BAB IV
PERANCANGAN DAN IMPLEMENTASI

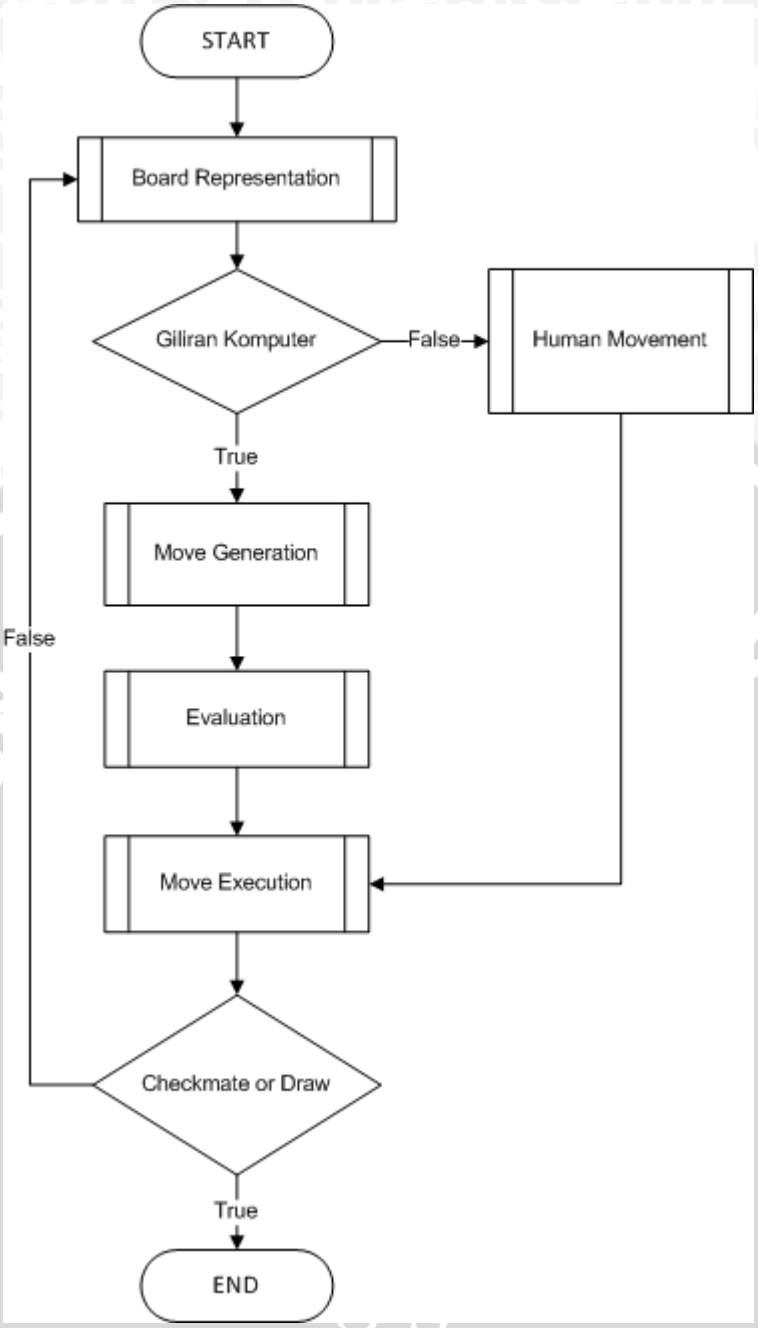
Pada bab ini akan dibahas tentang perancangan aplikasi permainan catur yang dikerjakan melalui 3 tahap dasar, yaitu tahap representasi papan, pembuatan dan pengecekan *move generation* dan tahap evaluasi serta pengambilan langkah. Pembuatan yang bertahap agar bisa dilakukan analisa pada setiap bagiannya 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 *Diagram Alir Proses Utama*

Fungsi dari masing – masing blok diagram adalah sebagai berikut :

1. *Board Representation*, yaitu pembuatan seluruh variabel yang nantinya akan digunakan, beserta tampilannya didalam antarmuka (contoh: a1 untuk *White Pawn*).
2. *Move Generation* yaitu memasukkan aturan-aturan yang berlaku pada catur kedalam program, baik pseudo-legal maupun legal. Setelah itu dilakukan proses *debugging* untuk tiap bidak apakah sudah memenuhi aturan atau tidak.
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.
4. *Move Execution* yaitu Eksekusi gerakan sesungguhnya, kemudian mengakhiri giliran.

4.1.2 Cara Kerja Aplikasi

Cara kerja aplikasi ini dimulai dengan input 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 *dragging* (menggeser) bidak putih ke petak tujuan.

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.

Jika seluruh pengecekan diatas sudah dilakukan dan semua syarat terpenuhi, maka dilakukan proses perpindahan bidak, yaitu dengan memasukkan nilai *null* pada petak asal dan merubah nilai petak tujuan dengan nilai bidak yang digerakkan. Setelah proses ini selesai, dilakukan perubahan nilai pada variabel *turn* yang menyatakan giliran, sehingga 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 (contoh nilai nya adalah “2,7-2,6”). Dimana angka yang dimasukkan kedalam variabel tersebut merupakan koordinat pada papan.

Setelah seluruh gerakan yang mungkin selesai dicatat akan dilakukan proses evaluasi untuk setiap gerakan tersebut (sisa gerakan yang jelas tidak mungkin dilakukan tidak akan dievaluasi). Proses evaluasi inilah yang menggunakan algoritma Shannon Type-A yang akan dijelaskan lebih lanjut.

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.

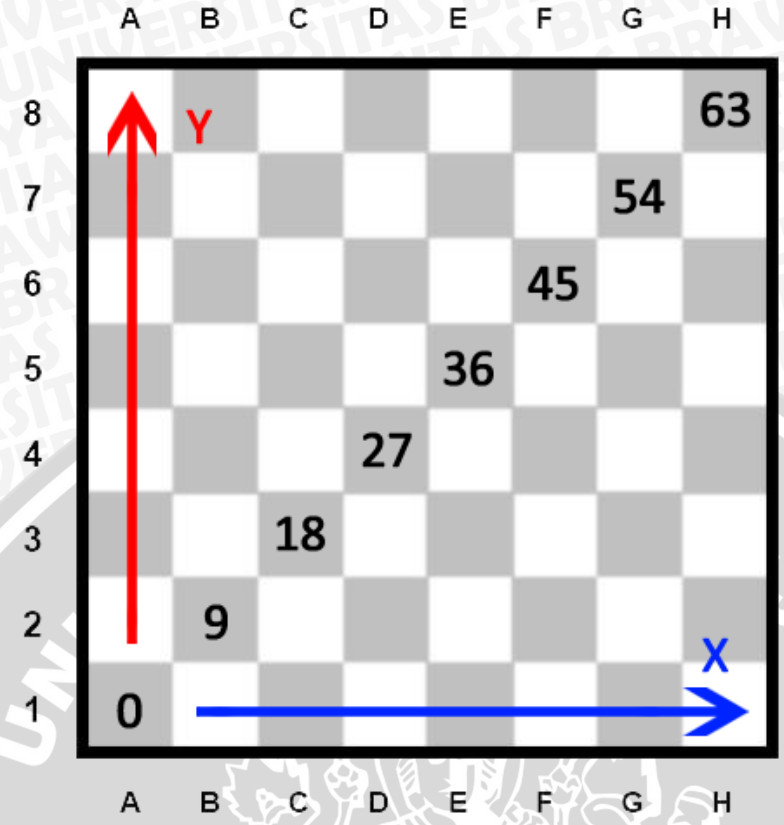
Setelah gerakan selesai dilakukan, variabel “turn” berubah lagi, menandakan saat ini adalah giliran Putih untuk bermain lagi. Hal ini akan terus diulangi hingga terjadi *checkmate* (skakmat) atau *draw* (seri/remis).

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 sebelumnnya, proses ini terdiri dari tiga bagian.

4.2.1 Board Representation

Board Representation yang digunakan pada program ini adalah Array 2-Dimensi berukruan 8x8. Dikarenakan jumlahnya yang sama dengan jumlah petak (squares) pada papan catur.



Gambar 4.2 *Board Representation*

Pada gambar diatas, Array dimulai dari pojok kiri bawah, seperti system koordinat Cartesian. Yaitu index pertama atau [0][0] berada pada koordinat A,1 dan index terakhir atau [7][7] berada pada koordinat H,8.

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

```
//Piece's Worth
const int pawn = 100;
const int bishop = 325;
const int knight = 300;
const int rook = 500;
const int queen = 900;
const int king = 30000;
```

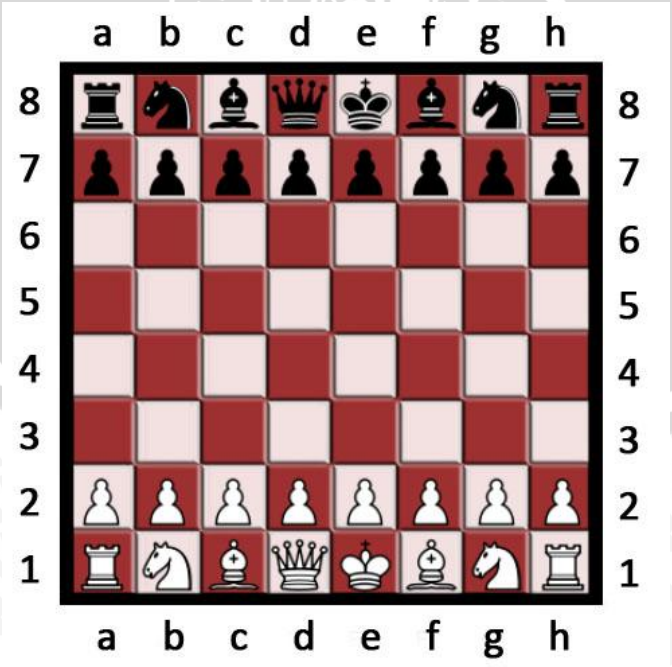
Pada awal permainan setiap elemen array akan diisi seluruhnya oleh fungsi inisialisasi dengan langkah sebagai berikut:

1. Isi seluruh elemen array, dari index pertama sampai index terakhir dengan nilai yang melambangkan petak kosong (bukan *null*). Dalam hal ini petak

kosong akan diisi dengan nilai "NU". "NU" disini merupakan sebuah string, bukan elemen kosong (*null*).

2. Rubah nilai dari index [0][0] sampai index [7][1] dengan nilai Bidak Putih. Dalam program ini nilai bidak putih akan selalu diawali dengan "a". Sehingga nilai untuk bidak putih yaitu: a1, a2, a3, a4, a5, a6 dengan urutan a1 merupakan White Pawn dan a6 merupakan White King. Nilai ini akan dimasukkan sesuai dengan posisi awal dari catur.
3. Rubah nilai dari index [0][6] sampai index [7][7] dengan nilai Bidak Hitam. Dalam program ini nilai bidak hitam akan selalu diawali dengan "b". Sehingga nilai untuk bidak hitam yaitu: b1, b2, b3, b4, b5, b6 dengan urutan b1 merupakan Black Pawn dan b6 merupakan Black King. Nilai ini akan dimasukkan sesuai dengan posisi awal dari catur.
4. Setelah semua nilai selesai dinisialisasi, maka akan ditampilkan gambar dasar papan. Lalu akan ditampilkan gambar yang telah dibuat sebelumnya untuk tiap-tiap bidak sesuai nilai yang dimasukkan pada langkah 2 dan 3.

Fungsi untuk melakukan representasi papan ini dinamakan `InitChessTable()` dan `DrawPieces()`.



Gambar 4.3 *Board Representation* Selesai

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 atau jalan menuju petak tujuan sudah ditempati oleh bidak dengan warna yang sama (*allied pieces*). Dengan pengecualian untuk Kuda yang bisa "melompat".
   b. Gerakan tidak akan bisa dilakukan jika gerakan tersebut akan mengakibatkan bidak "King" player tersebut berada dalam posisi skak (*check*).

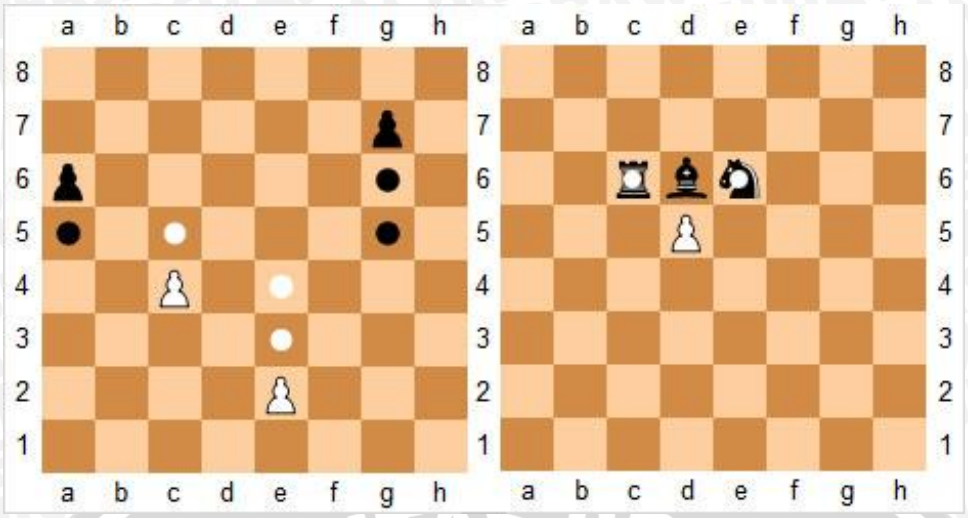
   Kedua hal tersebut berlaku untuk semua bidak tanpa kecuali walaupun sudah memenuhi peraturan individual untuk semua jenis bidak.

2. Pawn (Pion)

   Pawn hanya bisa bergerak maju satu langkah (x+0, y+1 untuk putih dan (x+0, y-1 untuk hitam), namun jika Pawn belum bergerak (masih berada di posisi awal) maka Pawn bisa bergerak maju sebanyak dua langkah (x+0, y+2 untuk putih dan (x+0, y-2 untuk hitam).

   Pawn bisa menangkap (*capturing*) bidak lawan dengan cara bergerak maju secara diagonal sebanyak satu petak ((x+1 || x-1) && y+1 untuk putih dan ((x+1 || x-1) && y11 untuk hitam). Pawn juga mempunyai gerakan spesial yang dinamakan "en passant".

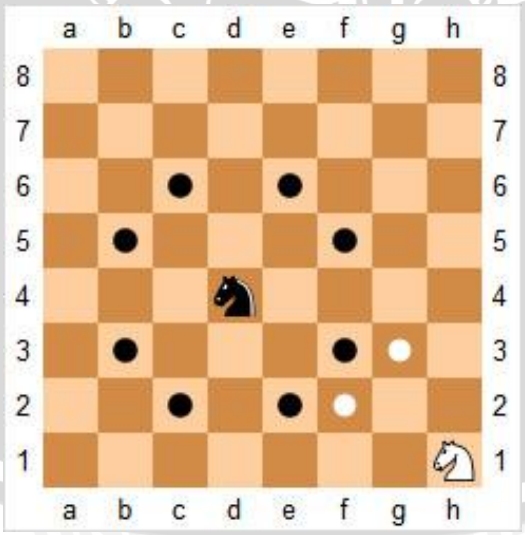
   Pawn merupakan satu-satunya bidak yang cara menangkap bidak lawan tidak sama dengan cara pergerakannya.



Gambar 4.4 *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 4.5 *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 4.6 *Bishop Movements and Captures*
Sumber: Wikipedia

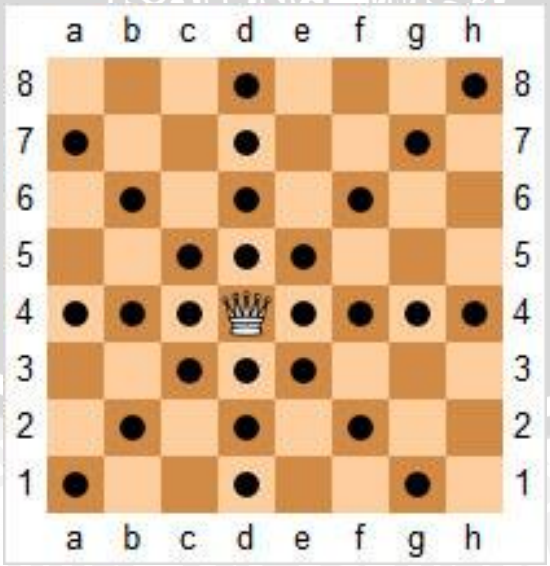
5. Rook (Benteng)

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 special dengan King, yang disebut *castling* (rokade).

Gambar 4.7 *Rook Movements and Captures*
Sumber: Wikipedia

6. Queen (Ratu/Menteri)

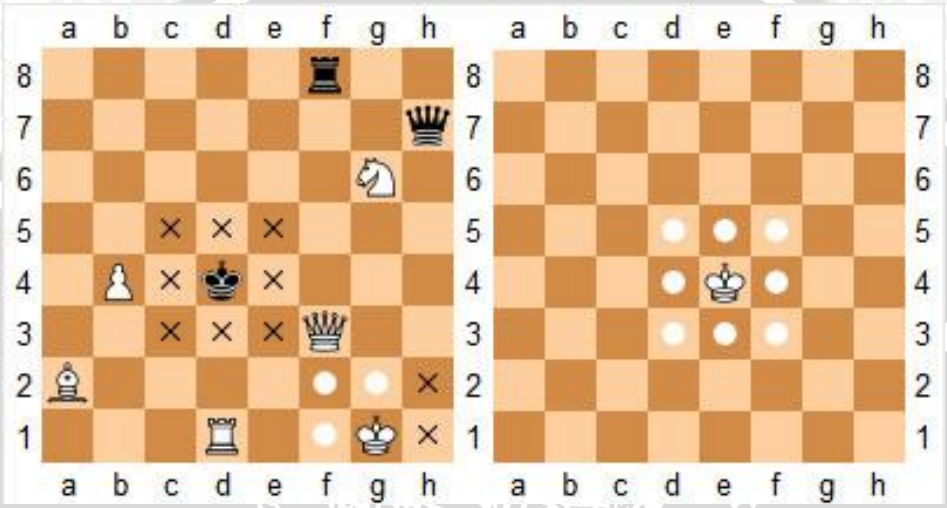
Ratu bisa bergerak secara horizontal, vertikal maupun diagonal melalui berapa saja petak yang kosong, seperti gabungan antara Rook dan Bishop. Queen menangkap pion lawan dengan menempati petak dimana lokasi pion lawan berada.



Gambar 4.8 *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 special dengan Rook, yang disebut *castling* (rokade).



Gambar 4.9 *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 computer. Komputer akan mengevaluasi papan berdasarkan kondisi saat ini (yaitu setelah pemain sebelumnya *selesai* melakukan gerakan). Potongan kode berikut akan dijalankan jika pemain sebelumnya sudah selesai melakukan gerakan:

```
turn = "b";
lblTurn.Text = "Turn " + (Phase + 1).ToString() + "\nWaiting for black's
movement";
Phase++;
```

Lalu dilakukan pengecekan apakah kondisi saat ini merupakan skakmat atau remis atau bukan. Karena jika terjadi skakmat atau remis permainan akan berakhir sampai disini, sehingga pemain selanjutnya (baik manusia ataupun computer) tidak akan mendapat giliran. Berikut potongan kode untuk mengecek apakah terjadi skakmat atau remis:

```
else if (!BlackToMove() && !WhiteFlash())
{
        …
        MessageBox.Show("DRAW!");
        …
}

else if (!BlackToMove() && WhiteFlash())
{
        …
        MessageBox.Show("CHECKMATE. You Win.");
        …
}
```

BlackToMove() merupakan fungsi untuk memeriksa apakah dalam kondisi papan saat ini pemain Black mempunyai paling tidak satu gerakan yang legal atau mungkin dilakukan, sedangkan WhiteFlash() merupakan fungsi untuk

memeriksa apakah dalam kondisi papan saat ini Black King sedang dalam posisi ter-skak oleh Putih.

Jika BlackToMove() bernilai salah dan WhiteFlash() bernilai salah permainan dinyatakan remii karena Black tidak dalam kondisi ter-skak, namun tidak bisa melakukan gerakan apapun.

Jika BlackToMove() bernilai salah dan WhiteFlash() bernilai benar permainan dinyatakan berakhir dengan White sebagai pemenang karena Black tidak bisa melakukan gerakan apapun karena dalam posisi ter-skak, sehingga merupakan *checkmate* (skakmat) untuk Black.

Jika BlackToMove() bernilai benar, maka permainan dilanjutkan dengan pergantian giliran ke Black. Dalam hal ini proses evaluasi akan dipanggil melalui:

```
backgroundWorker1.RunWorkerAsync();
```

Proses ini akan memanggil fungsi untuk melakukan evaluasi dengan memanggil fungsi:

```
Accelerate();
```

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). Prosesnya seperti pada potongan kode berikut:

```
for (xBlackDetect = 0; xBlackDetect <= 7; xBlackDetect++)
{
        for (yBlackDetect = 0; yBlackDetect <= 7; yBlackDetect++)
        {
                //Black Pawn Detections

                if  (Board[xBlackDetect,  yBlackDetect].IndexOfAny(new
                char[] { 'b' }) >= 0)
                {
                        Meltdown();
                        BlackDetected++;
                }

        }
}
```

Potongan kode diatas akan memeriksa seluruh isi array Board (array untuk representasi papan). Jika dalam suatu elemen array terdapat suatu huruf ‘b’ maka bisa dipastikan dalam index ini terdapat bidak hitam (dalam hal ini b adalah symbol untuk Black). Jika terdeteksi suatu bidak hitam dalam Loop tersebut, akan langsung dipanggil fungsi untuk melakukan proses selanjutnya.

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 ataupun jika pergerakan bidak tersebut akan mengakibatkan King pemain tersebut berada dalam kondisi skak.

```
private void Meltdown()
{
        //Rule Check Loop
        for (xBlackTrace = 0; xBlackTrace <= 7; xBlackTrace++)
        {
                for (yBlackTrace = 0; yBlackTrace <= 7;
                yBlackTrace++)
                {
                                if(RuleCheckBlack(Board[xBlackDetect,
                                yBlackDetect],          xBlackDetect,
                                yBlackDetect, xBlackTrace, yBlackTrace))
                                {
                                        BlackTraceDetected++;

                                        if
                                        (!CheckDetectBlack(xBlackDetect,
                                        yBlackDetect,       xBlackTrace,
                                        yBlackTrace))
                                        {

                                                AllowedMovesBlack.Add(xBla
                                                ckDetect    +    ","    +
                                                yBlackDetect   +   "-"   +
                                                xBlackTrace   +   ","   +
                                                yBlackTrace);

                                        }

                                }
                }
        }
        //End Rule Check Loop
}
```

Proses diatas dilakukan didalam kondisi Looping dari proses pertama, proses pertama akan mengirimkan data dari bidak yang akan diperiksa pergerakannya, data tersebut berupa lokasi bidak (xBlackDetect dan yBlackDetect).

Setelah itu akan dilakukan proses pengecekan melalui fungsi diatas. Fungsi ini akan melakukan satu kali Loop lagi, yang berfungsi untuk menentukan petak tujuan bidak tersebut akan bergerak, dalam fungsi ini petak tujuan dilambangkan sebagai xBlackTrace dan yBlackTrace.

Proses pengecekan dibagi menjadi dua bagian yaitu:

a. Memeriksa apakah bidak tersebut bisa bergerak ke petak tujuan berdasarkan peraturan yang telah dibuat sebelumnya (Move Generation) melalui fungsi RuleCheckBlack().
b. Memeriksa apakah jika gerakan tersebut dilakukan akan mengakibatkan posisi King pemain tersebut berada dalam kondisi skak melalui fungsi CheckDetectBlack().

Jika sudah diperiksa gerakan tersebut boleh dilakukan menurut peraturan dan gerakan tersebut tidak membahayakan keselamatan King pemain tersebut maka akan dilakukan eksekusi kode berikut:

```
AllowedMovesBlack.Add(xBlackDetect + "," + yBlackDetect + "-" +
xBlackTrace + "," + yBlackTrace);
```

Proses tersebut akan melakukan pengisian data kedalam Array yang bernama `AllowedMovesBlack`.

Array tersebut berfungsi untuk mencatat data dari suatu gerakan yang boleh untuk dilakukan, dalam hal ini data yang akan dimasukkan kedalam Array tersebut adalah koordinat dari suatu bidak dan koordinat tujuan dimana bidak tersebut boleh bergerak (yang telah divalidasi melalui proses-proses sebelumnya)

Proses ini akan terus berulang hingga seluruh bidak yang berwarna sama telah diperiksa kemungkinan bergeraknya.

3. Pemberian nilai untuk gerakan yang sudah dicatat.

Setelah seluruh gerakan yang legal untuk dilakukan selesai dicatat kedalam Array dalam proses selanjutnya, akan dilakukan proses pemberian nilai untuk mencari gerakan mana yang akan paling menguntungkan, dalam hal ini akan menghasilkan nilai tertinggi dengan eksekusi kode dibawah ini:

```
for (int EvaluateAllowedMoves = 0; EvaluateAllowedMoves < of;
EvaluateAllowedMoves++)
{
        …
        //Do THE Move
        Board[Convert.ToInt32(preDrei[0]),
        Convert.ToInt32(preDrei[1])]                                    =
        Board[Convert.ToInt32(preZwei[0]),
        Convert.ToInt32(preZwei[1])];
        Board[Convert.ToInt32(preZwei[0]),
        Convert.ToInt32(preZwei[1])] = "NU";

        //Board Position is now After (Projected) Black Moves
        MentalOut();

        …
        //UnDO THE Move
        Board[Convert.ToInt32(preZwei[0]),
        Convert.ToInt32(preZwei[1])] = originalZwei;
        Board[Convert.ToInt32(preDrei[0]),
        Convert.ToInt32(preDrei[1])] = originalDrei;

        …
}
```

Looping utama dilakukan berdasarkan jumlah gerakan legal yang telah ditemukan pada proses sebelumnya. Setelah itu akan dilakukan perubahan sementara pada papan sesuai data yang diberikan oleh gerakan dalam Array sebelumnya. Eksekusi sementara dari gerakan ini bertujuan untuk menerapkan algoritma Shannon Type-A atau disebut juga metode minimax. Proses ini dilakukan dengan cara memanggil fungsi `MentalOut()`.

4. Penerapan Algoritma Shannon Type-A.

Seperti telah dijelaskan dalam langkah sebelumnya, proses penerapan ini semua berlangsung dalam fungsi `MentalOut()`.

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.

Fungsi ini akan terus diulang berdasarkan jumlah kedalaman pencarian (ply) yang telah ditentukan sebelumnya.

```
for (CurrentLevel = 1; CurrentLevel < MaxLevel; CurrentLevel++)
```

CurrentLevel merupakan iterasi saat ini, sedangkan MaxLevel merupakan batasan kedalaman pencarian. Untuk tiap iterasi akan dilakukan pergantian pemain. Dalam iterasi pertama ini, karena saat ini giliran hitam, maka Loop ini akan melakukan prediksi dari lawan, yaitu White.

```
if   (!CheckDetectWhite(xWhiteDetect,   yWhiteDetect,   xWhiteTrace,
yWhiteTrace))
{

        TempWhiteDetect = Board[xWhiteDetect, yWhiteDetect];
        TempWhiteTrace = Board[xWhiteTrace, yWhiteTrace];

        //Do THE Move
        Board[xWhiteTrace, yWhiteTrace] = TempCheckWhiteDetect;
        Board[xWhiteDetect, yWhiteDetect] = "NU";

        //Recount
        BurstLink("Sentinel");

        TraceAllowedMovesWhite.Add(xWhiteDetect + "," + yWhiteDetect
        + "-" + xWhiteTrace + "," + yWhiteTrace);
        TraceEvaluatedMovesWhite.Add(TotalBurst);

        //UnDO THE Move
        Board[xWhiteDetect, yWhiteDetect] = TempCheckWhiteDetect;
        Board[xWhiteTrace, yWhiteTrace] = TempCheckWhiteTrace;


}
```

Proses diatas mirip dengan proses pengecekan gerakan yang legal pada langkah sebelumnya, dalam hal ini karena kita mencoba mencari langkah terbaik untuk White, maka kali ini akan dicari seluruh langkah yang legal untuk White.
Jika suatu gerakan yang legal ditemukan maka gerakan tersebut akan dilakukan sementara untuk dihitung nilainya melalui fungsi BurstLink().

Sistem untuk perhitungan nilai adalah sebagai berikut:

Nilai Evaluasi Putih = Jumlah Bidak Putih – Jumlah Bidak Hitam

Nilai yang dimaksud disini adalah nilai untuk setiap bidak yang telah ditentukan pada saat proses Board Representation.

Setelah seluruh proses penilaian selesai maka kita akan mendapatkan dua Array baru yaitu `TraceAllowedMovesWhite` yang berisi koordinat gerakan legal tersebut dan `TraceEvaluatedMovesWhite` yang berisi nilai evaluasi papan jika gerakan tersebut dilakukan.

Kedua Array tersebut mempunyai jumlah isi yang sama, sehingga secara tidak langsung sudah saling terhubung. Contohnya gerakan yang tercatat dalam index ke-3 `TraceAllowedMovesWhite` nilainya akan berada pada index ke-3 juga dari array `TraceEvaluatedMovesWhite.`

Langkah selanjutnya yaitu mencari gerakan mana yang akan menghasilkan evaluasi paling besar dengan menggunakan fungsi:

```
int WhiteIndex =
System.Convert.ToInt32(TraceEvaluatedMovesWhite.ToList().IndexOf(Tr
aceEvaluatedMovesWhite.Max()));
```

dan

```
int BlackIndex =
System.Convert.ToInt32(TraceEvaluatedMovesBlack.ToList().IndexOf(Tr
aceEvaluatedMovesBlack.Max()));
```

Setelah itu gerakan yang akan menghasilkan nilai evaluasi terbesar akan dilakukan, dan proses terus berlanjut hinga mencapai tingkat kedalaman yang sudah ditentukan. Setelah itu fungsi akan mengirim hasil akhir penilaian ini ke fungsi utama.

5. **Perhitungan dan pemilihan nilai akhir.**

   Sebelum nilai terakhir dikembalikan, akan dilakukan dulu pengecekan apakah gerakan ini akan mengakibatkan terjadinya remis atau tidak.

```
//Anti-Draw Mechanism
if (!WhiteToMove() && !BlackLotus())
{
   Lancelot = 99999;
}
```

Kode diatas bertujuan untuk melakukan pengecekan, jika gerakan ini akan mengakibatkan lawan tidak mempunyai gerakan legal sama sekali (`WhiteToMove()` Bernilai false) tetapi kita tidak melakukan skak terhadap King lawan (`BlackLotus()` Bernilai false) maka nilai akhir akan dirubah

menjadi sangat besar, sehingga dalam proses selanjutnya gerakan ini tidak akan pernah dipilih, kecuali jika ini satu-satunya gerakan yang tersedia.

Setelah semua selesai divalidasi, maka akan didapatkan sebuah Array terakhir yaitu:

```
EvaluatedMovesBlack.Add(Lancelot);
```

Untuk setiap isi dari array tersebut bisa dianalogikan sebagai berikut:

"*Jika saya melakukan gerakan ini, maka setelah kedalaman pencarian sekian akan saya dapatkan nilai dari kemungkinan terburuk yang bisa terjadi*".

Sehingga sesuai dengan prinsip Minimax, untuk mendapatkan nilai terbesar maka harus dipilih gerakan yang akan menghasilkan nilai terkecil untuk lawan, dengan eksekusi kode berikut:

```
var Zero =
EvaluatedMovesBlack.ToList().IndexOf(EvaluatedMovesBlack.Min());
```

Sehingga didapatkan index dari Array yang akan menghasilkan gerakan yang paling menguntungkan.

6. **Eksekusi gerakan.**

Setelah didapatkan index dari array dari proses sebelumnya maka akan dilakukan proses terkahir, yaitu perpindahan sesungguhnya dari bidak dengan eksekusi kode berikut:

```
…
Board[Convert.ToInt32(Drei[0]), Convert.ToInt32(Drei[1])] =
Board[Convert.ToInt32(Zwei[0]), Convert.ToInt32(Zwei[1])];
Board[Convert.ToInt32(Zwei[0]), Convert.ToInt32(Zwei[1])] = "NU";
…
```

Setelah bidak benar-benar berpindah, maka giliran pemain tersebut sudah selesai dan akan dieksekusi kode terakhir sebagai berikut:

```
turn = "a";
lblTurn.Text = "Turn " + (Phase + 1).ToString() + "\nWaiting for
white's movement";
DrawPieces();
panelBase.Invalidate();
```

Setelah kode diatas dijalankan maka giliran akan berpindah ke pemain selanjutnya. Perulangan ini akan terus berlanjut hingga tiba saatnya permainan berakhir (Skakmat atau Remis).