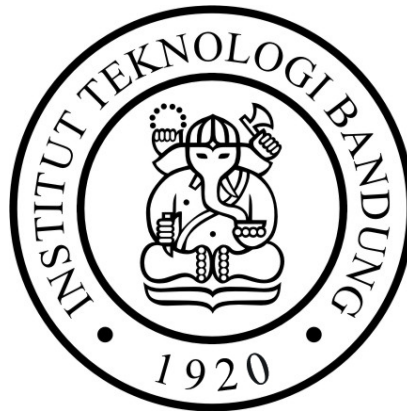


**Tugas Besar 1 IF3170 Inteligensi Buatan**  
**Implementasi *Minimax Algorithm dan Local Search* pada Permainan**  
***Dots and Boxes***



**Disusun oleh:**

**Kelompok 34**

Fayza Nadia / 13520001

Nadia Mareta Putri Leiden / 13520007

Taufan Fajarama Putrawansyah / 13520031

Mohamad Daffa Argakoesoemah / 13520118

**Program Studi Teknik Informatika**

**Sekolah Teknik Elektro dan Informatika - Institut Teknologi Bandung**

**Jl. Ganesha 10, Bandung 40132**

**2022**

# Daftar Isi

<b>Daftar Isi</b>	<b>1</b>
<b>Algoritma MiniMax</b>	<b>2</b>
a. Utility Function	2
b. Penjelasan Umum Algoritma MiniMax	2
<b>Algoritma Local Search</b>	<b>4</b>
a. Heuristic Function	4
b. Penjelasan Umum Algoritma Local Search	4
<b>Penjelasan Hasil Pertandingan</b>	<b>5</b>
a. Bot Minimax vs Manusia	5
b. Bot Local Search vs Manusia	7
c. Bot Minimax vs Local Search	10
<b>Saran Perbaikan terhadap Kode yang Dibuat</b>	<b>13</b>
a. Algoritma Minimax	13
b. Algoritma Local Search	13
<b>Tabel Pembagian Tugas</b>	<b>14</b>

# Algoritma MiniMax

## a. Utility Function

*Utility function* dari Algoritma MiniMax adalah memastikan bahwa banyak kotak dari bot lebih besar dibandingkan lawan. Sehingga *utility function* nya adalah

1. Dari perspektif Player1: Jumlah box player1 - jumlah box player 2
2. Dari perspektif Player 2: Jumlah box player 2 - jumlah box player 1

Bot dinyatakan menang jika nilai selisihnya adalah positif karena sesuai dengan definisi di atas yaitu jika jumlah kotaknya lebih besar dibandingkan lawan.

## b. Penjelasan Umum Algoritma MiniMax

Algoritma MiniMax disimpan dalam file `MinimaxBot.py` dimulai dengan dibuatnya kelas `MinimaxBot` yang terdiri dari beberapa fungsi:

### 1. Fungsi `utility_function`

Jika garis yang dibentuk bernilai -4 maka player 1 akan mendapatkan poin, sebaliknya jika garis yang dibentuk bernilai 4 maka player 2 akan mendapatkan poin. Nilai 4 merepresentasikan 1 kotak utuh yang dibentuk oleh 4 garis.

Penjelasan *utility function* sama dengan penjelasan pada bagian (a).

### 2. Fungsi `terminal_test`

Fungsi ini bertujuan untuk memeriksa apakah *current state* sudah mencapai *terminal state* atau belum. Jika belum *state* belum mencapai *state* terminal, fungsi akan me-return False. Sebaliknya, jika sudah mencapai *terminal state* maka fungsi me-return True.

### 3. Fungsi `get_possible_actions`

Fungsi ini mengembalikan aksi-aksi yang mungkin dari *state* masukan. Baik itu secara *row* yaitu garis horizontal dan secara *col* yaitu garis vertikal.

#### 4. Fungsi `get_state_from_action`

Fungsi ini digunakan untuk mengembalikan *state* yang akan dihasilkan dari pergerakan yang dilakukan dari aksi yang telah dipilih. Jika misalnya berhasil membentuk 4 garis dalam `board_status`, maka variabel `point_scored` akan bernilai `True`. Setelah itu fungsi *me-return* *state* yang baru dan variabel `point_scored`.

#### 5. Fungsi `minimax`

Fungsi `minimax` dijalankan secara rekursif. Pada dasarnya fungsi ini mencari hasil terbaik yang paling mungkin. Idenya adalah bot mendapatkan value “Max”, sedangkan lawan mendapatkan value “Min”. Variabel `is_maximize` berbentuk *boolean* untuk mengetahui saat ini giliran dari bot atau lawannya. Jika giliran dari bot maka akan bernilai `True`. Jika `is_maximize` bernilai `True`, maka program akan memastikan bot mendapatkan nilai maksimum dengan masuk ke *if(is\_maximize)*. Jika pergerakan atau aksi menghasilkan kotak ditandai dengan adanya `point_scored`, maka nilai variabel `maximize` akan `True` kembali. Artinya pergerakan akan terus dilakukan selama masih bisa membentuk kotak.

Sehingga langkahnya adalah Max terus sampai giliran lawan.

Kemudian, jika saat ini merupakan giliran lawan maka akan masuk ke dalam algoritma selanjutnya yang ditandai dengan `else`. Adapun untuk algoritma ini merupakan kebalikan dari yang sebelumnya, yaitu untuk memilih langkah yang memperburuk kondisi dari lawannya. Jika pergerakan lawan membentuk kotak, maka nilai `maximize` akan menjadi `False`.

Sehingga langkahnya adalah Min sampai giliran bot yang menggunakan Algoritma `MiniMax`.

Selanjutnya, untuk `alpha beta pruning` akan dilakukan jika nilai `beta` lebih kecil atau sama dengan nilai `alpha`.

# Algoritma Local Search

## a. Heuristic Function

Fungsi heuristik yang digunakan adalah jumlah box milik *bot*. Hal ini karena kita ingin memaksimalkan jumlah *box* yang dimiliki oleh *bot*. Alasan lain adalah karena algoritma yang dipilih, yaitu *sideways move hill climbing*. Pada algoritma ini, *state* akan berpindah jika nilai fungsi heuristik lebih besar atau sama dengan nilai heuristik *current state*. Dengan menghitung jumlah *box*, nilai heuristik tidak akan kurang dari nilai heuristik *current state*.

$$h = \sum b$$

## b. Penjelasan Umum Algoritma Local Search

Algoritma local search yang digunakan adalah algoritma *sideways move hill-climbing*.

Langkah pencarian dengan *sideways move hill-climbing* sebagai berikut:

1. Pencarian solusi dimulai dari *generate possible action* yang dapat dilakukan (membentuk satu garis di manapun) dan memasukkan semua *next possible state* dari *possible action* ke dalam suatu *list of possible states*.
2. Menghitung *heuristic value* pada state awal (*initial state*).
3. Melakukan *loop* pada setiap *next possible state* untuk membandingkan nilai *heuristic function* setiap *next possible state*. Jika *heuristic value* tersebut lebih besar dari *current heuristic value*, maka *current heuristic value* akan diganti dengan *heuristic value* dari *next possible state*. Kemudian, *index next possible state* disimpan pada suatu *list*, beserta *heuristic value*-nya.
4. *Looping* diulang hingga semua *possible state* sudah diperiksa dan menghasilkan *list of index next possible state* dan *list of next heuristic value*.
5. Kemudian, jika terdapat beberapa *next possible state* dengan *heuristic value* yang sama, maka *index next possible state* dipilih secara *random*. Akan tetapi, jika hanya terdapat satu *next possible state* dengan nilai maksimum, maka melakukan *possible action* tersebut.
6. Algoritma dilakukan hingga tidak garis yang dapat dibentuk.

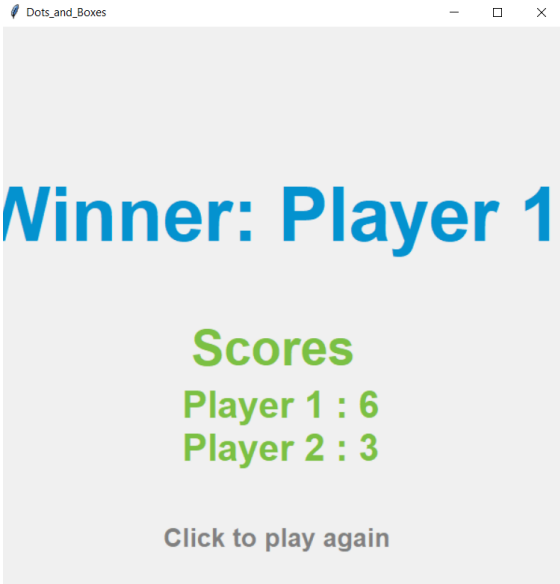
## Penjelasan Hasil Pertandingan

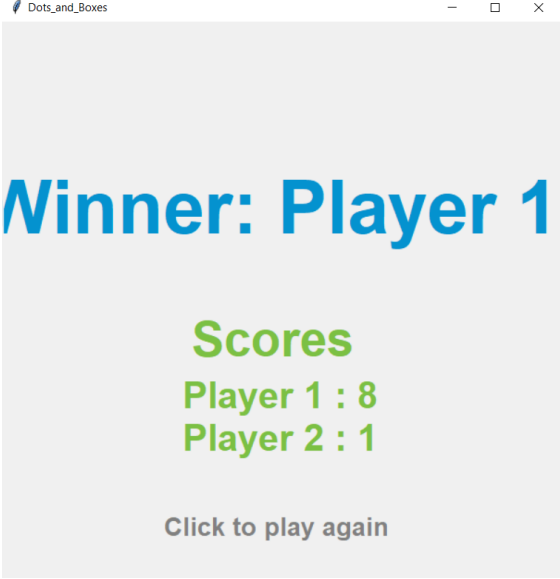
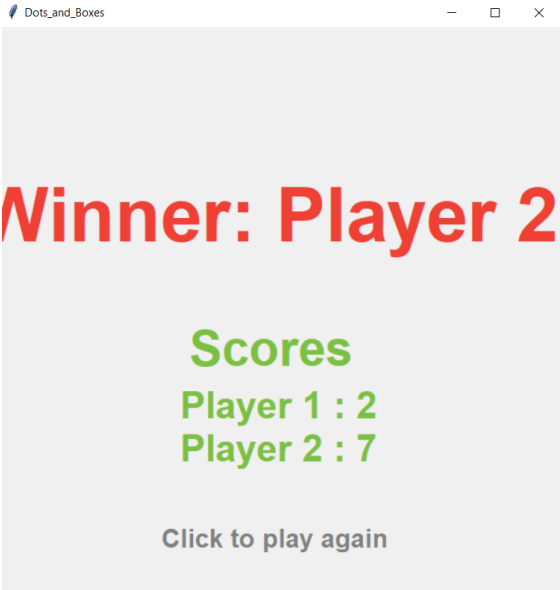
### a. Bot Minimax vs Manusia



Jumlah Menang: 2

Jumlah Kalah: 3

Persentase Kemenangan:  $\frac{2}{5} * 100 = 40\%$

No	Gambar	Status Bot
1		Kalah

2		Kalah
3		Menang

4		Menang
5		Kalah

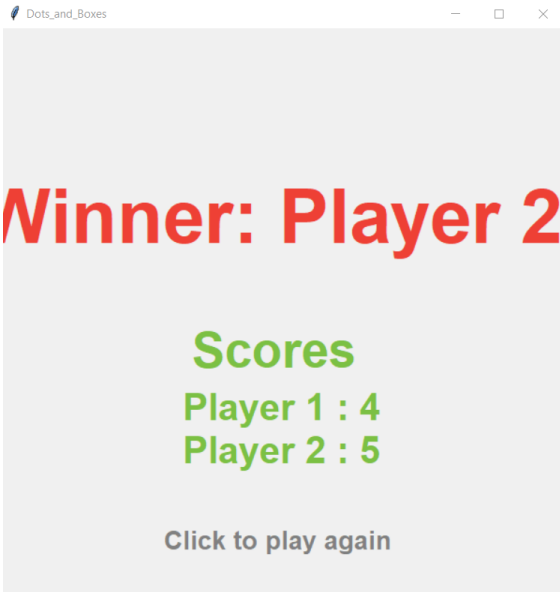
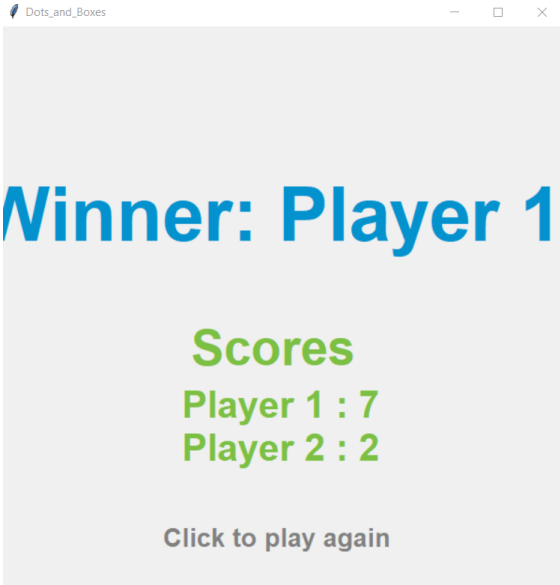
**b. Bot *Local Search* vs Manusia**

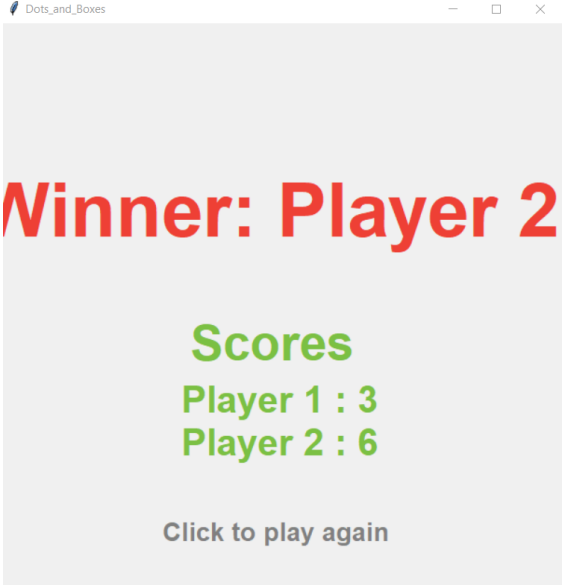
Jumlah Menang: 4


Jumlah Kalah: 1

Persentase Kemenangan:  $\frac{4}{5} * 100\% = 80\%$



No	Gambar	Status Bot
1		Menang
2		Kalah

3		Menang
4		Menang

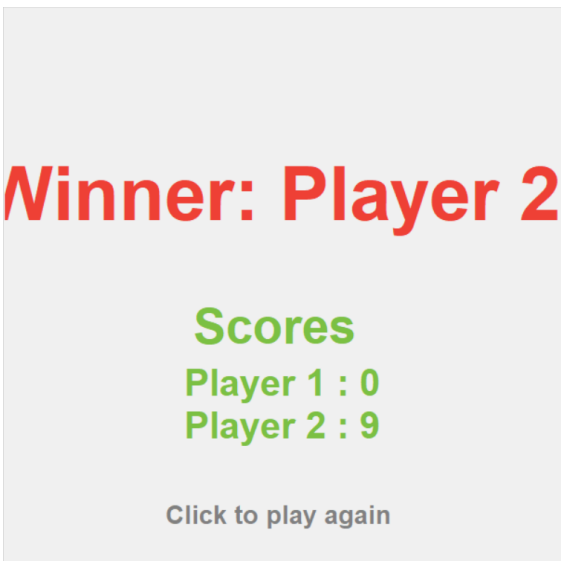
5		Menang
---	---	--------

c. **Bot Minimax vs *Local Search***

Jumlah Menang (*local search* - minimax): 5 - 0

Jumlah Kalah (*local search* - minimax): 0 - 5

Persentase Kemenangan (*local search* - minimax): 100% - 0%

No	Gambar	Status Bot
1		<i>Local search</i> menang

2		<i>Local search menang</i>
3		<i>Local search menang</i>

4		<i>Local search menang</i>
5		<i>Local search menang</i>

## Saran Perbaikan terhadap Kode yang Dibuat

### a. Algoritma Minimax

Algoritma Minimax dengan Alpha Beta Pruning dapat ditingkatkan efisiensinya dengan mengimplementasikan *transposition table*. Dengan adanya tabel ini, *state* yang sudah di-*explore* anaknya tidak perlu di-*explore* lagi. Hal ini bisa mempercepat pencarian. Selain itu, fungsi objektif yang digunakan pada algoritma Minimax dengan Alpha Beta Pruning dapat diperbaiki menjadi lebih bagus. Misalnya, fungsi objektif tidak hanya berdasarkan selisih jumlah kotak *bot* dengan lawan, tetapi juga melihat posisi kotak-kotak *bot* maupun musuh.

### b. Algoritma Local Search

Algoritma *local search* dapat ditingkatkan efisiensinya dengan mengimplementasikan pertimbangan *state* yang memiliki *chain loop*. *Chain loop* memiliki peran yang cukup krusial dalam permainan *Dots & Boxes* di mana pemain dapat memprediksi alur gerak permainan. Tidak hanya itu, pemain juga dapat ‘memutus’ alur lawan untuk memenangkan pertandingan. Implementasi *chain loop* dapat dilakukan pada saat perhitungan *value* atau nilai heuristik dari masing-masing *state*. Sehingga, *bot* dapat memenangkan pertandingan dengan gerakan yang lebih pintar dengan adanya implementasi *chain loop* tersebut.

**Tabel Pembagian Tugas**

No	Nama	NIM	Tugas
1	Fayza Nadia	13520001	Algoritma <i>local search</i>
2	Nadia Mareta Putri Leiden	13520007	Algoritma Minimax
3	Taufan Fajarama Putrawansyah	13520031	Algoritma <i>local search</i>
4	Mohammad Daffa Argakoesoemah	13520118	Algoritma Minimax