# BAB 2
# LANDASAN TEORI

## 2.1 *Game*

*Game* adalah permainan yang menggunakan interaksi dengan antarmuka pengguna melalui gambar yang dihasilkan oleh piranti video. Permainan video umumnya menyediakan sistem penghargaan, misalnya skor yang dihitung berdasarkan tingkat keberhasilan yang dicapai dalam menyelesaikantugas-tugas yang ada di dalam permainan. Sistem elektronik yang digunakan untuk menjalankan *game* disebut *platform*, seperti komputer pribadi (*Personal computer*) ataupun konsol permainan (*game console*) [7]. *Game* bertujuan untuk menghibur, biasanya game banyak disukai oleh anak-anak hingga orang dewasa. *Games* sebenarnya penting dalam perkembangan otak, meningkatkan konsentrasi dan melatih untuk memecahkan masalah dengan tepat dan cepat. Karena dalam *game* terdapat berbagai konflik atau masalah yang menuntut kita untuk menyelesaikannya dengan cepat dan tepat [8]. Sebuah *Game* bisa dikarakteristik dari apa yang dilakukan. Hal tersebut [9] adalah sebagai berikut:

a. Peralatan, seperti bola, kartu, atau sebuah komputer.
b. Peraturan bisa menentukan giliran permainan, hak dan kewajiban pemain dan tujuan permainan.
*c.* *Skill* strategi dan keberuntungan. *Game* dengan *skill* misalnya kekuatan fisik dan kekuatan pikiran, dan mental.

### 2.1.1 *Genre Game*

*Game* memiliki banyak tipe atau *genre.* Masing-masing genre memiliki karakteristik berbeda. *Genre game* [7] diantaranya yaitu:

a. *Action Game*

   *Game* ini dikenali dari aksi didalamnya yang berupa pergerakan, *attack*, reaksi dan pergerakan-pergerakan lainnya.

b. *Adventure Game*

   *Game* tipe ini menuntut *player* untuk menyelesaikan misi yang diberikan sepanjang petualangan di dalam *game.*

c. *Education Game*

   *Game* edukasi memberikan penekanan pada unsur pendidikan dan pembelajaran.

d. *Puzzle Game*

   *Puzzle game* memiliki tujuan yang sangat sederhana untuk menyelesaikan *puzzle* yang disediakan.

e. *Role-Playing Game*

   Pada *game* bertipe ini memiliki dunia yang luas untuk dieksplorasi oleh player, baik untuk mencari harta karun, objek dan status (*leveling*), sambil menghancurkan *monster* dan objek yang menghalanginya.

f. *Simulation Game*

   *Simulation Game* atau *sims* adalah suatu bentuk permainan yang menggambarkan situasi dunia nyata, baik dalam rupa simulasi driving maupun dalam hubungannya dengan lingkungan interaksi antara manusia.

g. *Strategi Game*

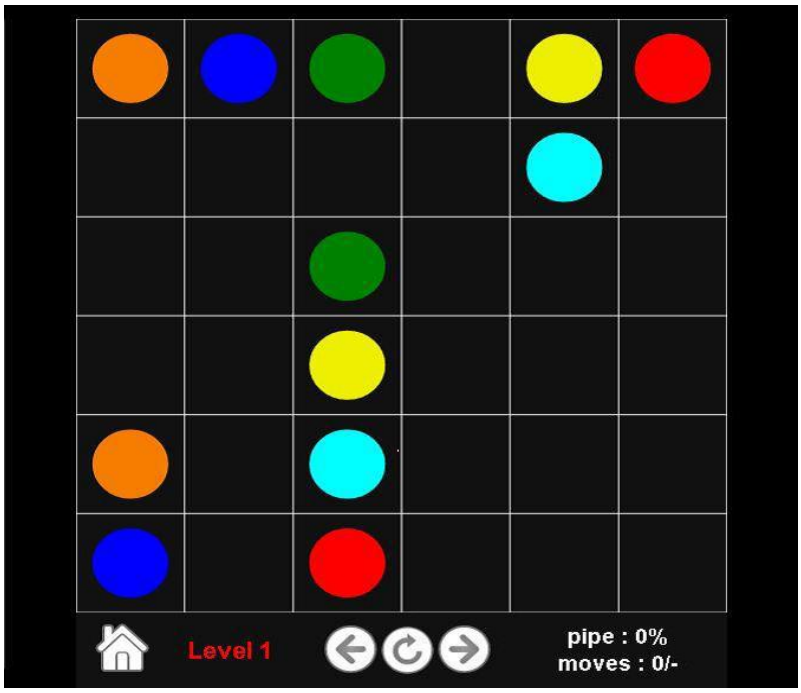
   Dalam *game* tipe ini dibutuhkan pemikiran dan perencanaan yang benar untuk dapat memenangkan permainan dan biasanya menggunakan sudut pandang *overhead* sehingga *player* dapat melihat seluruh area permainan.

### 2.1.2 ***Puzzle Game***

Puzzle merupakan salah satu jenis permainan yang cukup memeras otak untuk menyelesaikannya. Pemain ditantang untuk berfikir kreatif bagaimana untuk membuat semua bagian puzzle terletak pada posisi sebenarnya. Cara memainkannya cukup mudah, pemain hanya menggeser puzzle satu demi satu sampai akhirnya semua puzzle terletak pada posisi sebenarnya [1]**.** Namun kini permainan puzzle sudah banyak mengalami perkembangan, ada yang berupa membentuk urutan huruf, gambar, dan juga yang baru saat ini adalah puzzle yang membuat lintasan dan menghubungkan sesuai dengan jenis warnanya.

## 2.2 Flow Free Color

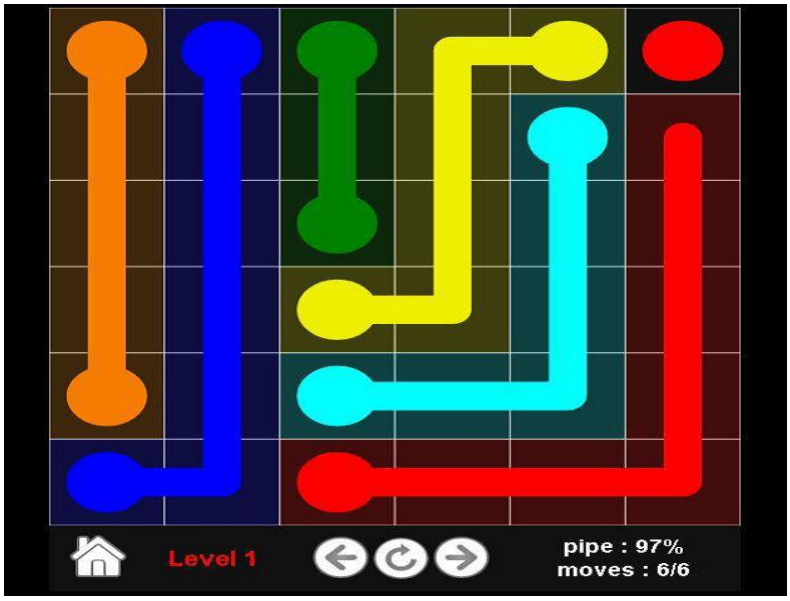
Flow Free Color adalah sebuah permainan berjenis puzzle, game ini terdiri dari sebuah matriks n x n berisi beberapa pasangan lingkaran yang berwarna yang tersebar di beberapa grid [2]**.** *Game* ini tersedia dalam berbagai macam *platform*, dari *web* hingga *mobile Game* ini terdiri dari sebuah matriks n x n berisi beberapa pasangan lingkaran berwarna yang tersebar di dalam *grid* seperti terdapat pada Gambar 2. 1.




**Gambar 2. 1** Contoh Tampilan *Game Flow Free Color*

Tujuan dari game ini adalah membangun lintasan dengan menelusuri *grid* dari setiap *grid* berisi lingkaran berwarna ke *grid* pasangannya sedemikian sehingga tidak ada lintasan yang berpotongan. Untuk membentuk lintasan, pemain harus melakukan klik atau *touch* pada salah satu *colored cell* lalu dengan melakukan *drag*, pemain menelusuri *grid* ke kiri, ke kanan, ke atas atau ke bawah sedemikian sehingga lintasan berakhir di *colored cell* pasangannya. Lintasan dianggap tidak valid jika tidak berakhir di *colored cell* pasangannya atau jika lintasan memotong lintasan lain yang telah dibuat sebelumnya. Pemain juga dapat menghapus lintasan yang telah dibuat tetapi dengan konsekuensi penambahan jumlah *move* yang

dicatat, sehingga menyebabkan pengurangan nilai.Tingkat kesulitan dari *game* ini berdasarkan kepada besar matriks yang digunakan serta sebaran pasangan dari *colored cell*. Untuk memenangkan permainan, pemain harus meng-hubungkan setiap pasangan lingkaran berwarna dengan lintasan seperti terdapat pada Gambar 2. 2.




**Gambar 2. 2** Contoh Solusi *Game Flow Free Color*

## 2.3 Kecerdasan Buatan

Kecerdasan buatan berasal dari bahasa Inggris "*Artificial Intelligence*" atau disingkat AI, yaitu *intelligence* adalah kata sifat yang berarti cerdas, sedangkan *artificial* artinya buatan. Kecerdasan yang dimaksud disini merajuk pada mesin yang mampu berpikir seperti manusia, berfikir secara rasional, dan mampu mengambil keputusan seperti yang dilakukan oleh manusia. Kecerdasan buatan telah banyak digunakan dan diaplikasikan kedalam kehidupan sehari-hari.

Kecerdasan buatan memiliki beberapa kelebihan dibandingkan dengan kecerdasan buatan yang dimiliki oleh manusia [3], yaitu :

a. Bersifat lebih permanen: Selama sistem dan program tidak berubah, maka kecerdasan buatan tersebut tidak akan berubah.

b. Lebih mudah untuk diduplikasikan dan disebarkan: Akan lebih mudah memindahkan data dari satu komputer ke komputer lain dibandingkan dengan memindahkan pengetahuan dari satu manusia ke manusia yang lain.
c. Lebih murah dibandingkan dengan mendatangkan seorang ahli.
d. Konsisten: Kecerdasan buatan merupakan sebuah teknologi komputer sedangkan kecerdasan buatan memiliki kecendrungan untuk berubah.
e. Bisa didokumentasi: Tiap aktifitas yang dilakukan oleh kecerdasan buatan dapat dilacak dengan mudah sedangkan kecerdasan alami termasuk sulit untuk direproduksi.
f. Mengerjakan pekerjaan dengan waktu lebih cepat dan lebih baik.

Kecerdasan buatan menyediakan dasar-dasar ilmu pengetahuan pada beberapa bidang teknologi yang dapat digunakan secara komersial [3], diantaranya:

a. Sistem pakar (*expert system*)

   Sebuah sistem komputer yang digunakan sebagai sarana untuk menyimpan pengetahuan yang dimiliki oleh seorang atau lebih pakar dengan tujuan agar komputer memilki keahlian untuk menyelesaikan permasalahan dengan meniru keahlian yang dimilki pakar tersebut.
b. Pengolahan bahasa alami (*natural language processing*)

   Pemrograman sistem komputer dimana pengguna dapat berkomunikasi dengan komputer menggunakan bahasa sehari-hari
c. Pengenalan suara/ucapan (*speech recognition*)

   Pengguna dapat berkomunikasi dengan komputer untuk memberi perintah kepada untuk melakukan sebuah pekerjaan.
d. Robotika dan sistem sensor (*robotic and sensory system*)

   Sebuah lengan robotik yang dilengkapi oleh berbagai sensor yang diprogram untuk mampu mendeteksi jenis pekerjaan yang perlu dilakukan oleh lengan tersebut.
e. *Computer vision*

   Pemrograman sistem yang bertujuan untuk menginterpretasikan gambar dan objek tampak melalui komputer untuk proses lebih jauh.

f. *Game playing*

*Game playing* merupakan implementasi *artificial intelegence* yang berguna untuk meniru cara berfikir seorang manusia dalam bermain *game.* Contohnya adalah program *Perfect Chessmate* yang mampu berfikir setara dengan seorang *grandmaster* catur.

Adapun teknik pemecahan masalah yang ada didalam kecerdasan buatan terbagi menjadi empat [4] yaitu :

a. *Searching*

*Searching* merupakan teknik menyelesaikan masalah dengan cara merepresentasikan masalah ke dalam *state* dan ruang masalah serta menggunakan strategi pencarian untuk menemukan solusi.

b. *Reasoning*

*Reasoning* merupakan teknik menyelesaikan masalah dengan cara merepresentasikan masalah ke dalam basis pengetahuan (*knowladge base)* menggunakan *logic* atau bahasa formal.

c. *Planing*

*Planing* merupakan teknik menyelesaikan masalah dengan cara memecah masalah ke dalam sub-sub masalah dan menyelesaikan sub-sub masalah satu demi satu.

d. *Learning*

*Learning* merupakan teknik menyelesaikan masalah dengan mengotomatisasi mesin untuk menemukan aturan yang diharapkan bisa berlaku umum untuk data-data yang belum pernah diketahui.

## 2.4 Teknik-Teknik Dasar Pencarian

Pencarian atau pelacakan merupakan salah satu teknik untuk menyelesaikan permasalahan AI. Keberhasilan suatu sistem salah satunya ditentukan oleh kesuksesan dalam pencarian dan pencocokan. Teknik dasar pencarian memberikan suatu kunci bagi banyak sejarah penyelesaian yang penting dalam bidang AI. Ada beberapa aplikasi yang menggunakan teknik pencarian ini [3], yaitu:

a. Papan *game* dan *puzzle* (tic-tac-toe, catur, menara hanoi).
b. Penjadwalan dan masalah *routing (travelling salesman problem).*
c. *Parsing* bahasa dan inteprestasinya (pencarian struktur dan arti).
d. Logika pemrograman (pencarian fakta dan implikasinya).
e. *Computer vision* dan pengenalan pola.
f. Sistem pakar bebasis kaidah *(rule based expert system).*

Pencarian adalah proses mencari solusi dari suatu permasalahan melalui sekumpulan kemungkinan ruang keadaan *(state space)*. Ruang keadaan merupakan suatu ruang yang berisi semua keadaan yang mungkin.

Kondisi suatu pencarian meliputi:

a. Keadaan sekarang atau awal.
b. Keadaan tujuan-solusi yang dijangkau dan perlu diperiksa apakah telah mencapai sasaran.
c. Biaya atau nilai yang diperoleh dari solusi.

Solusi merupakan suatu lintasan dari keadaan awal sampai keadaan tujuan. Secara umum, proses pencarian dapat dilakukan seperti berikut :

a. Memeriksa keadaan sekarang atau awal.
b. Mengeksekusi aksi yang dibolehkan untuk memindahkan ke keadaan berikutnya.
c. Memeriksa jika keadaan baru merupakan solusinya. Jika tidak, keadaan baru tersebut menjadi keadaan sekarang dan proses ini diulangi sampai solusi ditemukan atau ruang keadaan habis terpakai [3].

## 2.5 Metode-metode Pencarian

Metode pencarian secara umum dibagi menjadi 2 kelompok, yaitu *blind* atau *uninformed search* (pencarian buta atau tidak dibekali informasi) dan *heuristic* atau *informed search* (pencarian dengan panduan atau berbekal informasi). Setiap metode memiliki karakteristik yang saling membedakan satu sama lain dengan kelebihan dan kekurangan masing-masing, yang bisa diukur performansinya dengan empat kriteria [4] berikut:

a. *Completeness,* yaitu apakah metode pencarian tersebut menjamin ditemukannya solusi jika solusinya memang ada.
b. *Time complexity,* yaitu berapa lama waktu yang diperlukan selama proses pencarian.
c. *Space complexity,* yaitu berapa banyak memori yang diperlukan selama proses pencarian.
d. *Optimality,* yaitu apakah metode pencarian tersebut menjamin ditemukannya solusi yang terbaik jika ada beberapa solusi yang berbeda.

Metode pencarian sangat penting untuk menyelesaikan permasalahan karena setiap *state* atau keadaan menggambarkan langkah-langkah untuk menyelesaikan permasalahan. Dalam sebuah *game*, metode pencarian akan menentukan apa yang harus dilakuan dimana setiap *state* menggambarkan kemungkinan posisi pada saat tertentu.

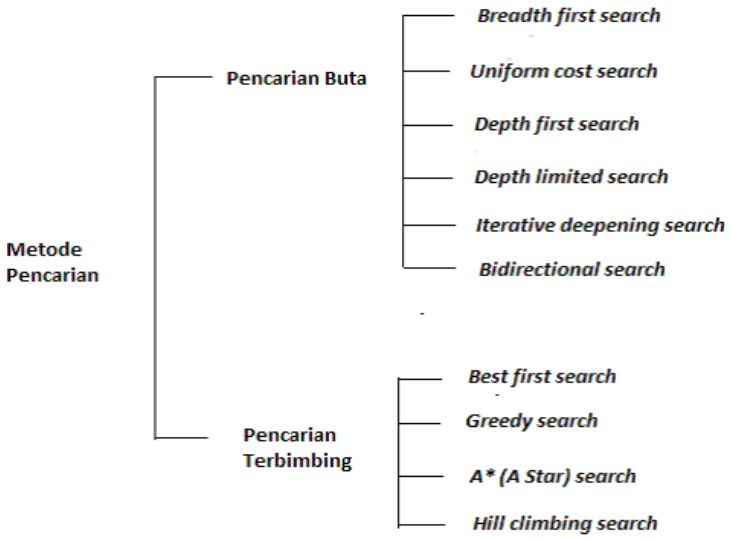
### 2.5.1 Algoritma Pencarian (*Search Algorithm*)

Permasalahan pencarian dapat diselesaikan dengan beberapa metode yaitu:

a. Metode pencarian yang pertama adalah metode yang sederhana yang hanya berusaha mencari kemungkinan penyelesaian yang disebut juga pencarian buta.
b. Metode yang lebih kompleks yang akan mencari jarak terpendek. Metode ini adalah *British Museum Procedure, Branch and Bound*, *Dynamic Programming, Best First Search,Greedy Search,* A* *(A Star) Search,* dan *Hill Climbing Search.* Metode-metode ini digunakan pada saat harga perjalanan untuk mencari kemungkinan menjadi perhitungan. Beberapa *procedure*/metode yang kita terapkan saat berhadapan dengan musuh. Prosedur ini adalah *minimax search* dan

*alpha-beta pruning.* Metode ini banyak digunakan pada program-program seperti catur dan sebagainya.

Metode pencarian dikatakan penting untuk meyelesaikan permasalahan karena setiap *state* (keadaan) menggambarkan langkah-langkah untuk menyelesaikan permasalahan. Metode pencarian dikatakan penting untuk perencanaan karena dalam sebuah permainan akan menentukan apa yang harus dilakukan, dimana setiap *state* menggambarkan kemungkinan posisi pada suatu saat. Metode pencarian adalah bagian dari kesimpulan, dimana setiap *state* menggambarkan hipotesis dalam sebuah rangkaian deduktif [4].




**Gambar 2. 3** Bagan metode pencarian (*searching*)

Menurut cara algoritma mengembangkan *node* dalam proses pencarian, gambar bagan metode penulusuran dibagi menjadi dua golongan, yakni pencarian buta *(blind search)* dan pencarian terbimbing *(heuristic search)* [4].

### 2.5.2 **Pencarian Buta (*Blind Search*)**

*Blind Search* adalah pencarian solusi tanpa adanya informasi yang dapat mengarahkan pencarian untuk mencapai *goal state* dari *current state* (keadaan sekarang). Informasi yang ada hanyalah definisi *goal state* itu sendiri, sehingga algoritma dapat mengenali *goal state* bila menjumpainya.

Metode yang tergolong *blind search* diantaranya : *Breadth First Search* (BFS), *Uniform Cost-Search* (UCS), *Depth First Search* (DFS), *Depth Limited Search* (DLS), *Iterative Deepening Search* (IDS), dan *Bidirectional Search* (BDS). Metode *blind search* biasanya digunakan pada pencarian yang memiliki ruang masalah yang kecil. Jika ruang masalahnya sudah besar sebaiknya digunakan metode *informed search* agar pada saat melakukan pencarian dapat lebih terarah dengan informasi-informasi yang diberikan [4].

### 2.5.3 **Pencarian Terbimbing (*Heuristic Search*)**

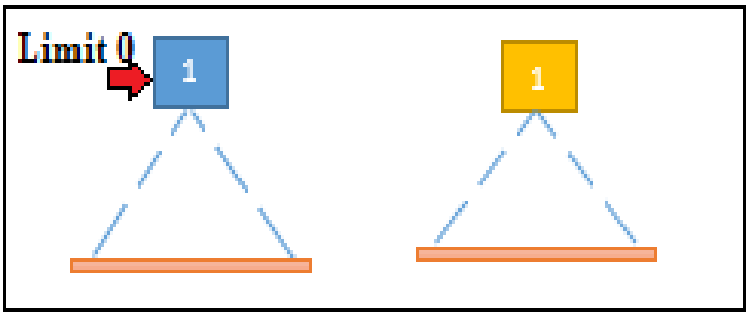
Metode heuristik merupakan metode pencarian yang mempunya informasi tentang biaya untuk mencapai *goal state* dari *current state.* Informasi ini adalah suatu fungsi yang disebut dengan fungsi heuristik. Metode heuristik menggunakan pendekatan nilai perkiraan untuk membantu algoritma dalam melakukan pencarian dan tidak harus *valid* setiap waktu. Metode heuristik dewasa ini banyak berkembang. Telah banyak algoritma yang menerapkan metode heuristik dalam melakukan proses pencarian, metode yang umum dikenal adalah A*. Beberapa contoh algoritma pencarian lain yang menggunakan metode *heuristic search* adalah *Best First Search, Generate and Test, Hill Climbing,* dan *Greedy Search* [4].

### 2.5.4 **Algoritma *Iterative Deepening* A* (IDA*)**

Algoritma Iterative Deepening A* merupakan gabungan dari algoritma IDS dan A*. Algoritma Iterative Deepening Search adalah suatu teknik yang menggabungkan keuntungan pencarian dengan menggunakan teknik Breadth First Search (BFS) dalam hal complete dan optimal serta keuntungan dari teknik pencarian Depth First Search (DFS) dalam hal Space Complexity [3]**.** Sedangkan Algoritma A*(A Star) adalah algoritma pencarian graf/pohon yang mencari jalur dari satu titik awal ke sebuah titik akhir yang telah ditentukan [4]**.** Kelebihan algoritma A* yaitu biaya yang diperhitungkan didapat dari biaya sebenarnya ditambah dengan biaya perkiraan. Sedangkan kelebihan algoritma IDS menggabungkan kelebihan *Breadth First Search* (*complete* dan *optimal*) dan kelebihan *Depth First Search* (membutuhkan sedikit memori) [10]. IDS merupakan

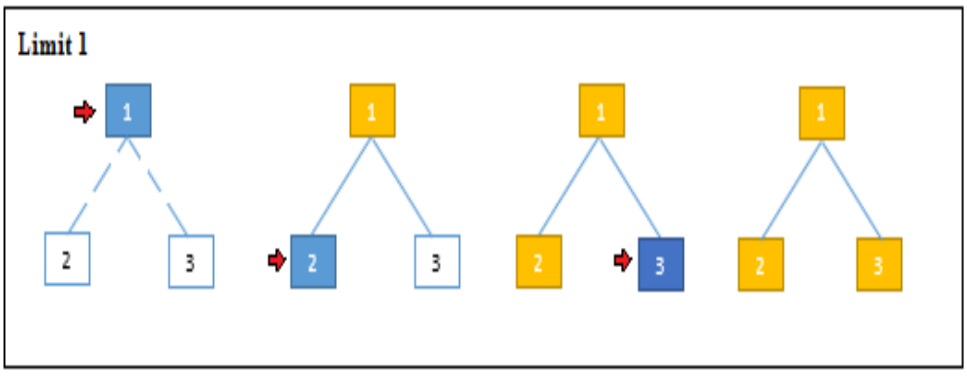
algoritma *Blind Search* yang melakukan pencarian secara iteratif menggunakan penelusuran *Depth Limited Search* (DLS) dimulai dengan batasan limit 0 [5]. Jika belum ditemukan solusi, maka dilakukan iterasi ke-2 dengan batasan limit 1 dan seterusnya sampai ditemukan solusi yang tepat. Berikut adalah ilustrasi penelusuran Algoritma IDS :

a. Dimisalkan simpul awal adalah 1 dan simpul tujuan adalah 7, maka penelusuran awal untuk pencarian IDS dimulai dari simpul 1 pada limit 0. persegi dengan tanda panah menunjukkan simpul yang sedang diperiksa. Sedangkan persegi berwarna kuning menunjukkan bahwa pencarian simpul A sudah ditelusuri. Dapat dilihat pada Gambar 2. 4.



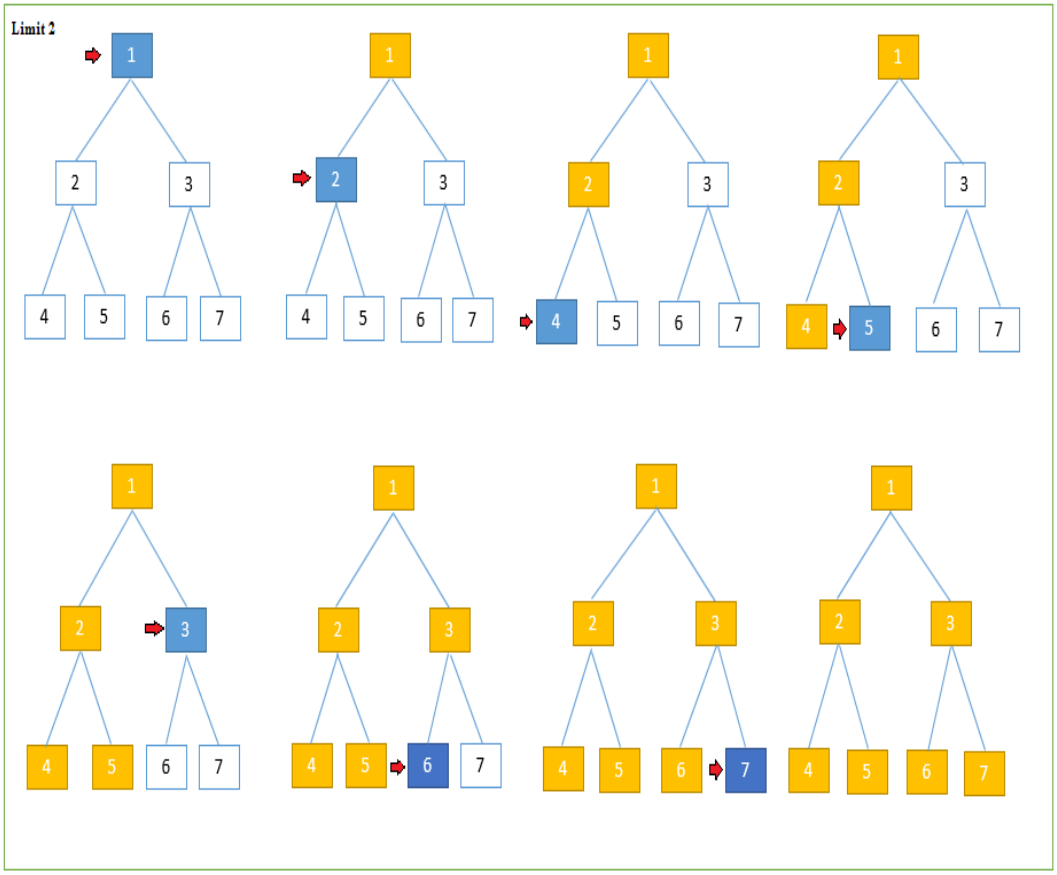

**Gambar 2. 4** Penelusuran Iterative Deepening Search (IDS) Limit 0

b. Karena simpul 1 bukan simpul tujuan, maka ulangi pencarian dari status awal. Sekarang pencarian dimulai dari limit ke-1 dengan mengekspansi suksesor dari simpul 1 kemudian lakukan pencarian dari simpul kiri ke simpul anak kanan. Dapat dilihat pada Gambar 2. 5.




**Gambar 2. 5** Penelusuran *Iterative Deepening Search* (IDS) Limit 1

c. Jika pada limit 1 belum ditemukan simpul tujuan, maka pencarian diulang kembali dari status awal secara iteratif dan sekarang dilanjutkan dari limit ke-2 dengan simpul 1 diekspansi, simpul 2 juga diekspansi dan simpul 3 juga diekspansi, kemudian lakukan penelusuran ke suksesor 4 dan 5, setelah sampai akhir simpul 3 diekspansi, dan simpul tujuan ditemukan yaitu simpul 7. Maka pencarian dihentikan setelah simpul tujuan ditemukan. Dapat dilihat pada Gambar 2. 6 [4].




**Gambar 2. 6** Penelusuran *Iterative Deepening Search* (IDS) Limit 2

Berdasarkan tahap-tahap pencarian IDS, batasan limit (kedalaman) pada setiap iterasi pencarian IDS di atas yang diterapkan pada algoritma IDA* dengan batasan yang berupa *f-limit*. Jika jalur solusi belum ditemukan, setiap iterasi mengembalikan nilai *f-limit* baru yang akan digunakan sebagai batasan pencarian untuk iterasi berikutnya. Nilai *f-limit* itu sendiri diperoleh dari gabungan biaya sebenarnya dan biaya perkiraan seperti terdapat pada persamaan (2.1) , yang secara matematika dinyatakan sebagai :

$f(n) = g(n) + h(n)$ ...(2. 1)

Dengan :

$n$ = simpul

$g(n)$ = Jarak asli dari lokasi awal ke lokasi ke n

$h(n)$ = Jarak dari lokasi awal ke node lokasi n

$f(n)$ = Nilai evaluasi

Sedangkan untuk melakukan pencarian *h(n)* seperti terdapat pada persamaan (2.2) dilakukan dengan cara sebagai berikut :

$h(n) = \sqrt{(\boldsymbol{x - x1})^2 + (\boldsymbol{y - y1})^2}$ ...(2. 2)

Dengan :

$h(n)$ = perkiraan biaya minimum dari simpul *n* ke simpul tujuan

X = posisi X simpul awal

Y = posisi Y simpul awal

X1 = posisi X dari simpul lokasi ke simpul tujuan

Y 1= posisi Y dari simpul lokasi ke simpul tujuan

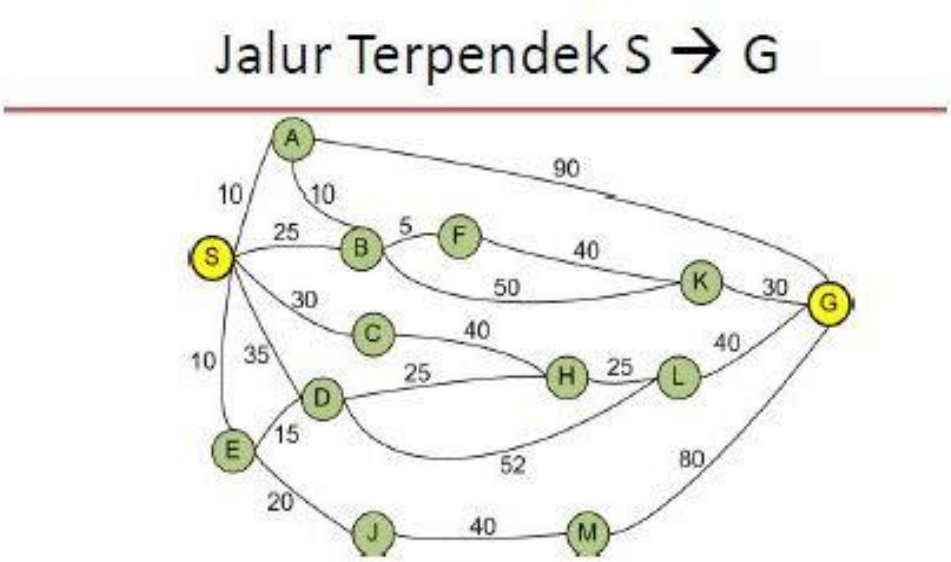
Algoritma IDA* adalah algoritma pencarian yang menggunakan fungsi heuristik, yaitu fungsi yang menghitung biaya perkiraan dari simpul tertentu menuju simpul tujuan. IDA* bersifat *complete* (pasti menemukan jalur dari simpul asal ke simpul tujuan) dan *optimal* (jalur yang ditemukan adalah jalur terbaik dengan biaya minimun) asalkan fungsi heuristiknya *admissible* ( tidak melakukan overtimasi terhadap nilai minimal) [4].

Langkah-langkah pencarian IDA* adalah sebagai berikut :

1. Tentukan nilai simpul awal dan simpul akhir
2. Hitung nilai f(n)
3. Jadikan nilai f(n) sebagai *f-limit*
4. Telusuri node dari status awal
5. Cek nilai f(n) apakah lebih kecil dari limit?
   - Jika Iya, bangkitkan semua anak dari node dengan f(n) terkecil tersebut, jika tidak, jadikan f(n) tersebut sebagai limit baru.
6. Kemudian ulangi pencarian dari status awal (restart).

7. Pencarian akan berakhir apabila pencarian mencapai simpul tujuan.

Adapun contoh masalah penyelesaian rute terpendek menggunakan alogoritma IDA* adalah sebagai berikut :




| Jarak Garis Lurus Kota n ke Kota G | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| n | S | A | B | C | D | E | F | G | H | J | K | L | M |
| h(n) | 80 | 80 | 60 | 70 | 85 | 74 | 70 | 0 | 40 | 100 | 30 | 20 | 70 |

**Gambar 2. 7** Contoh kasus rute terpendek

Langkah-langkah penyelesaian rute terpendek menggunakan algoritma IDA* adalah sebagai berikut :

1. Tentukan simpul awal $(n_a)$ dan simpul tujuan $(n_t)$

   Simpul awal $(n_a)$ = S

   Simpul tujuan $(n_t)$ = G
2. Hitung $f(n_a)$ simpul awal

   $f(s)$ = *g(s) + c(s,s) + h(s)*

   *= 0 + 0 + 80*

   *= 80*
3. Jadikan nilai $f(n_a)$ sebagai *f-limit*

   *f-limit* = 80
4. Inisialisasi next –*f* = ∞

5. Kemudian lakukan perulangan untuk setiap
   suksesor $n_i$ dari simpul $n$ (kiri ke kanan). Suksesor adalah simpul anak yang akan dikunjungi berikutnya. Simpul $S$ mempunyai 3 simpul suksesor, maka lakukan perulangan sebanyak 3 kali.
   suksesor 1 dari simpul $S$
   a. $n_i = A$

   b. Hitung $f(n_i)$ dari simpul suksesor.
      $f(A) = g(S) + c(S,A) + h(A)$
      $= 0 + 10 + 74 = 84$
   c. Karena $f(A) > f\text{-}limit$ dan $f(A) < next\text{-}f$, maka $f(A)$ dijadikan sebagai $next\text{-}f$.
      $f(A) < \infty$, maka $next\text{-}f = f(A) = 84$

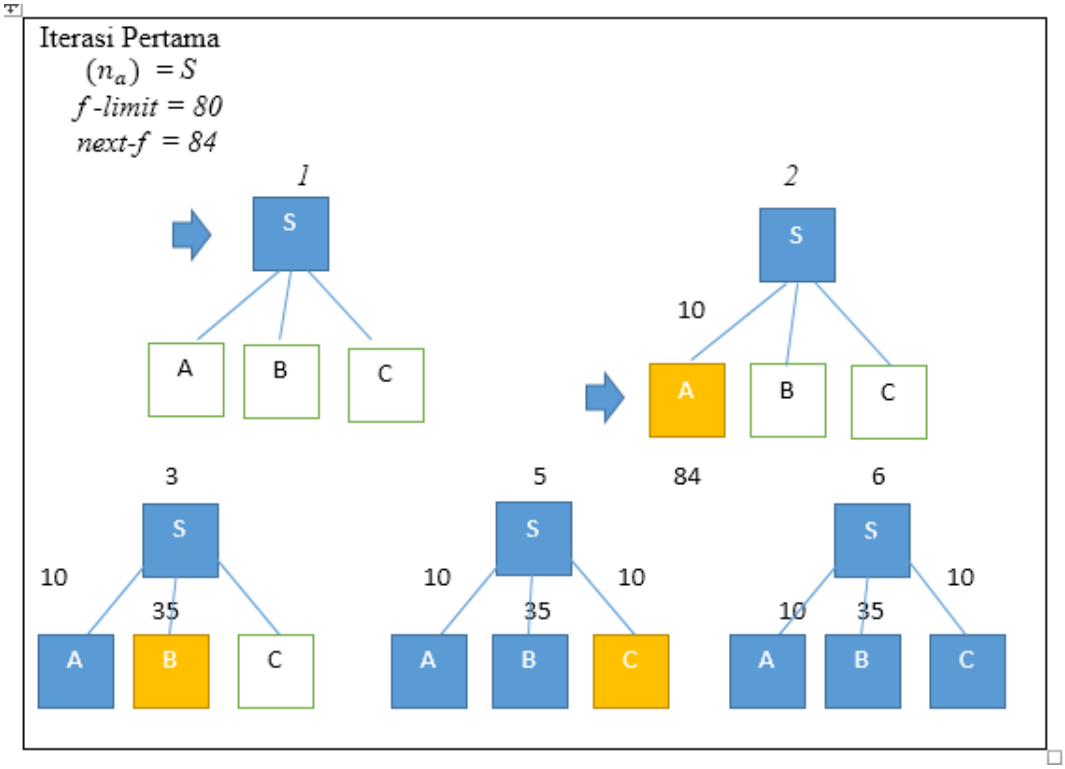
suksesor 2 dari simpul $S$
   a. $n_i = B$

   b. Hitung $f(n_i)$ dari simpul suksesor.

      $f(B) = g(S) + c(S,B) + h(B)$
      $= 0 + 35 + 85 = 120$
   c. Karena $f(B) > f\text{-}limit$, tetapi $f(B) > next\text{-}f$, maka nilai $next\text{-}f$ tetap.
      $next\text{-}f = 84$

suksesor 3 dari simpul $S$
   a. $n_i = C$

   b. Hitung $f(n_i)$ dari simpul suksesor.
      $f(C) = g(S) + c(S,C) + h(C)$
      $= 0 + 10 + 80 = 90$
   c. Karena $f(C) > f\text{-}limit$, tetapi $f(C) > next\text{-}f$, maka nilai $next\text{-}f$ tetap.
      $next\text{-}f = 84$




**Gambar 2. 8** Langkah ke 5 pencarian IDA*

6. Setelah semua suksesor diekspansi, set *f-limit* = *next-f* dan *next-f* = ∞.
   *f-limit* = 84 ; *next-f* = ∞.
7. Karena simpul tujuan belum ditemukan maka ulangi pencarian dari status awal. Lakukan perulangan seperti langkah 5 untuk mencari suksesor $n_i$ dari simpul *S* dengan *f-limit* sekarang yaitu 84. Simpul *S* mempunyai 3 simpul suksesor, maka lakukan perulangan sebanyak 3 kali.

   suksesor 1 dari simpul S

   a. $n_i = A$, $f(A) = 84$

   b. Karena $f(A) \leq$ *f-limit*, lakukan perulangan untuk setiap suksesor $n_i$ dari simpul *A*. Simpul *A* mempunyai 2 simpul suksesor, maka lakukan perulangan sebanyak 2 kali. Lihat Gambar 2.9.

   suksesor 1 dari simpul *A*

   a. $n_i = J$

   b. Hitung $f(n_i)$ dari simpul suksesor.

   $f(J) = g(A) + c(A,J) + h(J)$

   $= 10 + 20 + 100 = 130$

c. Karena $f(J) > f\text{-}limit$ dan $f(J) < next\text{-}f$, maka $f(J)$ dijadikan sebagai $next\text{-}f$.

$f(J) < \infty$, maka $next\text{-}f = f(J) = 130$

suksesor 2 dari simpul $A$

a. $n_i = B$ (melalui simpul $A$)

b. Hitung $f(n_i)$ dari simpul suksesor.

$f(B) = g(A) + c(A,B) + h(B)$

$= 10 + 15 + 85 = 110$

c. Karena $f(B) > f\text{-}limit$ dan $f(n_i) < next\text{-}f$, maka $f(B)$ dijadikan sebagai $next\text{-}f$.

$f(B) < 130$, maka $next\text{-}f = f(B) = 110$

Hasil ekspansi simpul $A$ belum menemukan simpul tujuan, oleh karena itu pencarian dilanjutkan arah ke simpul kanan yaitu simpul $B$ dan $C$.

suksesor 2 dari simpul $S$

a. $n_i = B$

b. Hitung $f(n_i)$ dari simpul suksesor.

$f(B) = g(S) + c(S,B) + h(B)$

$= 0 + 35 + 85 = 120$

c. Karena $f(B) > f\text{-}limit$, tetapi $f(B) > next\text{-}f$, maka nilai $next\text{-}f$ tetap.

$next\text{-}f = 110$

suksesor 3 dari simpul $S$

a. $n_i = C$

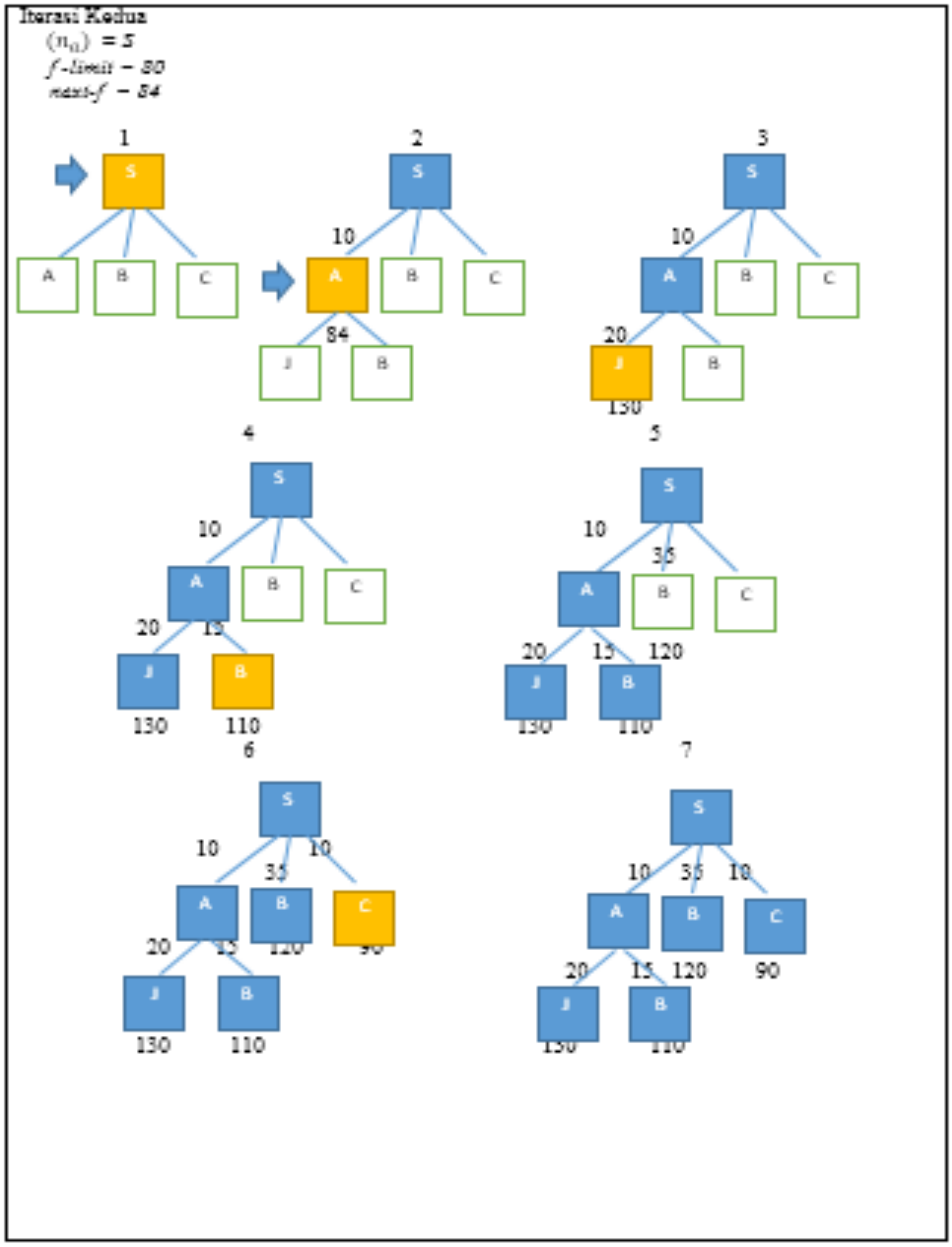
b. Hitung $f(n_i)$ dari simpul suksesor.

$f(C) = g(S) + c(S,C) + h(C)$

$= 0 + 10 + 80 = 90$

c. Karena $f(C) > f\text{-}limit$, tetapi $f(C) < next\text{-}f$, maka $f(C)$ dijadikan sebagai $next\text{-}f$.

$f(C) < 110$, maka $next\text{-}f = f(C) = 90$




**Gambar 2. 9** Langkah ke 7 pencarian IDA*

8. Setelah semua suksesor diekspansi, set *f-limit* = *next-f* dan *next-f* = ∞.
   *f-limit* = 90 ; *next-f* = ∞.
9. Karena simpul tujuan belum ditemukan, ulangi pencarian dari status awal. Lakukan perulangan seperti langkah 7 untuk mencari suksesor $n_i$ dari simpul *S* dengan *f-limit* sekarang yaitu 90. Pada suksesor ke-3 dari simpul *S* yaitu simpul *C* diperoleh nilai $f(C) \leq$ *f-limit* maka lakukan perulangan untuk setiap suksesor $n_i$ dari simpul *C*. Simpul *C* mempunyai 2 simpul suksesor, maka lakukan perulangan sebanyak 2 kali.

suksesor 1 dari simpul *C*

a. $n_i = K$

b. Hitung $f(n_i)$ dari simpul suksesor.

$f(K) = g(C) + c(C,K) + h(K)$

$= 10 + 55 + 30 = 95$

c. Karena $f(K) > f\text{-}limit$ dan $f(K) < next\text{-}f$, maka $f(K)$ dijadikan sebagai $next\text{-}f$.

$f(K) < \infty$, maka $next\text{-}f = f(K) = 95$

suksesor 2 dari simpul *C*

a. $n_i = G$

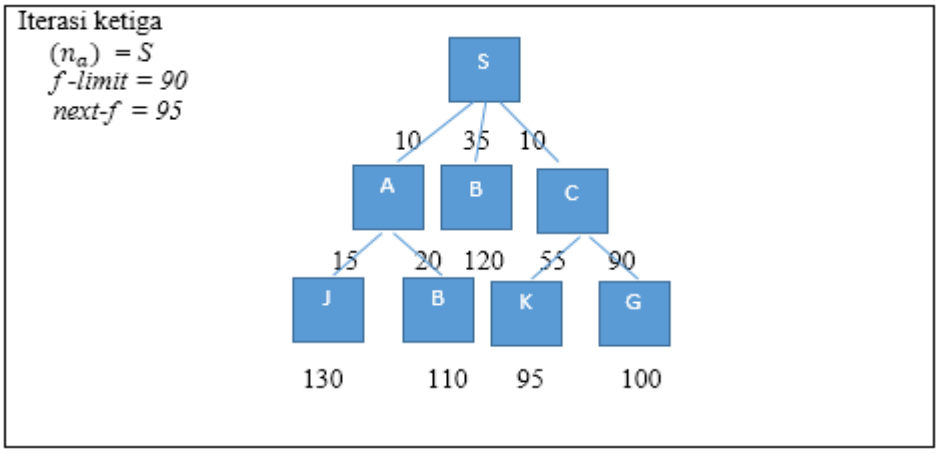
b. Hitung $f(n_i)$ dari simpul suksesor.

$f(G) = g(C) + c(C,G) + h(G)$

$= 10 + 90 + 0 = 100$

c. Simpul *G* merupakan simpul tujuan, tetapi nilai $f(G) > f\text{-}limit$ dan $f(G) > next\text{-}f$, maka nilai $next\text{-}f$ tetap.

$next\text{-}f = 95$




**Gambar 2. 10** Langkah ke 9 pencarian IDA*

10. Setelah perulangan selesai, set *f-limit* = *next-f* dan *next-f* = ∞.

    *f-limit* = 95 ; *next-f* = ∞.

11. Karena simpul tujuan belum ditemukan, ulangi pencarian dari status awal. Lakukan perulangan seperti langkah 9 untuk mencari suksesor $n_i$ dari simpul *S* dengan *f-limit* sekarang yaitu 95. Pada suksesor pertama dari simpul *C* yaitu simpul *K* diperoleh nilai $f(K) \leq$ *f-limit* maka lakukan perulangan untuk setiap suksesor $n_i$ dari simpul *n*. Simpul *K* mempunyai 1 simpul suksesor.

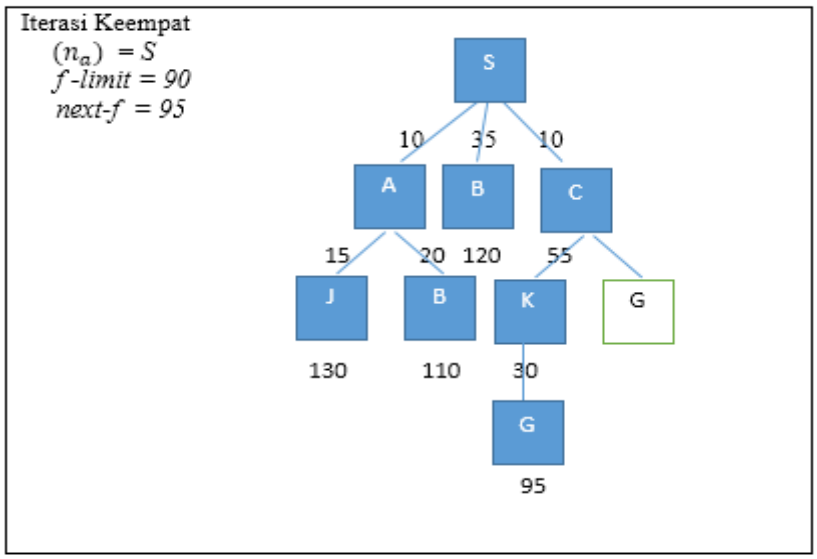
suksesor 1 dari simpul C

a. $n_i = G$ (melalui simpul *K*)

b. Hitung $f(n_i)$ dari simpul suksesor.

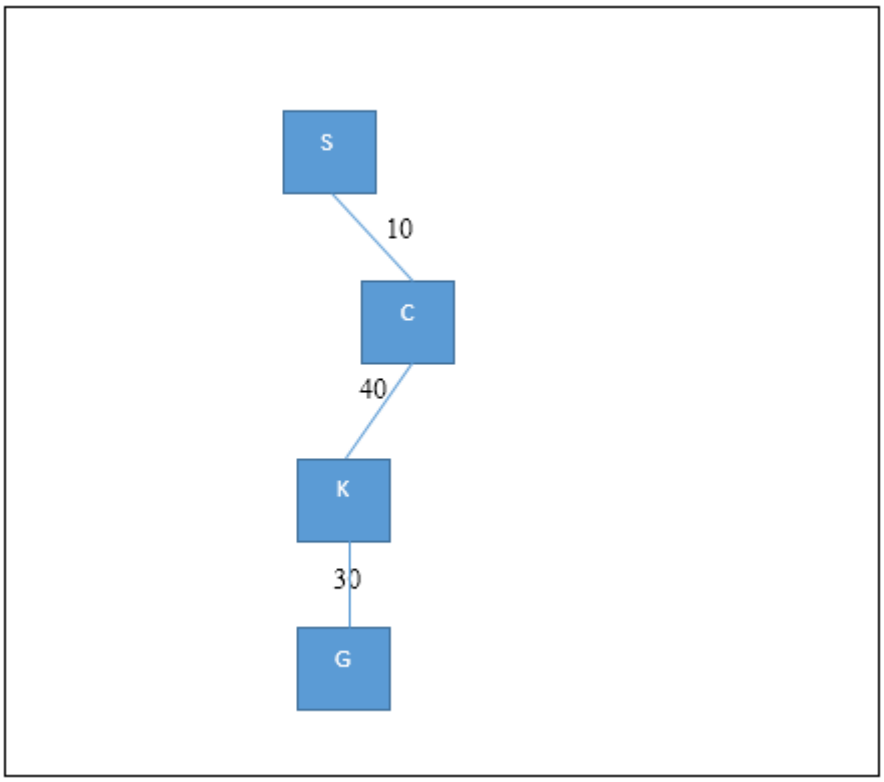
$$f(G) = g(K) + c(K,G) + h(G)$$
$$= 65 + 30 + 0 = 95$$

c. Simpul *G* merupakan simpul tujuan dan $f(G) \leq$ *f-limit*, oleh karena itu pencarian IDA* berhenti dengan menemukan tujuan yaitu simpul *G*.




**Gambar 2. 11** Langkah ke 11 Pencarian IDA*

Setelah simpul tujuan ditemukan, berikut hasil akhir diperoleh jalur terpendek yang ditemukan adalah S-C-K-G dengan biaya keseluruhan yaitu 95

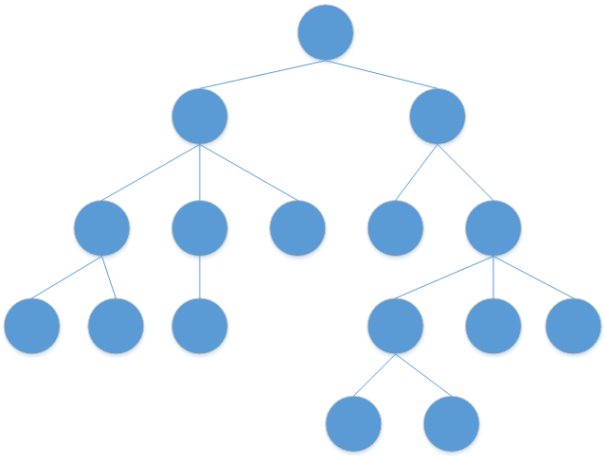



**Gambar 2. 12** Hasil Akhir pencarian rute S ke G

### 2.5.5 **Metode Pruning**

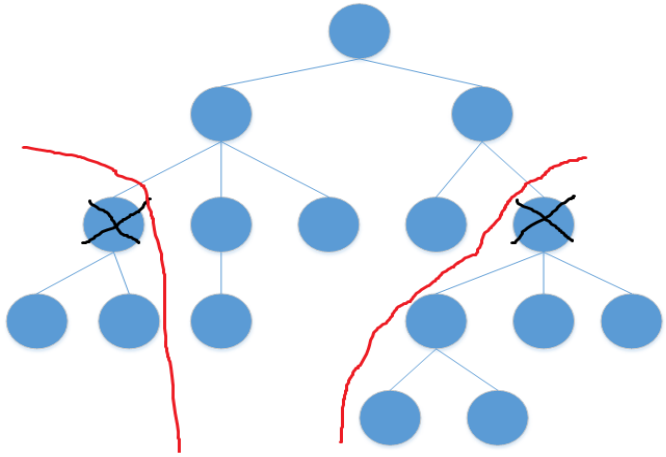
Pruning adalah metode yang mengabaikan state yang tidak mungkin menghasilkan solusi [11]**.**

Misalkan tree di bawah merepresentasikan ruang pencarian yang harus ditelusuri untuk menemukan solusi seperti terdapat pada Gambar 2. 13.



**Gambar 2. 13** Contoh Pohon Ruang Pencarian

Dengan *pruning* dapat dibuktikan secara matematis, dapat memangkas pencarian dengan tidak menelusuri state pencarian yang tidak perlu seperti pada Gambar 2. 14.

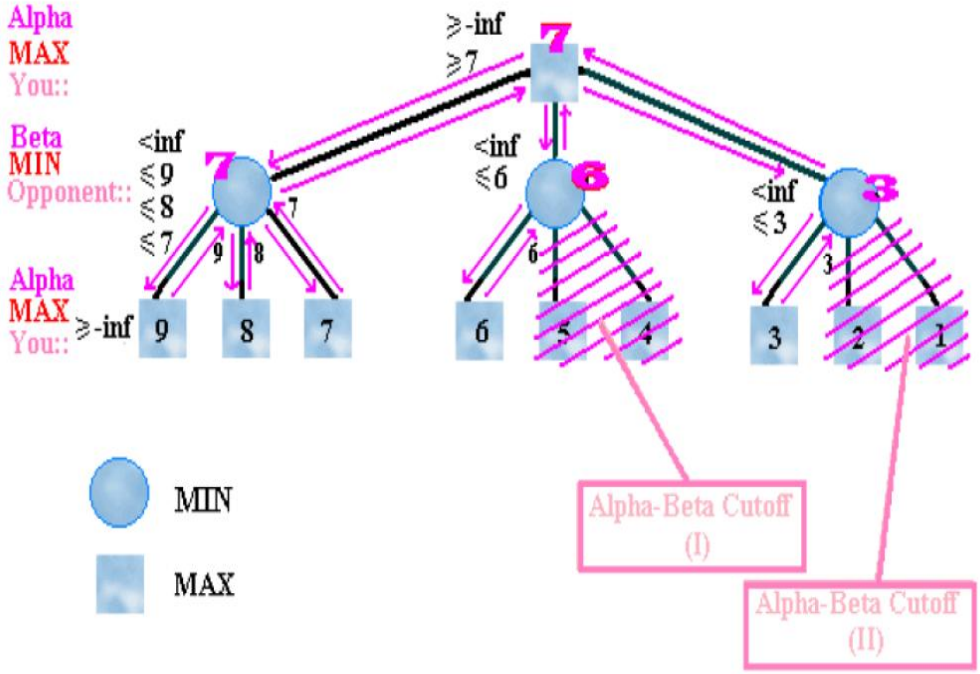


**Gambar 2. 14** Contoh pruning pada pohon ruang pencarian

Karekteristik dari metode pruning ini adalah :

a. Semakin tinggi state atau semakin kecil kedalaman state yang kita pruning maka semakin banyak state pencarian yang kita pangkas
b. Dengan pruning yang tepat, keuntungan dari pemangkasan ini dapat mengurangi waktu pencarian secara eksponensial.

Dengan metode ini, waktu penelusuran dapat dikurangi. Salah satu aplikasi dari metode pruning yang terkenal adalah penggunaan *Alpha-Beta pruning* pada komputer catur [5]. *Alpha-beta pruning* mengeliminasi kemungkinan langkah ke depan dari himpunan hasil analisis jika mereka terbukti tidak lebih baik dari langkah yang telah ditemukan. Pengembang dari komputer catur Deep Blue menggunakan algoritma ini dan menghasilkan inovatif teori yang disebut sebagai "*singular extensions*" [5].




**Gambar 2. 15** Contoh Algoritma *Alpha-Beta Pruning*

Algoritma akan mendata dua nilai : alpha dan beta, yang akan merepresentasikan nilai minimum untuk MAX, dan nilai maksimum untuk MIN. Keadaan awal nilai alpha adalah –infinity, dan beta bernilai awal +infinity. Sedangkan alpha-beta Cutoff adalah node alpha-beta yang tidak perlu dievaluasi. Selama jalannya rekursi, perbedaan nilai alpha dan beta akan semakin mengecil. Ketika nilai beta menjadi lebih kecil dari alpha, berarti keadaan saat ini bukan merupakan hasil terbaik dari kedua pemain, dengan demikian tidak perlu dieksplorasi lebih jauh seperti terdapat pada Gambar 2. 15.

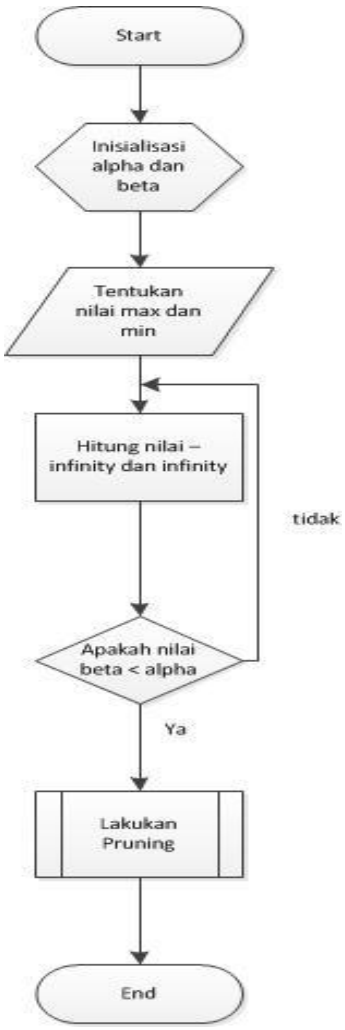
Langkah-langkah metode pruning :

1. Hampiri node pertama dengan nilai 9, naik ke node parent level min masukkan nilai 9, hampiri nilai 8, karena 9<8 maka ganti node parent 8, hampiri 7, karena 8<7 maka nilai 7 naik k level min.
2. Setelah ketiga node pertama dihampiri, naik ke level max masukkan nilai 7.
3. Hampiri node keempat pada dengan nilai simpul 6, naik ke node parent pada level min masukkan nilai 6. Jika menghampiri node berikutnya, maka

mencari nilai yang lebih tinggi dari 6, sementara pada level max mencari yang lebih kecil dari 7, maka node 5 dan 4 di *pruning* (tidak dihampiri).

4. Setelah itu hampiri node ketujuh pada nilai simpul 3, naik ke node parent pada level min masukkan nilai 3. Jika menghampiri node berikutnya, maka akan mencari nilai yang lebih tinggi dari 3, sementara pada level max mencar lebih kecil dari 7, maka node 2 dan 1 di *pruning* ( tidak dihampiri).

Berdasarkan langkah-langkah diatas, adapun *flowchart* dari *pruning* terdapat pada Gambar 2. 16 berikut ini :




**Gambar 2. 16** *FlowChart Pruning*

sedangkan pseudocode untuk *pruning* terdapat pada

```
evaluate (node, alpha, beta)
    if node is a leaf
        return the heuristic value of node
    if node is a minimizing node
        for each child of node
            beta = min (beta, evaluate (child, alpha, beta))
            if beta <= alpha
                return beta
        return beta
    if node is a maximizing node
        for each child of node
            alpha = max (alpha, evaluate (child, alpha, beta))
            if beta <= alpha
                return alpha
        return alpha
```

**Gambar 2. 17** Pseudocode *Pruning*

## 2.6 OOP *(Object Oriented Programming)*

Objek adalah kesatuan entitas yang memiliki sifat dan tingkah laku. Dalam kehidupan sehari-hari, objek adalah benda, baik benda berwujud nyata seperti manusia, hewan, mobil, komputer, handphone, pena, ataupun benda yang tidak nyata atau konsep, seperti halnya tabungan bank, sistem antrian, sistem internet banking, dan sebagainya. Jadi pengertian OOP adalah konsep yang membagi program menjadi objek-objek yang saling berinteraksi satu sama lain. Objek adalah benda, baik benda yang berwujud nyata maupun benda yang tidak nyata (konsep). Jika menggunakan OOP maka akan ada enam keuntungan yang dapat diperoleh, yaitu [12]:

a. Alami (*Natural*).

b. Dapat diandalkan (*Reliable*).

c. Dapat digunakan kembali (*Reusable*).

d. Mudah untuk dalam perawatan (*Maintainable*).

e. Dapat diperluas (*Extendable*).

f. Efisiensi waktu.

Berikut ini beberapa bahasa pemrograman yang sudah menggunakan konsep

OOP, adalah :

1. *C++*.

2. *Visual C++*.

3. *Visual Basic*.

*4. Java*.

### 2.6.1 **Analisis Berorientasi Objek**

Analisis berorientasi objek atau *Object Oriented Analysis* (OOA) adalah tahap untuk menganalisis spesifikasi atau kebutuhan akan sistem yang akan dibangun dengan konsep berorientasi objek, apakah benar kebutuhan yang ada dapat diimplementasikan menjadi sebuah sistem berorientasi objek.

OOA biasanya menggunakan kartu CRC (*Component*, *Responsibility*, *Collaborator*) untuk membangun kelas-kelas yang akan digunakan atau menggunakan UML (*Unifed Modeling Language*) pada bagian diagram use case, diagram kelas, dan diagram objek [13].

### 2.6.2 **Desain Berorientasi Objek**

Desain berorientasi objek atau Object Oriented Design (OOD) adalah tahapan perantara untuk spesifikasi atau kebutuhan sistem yang akan dibangun dengan konsep berorientasi objek ke desain pemodelan agar lebih mudah diimplementasikan dengan pemograman berorientasikan objek.

Pemodelan berorientasi objek biasanya dituangkan dalam dokumentasi perangkat lunak dengan menggunakan perangkat lunak dengan menggunakan perangkat pemodelan berorientasi objek, diantaranya adalah UML (Unified Modeling Language). Kendala dan permasalahan pembangunan sistem berorientasi objek biasanya dapat dikenali dalam tahap ini [13].

### 2.6.3 **Pemodelan**

Pemodelan adalah gambaran dari realita yang sederhana dan dituangkan dalam bentuk pemetaan dengan aturan tertentu. Pemodelan dapat menggunakan

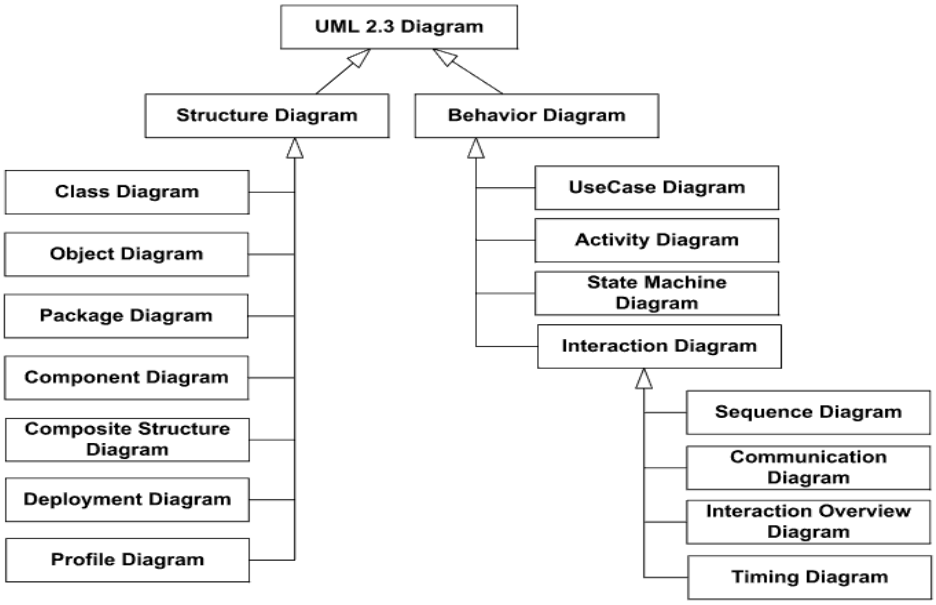
bentuk yang sama dengan realitas. Pemodelan juga banyak digunakan untuk merencanakan suatu hal agar kegagalan dan resiko yang mungkin terjadi dapat meminimalisir.

Pada dunia pembangunan perangkat lunak juga diperlunakan pemodelan. Pemodelan perangkat lunak digunakan untuk mempermudah langkah berikutnya dari pengembangan sebuah sistem sehingga lebih terencana. Pemodelan pada pembangunan perangkat lunak untuk menvisualkan perangkat lunak yang akan dibuat [13].

## 2.7 **UML (Unified Modeling Language)**

UML adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sebuah sistem. UML adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM. UML mendefinisikan diagram-diagram sebagai berikut [14]:

1. *Use case diagram*
2. *Activity diagram*
3. *Sequence diagram*
4. *Class diagram*

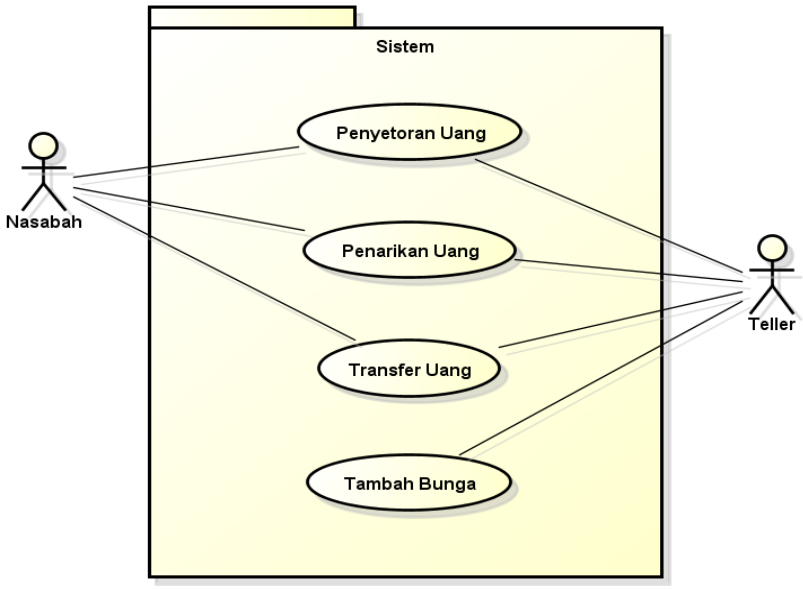



**Gambar 2. 18** Contoh Diagram UML **[14]**

UML dapat digunakan untuk membuat model untuk semua jenis perangkat lunak, dimana perangkat lunak tersebut dapat berjalan pada piranti keras, sistem operasi dan jaringan apapun, serta ditulis dalam bahasa pemrograman apapun. Tetapi karena UML juga menggunakan class dan operation dalam konsep dasarnya, maka ia lebih cocok untuk penulisan piranti lunak dalam bahasa berorientasi objek.

### 2.7.1 **Use Case Diagram**

*Use case diagram* adalah suatu model untuk menggambarkan fungsionalitas yang diharapkan dari sebuah sistem. Yang ditekankan adalah "apa" yang diperbuat sistem, dan bukan "bagaimana". Sebuah *use case* merepresentasikan sebuah interaksi antara aktor dengan sistem. *Use case* merupakan sebuah pekerjaan tertentu, misalnya *login* ke sistem, meng-*create* sebuah daftar belanja, dan sebagainya. Seorang atau sebuah aktor adalah sebuah entitas manusia atau mesin yang berinteraksi dengan sistem untuk melakukan pekerjaan-pekerjaan tertentu seperti terdapat pada Gambar 2. 19.




**Gambar 2. 19** Contoh *Use Case Diagram*

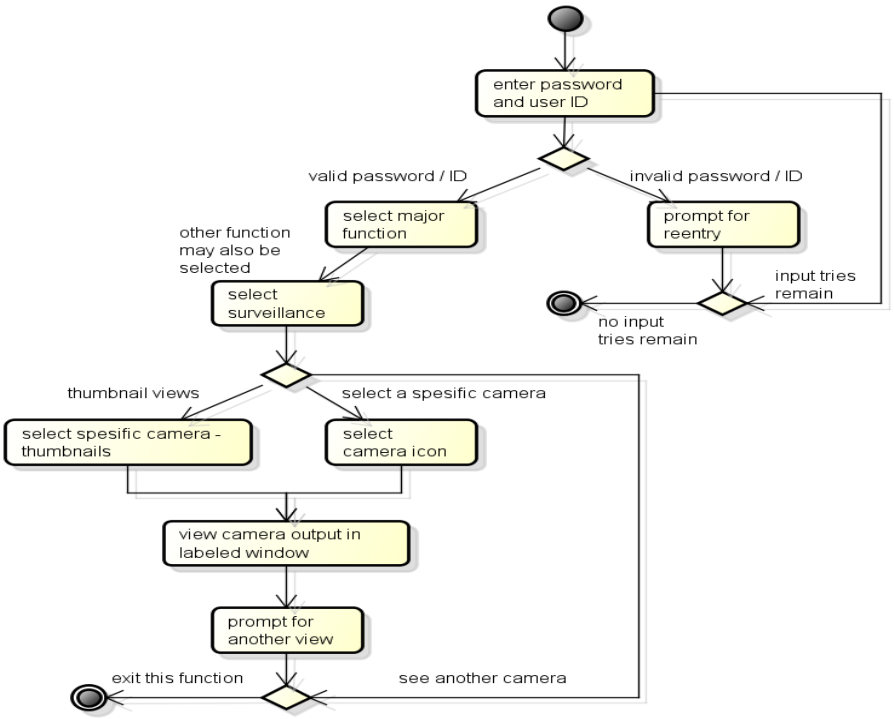
*Use case diagram* dapat sangat membantu menyusun *requirement* sebuah sistem, mengkomunikasikan rancangan dengan klien, dan merancang *test case* untuk semua *feature* yang ada pada sistem. Sebuah *use case* dapat meng-*include* fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang meng-*include* dieksekusi secara normal.

Sebuah *use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. Sebuah *use case* juga dapat meng-*extend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain.

### 2.7.2 **Activity Diagram**

*Activity diagram* menggambarkan berbagai alur aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alur berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat

menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi seperti terdapat pada Gambar 2. 20.

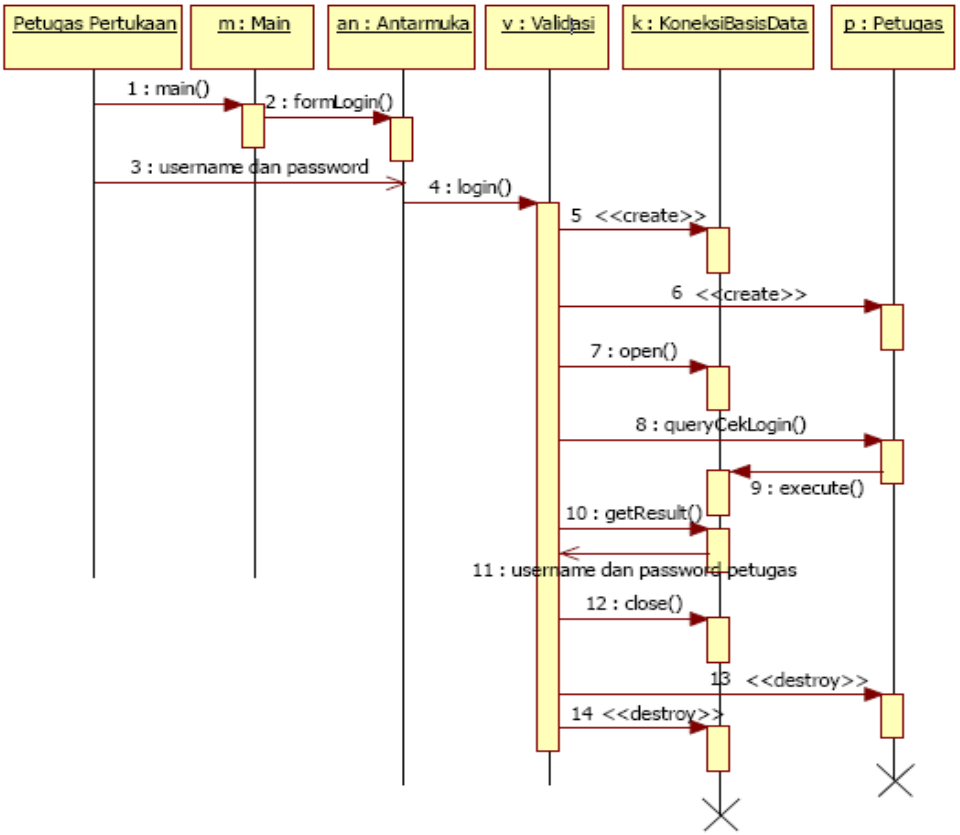



**Gambar 2. 20** Contoh *Activity Diagram*

*Activity diagram* merupakan *state diagram* khusus, dimana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

### 2.7.3 ***Sequence* Diagram**

*Sequence diagram* menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pemain, *display*, dan sebagainya) berupa *message* yang digambarkan terhadap waktu. *Sequence diagram* terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait) seperti terdapat pada Gambar 2. 21.




**Gambar 2. 21** Contoh *Sequence Diagram*

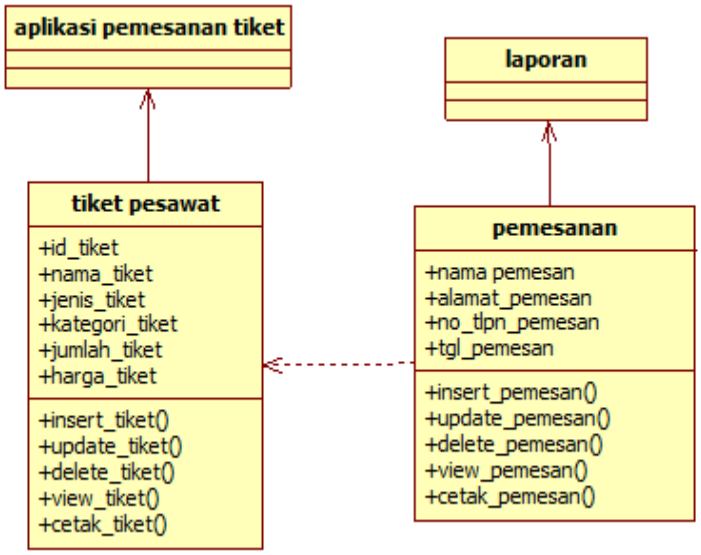
*Sequence diagram* biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah *event* untuk menghasilkan *output* tertentu. Diawali dari apa yang men-*trigger* aktivitas tersebut, proses dan perubahan apa saja yang terjadi secara internal dan *output* apa yang dihasilkan.

### 2.7.4 ***Class* Diagram**

*Class diagram* merupakan diagram statis. *Class diagram* mewakili pandangan statis dari aplikasi. *Class diagram* tidak hanya digunakan untuk memvisualisasikan, menggambarkan, dan mendokumentasikan perbedaan aspek dari sistem, tetapi juga untuk membangun kode *executable* dari aplikasi *software*.

*Class diagram* menampilkan koleksi dari *class, interface, associations, collaborations*, dan *constraints*. Hal ini juga dikenal sebagai diagram struktural.

*Class diagram* memberikan pandangan dari kelas desain dari sistem. *Class diagram* mengandung representasi dari kelas dan paket dan bagaimana mereka berhubungan. Berikut contoh *class* diagram yang terdapat pada Gambar 2. 22.




**Gambar 2. 22** *Class* diagram

## 2.8 Javascript

Javascript adalah sekumpulan perintah khusus yang digunakan untuk membuat halaman web yang lebih responsif dan interaktif. Javascript merupakan bahasa script yang dicantumkan pada sebuah halaman web dan dijalankan pada penjelajah web yang memberikan kemampuan tambahan pada HTML dengan mengizinkan pengeksekusian perintah di sisi user (penjelajah web), bukan di sisi server web.

Javascript bergantung kepada penjelajah web yang memanggil halaman web yang berisi script-script dari Javascript yang terselip di dalam dokumen HTML. Script javascript yang dimasukkan di dalam file HTML harus dimasukkan di antara tag <script> dan </script>. Javascript tidak memerlukan kompilator atau program khusus untuk menjalankannya karena Javascript sendiri sudah termasuk di dalam penjelajah web tersebut.

## 2.9 Jetbrains WebStorm

JetBrains WebStrom merupakan salah satu aplikasi yang dibuat untuk membangun halaman web berbasis PHP. WebStrom menyediakan *platform* untuk kode PHP, HTML, dan JavaScript. Aplikasi inilah yang digunakan dalam pembuatan simulasi game flow free color.

## 2.10 Pengujian Black Box

Pengujian black box merupakan pendekatan komplementer dari teknik white box, karena pengujian black box diharapkan mampu mengungkap kelas kesalahan yang lebih luas dibandingkan teknik white box. Pengujian black box berfokus pada pengujian persyaratan fungsional perangkat lunak, untuk mendapatkan serangkaian kondisi input yang sesuai dengan persyaratan fungsional suatu program.

Pengujian black box adalah pengujian aspek fundamental sistem tanpa memperhatikan struktur logika internal perangkat lunak. Metode ini digunakan untuk mengetahui apakah perangkat lunak berfungsi dengan benar. Pengujian black box merupakan metode perancangan data uji yang didasarkan pada spesifikasi perangkat lunak. Data uji dibangkitkan, dieksekusi pada perangkat lunak dan kemudian keluaran dari perangkat lunak dicek apakah telah sesuai dengan yang diharapkan. Pengujian black box berusaha menemukan kesalahan dalam kategori :

1. fungsi-fungsi yang tidak benar atau hilang
2. kesalahan interface
3. kesalahan dalam struktur data atau akses database eksternal
4. kesalahan kinerja
5. inisialisasi dan kesalahan terminasi.