BAB IV
PERANCANGAN DAN IMPLEMENTASI

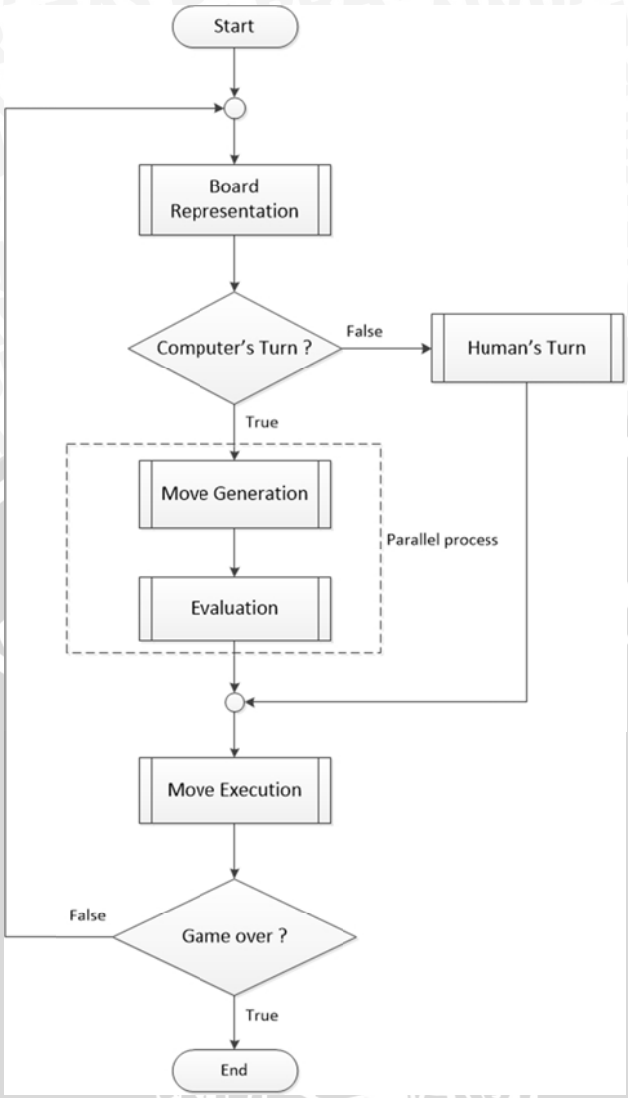
Pada bab ini akan dibahas tentang perancangan aplikasi permainan catur yang dikerjakan melalui 3 tahap dasar, yaitu tahap representasi papan, pembuatan dan pengecekan *move generation*, tahap evaluasi dan eksekusi gerakan. Pembuatan yang bertahap agar bisa dilakukan analisa pada setiap bagiannya maupun secara keseluruhan.

4.1 Perancangan Secara Umum

Perancangan aplikasi secara umum merupakan langkah awal sebagai acuan untuk pembuatan aplikasi lebih lanjut. Perancangan ini didahului dengan menentukan langkah – langkah paling dasar pada aplikasi permainan catur.

4.1.1 Diagram Sistem

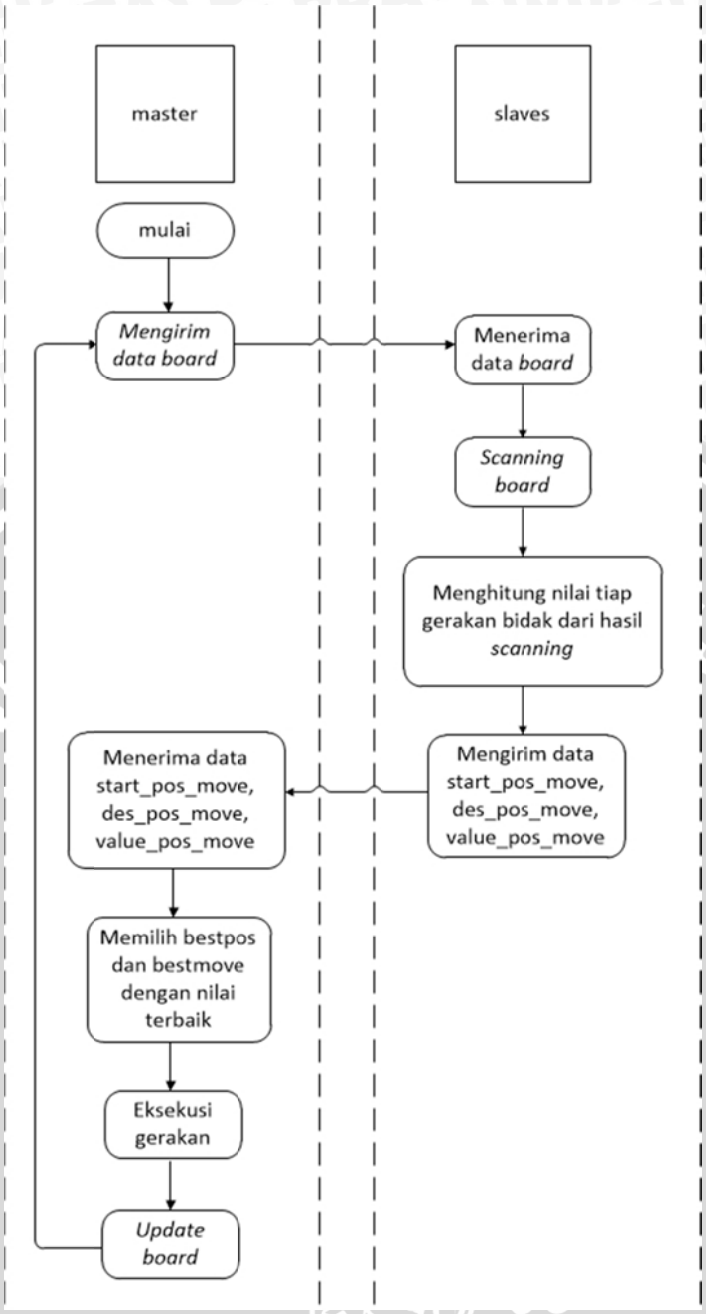
Seperti yang dijabarkan pada bab sebelumnya, aplikasi ini dibagi menjadi 3 bagian utama, yaitu *board representation*, *move generation,* dan *evaluation*. Proses ini bisa digambarkan pada diagram di bawah ini :



Gambar 4.1 *Flowchart* Sistem Pemrosesan Paralel pada Permainan catur

Fungsi dari masing – masing blok diagram adalah sebagai berikut :

1. *Board Representation*, yaitu pembuatan seluruh variabel yang nantinya akan digunakan, beserta tampilannya di dalam antarmuka.
2. *Move Generation* yaitu memasukkan aturan-aturan yang berlaku pada catur ke dalam program.
3. *Evaluation* yaitu proses perhitungan untuk seluruh gerakan yang mungkin dilakukan, dalam hal ini seluruh gerakan yang boleh dilakukan dalam catur (sesuai no. 2). Dan setelah diketahui gerakan apa saja yang mungkin akan dilakukan proses penilaian (evaluasi) untuk setiap gerakan tersebut. *Evaluation* ini dilakukan berulang-ulang dan diproses secara tunggal dan paralel.
4. *Move Execution* yaitu Eksekusi gerakan sesungguhnya, kemudian mengakhiri giliran.



Gambar 4.2 *Dataflow* Proses dari *Master* ke *Slave*

Keterangan:

Pada bagian *scanning board*:

- 2 *depth* dengan 2 *slave*:

  Depth 1:

  slave 1 *scanning* mulai *index* 0-31

  slave 2 *scanning* mulai *index* 32-63

  Depth 2:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

- 2 *depth* dengan 4 *slave*:

  Depth 1:

  slave 1 *scanning* mulai *index* 0-15

  slave 2 *scanning* mulai *index* 16-31

  slave 3 *scanning* mulai *index* 32-47

  slave 4 *scanning* mulai *index* 48-63

  Depth 2:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

  slave 3 *scanning* mulai *index* 0-63

  slave 4 *scanning* mulai *index* 0-63

- 4 *depth* dengan 2 *slave*:

  Depth 1:

  slave 1 *scanning* mulai *index* 0-31

  slave 2 *scanning* mulai *index* 32-63

  Depth 2:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

  Depth 3:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

  Depth 4:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

- 4 *depth* dengan 4 *slave*:

  Depth 1:

  slave 1 *scanning* mulai *index* 0-15

  slave 2 *scanning* mulai *index* 16-31

  slave 3 *scanning* mulai *index* 32-47

  slave 4 *scanning* mulai *index* 48-63

  Depth 2:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

  slave 3 *scanning* mulai *index* 0-63

slave 4 *scanning* mulai *index* 0-63

Depth 3:

slave 1 *scanning* mulai *index* 0-63

slave 2 *scanning* mulai *index* 0-63

slave 3 *scanning* mulai *index* 0-63

slave 4 *scanning* mulai *index* 0-63

Depth 4:

slave 1 *scanning* mulai *index* 0-63

slave 2 *scanning* mulai *index* 0-63

slave 3 *scanning* mulai *index* 0-63

slave 4 *scanning* mulai *index* 0-63

- 6 *depth* dengan 2 *slave*:

  Depth 1:

  slave 1 *scanning* mulai *index* 0-31

  slave 2 *scanning* mulai *index* 32-63

  Depth 2:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

  Depth 3:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

  Depth 4:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

  Depth 5:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

  Depth 6:

  slave 1 *scanning* mulai *index* 0-63

  slave 2 *scanning* mulai *index* 0-63

- 6 *depth* dengan 4 *slave*:

  Depth 1:

  slave 1 *scanning* mulai *index* 0-15

  slave 2 *scanning* mulai *index* 16-31

slave 3 *scanning* mulai *index* 32-47

slave 4 *scanning* mulai *index* 48-63

Depth 2:

slave 1 *scanning* mulai *index* 0-63

slave 2 *scanning* mulai *index* 0-63

slave 3 *scanning* mulai *index* 0-63

slave 4 *scanning* mulai *index* 0-63

Depth 3:

slave 1 *scanning* mulai *index* 0-63

slave 2 *scanning* mulai *index* 0-63

slave 3 *scanning* mulai *index* 0-63

slave 4 *scanning* mulai *index* 0-63

Depth 4:

slave 1 *scanning* mulai *index* 0-63

slave 2 *scanning* mulai *index* 0-63

slave 3 *scanning* mulai *index* 0-63

slave 4 *scanning* mulai *index* 0-63

Depth 5:

slave 1 *scanning* mulai *index* 0-63

slave 2 *scanning* mulai *index* 0-63

slave 3 *scanning* mulai *index* 0-63

slave 4 *scanning* mulai *index* 0-63

Depth 6:

slave 1 *scanning* mulai *index* 0-63

slave 2 *scanning* mulai *index* 0-63

slave 3 *scanning* mulai *index* 0-63

slave 4 *scanning* mulai *index* 0-63

4.1.2 Cara Kerja Aplikasi

Cara kerja aplikasi ini dimulai dengan masukan gerakan dari pemain manusia. Dalam program ini dibatasi pemain manusia hanya bisa bermain sebagai putih. Setelah papan selesai dinisialisasi, Maka Putih bisa langsung memulai permainan dengan menggerakkan bidak putih dengan cara menuliskan koordinat awal dan koordinat akhir bidak putih ke petak tujuan di dalam *command prompt*.

Selanjutnya gerakan tersebut akan divalidasi oleh fungsi didalam program, apakah sudah sesuai peraturan atau belum (contoh: tidak ada bidak yang bisa melompati bidak lain kecuali kuda). Setelah itu dilakukan proses pengecekan apakah ada bidak teman (*allied pieces*) di petak tujuan tadi.

Setelah proses ini selesai, giliran berpindah pada Hitam (computer). Pada saat giliran berpindah pada hitam akan dijalankan fungsi untuk menemukan seluruh gerakan yang mungkin untuk hitam pada keadaan papan saat ini (karena kondisi papan berubah setelah Putih melakukan gerakan tadi). Lalu seluruh gerakan tadi disimpan kedalam sebuah variabel board (history_board).

Setelah seluruh gerakan yang mungkin selesai disimpan, akan dilakukan proses evaluasi algoritma Shannon Type-A untuk setiap gerakan tersebut. Setelah semua gerakan selesai dievaluasi akan dipilih gerakan yang akan menghasilkan nilai terbaik untuk hitam (paling menguntungkan). Lalu gerakan tersebut akan dieksekusi dengan cara yang sama seperti Putih.

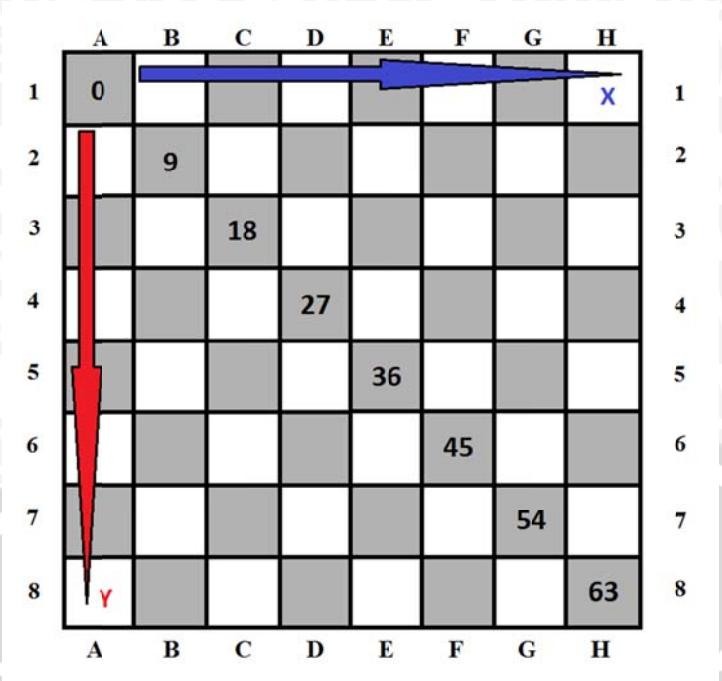
Setelah gerakan selesai dilakukan, giliran akan berpindah pada Putih untuk bermain lagi. Hal ini akan terus diulangi hingga terjadi *checkmate* (skakmat).

4.2 Perancangan Perangkat Lunak

Pada sub-bab ini akan dibahas lebih detil tentang perancangan proses-proses dasar pada aplikasi algoritma pencarian Shannon Type-A pada program permainan catur. Seperti yang telah dibahas pada sub-bab sebelumnya, proses ini terdiri dari tiga bagian.

4.2.1 Board Representation

Board Representation yang digunakan pada program ini adalah array 1 dimensi berukuran 64 (0-63). Dikarenakan jumlahnya yang sama dengan jumlah petak (squares) pada papan catur.



Gambar 4.3 *Board Representation*

Pada gambar diatas, array dimulai dari pojok kiri atas, dengan berdasar pada sistem koordinat Cartesian versi pemrograman. Yaitu *index* pertama atau [0] berada pada koordinat A,1 dan *index* terakhir atau [63] berada pada koordinat H,8.

Gambar 4.4 *Flowchart Board Representation*

Langkah selanjutnya adalah menetapkan nilai dan ID untuk setiap bidak, yang nantinya akan digunakan untuk evaluasi dengan langkah sebagai berikut:

```
// PIECE_VALUE
#define V_NULL          0           //value of null space
#define V_PAWN          100         //value of pawn
#define V_KNIGHT        300         //value of knight
#define V_BISHOP        325         //value of bishop
#define V_ROOK          500         //value of rook
#define V_QUEEN         900         //value of queen
#define V_KING          30000       //value of king
#define V_MAX           900000      //maximum value of minimax


// PIECE_CODE
#define EMPTY           0
#define PAWN            1
#define KNIGHT          2
#define BISHOP          3
#define ROOK            4
#define QUEEN           5
#define KING            6

#define MASK_COLOR      8       //1000
#define MASK_PIECE      7       //0111

#define COLOR_WHITE     0       //0000
#define COLOR_BLACK     8       //1000

/*
    piece = COLOR_X or PIECE_CODE
    1  = white pawn
    9  = black pawn
    10 = black knight
    dst
*/
```

Pada awal permainan setiap elemen array akan diisi seluruhnya oleh fungsi inisialisasi dengan langkah sebagai berikut:

1. Isi seluruh elemen array, dari index pertama sampai *index* terakhir dengan ID setiap bidak dan petak kosong (bukan *null*). Petak kosong bernilai 0.
2. Rubah nilai dari *index* [0] sampai *index* [14] dengan nilai Bidak Putih. Nilai untuk bidak putih yaitu: b, c, d, e, f, g dengan urutan b merupakan White Pawn dan g merupakan White King. Nilai ini akan dimasukkan sesuai dengan posisi awal dari catur.
3. Rubah nilai dari *index* [48] sampai *index* [63] dengan nilai Bidak Hitam. Nilai untuk bidak hitam yaitu: j, k, l, m, n, o dengan urutan j merupakan Black Pawn dan o merupakan Black King. Nilai ini akan dimasukkan sesuai dengan posisi awal dari catur.

Setelah semua nilai selesai dinisialisasi, maka akan ditampilkan gambar dasar papan beserta seluruh bidaknya.

4.2.2 Move Generation

Move Generation berlaku baik untuk Putih maupun Hitam. Proses ini bertujuan untuk memeriksa legalitas dari gerakan yang akan dilakukan. Legalitas yang dimaksud adalah bisa atau tidaknya suatu bidak akan bergerak pada petak tujuan sesuai dengan peraturan yang berlaku pada catur.

Dalam program ini Move Generation akan dipecah menjadi satu peraturan global, dan sisanya adalah peraturan untuk setiap jenis bidak, dengan rincian:

1. Peraturan Global

   Terdiri dari dua bagian yaitu:

a. Gerakan tidak akan bisa dilakukan jika petak tujuan sudah ditempati oleh bidak dengan warna yang sama.

b. Jalan menuju petak tujuan sudah ditempati oleh bidak dengan warna yang sama (*allied pieces*). Dengan pengecualian untuk Kuda yang bisa "melompat".

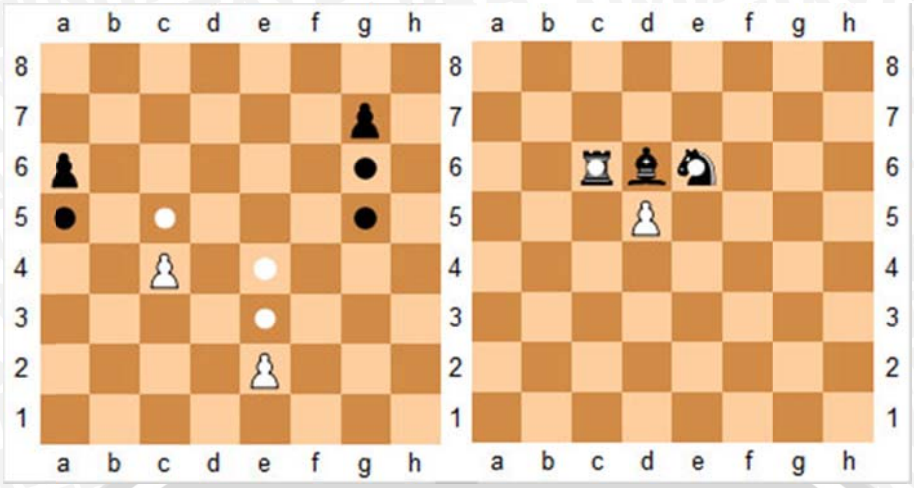
   Kedua hal tersebut berlaku untuk semua bidak tanpa kecuali walaupun sudah memenuhi peraturan individual untuk semua jenis bidak.

2. Pawn (Pion)

   Pawn hanya bisa bergerak maju satu langkah, namun jika Pawn belum bergerak (masih berada di posisi awal) maka Pawn bisa bergerak maju sebanyak dua langkah.

   Pawn bisa menangkap (*capturing*) bidak lawan dengan cara bergerak maju secara diagonal sebanyak satu petak. Pawn juga mempunyai gerakan spesial yang dinamakan "en passant".

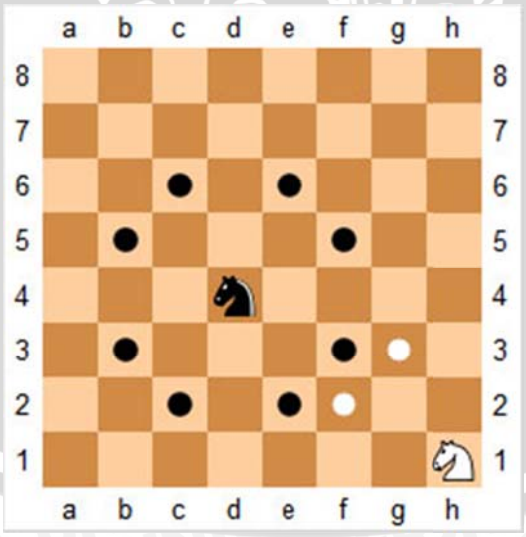
   Pawn merupakan satu-satunya bidak yang cara menangkap bidak lawan tidak sama dengan cara pergerakannya.



Gambar 4.5 *Pawn Movements and Captures*
Sumber: Wikipedia

3. Knight (Kuda/Ksatria)

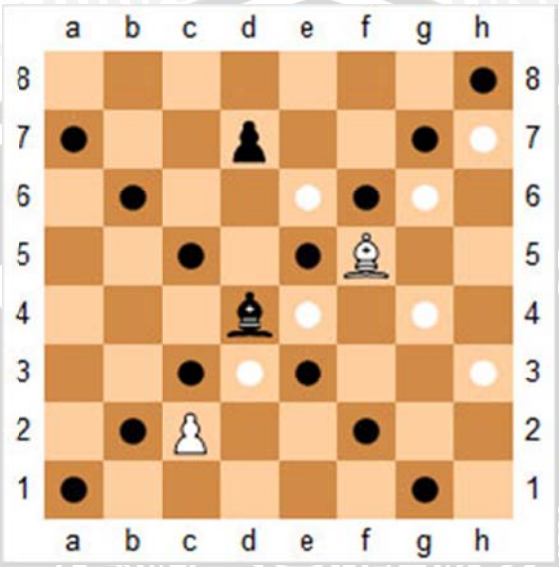
Gerakan dari Kuda sangat berbeda dengan pion lain. Kuda bisa bergerak dua petak horizontal dan satu petak vertikal atau satu petak horizontal dan dua petak vertikal sehingga terlihat seperti huruf "L". Dan juga Kuda bisa "melompati" bidak lain (untuk semua warna) untuk menuju ke petak tujuan.



Gambar 4.6 *Knight Movements and Captures*
Sumber: Wikipedia

4. Bishop (Gajah/Uskup)

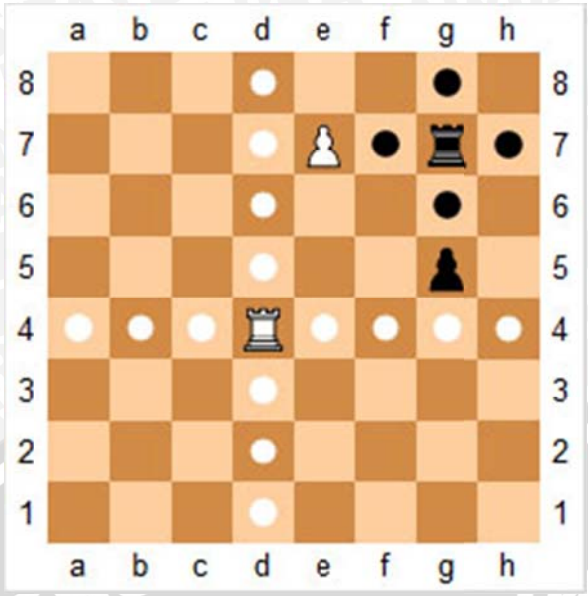
Bishop tidak mempunyai batasan jarak untuk setiap pergerakannya, tetapi dibatasi oleh gerakannya yang diagonal. Bishop, seperti bidak lainnya kecuali Kuda, tidak bisa melompati bidak lain. Bishop menangkap pion lawan dengan menempati petak dimana lokasi pion lawan berada.



Gambar 4.7 *Bishop Movements and Captures*
Sumber: Wikipedia

5. Rook (Benteng)

Rook bergerak secara horizontal atau vertikal, melewati berapa saja petak yang kosong. Seperti dengan bidak lain, Rook menangkap pion lawan dengan menempati petak dimana lokasi pion lawan berada. Rook juga bisa melakukan gerakan khusus dengan King, yang disebut *castling* (rokade).

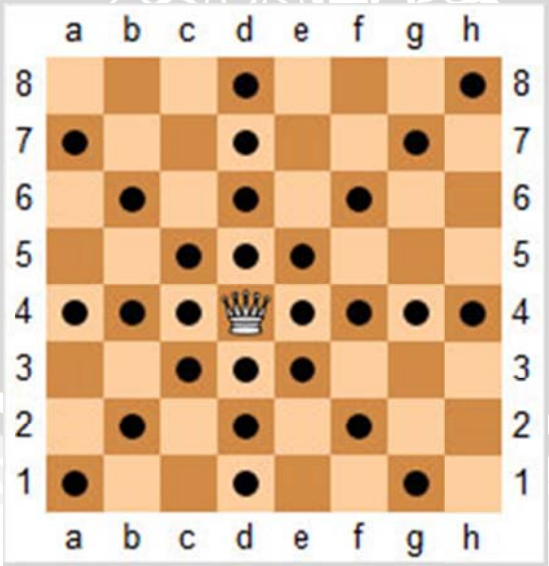


Gambar 4.8 *Rook Movements and Captures*

Sumber: Wikipedia

6. Queen (Ratu/Menteri)

Ratu bisa bergerak secara horizontal, vertikal maupun diagonal melalui berapa saja petak yang kosong, seperti gabungan antara Rook dan Bishop. Queen menangkap pion lawan dengan menempati petak dimana lokasi pion lawan berada.

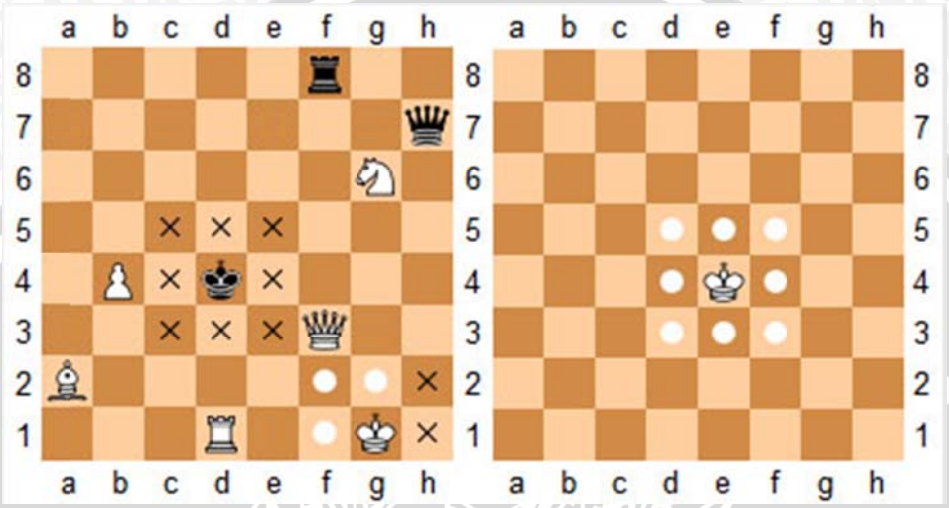


Gambar 4.9 *Queen Movements and Captures*

Sumber: Wikipedia

7. King (Raja)

King bisa bergerak satu petak pada arah mana saja (horizontal, vertikal, atau diagonal), kecuali jika petak tujuan ditempati oleh bidak lain dengan warna yang sama atau jika gerakan tersebut akan mengakibatkan posisi *check*. King juga bisa melakukan gerakan khusus dengan Rook, yang disebut *castling* (rokade).



Gambar 4.10 *King Movements and Captures*
Sumber: Wikipedia

4.2.3 Evaluation

Setiap gerakan yang mungkin harus dianalisa dan dinilai berdasarkan perubahan papan yang terjadi akibat langkah tersebut. Setelah seluruh evaluasi selesai, dipilih gerakan yang paling bagus, yaitu gerakan yang akan menghasilkan hasil evaluasi paling tinggi.

Proses Evaluasi akan dijalankan saat *turn* (giliran) berpindah ke komputer. Komputer akan mengevaluasi papan berdasarkan kondisi saat ini (yaitu setelah pemain sebelumnya *selesai* melakukan gerakan).

Jika salah satu raja sudah di*capture*, maka permainan akan berakhir. Dan pemain selanjutnya (baik manusia ataupun komputer) tidak akan mendapat giliran. Berikut potongan listing program yang menyatakan giliran dan eksekusi gerakan:

```
while(!gameover)
	{
		if(ply%2==0)//white one
		{
			//input
```

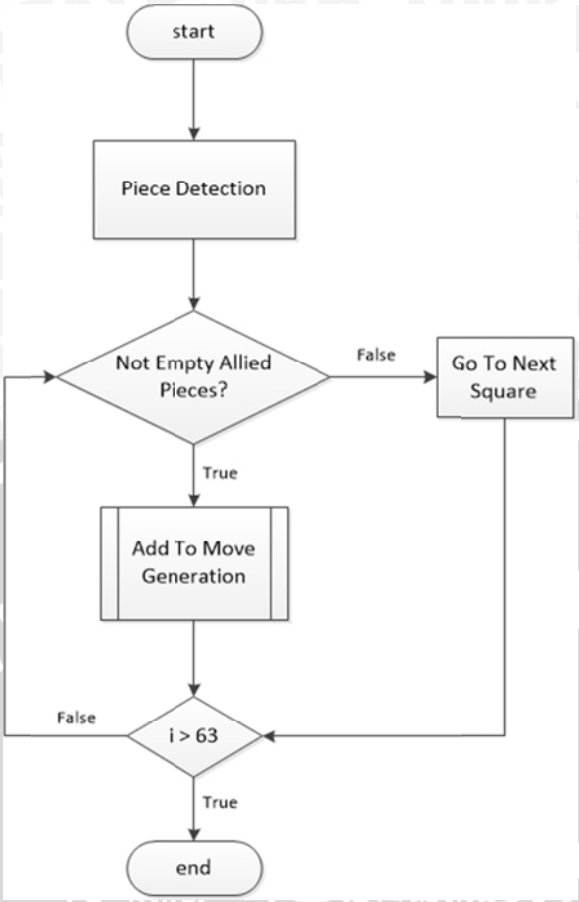
```
                //check
                //if(check)
                //-execute
                //-ply++
                display(board);
                printf("\n input your move ");
                scanf("%s",&c1);
                input_return=check_input(c1);
                //check jika masukan valid atau tidak
                if(input_return<0)
                {
                        //input_return -1 berarti tidak valid
                printf("\nwrong input, try again\n\n");
                }
        else
                {
                       //jika valid, maka Dekode input_return
                tpos=input_return>>6;
                tdes=input_return&63;
                printf("\nmove from %d to %d\n",tpos,tdes);
                if(check_move_player(tpos,tdes))
                {
                move_player_piece(tpos,tdes);//eksekusi gerakan manusia
                ply++;
                }
        else
                {
              printf("\nOops!! illegal move, try again\n\n");
                }
                }

        }
        else          //computer's turn
        {
                if(ply<3)   //opening book
                {
                move_piece(52,-10);                     //eksekusi gerakan book
                }
                else
                {
                copy_board(tboard,board);

                maxi(2,false);
                move_piece(start_pos_move, des_pos_move); //eksekusi gerakan
                komputer dengan minimax
                //printf("\nbest move %d %d\n",start_pos_move,des_pos_move);

                display(board);
                //printf("\nvalue %d\n",evaluate());
                }
                ply++;
        }
        winner=check_winning_player();
        if(winner!=0)
        {
        gameover=true;
        if(winner==1)
        {
          printf("\n\nwinner is WHITE\n");
        }
        else
        {
          printf("\n\nwinner is BLACK\n");
        }
        }
}
```

Gambar 4.11 *Flowchart Move Generation, Evaluation,* dan *Execution*



Gambar 4.12 *Scanning* Bidak

1. **Pengecekan seluruh bidak dengan warna yang sama.**

   Pengecekan ini dilakukan dengan tujuan untuk menemukan bidak apa saja yang berwarna sama dengan warna pemain saat ini beserta lokasinya. Hal ini dikarenakan kita hanya bisa menggerakkan bidak milik kita sendiri (yang berwarna sama).

2. **Pengecekan apakah bidak yang ditemukan bisa bergerak.**

   Proses ini dilakukan untuk memeriksa apakah bidak yang ditemukan bisa bergerak atau tidak, karena walaupun bidak tersebut adalah milik pemain tersebut, belum tentu bidak tersebut bisa bergerak karena bisa saja bidak tersebut tertutup (*blocked*) oleh bidak lain.

3. **Proyeksi gerakan yang sudah bisa dilakukan.**

   Setelah seluruh gerakan yang legal untuk dilakukan selesai disimpan kedalam variabel dalam proses selanjutnya, akan dilakukan proses proyeksi gerakan untuk mencari gerakan mana yang akan paling

menguntungkan, dalam hal ini akan menghasilkan nilai terbaik dengan menggunakan algoritma Shannon Type-A.

4. Penerapan Algoritma Shannon Type-A.

Penerapan algoritma ini bisa dianalogikan dengan tiga langkah dasar Minimax berikut:

a) Saya akan mencoba melakukan gerakan ini, dan akan melihat bagaimana kira-kira respon dari lawan saya.
b) Jika gerakan tadi sudah saya lakukan, saya sekarang akan berpikir seolah-olah saya adalah lawan saya.
c) Dalam keadaan saat ini saya akan memilih gerakan yang paling menguntungkan untuk saya.

Dari ketiga langkah tersebut, setelah langkah c) dilakukan maka akan diketahui:

"*Jika saya jadi melakukan langkah tadi, maka saya akan tahu hal terburuk yang akan terjadi sebagai akibat dari langkah saya tersebut*".

Sehingga isi dari fungsi ini adalah prediksi dari langkah yang akan diambil oleh lawan.

5. Perhitungan dan pemilihan nilai akhir.

Setelah semua selesai divalidasi, maka akan dihitung nilai dari hasil pergerakan tersebut dan dibandingkan dengan nilai dari hasil gerakan bidak lain yang memungkinkan untuk bergerak pada giliran pemain saat ini. Kemudian, dipilih gerakan dengan nilai terbaik yang menguntungkan untuk giliran pemain saat ini.

6. Eksekusi gerakan.

Setelah mendapatkan nilai terbaik maka akan dilakukan proses terakhir, yaitu perpindahan sesungguhnya dari proyeksi gerakan bidak dengan nilai terbaik. Setelah dijalankan, maka giliran akan berpindah ke pemain

selanjutnya. Perulangan ini akan terus berlanjut hingga tiba saatnya permainan berakhir.

Berikut listing program dari proses 1 sampai 6:

```
if ( depth <= 0 ) return evaluate();
        if ( maximize )
                {
                //printf("depth %d\n ",depth);
                max = -V_MAX;

                copy_board( history_board[depth], tboard);//record history (of three
                kingdom)
                for ( i=0; i<64; i++ )
                        {
                        //scan board
                        lpiece = tboard[i] & MASK_PIECE;    //pengecekan warna putih
                        lcolor = tboard[i] & MASK_COLOR;
                        if(( lpiece != EMPTY) && ( lcolor == COLOR_WHITE))
                        //apakah warnanya putih?
                        {
                                if(lpiece==PAWN) //apakah pion? (pion diistimewakan
                karena memiliki gerakan yang berbeda dengan bidak lainya)
                                {
                                //move generation untuk pion
                                        for ( n_move = 0; n_move < 3; n_move++ )
                                        {
                                                des_move = movement[lpiece][n_move];
                                /* des_move = -movement[lpiece][n_move]; if its BLACK */
                                                if(check_pawn_move(i,des_move))
                                                //mengecek gerakan pawn putih
                                                {
                                                        move_pseudo_pawn(i,des_move);
                                                        score = maxi( depth - 1, false );
                                                        if(score>max)  //membandingkan
                nilai terbaik untuk maximum
                                                                {
                                                                        max=score;
                                                                }
                                                copy_board(tboard,history_board[depth]);
                                                }
                                        }

                                }else          //jika bukan pion
                                {
                                //move generation

                                for ( n_move = 0; n_move < number_of_move[lpiece];
                                n_move++ )
                                        {
                                                des_move=0;
                                                do
                                                {

                                des_move=des_move+movement[lpiece][n_move];
                                                        if(!check_move(i,des_move))
                                                        //mengecek gerakan bidak putih
                                                        {
                                                                break;
                                                        }
```

```
                                        if(check_target_color(i,des_move))
                                        //if can captured
                                        {

                                    move_pseudo_piece(i,des_move);

                                    score = maxi( depth - 1, false );
                                                if(score>max)
                                                {
                                                        max=score;

                                                }

                                    copy_board(tboard, history_board[depth]);
                                    //return board
                                                break; //stop
                                        }
                                        // if empty
                                        move_pseudo_piece(i,des_move);
                                        score = maxi( depth - 1, false );
                                        if(score>max)
                                                {
                                                        max=score;

                                                }

                                        copy_board(tboard, history_board[depth]);
                                                //return board

                                        }while(slide[lpiece]);

                                }
                        }
                }
        }

        return max;
}
else    //minimum
{
//sama dengan maximum, tapi yang digerakan bidak hitam dan nilai minimum
max = V_MAX;

        copy_board( history_board[depth], tboard); //record history (of three
        kingdom)
        for ( i=0; i<64; i++ ) { //scan board
                lpiece = tboard[i] & MASK_PIECE;//pengecekan warna hitam
                lcolor = tboard[i] & MASK_COLOR;
                if(( lpiece != EMPTY) && ( lcolor == COLOR_BLACK))
                //apakah tidak kosong dan warnanya hitam?
                {
                        if(lpiece==PAWN)
                        {
                                for ( n_move = 0; n_move < 3; n_move++ )
                                {
                                        des_move = -movement[lpiece][n_move];
                                /* des_move = -movement[lpiece][n_move]; if its BLACK */
                                        if(check_pawn_move(i,des_move))
                                        //mengecek gerakan pawn hitam
                                        {
                                                move_pseudo_pawn(i,des_move);
                                                score = maxi( depth - 1, true );
                                                        if(score<max)//jika nilai
                        lebih baik(lebih kecil) maka simpan sementara informasi gerakan
                                                        {
                                                                max=score;
```

```
                                        lbestpos=i;
                                        lbestmove=des_move;
                                }

                        copy_board(tboard, history_board[depth]);
                        }
                }

        }
        else
        {

        for ( n_move = 0; n_move < number_of_move[lpiece];
        n_move++ )
                {
                        des_move=0;
                        do
                        {

                des_move=des_move+movement[lpiece][n_move];
                                if(!check_move(i,des_move))
                                //mengecek gerakan bidak hitam
                                {
                                        break;
                                }

                                if(check_target_color(i,des_move))
                                //if can captured
                                {

                                move_pseudo_piece(i,des_move);

                                score = maxi( depth - 1, true );
                                        if(score<max)
                                        {
                                                max=score;
                                                lbestpos=i;
                                                lbestmove=des_move;
                                        }

                        copy_board(tboard, history_board[depth]);
                        //return board
                                        break;
                                //stop
                                }

                                move_pseudo_piece(i,des_move);
                                score = maxi( depth - 1, true );
                                if(score<max)
                                        {
                                                max=score;
                                                lbestpos=i;
                                                lbestmove=des_move;
                                        }
                        copy_board(tboard, history_board[depth]);
                        //return board


                        }while(slide[lpiece]);

                }
        }
                }
        }
//informasi gerakan terakhir adalah gerakan yang terbaik
 start_pos_move=lbestpos;
```

```
            des_pos_move=lbestmove;
                return max;
            }
}
```

4.3 Konfigurasi Perangkat Keras

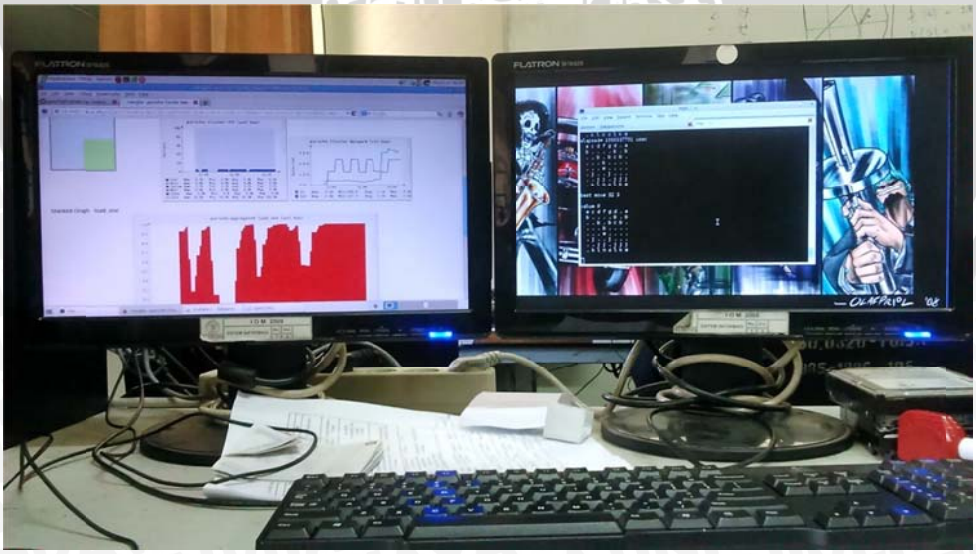
Sistem komputasi paralel menggunakan 5 komputer: 1 *front-end*, 4 komputer slave. Jaringan area lokal sistem komputasi paralel menggunakan *switch* TL-SG1024D dengan perkabelan Cat 5e.

4.3.1 Konfigurasi Perangkat Keras Komputer *Front-end*

Komputer *front-end* adalah komputer tunggal yang digunakan sebagai antarmuka pengguna dan sistem komputasi paralel.

Tabel 4.1 Perangkat Keras Komputer *Front-end*

Prosesor	Intel Pentium E7400 (2M Cache, 2.70 GHz, 800 MHz FSB)
Board	BIOSTAR G31-M7 TE
Memori	2 x 1GB DDR2 6400
Storage	Seagate ST3500414CS Seagate ST3250318AS
NIC	Realtek Semiconductor Co., Ltd. RTL8101E/RTL8102E PCI Express Fast Ethernet controller (rev 02)



Gambar 4.13 Unit Komputer *Front-end*

4.3.2 Konfigurasi Perangkat Keras Komputer *Slave*

Komputer yang digunakan sebagai komputer *slave* berjumlah 4 unit dan digunakan untuk semua tahap pengujian, 4 unit menggunakan prosesor Intel Core i3-550. Media *boot* yang digunakan adalah flashdrive dengan ukuran nominal 4 GB.

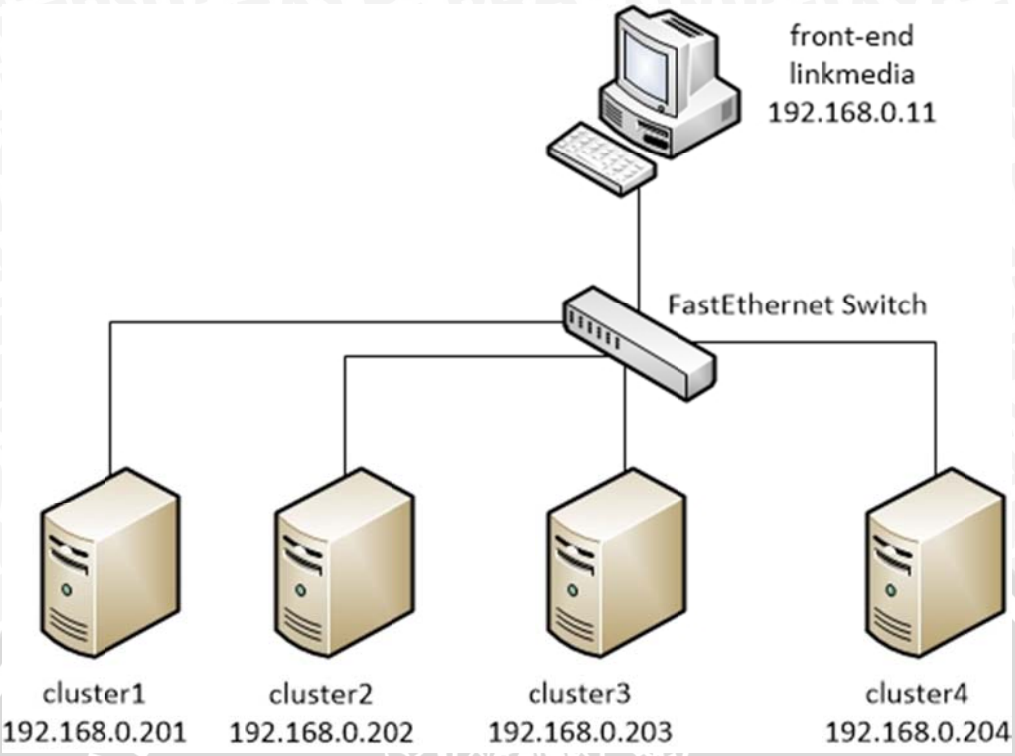
Tabel 4.2 Perangkat Keras Komputer *Slave*

Prosesor	Intel Core i3 550 (4M Cache, 3.20 GHz)
Board	Intel Desktop Board DH55PJ
Memori	2GB DDR3 10600
Storage	ADATA DashDrive UV100 4GB
NIC	Intel Corporation 82578DC Gigabit Network Connection (rev 06)

Gambar 4.14 Empat Unit Komputer *Slave*

4.3.3 Konfigurasi Jaringan Lokal

Jaringan sistem komputasi paralel menggunakan jaringan lokal dengan topologi bintang. NIC dari tiap komputer terhubung langsung dengan *switch* TL-SG1024D melalui kabel UTP dengan kategori 5e. Konfigurasi pengalamatan jaringan lokal menggunakan IPv4 dengan lingkup jaringan 192.168.0.0/24. Gambar 4.1.3 memperlihatkan topologi jaringan, asosasi nama *host* dan alamat IP.



Gambar 4.15 Jaringan Lokal Sistem Komputasi Paralel

4.4 Konfigurasi Perangkat Lunak

Konfigurasi perangkat lunak sistem komputasi paralel melingkupi konfigurasi jaringan, *environment* sistem operasi, *filesystem* terdistribusi, kompilasi dan instalasi paket program dependensi, dan implementasi program komputasi paralel.

4.4.1 Konfigurasi Komputer *Front-end* (E7400)

Komputer *front-end* bertanggung jawab atas distribusi berkas *executable* program melalui *filesystem* terdistribusi (NFS), kompilasi (mpicc) dan inisialisasi eksekusi (mpirun) program komputasi paralel.

4.4.1.1 Konfigurasi Alamat IP dan Host Komputer *Front-end*

Front-end menggunakan sistem operasi Salix64 14.1 dengan gcc 4.8.2, glibc 2.17, dan kernel Linux 3.10.17 64-bit. Paket-paket dependensi: gcc, nfsutils, openssh merupakan telah terpasang sejak instalasi *default.*

Front-end menggunakan nama *host* linkmedia dengan alamat IP 192.168.0.11. Sistem operasi Salix64 menggunakan berkas `/etc/hosts` untuk mengasosiasikan nama *host* dan alamat IP. Semua komputer dalam sistem komputasi paralel menggunakan

berkas `/etc/hosts` yang berisi semua entri semua *hosts* dalam sistem komputasi paralel agar dapat mengenali *hosts* lainnya.

Pengaturan alamat IP *front-end* dilakukan dengan menambahkan entri:

```
IPADDR[0]="192.168.0.11"
NETMASK[0]="255.255.255.0"
```

ke berkas `/etc/rc.d/rc.inet1.conf`.op

Pengaturan nama *host* dilakukan dengan penambahan berkas `/etc/HOSTNAME` dengan entri: `linkmedia.link.elektro.ub.ac.id`

4.4.1.2 Konfigurasi User mpi

Sebuah *user* identik diperlukan untuk menjalankan Open MPI di beberapa *user* sekaligus, hal ini mengharuskan adanya instans *user* dengan UID (*user id)* dan GID (*group id*) yang identik di semua *hosts* sistem komputasi paralel. Untuk menambahkan *user* dengan nama mpi dan sandi lewat mpi, berkas `/etc/passwd` ditambahkan entri:

```
mpi:$5$CUgEJj9IK3$/txgiLKoRWmB2VOVQ
/Sh4OsxEMIR0qnYByJcXOtJoK1:2000:2000::/home/mpi:/bin/bash
```

dan berkas /etc/group ditambahkan entri:

```
mpi:x:2000:
```

User mpi perlu memilki direktori *home* agar berbagai berkas dapat disimpan. Konfigurasi yang dilakukan:

```
# mkdir /home/mpi
# chmod 777 /home/mpi
# chown mpi:mpi /home/mpi
```

4.4.1.3 Konfigurasi Server NFS

Distribusi kunci publik SSH dan program komputasi paralel menggunakan NFS (Network File System) sebagai media. Direktori yang di-*export* melalui NFS akan dapat di-*mount* di *node*.

Penambahan direktori *home user* mpi (`/home/mpi`) dengan NFS di *front-end* yang bertindak sebagai server dilakukan dengan pengubahan berkas `/etc/exports` dengan entri:

```
/home/mpi 192.168.0.0/255.255.255.0(rw)
```

Pengaktifan *service* NFS dilakukan dengan cara memberikan moderasi *executable* terhadap berkas `/etc/rc.d/rc.rpc` dan `/etc/rc.d/rc.nfsd`:

```
# chmod +x /etc/rc.d/rc.rpc
# chmod +x /etc/rc.d/rc.nfsd
```

4.4.1.4 Konfigurasi SSH

Eksekusi Open MPI (mpirun) membutuhkan login *remote* dengan SSH di *node* tanpa *prompt* sandi lewat. Hal ini dilakukan dengan penambahan entri di berkas `/home/mpi/.ssh/authrorized_keys` dengan kunci publik *user* mpi di *front-end*.

```
$ cp ~/.ssh/id_ecdsa.pub ~/.ssh/authorized_keys
```

4.4.1.5 Kompilasi dan Instalasi OpenMPI Komputer *Front-end*

Program Open MPI harus dikompilasi dan dipasang pada *path* yang sesuai di semua komputer sistem komputasi paralel. Langkah konfigurasi dan instalasi adalah sebagai berikut:

```
# wget http://www.open-
  mpi.org/software/ompi/v1.8/downloads/openmpi-1.8.1.tar.bz2
# tar -jxvf openmpi-1.8.1.tar.bz2 -C /tmp
# cd /tmp/openmpi-1.8.1
# ./configure && make && make install
# ldconfig
```

Proses instalasi telah berhasil jika mpirun dan mpicc dikenali dalam *path* sistem dan pustaka yang berkaitan berhasil dimuat.

4.4.1.6 Instalasi Ganglia Monitoring System

Sebelum melakukan instalasi Ganglia, beberapa dependent package harus diinstal terlebih dahulu dengan perintah :

```
# yum -y install ganglia-gmond ganglia-gmetad ganglia
```

Dependent package yang lainnya adalah rrdtool dan confuse yang akan dibahas pada langkah – langkah instalasi.

Selanjutnya dilakukan langkah instalasi sebagai berikut:

- Membuat user ganglia yang akan kita jalankan

```
# useradd ganglia
# password ganglia
```

- Membuat direktori dimana kita akan mendownload ganglia

```
# mkdir /usr/local/src
# cd /usr/local/src
```

- Install semua file yang diperlukan untuk confuse

```
# wget http://savannah.nongnu.org/download/confuse/confuse-
2.7.tar.gz
# tar -xzvf confuse-2.7.tar.gz
# cd confuse-2.7
# ./configure
# make
# make install
```

- Install semua file yang diperlukan untuk rrdtool

```
# cd /usr/local/src
# wget http://oss.oetiker.ch/rrdtool/pub/rrdtool/.tar.gz
# tar -xzvf rrdtool.tar.gz
# cd rrdtool-1.4.8/
# ./configure --prefix=/usr
# make -j8
# make install
# which rrdtool
```

- Memastikan semua package yang telah diinstall mempunyai library yang saling terhubung dengan menggunakan perintah *ldconfig*

```
# vi /etc/ld.so.conf
#    /usr/local/lib
# ldconfig
```

- Setelah semua dependent package telah diinstall, kemudian install semua file yang diperlukan untuk ganglia-core

```
# cd /usr/local/src
# wget
http://sourceforge.net/projects/ganglia/files/ganglia%20monto
ring%20core/3.6.0.ganglia-3.6.0.tar.gz
# tar -xzvf ganglia-3.6.0.tar.gz
# ./configure --with-gmetad
# make -j8
# make install
```

4.4.1.7 Konfigurasi Ganglia Monitoring System

Langkah konfigurasi Ganglia adalah sebagai berikut:

- Buat direktori konfigurasi untuk ganglia

```
# mkdir /etc/ganglia
```

- Salin file sampel konfigurasi gmetad (master)

```
# cp gmetad/gmetad.conf /etc/ganglia/
```

- Buat direktori penyimpanan untuk RRDTool

```
# mkdir -p /var/lib/ganglia/rrds
# chown ganglia:ganglia /var/lib/ganglia/rrds
```

- Mengubah beberapa parameter dalam file konfigurasi gmetad

```
# vi /etc/ganglia/gmetad.conf
   data_source “parsche cluster” localhost
   setuid_username “root”
   case_sensitive_hostname 1
```

4.4.2 Konfigurasi Komputer *Slave* (4 Unit i3 550)

Komputer *slave* adalah komputer yang digunakan di semua tahap pengujian sistem komputasi paralel, menggunakan alamat IP 192.168.0.201-192.168.0.204, mendapatkan berkas *executable* dan kunci publik SSH melalui direktori NFS. Sistem operasi yang digunakan adalah Salix64 14.0 dengan gcc 4.7.1, glibc 2.15, dan kernel Linux 3.2.29. Paket nfsutils dan openssh telah terpasang sejak instalasi *default.*

Komputer *slave* dikonfigurasi *headless* (tanpa layar dan perangkat I/O) sehingga penyalaan dilakukan dengan paket *wake-on-LAN*.

4.4.2.1 Konfigurasi Alamat IP dan Host Komputer *Slave*

Pengaturan alamat IP *node* dan nama *host* sama seperti pengaturan alamat IP dan nama *host front-end*.

Tabel 4.3 Asosiasi Nama *Host* dan Alamat IP Komputer *Slave*

Nama *Host*	Alamat IP
cluster1	192.168.0.201
cluster2	192.168.0.202
cluster3	192.168.0.203
cluster4	192.168.0.204

4.4.2.2 Konfigurasi User mpi

Konfigurasi *user* mpi di komputer *slave* sama dengan konfigurasi *user* mpi di *front-end.*

4.4.2.3 Konfigurasi Klien NFS

Komputer slave melakukan mounting direktori NFS dari *front-end* secara otomatis tiap *boot* dengan menambahkan entri ke berkas `/etc/fstab`:

```
192.168.0.11:/home/mpi /home/mpi nfs sloppy,rw 0 0
```

4.4.2.4 Konfigurasi SSH

Komputer *slave* mendapatkan kunci publik SSH secara otomatis setelah direktori home user mpi di-mount dan konfigurasi SSH di *front-end* telah dilakukan.

4.4.2.5 Instalasi OpenMPI Komputer *Slave*

Langkah konfigurasi dan instalasi adalah sebagai berikut:

```
# wget http://www.open-
  mpi.org/software/ompi/v1.8/downloads/openmpi-1.8.1.tar.bz2
# tar -jxvf openmpi-1.8.1.tar.bz2 -C /tmp
# cd /tmp/openmpi-1.8.1
# ./configure && make && make install
# ldconfig
```

Program mpirun dan mpicc akan dikenali dalam *path* sistem dan pustaka yang berkaitan akan dimuat jika program dijalankan.

4.4.2.6 Konfigurasi wake-on-LAN

Komputer *slave* dikonfigurasi *headless* sehingga tidak terdapat tombol untuk menyalakan unit komputer, sebagai gantinya *node* dinyalakan dengan paket *wake-on-LAN* dengan alamat MAC (*media access control*) yang bersesuaian.

Tabel 4.4 Asosiasi Nama *Host* dan Alamat MAC Komputer *Slave*

Nama *Host*	Alamat MAC
cluster1	70:71:bc:a3:9e:82
cluster2	70:71:bc:a3:9e:6f
cluster3	70:71:bc:a3:9e:7e
cluster4	70:71:bc:a3:9e:85

Langkah penyalaan komputer *slave* dilakukan dengan menjalankan skrip *shell* `wakeonlan.sh` sesuai dengan argumen alamat MAC komputer *slave* dari *front-end.*

4.4.2.7 Konfigurasi Boot

Komputer *slave* melakukan *boot* dari media *flashdrive*, hal ini menyebabkan kernel perlu menggunakan *initrd* (initial RAM disk). Konfigurasi *boot* dilakukan dengan melakukan instalasi paket kernel *generic* dan modul kernel, pembuatan *initrd* dan pengaturan *bootloader* lilo yang merupakan *bootloader* sistem operasi Salix64.

```
# installpkg kernel-generic-3.2.29-x86_64-1.txz
# cd /usr/share/mkinitrd/
# . mkinitrd_command_generator.sh -a “-w 10” > /boot/new.sh
# . /boot/new.sh
```

Initrd yang dihasilkan akan membuat kernel menunggu selama 10 detik sebelum melakukan *boot* ke partisi *root* karena perangkat *flashdrive* memerlukan waktu yang lebih lama untuk dikenali oleh kernel.

Konfigurasi *bootloader* diperlukan untuk mengatur parameter kernel agar *boot* dilakukan ke perangkat *flashdrive* yang telah dienumerasi menurut UUID oleh udev. Berkas `/etc/lilo.conf` ditambahkan dengan entri:

```
image = /boot/vmlinuz-generic
initrd = /boot/initrd.gz
```

```
label = salix
append = "root=/dev/disk/by-uuid/1bb9bf67-3acf-4ae1-be72-
9c10653b1f95"
read-only
```

Langkah terakhir konfigurasi *bootloader* adalah instansiasi *boot*loader pada MBR dengan lilo.

```
# lilo –v
```

4.4.2.8 Duplikasi Media *Boot* Komputer *Slave*

Duplikasi media *boot* komputer *slave* dilakukan dengan utilitas `dd`:

```
# dd if=/dev/sda of=/dev/sdb bs=4M
```

dengan `/dev/sda` adalah enumerasi *flashdrive* berisi sistem operasi yang telah dikonfigurasi dan `/dev/sdb` adalah enumerasi *flashdrive* yang belum dikonfigurasi.

Media *boot* yang hasil salinan hanya perlu mengulangi prosedur konfigurasi alamat IP dan *host* sesuai alamat IP dan nama *host* masing-masing.

4.4.2.9 Instalasi Ganglia Monitoring System

Sebelum melakukan instalasi Ganglia, beberapa dependent package harus diinstal terlebih dahulu dengan perintah :

```
# yum -y install ganglia-gmond ganglia-gmetad ganglia
```

Dependent package yang lainnya adalah rrdtool dan confuse yang akan dibahas pada langkah – langkah instalasi.

Selanjutnya dilakukan langkah instalasi sebagai berikut:

- Membuat user ganglia yang akan kita jalankan

```
# useradd ganglia
# password ganglia
```

- Membuat direktori dimana kita akan mendownload ganglia

```
# mkdir /usr/local/src
# cd /usr/local/src
```

- Install semua file yang diperlukan untuk confuse

```
# wget http://savannah.nongnu.org/download/confuse/confuse-
 2.7.tar.gz
# tar -xzvf confuse-2.7.tar.gz
# cd confuse-2.7
# ./configure
# make
# make install
```

- Install semua file yang diperlukan untuk rrdtool

```
# cd /usr/local/src
# wget http://oss.oetiker.ch/rrdtool/pub/rrdtool/.tar.gz
# tar -xzvf rrdtool.tar.gz
# cd rrdtool-1.4.8/
# ./configure --prefix=/usr
# make -j8
# make install
# which rrdtool
```

- Memastikan semua package yang telah diinstall mempunyai library yang saling terhubung dengan menggunakan perintah *ldconfig*

```
# vi /etc/ld.so.conf
#    /usr/local/lib
# ldconfig
```

- Setelah semua dependent package telah diinstall, kemudian install semua file yang diperlukan untuk ganglia-core

```
# cd /usr/local/src
# wget
  http://sourceforge.net/projects/ganglia/files/ganglia%20monit
  oring%20core/3.6.0.ganglia-3.6.0.tar.gz
# tar -xzvf ganglia-3.6.0.tar.gz
# ./configure --with-gmetad
# make -j8
# make install
```

4.4.2.10 Konfigurasi Ganglia Monitoring System

Langkah konfigurasi Ganglia adalah sebagai berikut:

- Buat direktori konfigurasi untuk ganglia

```
# mkdir /etc/ganglia
```

- Salin file sampel konfigurasi gmetad (master)

```
# cp gmetad/gmetad.conf /etc/ganglia/
```

- Membangkitkan file awal konfigurasi gmond

```
# gmond –t | tee /etc/ganglia/gmond.conf
```

- Buat direktori penyimpanan untuk RRDTool

```
# mkdir -p /var/lib/ganglia/rrds
# chown ganglia:ganglia /var/lib/ganglia/rrds
```

- Mengubah beberapa parameter dalam file konfigurasi gmetad

```
# vi /etc/ganglia/gmetad.conf
  data_source “parsche cluster” localhost
```

```
setuid_username “root”
case_sensitive_hostname 1
```

- Mengubah beberapa parameter dalam file konfigurasi gmond

```
# vi /etc/ganglia/gmond.conf
  user = ganglia
  cluster {
        name = "parsche"
        owner = "indraharis"
        latlong = "somewhere over the rainbow"
        url = "link.elektro.ub.ac.id"
  }

  udp_send_channel {
        host = localhost
        port = 8649
        ttl = 1
  }

  udp_recv_channel {
        port = 8649
  }

  tcp_accept_channel {
        port = 8649
  }
```

4.4.2.11 Membangun Ganglia Web

- Mendownload package

```
# cd /usr/local/src/
# wget http://sourceforge.net/projects/ganglia/files/ganglia-web/3.5.11/ganglia-web-3.5.11.tar.gz
# tar -xzvf ganglia-web-3.5.11.tar.gz
# cd ganglia-web-3.5.11
```

- Mengubah konfigurasi Makefile yang digunakan untuk membangun ganglia web

```
# vi Makefile
//location where gweb should be installed to (excluding conf, dwoo dirs)
# GDESTDIR = /var/www/html/ganglia
// Gweb statedir (where conf dir and Dwoo templates dir are stored)
# GWEB_STATEDIR = /var/lib/ganglia-web
// Gmetad rootdir (parent location of rrd folder)
# GMETAD_ROOTDIR = /var/lib/ganglia
// User by which your webserver is running
# APACHE_USER = apache
```

- Install Ganglia web

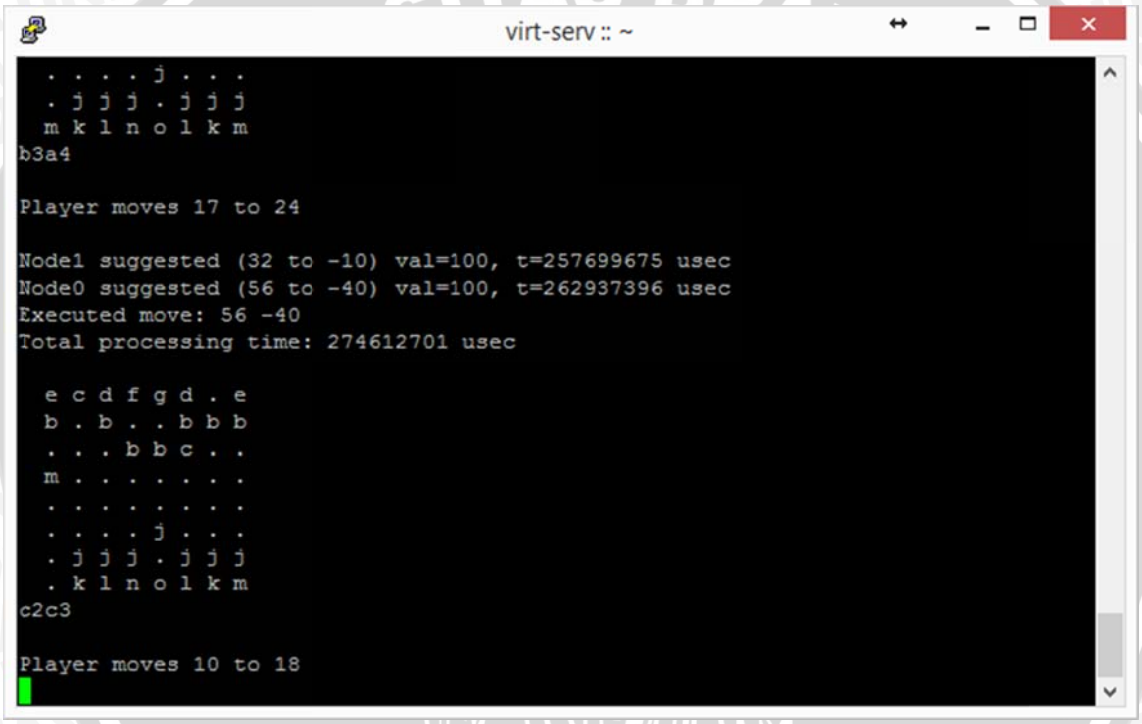
```
# make install
```

- Buka Ganglia Web User Interface di web browser
  http://link.elektro.ub.ac.id/ganglia

4.4.3 Implementasi Sistem

Implementasi sistem sesuai dengan perancangan perangkat lunak serta konfigurasi perangkat keras dan perangkat lunak yang telah dipersiapkan. Hasil konfigurasi tersebut antara lain :

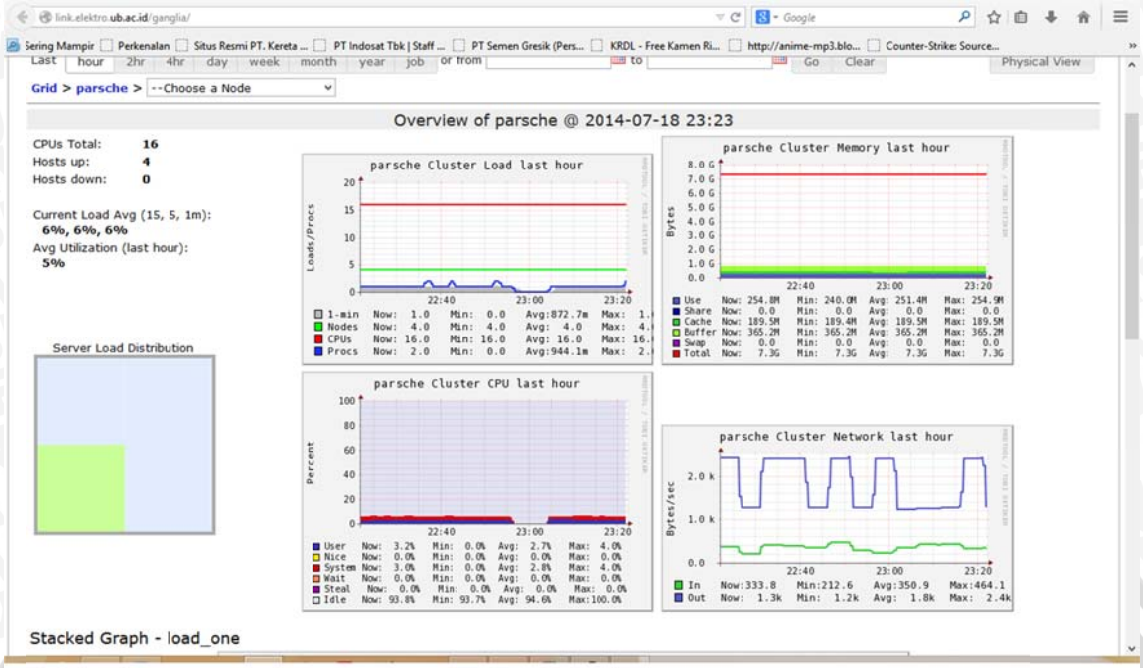
4.4.3.1 *Interface* Program Permainan Catur dalam *Command Prompt* dengan Open MPI



Gambar 4.16 Tampilan Program Permainan Catur

Sumber: Implementasi

4.4.3.2 *Monitoring* Menggunakan Ganglia Monitoring System



Gambar 4.17 Hasil installasi dan konfigurasi Ganglia Monitoring System

Sumber: Implementasi