Perbandingan Algoritma Greedy dan Algoritma Minmax pada Permainan Tic Tac Toe

Richard Matthew 13515094
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13515094@std.stei.itb.ac.id

***Abstract*—Makalah ini akan membahas perbandingan algoritma greedy dan algoritma minmax pada permainan Tic Tac Toe. Makalah ini akan memakai teori *Greedy* dan *Minmax*.**

***Keywords*—permainan, *Tic tac toe*, *Greedy*, *Minmax*.**

I. Introduction

Tic tac toe adalah permainan yang dimainkan oleh 2 orang. Cara memenangkannya adalah dengan membuat 3 gambar bersebelahan. Permainan ini menggunakan persegi berukuran 3x3. Biasanya permainan ini menggunakan simbol x dan o.

Tic tac toe mempunyai beberapa nama lain, seperti catur jawa, *nought and crosses* , Xs and Os.

Menurut sejarah, *tic tac toe* berasal dari kekaisaran Romawi kuno sekitar abad pertama sebelum masehi, waktu itu permainan ini disebut *terni lapili.* Aturan permainannya sedikit berbeda, pemain hanya memiliki tiga buah benda/simbol, mereka harus menggerak-gerakan benda/simbol sampai membuat 3 gambar bersebelahan. Pertama kali referensi cetak muncul di Inggris dengan nama "*Noughts and Crosses*" pada tahun 1864. Permainan ini baru berganti nama ke "*tic tac toe*" di abag ke-20 di Amerika Serikat. Permainan ini pada abad tersebut, digunakan untuk menampilkan visual pada monitor video.

II. Landasan Teori

2.1 Teori Algoritma *Greedy*

Algoritma g*reedy* adalah metode yang paling populer untuk memecahkan persoalan optimasi, persoalan optimasi adalah persoalan mencari solusi optimum. Persoalan optimasi hanya dibagi menjadi dua macam persoalan, yaitu maksimasi dan minimasi.

Prinsip yang digunakan pada algoritma *Greedy* adalah mengambil apa yang kita dapat ambil sekarang. Algoritma greedy membentuk solusi langkah per langkah. Pada setiap langkah, kita mencari pilihan yang terabaik pada langkah tersebut, disebut juga optimum lokal, dengan harapan langkah-langkah berikutnya mengarah ke solusi optimum global.

Elemen-elemen pada algoritma greedy adalah himpunan kandidat, himpunan solusi, fungsi seleksi, fungsi kelayakan dan fungsi obyektif.

2.2 Teori Algoritma *Minmax*

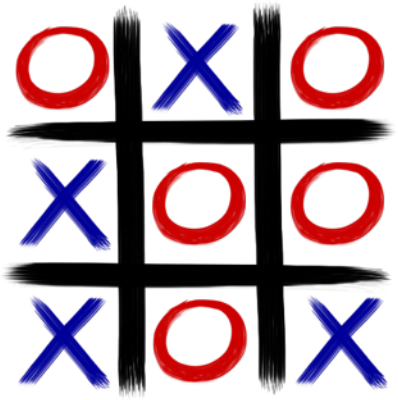
Algoritma m*inmax* atau m*inimax* adalah sebuah algoritma yang digunakan pada permainan dengan dua pemain dengan cara memprediksi keadaan masa depan, dan kemudian mengambil langkah yang memaksimalkan peluang kemenangan. Teori di balik *minimax* adalah bahwa algoritma lawan akan mencoba untuk meminimalkan nilai apapun yang algoritma pemain berusaha maksimalkan. Dengan konsep seperti itu, maka algoritma ini akan melakukan gerakan yang membuat lawannya memiliki peluang lebih kecil untuk menang.

Idealnya algoritma ini menganalisa setiap hasil yang dimungkinkan terjadi dalam sebuah permainan. Untuk permainan seperti *tic tac toe,* hanya ada tiga pilihan hasil, yaitu menang, kalah dan seri. Tetapi pada permainan yang lebih kompleks seperti *poker,* skor yang akan menjadi tujuan utama.

Algoritma m*inmax* merupakan perkembangan dari algoritma *backtracking*. Algoritma ini memanfaatkan pohon keputusan dan juga b*readth-first search* atau d*epth-first search*. Algoritma ini juga menggunakan fungsi pembatas jika dibutuhkan.

2.3 Permainan *Tic Tac Toc*

2.3.1 Gambar Permainan *Tic Tac Toe*



Gambar 1: *Tic tac toe*
Sumber : http://csharpcorner.mindcrackerinc.netdna-cdn.com/UploadFile/75a48f/tic-tac-toe-game-in-C-Sharp/Images/TicTacToe_HD_iTunesArtwork.png

2.3.2 Data dari Permainan T*ic Tac Toe*

Meskipun permainan ini terlihat sederhana tetapi secara teori, ada 19.683 tata letak papan mungkin (39 karena masing-masing dari sembilan ruang dapat X, O atau kosong), dan 362.880 urutan yang berbeda untuk menempatkan Xs dan Os di papan (yaitu 9!); yaitu, tanpa memperhitungkan kombinasi pemenang pertimbangan yang akan membuat banyak dari mereka tidak terjangkau dalam permainan yang sebenarnya.

Ketika kombinasi pemenang dianggap, ada 255.168 mungkin game. Dengan asumsi bahwa X yang melakukan langkah pertama setiap kali:

- 131.184 permainan selesai dimenangkan oleh (X)
- 1440 yang dimenangkan oleh (X) setelah 5 bergerak
- 47.952 dimenangkan oleh (X) setelah 7 bergerak
- 81.792 dimenangkan oleh (X) setelah 9 bergerak
- 77.904 permainan selesai dimenangkan oleh (O)
- 5328 yang dimenangkan oleh (O) setelah 6 bergerak
- 72.576 dimenangkan oleh (O) setelah 8 bergerak
- 46.080 permainan selesai dengan seri

Mengabaikan urutan Xs dan Os, dan setelah menghilangkan hasil simetris (yaitu rotasi dan / atau refleksi dari hasil lainnya), hanya ada 138 hasil unik. Dengan asumsi sekali lagi bahwa X yang melakukan langkah pertama setiap kali:

- 91 hasil yang unik dimenangkan oleh (X)
- 21 dimenangkan oleh (X) setelah 5 bergerak
- 58 dimenangkan oleh (X) setelah 7 bergerak
- 12 dimenangkan oleh (X) setelah 9 bergerak
- 44 hasil yang unik dimenangkan oleh (O)
- 21 dimenangkan oleh (O) setelah 6 bergerak
- 23 dimenangkan oleh (O) setelah 8 bergerak
- 3 hasil yang unik yang seri

Contoh akhir permainan dapat dilihat pada gambar 4(untuk x menang), 5(untuk o menang), dan 3(jika seri

Gambar 2: ilustrasi ketika x menang
Sumber: https://www.colourbox.com/preview/4188073-tic-tac-toe-winning.jpg

Gambar 3: ilustrasi ketika o menang
Sumber: http://vignette3.wikia.nocookie.net/uncyclopedia/images/5/53/Tictactoe02.png/revision/latest?cb=20110409145750

2.3.3 Strategi Permainan

Ada beberapa strategi yang digunakan pada permainan ini, yaitu:

s

1. *Win:* Jika pemain memiliki dua gambar bersebelahan, letakkan yang ketiga agar mendapat tiga gambar bersebelahan.
2. *Block:* Jika lawan memiliki dua gambar bersebelahan, letakan pada sebelahnya untuk memblokir lawan.
3. *Fork:* Buat kesempatan di mana Anda bisa menang dalam dua cara.
4. *Block fork* lawan:
   - Opsi 1: Buat dua bersebelahan untuk memaksa lawan untuk *block*, asalkan tidak mengakibatkan mereka menciptakan *fork* atau *win*. Sebagai contoh, jika "X" memiliki sudut, "O" memiliki pusat, dan "X" memiliki sudut yang berlawanan juga, "O" tidak harus memainkan sudut untuk menang. (Bermain sudut dalam skenario ini menciptakan garpu untuk "X" untuk menang.)
   - Opsi 2: Jika ada konfigurasi di mana lawan bisa *fork*, memblokir *fork* itu.
5. *Center*: Meletakkan di tengah.
6. *Opposite corner* : Jika lawan meletakkan di sudut, letakkan di sudut yang berlawanan.
7. *Empty corner*: meletakkan di sudut.
8. *Empty side:* Meletakkan di lapangan tengah pada salah satu dari 4 sisi.

Pemain pertama, siapakah kami akan menunjuk "X", memiliki 3 posisi mungkin untuk menandai saat giliran pertama. Sekilas, mungkin tampak bahwa ada 9 kemungkinan posisi, sesuai dengan 9 kotak dalam grid. Namun, dengan memutar papan, kita akan menemukan bahwa di giliran pertama, setiap sudut mark adalah strategis setara dengan setiap tanda sudut lain. Hal yang sama berlaku setiap mark tepi. Untuk tujuan strategi, ada karena hanya ada tiga kemungkinan pertama tanda: sudut, tepi, atau pusat. Pemain X dapat menang atau memaksakan hasil imbang dari semua ini tanda mulai; Namun, bermain

sudut memberikan lawan pilihan terkecil kotak yang harus dimainkan untuk menghindari kekekalahan

Pemain kedua, yang akan kita menunjuk "O", harus menanggapi tanda pembukaan X sedemikian rupa untuk menghindari kemenangan paksa. Pemain O harus selalu menanggapi pembukaan sudut dengan tanda pusat, dan untuk pembukaan pusat dengan tanda sudut. Pembukaan tepi harus dijawab baik dengan tanda pusat, tanda sudut sebelah X, atau tanda tepi berlawanan X. Setiap tanggapan lain akan memungkinkan X untuk memaksa menang. Setelah pembukaan selesai, tugas O adalah untuk mengikuti daftar di atas prioritas untuk memaksa hasil imbang, atau yang lain untuk mendapatkan kemenangan jika X membuat kesalahan.

III. PERBANDINGAN ALGORITMA GREEDY DAN ALGORITMA MINMAX PADA PERMAINAN *TIC TAC TOE*

3.1 Implementasi Algoritma *Greedy* pada Permainan *Tic Tac Toe*

Implementasi algoritma *greedy* pada permainan *tic tac toe* ini dengan spesifikasi:

- Himpunan kandidat: Himpunan koordinat kotak yang belum terisi x ataupun o
- Himpunan solusi: membentuk 3 x bersebelahan (dengan asumsi algoritma *greedy* diterapkan pada x) atau kesembilan kotak telah terisi
- Fungsi seleksi: memilih kotak yang memiliki nilai tertinggi, sesuai dengan nilai yang ditentukan sebagai berikut:

Nilai	kotak 1	kotak 2	kotak 3
10000	X	X	X
1000	X	O	O
1000	O	X	O
200	X	n	X
100	X	X	n
20	X	n	n
15	X	n	O
10	n	X	n
5	O	X	n
5	X	O	n

Tabel 1. nilai fungsi seleksi

* berlaku pencerminannya

* n artinya kosong

Juga ada kasus khusus ketika pada langkah pertama, lawan meletakkan pada sudut, maka algoritma akan mengambil kotak tengah.

Lalu kotak sudut mempunyai nilai tambahan sebesar 100;

- Fungsi layak : Memeriksa apakah permainan sudah dimenangkan atau sudah tidak ada kotak yang dapat terisi
- Fungsi obyektif: Memenangkan permainan atau seri

Tahap memutuskan penilaiannya adalah dengan cara memilih nilai terbesar dari perhitungan berdasarkan tabel diatas juga mempertimbangkan kasus khusus. Nilai dihitung hanya pada baris/kolom/diagonal yang diletakkan X baru.

3.2 Implementasi Algoritma Minmax pada permainan tic tac toe

Implementasi Algoritma Minmax pada permainan tic tac toe ini memanfaatkan algoritma Depth-First search. Dengan menghitung nilai berdasarkan berapa besar kemungkinan menang jika menggunakan cabang tersebut. Algoritma ini tidak menggunakan fungsi pembatas karena waktu yang digunakan masih terjangkau.

3.3 Source Code

Adapun source code yang digunakan untuk mensimulasikan adalah perkembangan dari source code yang dibuat oleh Matthew Steel. Source code sebagai berikut :

```
//Tic-tac-toe playing AI. Exhaustive tree-search. WTFPL
//Matthew Steel 2009, www.www.repsilat.com
//Edited by Richard Matthew

#include <stdio.h>
#include <time.h>

float timerGreedy=0;
float timerMinmax=0;
int countWinGreedy=0;
int countWinMinmax=0;
int countDraw=0;
char gridChar(int i) {
  switch(i) {
    case -1:
      return 'X';
    case 0:
      return ' ';
    case 1:
      return 'O';
    default:
      return 'a';
  }
}

void draw(int b[9]) {
  printf(" %c | %c | %c\n",gridChar(b[0]),gridChar(b[1]),gridChar(b[2]));
  printf("---+---+---\n");
  printf(" %c | %c | %c\n",gridChar(b[3]),gridChar(b[4]),gridChar(b[5]));
  printf("---+---+---\n");
  printf(" %c | %c | %c\n",gridChar(b[6]),gridChar(b[7]),gridChar(b[8]));
}

int win(const int board[9]) {
```

```
 //determines if a player has won, returns 0 otherwise.
 unsigned wins[8][3] = {{0,1,2},{3,4,5},{6,7,8},{0,3,6},{1,4,7},{2,5,8},{0,4,8},
{2,4,6}};
 int i;
    return board[wins[i][2]];
 }
 return 0;
}

int minimax(int board[9], int
player) {
 //How is the position like for
player (their turn) on board?
 int winner = win(board);
 if(winner != 0) return
winner*player;

 int move = -1;
 int score = -2;//Losing
moves are preferred to no
move
 int i;
 for(i = 0; i < 9; ++i) {//For
all moves,
   if(board[i] == 0) {//If legal,
     board[i] = player;//Try the
move
     int thisScore = -
minimax(board, player*-1);
     if(thisScore > score) {
       score = thisScore;
       move = i;
     }//Pick the one that's
worst for the opponent
     board[i] = 0;//Reset board
after try
   }
 }
 if(move == -1) return 0;
 return score;
}


void computerMove(int
board[9]) {
 clock_t t1, t2;
 t1 = clock();

 int move = -1;
 int score = -2;
 int i;
 for(i = 0; i < 9; ++i) {
   if(board[i] == 0) {
     board[i] = 1;
     int tempScore = -
minimax(board, -1);
     board[i] = 0;
     if(tempScore > score) {
       score = tempScore;
       move = i;
     }
   }
 }
 //returns a score based on
minimax tree at a given node.
 board[move] = 1;
 t2 = clock();
 float diff = ((float)(t2 - t1) /
1000000.0F ) * 1000;
 timerMinmax+= diff;

}


int SubPoint(int board[9],int
idx1,int idx2, int idx3, int
idxinput){


 int tempboard[3]=
{board[idx1],board[idx2],boa
rd[idx3]};
 if (idx1 == idxinput){
   tempboard[0] = -1;
 } else if (idx2 == idxinput){
   tempboard[1] = -1;
 } else if (idx3 == idxinput){
   tempboard[2] = -1;
 }

 for (int i=0; i<3;i++){
  switch (tempboard[i]) {
   case 1:
     tempboard[i]=-1;
     break;
   case -1:
     tempboard[i]=1;
     break;

   default:
     break;
  }
 }
 int score = 0;

 switch(tempboard[0]){
  case -1:
   switch(tempboard[1]){
    case -1:
     switch(tempboard[2]){
      case 1:
       score =+10000;
       break;
     }
     break;

    case 0:
     switch(tempboard[2]){
      case 1:
       score =+15;
       break;
     }
     break;

    case 1:
     switch(tempboard[2]){
      case -1:
       score =+10000;
       break;

      case 0:
       score =+5;
       break;
     }
     break;
   }
   break;

  case 0:
   switch(tempboard[1]){
    case -1:
     switch(tempboard[2]){
      case 1:
       score =+5;
       break;
     }
     break;

    case 0:
     switch(tempboard[2]){
      case 1:
       score =+20;
       break;
     }
     break;

    case 1:
     switch(tempboard[2]){
      case -1:
       score =+5;
       break;

      case 0:
       score =+10;
       break;

      case 1:
       score =+100;
       break;
     }
     break;
   }
   break;

  case 1:
   switch(tempboard[1]){
    case -1:
     switch(tempboard[2]){
      case -1:
       score =+10000;
       break;

      case 0:
       score =+5;
       break;

      case 1:
       score =+200;
       break;
     }
     break;

    case 0:
     switch(tempboard[2]){
      case -1:
       score+= 15;
       break;

      case 0:
       score+=20;
       break;

      case 1:
       score+= 200;
       break;
     }
     break;

    case 1:
     switch(tempboard[2]){
      case 0:
       score+= 100;
       break;

      case 1:
       score+= 10000;
       break;
     }
     break;
   }
   break;
 }

 return score;
};


int Point(int idx, int board[9])
{
 int score = 0;
 switch (idx) {
  case 0:{
    score = SubPoint(board,
0, 1,2,0 );
    score += SubPoint(board,
0, 4,8,0 );
    score += SubPoint(board,
0, 3,6,0 );
   }break;
  case 1:{
    score = SubPoint(board,
0, 1,2,1 );
    score += SubPoint(board,
1, 4,7,1 );
   }break;
  case 2:{
    score = SubPoint(board,
0, 1,2,2 );
    score += SubPoint(board,
2, 4,6,2 );
    score += SubPoint(board,
2, 5,8,2 );
   }break;
  case 3:{
    score = SubPoint(board,
0, 3,6,3 );
    score += SubPoint(board,
3, 4,5,3 );
   }break;
  case 4:{
    score = SubPoint(board,
0, 4,8,4 );
    score += SubPoint(board,
2, 4,6,4 );
    score += SubPoint(board,
1, 4,7,4 );
    score += SubPoint(board,
3, 4,5,4 );
   }break;
  case 5:{
    score = SubPoint(board,
2, 5,8,5 );
    score += SubPoint(board,
3, 4,5,5 );
   }break;
  case 6:{
    score = SubPoint(board,
0, 3,6,6 );
    score += SubPoint(board,
2, 4,6,6 );
    score += SubPoint(board,
6, 7,8,6 );
   }break;
  case 7:{
    score = SubPoint(board,
1, 4,7,7 );
    score += SubPoint(board,
6,7,8,7 );
   }break;
  case 8:{
    score = SubPoint(board,
2, 5,8,8 );
    score += SubPoint(board,
0, 4,8,8 );
    score += SubPoint(board,
6, 7,8,8 );
   }break;
   default:
     break;
 }
 return score;
};

void greedy(int board[9]){
 clock_t t1, t2;
 t1 = clock();
 int idxpilih = -1;
 int max = 0;
 int moves[9] =
{100,0,100,0,0,0,100,0,100};
 int count =0;
 for(int i=0;i<9;i++){
  if (board[i]!=0){
    count++;
  }
 }
 if ((count==1) &&
((board[0]== 1)|| (board[2]==
1)|| (board[6]== 1)||
(board[8]== 1))){
   idxpilih = 4;
 } else{

 for(int i =0 ; i<9;i++){
  if (board[i]==0){
    int temp =(Point(i, board)
+moves[i]);
    if(temp >max){
      idxpilih = i;
      max = temp;
    }

  }
 }
 }

for(i = 0; i < 8; ++i) {
 if(board[wins[i][0]] != 0 &&
   board[wins[i][0]] == board[wins[i][1]] &&
   board[wins[i][0]] == board[wins[i][2]])
     board[idxpilih]=-1;
 t2 = clock();
 float diff = ((float)(t2 - t1) /
1000000.0F ) * 1000;
 timerGreedy+= diff;
};

void main2(int pick){
 int board[9] =
{0,0,0,0,0,0,0,0,0};

 unsigned turn;
 for(turn = 0; turn < 9 &&
win(board) == 0; ++turn) {
   if((turn+pick) % 2 == 0){
     greedy(board);
     printf("Greedy Turn:\n");
     draw(board);
   }
   else {
     computerMove(board);
     printf("Minmax Turn:\n");
     draw(board);
   }
 }
 switch(win(board)) {
   case 0:
     printf("A draw.\n");
     countDraw++;
     break;
   case -1:
     draw(board);
     printf("Greedy win.\n");
     countWinGreedy++;
     break;
   case 1:
     printf("Minmax win\n");
     countWinMinmax++;
     break;
 }


}

int main() {
 //greedy are 1, minmax are
-1.
 //printf("Computer: O, You:
X\nPlay (1)st or (2)nd? ");

 printf("Greedy: X, Minmax:
O\nSet number of matches :
");
 int pick=0;
 scanf("%d",&pick);
 printf("\n");

 for(int i=0;i<pick;i++){
   main2(i);
 }

 printf("Total Minmax time:
%f\n",timerMinmax);
 printf("Number of Minmax
Win:
%d\n",countWinMinmax);
 printf("Total Greedy time:
%f\n",timerGreedy);
 printf("Number of Greedy
Win:
%d\n",countWinGreedy);
 printf("Draw:
%d\n",countDraw);


}
```

3.4 Hasil Data dan Analisa

Jalan permainan yang terbentuk dari kedua algoritma diatas adalah sebagai berikut

Ketika algoritma greedy yang memulai:

```
Greedy Turn:
 X |   |
---+---+---
   |   |
---+---+---
   |   |
Minmax Turn:
 X |   |
---+---+---
   | O |
---+---+---
   |   |
Greedy Turn:
 X |   | X
---+---+---
   | O |
---+---+---
   |   |
Minmax Turn:
 X | O | X
---+---+---
   | O |
---+---+---
   |   |
Greedy Turn:
 X | O | X
---+---+---
   | O |
---+---+---
   | X |
Minmax Turn:
 X | O | X
---+---+---
 O | O |
---+---+---
   | X |
Greedy Turn:
 X | O | X
---+---+---
 O | O | X
---+---+---
   | X |
Minmax Turn:
 X | O | X
---+---+---
 O | O | X
---+---+---
   | X | O
Greedy Turn:
 X | O | X
---+---+---
 O | O | X
---+---+---
 X | X | O
A draw.
```

Ketika algoritma minmax yang memulai:

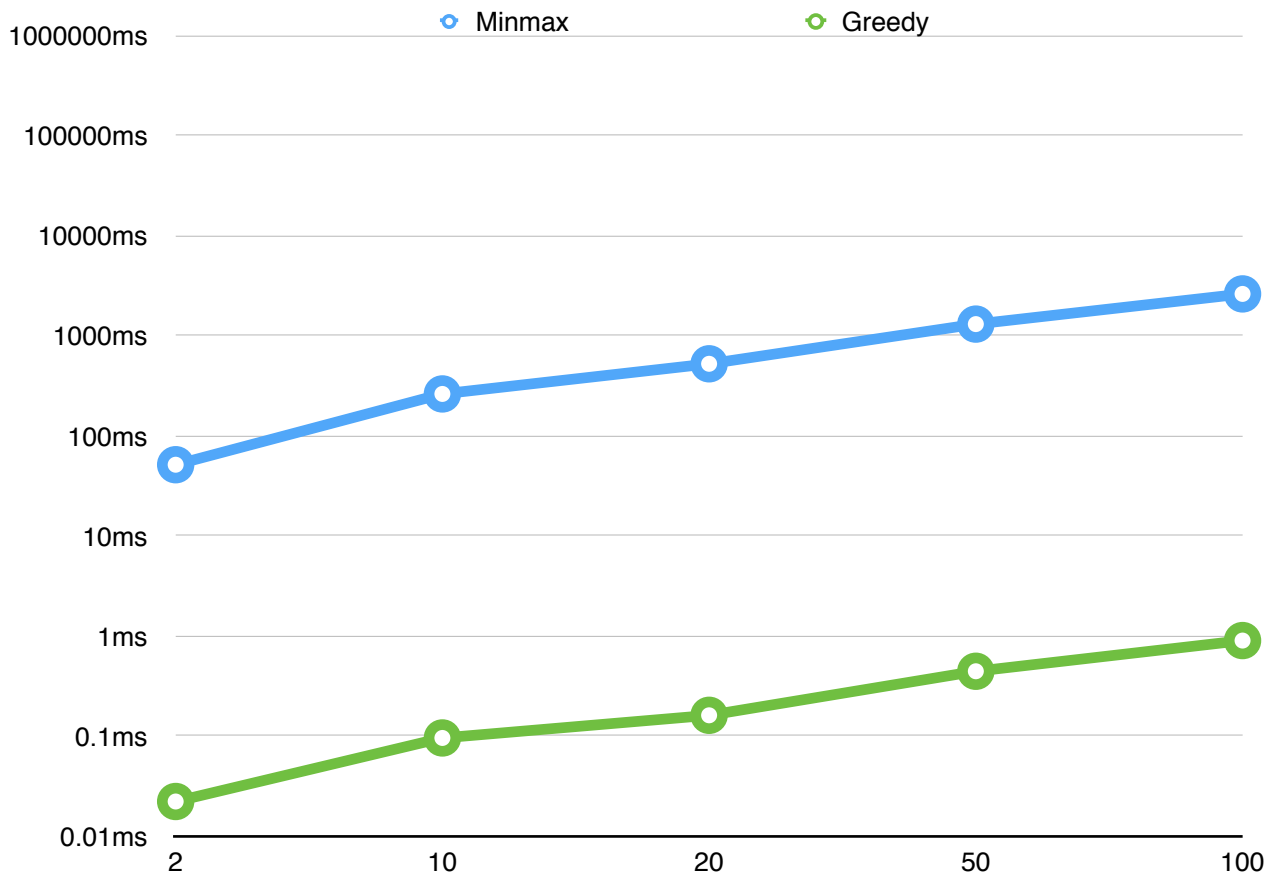
```
Minmax Turn:
 O |   |
---+---+---
   |   |
---+---+---
   |   |
Greedy Turn:
 O |   |
---+---+---
   | X |
---+---+---
   |   |
Minmax Turn:
 O | O |
---+---+---
   | X |
---+---+---
   |   |
Greedy Turn:
 O | O | X
---+---+---
   | X |
---+---+---
   |   |
Minmax Turn:
 O | O | X
---+---+---
   | X |
---+---+---
 O |   |
Greedy Turn:
 O | O | X
---+---+---
 X | X |
---+---+---
 O |   |
Minmax Turn:
 O | O | X
---+---+---
 X | X | O
---+---+---
 O |   |
Greedy Turn:
 O | O | X
---+---+---
 X | X | O
---+---+---
 O |   | X
Minmax Turn:
 O | O | X
---+---+---
 X | X | O
---+---+---
 O | O | X
A draw.
```

Lalu pada percobaan dalam 2,10,20,50 dan 100

pertandingan, semuanya menghasilkan seri. Berikut adalah waktu yang digunakan untuk menjalankan algoritma greedy dan minmax di atas:

	2	10	20	50	100
Minmax	51.198994	261.162018	522.626099	1303.337646	2611.14502
Greedy	0.022	0.095	0.16	0.441	0.893997

Tabel 2 Perbandingan waktu algoritma minmax dan greedy



Terlihat perbedaan yang signifikan dari waktu yang digunakan, algoritma greedy menggunakan waktu yang lebih singkat dari algoritma minmax.

IV. Kesimpulan

Dengan menggunakan fungsi seleksi yang tepat, algoritma greedy dapat memberikan hasil yang maksimal dengan waktu yang lebih cepat daripada algoritma minmax pada permainan tic tac toe.

Dalam permainan tic tac toe, sangat sulit untuk mendapatkan kemenangan.

V. Ucapan Terima Kasih

Pertama-tama penulis mengucapkan terima kasih kepada Tuhan Yang Maha Esa oleh karena anugerah- Nya penulis dapat menyelesaikan semua tulisan di makalah ini. Penulis ingin berterima kasih kepada dosen IF 2211 yaitu Bapak Dr.Ir. Rinaldi Munir, MT. , Ibu Dr. Masayu Leylia Khodra, ST.MT dan Ibu Dr.Nur Ulfa Maulidevi, ST., M.Sc. Serta penulis juga mengucapakan terima kasih yang tidak terhingga kepada semua teman-teman seperjuangan yang membantu penulis untuk menyelesaikan tulisan ini. Penulis pun tidak lupa mengucapkan terima kasih kepada semua pembaca tulisan ini dan semoga tulisan ini dapat bermanfaat bagi para pembacanya.

VI. References

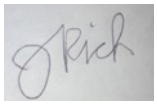
1. http://www.sweetoothdesign.com/games-tic-tac-toe. 19 Mei 2017 pukul 4.55
2. http://anna.fi.muni.cz/~x139877/prezentace/2011_03_24_prednaska_mafye_teorie_her/piskvorky.pdf. 19 Mei 2017 pukul 4.55

3. http://www-cs-faculty.stanford.edu/people/eroberts/courses/soco/projects/2003-04/intelligent-search/minimax.html 19 mei 2017 pukul 5.00
4. https://gist.github.com/MatthewSteel/3158579 15 Mei 2017 pukul 19.00

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,19 Mei 2017



Richard Matthew
13515094