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.

Perpustakaan Univeristas Multimedia Nusantara

BAB II

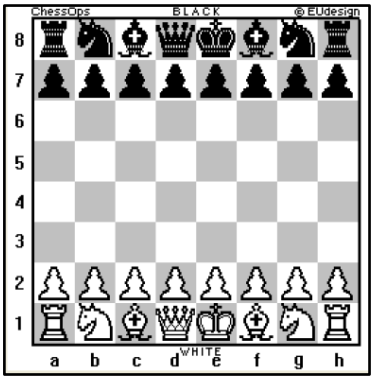
LANDASAN TEORI

2.1 Catur

Ada beberapa jenis permainan catur dengan gaya permainan yang berbeda juga. Jenis permainan catur yang dibahas dalam penelitian ini merupakan catur pada umumnya yang sesuai dengan peraturan organisasi catur internasional, yaitu World Chess Federation (FIDE).

FIDE (2018) menjelaskan bahwa permainan catur adalah permainan yang dimainkan oleh dua lawan yang menggerakkan bidak mereka di papan persegi yang disebut papan catur. Papan catur terdiri dari persegi delapan kali delapan dengan warna terang dan gelap secara bergantian. Delapan persegi pada kolom vertikal disebut "*files*" dan diberi label 'a' hingga 'h'. Sedangkan delapan persegi pada baris horizontal disebut "*ranks*" dan diberi label 1 hingga 8. Persegi dengan warna sama yang membentuk garis lurus disebut "*diagonal*".

Pemain dengan bidak putih memulai langkah pertama, kemudian pemain bergerak secara bergantian dengan pemain dengan bidak hitam. Tujuan setiap pemain adalah *checkmate* atau skakmat, yaitu menyerang dan menempatkan raja lawan sedemikian rupa sehingga lawan tidak memiliki langkah yang *legal* lagi. Jika terjadi suatu posisi dimana tidak ada pemain yang dapat melakukan *checkmate*, maka permainan dianggap *draw* atau seri.

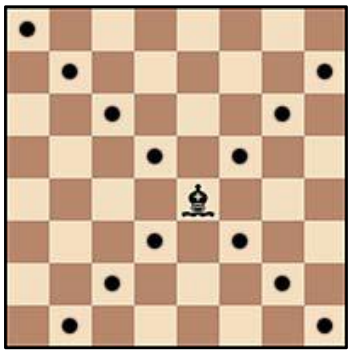


Gambar 2.1 Posisi awal permainan catur
(Zirbel, 2006)

Berdasarkan *handbook* yang dikeluarkan oleh FIDE (2018), ada beberapa jenis bidak catur, diantaranya sebagai berikut.

1) *Bishop* atau gajah

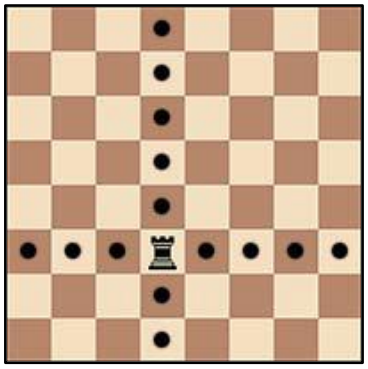
Bishop dapat bergerak ke arah diagonal mana saja, tetapi tidak dapat melompati bidak lain. Tiap pemain memulai permainan dengan dua *bishop*. Masing-masing *bishop* diletakkan di persegi berwarna terang dan gelap di sebelah *knight*.



Gambar 2.2 Posisi pergerakan *bishop*
(FIDE, 2018)

2) *Rook* atau benteng

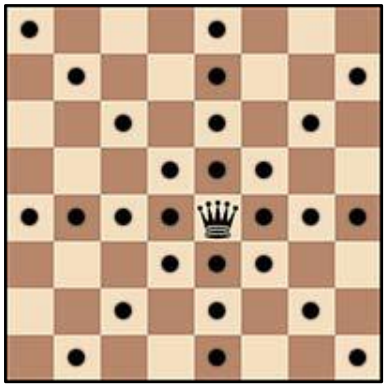
Rook dapat bergerak searah dengan *file* atau *rank*, tetapi tidak dapat melompati bidak lain. Tiap pemain memulai permainan dengan dua *Rook*. Masing-masing *Rook* diletakkan di pojok kiri dan kanan papan catur.



Gambar 2.3 Posisi pergerakan *rook*
(FIDE, 2018)

3) *Queen* atau ratu

Queen dapat bergerak searah dengan *file*, *rank*, maupun diagonal, tetapi tidak dapat melompati bidak lain. Tiap pemain memulai permainan dengan satu *queen* yang diletakkan di sebelah *king*.



Gambar 2.4 Posisi pergerakan *queen*
(FIDE, 2018)

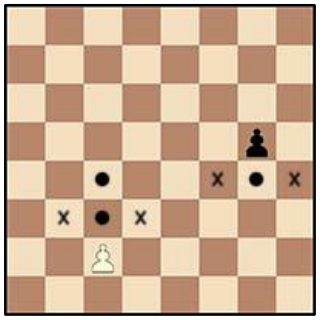
4) *Knight* atau kuda

Knight dapat bergerak ke suatu persegi terdekat dari posisinya tetapi tidak berada di *file, rank*, atau *diagonal* yang sama. Tiap pemain memulai permainan dengan dua *knight* yang masing-masing diletakkan di sebelah *rook*.

Gambar 2.5 Posisi pergerakan *knight*
(FIDE, 2018)

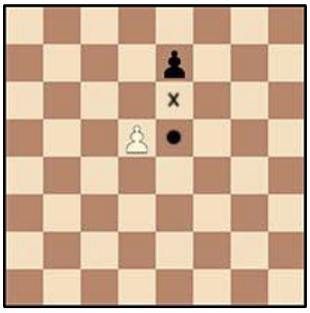
5) *Pawn* atau pion

Pawn dapat bergerak maju satu langkah ke depan pada *file* yang sama, selama tidak ada bidak lain yang menghalanginya. Khusus pada pergerakan pertama, pawn dapat bergerak maju satu langkah atau dua langkah, selama tidak ada bidak lain yang menghalanginya. *Pawn* juga dapat bergerak satu langkah *diagonal* di depannya hanya untuk menangkap bidak lawan bila ada. Setiap pemain memiliki delapan buah *pawn* yang masing-masing diletakkan di depan delapan bidak lainnya.



Gambar 2.6 Posisi pergerakan *pawn*
(FIDE, 2018)

Pawn juga dapat menyerang satu langkah secara *diagonal* bila ada *pawn* lawan yang bergerak maju dua langkah dalam satu giliran, dan hanya dapat dilakukan tepat setelah giliran tersebut. Langkah ini disebut juga dengan *en passant*.

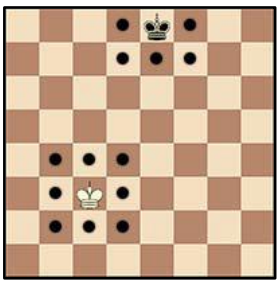


Gambar 2.7 *En Passant* pada *pawn*
(FIDE, 2018)

Selain itu, *pawn* yang berhasil mencapai *rank* terjauh dari posisi awalnya harus ditukarkan dengan queen, *rook*, *bishop*, atau *knight* yang baru sebagai bagian dari satu langkah yang sama pada persegi yang sama. Pilihan pemain tidak ada limitasi pada bidak yang sudah di tangkap sebelumnya. Pertukaran *pawn* dengan bidak lain ini disebut dengan *promotion*.

6) *King* atau raja

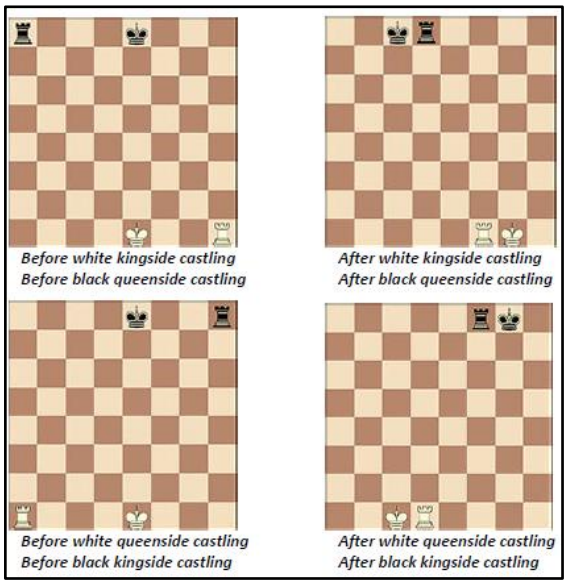
King dapat bergerak satu langkah di dekatnya ke arah manapun selama persegi tersebut tidak diserang oleh satu atau lebih bidak lawan. Setiap pemain hanya memiliki satu buah *king*.



Gambar 2.8 Posisi pergerakan *king*
(FIDE, 2018)

King juga dapat melakukan suatu langkah khusus yang disebut dengan *castling* atau rokade, yaitu pergerakan dimana *king* dan *rook* milik pemain yang berada di *rank* pertama pemain terhitung sebagai satu pergerakan yang dieksekusi seperti ini:

king di kirimkan dari posisinya mendekati dua langkah ke arah *rook*, lalu *rook* di kirimkan ke persegi dimana *king* telah melompatinya.



Gambar 2.9 *Castling* pada *king* dan *rook*
(FIDE, 2018)

Langkah rokade memiliki beberapa syarat, antara lain:

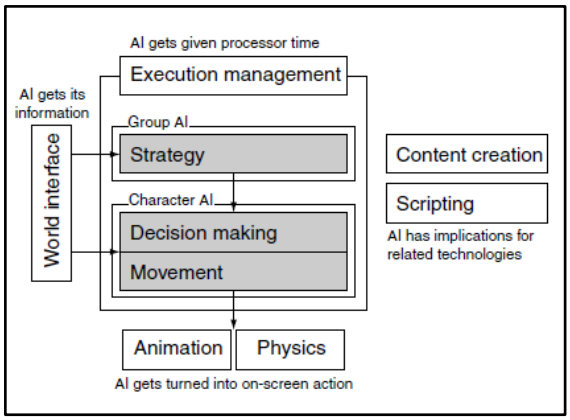
a. Rokade sudah tidak dapat dilakukan jika *king* pernah bergerak, atau dengan *rook* yang sudah pernah bergerak.
b. Rokade tidak dapat dilakukan secara sementara jika persegi dimana *king* berdiri, atau persegi yang harus dilewatinya sedang diserang oleh bidak lawan, atau jika ada bidak lain yang berada di antara *king* dan *rook* yang ingin dilakukan rokade.

King dikatakan *check* atau skak bila sedang diserang oleh satu atau lebih bidak lawan, lalu pemain dibatasi dalam melakukan pergerakannya, yaitu pemain harus meninggalkan status *check* tersebut dengan cara menggerakkan *king* ke posisi lain yang tidak sedang diserang, menangkap bidak lawan yang menyerang, atau menutup serangan lawan dengan menempatkan bidak lain di antaranya. Apabila

tidak ada langkah yang *legal* untuk meninggalkan status *check* tersebut, maka pemain telah kalah dengan status yang disebut *checkmate* atau skakmat.

2.2 Kecerdasan Buatan

Ada banyak bidang yang mempelajari tentang kecerdasan buatan dengan kegunaan yang berbeda-beda pula. Kecerdasan buatan yang dibahas dalam penelitian ini terfokus pada kecerdasan buatan yang digunakan untuk permainan komputer khususnya *board game* atau permainan papan. Menurut Yannakakis dan Togelius (2018), permainan papan komputer telah menjadi bidang yang terkenal untuk AI karena kaku dan banyak batasan, tetapi memiliki lingkungan pengambilan keputusan yang kompleks.



Gambar 2.10 Ilustrasi model AI
(Millington dan Funge, 2009)

Menurut Yannakakis dan Togelius (2018), kecerdasan buatan atau *artificial intelligence* (AI) adalah komputer yang mampu melakukan tugas berpikir yang dapat dilakukan oleh manusia dan hewan. Tugas AI terbagi dalam tiga tingkatan, yaitu pergerakan, pembuat keputusan, dan strategi, seperti yang dapat dilihat pada Gambar 2.10. Pergerakan dan pembuat keputusan berisi algoritma yang bekerja pada basis antar karakter, dan strategi mengoperasikan keseluruhan atau sebagian

tim. Tidak semua aplikasi membutuhkan semua tingkatan tersebut, seperti permainan catur hanya membutuhkan tingkatan strategi.

Yannakakis dan Togelius (2018) juga menjelaskan tentang algoritma, yaitu proses langkah demi langkah yang menghasilkan solusi untuk permasalahan AI. Sedangkan struktur data menyimpan data sedemikian rupa agar algoritma dapat secara cepat memanipulasinya untuk mencapai suatu solusi.

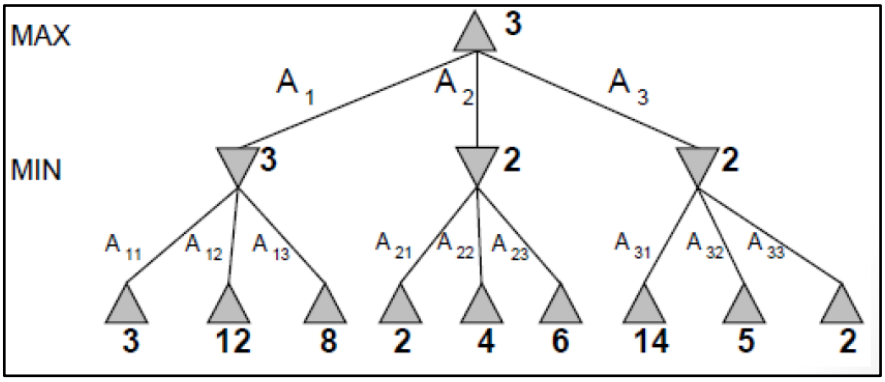
Menurut Millington dan Funge (2009), algoritma yang digunakan suatu AI untuk permainan *turn-based* dirancang untuk *two-player*, *zero-sum*, *perfect information games*. Mereka juga mengatakan, bahwa semua permainan *turn-based* dapat direpresentasikan sebagai struktur data *tree*. Sebagai contohnya pada *board games*, setiap simpul dalam *tree* merepresentasikan posisi atau kondisi papan, sedangkan setiap cabang merepresentasikan satu langkah yang mungkin dilakukan. Selain itu, suatu langkah menuntun dari satu posisi papan ke posisi papan lainnya. Lalu karena permainannya bersifat *turn-based*, maka papan hanya akan berubah ketika satu pemain membuat suatu langkah.

Millington dan Funge (2009) juga menjelaskan, bahwa banyaknya cabang dari setiap papan sama dengan banyaknya langkah yang memungkinkan yang dapat dibuat oleh pemain. Selain itu, ada beberapa posisi papan yang tidak memiliki langkah yang memungkinkan dan merepresentasikan akhir dari permainan, posisi ini disebut *terminal positions* atau *terminal state*.

2.3 Minimax

Menurut Yannakakis dan Togelius (2018), untuk permainan *single-player*, algoritma pencarian sederhana dapat digunakan untuk mencari jalan ke status

permainan yang optimal. Namun, untuk permainan permusuhan pada *two-player*, ada pemain lain yang mencoba untuk menang juga, sehingga aksi untuk setiap pemain sangat bergantung pada aksi yang dilakukan oleh pemain lain. Oleh sebab itu, permainan permusuhan pada *two-player* dibutuhkan *adversarial search*, dimana ada aksi dua atau lebih pemain yang bermusuhan. Algoritma dasar mengenai *adversarial search* ini adalah Minimax. Tujuan dari algoritma Minimax adalah memiliki langkah untuk bergerak dengan nilai *minimax* tertinggi (Louis, 2013). Contoh ilustrasi Minimax untuk dua pemain dapat dilihat pada Gambar 2.11.



Gambar 2.11 Ilustrasi *tree* pada algoritma Minimax
(Louis, 2013)

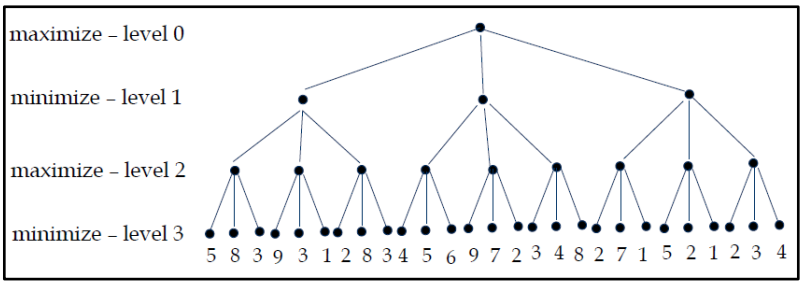
Permainan catur memiliki sekitar 10^{40} simpul yang berbeda (Russel dan Norvig, 2010), sehingga diberikan batasan maksimum kedalaman dalam melakukan pencarian (Millington dan Funge, 2009).

2.4 Alpha-Beta Pruning

Menurut Russel dan Norvig (2010), masalah yang dihadapi oleh algoritma Minimax adalah banyaknya kodisi permainan yang harus ditelusuri merupakan eksponensial di kedalaman *tree*. Masalah tersebut dapat diatasi dengan cara memotong sebagian cabang *tree*. Caranya dengan mencari kemungkinan untuk

menentukan keputusan Minimax tanpa melihat setiap simpul yang ada. Menurut Russel dan Norvig (2010), teknik yang umum digunakan adalah algoritma Alpha-Beta Pruning. Menurut Millington dan Funge (2009), Alpha-Beta Pruning memungkinkan AI untuk mengabaikan bagian cabang yang tidak mungkin berisi langkah terbaik.

Alpha-Beta Pruning menyimpan dua nilai, yaitu nilai *alpha* yang merepresentasikan nilai terbaik yang bisa diambil oleh pemain Max, sedangkan nilai *beta* merepresentasikan nilai terbaik untuk pemain Min (Timothy, 2014). Utomo (2005) memberikan contoh kondisi awal yang akan ditinjau dengan menggunakan metode Alpha-Beta Pruning yang dapat dilihat pada Gambar 2.12.



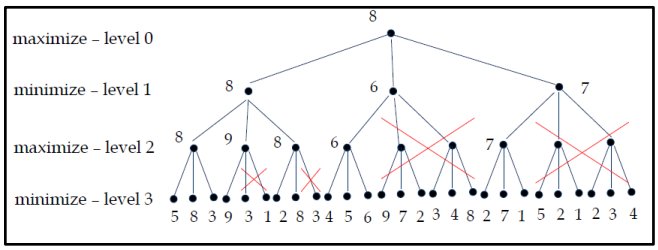
Gambar 2.12 Ilustrasi kondisi awal *tree* pada Alpha-Beta Pruning (Utomo, 2005)

Utomo (2005) memaparkan contoh pengerjaan algoritma Alpha-Beta Pruning seperti berikut ini.

a. Hampiri simpul pertama pada *leaf* dengan nilai lima, naik ke *parent* pada level dua dan masukan nilai lima, lalu hampiri *leaf* dengan nilai delapan, karena delapan lebih besar dari lima maka ganti nilai *parent* dengan delapan, lalu hampiri *leaf* dengan nilai tiga.
b. Setelah ketiga *leaf* pertama terhampiri, naik lebih tinggi lagi ke level satu, masukan nilai delapan.

c. Hampiri simpul keempat pada *leaf* dengan nilai sembilan, naik ke *parent* pada level dua dan masukan nilai sembilan. Jika menghampiri *leaf* berikutnya, akan mencari nilai yang lebih tinggi dari sembilan, sementara pada level satu sedang mencari nilai yang lebih kecil dari delapan, maka *leaf* dengan nilai tiga dan satu dipotong (tidak dihampiri).

d. Begitu seterusnya hingga didapatkan hasil akhirnya seperti Gambar 2.13.



Gambar 2.13 Ilustrasi kondisi akhir *tree* algoritma Alpha-Beta Pruning (Utomo, 2005)

Berikut ini adalah *pseudocode* untuk Alpha-Beta Pruning yang telah dikutip oleh Hariadi dkk. (2017).

```
function alphabeta(node, depth, α, β, maximizingPlayer)
    if depth = 0 or node is a terminal node
        return the heuristic value of node
    if maximizingPlayer
        v := -∞
        for each child of node
            v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
            α := max(α, v)
            if β ≤ α
                break (* β cut-off *)
        return v
    else
        v := ∞
        for each child of node
            v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
            β := min(β, v)
            if β ≤ α
                break (* α cut-off *)
        return v
```

Gambar 2.14 *Pseudocode* algoritma Alpha-Beta Pruning (Hariadi dkk., 2017)

Algoritma Alpha-Beta Pruning ini sangatlah populer di bidang catur sejak tahun 1997. AI yang diberi nama "Deep Blue" yang diciptakan oleh *The International Business Machines Corporation* (IBM) menggunakan algoritma Alpha-Beta Pruning (Hsu, 1999). Pada tahun 1997, Deep Blue telah membuktikan

bahwa algoritma Alpha-Beta Pruning dapat mengalahkan Garry Kasparov, sang juara dunia catur pada tahun 1986 hingga 2005. Selain itu, AI yang diberi nama "Hydra" juga menggunakan algoritma Alpha-Beta Pruning dan berhasil mengalahkan peringkat tujuh *grandmaster* pada tahun 2005, Michael Adams (Ertel, 2017). Ertel (2017) mengatakan, Deep Blue dapat melakukan proses pencarian hingga *depth 12* dan Hydra mencapai *depth* 18. Hal tersebut dikarenakan kedua AI tersebut menggunakan *hardware* yang sangat mahal pada saat itu. Deep Blue menggunakan superkomputer milik IBM (Hsu, 1999), sedangkan Hydra menggunakan 64 *parallel processors* (Ertel, 2017).

2.5 Killer Heuristic

Alpha-Beta Pruning bergantung pada urutan pencarian langkahnya, sehingga algoritma tersebut bekerja sangat baik ketika langkah terbaik ditelusuri pertama. Namun, kasus terburuknya dapat menghasilkan kompleksitas yang sama dengan algoritma Minimax (Carolus, 2006). Polk dan Oommen (2016) mengatakan bahwa optimasi Alpha-Beta Pruning dengan cara mengurutkan langkah (*move ordering*) yang populer adalah History Heuristic dan Killer Heuristic. Merekapun telah membuktikan bahwa Killer Heuristic lebih baik dibandingkan History Heuristic. Schaeffer (1989) juga mengatakan bahwa Killer Heuristic adalah algoritma yang paling populer untuk mengoptimasi algoritma Alpha-Beta Pruning. Baier (2015) mengatakan bahwa metode optimasi *move ordering* tersebut cukup efektif pada berbagai bidang.

Baier (2015) mengatakan bahwa metode optimasi Killer Heuristic menggunakan ide bahwa suatu *move* yang baik (yang dapat mengasilkan *cut-off*) pada suatu *state* mungkin juga menjadi *move* yang baik pada *state* lain yang mirip.

Menurut Schaeffer (1989), Killer Heuristic digunakan untuk mengingat setiap kedalaman langkah yang dicari yang terlihat paling banyak menyebabkan pemotongan (*killer move*). Lalu, ketika selanjutnya pada kedalaman yang sama pada *tree* tercapai, *killer move* ini diambil dan digunakan, jika valid pada posisi tersebut. Menurut Hashimoto dkk. (2007), *killer move* yang ditempatkan sebelum *move* lainnya dapat membuat algoritma sering menghasilkan *cut-off* lebih awal, sehingga akan menjadi lebih banyak *move* yang tidak diperlukan untuk tidak ditelusuri.

Huberman (1968) menjelaskan konsep dasar dari cara kerja Alpha-Beta Pruning dengan optimasi Killer Heuristic. Pada suatu simpul di suatu kedalaman pohon, riwayat cabang ke titik ini diberikan oleh semua posisi lawan untuk bergerak dimana AI telah memeriksa perjalanan ke simpul ini. Jika posisi lawan pada simpul ini ditemukan lebih baik daripada AI, maka AI mencari langkah sebelumnya yang diambil oleh AI yang menghasilkan posisi lawan tersebut, lalu mengingat langkah tersebut. Selanjutnya, setiap saat setelah itu jika langkah tersebut dilakukan lagi, maka posisi lawan terbaik itu akan diperiksa terlebih dahulu.

Huberman (1968) mengatakan, bahwa secara teori pencarian posisi dalam pohon adalah sama, sehingga langkah yang menuntun terbaik di suatu titik mungkin dapat menuntun posisi terbaik lagi di lain waktu. Dengan memeriksa posisi terbaik yang pertama, maka dapat mengeliminasi banyak perbandingan untuk posisi lainnya. Jika teori tersebut gagal, maka hanya akan kehilangan sedikit waktu.

Menurut Huberman (1968), teori tersebut bekerja sangat baik pada akhir pertandingan. Huberman menjelaskan, jika pemeriksaan dilakukan dari langkah terakhir ke posisi yang lebih baik selama pencarian *tree*, biasanya hanya ada satu atau dua langkah yang seperti itu (Winands dkk., 2004). Lalu, ketika posisi yang ditaruh pertama ternyata merupakan langkah yang diambil oleh AI, maka waktu yang dihemat cukup banyak daripada kompensasi waktu yang dibutuhkan dalam mengurutkan posisi.

Berdasarkan seluruh konsep dasar yang telah dijelaskan diatas, dapat disimpulkan bahwa Alpha-Beta Pruning dengan optimasi Killer Heuristic menyimpan satu atau dua langkah (*killer moves*) di setiap jarak kedalaman ketika terjadi *cut-off* pada algoritma Alpha-Beta Pruning. Kemudian, pada saat algoritma melakukan pencarian pada cabang lain di jarak kedalaman yang sama, maka *killer moves* tersebut akan ditelusuri pertama kali (*move ordering*) sebelum menelusuri langkah-langkah lainnya dengan syarat *killer moves* tersebut masih merupakan langkah yang *legal* pada pencarian saat itu.

Namun menurut Schaeffer (1989), efektifitas dari optimasi Killer Heuristic sebagai optimasi dari Alpha-Beta Pruning masih dipertanyakan. Schaeffer mengatakan, bahwa Gillogy telah melakukan observasi dan menyimpulkan bahwa tidak ada keuntungan yang dihasilkan, sedangkan Hyatt mengatakan terdapat pengurangan yang cukup besar hingga 80% pada ukuran *tree* yang dibentuk oleh Killer Heuristic. Sehingga, masih belum diketahui dengan pasti seberapa besar efektifitas yang dihasilkan oleh metode optimasi menggunakan Killer Heuristic ini.