BAB III

METODOLOGI PENELITIAN DAN PERANCANGAN

3.1 Metodologi Penelitian

Adapun tahap-tahap metodologi yang dilakukan dalam penelitian ini yaitu sebagai berikut.

1) Studi literatur

Dalam tahap ini, dilakukan studi untuk mengumpulkan informasi mengenai teori dan konsep yang berhubungan dengan penelitian, diantaranya adalah permainan catur, kecerdasan buatan, algoritma Minimax, Alpha-Beta Pruning, dan Killer Heuristic. Referensi yang digunakan berupa buku, *web site*, jurnal, artikel, *conference proceeding*, dan lain-lain.

2) Perancangan

Proses perancangan dilakukan dengan membuat *flowchart* terkait dengan algoritma Alpha-Beta Pruning dengan atau tanpa optimasi Killer Heuristic. Selain itu, dibuat juga *flowchart* alur aplikasi dan rancangan antarmuka aplikasi sebagai pedoman pembuatan aplikasi dalam penelitian.

3) Pembangunan

Berdasarkan rancangan yang telah dijabarkan, tahap ini menjelaskan komponen-komponen yang digunakan dalam pembuatan aplikasi dan detail implementasi dari algoritma Alpha-Beta Pruning dengan atau tanpa optimasi Killer Heuristic.

4) Pengujian

Pengujian efisiensi dilakukan dari kedua algoritma dengan cara mengukur nilai rata-rata dari jumlah simpul (*node*) yang terbentuk serta waktu yang dibutuhkan

kedua algoritma dalam membuat suatu keputusan langkah dalam beberapa pertandingan. Proses pengujian dilakukan dengan cara melakukan simulasi pertandingan dua buah AI dan pengukurannya dilakukan ke setiap entitas mulai dari tingkat kedalaman dua hingga lima. Proses simulasi divisualisasikan ke dalam aplikasi permainan catur yang telah dibuat.

5) Evaluasi

Evaluasi dilakukan berdasarkan hasil pengujian yang telah dilakukan untuk menjawab rumusan masalah yang telah dijabarkan sebelumnya, yaitu mengimplementasikan algoritma Alpha-Beta Pruning dengan dan tanpa optimasi Killer Heuristic serta membandingkannya berdasarkan ukuran pohon yang dibentuk dan waktu yang dibutuhkan kedua algoritma dalam membuat suatu keputusan langkah pada permainan catur.

6) Konsultasi dan penulisan

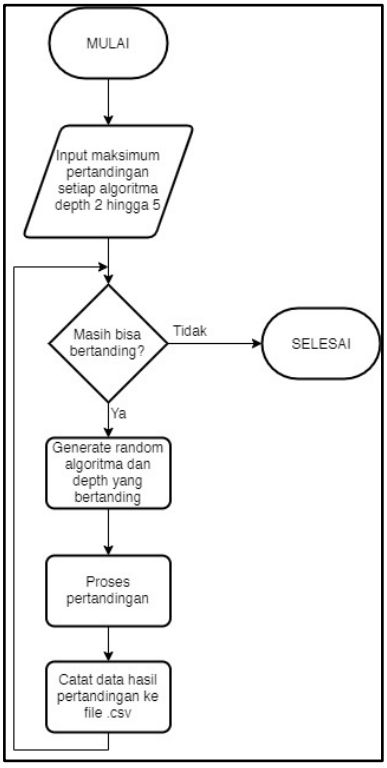
Pada tahapan ini, dilakukan konsultasi dengan pembimbing dan penulisan laporan dengan tujuan mendokumentasikan proses penelitian yang telah dilakukan dan menyimpulkan hasil akhir yang didapat, serta dapat dijadikan sebagai bukti bahwa penelitian telah dilakukan dan diselesaikan.

3.2 Perancangan

Proses perancangan dimulai dengan membuat *flowchart* alur aplikasi dan rancangan antarmuka aplikasi, serta *flowchart* untuk kedua algoritma (Alpha-Beta Pruning dengan atau tanpa Killer Heuristic) pada permainan catur.

3.2.1 Flowchart Alur Aplikasi

Flowchart alur aplikasi yang digunakan untuk melakukan penelitian ini dapat dilihat pada Gambar 3.1. Pada menu, terdapat *input* maksimum pertandingan yang dapat dilakukan setiap algoritma di setiap kedalamannya. Lalu, aplikasi akan menentukan algoritma apa dan tingkat kedalaman berapa yang akan bertanding secara acak. Setelah itu, proses pertandingan catur dilakukan. Kemudian, data hasil pertandingan akan ditulis secara otomatis ke dalam *file* berekstensi .csv. Lalu, aplikasi akan secara otomatis melanjutkan ke pertandingan selanjutnya apabila masih ada yang dapat bertanding.

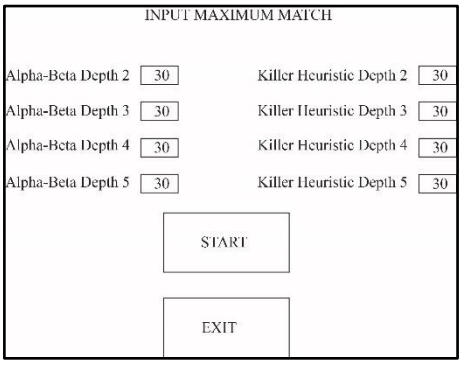


Gambar 3.1 *Flowchart* alur aplikasi

3.2.2 Rancangan Antarmuka Aplikasi

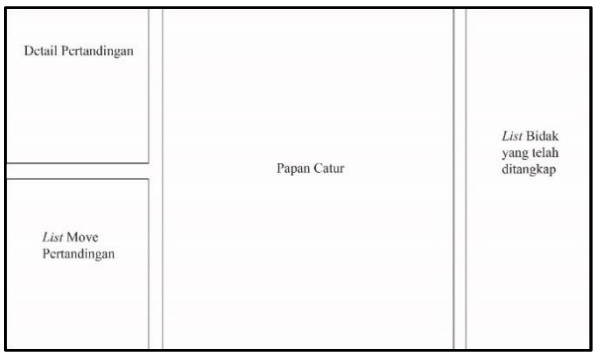
Rancangan antarmuka aplikasi yang digunakan untuk melakukan penelitian ini terdiri dari dua rancangan, yaitu menu dan pertandingan catur. Pada Gambar 3.2,

rancangan antarmuka menu terdiri dari beberapa *input text box field* yang merepresentasikan maksimum pertandingan yang dapat dilakukan untuk suatu algoritma pada suatu *depth*. *Button start* digunakan untuk memulai pertandingan. *Button exit* digunakan untuk keluar dari aplikasi.



Gambar 3.2 Rancangan antarmuka menu

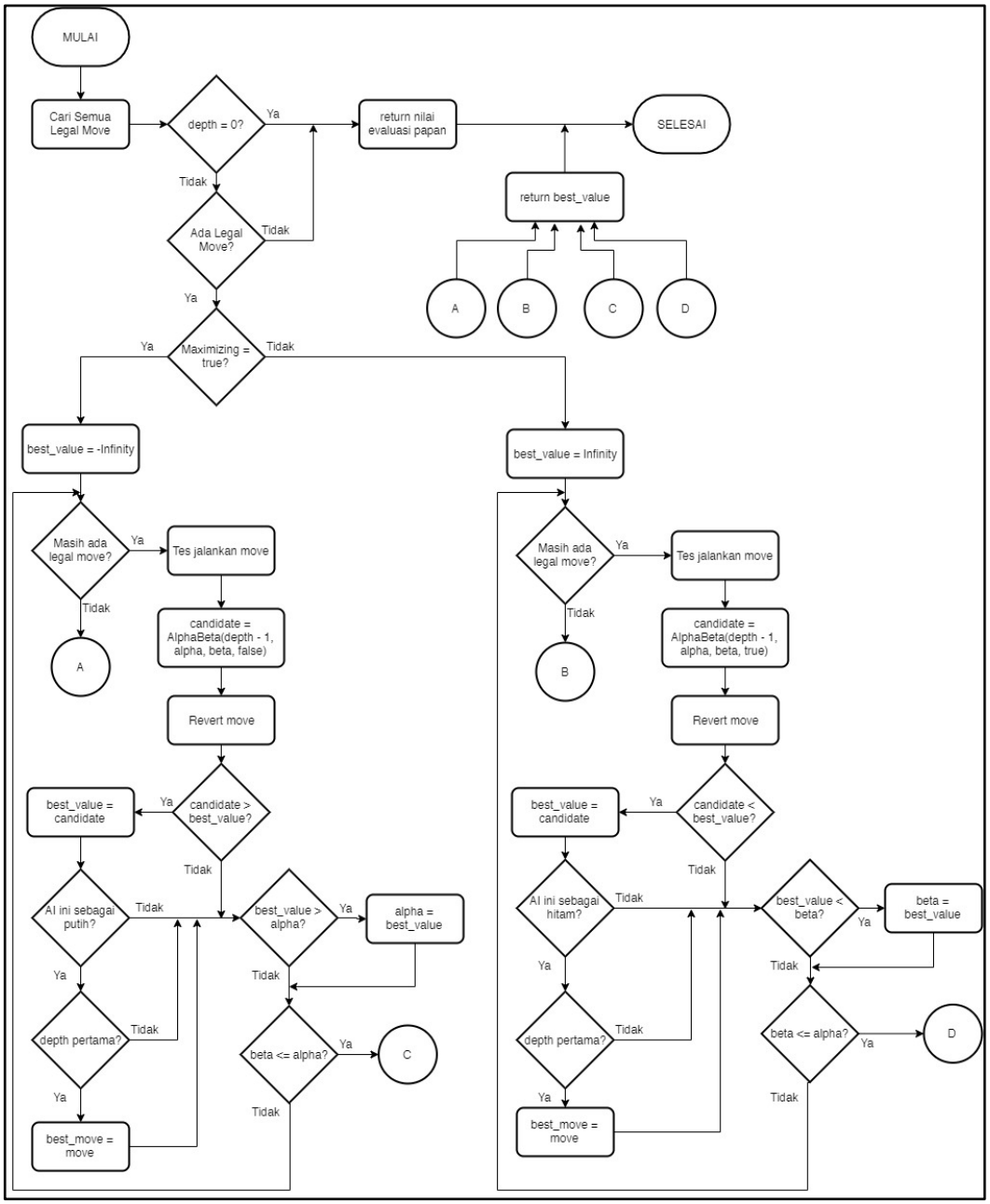
Pada Gambar 3.3, rancangan antarmuka pertandingan catur menampilkan papan catur sebagai visualisasi pertandingan, *list* bidak yang telah ditangkap oleh putih dan hitam, *list move* pertandingan yang dilakukan oleh masing-masing AI, dan detail pertandingan yang menampilkan data-data pertandingan. Detail pertandingan menunjukkan algoritma apa saja yang sedang bertanding beserta tingkat kedalamannya, rata-rata simpul yang dibuat dan waktu yang dipakai masing-masing AI dalam satu pertandingan, serta total simpul yang dibuat dan waktu yang dipakai masing-masing AI dalam satu pertandingan.



Gambar 3.3 Rancangan antarmuka pertandingan catur

3.2.3 Flowchart Alpha-Beta Pruning

Flowchart algoritma Alpha-Beta Pruning menggambarkan alur proses algoritma Alpha-Beta Pruning pada aplikasi permainan catur yang dibuat. *Flowchart* algoritma Alpha-Beta Pruning dapat dilihat pada Gambar 3.4.



Gambar 3.4 *Flowchart* Alpha-Beta Pruning

Algoritma Alpha-Beta Pruning menggunakan *function* yang rekursif dengan empat parameter, yaitu *integer depth*, *alpha*, beta, dan *boolean maximizing*. Parameter *depth* digunakan untuk melakukan rekursif pada algoritma hingga mencapai nilai nol sebagai salah satu *base case*-nya. Parameter *alpha* merepresentasikan nilai maksimum yang dimiliki AI putih selama proses pencarian. Sedangkan parameter beta merepresentasikan nilai minimum yang dimiliki AI hitam selama proses pencarian. Parameter *maximizing* merepresentasikan giliran putih atau hitam di suatu kedalaman dalam proses pencarian. Nilai *alpha* dan beta digunakan untuk melakukan pemotongan cabang (*cut-off*). Nilai *alpha* selalu diinisialisasi dengan nilai negatif terbesar dan nilai beta diinisialisasi dengan nilai positif terbesar.

Algoritma diawali dengan mengambil semua *legal moves* yang tersedia. Lalu dilakukan pengecekan apakah *depth* telah mencapai batasnya atau tidak ada *legal move* (*terminal state*), bila salah satu kondisi tersebut bernilai *true* maka *function* akan mengembalikkan nilai evaluasi papan saat itu. Namun bila kedua kondisi bernilai *false*, maka algoritma dilanjutkan. Sehingga, nilai evaluasi papan dari *statement* tersebut hanya akan diberikan pada simpul *leaf* pada *tree*.

Pemisahan simpul *tree* apakah giliran putih atau hitam dilakukan dengan pengecekan nilai *maximizing*. Apabila bernilai *true* maka simpul tersebut merupakan giliran putih untuk mencari nilai maksimum. Namun, bila *maximizing* bernilai *false* maka simpul tersebut merupakan giliran hitam untuk mencari nilai minimum (*minimizing*).

Saat melakukan *maximizing*, *best_value* diinisialisasi dengan nilai negatif terbesar. Sedangkan saat *minimizing*, nilai *best_value* diinisialisasi dengan nilai

positif terbesar. Kemudian dilakukan *looping* pada setiap *legal moves*. Bila *looping* telah selesai dilakukan, maka akan mengembalikkan nilai *best_value*. Pengembalian nilai tersebut digunakan saat algoritma kembali ke *parent* simpul.

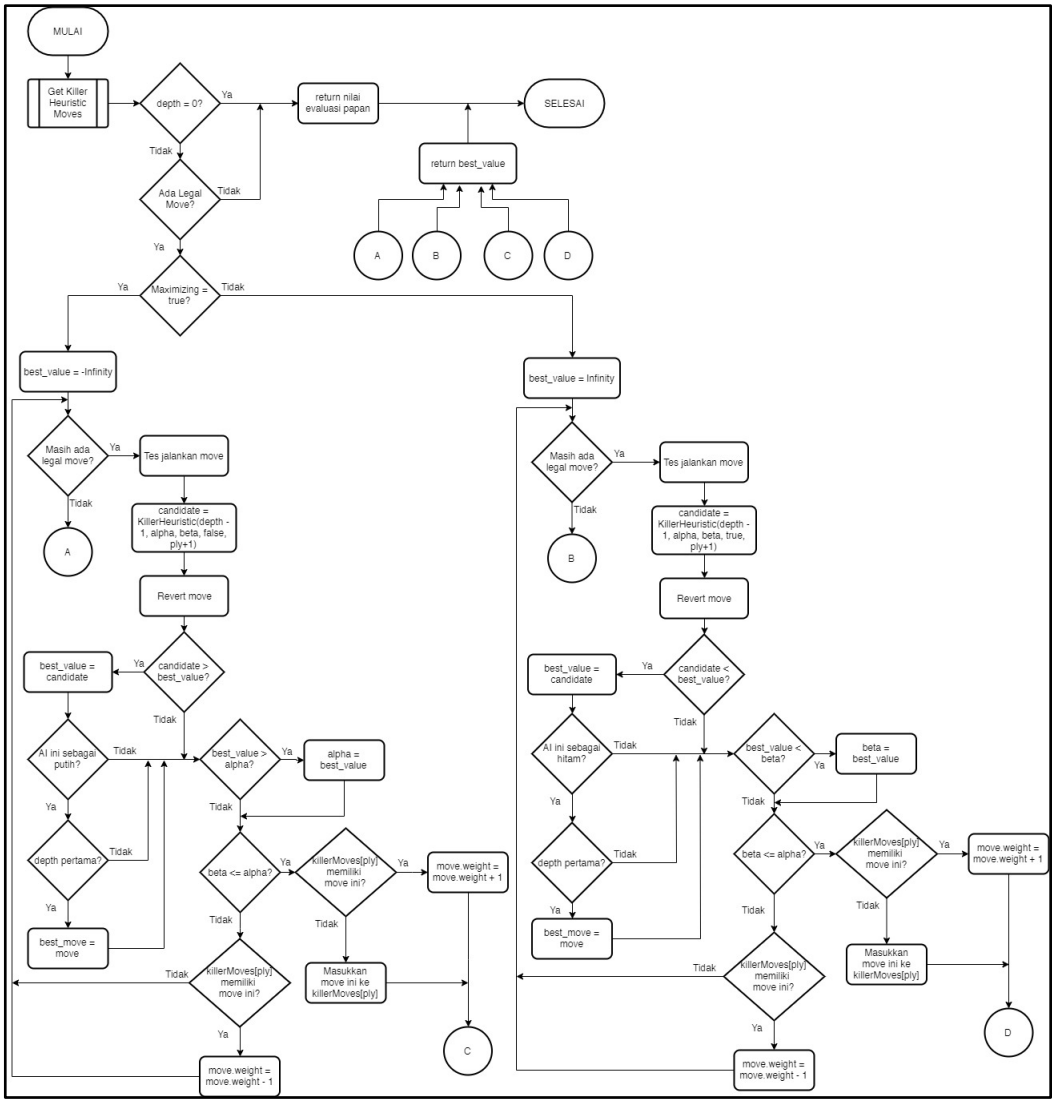
Looping diawali dengan melakukan tes menjalankan *move* tersebut. Setelah itu dilakukan pemanggilan rekursif untuk mendapatkan nilai *candidate* (nilai dari evaluasi papan atau *best_value*), bila sedang *maximizing* maka parameter *function* terakhirnya menggunakan nilai *false*, dan bila sedang *minimizing* maka digunakan nilai *true*. Setelah itu, *revert move* tersebut agar papan kembali seperti semula.

Kemudian, dilakukan pengecekan apakah nilai *candidate* lebih baik dari pada nilai *best_value*. Bila lebih baik, maka nilai *best_value* diganti dengan nilai *candidate*. Selain itu, dilakukan pengecekan juga apakah simpul tersebut merupakan *depth* teratas dari *tree* dan juga apakah AI bertindak sebagai putih (*maximizing*) atau hitam (*minimizing*). Bila *true*, maka dapat ditarik kesimpulan bahwa *move* tersebut adalah *move* terbaik (*best_move*) yang akan dipilih AI.

Selanjutnya, saat *maximizing* maka nilai *alpha* akan diganti oleh nilai terbesar antara nilai *alpha* atau nilai *best_value*. Sedangkan saat *minimizing*, nilai beta akan diganti oleh nilai terkecil antara nilai beta atau nilai *best_value*. Lalu, yang terakhir dilakukan pengecekan apakah nilai beta lebih kecil atau sama dengan nilai *alpha*. bila kondisi tersebut *true*, maka *cut-off* terjadi disini (*break looping*) dan mengembalikkan nilai *best_value*. Namun bila kondisi tersebut *false*, maka *looping* pada *legal moves* akan dilanjutkan.

3.2.4 Flowchart Alpha-Beta Pruning dengan Killer Heuristic

Flowchart algoritma Alpha-Beta Pruning dengan optimasi Killer Heuristic tidak berbeda jauh dengan *flowchart* algoritma Alpha-Beta Pruning tanpa optimasi. *Flowchart* algoritma Alpha-Beta Pruning dengan optimasi Killer Heuristic dapat dilihat pada Gambar 3.5.



Gambar 3.5 *Flowchart* Alpha-Beta Pruning dengan optimasi Killer Heuristic

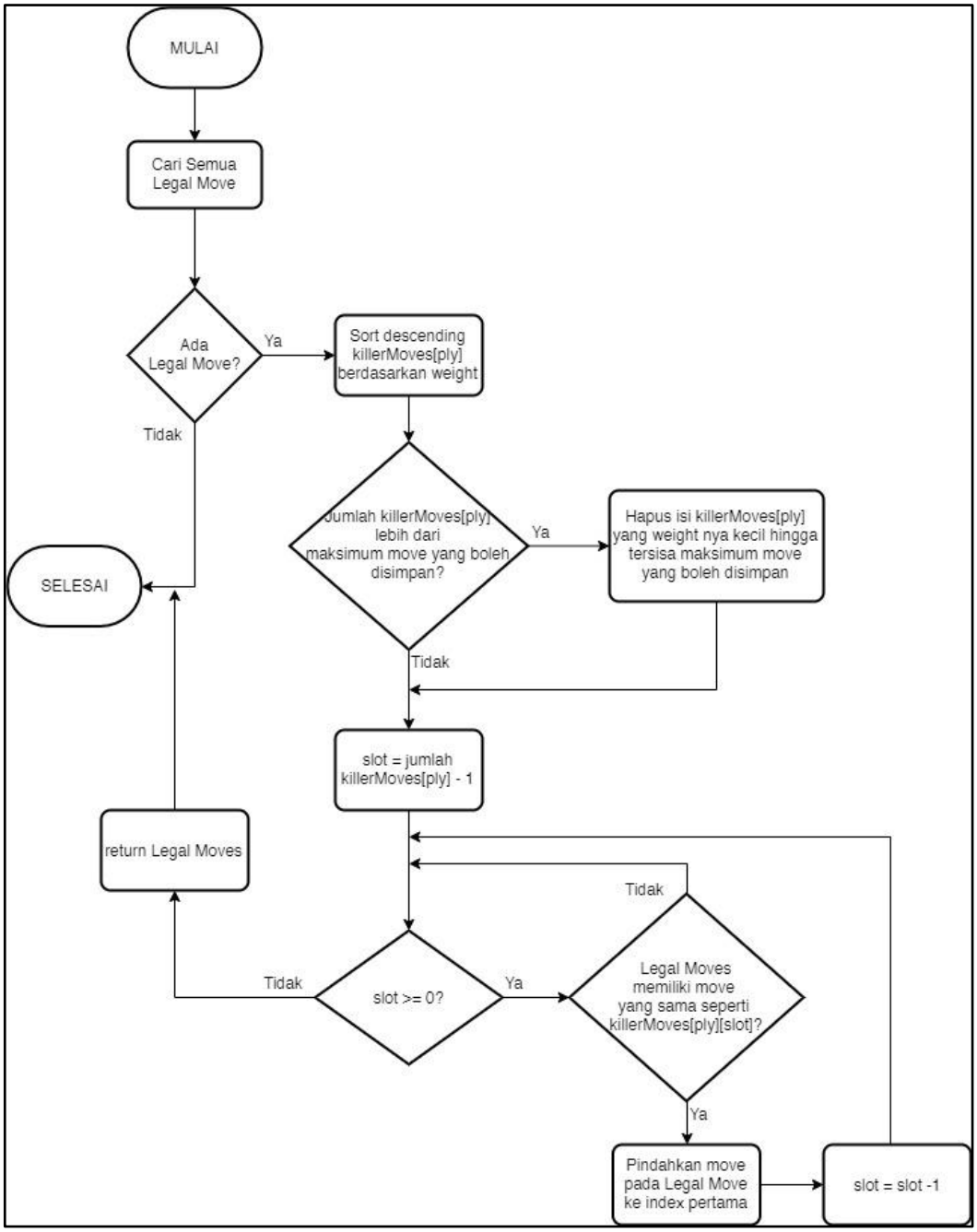
Pada algoritma Alpha-Beta Pruning dengan optimasi Killer Heuristic, *function* rekursif memiliki tambahan satu parameter lain, yaitu *ply* yang merepresentasikan jarak kedalaman suatu *node* dengan *root* pada *tree*. Algoritma diawali dengan

pemanggilan modul *Get Killer Heuristic Moves* untuk mendapatkan semua *legal moves* yang dapat dilakukan.

Perbedaan lainnya adalah ketika akan melakukan pemotongan cabang (*cut-off*). Bila kondisi pengecekan nilai beta lebih kecil atau sama dengan nilai alpha adalah *true*, maka sebelum melakukan pemotongan akan dilakukan pengecekan tambahan, yaitu apabila *killer moves* (killerMoves[ply]) tidak memiliki *move* yang sedang diiterasi, maka *move* tersebut dianggap *killer* dan dimasukkan kedalam killerMoves[ply].

Namun apabila *killer moves* memiliki *move* tersebut, maka nilai *weight* pada *move* tersebut ditambah satu. Nilai *weight* ini merepresentasikan sebagai efektifitas suatu *killer move* untuk melakukan pemotongan cabang. Apabila saat kondisi pengecekan nilai beta lebih kecil atau sama dengan nilai alpha adalah *false* dan *move* tersebut berada pada killerMoves[ply], maka dapat diartikan bahwa efektifitas *killer move* berkurang karena gagal melakukan pemotongan cabang. Sehingga, nilai *weight* pada *move* tersebut dikurang satu.

Flowchart modul *Get Killer Heuristic Moves* pada algoritma yang digunakan untuk mendapatkan semua *legal moves* pada suatu kedalaman *tree* dapat dilihat pada Gambar 3.6.



Gambar 3.6 *Flowchart* modul *Get Killer Heuristic Moves*

Pada *flowchart* Gambar 3.6, algoritma mengambil semua *legal moves* dengan cara yang sama seperti Alpha-Beta Pruning. Kemudian, bila tidak ada *legal move* maka itu artinya telah mencapai *terminal state* sehingga tidak ada data yang dikembalikkan. Namun bila ada *legal moves*, maka algoritma akan melakukan pengurutan (*sort*) pada *killer moves* terlebih dahulu secara *descending* berdasarkan

weight setiap *move*-nya. Lalu, karena *killer moves* memiliki batas maksimum yang boleh disimpan hanya satu atau dua *move* (dalam penelitian ini *killer moves* menyimpan maksimum dua data *move*), maka dua *killer moves* yang memiliki *weight* terbesar tetap disimpan, sedangkan semua *killer moves* lainnya dihapus.

Setelah itu, dilakukan *looping* untuk setiap *killer moves* yang tersisa. Apabila *legal moves* yang telah didapat pada langkah sebelumnya memiliki *move* yang sama dengan *killer moves*, maka dapat diartikan bahwa *killer move* tersebut masih merupakan *move* yang *valid* dalam kondisi pencarian saat itu. Sehingga, *move* yang berada pada *legal moves* tersebut dipindahkan ke *index* pertama (*move ordering*) agar nantinya algoritma melakukan pencarian nilai untuk *move* tersebut terlebih dahulu. Setelah melakukan *move ordering*, modul ini mengembalikkan data semua *legal moves* tersebut.