Perancangan Lawan Bermain Permainan Tic-tac-toe dengan Menggunakan Algoritma *Branch* and *Bound*

Andri Hardono Hutama
Teknik Informatika, Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
Bandung, Indonesia
13514031@std.stei.itb.ac.id

***Abstract*—Tic-tac-toe merupakan salah satu contoh *Zero-sum* Game dengan Informasi Sempurna. Pada permainan ini setiap pemain secara teori dapat menentukan langkah paling optimal pada setiap giliran. Dengan memanfaatkan algoritma yang berdasarkan pada *Branch and Bound* dapat dibuat lawan bermain yang diharapkan mampu memilih langkah yang cukup optimal.**

Keywords—Batasan, Branch and Bound, Constraint, Cost, Fungsi, Tic-tac-toe, Zero-sum Game Informasi Sempurna

I. PENDAHULUAN

Dari waktu ke waktu, manusia senantias menciptakian berbagai macam karya. Salah satu bentuk dari karya manusia tersebut adalah permainan. Permainan – permainan ini memiliki berbagai variasi, termasuk dari jumlah pemainnya. Tidak jarang permainan mengadu pemikiran, penalaran, dan juga strategi dari pihak yang bermain untuk menentukan pemenangnya. Beberapa contoh permainan yang mengadu dua pemain misalnya adalah Catur, Tic-tac-toe, dan SOS.

Salah satu permainan yang paling sederhana adalah Tic-tac-toe. Tic-tac-toe sendiri adalah sebuah permainan dimana seorang pemain menggambarkan X dan pemain lainnya menggambarkan O ke dalam suatu bidang dengan tujuan memenuhi sebuah baris dengan X atau O [1]. Bidang tersebut pada umumnya adalah sebuah persegi yang tersusun dari 9 buaah persegi lebih kecl (panjang 3 persegi dan lebar 3 persegi). Tentunya bidang ini dapat diperluas. Namun, permainan dengan ukuran bidang lebih dari 3 x 3 tersebut kerap memiliki hasil seri. Baris yang dibentuk dalam hal ini bisa berupa sebuah baris (horisontal), sebuah kolom (vertikal), ataupun menyilang (diagonal). Jika seorang pemain berhasil mengisi sebauh baris dengan objek pengisiannya, maka permainan Tic-tac-toe ini dinyatakan selesai dengan kemenangan pemain tersebut.

Tic-tac-toe sendiri termasuk ke dalam jenis *zero sum game*. *Zero sum game* adalah jenis permainan di mana jika seorang pemain menang maka pemain lainnya dinyatakan kalah (dalam kasus *zero sum game* untuk dua orang) [2]. Selain itu *zero sum game* sendiri memiliki dua tipe yang umum, yaitu *zero sum game* dengan informasi sempurna dan *zero sum game* dengan informasi tak sempurna [2].

Tic-tac-toe sendiri termasuk dalam *zero sum game* dengan informasi sempurna. Maksud dari informasi sempurna adalah setiap pemain mengetahui hasil dari semua pergerakan sebelumnya [2]. Ini berarti seorang pemain dapat mengetahui semua informasi mengenai kondisi saat ini. Dari kondisi tersebut seorang pemain secara teori Seharusnya dapat menentukan strategi terbaik untuk permainan tersebut. Namun misalnya untuk contoh permainan Catur, jumlah kemungkinan yang ada terlalu banyak sehingga dengan bantuan komputerpun sulit menentuka strategi terbaik tersebut [2].

Dalam permainan Tic-tac-toe jika kedua pemain bermain dengan strategi terbaik, maka hasil yang didapatkan adalah selalu seri. Strategi terbaik ini selalu menjamin seorang pemain tidak akan kalah namun tidak menjamin ia akan menang.

Untuk Tic-tac-toe tanpa modifikasi salah satu model yang diketahui saat ini adalah dengan melakukan langkah pertama yang mungkin dari daftar perintah berikut:

- Jika memungkinkan untuk memanangkan permaian maka menangkan permainan tersebut.
- Jika memungkinan melakukan *block* maka lakukan *block*. *Block* adalah mencegah lawan memenangkan permainan.
- Jika memungkinan melakukan *fork* maka lakukan *fork*. *Fork* adalah langkah yang memungkinkan dua cara memenangkan permainan pada langkah berikutnya. Langkah ini akan membuat lawan tidak dapat melakukan *block*.
- Jika memungkinan melakukan *block fork* maka lakukan *block fork*. *Block fork* adalah langkah melakukan untuk mencegah lawan melakukan *fork* pada langkah berikutnya.
- Jika memungkinkan mengisi bagian tengah maka isi pada bagian tengah.
- Jika memungkinkan mengisi pojok berlawanan maka isi pada pojok berlawanan. Jika suatu pojok diisi oleh lawan, maka isikan pojok yang berlawanan dengan sisi tersebut.

- Jika memungkinkan mengisi pojok maka isi pada pojok.
- Jika memungkinkan mengisi bagian sisi maka isi pada bagian sisi.

II. DASAR TEORI

Branch and Bound adalah salah satu algortima yang memanfaatkan pohon ruang status untuk menyelesaikan masalah. Penyelesaian masalah dengan menggunakan algoritma *Branch and Bound* akan mengandalkan sebuah fungsi objektif yang akan dimaksimalkan atau diminimalkan sesuai dengan permasalahan tersebut [4]. Namun, pemaksimalan ataupun peminimalan tersebut tidak boleh melanggar batasan (*constraint*) dari persoalan. Algoritma *Branch and Bound* memiliki pembangkitan yang sekilas mirip dengan *BFS (Breadth First Search*) namun memiliki cara penentuan simpul mana yang harus di-*expand* yang berbeda.

Setiap simpul dalam pohon ruang status akan diberikan sebuah nilai *cost* [4]. Berbeda dengan algortima *BFS* yang penentuan simpul yang akan di-*expand* berdasarkan urutan pembangkitannya (*First In First Out*), algoritma *Branch and Bound* ini mengandalakan *cost* tersebut untuk menentukan simpul mana yang akan di-*expand*. Dalam kasus pencarian minimal tentunya akan diambil simpul dengan *cost* terkecil dan dalam kasus pencarian maksimal tentunya akan diambil simpul dengan *cost* terbesar.

Algoritma *Branch and Bound* ini dapat dikatakan melakukan pemangkan pada pada jalur yang tidak lagi mengarah ke solusi [4]. Pemangkasan tersebut dilakukan jika terjadi pelanggaran batasan tertentu sehingga tidak mengarah ke solusi yang mungkin. Misalnya saja pada *knapsack problem* suatu simpul memiliki total beban melebihi kapasistas yang mampu diangkut. Tentunya kita tidak perlu lagi memerika simpul seperti ini dan anak – anaknya karena tidak mungkin terdapat solusi. Karena itulah kita lakukan pemangkasan pada simpul tersebut.

Jika setelah menelusuri anak dari suatu upapohon ternyata *cost* dari simpul – simpul yang dibangkitkan tidak ada yang minimum dalam keseluruhan pohon, maka ada beberapa pendekatan. Idealnya tentunya kita akan mengambil simpul dengan *cost* minimum tanpa memedulikan tingkatnya. Hal ini berdasarkan pemikiran bahwa simpul tersebut memiliki kemungkinan menghasilkan solusi optimal juga. Jadi idealnya kita akan mengambil simpul dengan nilai terendah dimanapun letaknya. Pendekatan lainnya adalah kasus yang menganggap pilihan untuk tingkat sebelumnya sudah merupakan langkah terbaik dan kita tidak dapat kembali ke tingkat sebelumnya. Hasil pendekatan tersebut akan menghasilkan algoritma yang mirip dengan *Greedy Best First Search*.

Mirip dengan hal itu, terdapat juga berbagai pendekatan untuk kasus setelah solusi pertama ditemukan. Pendekatan paling ideal tentunya adalah kita harus memeriksa apakah ada simpul yang masih hidup. Jika terdapat simpul yang masih hidup, kita harus membangkitakan anak – anak dari simpul tersebut untuk memeriksa apakah terdapat solusi yang lebih optimal dari simpul tersebut. Pendekatan lainnya adalah langsung mengambil solusi utama yang kita temukan tersebut.

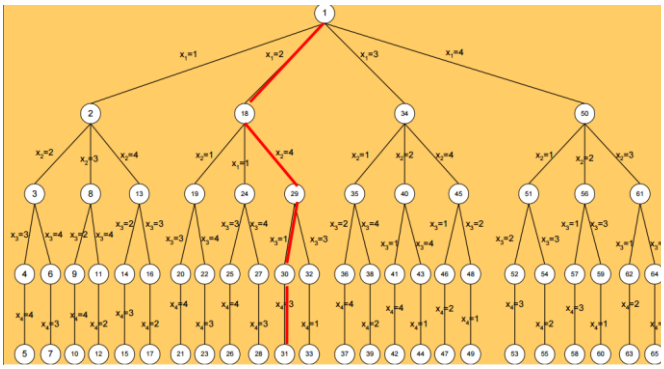
Dari berbagai pendekatan yang diceritakant ersebut, pada dasarnya algoritma *Branch and Bound* hanya memberikan garis besar yang membantu alur berpikir. Pada akhirnya algoritma yang kita terapkan tentunya akan disesuaikan dengan permasalahn yang dihadapi.

Algoritma umum *Branch and Bound* secara umum untuk kasus minimalisasi adalah sebagai berikut: [3]

- Masukan simpul akar ke dalam antrian. Periksalah apakah simpul akan tersebut adalah solusi. Jika merupakan solusi maka hentikan pencarian. Jika tidak lanjut ke langkah berikutnya.
- Jika antrian kosong, dalam hal ini berarti semua simpul dalam pohon ruang status melanggar batasan yang diberikan, berarti permasalah ini tidak memiliki solusi. Jika hal ini terjadi tentunya pencarian akan dihentikan.
- Jika antrian tidak kosong, pilihlah dari antrian simpul dengan *cost* paling rendah. Jika terdapat dua atau lebih simpul dengan *cost* paling rendah maka lakukan pemilihan secara sembarang. Sebut simpul yang dipilih tersebut dengan simpul i.
- Jika simpul i tersebut merupakan simpul solusi maka hentikan pencarian.
- Jika simpul i tersebut bukanlah simpul solusi maka bangkitkan anak – anak dari simpul i tersebut, hitung *cost* dari setiap anak tersebut, dan masukan ke dalam antrian. Setalah dimasukan ke dalam antrian maka kembali ke langkah ke-2 (pengecekan kasus antrian kosong).
- Jika simpul i tidak memiliki anak, maka kembali ke langkah ke-2 (pengecekan kasus antrian kosong).

Melihat algoritma umum yang diberikan dari sumber tersebut dan membandingkannya dengan penjelasan sebelumnya, dapat dikatakan algoritma tersebut menggunakan pendekatan untuk menganggap solusi utama sudah merupakan solusi yang baik dan juga pendekatan boleh melakukan perpindahan upapohon jiak simpul dengan *cost* terkecil berada pada upapohon lain. Jika diinginkan maka bisa saja algoritma tersebut dianggap sebagai standard baku algoritma *Branch and Bound* sedangkan pendekatan yang dijelaskan sebelumnya adalah variasi dari algoritma *Branch and Bound*.

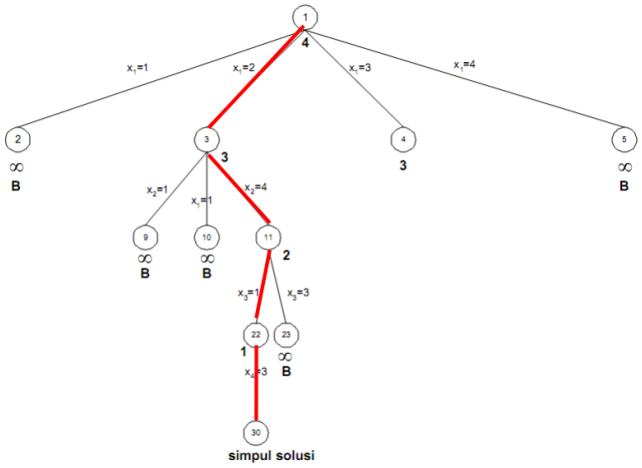
Mari kita lihat contoh penggunaan algoritma *Branch and Bound* untuk *N-Queens Problem* [4]. *N-Queens Problem* adalah masalah yang berdasarkan permainan Catur. Penyelesaian masalah ini adalh dengan mencari penempatan ratu di dalam papan permaian berukuran tertentu sehingga tidak terdapat dua ratu atau lebih yang terdapat dalam baris, kolom, ataupun diagonal yang sama. Jika kita membangkitkan semua pohon ruang status dari permasalah ini maka akan didapatkan pohon seperti pada gambar berikut.



Gambar 1 Pohon Ruang Status Lengkap untuk N-Queens Problem [4]

Terlihat dari gambar tersebut dibutuhkan pembangkitan 65 buah simpul yang kemudian ditelusuri. Tentunya pembangkitan secara dinamis akan lebih efektif. Sedangkan dengan menggunakan algoritma *Backtracking* hanya diperlukan pembangkitan 16 simpul.

Strategi *Branch and Bound* untuk *N-Queens Problem* kali ini akan menggunakan nilai batas yang bergantung dengan panjang lintasan dari simpul tertentu sampai simpul solusi terdekat dalam anak – anaknya. Tentunya pendekatan ini adalah pendekatan yang ideal dan tidak menyelesaikan masalah. Hal ini karena kita harus mengetahui letak simpul solusi untuk menentukan *cost* dari suatu simpul. Namun untuk penyederhanaan maka kita akan mengikuti pendekatan ini dikarenakn dari contoh sebelumnya letak simpul – simpul solusi sudah diektahui. Dengan demikian *cost* dari suatu simpul adalah jarak simpul tersebut sampai ke simpul solusi terdekat dalam anak – anaknya. Untuk simpul yang tidak memiliki anak akan diberikan *cost* tak hingga atau dapat dikatakan simpul telah dipangkas karena dianggap tidak mengarah ke solusi. Hasil pohon ruang status yang didapat adalah seperti pada gambar berikut ini.



Gambar 2 Pohon Ruang Status yang Dibangkitkan dengan Algoritma *Branch and Bound* [4]

Terlihat bahwa hanya perlu dibangkitan 10 simpul. Walau demikian hal ini tidak berarti algoritma ini lebih baik daripada algoritma *backtracking*. Dalam kasus ini hasil tersebut lebih bergantung dari masalah persoalan yang dihadapi.

III. Implementasi *Branch and Bound* untuk Penentuan Strategi Pada Permainan Tic-tac-toe

Hal terpenting dalam penerapan algorimta *Branch and Bound* adalah menentukan fungsi batas yang juga digunakan untuk menghitung *cost* dari suatu simpul. Kita akan mendasari perhitungan *cost* ini faktor – faktor yang dapat dihitung dari permainan Tic-tac-toe. Berikut akan dijelaskan beberapa faktor yang dianggap menentukan pilihan yang optimal pada permainan Tic-tac-toe.

Pertama, jika dapat memenangkan permainan, maka pilihlah tempat tersebut. Tentunya hasil pengisian setelahnya tidak lagi penting dikarenakan permainan telah berakhir.

Kedua, jika langkah lawan berikutnya pada posisi tersebut memungkinkan lawan untuk menang, maka isikan posisi tersebut. Langkah ini adalah wujud dari *block*.

Ketiga, langkah yang baik adalah langkah yang memberikan peluang *fork* yang tinggi. Hal ini dikarenakan kunci kemenangan permainan ini adalah *fork* yang dapat membuat lawan tidak dapat melakukan *block*.

Keempat, langkah yang baik adalah langkah yang mencegah lawan mengisi satu baris, kolom, ataupun diagonal sampai penuh dengan objek pengisiannya. Langkah ini berarti dilakukan pencarian langkah *block fork* yang baik.

Dari 4 dasar pemikiran tersebut maka salah satu perhitungan *cost* (atau dalam hal ini lebih mirip dengan *performance* daripada *cost*) yang mungkin adalah

$$ca = opr(i,j) + opc(i,j) + opd(i,j) \quad (1)$$

$$cd = or(i,j) + oc(i,j) + od(i,j) + br(i,j) + bc(i,j) + bd(i,j) \quad (2)$$

Dengan opr(i,j) menandakan *open row* atau baris yang belum diisi oleh lawan. opr(i,j) ini akan bernilai 1 jika baris yang membuat *cell* dari baris ke-i dan kolom ke-j dalam bidang permainan adalah sebuah *open row*. Sedangkan opc(i,j) menandakan *open column* dan opd(i,j) adalah *open diagonal*. Nilai dari opd(i,j) sendiri adalah maksimal 2, yaitu untuk *cell* yang berada pada tengah bidang. Tentunya hal ini dikarenakan bidang tersebut dilewati dua buah diagonal bidang permainan. Jumlah dari semua ini menentukan ca yaitu *Cost* of *Attacking*. Dasar pemikiran pengambilan perhitungan ini adalah dengan berdasarkan pemikiran bahwa *cell* dengan jumlah terbesar *row, column,* dan *diagonal* yang belum diisi lawan adalah *cell* yang paling menguntungkan untuk penyerangan sekaligus memungkinkan kemenangan. Jika suatu *row* sudah ada satu *cell* yang diisi oleh lawan, maka *row* tersebut sudah tidak mungkin menghasilkan kemenangan, karena itulah dipilih *cell* yang

paling *open*. Selain itu dengan pemilihan banyak tempat yang *open* diharapkan juga lebih memungkinkan *fork*.

Seperti ca, cd adalah *Cost* of *Defending*. Nilai cd sendiri ditentukan dari total beberapa variabel. Variabel tersebut salah satunya adalah or(i,j) yang menghitung jumlah *cell* yang terisi oleh lawan pada baris yang memuat *cell* baris ke-i dan kolom ke-j. Mirip dengan or(i,j), oc(i,j) adalah variasi untuk kolom dari variabel tersebut. Sementara od(i,j) adalah variasi untuk diagonal dari vairabel tersebut. Selain itu terdapat juga variabel br(i,j) yang akan bernilai tak hingga jika pada baris yang memuat *cell* baris ke-i dan kolom ke-j terdapat (panjang bidang – 1) bidang yang telah diisikan oleh lawan. Variabel ini menentukan kapan harus dilakukan *block* pada suatu *cell*. Mirip dengan br(i,j), bc(i,j) adalah variasi untuk kolom sedangkan bd(i,j) adalah variasi untuk diagonal. Perhitungan ini memastikan untuk melakukan *block* pada *cell* yang mengakibatkan banyak *cell* yang diisikan oleh lawan tidak dapat menghasilkan kemenangan lagi dan juga memastikan melakukan *block* untuk mencegah kekalahan sebaik mungkin.

Salah satu pertanyaan yang muncul mungkin adalah mengapa tedapat dua buah *cost* dalam rancangan algoritma ini. Hal ini sebenarnya disebabkan oleh 4 dasar pemikiran yang ada. Dasar pemikiran tersebut secara logika sudah dapat dikatakan masuk akal. Jika kita lihat terdapat langkah menyerang dan langkah bertahan dalam permainan ini. Untuk dapat lebih optimal seharusnya terdapat cara perhitungan yang dapat menghitung sebaiknya saat ini dilakukan penyerangan atau pertahanan. Namun perhitungan tersebut akan sulit. Dikarenakan lawan bermain yang dibuat dalam kasus ini adalah difokuskan pada pemain yang jalan kedua, maka diputuskan untuk membuat strategi yang menekankan pada pertahanan. Hal ini dikarenakan jika pemain pertama bermain secara optimal, hasil terbaik yang dapat diperoleh pemain kedua adalah seri. Artinya penyerangan tidak begitu berarti untuk pemain kedua. Dengan demikian *cost* yang menjadi pembanding utama adalah cd (*Cost of Defending*). Sedangkan ca dalam hal ini berfungsi untuk memberikan pilihan yang optimal jika terdapat dua atau lebih cd bernilai sama. Pilihan optimal yang dibentuk dalam hal ini menjadi pilihan posisi yang mencegah (*block*) lawan menang sebaik mungkin dengan prioritas kedua memilih tempat paling terbuka untuk melakukan *fork*.

Namun, jika hanya sebatas itu, maka pemain ini akan cenderung tidak mengambil kesempatan kemenangan yang ada karena dalam alur berpikir program, hal yang harus diutamakan adalah pertahanan. Untuk itu untuk menciptakan lawan bermain yang lebih optimal, akan diperlukan pengecekan terlebih dahulu apakah langkah tertentu memungkinkan kemanangan. Jika terdapat langkah demikian, maka langkah tersebut akan diambil karena merupakan langkah paling optimal.

Sampai pada bagian ini memang pemikiran yang dihasilkan tampak tidak sedikit perbedaannya dengan algoritma *Branch and Bound* yang dijelaskan sebelumnya. Namun pada dasarnya sebenarnya hasil algoritma akhir program masih dapat dikatakan variai dari algoritma *Branch and Bound* dengan pendekatan tidak boleh pindah upapohon dan solusi pertama (karena permainan akan berakhir).

Jika dideskripsikan maka, pertama pohon ruang status hanya akan memiliki akar yang kemudian di-*expand* menghasilkan 9 anak. Namun pemilihan anak yang akan di-*expand* berikutnya tidak bergantung pada algoritma karena giliran pertama adalah langkah dari pemain. Pemain ini bisa saja memilih langkah yang tidak optimal. Misalkan saja pemain memilih langkah yang mengarah ke anak ke-a. Kemudian simpul tersebut akan di-*expand* menghasilkan 8 anaknya. Pemilihan anak setelah ini bergantung pada algoritma dikarenakan merupakan langkah dari program. Program kemudian akan memilih langkah yang paling optimal sesuai dengan deskripsi di atas. Simpul pilihan kemudian di-*expand* lagi menghasilkan 7 anaknya. Proses ini akan terus diulang sampai dapat ditentukan hasil dari permainan. Pohon ruang status tersebut dalam hal ini menggambarkan alur langkah – langkah yang diambil dalam permainan baik optimal maupun tidak.

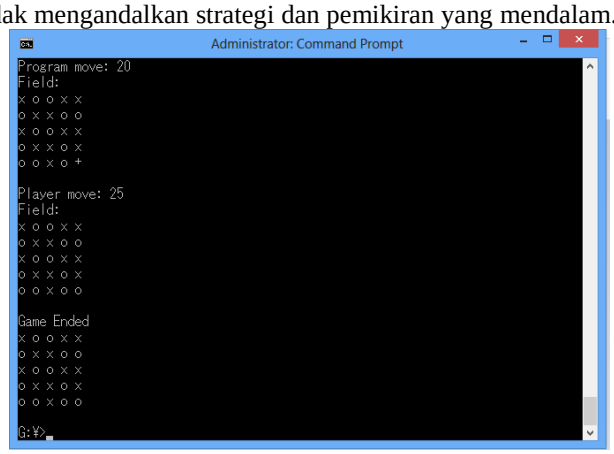
Berikut akan disajikan *pseudo-code* dengan Bahasa menyerupai Bahasa C++ dari program Tic-tac-toe yang dibuat berdasarkan strategi ini.

```
//TabInt adalah array of integer

//Melakukan pencetakan array
Procedure (input TabInt field, input int size)
	for (int i <- 0 ; i < size ; i++)
		for (int j <- 0 ; j < size ; j++)
			if (field[(i * size) + j] =
1)
				output("0 ")

			else if (field[(i * size) +
j] = 2)
				output("x ")

			else
				output("+ ")

//Memerika kondisi kemenangan
Function isWin (TabInt oRow, TabInt xRow, TabInt
oCol, TabInt xCol, TabInt oDia, TabInt xDia, int
size) : bool
	bool win = false
	int i = 0
	int count = 0

	while ((NOT win) AND (i < size))
		if ((oRow[i] = size) OR (xRow[i] =
size) OR (oCol[i] = size) OR (xCol[i] = size))
			win <- true

		count <- count + oRow[i] + xRow[i]
		i++

	if (NOT win)
		if ((oDia[0] = size) OR (xDia[0] =
size) OR (oDia[1] = size) OR (xDia[1] = size))
			win <- true

	if (NOT win)
		if (count = size * size)
			win <- true

	-> win
```

```
//Pengecekan jumlah pengisian pemain 1 dan 2 pada tiap baris
Procedure checkRow(TabInt field, TabInt oRow, TabInt xRow, int size)
	for (int i <- 0; i < size ; i++)
		oRow[i] <- 0
		xRow[i] <- 0
		for (int j <- 0; j < size; j++)
			if (field[((i * size) + j)] = 1)
				oRow[i]++

			else if (field[((i * size) + j)] = 2) {
				xRow[i]++

/*
CheckCol dan CheckDia melakukan hal yang sama untuk kolom dan diagonal dari bidang permainan. Hasil disimpan pada oCol, xCol, oDia, dan xDia
*/

//Program Utama
Program Tic-tac-toe
	TabInt field
	int size
	int input
	bool pTurn <- true
	bool win <- false
	TabInt oRow
	TabInt xRow
	TabInt oCol
	TabInt xCol
	TabInt oDia
	TabInt xDia

	Input(size)
	InisialiasaiBidang()
	InisialisasiArrayPencatat()

	while (NOT win)
		if (pTurn)
			Input(input)
			VerifikasiInput (field,input)
			field[input - 1] <- 1
			pTurn <- false
		else
			input = decideInput(field, oRow, xRow, oCol, xCol, oDia, xDia, size)
			field[input] <- 2
			pTurn <- true

		checkRow(field, oRow, xRow, size)
		checkCol(field, oCol, xCol, size)
		checkDia(field, oDia, xDia, size)
		win <- isWin(oRow, xRow, oCol, xCol, oDia, xDia, size)
		print(field, size)

	print(field, size)
```

IV. Hasil Pengujian dan Pembahasan

Hasil pengujian program yang dibuat berdasarkan *pseudo-code* di atas menghasilakan hasil seperti yang tersaji pada tabel berikut. Pengujian dilakukan oleh pemain manusia berumur 20 tahun tanpa riwayat medis untuk gangguan kecerdasan yang bermain secara intutif (reaksi, merasa, dan sedikit berpikir) dan tidak mengandalkan strategi dan pemikiran yang mendalam.



Gambar 3 Uji Coba Program

Ukuran	Jumlah Pengujian	Jumlah Menang	Jumlah Kalah	Jumlah Seri
3x3	10	0	3	7
4x4	5	0	1	4
5x5	3	0	0	3

Tabel 1 Hasil Uji Coba Program

Dari hasil pengujian tersebut terlihat pemain manusia yang mengandalakan intusi tidak pernah memperoleh kemenangan dan mengalami beberapa kali kekalahan.

Seperti yang dijelaskan sebelumnya permainan Tic-tac-toe dengan ukuran lebih dari 3x3 dan kemenangan satu baris sudah cenderung tidak memungkinkan untuk dimenangkan oleh kedua belah pihak. Hal ini dikarenakan dibutuhkan langkah yang banayk untuk mencapai kemenangan dan langkah *fork* memerlukan banyak langkah juga. Untuk permainan ukuran 4x4 hal tersebut masih lebih memungkinkan.

Dari data pengujian ukuran 3x3 diektahui bahwa pemain kalah 3 kali dan 7 kali mengalami seri. Dari hal ini dapat disimpulkan bahwa algoritma ini cukup baik untuk menjadi lawan bermain pemain yang hanya mengandalkan intuisi untuk ukuran 3x3.

Walau pada semua pengujian tidak terdapat kemenangan, masih belum dapat disimpulkan apakah algoritma ini sudah berada pada tingkat *expert player* yang telah menerapkan strategi optimal seperti yang telah dijelaskan pada dasar teori. Untuk kasus 3x3 sendiri pengujian 10x belum cukup untuk membuktikan apakah semua kemungkinan gerakan akan berhasil ditangani sampai minimal seri. Pengecekan semua langkah yang dimungkinkan pemain manusia yang bergerak pertama untuk ukuran 3x3 adalah 9 x 7 x 5 x 3 x 1 = 945 kemungkinan. Uji coba ini baru mencoba sekitar 1% dari kemungkinan tersebut.

Terlepas apakah program ini sudah berada pada tingkat *expert* player atau belum, perlu diingat teknik penghitungan *cost* yang digunakan dalam makalah ini hanyalah salah satu dari banyak kemungkinan. Artinya masih dimungkinkan adanya perhitungan *cost* yang lebih baik untuk menentukan langkah yang optimal.

V. KESIMPULAN

Kesimpulan dari pembuatan lawan bermain Tic-tac-toe dengan berdasarkan algoritma *Branch and Bound* adalah dimungkinkan untuk membuat lawan bermain permainan Tic-tac-toe yang cukup kompetitif dengan berdasarkan algoritma *Branch and Bound*. Algoritma *Branch and Bound* yang digunakan sendiri memerlukan sedikit modifikasi namun inti dari algoritma tersebut masih tetap algoritma *Branch and Bound*.

VI. UCAPAN TERIMA KASIH

Rasa terima kasih penulis ucapkan terhadap Tuhan Yang Maha Esa karena hanya berkat rahma dan karunia-Nya makalah ini dapat diselesaikan. Penulis juga mengucapkan terima kasih kepada dosen pengajar mata kuliah IF2211 Strategi Algoritma, yaitu Dr. Ir. Rinaldi Munir, MT. dan juga Dr. Nur Ulfa Maulidevi, S.T., M.Sc. yang telah memberikan dasar ilmu untuk Strategi Algoritma.

REFERENCES

[1] Merriam-Webster. http://www.merriam-webster.com/dictionary/tic%E2%80%93tac%E2%80%93toe diakses pada 8 Mei 2016 pukul 23.23,

[2] Janet Chen, Su-I Lu, and Dan Vekhter, *Zero-Sum Game*. https://cs.stanford.edu/people/eroberts/courses/soco/projects/1998-99/game-theory/zero.html diakses pada 8 Mei 2016 pada 23.54 .

[3] Kevin Crowley, Robert S. Sieglerm, *Flexible Strategy Use in Young Children's Tic-Tac-Toe*. *Cognitive Science* 17 : 536.

[4] Munir, Rinaldi. *Branch & Bound*. http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2015-2016/Algoritma-Branch-&-Bound-(2016).pptx diakses pada 9 Mei 2016 01.00

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain dan bukan plagiasi.

Bandung, 9 Mei 2016

Andri Hardono Hutama, 13514031