BAB 2

TINJAUAN PUSTAKA

2.1 Permainan (*Game*)

Permainan merupakan sebuah aktivitas rekreasi dengan tujuan bersenang-senang ataupun untuk mengisi waktu luang. Permainan biasanya dilakukan secara sendiri-sendiri ataupun secara kelompok. Permainan sendiri terbagi menjadi 2 bagian yaitu:

1. Permainan Anak-Anak

   Banyak permainan yang sering dilakukan oleh anak-anak bersama teman-teman mereka di sekitar rumah mereka. Mereka biasanya berkelompok, senang berlarian, ataupun kadang duduk melingkar untuk memainkan salah satu permainan, dan dari situlah biasanya akan tercipta suatu keakraban diantara mereka. Beberapa permainan dari masa lalu disebut dengan permainan tradisional, sedangkan untuk permainan yang muncul di jaman sekarang ini disebut dengan permainan modern dan biasanya menggunakan peralatan canggih seperti halnya *video game*. Dan berikut beberapa jenis permainan yang bisa dimainkan oleh anak-anak :

   a. Bentengan.
   b. Boneka.
   c. Congklak.
   d. Goncang Kaleng.
   e. Kasti.

f. Layangan.

g. Main Hadang.

h. Petak Umpet.

i. Yoyo.

2. Permainan Dewasa

Untuk permainan orang dewasa biasanya memerlukan pemikiran untuk memainkannya. Dan berikut beberapa jenis permainan yang bisa dimainkan oleh orang dewasa :

a. Kartu Remi atau Bridge.

b. Catur.

c. Domino.

d. Kuis.

e. Sudoku.

2.1.1 Permainan Video (*Video Game*)

Permainan video (*video game*) adalah permainan yang menggunakan interaksi antarmuka pengguna melalui gambar yang dihasilkan oleh piranti video. Permainan video umumnya menyediakan sistem penghargaan, misalnya *score* yang dihitung berdasarkan tingkat keberhasilan yang dicapai dalam menyelesaikan tugas-tugas yang ada di dalam permainan video tersebut. Sistem elektronik yang digunakan untuk menjalankan permainan video disebut *platform*, contohnya adalah komputer pribadi (*personal computer*) ataupun konsol permainan (*game console*).

1. Permainan Komputer (*Computer Game*)

Permainan komputer adalah permainan video yang dimainkan pada komputer pribadi, dan bukan pada konsol permainan maupun mesin ding-dong. Permainan komputer telah berevolusi dari sistem grafis sederhana sampai menjadi sangat kompleks dan mutakhir.

Permainan *online* (*online game*) adalah jenis permainan video atau permainan komputer dengan menggunakan jaringan komputer dan juga internet sebagai media interaksinya.

Permainan online terdiri dari dua unsur utama, yaitu *server* dan *client*. *Server* adalah penyedia layanan gaming yang merupakan basis agar *client-client* yang terhubung dapat memainkan permainan dapat melakukan komunikasi dengan baik. Suatu *server* pada prinsipnya hanya melakukan administrasi permainan dan menghubungkan *client-client*. Sedangkan *client* adalah pengguna permainan dan memakai kemampuan *server*. Contoh permainan *online* adalah *Ragnarok Online, Risk Your Life, Xian* dll.

2. Konsol Permainan (*Game Console*)

Konsol permainan adalah sebuah mesin elektronik yang dirancang khusus untuk memainkan permainan video. Perangkat keluarannya biasa berupa monitor komputer atau televisi dan alat masukan utamanya berupa sebuah pengendali. Konsol permainan yang pertama kali dibuat adalah *Atari*, kemudian dilanjutkan dengan *Nintendo* yang sukses pada tahun 1985-1989. Konsol permainan modern sekarang ini antara lain adalah *Playstation* buatan perusahaan *Sony*. Untuk

Console yang berbentuk kecil dan dapat dibawah kemana-mana biasa disebut *Portable Console* [5].

2.1.2 Klasifikasi Permainan (*Game Classification*)

Berikut ini adalah klasifikasi permainan berdasarkan pembatasan usia dari pemainnya, menurut ESRB (*Entertainment Software Rating Board*) adalah sebagai berikut :

1. ***Early Childhood (3+)*** memiliki isi yang mungkin cocok untuk usia 3 keatas. Tidak mengandung materi yang tidak pantas seperti adegan kekerasan.
2. ***Everyone (6+)*** memiliki konten yang mungkin cocok untuk usia 6 keatas. Judul-judul dalam kategori ini mungkin berisi sedikit kartun, fantasi atau kekerasan ringan dan tidak jarang menggunakan bahasa yang ringan.
3. ***Everyone (10+)*** memiliki konten yang mungkin cocok untuk usia 10 keatas. Judul-judul dalam kategori ini mungkin berisi lebih banyak kartun, fantasi atau kekerasan ringan, bahasa ringan dan juga minimal tema sugestif.
4. ***Teen (13+)*** memiliki konten yang mungkin cocok untuk usia 13 keatas. Judul-judul dalam kategori ini mungkin mengandung kekerasan, tema sugestif, humor kasar, sedikit darah, simulasi perjudian, dan tidak jarang menggunakan bahasa yang kuat.
5. ***Mature (17+)*** memiliki konten yang mungkin cocok untuk orang usia 17 dan lebih tua. Judul-judul dalam kategori ini mungkin berisi *intens* kekerasan, darah, konten seksual dan bahasa yang kuat.

6. ***Adults Only (18+)*** memiliki konten yang hanya boleh dimainkan oleh orang-orang 18 tahun keatas. Judul-judul dalam kategori ini mungkin termasuk adegan yang *intens,* kekerasan dan juga mengandung konten seksual.
7. ***RP (Rating Pending)*** telah disampaikan kepada ESRB dan sedang menunggu penilaian akhir. (Simbol ini hanya muncul dalam iklan sebelum permainan dibebaskan) [6].

2.1.3 Sudut Pandang Permainan (*Game View Point*)

Sebuah *game* biasanya mempunyai sudut pandang permainan tersendiri disesuaikan berdasarkan *genre game* yang diambil. Berikut beberapa macam sudut pandang permainan yang biasa digunakan :

1. *Side Scrolling*

   Adalah sudut pandang permainan yang terlihat dari samping dan memungkinkan karakter utama untuk bergerak dari kiri ke kanan serta memungkinkan *background* pada *game* seolah-olah bergeser mengikuti pergerakan karakter utama.

2. *Top Down*

   Adalah sudut pandang permainan yang memungkinkan karakter utama bermanuver ke empat arah namun cara permainannya sendiri bergeser dari bawah ke atas, dan biasanya *game* yang menggunakan sudut pandang permainan jenis ini adalah *shooter game*.

3. *Isometric*

Adalah sudut pandang permainan yang memungkinkan permainan terlihat diantara sisi *Side Scrolling* dan juga *Top Down*, dan biasanya diterapkan pada *game* dengan *genre* RTS (*Real Time Strategy*).

2.1.4 Jenis Permainan (*Genre Game*)

Game dari tahun-tahun semakin banyak saja jenisnya, teknologi yang dipakaipun semakin canggih. Nah dari ratusan atau bahkan ribuan *game* yang telah bermunculan di dunia, *game-game* tersebut dikelompokkan menjadi beberapa *genre*. Pada dasarnya, *genre-genre* game yang beredar di pasaran dapat dikelompokkan sebagai berikut.

1. *Action Game*

Adalah tipe *game* dengan fitur utama berupa banyaknya aksi di mana pemain harus memiliki keterampilan reaksi yang cepat untuk menghindari musuh atau menghindari rintangan. Pengembang *game* tipe ini perlu memastikan *game* yang dibuat dioptimasi sehingga pemain memiliki pengalaman bermain yang baik, yang tidak terganggu oleh *delay* proses yang lama, contohnya : *Metal Gear Solid*.

2. *Adventure Game*

Adalah tipe *game* yang umumnya membuat pemain harus berjalan mengelilingi suatu tempat yang terkondisi, seperti sebuah istana, gua yang berkelok, dan planet yang jauh. Pemain melakukan navigasi suatu area, mencari pesan-pesan rahasia, memperoleh obyek yang memiliki kemampuan yang

bervariasi, bertempur dengan musuh, dan lain-lain. Untuk membuat *game* ini, diperlukan perencanaan yang akurat sehingga memiliki alur cerita yang menarik bagi pemain, contohnya : *Tomb Raider*.

3. *Sport Game*

Adalah tipe *game* yang berupa kompetisi antara dua pemain atau lebih, di mana pemain dapat berupa *individual* atau tim. Contoh *game* tipe ini antara lain sepakbola, bola basket, tenis, dan bilyard. Tergantung seberapa cepat permainan yang terjadi, aplikasi *game* perlu dioptimalkan, contohnya : *Pro Evolution Soccer*.

4. *RPG* (*Role Playing Game*)

Adalah tipe *game* yang seringkali berupa *multi* karakter *game* di mana setiap pemain memiliki karakter dengan kemampuan, kekuatan, dan kelemahan yang spesifik. Para pemain saling berkompetisi, berinteraksi, dan bertempur satu sama lain. Tampilan grafis yang khas untuk setiap karakter pemain ditambah dengan *storyline* yang mendebarkan akan sangat menarik dan memberikan pengalaman yang berbeda di dalam bermain, contohnya : *Final Fantasy*.

5. *Platform Game*

Adalah tipe *game* yang mengharuskan pemain mengarahkan suatu obyek dengan melalui berbagai tahap atau tingkatan area untuk menyerang musuh dan menghindar terhadap serangan. Tipe *game* ini sedikit serupa dengan *action game*, tetapi aksinya tidak secepat *action game*. Teknik *collision detection* sangat sering dimanfaatkan pada tipe *game* ini, contohnya : *Sonic The Hedgehog & Mario Bros.*

6. *Puzzle Game*

Adalah tipe *game* yang umumnya membuat pemain menggunakan kemampuan berpikirnya sebagai pengganti keterampilan reaksi yang cepat karena terdapat rahasia yang perlu dipecahkan. *Game* ini lebih bersifat *statis* dibanding *action game*. Pembuatan *game* tipe ini seringkali ditunjang dengan AI (*Artificial Intelligence*), contohnya : *7 Wonders of The Ancient World.*

7. *Sandbox Game*

Adalah tipe *game* yang umumnya ditujukan untuk menjelajahi suatu kota atau tempat dan bebas berinteraksi dengan objek disekitarnya. Mungkin *game* ini bisa di bilang mirip RPG tapi *Sandbox* sangat berbeda dengan RPG *game*, perbedaannya adalah tidak adanya peningkatan level karakternya, contohnya : *GTA & Bully*.

8. *FPS* (*First Person Shooter*)

Adalah tipe *game* yang mengunakan sudut pandang orang pertama untuk membidik atau membunuh musuh, sehingga kita hanya melihat tangannya saja dan tidak melihat tubuh karakter yang dimainkan, contohnya : *Call of Duty*.

9. *TPS* (*Third Person Shooter*)

Adalah tipe *game* yang mirip dengan FPS, tapi bedanya tipe *game* ini mengunakan sudut pandang orang ke-3, sehingga kita bisa melihat seluruh tubuh karakter yang dimainkan, contohnya : *Gears of Wars.*

10. *Music Game*

Adalah tipe *game* yang menuntut pemainnya untuk menekan tombol sesuai dengan tombol yang ada di layar dengan diiringi dengan musik, contohnya : *Guitar Hero & Dance Dance Revolution.*

11. *Fighting Game*

Adalah tipe *game* yang intinya harus menjatuhkan lawan tandingnya, entah itu dengan pukulan, tendangan, *combo*, maupun dengan jurus *special*, contohnya : *Tekken & Mortal Kombat.*

12. *RTS* (*Real Time Strategy*)

Adalah tipe *game* yang mengharuskan pemainnya menggunakan taktik dan strategi untuk jeli dalam melihat setiap peluang, kelemahan musuh dan bijaksana dalam menggunakan sumber daya yang ada, contohnya : *Command & Conquer.*

13. *Simulation Game*

Adalah tipe *game* yang memberikan pengalaman atau interaksi sedekat mungkin dengan keadaan yang aslinya, meskipun terkadang keadaan tersebut masih eksperimen atau bahkan fiktif, tapi ada penekanan khusus pada detail dan pengalaman *realistik* menggunakan keadaan tersebut, contohnya : *Gran Turismo.*

14. *Racing Game*

Adalah tipe *game* yang tujuannya adalah mencapai garis finish dari suatu *race*, dalam *game* ini biasanya pemain dapat memilih & membeli kendaraan, mendandani, melakukan *upgrade* mesin , contohnya : *Need For Speed* [7].

2.2 Kecerdasan Buatan / AI (*Artificial Intelligence*)

Sebagian kalangan mengartikan AI sebagai kecerdasan buatan, kecerdasan artifisial, intelijensia artifisial, atau intelijensia buatan. Istilah AI pun sudah sangat melekat dengan berbagai media ilmiah maupun non ilmiah.

2.2.1 Pengertian Kecerdasan Buatan / AI (*Artificial Intelligence*)

Berikut ini adalah beberapa definisi AI yang disampaikan oleh beberapa ahli. Para ahli mendefinisikan AI secara berbeda-beda tergantung pada sudut pandang mereka masing-masing. Ada yang fokus pada logika berpikir manusia saja, tetapi ada juga yang mendefinisikan AI secara lebih luas pada tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokan definisi AI dalam empat kategori berdasarkan beberapa *textbook* yang berbeda, yaitu :

1. ***Thinking Humanly*** : *The Cognitive Modeling Approach*

Pendekatan ini dilakukan dengan dua cara sebagai berikut :

a. Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita berfikir. Tetapi, seseorang psikologi barat mengatakan "*how do you know that you understand?*" yang berarti bagaimana anda tahu bahwa anda mengerti?, karena pada saat anda menyadari pemikiran anda ternyata pemikiran tersebut sudah lewat dan digantikan kesadaran anda. Dengan demikian definisi ini terkesan mengada-ada dan tidak mungkin dilakukan.

b. Melalui eksperimen-eksperimen psikologi.

2. ***Acting Humanly*** : *The Turing Test Approach*

Pada tahun 1950, Alan Turing merancang suatu ujian bagi komputer berintelijensia apakah komputer tersebut mampu mengelabuhi seorang manusia yang menginterogasinya melalui *teletype* (komunikasi berbasis teks jarak jauh). Jika *interrogator* tidak dapat membedakan yang tidak dapat membedakan yang diinterogasi adalah manusia atau komputer, maka komputer berintelijensia tersebut lolos dari *turing test.* Komputer tersebut harus memiliki kemampuan berikut :

a. *Natural Language Processing*.
b. *Knowledge Representation.*
c. *Automated Reasoning*.
d. *Machine Learning*.
e. *Computer Vision*.
f. *Robotics*.

Turing test sengaja menghindari interaksi fisik antara *interrogator* dan komputer karena simulasi fisik manusia tidak memerlukan intelinjensia.

3. ***Thinking Rationally*** : *The Laws of Thought Approach*

Terdapat dua masalah dalam pendekatan ini, yaitu :

a. Tidak mudah untuk membuat pengetahuan informal dan menyatakan pengetahuan tersebut ke dalam *formal term* yang diperlukan oleh notasi logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.

b. Terdapat perbedaan besar antara dapat memecahkan masalah “dalam perinsip” dan memecahkannya “dalam dunia nyata”.

4. ***Acting Rationally*** : *The Rational Agent Approach*

Membuat inferensi yang logis merupakan bagian dari suatu *rational agent*. Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa didapatkan kesimpulan bahwa asik yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka *agent* dapat melakukan aksi berdasarkan kesimpulan tersebut.

Thinking humanly dan *acting humanly* adalah dua definisi dalam arti yang sangat luas. Sampai saat ini, pemikiran manusia yang diluar rasio yakni refleks dan intuitif (berhubungan dengan perasaan), belum dapat ditirukan sepenuhnya oleh komputer. Dengan demikian, kedua definisi ini dirasa kurang tepat untuk saat ini. Jika kita menggunakan definisi ini maka banyak produk komputasi cerdas saat ini yang tidak layak disebut sebagai produk AI.

Definisi *thinking rationally* terasa lebih sempit daripada *acting rationally*. Oleh karena itu, definisi AI yang paling tepat untuk saat ini adalah *acting rationally* dengan pendekatan *rational agent*. Hal ini berdasarkan pemikiran bahwa komputer bisa melakukan penalaran secara logis dan juga bisa melakukan aksi secara rasional berdasarkan hasil penalaran tersebut.

2.2.2 Kecerdaan Buatan / AI (*Artificial Intelligence*) Pada Saat Ini

Dengan semakin cepatnya perkembangan *hardware* dan *software* berbagai produk AI telah berhasil dibangun dan digunakan dalam kehidupan sehari-hari. Produk-produk tersebut dikelompokkan ke dalam teknik yang ada pada AI, yaitu : *searching, reasoning, planning,* dan *learning.*

Teknik *searching* digunakan untuk pencarian rute optimum untuk memandu seseorang di perjalanan. Misal di kota Goteborg Swedia, setiap taksi dilengkapi dengan sebuah komputer dan GPS (*Global Positioning Systems*). Saat penumpang menyebutkan suatu tempat tujuan tertentu, maka sopir tinggal memasukkan tempat tujuan tersebut ke komputer. Di layar komputer akan terlihat rute jalan yang harus dilalui untuk menuju tempat tersebut.

Software permainan catur yang disebut *HITECH* adalah sistem AI pertama yang berhasil mengalahkan Arnold Danker seorang *grandmaster* dunia. *HITECH* adalah sistem berbasis pengetahuan yang menggunakan teknik *reasoning*. Teknik *reasoning* ini dapat digunakan untuk melakukan penalaran dan menyimpan ribuan pengetahuan.

Di dunia manufaktur dan robotik, teknik *planning* memainkan peranan yang sangat penting. *Optimum-AIV* adalah *planner* (*software* yang menggunakan teknik *planning*) yang digunakan oleh *European Space Agency* untuk *Assembly* atau perakitan, *Integration* atau penggabungan, dan *Verification* (AIV) pesawat terbang. Software tersebut digunakan untuk membuat perencanaan dan untuk memonitor eksekusi terhadap perencanaan-perencanaan tersebut.

Teknik *learning* telah digunakan dalam berbagai bidang seperti transportasi, *speech processing*, *computer vision*, *robotics*, dan sebagainya. Seperti sebuah sistem *auto driver* yang disebut ALVINN telah berhasil dibuat oleh Dean Pomerleau dari Carnegie Mellon University, USA. Dengan sistem ALVINN ini sebuah mobil bisa berjalan sendiri tanpa disetir oleh manusia. Sistem tersebut dibuat menggunakan jaringan syaraf tiruan (JST) yang dilatih dengan berbagai gambar kondisi jalan raya yang ditangkap melalui kamera yang diletakkan di mobil. JST dilatih sedemikian rupa sehingga dapat menjadi lebih pintar untuk mengetahui kapan harus belok ke kiri,lurus atau belok ke kanan, dan seberapa tajam sudut beloknya.

2.2.3 AI yang digunakan

1. Algoritma A* (A Star)

Algoritma ini merupakan algoritma Best First Search yang menggabungkan Uniform Cost Search dan Greedy Best First Search. Biaya yang diperhitungkan didapat dari biaya sebenarnya ditambah dengan biaya perkiraan. Dalam notasi matematika dituliskan sebagai : f(n) = g(n) + h(n). Dengan perhitungan biaya seperti ini, algoritma A* (A Star) adalah complete dan optimal.

Sama dengan algoritma dasar Best First Search, algoritma A* (A Star) ini juga menggunakan OPEN dan CLOSED. Terdapat tiga kondisi bagi setiap suksesor yang dibangkitkan, yaitu :

1. Sudah berada di OPEN.
2. Sudah berada di CLOSED.

3. Tidak berada di OPEN maupun CLOSED.

Pada ketiga kondisi tersebut diberikan penanganan yang berbeda-beda. Jika suksesor sudah berada di OPEN, maka dilakukan pengecekan apakah perlu pengubahan parent atau tidak tergantung pada nilai g nya melalui parent lama atau parent baru. Jika melalui parent baru memberikan nilai g yang lebih kecil, maka dilakukan pengubahan parent. Jika pengubahan parent dilakukan, maka dilakukan pula perbaruan (update) nilai g dan f pada suksesor tersebut. Dengan perbaruan ini, suksesor tersebut memiliki kesempatan yang lebih besar untuk terpilih sebagai simpul terbaik (best node).

Jika suksesor sudah pernah berada di CLOSED, maka akan dilakukan pengecekan apakah perlu pengubahan parent atau tidak. Jika ya, maka dilakukan perbaruan nilai g dan f pada suksesor tersebut serta pada semua “anak cucunya” yang sudah pernah berada di OPEN. Dengan perbaruan ini, maka anak cucunya tersebut memiliki kesempatan lebih besar untuk terpilih sebagai simpul terbaik (best node).

Jika suksesor tidak berada di OPEN maupun CLOSED, maka suksesor tersebut dimasukkan ke dalam OPEN. Tambahkan suksesor tersebut sebagai suksesornya best node. Hitung biaya suksesor tersebut dengan rumus f = g + h. Pseudo-code algoritma A* (Star) dapat di lihat pada gambar 2.1 :

```
function A*(masalah) return solusi
        OPEN ← S
        CLOSED ← array kosong

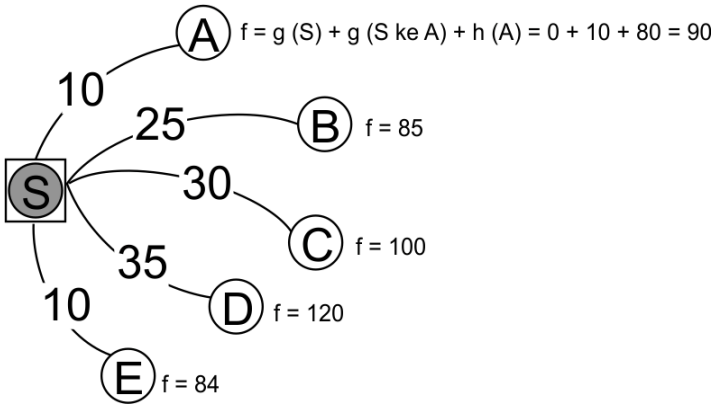
loop sampai goal ditemukan atau sampai tidak ada simpul didalam OPEN
        if OPEN = kosong then
            Gagal
        else
            BestNode = simpul yang ada di OPEN dengan f minimal
            Pindahkan simpul terbaik tersebut dari OPEN ke CLOSED
        if BestNode = goal then
            Sukses
        else
            Bangkitkan semua suksesor BestNode tapi jangan buat pointer
            Untuk setiap suksesor kerjakan :
            Hitung g (suksesor) = g (BestNode) + actual cost (dari BestNode ke
            suksesor)
            {Periksa suksesor}
            if suksesor ada di OPEN then {sudah pernah dibangkitkan tapi
            belum diproses}
                OLD = simpul di OPEN yang sama dengan suksesor tersebut
                Tambahkan OLD sebagai suksesor BestNode
                Buat pointer dari OLD ke BestNode
```

```
        Bandingkan nilai g (OLD) dengan g (suksesor)
        if g (OLD) lebih baik then
            Ubah parent OLD ke BestNode
            Ubah nilai g dan f yang ada pada OLD
        end
    else
        if suksesor ada di CLOSED then {sudah pernah dibangkitkan
        tapi belum diproses}
            OLD = simpul di CLOSED yang sama dengan suksesor
            tersebut
            Tambahkan OLD sebagai suksesor BestNode
            Bandingkan nilai g (OLD) dengan g (suksesor)
            if g (OLD) lebih baik then
                Ubah parent OLD ke BestNode
                Ubah nilai g dan f  yang ada pada OLD
                Propagasi untuk semua suksesor OLD dengan
                penulusuran DFS dengan aturan
                loop sampai simpul suksesor tidak ada di OPEN atau
                simpul tidak punya suksesor
                  if suksesor ada di OPEN then
                      Propagasi diteruskan
                  else
                      if nilai g via suksesor lebih baik then
```

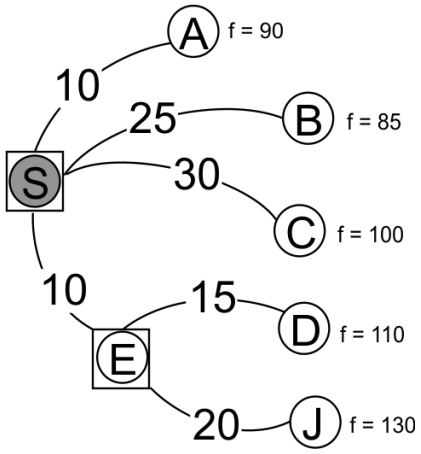
```
                                    Propagasi diteruskan
                                else
                                    Propagasi dihentikan
                                end
                            end
                        end
                    else {suksesor tidak ada di OPEN maupun CLOSED}
                        Masukkan suksesor ke OPEN
                        Tambahkan suksesor tersebut sebagai suksesornya
                        BestNode
                        Hitung f = g (suksesor) + h (suksesor)
                    end
                end
            end
end
```

Gambar 2.1 Notasi pseudo-code algoritma A* (*A Star*)

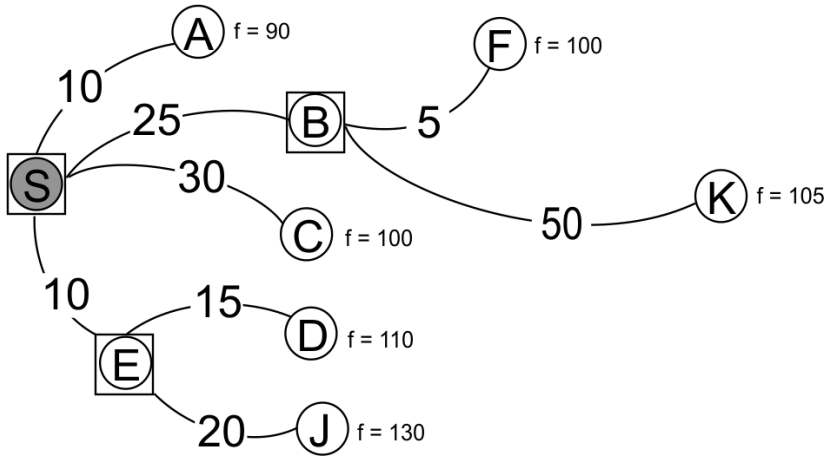
Untuk memperjelas pemahaman mengenai algoritma A* (*A Star*), maka berikut ini adalah langkah-langkah pencarian rute berdasarkan algoritma ini yang dapat dilihat pada gambar 2.2, gambar 2.3, gambar 2.4, gambar 2.5, gambar 2.6 dan gambar 2.7 :



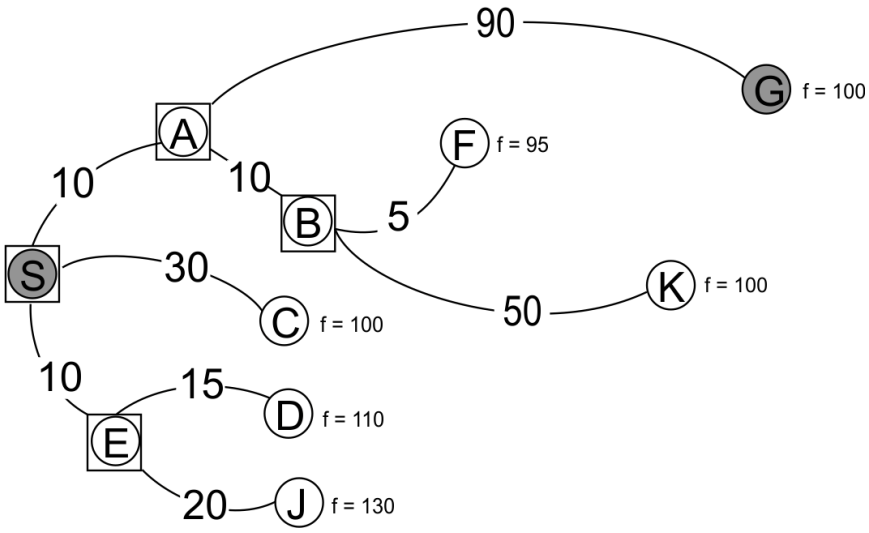
Gambar 2.2 Langkah 1 pencarian rute dengan algoritma A* (*A Star*)



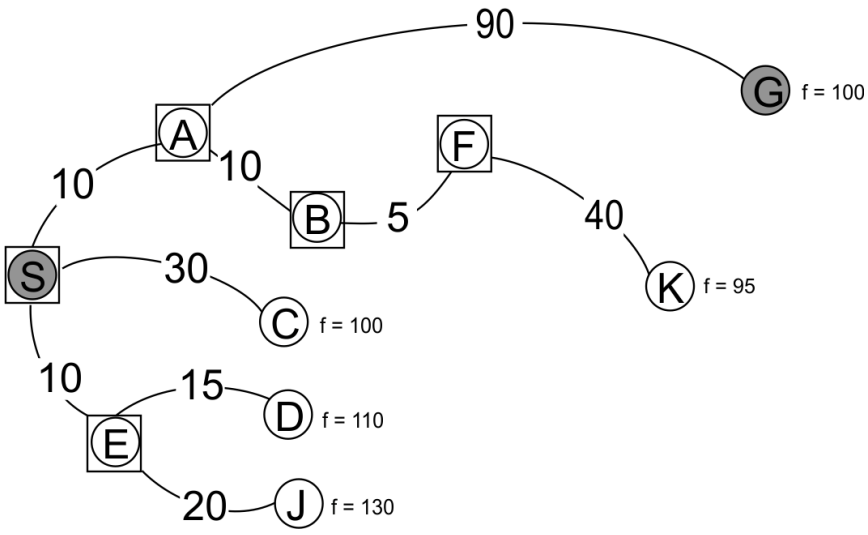
Gambar 2.3 Langkah 2 pencarian rute dengan algoritma A* (*A Star*)



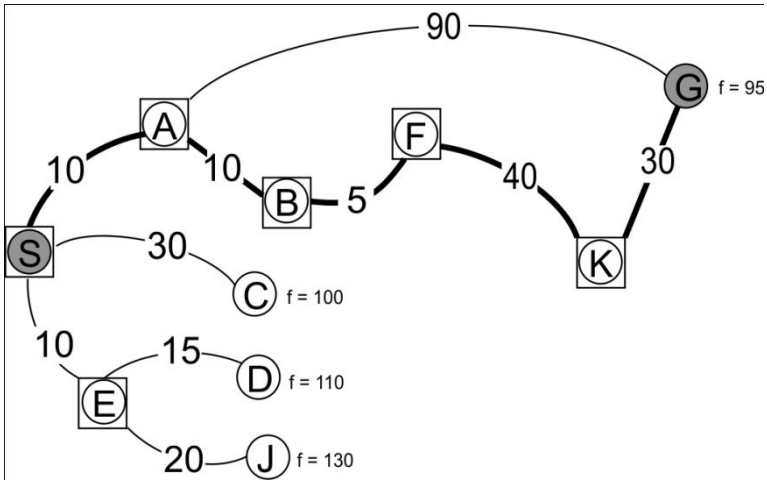
Gambar 2.4 Langkah 3 pencarian rute dengan algoritma A* (*A Star*)



Gambar 2.5 Langkah 4 pencarian rute dengan algoritma A* (*A Star*)



Gambar 2.6 Langkah 5 pencarian rute dengan algoritma A* (*A Star*)



Gambar 2.7 Langkah 6 pencarian rute dengan algoritma A* (*A Star*)

Simpul dengan tanda kotak menyatakan bahwa simpul tersebut berada di CLOSED (pernah terpilih sebagai BestNode) sedangkan simpul tanpa tanda kotak menyatakan simpul yang ada berada di OPEN.

Untuk langkah pertama, karena di OPEN hanya terdapat satu simpul yaitu S, maka S langsung terpilih sebagai BestNode dan dipindahkan ke CLOSED. Kemudian dibangkitkan semua suksesor S, yaitu : A, B, C, D dan E. Karena kelima suksesor tidak ada di OPEN dan CLOSED, maka kelimanya dimasukkan ke OPEN. Langkah pertama ini menghasilkan OPEN = [A, B, C, D, E] dan CLOSED = [S].

Langkah kedua, E dengan biaya terkecil (yaitu 84) terpilih sebagai BestNode dan dipindahkan ke CLOSED. Lalu semua suksesor E dibangkitkan, yaitu : D dan J. Karena belum pernah ada di OPEN maupun CLOSED sebelumnya maka J dimasukkan ke OPEN. Sedangkan simpul D sudah ada di OPEN, maka harus dicek apakah parent dari D perlu diganti atau tidak. Ternyata biaya dari S ke D melalui E (yaitu 10 + 15 = 25) lebih kecil daripada biaya S ke D (yaitu 35). Oleh karena itu, parent dari D harus dirubah, yang semula S menjadi E. Dengan perubahan parent ini maka nilai g dan f pada D juga diperbaharui (nilai g yang semula 35 menjadi 25, dan nilai f dari 120 menjadi 110). Langkah kedua ini menghasilkan OPEN = [A, B, C, D, J] dan CLOSED = [S, E].

Langkah ketiga, B dengan biaya terkecil (yaitu 85) terpilih sebagai BestNode dan dipindahkan ke CLOSED. Lalu semua suksesor B dibangkitkan, yaitu : A, F dan K. Karena belum pernah ada di OPEN maupun CLOSED, maka F dan K dimasukkan ke OPEN. Sedangkan simpul A sudah ada di OPEN, maka harus dicek apakah parent dari A perlu diganti atau tidak. Ternyata, biaya dari S ke A melalui B (yaitu 25 + 10 = 35) lebih besar daripada biaya S ke A (yaitu 10).

Oleh karena itu, parent dari A tidak perlu diubah (tetap S). Akhir dari langkah ke tiga ini menghasilkan OPEN = [A, C, D, F, J, K] dan CLOSED = [S, E, B].

Langkah keempat, A dengan biaya terkecil (yaitu 90) terpilih sebagai BestNode dan dipindahkan ke CLOSED. Lalu semua suksesor A dibangkitkan, yaitu : B dan G. Karena belum pernah ada di OPEN maupun CLOSED, maka G dimasukkan ke OPEN. Sedangkan simpul B sudah ada di CLOSED, maka harus dicek apakah parent dari B perlu diganti atau tidak. Ternyata biaya biaya dari S ke B melalui A (yaitu 10 + 10 = 20) lebih kecil daripada biaya dari S ke B (yaitu 25). Oleh karena itu parent dari B harus diubah, yang semula S menjadi A. Nilai g dan f pada B juga harus diperbarui (nilai g yang semula 25 menjadi 20, dan nilai f dari 85 menjadi 80). Nilai g dan f pada suksesor-suksesor B (sampai anak cucu) juga harus diperbarui menggunakan penulusuran DFS (Depth First Search). Dalam kasus ini, B hanya mempunyai dua anak (tidak punya cucu), yaitu F dan K. Nilai g (F) yang semula 30 diubah menjadi 25, dan nilai f (F) dari 100 menjadi 95. Nilai g (K) yang semula 75 diubah menjadi 70, dan nilai f (K) dari 105 menjadi 100). Akhirnya, OPEN = [C, D, F, G, J, K] dan CLOSED = [S, E, B, A].

Langkah kelima, F dengan biaya terkecil (yaitu 95) terpilih sebagai BestNode dan dipindahkan ke CLOSED. Lalu semua suksesor F dibangkitkan, yaitu : K. Karena K sudah ada di OPEN, maka harus dicek apakah parent dari K perlu diganti atau tidak. Biaya dari S ke K melalui F ternyata lebih kecil daripada biaya dari S ke K melalui parent lama (B). Oleh karena itu parent dari K harus diubah, yang semula B menjadi F. Selanjutnya nilai g (K) yang semula 70 diubah

menjadi 65, dan nilai f (K) dari 100 menjadi 95). Akhirnya, OPEN = [C, D, F, G, J, K] dan CLOSED = [S, E, B, A, F].

Langkah keenam, K dengan biaya terkecil (yaitu 95) terpilih sebagai BestNode dan dipindahkan ke CLOSED. Lalu semua suksesor K dibangkitkan, yaitu G. Karena G sudah ada di OPEN, maka harus dicek apakah parent dari G perlu diganti atau tidak. Biaya dari S ke G melalui K ternyata lebih kecil daripada biaya dari S ke G melalui parent lama (yaitu A). Oleh karena itu parent dari G harus diubah, yang semula A menjadi K. Selanjutnya nilai g (G) yang semula 100 diubah menjadi 95, dan nilai f (G) dari 100 menjadi 95). Pada akhir langkah keenam ini, OPEN = [C, D, G, J] dan CLOSED = [S, E, B, A, F, K].

Selanjutnya G dengan biaya terkecil (yaitu 95) terpilih sebagai BestNode. Karena BestNode sama dengan goal, berarti solusi sudah ditemukan. Rute dan total biaya bisa ditelusuri balik dari G menuju S karena setiap simpul hanya memiliki satu parent dan setiap simpul memiliki informasi biaya sebenarnya (g). Penelusuran balik menghasilkan rute S-A-B-F-K-G dengan total jarak sama dengan 95 kilometer. Rute ini merupakan rute terpendek yang ada di graph tersebut. Jadi, algoritma A* (A STAR) adalah optimal. Tanpa ada batasan waktu dan memory, A* (A STAR) adalah complete (selalu menemukan solusi jika solusinya ada). Pada kasus tersebut A* (A STAR) membangkitkan dan menyimpan 10 simpul (dari 13 simpul yang ada pada graph). Untuk masalah yang lebih kompleks, misalkan pencarian rute terpendek pada *graph* yang terdiri dari 100 juta simpul, A* (*A STAR*) akan menghadapi masalah waktu proses dan memori yang dibutuhkan. Untuk menyelesaikan kedua masalah tersebut, maka

telah diusulkan berbagai variasi A* (*A STAR*) dengan karakteristik yang sesuai untuk permasalahan tertentu yang muncul [2].

2. Algoritma Dijkstra

Algoritma Dijkstra diterapkan untuk mencari lintasan terpendek pada *graph* berarah. Namun algoritma ini juga banar untuk *graph* tak berarah. Algoritma ini mencari lintasan terpendek dengan sejumlah langkah dengan menggunakan konsep *Greedy*. Prinsip *Greedy* pada algoritma Dijkstra menyatakan bahwa pada setiap langkah kita memilih sisi yang berbobot minimum dan masukkannya dalam himpunan solusi.

Misalkan, sebuah *graph* berbobot dengan n buah simpul dinyatakan dengan matriks M=[mij], yang dalam hal ini :

- mij = bobot sisi (i,j) (pada *graph* tak berarah mij = mji).
- mii = 0
- mij = ∞, jika tidak ada sisi dari simpul I ke simpul j.

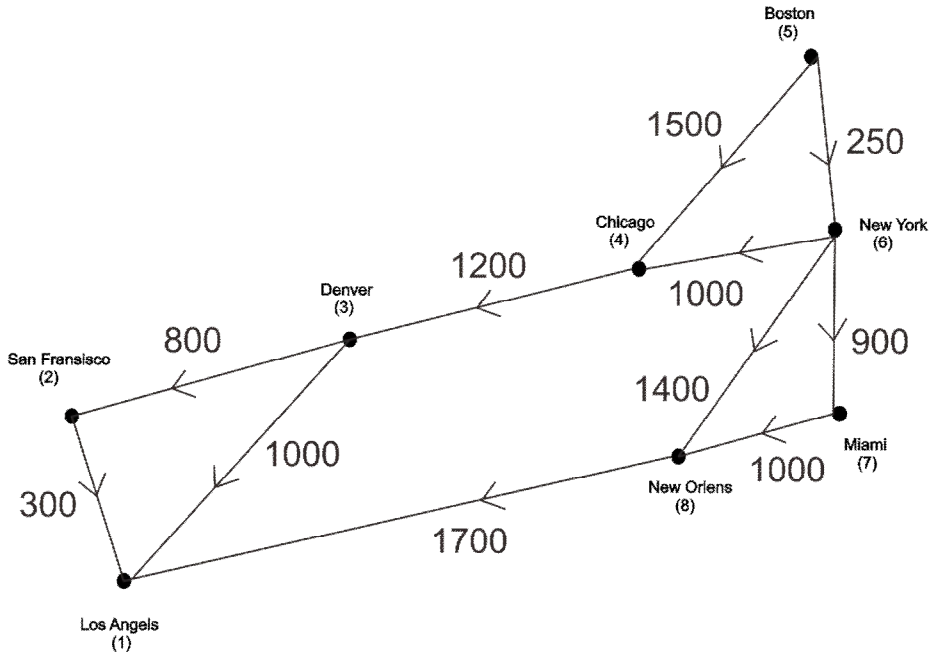
Selain matriks M, juga menggunakan tabel S = [si], yang dalam hal ini :

- si = 1, jika simpul i termasuk ke dalam lintasan terpendek.
- si = 0, jika simpul i tidak termasuk ke dalam lintasan terpendek.

Dan tabel D=[di], yang dalam hal ini :

- di = panjang lintasan dari simpul awal ke a ke simpul i.

Berikut contoh penggunaan algoritma Dijkstra pada kasus pencarian rute terpendek menggunakan *graph* yang dapat dilihat pada gambar 2.8, gambar 2.9 dan tabel 2.1 :



Gambar 2.8 *Graph* yang menyatakan beberapa nama kota di Amerika

Tabel 2.1 Matriks M

	j=1	2	3	4	5	6	7	8
i=1	0	∞	∞	∞	∞	∞	∞	∞
2	300	0	∞	∞	∞	∞	∞	∞
3	1000	800	0	∞	∞	∞	∞	∞
4	∞	∞	1200	0	∞	∞	∞	∞
5	∞	∞	∞	1500	0	250	∞	∞
6	∞	∞	∞	1000	∞	0	900	1400
7	∞	∞	∞	∞	∞	∞	0	1000
8	1700	∞	∞	∞	∞	∞	∞	0

```
procedure Dijkstra (input m: matriks, a: simpul awal){Mencari lintasan terpendek dari simpul awal a kesemua simpul lainnya. Masukan: matriks(m) dari graph berbobot G dan simpul awal a. Keluaran: lintasan terpendek dari a kesemua simpul lainnya}

Deklarasi
s1, s2,…, sn: integer
d1, d2,…, dn: integer
i, j, k : integer

Algoritma
{langkah 0 inisialisasi: }
for i ←1 to n do
    si←0
    di←mai
endfor
{langkah1:}
sa←1 {karena simpul a adalah simpul asal lintasan terpendek, jadi simpul a sudah pasti terpilih dalam lintasan terpendek}
da←∞ {tidak ada lintasan terpendek dari simpul a ke a}
{langkah 2, 3, …, n-1: }
for k ←2 to n-1 do
    j ← simpul dengan sj= 0 dan dj minimal
```

```
    sj←1b{simpul j sudah terpilih kedalam lintasan terpendek}
{perbarui table d}
for semua simpul i dengan si= 0 do
    if dj+ mji < di then
        di ← dj + mji
    endif
endfor
endfor
```

Gambar 2.9 Notasi *pseudo-code* Dijkstra [9]

2.3 *Tools* yang digunakan

2.3.1 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 kita menggunakan OOP maka akan ada enam keuntungan yang dapat diperoleh, yaitu :

1. Alami (*Natural*).
2. Dapat diandalkan (*Reliable*).
3. Dapat digunakan kembali (*Reusable*).
4. Mudah untuk dalam perawatan (*Maintainable*).
5. Dapat diperluas (Extendable).
6. Efisiensi waktu.

Berikut ini beberapa bahasa pemrograman yang sudah menggunakan konsep OOP, adalah :

1. *C++*.
2. *Visual C++*.
3. *Visual Basic*.
4. *Java*.

2.3.2 *Java*

Keunggulan *java* dibanding bahasa pemrograman lainnya, adalah sebagai berikut :

1. *Java* bersifat sederhana dan relatif mudah

   Java dimodelkan sebagian dari bahasa C++, namun dengan memperbaiki beberapa karakteristik dari C++ sendiri, seperti misalnya dalam mengurangi kompleksitas beberapa fitur, lalu penambahan fungsionalitas, serta penghilangan beberapa aspek pemicu ketidakstabilan sistem pada C++.

   Sebagai contoh, *Java* menggantikan konsep pewarisan lebih dari satu (*multiple inheritance*) dengan *interface,* menghilangkan konsep

pointer yang sering membingungkan, otomatis sistem alokasi *memory*, dan sebagainya. Ini membuat *Java* menjadi relatif sederhana dan mudah untuk dipelajari dibandingkan bahasa pemrograman lainnya.

2. *Java* berorientasi pada objek (*Object Oriented*)

   Java adalah bahasa pemrograman yang berorientasi objek bukan seperti *Pascal*, *Basic*, atau C yang masih berbasis prosedural. Dalam memecahkan masalah *Java* membagi program menjadi objek-objek, kemudian memodelkan sifat dan tingkah lakunya masing-masing. Selanjutnya *Java* menentukan dan mengatur interaksi antara objek yang satu dengan lainnya.

3. *Java* bersifat terdistribusi

   Pada dekade awal perkembangan PC (*Personal Computer*), komputer hanya bersifat sebagai *workstation* tunggal, tidak terhubung satu sama lain. Saat ini sistem komputerisasi cenderung terdisibusi, mulai dari *workstation client, e-mail server, database server, web server, proxy server dan sebainya.*

4. *Java* bersifat *Multiplatform*

   Dewasa ini kita mengenal banyak *platform Operating System*, mulai dari *Windows, Apple,* berbagai varian *UNIX* dan *Linux*, dan sebagainya. Pada umumnya program yang dibuat dan dikompile di suatu *platform* hanya bisa dijalankan di *platform* tersebut. *Java* bersifat *multi platform*, yakni dapat diterjemahkan oleh *Java Interpreter* pada berbagai sistem operasi.

5. *Java* bersifat *MultiThread*

Thread adalah proses yang dapat dikerjakan oleh program dalam suatu waktu. *Java* bersifat *Multithreading*, artinya dapat mengerjakan beberapa proses dalam waktu yang hampir bersamaan [10].

2.3.3 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 menyediakan 10 macam diagram yang dapat digunakan untuk memodelkan aplikasi berorientasi objek yang dibuat, yaitu :

1. *Use Case Diagram* untuk memodelkan proses bisnis.
2. *Conceptual Diagram* untuk memodelkan konsep-konsep yang ada di dalam aplikasi.
3. *Sequence Diagram* untuk memodelkan pengiriman pesan (*message*) antar *objects*.
4. *Collaboration Diagram* untuk memodelkan interaksi antar *objects*.

5. *State Diagram* untuk memodelkan perilaku *objects* di dalam sistem.
6. *Activity Diagram* untuk memodelkan perilaku *Use Cases* dan *objects* di dalam *system*.
7. *Class Diagram* untuk memodelkan struktur kelas.
8. *Object Diagram* untuk memodelkan struktur *object*.
9. *Component Diagram* untuk memodelkan komponen *object*.
10. *Deployment Diagram* untuk memodelkan distribusi aplikasi [8].

2.3.4 *Greenfoot*

Greenfoot merupakan *software* pembelajaran pemrograman berorientasi objek, dengan berbasis pemrograman *java* yang memungkinkan kita untuk membuat *game* 2 dimensi. *Greenfoot* diprakarsai oleh Michael Kolling pada tahun 2003. *Prototype* pertama dibangun oleh Michael Kolling dan Poul Henrikson pada tahun 2003-2004. Pembangunan dilanjutkan pada tahun 2005 oleh kelompok Blue J lain dari University of Kent dan Deakin University. *Greenfoot* versi 1.0 pertama dirilis pada tanggal 31 Mei 2006. Pada bulan Maret 2009, proyek *Greenfoot* menjadi *software Open Source* dan berlisensi dibawah GNU GPL. Untuk menjalankan *Greenfoot* ini diharuskan sudah menginstal program *java* 5 (JDK 1.5) atau program *java* terbaru. Didalam *Greenfoot* terdapat *project* yang dinamakan *scenario*.

Dalam *project* tersebut, kita dapat membuat dua macam tipe *class* dari *library Greenfoot*, yaitu *class world* dan *class actor*. *World* merupakan latar dari

game yang akan dibuat. *Class actor* adalah kelas yang berfungsi sebagai karakter pada *game* yang akan dibuat [11].

2.3.5 *Corel Draw*

Corel Draw merupakan salah satu aplikasi pengolah gambar berbasis *vector* yang banyak dipakai oleh pengguna PC. Karena berbagai kemudahan dan keunggulan yang dimiliki oleh *corel draw,* maka *corel draw* sering dimanfaatkan untuk *desktop publishing*, percetakan, dan bidang lain yang memerlukan pemrosesan visual.

Keunggulan mengolah gambar berbasis *vector* adalah ukuran hasil akhir yang dapat ditekan seminimal mungkin namun dengan kualitas yang tidak kalah dengan gambar berbasis *raster* atau *bitmap*.

Sebenarnya ada banyak sekali *software* desain grafis dan pengolah gambar berbasis *vector* yang bisa kita gunakan,misalnya saja diantaranya adalah *corel draw*, *photoshop*, *freehand*, *adobe ilustrator* dll. Namun *corel draw* merupakan program paling populer yang sering dimanfaatkan. Selain karena tampilannya yang *user friendly* dan mudah dipelajari, *corel draw* juga mempunyai berbagai keunggulan - keunggulan lain yang sangat bermanfaat [12].