Hak cipta dan penggunaan kembali:

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

Copyright and reuse:

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

BAB III

ANALISIS DAN PERANCANGAN APLIKASI

3.1 Metode Penelitian

Metode penelitian yang akan digunakan dalam penelitian ini antara lain adalah sebagai berikut.

1. Studi Literatur

   Melakukan studi mengenai teori – teori dan konsep yang berkaitan dengan pokok bahasan penelitian, seperti teori mengenai algoritma Minimax, konsep Alpha-Beta Pruning, konsep AI, peraturan permainan catur dan berbagai konsep pendukung lainnya. Referensi – referensi yang digunakan dapat berupa buku, jurnal ilmiah, artikel, dan lain – lain.

2. Perancangan Aplikasi

   Melakukan perancangan awal terhadap aplikasi yang akan dibangun, meliputi perancangan alur aplikasi dalam bentuk *flowchart* dan rancangan *interface* agar mudah dimengerti oleh user/pemain.

3. Pembangunan Aplikasi

   Melakukan pembangunan aplikasi permainan catur dengan mengimplementasikan rancangan dan metode yang telah didefinisikan sebelumnya dengan menggunakan bahasa pemrograman yang telah ditentukan pada awal perancangan.

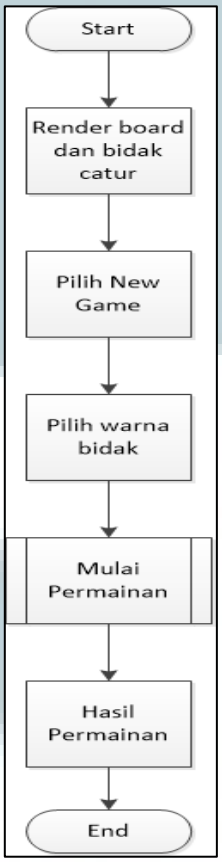
4. Uji Coba dan Evaluasi

Implementasi Algoritma ..., Ronal Gorba Timothy, FTI UMN, 2014

Melakukan uji coba terhadap aplikasi disertai dengan evaluasi hasil yang didapatkan.

3.2 Analisis Aplikasi

Pada penelitian ini, aplikasi kecerdasan buatan akan dibangun berdasarkan pemodelan yang merupakan peraturan dasar permainan catur. Kecerdasan buatan akan diterapkan setelah pemain membuat sebuah langkah untuk kemudian dianalisa sebagai suatu kondisi. Berikut adalah *system flow* yang menggambarkan cara kerja aplikasi kecerdasan buatan yang akan dibangun.

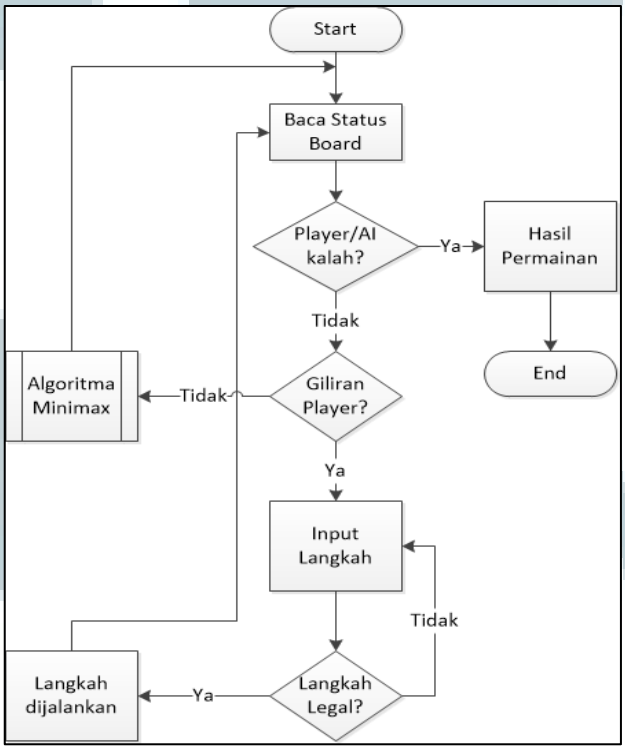


Gambar 3.1 *Flowchart* Gambaran Umum Program

Berdasarkan *flowchart* pada gambar 3.1, dapat dilihat bahwa proses kecerdasan buatan diawali dengan menerapkan terlebih dahulu aturan-aturan dari setiap bidak yang boleh dilakukan dalam area *board* yang telah dibuat sebelumnya. Hasil yang berupa langkah dari kecerdasan buatan kemudian dieksekusi sebagai keputusan terbaik dari analisa yang dilakukan dalam proses "Mulai Permainan". Algoritma *Minimax* akan digunakan dalam menganalisa kemungkinan seluruh langkah berdasarkan kondisi permainan dengan penjelasan sebagai berikut.

3.2.1 *Flowchart* Permainan Player vs AI

Pada proses permainan, alur proses mulai dari penentuan giliran sampai penentuan langkah komputer akan dijelaskan dalam *flowchart* berikut.



Gambar 3.2 *Flowchart* Player vs AI

3.2.2 Algoritma *Minimax* Pada Analisis Permainan

Kecerdasan buatan yang akan dibangun pada aplikasi akan dibagi menjadi tiga bagian utama sebagai berikut.

1. *Move Generator*

   Fungsinya adalah untuk membuat semua daftar langkah yang bisa dijalankan suatu pemain.

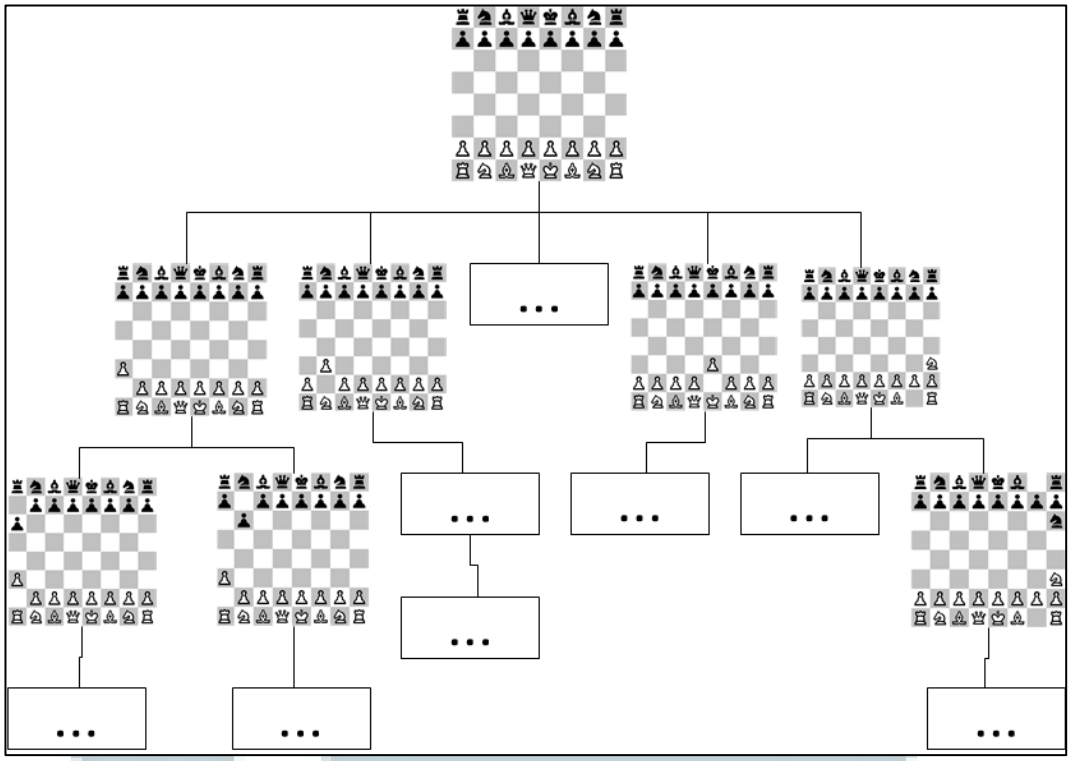
2. Fungsi Evaluasi

   Digunakan untuk mengevaluasi seberapa baik atau buruk sebuah posisi dari sudut pandang pemain tertentu (mengevaluasi semua daftar langkah yang telah dibuat sebelumnya).

3. Algoritma Pencarian (*Minimax*)

   Cara kerja dalam menelusuri semua kemungkinan langkah sampai kedalaman tertentu.

Algoritma *Minimax* sendiri memiliki konsep pencarian dengan mengacu pada teknik pencarian *Depth First Search* (DFS) sebagai dasarnya. Dengan DFS, maka seluruh kemungkinan langkah akan dirunut dalam bentuk *tree* yang memiliki relasi sebab akibat terhadap setiap langkah itu sendiri sampai kondisi di mana salah satu pemain memenangkan permainan. Berikut adalah gambaran analisis pertama yang dihasilkan dari pembangunan *tree* dengan menggunakan teknik pencarian DFS.



Gambar 3.3 Tree Permainan Catur Menggunakan Teknik Pencarian DFS

Pada gambar di atas dapat dilihat bahwa teknik DFS yang menjadi langkah analisis pertama dalam aplikasi akan merunut setiap kemungkinan yang ada. Kemudian setelah semua kemungkinan tersebut diperoleh, maka selanjutnya dilakukan fungsi evaluasi untuk menilai "seberapa efektif" langkah yang akan dipilih.

Pada tahap analisis ini, fungsi evaluasi bertujuan untuk memberikan estimasi mengenai kualitas kondisi *board* permainan dalam mengarahkan seorang pemain untuk memenangkan permainan, biasa disebut dengan *static board evaluator* dengan fungsi *f*(n) sebagai representasinya.

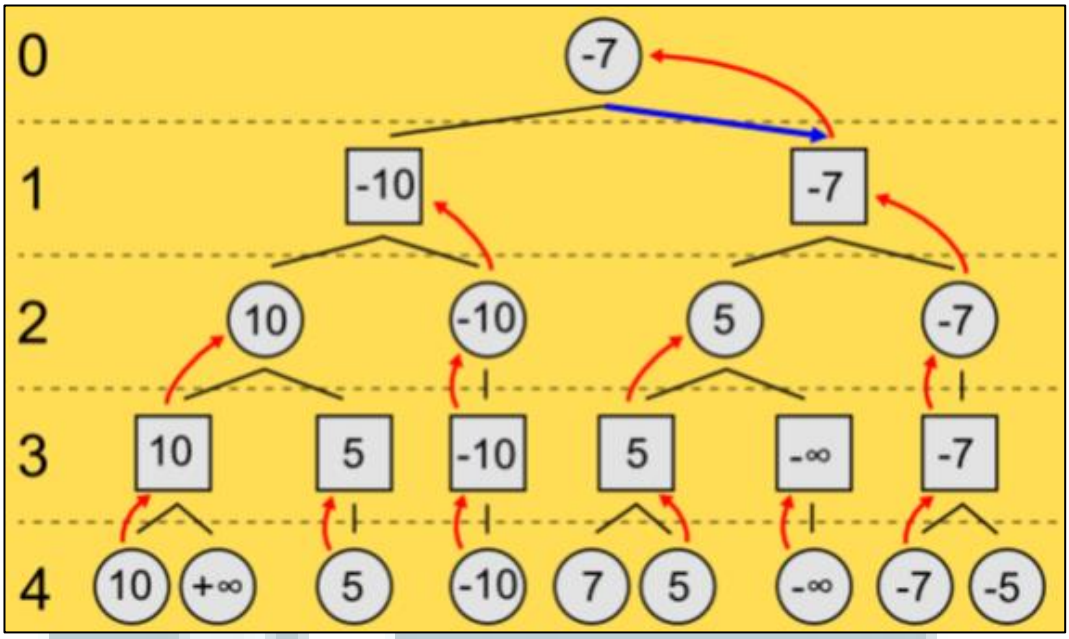
Berikut adalah evaluasi yang diberikan pada setiap simpul akar pada *tree* yang telah dibangun.

1. Jika *f*(n) berupa bilangan positif besar, artinya kondisi permainan dengan pemilihan simpul n bersifat “baik untuk saya dan buruk untuk lawan”.
2. Jika *f*(n) berupa bilangan negatif besar, artinya kondisi permainan dengan pemilihan simpul n bersifat “buruk untuk saya dan baik untuk lawan”.
3. Jika *f*(n) berupa nilai yang mendekati 0, artinya permainan dalam keadaan netral atau imbang.

Kemudian pada simpul daun pada *tree* yang telah dibangun estimasi evaluasinya adalah sebagai berikut.

1. Jika *f*(n) berupa bilangan positif “tak terhingga”, artinya kondisi “saya memenangkan permainan”.
2. Jika *f*(n) berupa bilangan negatif “tak terhingga”, artinya kondisi “lawan memenangkan pertandingan”.

Berdasarkan konsep dari analisis yang telah dibuat tersebut, kemudian dibuatlah sebuah nilai pada setiap simpul yang sudah dibangun. Nilai-nilai ini selanjutnya akan ditelusuri dan dihitung satu-persatu untuk menentukan sebuah langkah yang akan menghasilkan suatu nilai maksimum yang merepresentasikan keuntungan untuk pemain dan secara bersamaan juga menghasilkan suatu nilai minimum yang merepresentasikan keuntungan untuk lawan. Pada algoritma *Minimax*, nilai maksimum disebut dengan “*max*” dan nilai minimum disebut dengan “*min*”. Penjelasan sistem kerja analisis nilai *tree* pada algoritma *Minimax* digambarkan sebagai berikut.



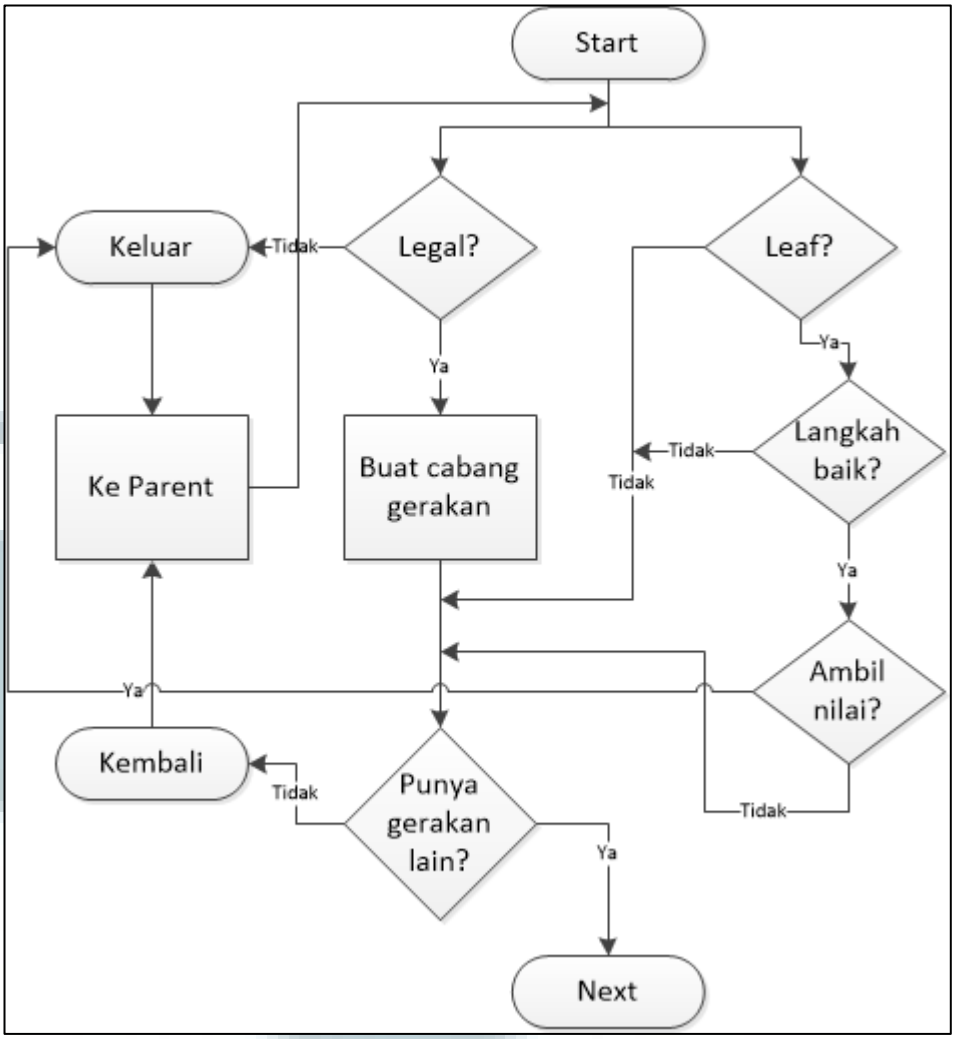
Gambar 3.4 Tree yang Dibangun dengan Algoritma Minimax (sumber : http://upload.wikimedia.org/wikipedia/commons/thumb/6/6f/Minimax.svg/701px-Minimax.svg.png)

Algoritma *Minimax* yang akan diterapkan dalam aplikasi akan membuat penilaian kepada setiap kemungkinan langkah yang dibangun pada *tree*, kemudian *tree* dibagi menjadi tingkatan-tingkatan level yang berjajar menurut barisnya. Baris pertama dimulai dengan level genap yaitu dimulai dari level 0, 2 dan seterusnya yang berisi tentang kondisi permainan terbaru yang kemudian akan dijabarkan lagi menjadi simpul-simpul langkah yang dapat dipilih sebagai langkah berikutnya dari pemain. Level genap pada *tree* ini disebut juga level "*max*" pada algoritma *Minimax*. Selanjutnya pada level ganji, mulai dari level 1, 3, dan seterusnya merupakan langkah-langkah yang mungkin terjadi pada saat giliran lawan. Level ganjil pada *tree* ini disebut juga level "*min*" pada algoritma

Minimax. Berikut adalah tahapan-tahapan analisis dalam menentukan pilihan langkah dengan algoritma *Minimax* yang ada pada gambar 3.3.

1. Penelusuran dilakukan satu persatu mengikuti cabang menurut aturan DFS.
2. Ketika telah sampai pada bagian *leaf*, dilakukan pengecekan apakah bagian tersebut berada di level genap (maksimum) atau ganjil (minimum), kemudian dilihat nilainya. Pada gambar 3.3, *leaf* pertama bernilai 10 dan terletak di level genap (maksimum).
3. Apabila berada di level genap, maka *parent* dari *child* (yang berada di level ganjil) mengambil nilai minimum dari *child* di bawahnya. Sebaliknya, apabila berada di level ganjil, maka *parent* dari *child* (yang berada di level genap) mengambil nilai maksimum dari *child* di bawahnya. Pada gambar 3.3, *parent* dari *leaf* pertama yang ada di level 3 (ganjil) memiliki 2 *child* yang bernilai 10 dan bilangan positif tak terhingga. Dengan demikian maka *parent* tersebut mengambil nilai yang terkecil, yaitu 10.
4. Proses diulang seterusnya sampai nilai dari seluruh *leaf* diseleksi dan diambil menjadi nilai pada puncak *tree*.

Gambaran proses keseluruhan algoritma *Minimax* dalam mengubah suatu kondisi permainan sehingga menghasilkan sebuah langkah diperlihatkan dalam *flowchart* sebagai berikut.

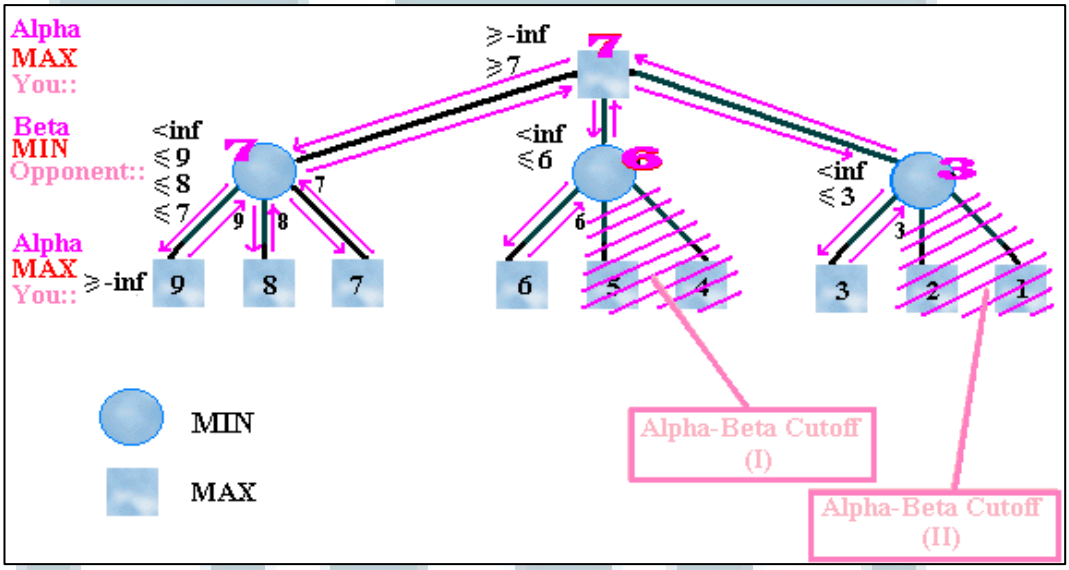


Gambar 3.5 *Flowchart* Algoritma Minimax

Pada *flowchart* di atas, algoritma *Minimax* dijalankan dengan membagi 2 kerja utama yang telah di bahas sebelumnya, yaitu fungsi *Move Generator* dan fungsi evaluasi. Karena penggunaan algoritma *Minimax* cukup memakan waktu dan berbanding lurus dengan banyaknya kemungkinan yang ada pada *tree*, maka aplikasi kecerdasan buatan dioptimasi dengan *Alpha-Beta Pruning*.

3.2.3 *Alpha-Beta Pruning* pada Optimasi Algoritma *Minimax*

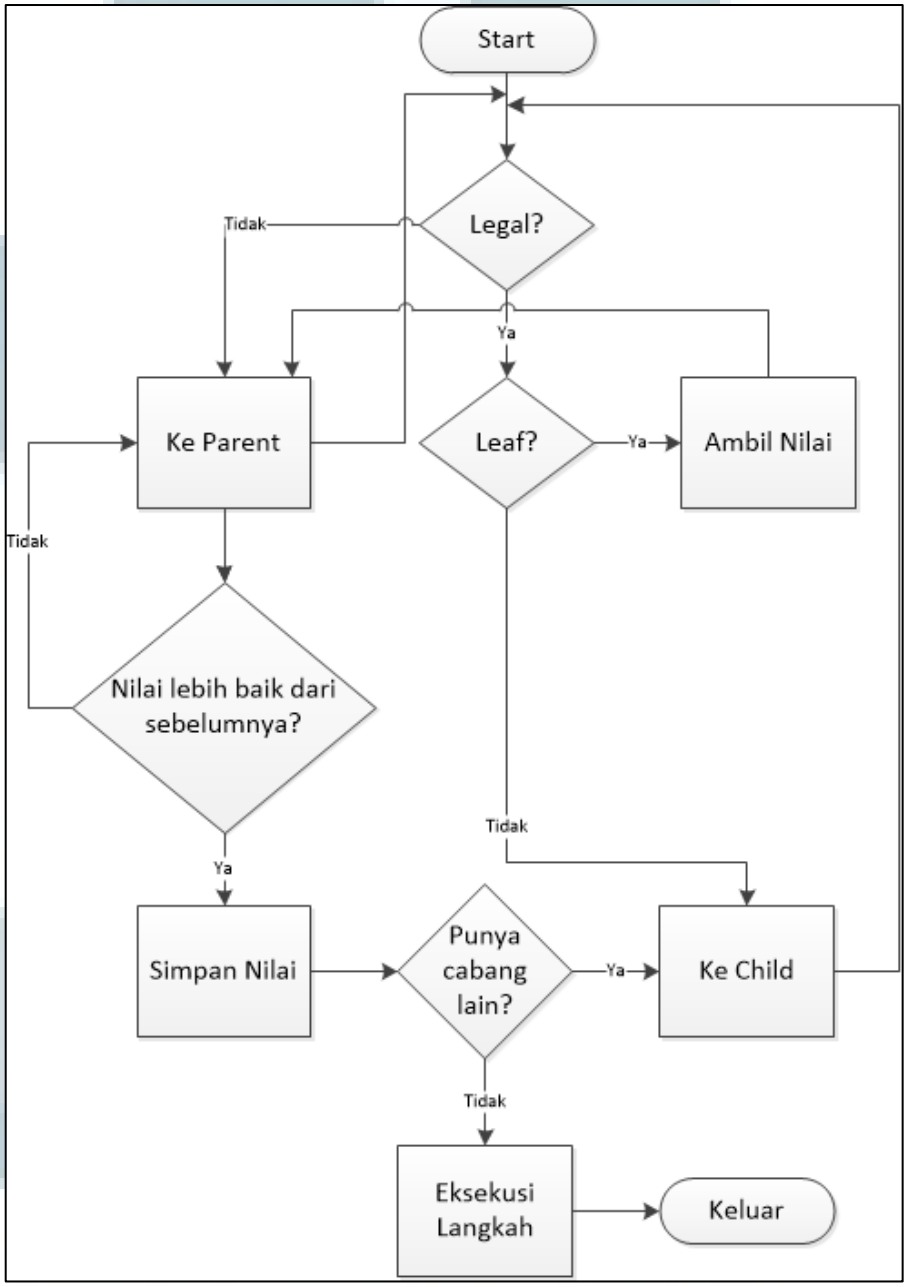
Model optimasi *Alpha-Beta Pruning* diterapkan dengan mengeliminasi cabang pada *tree* yang sudah dibuat sebelumnya dengan tujuan mengurangi waktu pencarian. Cabang akan dieliminasi dan tidak dilakukan pencarian lanjutan apabila setidaknya terdapat paling tidak satu kemungkinan yang ditemukan dan membuktikan bahwa langkah tersebut lebih buruk jika dibandingkan dengan langkah yang telah didapatkan sebelumnya. Berikut adalah hasil analisis penulis dalam menerapkan algoritma *Alpha-Beta Pruning* untuk perancangan kecerdasan buatan.



Gambar 3.6 Pencarian pada Tree dengan Alpha-Beta Pruning (sumber : https://chessprogramming.wikispaces.com/file/view/alphabeta.gif/310259838/alphabeta.gif)

Alpha-Beta Pruning memiliki konsep di mana setiap simpul/*node* memiliki suatu nilai pembatas/penjaga yang disebut sebagai suatu variabel bebas

yang disebut *Alpha* dan *Beta*. Didefinisikan suatu aturan bahwa *Alpha* adalah suatu batas bawah dan *Beta* adalah batas atas. Kemudian inisiasikan nilai *Alpha* dengan nilai negatif tak terhingga dan nilai *Beta* dengan nilai positif tak terhingga. Berikut adalah *flowchart* pada proses algoritma Alpha-Beta.



Gambar 3.7 *Flowchart* Algoritma Minimax dengan Optimasi Alpha-Beta

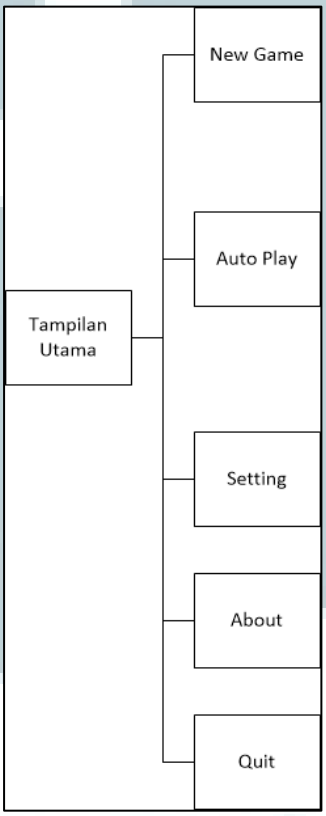
```
private int AlphaBeta(ChessBoard chessBoard, ChessBoard.PlayerColorE ePlayerColor, int iDepth, int iAlpha, int iBeta, int iWhiteMoveCount, int iBlackMoveCount, AlphaBetaInfo abInfo) {
    int                         iRetVal;
    List<ChessBoard.MovePosS>   moveList;
    int                         iPts;
    int                         iMoveCount;
    ChessBoard.PosInfoS         posInfo;
    TransEntryTypeE             eType = TransEntryTypeE.Alpha;
    ChessBoard.BoardStateMaskE  eBoardExtraInfo;
    ChessBoard.RepeatResultE    eResult;

    if (abInfo.m_dtTimeOut != DateTime.MaxValue && DateTime.Now >= abInfo.m_dtTimeOut) {
        iRetVal = Int32.MinValue;   // Time out!
    } else if (chessBoard.IsEnoughPieceForCheckMate()) {
        eBoardExtraInfo = chessBoard.ComputeBoardExtraInfo(ePlayerColor, true);
        iRetVal         = (abInfo.m_transTable != null) ? abInfo.m_transTable.ProbeEntry(chessBoard.CurrentZobristKey, eBoardExtraInfo, iDepth, iAlpha, iBeta) : Int32.MaxValue;
        if (iRetVal == Int32.MaxValue) {
            if (iDepth == 0 || m_bCancelSearch) {
                iRetVal = chessBoard.Points(abInfo.m_searchMode, ePlayerColor, abInfo.m_iMaxDepth - iDepth, iWhiteMoveCount - iBlackMoveCount, abInfo.m_posInfoWhite, abInfo.m_posInfoBlack);
                if (ePlayerColor == ChessBoard.PlayerColorE.Black) {
                    iRetVal = -iRetVal;
                }
                abInfo.m_iPermCount++;
                if (abInfo.m_transTable != null) {
                    abInfo.m_transTable.RecordEntry(chessBoard.CurrentZobristKey, eBoardExtraInfo, iDepth, iRetVal, TransEntryTypeE.Exact);
                }
            } else {
                moveList    = chessBoard.EnumMoveList(ePlayerColor, true, out posInfo);
                iMoveCount  = moveList.Count;
                if (ePlayerColor == ChessBoard.PlayerColorE.White) {
                    iWhiteMoveCount         = iMoveCount;
                    abInfo.m_posInfoWhite   = posInfo;
                } else {
                    iBlackMoveCount         = iMoveCount;
                    abInfo.m_posInfoBlack   = posInfo;
                }
                if (iMoveCount == 0) {
                    if (chessBoard.IsCheck(ePlayerColor)) {
                        iRetVal = -1000000 - iDepth;
                    } else {
                        iRetVal = 0;    // Draw
                    }
                    if (abInfo.m_transTable != null) {
                        abInfo.m_transTable.RecordEntry(chessBoard.CurrentZobristKey, eBoardExtraInfo, iDepth, iRetVal, TransEntryTypeE.Exact);
                    }
                } else {
                iRetVal = iAlpha;
                foreach (ChessBoard.MovePosS move in moveList) {
                    eResult = chessBoard.DoMoveNoLog(move);
                    abInfo.m_arrMovePos[iDepth - 1] = move;
                    if (eResult == ChessBoard.RepeatResultE.NoRepeat) {
                        iPts = -AlphaBeta(chessBoard,
                                          (ePlayerColor == ChessBoard.PlayerColorE.Black) ? ChessBoard.PlayerColorE.White : ChessBoard.PlayerColorE.Black,
                                          iDepth - 1,
                                          -iBeta,
                                          -iRetVal,
                                          iWhiteMoveCount,
                                          iBlackMoveCount,
                                          abInfo);
                    } else {
                        iPts = 0;
                    }
                    chessBoard.UndoMoveNoLog(move);
                    if (iPts == Int32.MinValue) {
                        iRetVal = iPts;
                        break;
                    } else {
                        if (iPts > iRetVal) {
                            iRetVal = iPts;
                            eType   = TransEntryTypeE.Exact;
                        }
                        if (iRetVal >= iBeta) {
                            iRetVal = iBeta;
                            eType   = TransEntryTypeE.Beta;
                            break;
                        }
                    }
                }
                if (abInfo.m_transTable != null && iRetVal != Int32.MinValue) {
                    abInfo.m_transTable.RecordEntry(chessBoard.CurrentZobristKey, eBoardExtraInfo, iDepth, iRetVal, eType);
                }
            }
        }
    }
} else {
    iRetVal = 0;
}
return(iRetVal);
```

Gambar 3.8 Potongan Kode Program Algoritma Minimax dengan Optimasi Alpha-Beta

3.3 Perancangan Aplikasi

3.3.1 Struktur Navigasi Menu

Berikut ini adalah struktur navigasi menu dan fitur-fitur yang dapat digunakan saat menjalankan aplikasi.



Gambar 3.9 Struktur Navigasi Menu Aplikasi

Pada tampilan utama aplikasi terdapat 5 menu utama dengan fungsi masing-masing menu tersebut adalah sebagai berikut.

1. *New Game*

   Pada menu ini, pemain atau *user* dapat memilih jenis permainan berdasarkan lawan main yang diinginkan untuk memulai langsung permainan. Menu ini memiliki tiga submenu sebagai berikut.

a. Submenu *Human VS AI*, yang berfungsi untuk memainkan satu kali permainan catur dengan kecerdasan buatan sebagai lawan main dari pemain atau *user*.

b. Submenu *Human VS Human*, yang berfungsi untuk memainkan satu kali permainan catur dengan pemain/*user* lain sebagai lawan mainnya.

c. Submenu AI VS AI, yang berfungsi untuk melakukan permainan satu lawan satu antara kecerdasan buatan dengan kecerdasan buatan.

Khusus pada pilihan submenu "*Human VS AI*", pemain diperbolehkan memilih untuk memainkan bidak putih (*White*) atau bidak hitam (*Black*).

2. *Auto Play*

Pada pilihan menu ini pemain dapat secara langsung mengubah mode permainan menjadi mode otomatis atau "AI VS AI". Menu ini dapat dipilih dan dijalankan kapan pun baik saat permainan belum dimulai maupun saat permainan sudah berjalan sebelumnya dalam mode apa pun.

3. *Setting*

Pada menu ini, pemain dapat melakukan konfigurasi terhadap proses pencarian keputusan pada kecerdasan buatan yang telah dibangun. Berikut adalah dua submenu yang dapat dipilih di dalamnya.

a. *Search Depth*, yaitu berfungsi untuk membatasi pencarian langkah pada kecerdasan buatan. Pada aplikasi permainan catur pada umumnya, fungsi ini digunakan sebagai fitur untuk mempermudah tingkat kesulitan permainan dalam melawan kecerdasan buatan. Pada menu ini

terdapat dua pilihan, yaitu berdasarkan kedalaman lapisan (*ply*) *tree* dan berdasarkan waktu pencarian (dalam satuan detik).

b. *Algorithm,* yaitu opsi yang berfungsi untuk memilih algoritma yang akan digunakan oleh kecerdasan buatan. Pada menu ini terdapat dua pilihan, yaitu "*Minimax*" dan "*Minimax + Alpha Beta*".

4. *About*

Menu ini akan menampilkan halaman yang berisi informasi mengenai pembuat aplikasi.

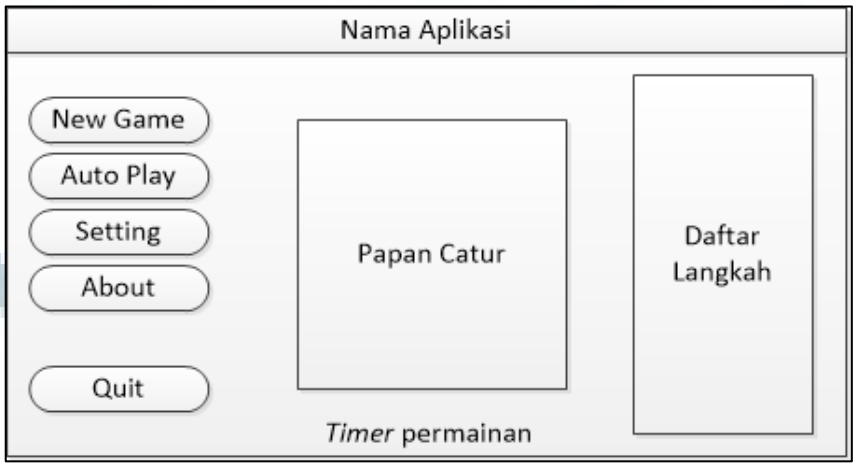
5. *Quit*

Menu ini berfungsi untuk keluar dari aplikasi permainan catur.

3.3.2 Desain Antarmuka Aplikasi

Desain antarmuka atau *user interface* pada aplikasi akan digunakan sebagai acuan dalam membangun tampilan aplikasi yang akan digunakan oleh pemain atau *user*.

1. Tampilan Utama



Gambar 3.10 Desain Antarmuka Tampilan Utama

Tampilan utama adalah tampilan halaman yang akan pertama kali dilihat oleh pengguna saat memulai aplikasi permainan catur ini. Pada tampilan utama terdapat lima menu yang dapat langsung dipilih oleh pengguna, yaitu menu *New Game*, *Auto Play*, *Option*, *About*, dan *Quit*. Pertama kali aplikasi dijalankan, pengguna dapat langsung bermain sebagai bidak putih dengan AI sebagai lawannya tanpa memilih menu *New Game* sebagai konfigurasi awal dari aplikasi. Kemudian pada bagian bawah tampilan utama terdapat *timer* yang akan menghitung waktu dari permainan kedua pemain secara bergantian dan otomatis dijalankan ketika bidak putih berjalan pertama kali.

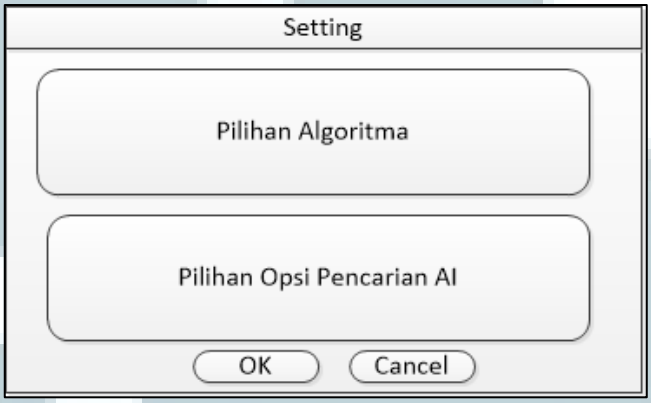
2. Menu *New Game*

Gambar 3.11 Desain Antarmuka Menu *New Game*

Tampilan antarmuka ini akan tampil saat pengguna atau pemain memilih tombol *New Game* yang ada di tampilan utama. Menu ini berisi tiga pilihan kombinasi jenis permainan yang ingin dilakukan oleh pengguna seperti *Human VS Human*, *Human* VS AI, dan AI VS AI. Permainan akan langsung dimulai ketika pengguna memilih tombol *OK* pada tampilan *New Game*.

Jika pengguna memilih tombol *Cancel* maka tampilan *New Game* akan ditutup.

3. Menu *Setting*



Gambar 3.12 Desain Antarmuka Menu *Setting*

Halaman ini akan ditampilkan di dalam aplikasi saat pengguna atau pemain memilih tombol *Setting* yang ada pada tampilan utama aplikasi. Dalam menu ini pengguna aplikasi dapat melakukan konfigurasi pada kecerdasan buatan seperti algoritma yang digunakan dan pembatasan kedalaman pencarian. Konfigurasi akan disimpan dan dijalankan saat pengguna memilih tombol *OK*. Jika pengguna ingin membatalkan perubahan konfigurasi, maka harus memilih tombol *Cancel*.

4. Menu *About*

Gambar 3.13 Desain Antarmuka Menu *About*

Tampilan ini akan ditampilkan ketika pengguna memilih tombol *About* yang ada pada tampilan utama aplikasi. Tampilan ini akan berisikan informasi mengenai aplikasi dan pembangun aplikasi, yaitu penulis sendiri.