BAB 2

LANDASAN TEORI

2.1 Permainan (*Game*)

Game adalah permainan elektronik yang menggunakan media komputer maupun mesin konsol seperti *Playstation* atau *Xbox* yang sudah merambah kemana mana,. Jika pada awalnya *game* dibuat untuk sekedar permainan sederhana yang menguji ketangkasan ataupun kecepatan reaksi pemaian, dalam perkembangan game itu mampu memberikan berbagai jenis game (genre) lain. Mulai dari game yang mengetengahkan cerita dan fantasi seperti jenis RPG (*role playing game*) misalnya *game* yang menjual simulasi dari keadaan nyata (balap mobil dan simulasi pesawat), sampai *game* yang sudah ada di dunia nyata mampu di hidupkan di dalam berbagai mesin g*ame* (c*atur, monopoly, billiard*). Berbagai genre terus bermunculan, sebagaian adalah perbaikan dari genre yang lama, sebagian lagi adalah genre baru dan sebagian lagi adalah kombinasi antar genre. Semua ini membuat game semakin variatif dan semakin banyak produk *game* yang muncul [1].

2.1.1 Klasifikasi Game Berdasarkan Jenis Permainan (Genre Game)

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

1. *Action Games*

   Pemain memerlukan refleks cepat, ketangkasan dan kecepatan reaksi untuk mengatasi rintangan dalam memainkan *game* ini. *Game* action cenderung memiliki *gameplay* dengan penekanan pada pertempuran, eksplorasi level dan

memecahkan teka-teki. Action games dibagi lagi kedalam beberapa kategori diantaranya :

a. *Action Adventure Game*, berfokus pada eksplorasi dan biasanya mempunyai unsur item gathering, penyelesaian *puzzle* simpel, dan pertarungan (*combat*). Contoh dari genre ini termasuk *The Legend of Zelda series* dan *Metroid series*.
b. *Stealth Game*, termasuk dalam genre *game* terbaru, biasanya digolongkan dalam mata-mata yang biasa melakukan aksinya secara rahasia. Contoh dari genre ini termasuk *Metal Gear series* dan *Tom Clancy's Splinter Cell*. *Survival Horror Game*, berusaha membuat player atau pemain menjadi tegang dan takut dengan elemen-elemen horor. Contoh dari genre ini termasuk *Resident Evil series* dan *Alone in the Dark*.
c. *Beat 'em Up Game*, seperti *Side-scrolling beat 'em up game*, adalah *game* combat satu orang melawan banyak musuh yang telah disediakan. Contoh dari genre ini termasuk *Dynasty Warriors series* dan *Final Fight*.
d. *Fighting Game*, adalah pertarungan dua player dengan jurus-jurus yang bisa dikeluarkan dengan menekan beberapa tombol pada *keyboard* dengan urutan tertentu. Contoh dari genre ini termasuk *Street Fighter II* dan *Tekken series.*
e. *Maze Game*, membutuhkan kecepatan berpikir dan bereaksi, serta berunsur ketepatan menavigasi. Contoh dari genre ini termasuk *Pac-Man*.
f. *Platform Game*, atau *platformers*, adalah game dengan *gameplay* berlari, melompat, mengayun, naik tangga, dan sebagainya. Contoh dari genre ini termasuk *Donkey Kong*, *Sonic the Hedgehog*, *Mario*, dan *Rayman*,

2. Adventure

Game adventure lebih menekankan unsur narasi dan cerita, biasanya pemain memecahkan teka-teki atau masalah dengan berinteraksi dengan karakter dalam

game dan lingkungan tanpa konfrontasi, pemain hanya mendapat sedikit tekanan dalam bentuk *action-based challanges* dengan waktu yang dibatasi. Hampir semua jenis game ini dibuat untuk dimainkan oleh satu pemain, dikarenakan penekanan yang sangat pada cerita dan karakter sehingga sulit untuk dimainkan bersama-sama pemain lainnya.

3. Management - Live Simulation

Merupakan *game* simulasi dimana tugas pemain untuk membangun, memperluas wilayah atau mengatur komunitas fiktif atau proyek dengan sumber terbatas. Terkadang pemain bertindak sebagai penguasa dengan mengontrol kehidupan orang-orang atau sebuah keluarga sampai peradaban tertentu.

4. Strategy

Dalam memainkan game jenis ini pemain memerlukan kehati-hatian, pemikiran dan perencanaan yang tepat untuk dapat meraih kemenangan. Selain memikirkan strategi dan taktik berperang, pemain diharuskan membuat konstruksi bangunan, mengendalikan populasi dan mengatur sumber daya alam [6].

5. *Game* Edukasi

Banyak pengamat *game* di Indonesia mengatakan hanya genre ini yang berhasil secara komersil (dalam artian tidak di bajak, seperti *game* keluaran luar). Genre ini sebenarnya lebih mengacu kepada isi dan tujuan *game,* bukan genre yang sebenarnya. Misalnya *game boby* bola sebenarnya campuran dari genre *arcade* dan *side scroller*. Terkadang ada genre lain yang di campurkan sehingga genre aslinya sudah kabur. Tapi secara keseluruhan *game* ini dikategorikan genre *edutainment*, bertujuan untuk memancing minat belajar anak sambil bermain [10].

Jenis-jenis Permainan Edukasi

1. *Sliding* (mainan luncuran)

   Ada banyak jenis mainan kayu yang memakai konsep luncuran. Misalnya *Sliding Car* (luncuran mobil), *Sliding Bug* (luncuran serangga).

Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Melatih kecepatan gerak mata.
b. Melatih konsentrasi, fokus dan pengamatan
c. Mengenalkan konsep tinggi-rendah, lamban-laju.
d. Juga mengandung unsur hiburan.

2. *Hammer Set*

Ada banyak jenis mainan palu yang dimainkan dengan cara mengetok atau memukulkan palu pada objeknya. Misalnya *Hammer* 3-6 *nails* (palu dengan 3-6 paku pasak).

Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Menyalurkan tenaga berlebih pada anak
b. Melatih anak untuk fokus
c. Mengenalkan konsep sebab-akibat
d. Melatih life-skill dasar
e. Melatih motorik kasar dan gerak otot
f. Mengenakan warna & bentuk (pada beberapa jenis)
g. Dapat menjadi media terapi ringan untuk kasus autisme & hiperaktifitas.

3. *Lacing/Sewing*

Mainan *Lacing/Sewing* (*meronce* & menjahit) memiliki kemiripan dalam memainkannya.

Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Mengenalkan *life-skill* dasar
b. Melatih kesabaran & kemandirian anak
c. Melatih motorik halus

4. Building Blocks

   Mainan Balok adalah cikal bakal dari semua mainan edukasi. Jenis dan bentuknya pun bermacam-macam, misalnya : tebal/tipis, berwarna/natural, bentuk campur/seragam.

   Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

   a. Melatih kesabaran & kemandirian anak
   b. Melatih anak untuk kreatif & imajinatif (anak akan dipandu untuk menggunakan otak kanan dengan lebih dominan)
   c. Mengenalkan anak pada bentuk atau elemen yang terdapat balok.
   d. Membiasakan anak menghadapi tantangan
   e. Mengenalkan anak pada konsep bangunan dan sebab-akibat.
   f. Menambah kepercayaan diri anak, karena bisa menumbuhkan rasa ‘puas’ sudah bisa menciptakan sesuatu.
   g. Jika dimainkan bersama (dengan orang tua/saudara/teman), juga bisa melatih kerjasama dan kemampuan berbahasa (dalam berbagi balok atau menceritakan apa yang sedang dikerjakan).

5. *Play Food*

   Play Food (mainan makanan) adalah mainan yang meniru bentuk-bentuk (atau gambar) makanan, sayur-buah-roti-kue.

   Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

   a. Bermain peran, melatih imajinasi
   b. Mengenalkan keterampilan hidup dasar (basic life skill)
   c. Melatih kemampuan bahasa (saat memperagakan bentuk apa yang sedang dipegang atau dikerjakan).
   d. Melatih motorik halus

6. Puzzle

   Puzzle dapat berupa jigsaw atau bentuk 3 dimensi, menganut asas potongan homogen ataupun acak, bisa berupa kepingan besar atau kecil atau gabungan keduanya.

   Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

   a. Melatih konsentrasi, ketelitian dan kesabaran

b. Memperkuat daya ingat
c. Mengenalkan anak pada konsep ‘hubungan’
d. Dengan memilih gambar/bentuk, dapat melatih anak untuk berfikir matematis (menggunakan otak kiri)

7. Stacking

Stacking adalah jenis mainan edukasi yang dimainkan dengan cara menyusun dan mengelompokkan. Dapat berupa stacking yang dimainkan dengan cara memasukkan kelompok warna/bentuk tertentu ke tiang pasak hingga menjadi kesatuan tertentu.

Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Mengenalkan konsep besar-kecil, jenis warna & bentuk
b. Belajar mengelompokkan warna/bentuk atau lainnya.
c. Melatih motorik halus (mengarahkan tangan untuk memasukkan kepingan melalui tiang yang disediakan)
d. Belajar memecahkan masalah (problem solving)

8. *Pulling* a*nd Pushing*

Mainan yang tergolong “*Pulling & Pushing*” adalah mainan yang melibatkan hampir semua bagian fisik anak (dalam gerak menarik & mendorong).

Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Melatih keseimbangan gerak tubuh
b. Melatih lokomotorik
c. Menambah kepercayaan diri anak

9. *Play Shapes*

Play Shapes adalah jenis mainan edukasi yang biasanya menggunakan berbagai bentuk sebagai dasar permainan. Misalnya bentuk-bentuk geometri dasar (*basic shapes*) 4/5/hingga 8 bentuk, atau bentuk-bentuk lainnya.

Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Mengenal bentuk, warna dan sifat benda.
b. Melatih motorik halus
c. Problem solving

10. Wire Game

Wire game adalah mainan edukasi yang menggunakan kawat yang dibuat meliuk-liuk dengan beraneka beads di dalamnya. Dimainkan dengan cara memegang beads lalu mengikuti alurnya dengan berkesinambungan, diusahakan tidak berhenti dari awal hingga akhir pondasi kawat.

Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Memperhalus gerakan tangan agar lebih lentur
b. Memperkuat konsentrasi anak
c. Dapat memperbaiki tulisan dan gambar anak
d. Menambah kepercayaan diri
e. Memperkuat hubungan otak kanan-kiri sehingga terlatih untuk menyelesaikan masalah dengan lebih cepat.

11. Maze

Secara ide pembuatannya, banyak kegunaan mainan maze (papan alur) yang sama dengan wire game. Namun maze memilki penampilan yang bermacam-macam hingga memperkaya pula kegunaan edukatifnya.

Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Memperhalus gerakan tangan
b. Dapat memperbaiki tulisan dan gambar anak
c. Melatih mengelompokkan
d. Memperkuat konsentrasi
e. Melatih menyelesaikan masalah (problem solving)

12. *Role and Music Toys*

Mainan yang tergolong dalam "*Role*" (bermain peran) misalnya *Tool Set* (alat bertukang), *Kitchen Set* (perangkat dapur). Sedangkan yang tergolong dalam "*Music Toys*" adalah semua mainan kayu yang mengandung unsur auditory atau yang berupa miniatur alat musik sesungguhnya.

a. Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :
b. Memperkaya imajinasi
c. Berlatih keterampilan hidup dasar
d. Memperbanyak kosakata / melatih bicara anak
e. Melatih kemampuan membedakan bunyi (music toys)

f. Menambah kepercayaan diri anak
g. Mengenalkan dengan aktifitas dan nama benda yang sesungguhnya

13. *Activity & Strategy Game*

Mainan yang tergolong dalam "*Activity & Strategy Game*" misalnya permainan Bowling, Dough (lilin mainan), Molding (mencetak).

Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Melatih gerak otot & motorik kasar
b. Melatih keterampilan
c. Melatih Koodinasi otak kanan-kiri
d. Mengenalkan bentuk, gambar, atau elemen lain yang melekat pada mainan yang bersangkutan.
e. Memperkuat kemampuan strategi anak / dapat melatih untuk menyelesaikan masalah dengan cepat & tepat
f. Melatih konsentrasi

14. *Cloth Books & Soft Toys*

Cloth Books (buku kain) dan *Soft Toys* (mainan lembut) biasanya identik dengan mainan bayi karena karakternya yang 'lunak' dan biasanya terbuat dari kain (ringan & gampang dibersihkan).

Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Memperkaya imajinasi
b. Warna-warna yang cerah/mencolok pada mainan ini dipercaya para ahli dapat meningkatkan point IQ bayi.
c. Memperkuat indera peraba & penglihatan
d. Memperbayak kosa kata (jika dijadikan media bercerita)

15. *Sorting & Matching*

Mainan jenis *Sorting & Matching* adalah mainan yang dimainkan dengan cara 'memilih & memasangkan'. Awalnya bahan mainan sorting 'dikuasai' oleh plastik. Pada umumnya, sisi edukasi mainan jenis ini berfungsi untuk :

a. Mengenal bentuk/angka/huruf
b. Berlatih mengelompokkan

c. Memperkuat kemampuan strategi anak / dapat melatih untuk menyelesaikan masalah dengan cepat & tepat
d. Menambah kepercayaan diri anak

2.1.2 Klasifikasi Game Berdasarkan Pembatasan Usia

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 rang-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) [3].

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*).

2.2 *Collison Detection*

Collision detection adalah proses pendeteksian tabrakan antara dua objek. Sebenarnya dalam game sendiri tabrakan tidak hanya terjadi antara dua objek, tetapi dapat terjadi juga antara satu objek dengan banyak atau banyak objek dengan banyak objek. Dalam game membutuhkan *collision detection* yg akurat karena setelah menentukan terjadinya *collision* atau tidak harus ditentukan apa yang terjadi pada objek yang ditabrak ataupun yang menabrak [13].

Collision detection ini juga berguna untuk menentukan posisi dari satu objek dengan objek yang lain sehingga tidak ada objek yang saling menembus. Sehingga game yang akan dibuat memiliki kesamaan dengan realita yang ada.

Collision detection memiliki teknik yg berbeda untuk diterapkan pada pembuatan *game* 2D dan game 3D.

Teknik *Collision Detection* pada 2D ada 2 cara yaitu :

a. *Rectangles collision detection*
b. *Circle rectangle collision detection*

Teknik *Collision Detection* pada 3D ada 2 cara yaitu :

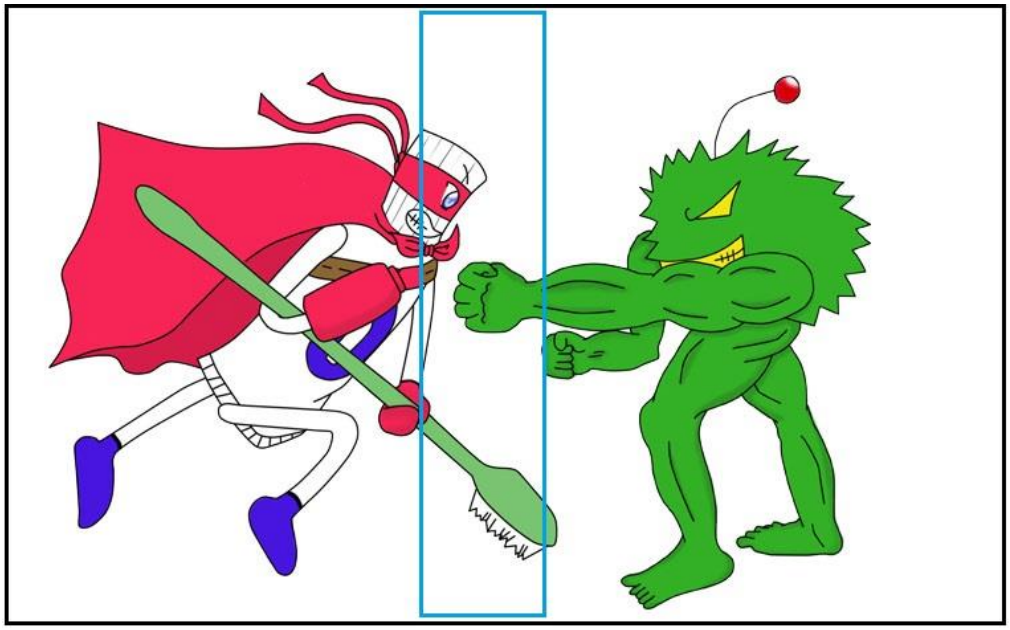
a. *Bounding box collision detection*
b. *Sphere collision detection*

2.2.1 *Game 2D Collision Detection*

Untuk *game* 2D setiap objek yang di perhatikan adalah vektor 2D nya, yaitu panjang dan lebar. Namun tidak setiap objek berbentuk persegi, lingkaran, segilima atau bentuk lain yang beraturan. Seperti bentuk pohon atau kuda dan lainnya, sehingga *collision detection* untuk objek2 seperti ini memerlukan pandekatan lain yang bisa mempermudah terutama dalam hal pemrograman. Karena untuk bentuk-bentuk yang tidak beraturan sangat sulit untuk dideteksi [13].

2.2.2 *Rectangles Collision Detection*

Teknik ini merupakan pendeteksian tabrakan menggunakan *bound* berbentuk persegi, dimana setiap sprite atau objek yang dibuat dalam game memiliki *bound* masing-masing. Cara ini cukup mudah karena hanya perlu membandingkan apakah ada *intersection*(irisan) pada kedua objek pada kordinat tertentu. Namun cara ini kurang akurat karena untuk *sprite* atau objek yang memiliki ruang kosong yang besar dengan *bound* nya. Hal ini akan muncul saat kondisi kedua batas objek tersebut beririsan walaupun sebenarnya tidak terjadi tabrakan maka akan dinggap terjadi tabrakan. Cara ini cocok untuk game yang tidak memperhatikan akurasi deteksi tabrakan atau untuk game yang menggunakan *sprite* yang bentuknya mendekati persegi [13].



Gambar 2.1 Contoh *Rectangles Collision Detection*

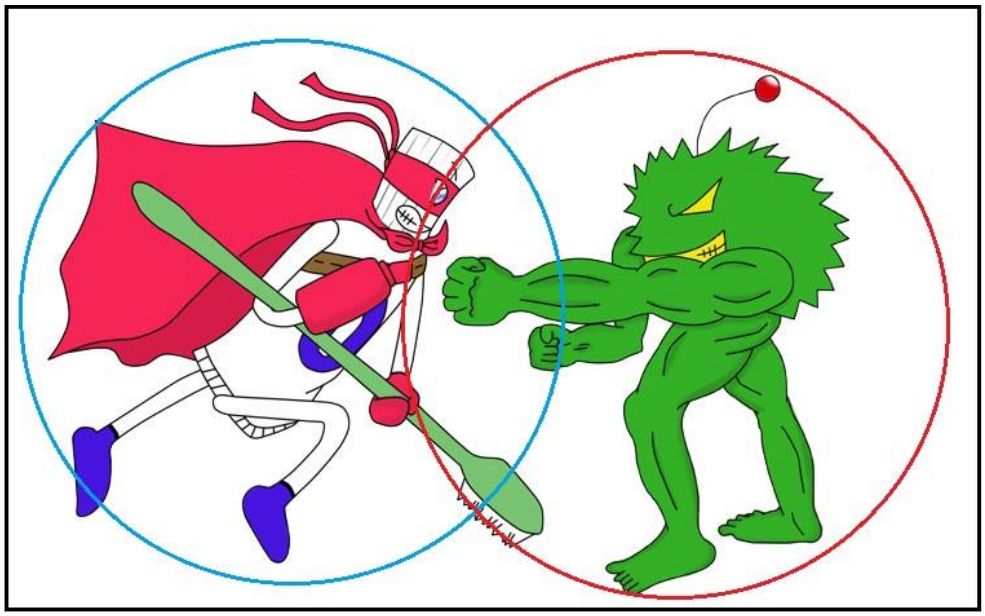
Adapun cara lain dari hasil modifikasi teknik ini yaitu *reduced size bounding box*. Dimana kotak batasnya diperkecil dengan mengurangi beberapa pixel dari ukuran *sprite* atau objek yang sebenarnya. Sehingga dengan cara ini pada saat melakukan *collision detection* akan lebih akurat karena ruang kosong antara *sprite* dan *bound* berkurang.

Kemudian cara lain dari modifikasi teknik ini adalah dengan cara menggunakan beberapa *bounding box* secara bersamaan untuk satu *sprite* atau objek. Contohnya untuk *sprite* berbentuk manusia, pertama *bounding box* digunakan untuk bagian kepala kemudian *bounding box* yang lainnya digunakan untuk badan dan kaki. Cara ini lebih akurat karena banyaknya *bounding box* yang digunakan dapat disesuaikan dengan bentuk gambar pada *sprite*. Namun semakin banyak *bounding box* yang digunakan akan semakin banyak pula waktu yang digunakan untuk *collision detection*.

2.2.3 *Circle Rectangle Collision Detection*

Untuk teknik ini memiliki cara yang hampir sama dengan yang sebelumnya hanya saja disini *bound* atau batas yang digunakan berbentuk lingkaran. Cara pengecekannya lebih mudah karena linggkaran memiliki jarak yg sama ke batas luarnya. Caranya yaitu dengan menghitung jarak kedua objek dan membandingkannya dengan jumlah radius masing-masing *bound* dari kedua objek, jika jaraknya lebih

kecil dari jumlah radius kedua objek maka terjadi tabrakan(*collision*) dan jika lebih besar maka tidak terjadi tabrakan. Cara sebelumnya yaitu rectangle collision detection lebih akurat daripada cara ini [13].



Gambar 2.2 Conrtoh rectangles collision detection

2.3 Kecerdasan Buatan / AI (*Artificial Intelligence*)

Sebagian kalangan mendefinisikan *Artifical Intelligence* sebagai kecerdasan buatan, kecerdasan artifisial, intelijensia artifisial, atau intelijensia buatan. Istilah *Artifical Intelligence* sudah sangat melekat di berbagai media ilmiah maupun non ilmiah [11].

Para Ahli mendefinisikan AI secara berbeda-beda tergantung pada sudut pandang mereka masing-masing. Ada yang focus 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 : *Natural Language Processing, Knowledge Representation, Automated Reasoning, Machine Learning, Computer Vision, 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.

Algoritma *A Star* (A*)

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 dua senarai : OPEN dan CLOSED. Terdapat tiga kondisi bagi setiap suksesor yang dibangkitkan, yaitu : sudah berada di OPEN, sudah berada di CLOSED, dan tidak berada di OPEN maupun CLOSED. Pada ketiga kondisi tersebut diberikan pengangan yang berbeda-beda. Berikut cara pengangan untuk tiga kondisi bagi setiap suksesor yang dibangkitkan :

1. Sudah berada di OPEN.

   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).

2. Sudah berada di CLOSED.

   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 per (best node).

3. Tidak berada di OPEN maupun CLOSED.

   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 [8].

Pseudo-code algoritma A* (Star) :

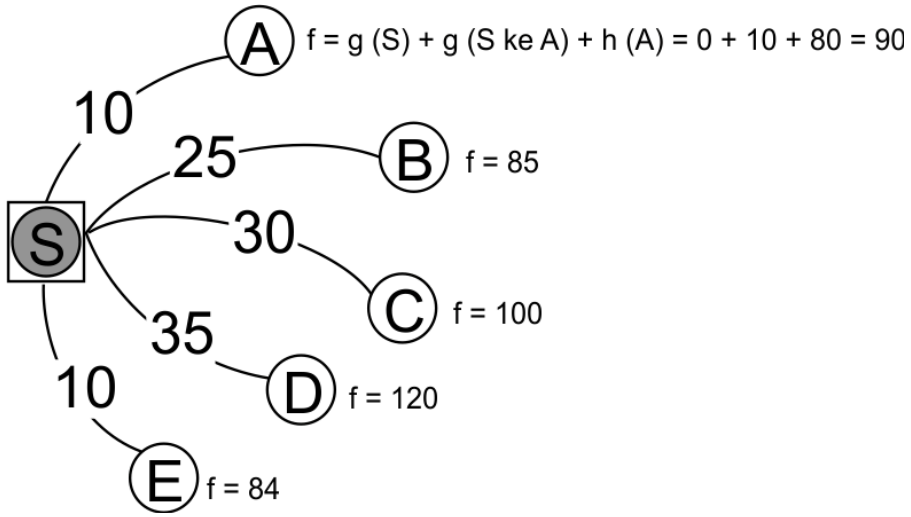
```
function A*(masalah) return solusi
  OPEN S
  CLOSED array kosong

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
```

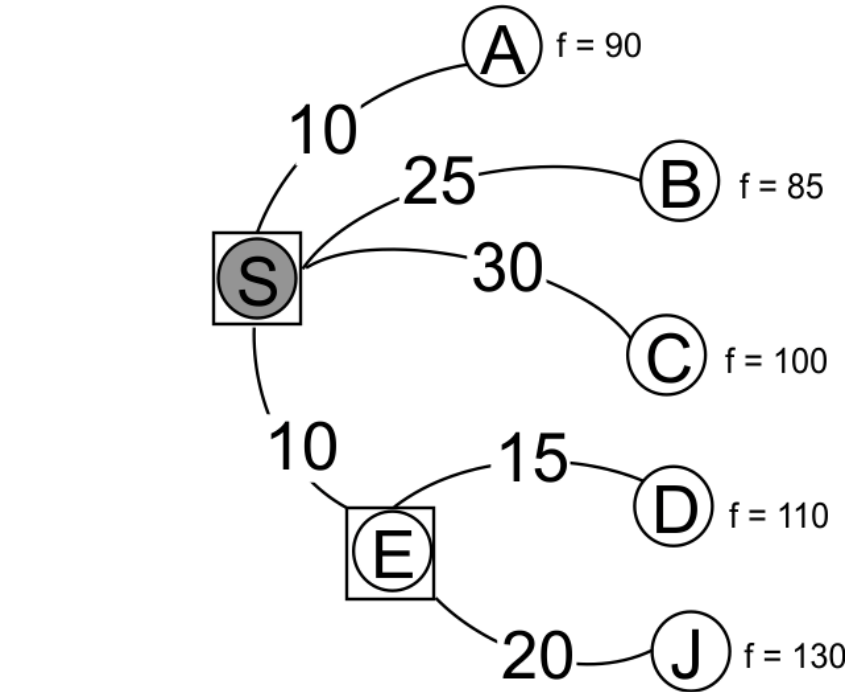
```
       belum diproses}
       OLD = simpul di OPEN yang sama dengan suksesor
       tersebut
Tambahkan OLD sebagai suksesor BestNode
       Buat pointer dari OLD ke BestNode
       Bandingkan nilai g (OLD) dengan g (suksesor)
       if g (OLD) lebih baik then
        Ubah parent OLD ke BestNode
        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
            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
```

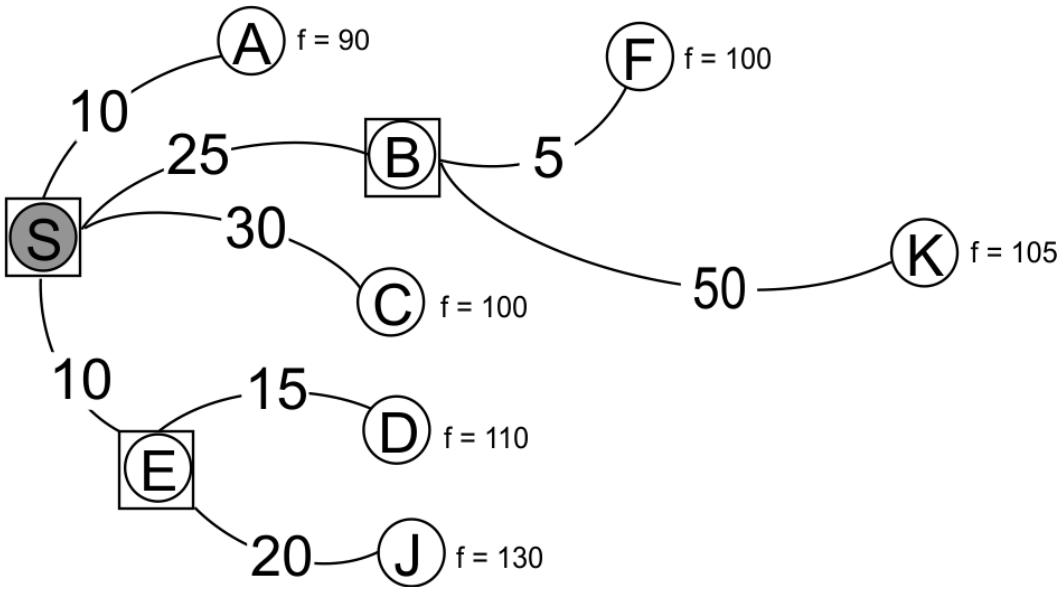
Untuk memperjelas pemahaman mengenai algoritma A* (A Star), maka berikut ini adalah langkah-langkah pencarian rute berdasarkan algoritma ini yang dapat dilihat pada gambar 2.3, gambar 2.4, gambar 2.5, gambar 2.6, gambar 2.7 dan gambar 2.8 :



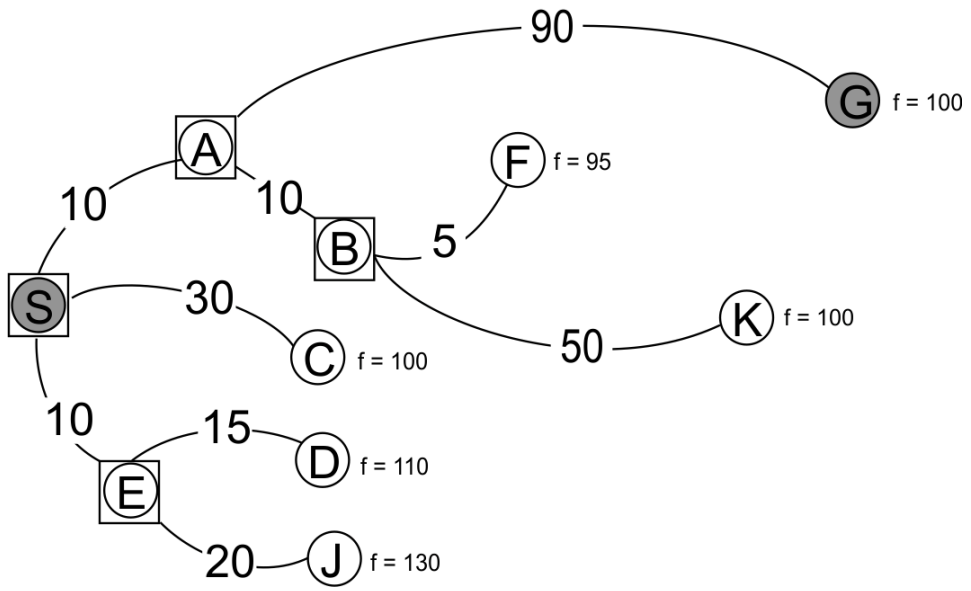
Gambar 2.3 Langkah 1 Pencarian Rute Dengan Algoritma A* (*A Star*)



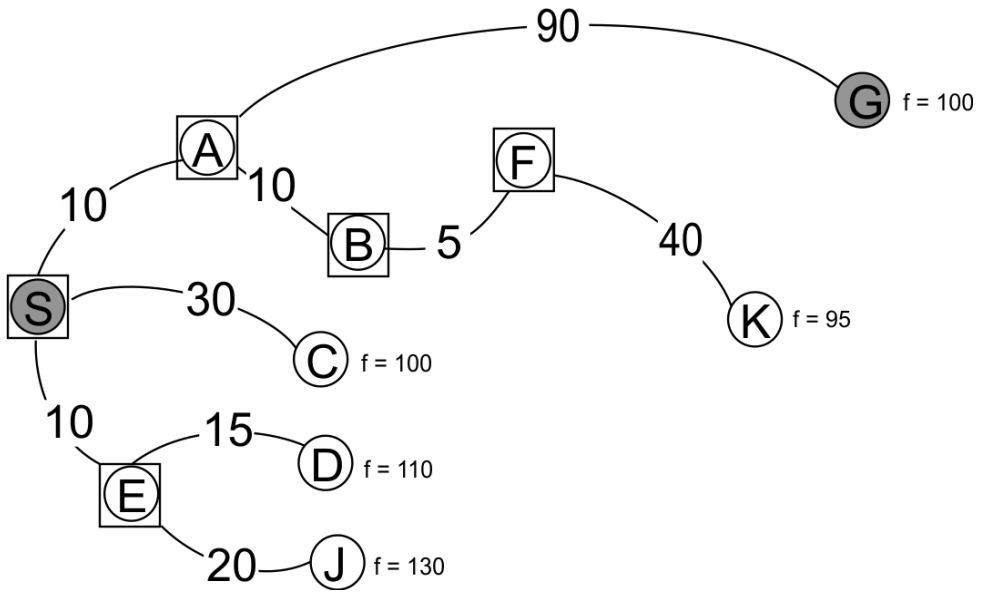
Gambar 2.4 Langkah 2 Pencarian Rute Dengan Algoritma A* (*A Star*)



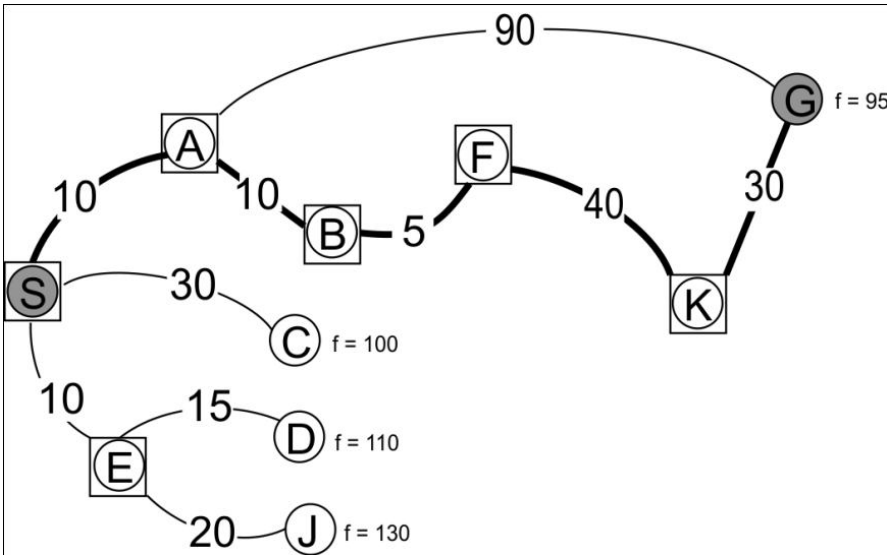
Gambar 2.5 Langkah 3 Pencarian Rute Dengan Algoritma A* (*A Star*)



Gambar 2.6 Langkah 4 Pencarian Rute Dengan Algoritma A* (*A Star*)



Gambar 2.7 Langkah 5 Pencarian Rute Dengan Algoritma A* (*A Star*)



Gambar 2.8 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 kotakmenyatakan 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].

2.4 Tools yang Digunakan

2.4.1 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 objekobjek yang saling berinteraksi satu sama lain. Objek adalah benda, baik benda yang berwujud nyata maupun benda yang tidak nyata (konsep) [5].

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 adalah :

1. OOP,
2. C++.
3. Visual C++.
4. Visual Basic.
5. Java [10].

2.4.2 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 [5].

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 sistem.
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.

2.4.3 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 37 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 [2].

2.4.4 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 40 game yang akan dibuat. Class actor adalah kelas yang berfungsi sebagai karakter pada game yang akan dibuat [7].