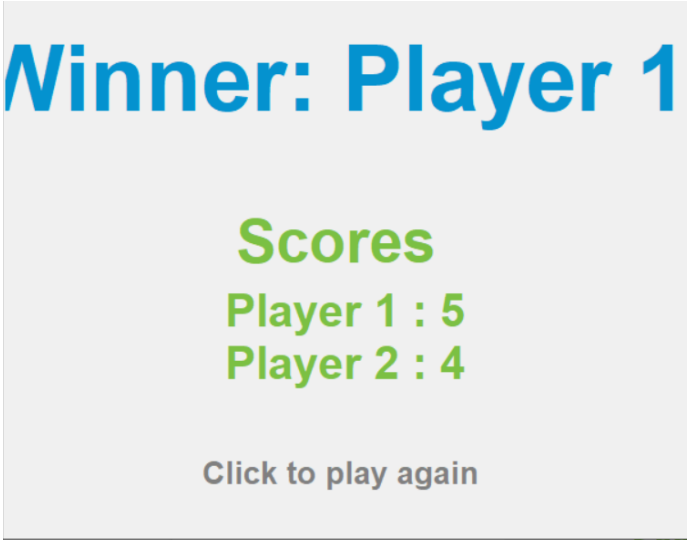
III. Hasil Pertandingan

1. Bot Minimax vs Manusia (5)

```
game_instance = Dots_and_Boxes(MinimaxBot(), None)
```

Permainan dilakukan sebanyak 5 ronde dengan Bot Minimax sebagai Player 1 dan Manusia sebagai Player 2. Pergantian ronde dilakukan dengan *play again* sehingga giliran kedua pemain akan berganti-gantian di tiap ronde.

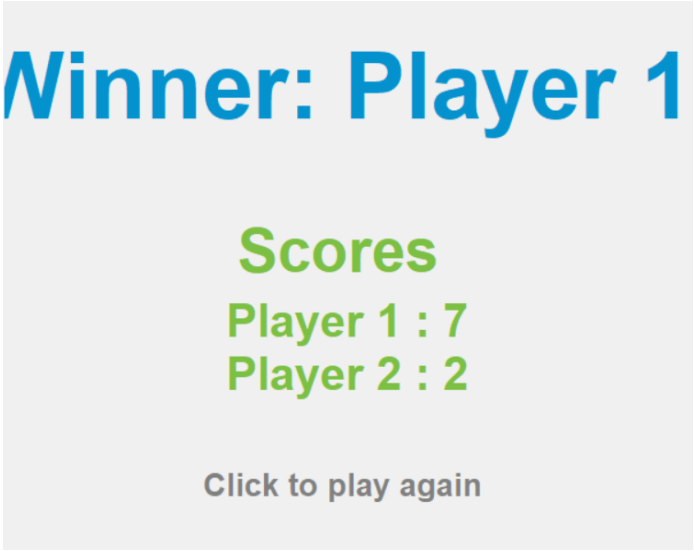
a. Ronde 1



Gambar 3.1 Ronde 1 Bot Minimax vs Manusia

Sumber : Arsip Pribadi

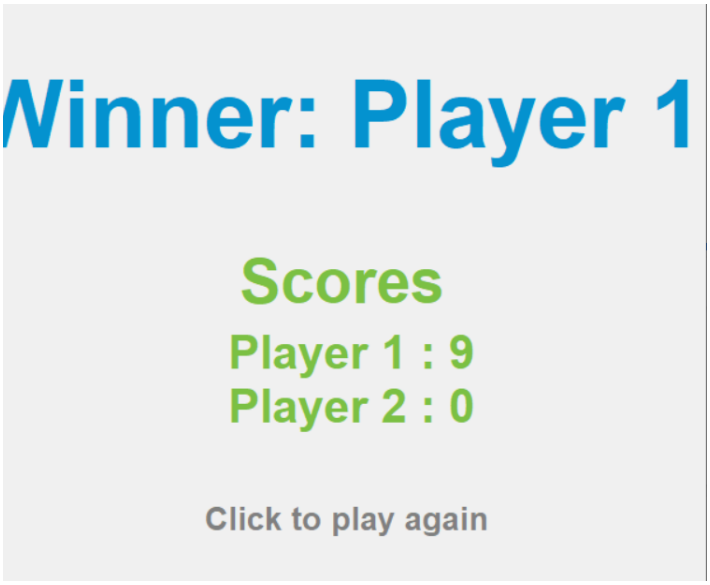
b. Ronde 2



Gambar 3.2 Ronde 2 Bot Minimax vs Manusia

Sumber : Arsip Pribadi

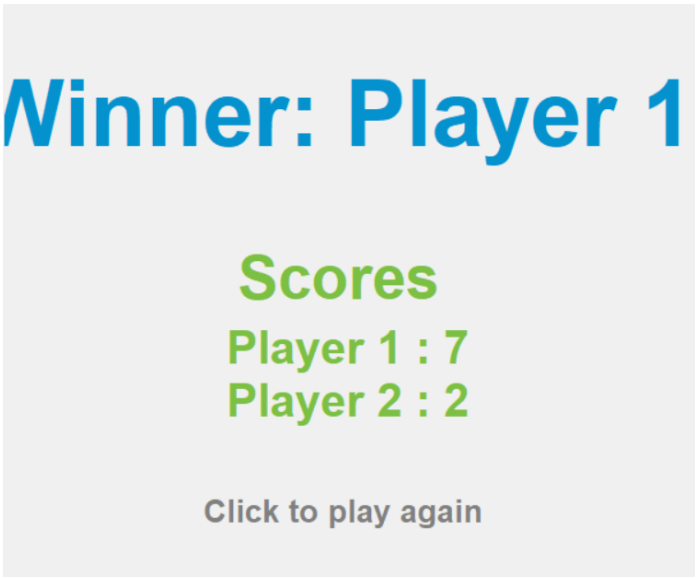
c. Ronde 3



Gambar 3.3 Ronde 3 Bot Minimax vs Manusia

Sumber : Arsip Pribadi

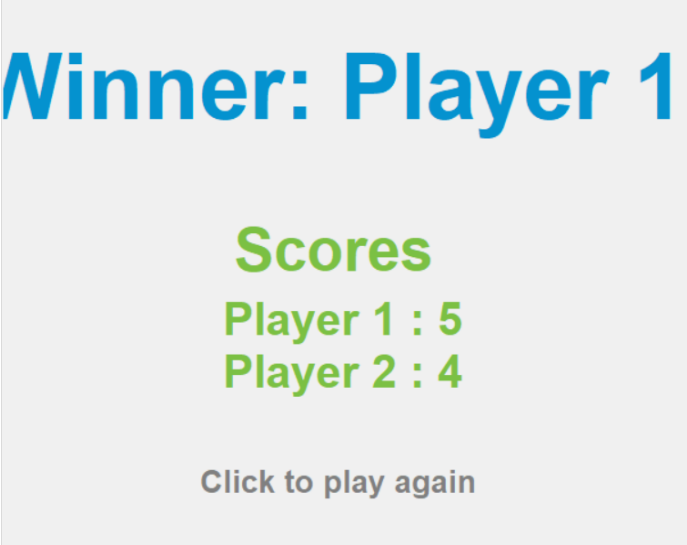
d. Ronde 4



Gambar 3.4 Ronde 4 Bot Minimax vs Manusia

Sumber : Arsip Pribadi

e. Ronde 5



Gambar 3.5 Ronde 5 Bot Minimax vs Manusia

Sumber : Arsip Pribadi

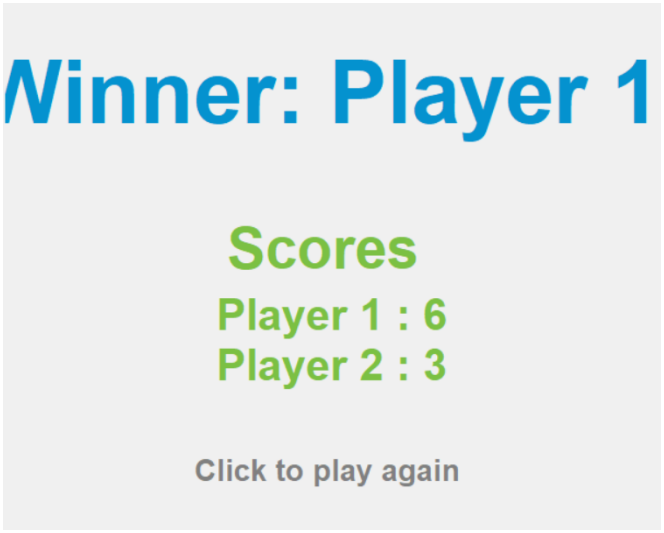
Dari 5 kali ronde yang dilakukan didapatkan hasil : 5 kali dimenangkan oleh Bot Minimax.

2. Bot Local Search vs Manusia (5)

```
game_instance = Dots_and_Boxes(LocalSearchBot(), None)
```

Permainan dilakukan sebanyak 5 ronde dengan Bot LocalSearch sebagai Player 1 dan Manusia sebagai Player 2. Pergantian ronde dilakukan dengan *play again* sehingga giliran kedua pemain akan berganti-gantian di tiap ronde.

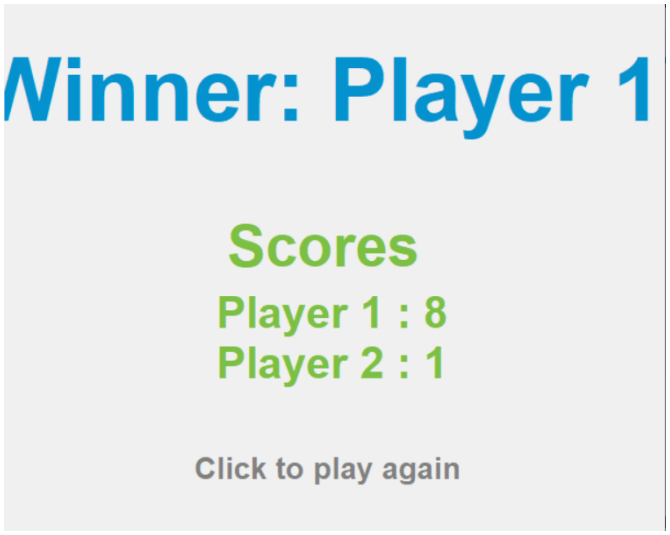
a. Round 1



Gambar 3.6 Ronde 1 Bot LocalSearch vs Manusia

Sumber : Arsip Pribadi

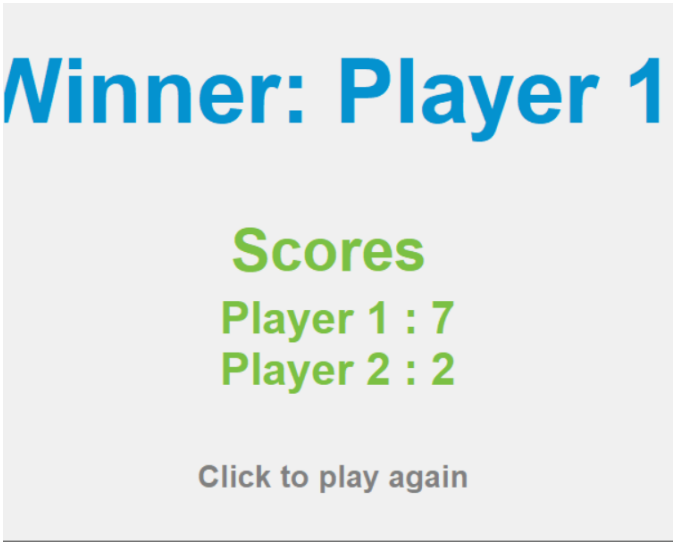
b. Round 2



Gambar 3.7 Ronde 2 Bot LocalSearch vs Manusia

Sumber : Arsip Pribadi

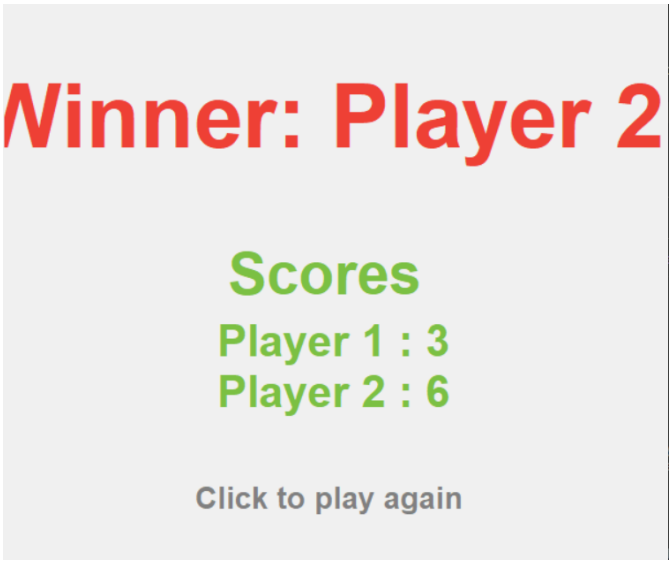
c. Round 3



Gambar 3.8 Ronde 3 Bot LocalSearch vs Manusia

Sumber : Arsip Pribadi

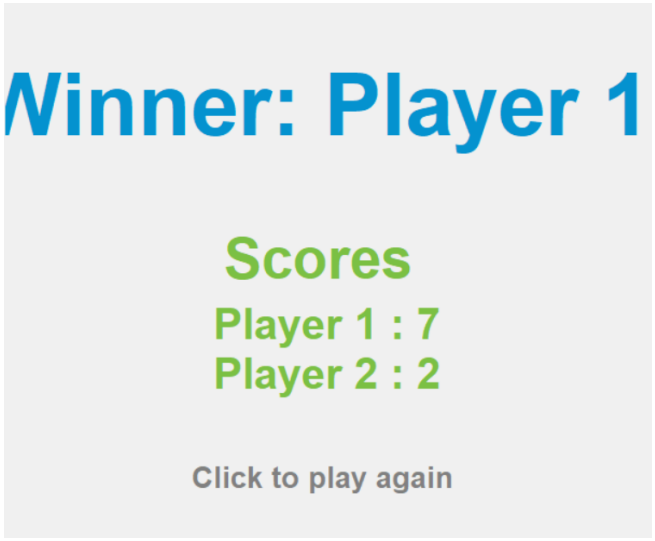
d. Round 4



Gambar 3.9 Ronde 4 Bot LocalSearch vs Manusia

Sumber : Arsip Pribadi

e. Round 5



Gambar 3.10 Ronde 5 Bot LocalSearch vs Manusia

Sumber : Arsip Pribadi

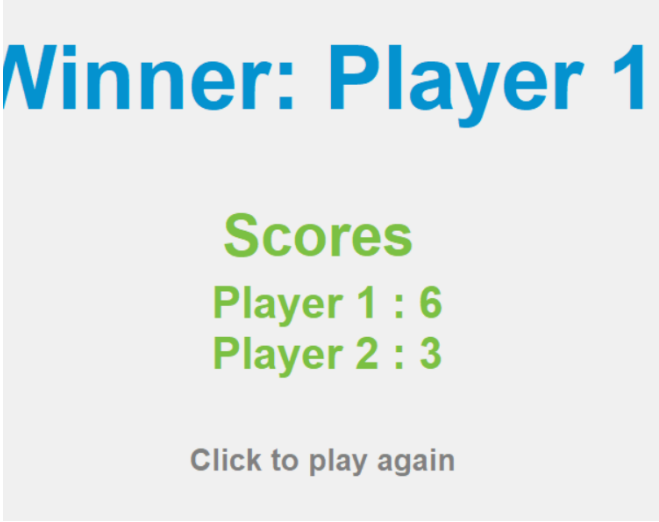
Dari 5 kali ronde yang dilakukan didapatkan hasil : 4 kali dimenangkan oleh Bot LocalSearch dan 1 kali oleh manusia.

3. Bot Minimax vs Bot Local Search (5)

```
game_instance = Dots_and_Boxes(MinimaxBot(), LocalSearchBot())
```

Permainan dilakukan sebanyak 5 ronde dengan MinimaxBot sebagai Player 1 dan LocalSearchBot sebagai Player 2. Pergantian ronde dilakukan dengan *play again* sehingga giliran kedua pemain akan berganti-gantian di tiap ronde.

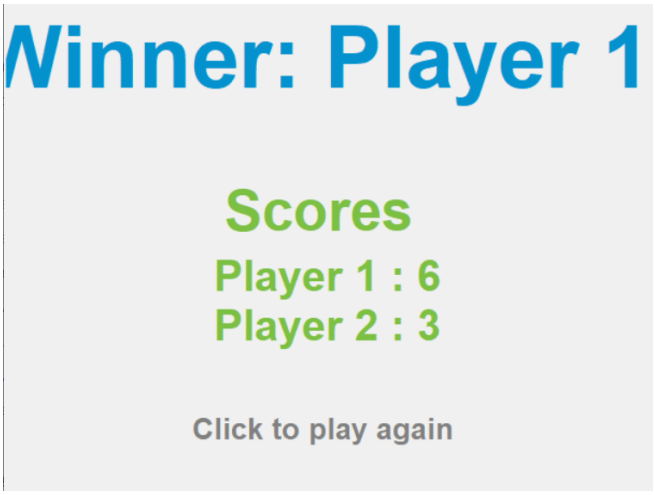
a. Ronde 1



Gambar 3.11 Ronde 1 Bot Minimax vs Bot Localsearch

Sumber : Arsip Pribadi

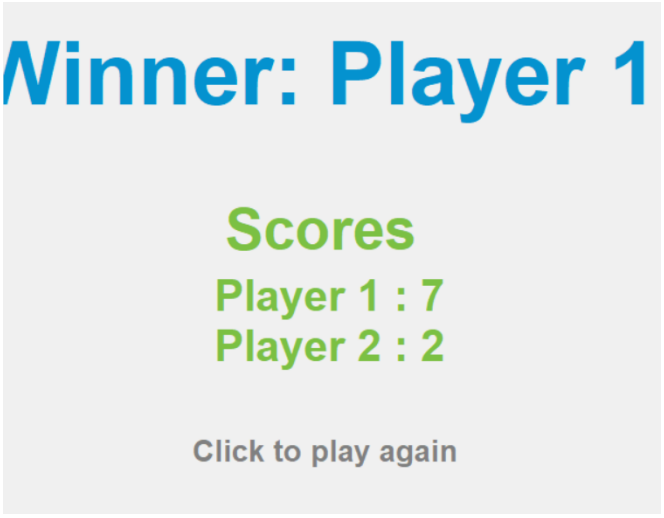
b. Ronde 2



Gambar 3.12 Ronde 2 Bot Minimax vs Bot Localsearch

Sumber : Arsip Pribadi

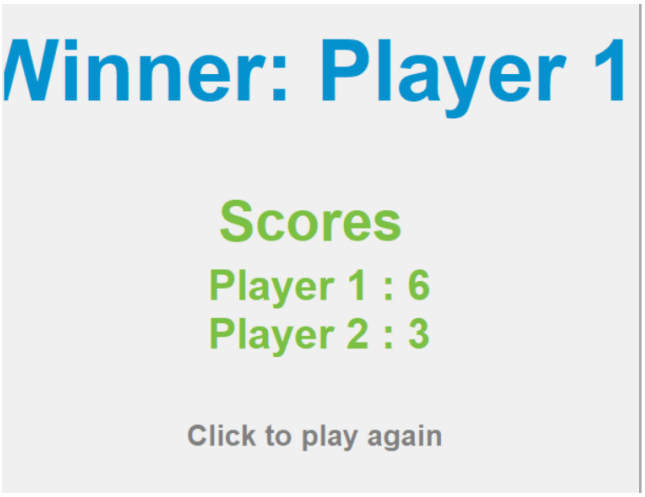
c. Ronde 3



Gambar 3.13 Ronde 3 Bot Minimax vs Bot Localsearch

Sumber : Arsip Pribadi

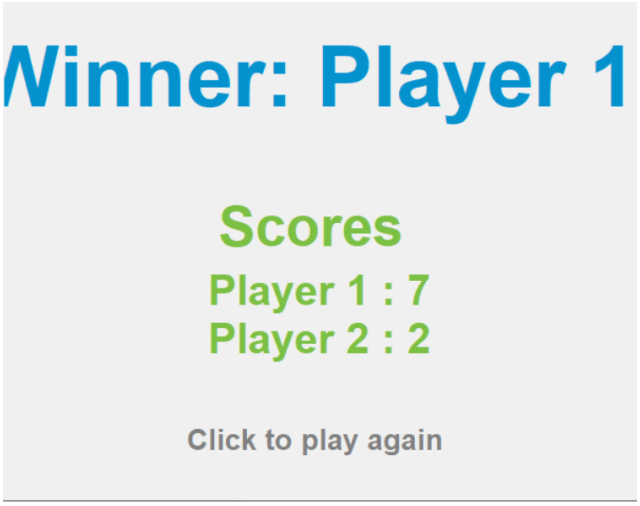
d. Ronde 4



Gambar 3.14 Ronde 4 Bot Minimax vs Bot Localsearch

Sumber : Arsip Pribadi

e. Ronde 5



Gambar 3.15 Ronde 5 Bot Minimax vs Bot Localsearch

Sumber : Arsip Pribadi

Dari 5 kali ronde yang dilakukan didapatkan hasil : 5 kali dimenangkan oleh Bot Minimax

4. Persentase Kemenangan

No	Keterangan	Bot Local Search	Bot Minimax
1.	Main	10	10
2.	Menang	4	10
3.	Kalah	6	0
4.	**Persentase**	**40%**	**100%**

Dari statistik yang di atas dapat dilihat bahwa algoritma Minimax bekerja lebih baik pada permainan *Dots and Boxes* dibandingkan algoritma *local search.*

IV. Saran

1. Untuk algoritma minimax, dapat dilakukan optimasi jika dihilangkan batasan waktu bagi bot untuk memikirkan dan mengambil suatu aksi karena untuk memaksimalkan kinerja bot bisa ditambahkan *depth* eksplorasi yang tentunya akan memperlama waktu kerja bot.
2. Mengembangkan *objective function* dengan melakukan eksplorasi terhadap kemungkinan-kemungkinan yang bisa terjadi permainan *dots and boxes* secara lebih mendetail.
3. Permainan ini tidak cocok jika menggunakan algoritma *local search* secara penuh, seperti algoritma *steepest ascent hill climbing*, karena tidak memungkinkannya dilakukan terminasi sehingga bot tidak berjalan dan berhenti bergerak (menyerah), sama juga dengan algoritma *simulated annealing* karena adanya temperatur yang tidak memungkinkan untuk memberhentikan permainan jika temperatur = 0.

V. Pembagian Tugas

No	Nama	NIM	Pembagian Tugas
1.	I Gede Arya Raditya P.	13520036	Laporan, Implementasi Bot Minimax
2.	Arik Rayi Arkananta	13520048	Laporan, Implementasi Bot Local Search
3.	Kevin Roni	13520114	Laporan, Implementasi Bot Local Search
4.	Yoseph Alexander Siregar	13520151	Laporan, Implementasi Bot Minimax