Tugas Besar I IF3170 Inteligensi Buatan

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



Oleh Kelompok 28:

Timothy Stanley Setiawan 13520028

Andreas Indra Kurniawan 13520091

Adzka Ahmadetya Zaidan 13520127

Thirafi Najwan Kurniatama 13520157

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2022

# *Objective Function* yang Digunakan

## *Utility Function* untuk Bot dengan *Minimax Alpha Beta Pruning*

Implementasi *utility function* pada bot dengan algoritma *minimax alpha beta pruning* merupakan sebagai berikut.

| Baris | Kode |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | def utility(self, state: GameState, player):  player1\_score = len(np.argwhere(state.board\_status == -4))  player2\_score = len(np.argwhere(state.board\_status == 4))    block3 = len(np.argwhere(abs(state.board\_status) == 3))  block2 = len(np.argwhere(abs(state.board\_status) == 2))  if block3 + block2 > 5:  if self.chain(state):point = -1  return player1\_score - player2\_score -5    return player1\_score - player2\_score |

Pada utility function, *state* yang di *pass* oleh program merupakan *state* ketika papan permainan ditambahkan garis oleh bot. Pertama akan dilakukan pengecekan pada status papan, jika bernilai -4 maka kotak pada papan tersebut sudah dimiliki oleh player 1 sedangkan jika bernilai 4 maka kotak pada papan tersebut dimiliki oleh player 2. Hal ini digunakan untuk melihat skor akhir dari kedua pemain.

Variabel block3, block2, merepresentasikan jumlah kotak pada papan yang sudah memiliki 3 garis(block3), 2 garis(block2). Variabel ini berfungsi untuk mengetahui jumlah *block* yang memiliki 3 garis dan 2 garis sehingga dapat menentukan apakah garis aman untuk diletakan pada kotak tersebut dengan asumsi kotak yang memiliki 3 garis dan 2 garis dapat menimbulkan “*chain*” (perolehan poin yang banyak) untuk lawan. *State* yang terindikasi menghasilkan “*chain*” akan diberi penalti -5 poin agar *state* tersebut nantinya tidak akan terambil.

## *Objective Function* untuk Bot dengan *Local Search Hill-Climbing with Sideways Move*

Implementasi *objective function* pada bot dengan algoritma *Hill-Climbing with Sideways Move* merupakan sebagai berikut.

| Baris | Kode |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | def objective(self, state: GameState) -> int:  total = 0  if state.player1\_turn:  sign = -1 # acting as player 1  else:  sign = 1 # acting as player 2  for i in state.board\_status:  for j in i:  nextMoveState = j \* sign  if nextMoveState == 3:  total -= 10  elif nextMoveState == 4:  total += 10  elif nextMoveState == 2:  total += 5  return total |

Bot akan melakukan optimasi *objective function* dengan mencari nilai *objective* (variable total pada baris 2) sebesar mungkin. Secara umum, *objective function* ini menilai seberapa bagus gerakan yang dilakukan *current state* untuk 1 *state* setelahnya. Baris 2 mengawali nilai *objective* sebesar 0. Kemudian melakukan *looping* pengecekkan setiap *move* yang mungkin dilakukan dengan representasi nilai pada variabel j.

Variabel j memiliki domain nilai berupa integer -4 hingga 4. Variabel j merupakan representasi jumlah sisi yang terisi dalam suatu sel kotak permainan *dots and boxes*. Negatif berarti pengisian terakhir dilakukan oleh player 1 (bot mungkin menjadi player 1 atau player 2), sedangkan nilai positif berarti pengisian terakhir dilakukan oleh player 2. Variabel j bernilai 4 apabila sudah terisi seluruh sisi dan membentuk kotak dengan pengisian terakhir dilakukan oleh player 2.

Untuk memudahkan, representasi j dibuat menjadi variabel nextMoveState dengan penentuan bot sebagai player 1 atau player 2 berdasarkan variabel sign yang didapat (-1 atau 1). Pada baris 11, nextMoveState bernilai 3 berarti *move* menghasilkan sel kotak yang memiliki sisi 3 yang dapat diisi oleh lawan pada giliran selanjutnya. Oleh karena itu, nilai *total objective function* dikurangi sebanyak 10 untuk menunjukkan gerakan yang merugikan.

Baris 13 kode merepresentasikan apabila *move* dari bot membentuk kotak dengan seluruh sisi membentuk garis. Maka, nilai *total objective function* ditambah sebanyak 10 untuk menunjukkan gerakan yang menguntungkan. Pada baris 15 kode, gerakan bot menghasilkan sel kotak dengan hanya 2 garis. Karena gerakan tersebut tidak merugikan, mungkin memaksa lawan membuat sel kotak dengan 3 sisi dan memberikan kesempatan untuk bot membuat kotak, maka gerakan bot tersebut bernilai positif sebanyak 5 untuk total *objective function*.

Setelah seluruh kemungkinan 1 *move* kedepan sudah dilakukan iterasi, maka *objective function* akan mengembalikan nilai total hasil nilai dari iterasi tersebut.

# Proses Pencarian *Minimax* dengan *Alpha Beta Pruning*

Algoritma minimax dengan *alpha beta pruning* merupakan salah satu algoritma yang akan melakukan pencarian dengan mencoba seluruh kemungkinan yang ada seperti DFS (*depth first search*). Untuk setiap gerakan yang dilakukan akan dilakukan ekspansi ke bawah terlebih dahulu hingga mencapai terminal state untuk kemudian ditentukan nilainya dengan *utility function* yang dimiliki. Namun pada algoritma yang digunakan untuk tugas besar kali ini akan diterapkan *depth limit* agar pencarian tidak memakan waktu yang sangat lama sebelum mendapatkan nilai dari *utility function*. *Depth limit* ini akan bertambah dalam seiring waktu dengan asumsi pilih *actions* yang bisa diambil agen semakin sedikit sehingga pohon ruang status yang terbentuk pun akan semakin sedikit.

Selain itu, pencarian juga akan dihentikan setelah 5 detik jika algoritma belum selesai, agen mengambil *action* yang terbaik pada saat itu. Kemudian, untuk *action-action* yang memiliki nilai utilitas yang sama, akan diterapkan heuristik dalam pemilihannya, bukan memilih secara *random*. Heuristik yang digunakan adalah dengan mengecek kondisi kotak setelah *action* tersebut diambil apakah aman untuk diambil, menguntung jika diambil, atau tidak memiliki pengaruh. Untuk kondisi hanya ada *action* yang tidak memiliki pengaruh, agen terpaksa memilih aksi-aksi tersebut secara *random*.

# Algoritma *Local Search* yang Digunakan

Algoritma yang digunakan untuk bot dengan *local* adalah algoritma *Hill-Climbing with Sideways Move*. Dengan algoritma tersebut, bot akan membangkitkan *neighbor state* sejauh 1 langkah ke depan dari *current state*. Kemudian, bot akan menilai seberapa baik setiap *neighbor state* yang dibangkitkan dengan menggunakan *objective function* yang telah dijelaskan sebelumnya. Kemudian, bot memilih *neighbor state* dengan *value* tertinggi ketika pembangkitan selesai dilakukan. Jika pembangkitan *neighbor state* menghasilkan *value* yang lebih rendah dari *current state* atau sudah melebihi dari 5 detik, maka pencarian berhenti.

*Local search* yang dipilih adalah Hill-Climbing with Sideways Move karena algoritma tersebut akan selalu mencoba bergerak ke state dengan value yang lebih baik daripada *current state*. Algoritma lain memiliki kemungkinan melakukan *random movement* atau tidak membangkitkan seluruh kemungkinan *neighbor state* seperti *Stochastic Hill-Climbing*. Dalam permainan *Dots and Boxes*, satu pergerakan yang salah akan sangat berpengaruh ke hasil pertandingan. Oleh karena itu, algoritma *Hill-Climbing with Sideways Move* cocok untuk permainan *Dots and Boxes* karena tidak mencoba melakukan pergerakan yang *random* dibanding algoritma *local search* lainnya.

# Hasil Pertandingan

Pada bagian ini akan ditunjukkan *screenshot* dan hasil dari pertandingan *Dots and Boxes* yang dilakukan. Pertandingan akan dilakukan sebanyak 5 kali dengan urutan player 1 dan player 2 sesuai dengan judul pertandingan.

## Bot *Minimax* vs Manusia

### *Screenshot* Hasil Pertandingan

Berikut merupakan hasil pertandingan yang dilakukan antara bot dengan algoritma *minimax* melawan manusia.

|  |
| --- |
|  |
|  |
|  |
|  |

### Penjelasan Hasil Pertandingan

Berdasarkan pertandingan antara bot *minimax* melawan player yang dilakukan sebanyak 5 kali, terlihat bahwa bot *minimax* belum dapat selalu memenangkan pertandingan melawan manusia. Nilai *utility function* pada bot minimax bukan untuk *state* akhir, tetapi hanya sampai kedalaman 5 yang terus bertambah seiring waktu. Hal tersebut dikarenakan untuk memenuhi batas waktu 5 detik untuk pengambilan langkah. Oleh karena itu, bot *minimax* dengan batasan seperti ini tidak dapat selalu mengungguli manusia secara keseluruhan.

Berikut merupakan jumlah kemenangan, kekalahan, serta persentase kemenangan bot:

* Jumlah kemenangan bot *minimax*: 2
* Jumlah kekalahan bot *minimax*: 3
* Persentase kemenangan bot *minimax*: 40%

## Bot *Local Search* vs Manusia

### *Screenshot* Hasil Pertandingan

Berikut merupakan hasil pertandingan yang dilakukan antara bot dengan algoritma *local search* melawan manusia.

|  |
| --- |
|  |
|  |
|  |
|  |

### Penjelasan Hasil Pertandingan

Berdasarkan hasil bot *local search* melawan manusia, terlihat juga bahwa bot *local search* belum dapat mengungguli pertandingan melawan manusia. Pengambilan langkah oleh bot mencoba mencari nilai *objective function* optimal untuk *neighbor state*, bukan keseluruhan pertandingan. Namun, terdapat batasan pencarian berhenti ketika sudah melebihi 5 detik atau tidak menemukan nilai *objective function neighbor* yang melebihi atau setara dengan nilai *objective function current state*.

Berikut merupakan jumlah kemenangan, kekalahan, serta persentase kemenangan bot:

* Jumlah kemenangan bot *local search*: 1
* Jumlah kekalahan bot *local search*: 4
* Persentase kemenangan bot *local search*: 20%

## Bot *Minimax* vs Bot *Local Search*

### *Screenshot* Hasil Pertandingan

Berikut merupakan hasil pertandingan yang dilakukan antara bot dengan algoritma *minimax* melawan bot dengan algoritma *local search*.

|  |
| --- |
|  |
|  |
|  |
|  |

### Penjelasan Hasil Pertandingan

Hasil pertandingan bot *minimax* melawan bot *local search* menunjukkan bahwa performa bot *minimax* lebih unggul sedikit dari bot *local search* atau bahkan setara dengan *bot local* search. Hal ini disebabkan karena bot minimax tidak bisa melihat utility function dengan kedalaman hingga state akhir, tetapi hanya sampai kedalaman sebanyak 5 yang terus bertambah seiring waktu dan harus memenuhi batas waktu 5 detik. Sedangkan bot *local search* mencari move terbaik untuk *neighbor state* yang dapat dibangkitkan dari *current state*. Secara umum, kedua bot tersebut memiliki performa yang sama. Hal ini mungkin bisa berbeda cerita apabila bot minimax bisa melakukan pencarian hingga *state* paling akhir, akan tetapi hal tersebut sangat tidak *feasible* akibat banyaknya pohon ruang status yang harus diambil.

Berikut merupakan jumlah kemenangan, kekalahan, serta persentase kemenangan masing-masing bot:

* Jumlah kemenangan bot *minimax*: 3
* Jumlah kekalahan bot *minimax*: 2
* Persentasi kemenangan bot *minimax*: 60%
* Jumlah kemenangan bot *local search*: 2
* Jumlah kekalahan bot *local search*: 3
* Persentase kemenangan bot *local search*: 40%

# Saran Perbaikan Kode Program

Pada *bot* yang menggunakan *local search*, diperlukan fungsi objektif yang lebih memperhitungkan strategi secara keseluruhan, termasuk kondisi lawan saat itu. Selain itu, karena *bot* ini hanya memeriksa kemungkinan state untuk satu langkah ke depan, diperlukan perbaikan agar *bot* bisa menelusuri seluruh kemungkinan lebih dari satu langkah kedepan. Pada *bot* yang menggunakan *minimax*, diperlukan fungsi utility yang dapat menggambarkan atau memperhitungkan kondisi *long chain* (kotak yang bisa menghasilkan banyak poin) yang dihubungkan dengan giliran *bot* bermain.

# Kontribusi Setiap Anggota Kelompok

| **NIM / Nama** | **Kontribusi** |
| --- | --- |
| 13520028 / Timothy Stanley Setiawan | Implementasi bot minimax |
| 13520091 / Andreas Indra Kurniawan | Implementasi bot minimax |
| 13520127 / Adzka Ahmadetya Zaidan | Laporan |
| 13520157 / Thirafi Najwan Kurniatama | Implementasi bot local search |