Laporan Tugas Besar 1
Inteligensi Buatan (IF3170)
Minimax Algorithm and Alpha Beta Pruning in Halma

Disusun oleh:
Muhammad Cisco Zulfikar - 13518073
Faris Muhammad Kautsar - 13518105
Gregorius Jovan Kresnadi - 13518135
Muhammad Fauzan Rafi Sidiq Widjonarto - 13518147

Laporan yang dibuat untuk memenuhi tugas mata kuliah IF3170

TAHUN PELAJARAN 2020/2021

DAFTAR ISI

BAB I
TAHAPAN PENGERJAAN

Aplikasi dikembangkan dengan basis web. Tahap pengerjaan yang dilakukan adalah sebagai berikut:

1. Membuat tampilan utama (menu)
   Diselesaikan oleh Cisco Zulfikar
2. Membuat tampilan permainan
   Diselesaikan oleh Gregorius Jovan Kresnadi
3. Membuat modul representasi papan halma
   Diselesaikan oleh Gregorius Jovan Kresnadi, Fauzan Rafi Sidiq
4. Membuat modul representasi lokasi bidak
   Diselesaikan oleh M. Fauzan Rafi Sidiq W
5. Membuat modul representasi aksi yang dilakukan
   Diselesaikan oleh M. Fauzan Rafi Sidiq W
6. Membuat modul timer
   Diselesaikan oleh Gregorius Jovan Kresnadi, Fauzan Rafi Sidiq
7. Membuat modul yang mengontrol *flow* jalannya permainan
   Diselesaikan oleh Gregorius Jovan Kresnadi, M. Fauzan Rafi Sidiq W, Faris M Kautsar
8. Membuat modul bot yang mengimplementasikan algoritma minimax alpha beta pruning
   Diselesaikan oleh Faris Muhammad Kautsar, Fauzan Rafi Sidiq
9. Membuat modul bot yang mengimplementasikan algoritma local search dan minimax alpha beta pruning
   Diselesaikan oleh Faris Muhammad Kautsar, Fauzan Rafi Sidiq

BAB II
REPRESENTASI DALAM PERMAINAN

Dalam permainan, dilakukan beberapa pemodelan yang relevan untuk menjalankan permainan serta melakukan search, antara lain adalah:

1. State

   State dimodelkan di dalam objek HalmaBoard, yaitu objek yang merepresentasikan keadaan papan sekarang. Objek HalmaBoard menyimpan setiap lokasi bidak yang ada di suatu atribut matriks state (array of array). Besar ukuran matriks menyesuaikan masukan dari masukan pengguna. Setiap sel dengan spesifikasi absis dan ordinat x dan y berisi keterangan sel tersebut, yaitu berisi kode pemain apabila diisi oleh bidak pemain (P untuk pemain manusia, K atau C untuk AI), dan tanda strip (-) apabila sel tidak berisi apa-apa.
2. Giliran

   Giliran disimpan di variabel turn pada modul untuk mengontrol alur permainan. Setiap mulainya suatu giliran, maka akan dinyalakan timer untuk mengukur seberapa lama giliran yang dibolehkan. Bila timer sudah habis, maka modul akan otomatis mengganti giliran ke giliran pemain selanjutnya.
3. Pilihan aksi yang legal

   Aksi yang legal dicek melalui metode di kelas Action yang menerima argumen HalmaBoard, mengembalikan apakah aksi tersebut legal di state papan tersebut. Namun, kelas Action tersebut hanya mengecek pergerakan ke satu kotak kosong atau satu kali lompatan. Untuk lompatan berkali-kali yang divalidasi sebagai aksi legal, digunakan pengecekan kembali pada metode getAllHops di kelas HalmaAI untuk AI dan pada saat rendering aksi untuk player manusia.
4. Perhitungan nilai aksi yang dibuat (fungsi objektif)

   Fungsi objektif dijadikan suatu metode di objek HalmaBoard, sehingga untuk suatu state yang disimpan oleh objek HalmaBoard bisa dihitung langsung nilai fungsi objektfnya melalui invokasi metode ini. Metode yang dipanggil langsung memproses internal state yang direpresentasikan (penjelasan di nomor 1) yang

BAB III
PENJELASAN ALGORITMA KESELURUHAN

Secara keseluruhan, algoritma dari aplikasi secara keseluruhan adalah sebagai berikut:

1. Menerima masukan pilihan-pilihan dari pengaturan permainan seperti warna bidak, ukuran papan, jenis pemain 1 dan 2, dan lama setiap giliran dari pengguna.
2. Menampilkan page sesuai dengan pengaturan, serta menampilkan tombol start.
3. Bila tombol start sudah ditekan, maka permainan akan dimulai.
4. Modul pengaturan permainan menginisialisasikan objek representasi seperti AI dan papan halma sesuai pengaturan dan menyalakan timer untuk giliran pertama
5. Bila timer belum habis, aplikasi menerima masukan tergantung dari giliran pemain: apabila pemain adalah manusia maka aplikasi akan menerima masukan aksi yang diberikan serta mengecek legalitas dari aksi tersebut. Apabila aksi tersebut legal, maka aksi tersebut akan dipakai untuk memperbarui state dari papan. Apabila pemain adalah AI, maka AI akan melakukan minimax alpha beta search (+ local search, tergantung dengan jenis AI) dan mengeluarkan aksi yang sesuai untuk selanjutnya diberikan untuk memperbarui state permainan.
6. Bila timer sudah habis, maka modul akan mengecek apakah state dalam keadaan final. Bila belum, melakukan pergantian giliran dan ulang dari langkah 5. Bila final, lanjut ke langkah 7
7. Permainan akan berakhir dan aplikasi menampilkan pemenangnya di page

Berikut merupakan pseudocode dari algoritma keseluruhan:

```
procedure aplikasi():
var
mode     : string
bsize    : integer
pcolor   : string
tlimit   : integer
action   : Action

algoritma
renderMainPage()
mode, bsize, tlimit, pcolor <- getGameParameters()
renderGamePage(pcolor)
initGame(mode, bsize, tlimit)
while notFinalState() do
       timerStart()
       while timerNotFinished():
       if player is human then
              action <- getAction()
       else
              if type is minimax + localsearch then
              action <- miniMaxLocalSearch()
              else
              action <- miniMax()
       timerForceEnd()
       updateBoard(action)
       checkFinalState()
       nextTurn()

renderWinPage()
```

BAB IV
PROSES PENCARIAN MINIMAX DAN ALPHA BETA PRUNING

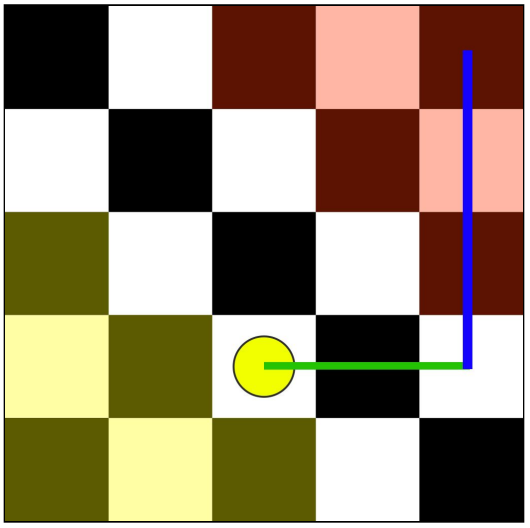
Proses pencarian minimax dilakukan di dalam *class* HalmaAI. Terdapat beberapa fungsi pembantu yang bertujuan mempermudah implementasi algoritma minimax serta local search. Fungsi-fungsi tersebut adalah sebagai berikut:

```
getAllActions(coord, board, player)
//method mengembalikan array berisi semua aksi yang dapat dihasilkan dari
suatu pion di suatu koordinat. Method ini memanggil getAllHops.

getAllHops(origin, beforejump, board, coordlist, player)
//method mengembalikan array berisi semua aksi berupa lompatan yang dapat
dihasilkan dari suatu pion di suatu koordinat.
```

Berikutnya adalah merancang *objective function* yang dibuat untuk memaksimalkan peluang kemenangan dalam aksi yang dipilih oleh algoritma. Fungsi objektif yang dibuat memiliki pertimbangan (aspek) sebagai berikut:

1. Jumlah jarak antara ujung rumah musuh dengan setiap bidak pemain
2. Jumlah jarak antara ujung rumah pemain dengan setiap bidak musuh
3. Jumlah jarak antara ujung rumah pemain dengan setiap bidak pemain
4. Jumlah jarak antara ujung rumah musuh dengan setiap bidak musuh

Dalam memainkan permainan, maka setiap bidak haruslah mendekat ke ujung rumah musuh dan menjauh dari ujung rumah sendiri. Dari observasi tersebut, maka fungsi objektif harus makin besar apabila semakin banyak bidak yang mendekat ke rumah musuh dan semakin jauh bidak musuh ke rumah sendiri. Sebaliknya, fungsi objektif harus makin kecil apabila semakin dekat bidak musuh ke rumah sendiri dan semakin dekat bidak sendiri ke rumah sendiri. Jarak yang dihitung merupakan jarak manhattan dari dua titik di satu koordinat.



Gambar 4.1 | Ilustrasi perhitungan jarak sebuah bidak ke ujung rumah lawan

Hal kedua yang menjadi observasi adalah makin cepat setiap bidak meninggalkan rumah sendiri, maka makin dekat ke state final. Maka, setiap aksi yang membuat bidak keluar dari rumahnya, maka fungsi objektif akan meningkat. Dan karena tujuan permainan adalah untuk mencapai rumah musuh, maka aksi yang membuat bidak masuk ke rumah lawan juga akan meningkatkan fungsi objektif. Kedua hal ini bisa dimetaforakan sebagai sebuah "motif" bagi AI untuk segera membuat bidak-bidaknya meninggalkan rumahnya dan memasuki rumah musuh. Koefisien peningkatan dari fungsi objektif didefinisikan secara heuristik, dan bernilai besar untuk mempercepat pergerakan bidak. Dalam kasus ini, koefisien yang dipilih adalah 30.

Maka, dirumuskan sebuah fungsi objektif yang memenuhi pertimbangan dan observasi di atas sebagai berikut:

```
fungsi objektif =
jumlah jarak bidak pemain ke rumah pemain - jumlah jarak bidak
pemain ke ujung rumah musuh + 30 * jumlah bidak pemain yang ada
di rumah musuh + jumlah jarak bidak pemain ke ujung rumah pemain
- 30 * jumlah bidak pemain yang ada di rumah pemain
```

Setelah fungsi-fungsi tersebut selesai diimplementasikan, maka masuk ke proses utama: algoritma minimax dengan alpha-beta pruning itu sendiri. Algoritma minimax dimulai dengan generasi seluruh aksi yang dapat dilakukan untuk suatu *state*. Kemudian dipilih aksi yang paling mungkin diimplementasikan berdasarkan tindakan rasional dari AI dan lawan - AI akan memaksimalkan fungsi objektif, sementara lawan akan meminimalisir fungsi tersebut. Prediksi langkah berlanjut hingga *depth* tertentu dimana suatu nilai diperoleh, kemudian di-*pass* ke atas sehingga diperoleh nilai-nilai untuk tiap aksi. Aksi yang dipilih adalah aksi yang menerima nilai tertinggi.

Dalam implementasinya, algoritma ini juga mengalami *pruning* berdasarkan nilai *alpha* dan *beta*. *Pruning* bertujuan menghentikan evaluasi suatu aksi apabila aksi tersebut terbukti lebih "buruk" dan dipastikan tidak akan dipilih dibandingkan aksi yang telah dievaluasi sebelumnya.

Berikut adalah rumusan ringkas mengenai implementasi algoritma minimax.

```
alphaBeta(depth)
//"memulai" algoritma dan menjadi bagian teratas dalam search
tree. Mengembalikan aksi.

maxValue(state, alpha, beta, depth)
//mencari aksi yang akan dipilih oleh player yang maximizing,
dalam kasus ini AI itu sendiri. Mengembalikan nilai integer.

minValue(state, alpha, beta, depth)
//mencari aksi yang akan dipilih oleh player yang minimizing,
dalam kasus ini lawan dari AI (bisa manusia maupun AI lainnya).
Mengembalikan nilai integer.
```

BAB V
PROSES PENCARIAN LOCAL SEARCH DAN KOMBINASINYA DENGAN MINIMAX

Proses pencarian local search yang dikombinasikan dengan minimax mirip dengan minimax biasa. Namun perbedaan yang mendasar dari minimax biasa terletak di pembangkitan state kemungkinan dalam pohon pencarian. Di local search, dipilih satu bidak secara acak untuk selanjutnya dibangkitkan semua kemungkinan gerakannya, berbeda dengan minimax biasa yang membangkitkan seluruh kemungkinan dari semua gerakan bidak yang ada. Hal ini terus diulangi hingga tinggi maksimum pohon tercapai atau menemukan state final.

Fungsi objektif yang digunakan dan fungsi-fungsi pembantu sama dengan pada algoritma minimax normal. Perumusan algoritma minimax+local search adalah sebagai berikut:

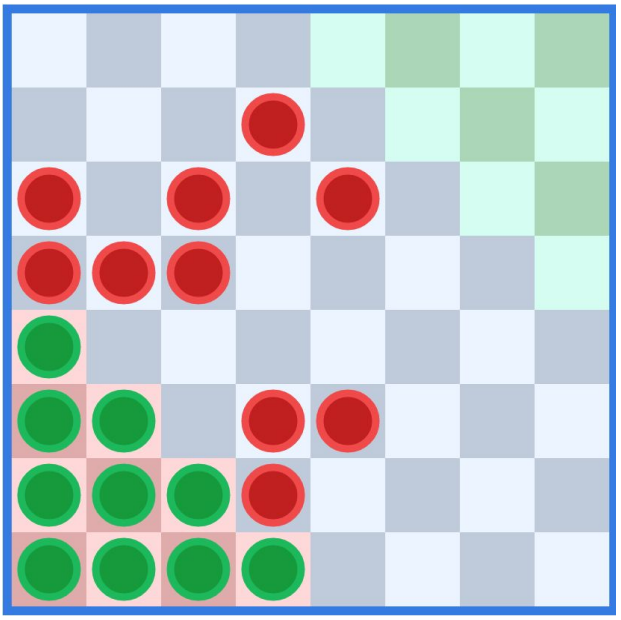
```
localSearch(depth)
//”memulai” algoritma dan menjadi bagian teratas dalam search
tree. Mengembalikan aksi. Sama dengan minimax, namun aksi-aksi
diperoleh dari pion random.

maxValueLS(state, alpha, beta, depth)
//mencari aksi yang akan dipilih oleh player yang maximizing,
dalam kasus ini AI itu sendiri. Mengembalikan nilai integer.

minValueLS(state, alpha, beta, depth)
//mencari aksi yang akan dipilih oleh player yang minimizing,
dalam kasus ini lawan dari AI (bisa manusia maupun AI lainnya).
Mengembalikan nilai integer.
```

BAB VI
ANALISIS HASIL PERTANDINGAN

Analisis pertandingan yang dilakukan menggunakan papan 8x8. Kedua bot diimplementasikan dengan maksimum tinggi pohon = 3 untuk pembangkitan ruang pencarian. Berikut merupakan hasil pertandingan dari Halma:

1. Pemain manusia vs Bot Minimax AlphaBeta Pruning

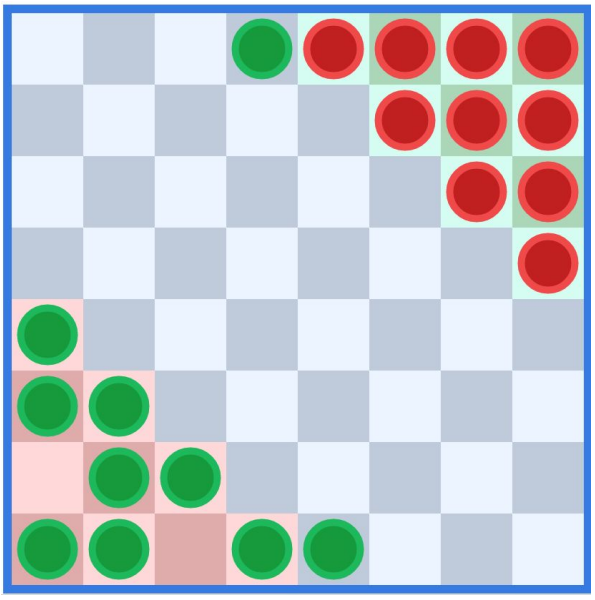
No	Time elapsed (s)	Mean time to think per turn (s)	Status bot
1	2.394	0.07041176470588235	Menang
2	3.1069999999999998	0.09709374999999999	Menang
3	3.153999999999999	0.047074626865671626	Kalah
4	2.0029999999999992	0.06069696969696967	Menang
5	2.293999999999999	0.06951515151515149	Menang
6	2.8650000000000007	0.05730000000000001	Menang
7	2.1099999999999994	0.07535714285714283	Menang
8	1.6339999999999997	0.060518518518518506	Menang
9	1.84	0.08	Menang
10	1.9249999999999998	0.07129629629629629	Menang



Gambar 6.1
Hasil akhir dari pertandingan no.10 manusia (merah) vs bot Minimax (hijau)

2. Pemain manusia vs Bot Minimax + Local Search

No	Time elapsed (s)	Mean time to think per turn (s)	Status bot
1	0.1680000000000001	0.0031698113207547186	Kalah
2	0.15900000000000009	0.002741379310344829	Kalah
3	0.1870000000000001	0.003462962962962965	Kalah
4	0.04400000000000002	0.0008979591836734698	Kalah
5	0.14800000000000008	0.0022424242424242437	Kalah
6	0.04500000000000001	0.001363636363636364	Kalah
7	0.04500000000000001	0.0012162162162162166	Kalah
8	0.047000000000000014	0.0010681818181818186	Kalah
9	0.048000000000000015	0.0014545454545454549	Kalah
10	0.047000000000000014	0.0016206896551724142	Kalah



Gambar 6.2
Hasil akhir dari pertandingan no.5 manusia (merah) vs bot Minimax + Local Search (hijau)

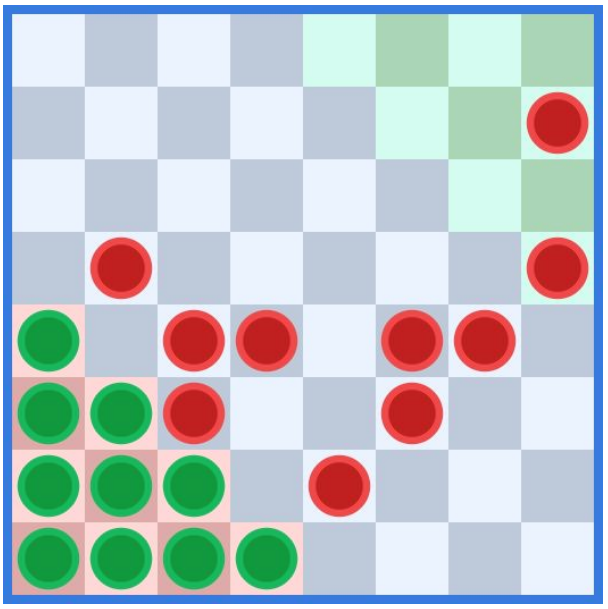
3. Bot Minimax vs Bot Minimax + Local Search
    a. Bot Minimax

No	Time elapsed (s)	Mean time to think per turn (s)	Status bot
1	2.662	0.09859259259259259	Menang

2	3.255999999999997	0.05814285714285709	Menang
3	3.1860000000000004	0.0838421052631579	Menang
4	2.9039999999999995	0.07642105263157893	Menang
5	2.038999999999998	0.033983333333333296	Menang
6	2.3089999999999984	0.014000000000000005	Menang
7	1.9769999999999972	0.030890624999999956	Menang
8	1.993	0.07972	Menang
9	3.7029999999999954	0.025715277777777747	Menang
10	3.003999999999999	0.06258333333333331	Menang

b. Bot Minimax + Local Search

No	**Time elapsed (s)**	**Mean time to think per turn (s)**	**Status bot**
1	0.031000000000000014	0.0011923076923076928	Kalah
2	0.040000000000000015	0.0007272727272727276	Kalah
3	0.05000000000000002	0.0013513513513513519	Kalah
4	0.08300000000000002	0.002243243243243244	Kalah
5	0.011000000000000003	0.00018644067796610175	Kalah
6	0.014000000000000005	0.00019718309859154937	Kalah
7	0.007	0.00011111111111111112	Kalah
8	0.009000000000000001	0.00037500000000000006	Kalah
9	0.025000000000000015	0.00017482517482517493	Kalah
10	0.019000000000000003	0.00040425531914893624	Kalah



Gambar 6.3
Hasil akhir dari pertandingan no.6 bot Minimax (hijau) vs bot Minimax + Local Search (merah)

Informasi yang didapat dari setiap pertandingan yang telah dilakukan adalah sebagai berikut:

1. Bot yang menggunakan algoritma minimax memiliki tingkat kemenangan yang sangat tinggi dibandingkan dengan bot yang menggunakan algoritma minimax + local search
2. Waktu yang dibutuhkan oleh bot dengan minimax biasa 10 kali lebih lambat dibandingkan waktu yang dibutuhkan oleh bot minimax dengan local search.

Informasi pertama diperoleh karena mekanisme dari algoritma local search dan minimax biasa: minimax biasa membangkitkan seluruh kemungkinan state dari satu *current state*, sedangkan algoritma minimax + local search hanya membangkitkan kemungkinan satu bidak saja per iterasi. Hal ini membuat solusi local search lebih lambat untuk menemukan langkah yang paling optimal di satu *current state* dibandingkan dengan minimax biasa.

Informasi kedua diperoleh juga karena mekanisme dari kedua algoritma. Minimax biasa membangkitkan seluruh kemungkinan yang jumlahnya besar, sehingga perlu waktu yang lama untuk membangkitkan ruang pencarian. Sedangkan local search hanya membangkitkan kemungkinan gerakan satu bidak saja, sehingga pembangkitan ruang pencarian jauh lebih sedikit dibandingkan dengan minimax biasa.