BAB 2
TINJAUAN PUSTAKA

2.1 Sampah

Menurut Undang-Undang Republik Indonesia Nomor 18 Tahun 2008 Tentang Pengelolaan Sampah, sampah adalah sisa kegiatan sehari-hari manusia dan atau proses alam yang berbentuk padat. Sampah spesifik adalah sampah yang karena sifat, konsentrasi, dan atau volumenya memerlukan pengelolaan khusus. Sedangkan pengertian sampah menurut beberapa ahli adalah sebagai berikut:

1. Basriyanta**:** sampah merupakan barang yang dianggap sudah tidak terpakai dan dibuang oleh pemilik atau pemakai sebelumnya, tetapi masih bisa dipakai atau dikelola dengan prosedur yang benar.
2. Ecolink (1996): sampah adalah suatu bahan yang terbuang atau dibuang dari sumber hasil aktivitas manusia maupun proses alam yang belum memiliki nilai ekonomis.
3. Setyo Purwendro: sampah merupakan bahan padat buangan dari kegiatan rumah tangga, pasar, perkantoran, rumah penginapan, hotel, rumah makan, industri ataupun aktivitas manusia lainnya sehingga dengan kata lain sampah merupakan hasil sampingan dari aktivitas manusia yang sudah tidak terpakai.

Sampah padat pada umumnya dapat dibagi menjadi dua, yaitu:

1. Sampah Organik

   Sampah organik terdiri dari bahan – bahan penyusun tumbuhan dan hewan yang diambil dari alam atau dihasilkan dari kegiatan pertanian, perikanan atau yang lain. Sampah ini dengan mudah diuraikan dalam proses alami. Sampah rumah tangga sebagian besar merupakan bahan organik, misalnya sampah dari dapur, sisa tepung, sayuran dan lain – lain.
2. Sampah Non-organik

   Sampah non-organik berasal dari sumber daya alam tak terbarui seperti mineral dan minyak bumi, atau dari proses industri. Beberapa dari bahan ini tidak terdapat di alam seperti plastik dan aluminium. Sebagian zat non-

organik secara keseluruhan tidak dapat diuraikan oleh alam, sedang sebagian lainnya hanya dapat diuraikan dalam waktu yang sangat lama. Sampah jenis ini pada tingkat rumah tangga, misalnya berupa botol, tas plastik atau lainnya.

Kertas, koran, dan karton merupakan pengecualian. Berdasarkan asalnya, kertas, koran, dan karton termasuk sampah organik. Tetapi karena kertas, karton, dan koran dapat didaur ulang seperti sampah non-organik lain misalnya gelas, kaleng, dan plastik maka dimasukkan ke dalam kelompok sampah non-organik.

Langkah bijak menyelesaikan sampah adalah *zero waste*. Untuk menuju kondisi *zero waste*, pada dasarnya menerapkan prinsip *reduce, reuse, recycle* (3R).

1. *Reduce*

   Upaya pertama menuju dunia tanpa sampah adalah mengurangi sampah yang "akan" dihasilkan. Sebagai konsumen, perlu untuk mulai mencegah atau meminimalisasi keberadaan sampah yang akan diakibatkan dari hasil belanja.

2. *Reuse*

   Pilihlah barang – barang yang bisa dipakai kembali. Hindari pemakaian barang – barang yang *disposable* (sekali pakai, buang). Hal ini dapat memperpanjang waktu pemakaian barang sebelum ia menjadi sampah.

3. *Recycle*

   Merupakan upaya untuk mendaur ulang sampah menjadi barang yang lebih bernilai misalnya botol plastik menjadi pot bunga, kertas koran menjadi kertas daur ulang dan lainnya.

2.2 *Game*

Game merupakan salah satu media hiburan yang paling popular untuk semua kalangan usia. Sejak pertama kali ditemukan sampai saat sekarang, teknologi *game* telah mengalami kemajuan yang terbilang pesat. Hal ini ditandai dengan berkembangnya jenis *game*, produk, alat dan jenis interaksi *game* dengan penggunaan yang semakin beragam bentuknya.

2.2.1 Pengertian *Game*

Game merupakan sebuah bentuk seni dimana penggunanya disebut dengan pemain (*player*), diharuskan membuat keputusan-keputusan dengan tujuan mengelola sumber daya yang diperoleh dari kesempatan-kesempatan bermain (*token*) miliknya untuk mencapai tujuan tertentu. Video *game* adalah bentuk *game* yang interaksi umumnya melibatkan media video dan audio.

Menurut [5] terdapat dua pengertian *game*. Pertama, *game* (permainan) adalah sebuah aktivitas bermain yang murni mencari kesenangan tanpa mencari menang atau kalah. Kedua, permainan diartikan sebagai aktifitas bermain yang dilakukan dalam rangka mencari kesenangan dan kepuasan, namun ditandai pencarian menang – kalah.

Berdasarkan pengertian – pengertian di atas, *game* merupakan sebuah aktivitas dalam rangka mencari kesenangan dimana penggunanya diharuskan membuat keputusan-keputusan untuk mencapai tujuan tertentu dengan ditandai menang – kalah.

2.2.2 Jenis-jenis *Game* Berdasarkan *Platform*

Berikut ini beberapa jenis *game* berdasarkan cara pembuatannya, cara pemasarannya dan mesin yang menjalankannya. Jenis-jenis *game* tersebut menurut [1] adalah:

1. *Game* PC

   Game PC adalah *game* yang dimainkan pada PC (*Personal Computer*) yang memiliki kelebihan yaitu tampilan antarmuka yang baik untuk *input* maupun *output*. *Output* visual berkualitas tinggi karena layar komputer biasanya memiliki resolusi yang jauh lebih tinggi dibandingkan dengan layar televisi biasa.Kekurangannya adalah spesifikasi komputer yang sangat bervariasi antar satu komputer dengan komputer yang lainnya menyebabkan beberapa *game* dapat ditampilkan dengan baik pada satu komputer tetapi tidak berjalan dengan baik pada komputer yang lainnya.

2. *Game Console*

   Game console adalah *game* yang dijalankan pada suatu mesin spesifik

yang biasanya tersedia di rumah seperti Xbox,Nintendo Wii dan lain-lain.

3. *Game Arcade*

   Game arcade adalah *game* yang dijalankan pada mesin dengan *input* dan *output* audio visual yang telah terintegrasi dan tersedia ditempat-tempat umum.

4. *Game Online*

   Game online adalah *game* yang hanya dapat dumainkan secara *online* melaui LAN atau internet.

2.2.3 Genre *Game*

Berdasarkan *genre* permainannya, *game* dapat dibagi menjadi beberapa *genre* yaitu:

1. *Action*

   Action game adalah *game* dimana kebanyakan dari tantangan yang disajikan merupakan dari tes *physical skill* dan koordinasi pemain. Salah satu *sub-genre action game* adalah *shooter game*, baik yang 2D ataupun 3D seperti *First Person Shooter (FPS).*

2. *Strategy*

   Strategy game menantang pemain untuk mencapai kemenangan dengan perencanaan, khususnya melalui perencanaan serangkaian tindakan yang dilakukan melawan satu lawan atau lebih. Kemenangan diraih dengan perencanaan matang dan pengambilan keputusan optimal.

3. *Role Playing Game* (RPG)

   RPG adalah *game* dimana pemain mengontrol satu atau lebih karakter yang biasanya di desain oleh pemain itu sendiri, dan memandu mereka melewati berbagai rintangan yang diatur oleh komputer. Perkembangan karakter dalam hal kekuatan dan kemampuannya adalah kunci dari *game* jenis ini.

4. *Sports*

   Sports game mensimulasikan berbagai aspek dari olahraga atletik nyata ataupun imajiner, apakah itu memainkan pertandingan, me-*manage* tim

dan karir, atau keduanya. Salah satu contoh *game* jenis ini adalah *Pro Evolution Soccer 2012* (PES 2012), dimana pemain bisa memainkan pertandingan, menjadi manajer tim, maupun menjadi pemain dan mengembangkan karirnya sendiri.

5. *Vehicle Simulation*

   Vehicle simulation membuat *feeling* mengendarai kendaraan, baik kendaraan nyata maupun kendaraan imajiner. Performa dan karakteristik kasar mesin harus menyerupai kenyataan, kecuali jika yang didesain adalah kendaraan imajiner.

6. *Construction and Management Simulation*

   CMS game adalah *game* tentang proses. Tujuan pemain bukan untuk mengalahkan musuh, tetapi membangun sesuatu dengan konteks proses yang sedang berjalan. Semakin pemain mengerti dan mengontrol proses, semakin sukses sesuatu yang ia bangun. *Game* seperti ini biasanya menyediakan dua jenis permainan, yaitu mode bebas dimana pemain bebas membangun sesuatu, dan mode misi dimana terdapat skenario hal apa yang harus dibangun oleh pemain.

7. *Adventures*

   Adventure game adalah cerita interaktif tentang karakter protagonist yang dimainkan oleh pemain. Penyampaian cerita dan eksplorasi adalah elemen inti dari *game* ini. Penyelesaian teka-teki dan tantangan konseptual adalah bagian besar dari permainan.

8. *Artificial Life and Puzzle Game*

   Artificial Life game adalah *game* yang membuat tiruan dari kehidupan sebenarnya. Biasanya ada dua jenis *game* ini, tiruan kehidupan manusia, contohnya The SIMS, dan tiruan kehidupan binatang, contohnya Tamagochi.

9. *Online Game*

   Istilah *online game* disini mengacu kepada *multiplayer game* dimana mesin dari para pemain terhubung dengan jaringan.

2.2.4 Kategori *Game* Lainnya

Selain berdasarkan genre permainannya terdapat pula kategori-kategori *game* lainnya, yaitu:

1. *Multiplayer Online*

   Game yang dapat dimainkan secara bersamaan oleh lebih dari 2 orang (bahkan dapat mencapai puluhan ribu orang dalam satu waktu) membuat pemain dapat bermain bersama dalam satu dunia virtual dari sekedar *chatting* hingga membunuh naga bersama teman yang entah bermain di mana. Umumnya permainan tipe ini dimainkan di PC dan bertema RPG, walau ada juga yang bertema musik atau *action*.

2. *Casual Games*

   Sesuai namanya, *game casual* itu tidak kompleks, mainnya rileks dan sangat mudah untuk dipelajari (bahkan cenderung langsung bisa dimainkan). Jenis ini biasanya memerlukan spesifikasi komputer yang standar. Genre permainannya biasanya *puzzle* atau *action* sederhana dan umumnya dapat dimainkan hanya menggunakan *mouse* (biasanya *game* lain menggunakan banyak tombol tergantung *game*-nya). Contohnya: *Dinner Dash, Zuma, Feeding Frenzy*.

   Fitur dasar pada *game casual*, yaitu:

   1) *Gameplay* sangat sederhana, bisa dimainkan menggunakan *mouse, keyboard* atau keypad ponsel.
   2) Dimainkan dalam waktu singkat. Biasanya dibawah 10 menit. Bisa dimainkan saat istirahat kerja atau pada transportasi umum.
   3) Kemampuan yang cepat dalam mencapai tahap akhir tanpa perlu *save game*.
   4) Kadang digunakan untuk model bisnis atau periklanan.
   5) Cocok dalam *genre* apapun.
   6) Dapat dimainkan oleh siapa saja, anak – anak sampai orang dewasa.
   7) Dapat dimainkan dengan atau tanpa pengalaman dalam bermain *game.*

3. *Edugames*

   Video game jenis ini dibuat dengan tujuan spesifik sebagai alat pendidikan, entah untuk belajar mengenal warna untuk balita, mengenal huruf dan angka, matematika, sampai belajar bahasa asing. *Developer* yang membuatnya, harus memperhitungkan berbagai hal agar game ini benar-benar dapat mendidik, menambah pengetahuan dan meningkatkan ketrampilan yang memainkannya. Target segmentasi pemain harus pula disesuaikan dengan tingkat kesulitan dan desain visual ataupun animasinya.

4. *Advergames*

   Jenis *game* yang biasanya mudah dimainkan ini mengusung dan menampilkan produk atau *brand* secara gamblang maupun tersembunyi. Di era tumbuhnya media-media baru berteknologi tinggi sekarang ini, dunia periklanan memang sudah tidak lagi terbatas pada TV, koran, majalah, *billboard* dan radio, *video game* sekarang telah menjadi sarana beriklan atau membangun *brand-awareness* yang efektif. Baik melalui internet maupun di mainkan di *event-event* mereka, *edugames* terasa semakin dibutuhkan untuk menjaring calon konsumen bagi produk yang menggunakan *advergames* ini.

2.2.5 Elemen Pada *Game*

Elemen – elemen yang ada dalam sebuah *game* biasanya terdiri dari:

1. Desain *game*

   Desain adalah langkah awal untuk membuat semua elemen *game*. Desain *game* dibuat semenarik mungkin agar pemain tidak cepat bosan yang dampaknya membuat *game* tersebut cepat ditinggalkan.

2. Pemrograman *game*

   Pemrograman *game* sebagian besar digunakan untuk mengontrol gerakan objek di layar. Selain itu, pemrograman *game* juga digunakan untuk pemrograman suara, *input device*, deteksi keadaan lain – lain.

3. Grafis *game*

   Grafis *game* memegang peranan penting dalam pembuatan tampilan. Tampilan haruslah dibuat semenarik mungkin, sehingga dengan melihatnya saja *end user* langsung tertarik untuk memainkannya.

4. Musik dan *sound*

   Musik dan *sound* dalam suatu *game* merupakan suatu hal yang wajib dikarenakan dalam suatu *game* akan lebih terasa efek emosi dari *game* tersebut, sehingga akan membuat kesan *game* tersebut lebih terasa nyata.

2.3 *Artificial Intelligence* (AI)

Definisi kecerdasan buatan atau *artificial intelligent* 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. [4]:

1. ***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 aksi yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka *agent* dapat melakukan aksi berdasarkan kesimpulan tersebut.

2.4 Kecerdasan Buatan Pada *Game*

Kecerdasan buatan atau AI merupakan kegiatan membuat komputer agar dapat berpikir dan mengerjakan kegiatan yang dapat dilakukan oleh manusia maupun binatang.

Saat ini dapat ditemukan program komputer yang memiliki kemampuan menangani masalah seperti aritmatik, sorting, searching. Bahkan komputer juga dapat bermain beberapa board *game* seperti catur dan reversi lebih baik daripada manusia.

Namun, masih banyak hal yang tidak dapat dilakukan dengan baik oleh komputer. Seperti, mengenali wajah, berbicara bahasa manusia, menentukan sendiri apa yang harus dilakukan, dan bertingkah kreatif. Hal itu semua merupakan domain dari AI untuk mencoba menentukan algoritma apa yang dibutuhkan untuk memenuhi kebutuhan diatas.

Dalam bidang akademik, beberapa peniliti AI termotivasi oleh filosofi, yaitu memahami alam pikiran dan alam kecerdasan dan membangun program untuk memodelkan bagaimana proses berpikir. Beberapa juga termotivasi oleh *psychology*, bertujuan untuk memahami mekanisme otak manusia dan proses mental. Dan lainya termotivasi oleh engineering, dengan tujuan membangun algoritma untuk melakukan kegiatan seperti manusia atau hewan.

Dalam pembangunan *game*, umumnya akan cenderung hanya pada sisi engineering yang bertujuan membangun algoritma yang dapat membuat *game* karakter mengerjakan kegiatan seperti yang dilakukan manusia atau binatang.

2.5 Metode – Metode Pencarian

Terdapat banyak metode pencarian yang telah diusulkan. Semua metode yang ada dapat dibedakan kedalam dua jenis: pencarian buta/tanpa informasi (*blind* atau *un-informed search*) dan pencarian heuristik/dengan informasi (*heuristic* atau *informed search*).

2.5.1.1 *Breadth-First Search* (BFS)

BFS termasuk kedalam jenis *blind/un-informed search*. Pencarian dilakukan pada semua simpul pada setiap level secara berurutan dari kiri ke kanan. Jika pada satu level belum ditemukan solusi, maka pencarian dilanjutkan pada level berikutnya. Demikian seterusnya sampai ditemukan solusi. Dengan cara seperti ini, BFS menjamin ditemukannya solusi (jika solusinya memang ada) dan solusi yang ditemukan pasti yang paling baik. Dengan kata lain BFS adalah *complete* dan *optimal*. Tetapi BFS harus menyimpan semua simpul yang pernah dibangkitkan. Hal ini harus dilakukan agar BFS dapat melakukan penelusuran simpul – simpul sampai di level bawah. [4]

2.6 Algoritma A* *(A Star)*

Algoritma A* digunakan dalam penentuan jalur terpendek, pada *game* yang dibangun algoritma ini diterapkan pada pergerakan rintangan mobil yang menghindari objek sampah yang ada di depannya untuk sampai ke tujuan.

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: $f(n) = g(n) + h(n)$ (2.1). Dengan perhitungan biaya seperti ini, algoritma A* adalah *complete* dan *optimal*.

Terdapat tiga kondisi bagi setiap suksesor yang dibangkitkan, yaitu:

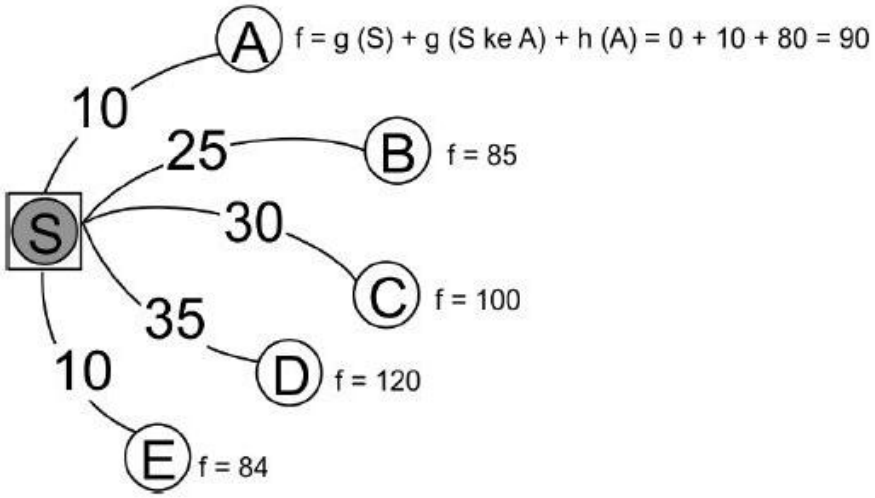
1. Sudah berada di *OPEN*
2. Sudah berada di *CLOSE*, dan
3. Tidak berada di *OPEN* maupun *CLOSED*.

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*).

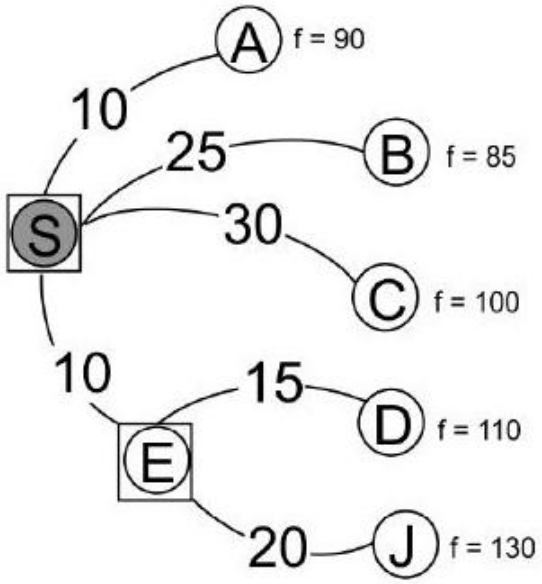
Jika suksesor sudah pernah berada di *CLOSED*, maka akan dilakukan pengecekan apakah perlu pengubahan parent atau tidak. Jika ya, maka dilakukan perbaruan nilai *g* dan *f* pada suksesor tersebut serta pada semua "anak cucunya" yang sudah pernah berada di *OPEN*. Dengan perbaruan ini, maka anak cucunya 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 $f = g + h$.

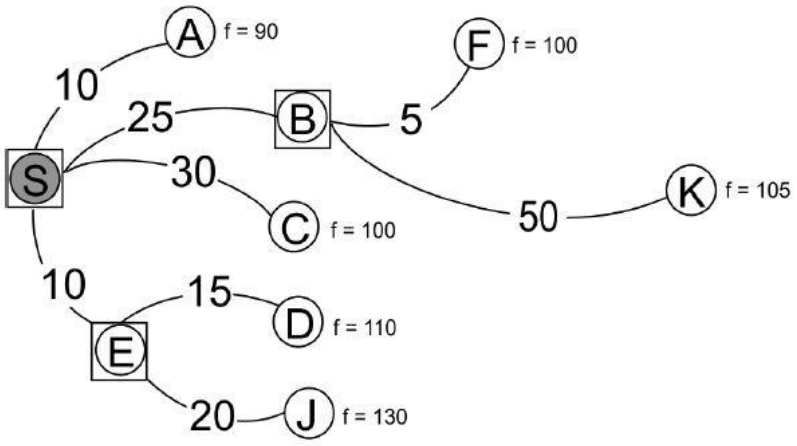
Untuk memperjelas pemahaman tentang algoritma A*, berikut adalah langkah – langkah pencarian rute berdasarkan berdasarkan algoritma ini:



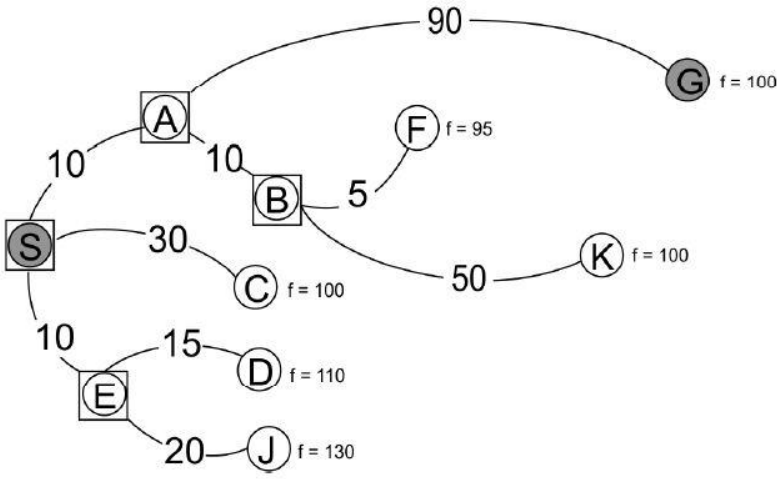
Gambar 2.1 Langkah 1 pencarian rute dengan algoritma A*



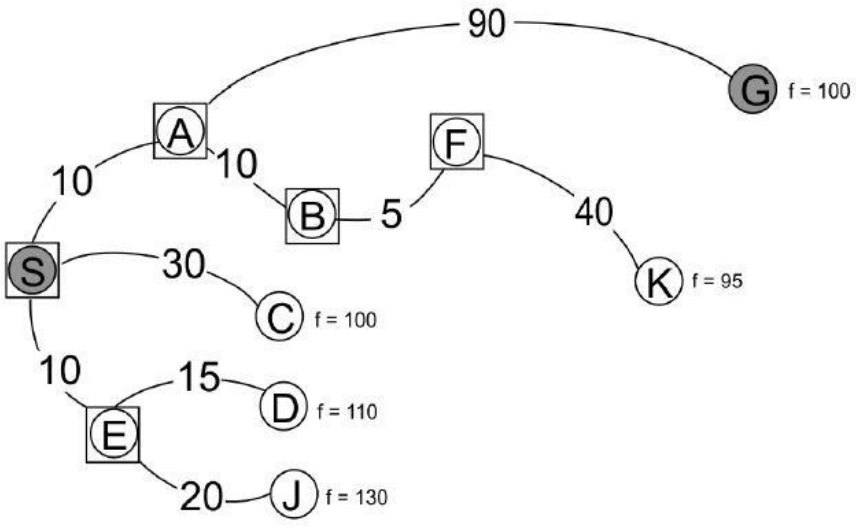
Gambar 2.2 Langkah 2 pencarian rute dengan algoritma A*



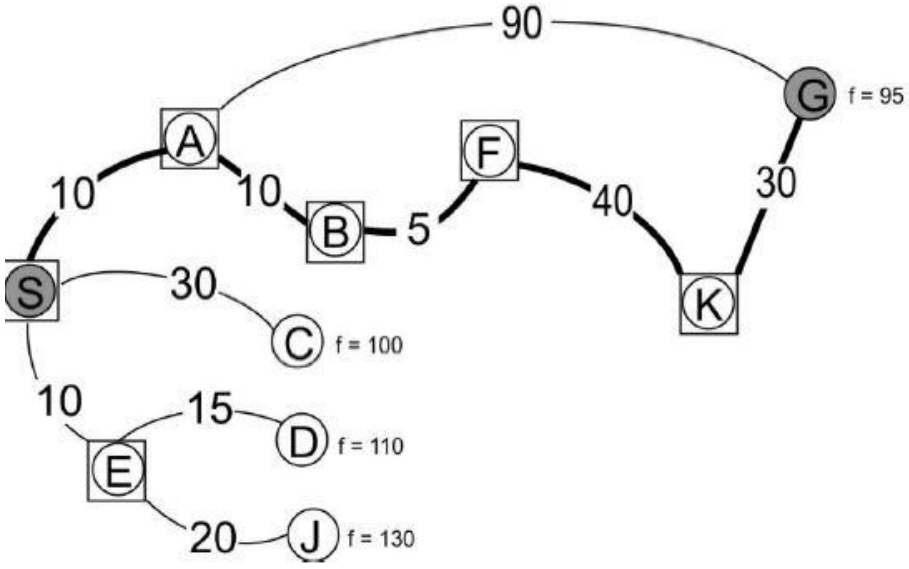
Gambar 2.3 Langkah 3 pencarian rute dengan algoritma A*



Gambar 2.4 Langkah 4 pencarian rute dengan algoritma A*



Gambar 2.5 Langkah 5 pencarian rute dengan algoritma A*



Gambar 2.6 Langkah 6 pencarian rute dengan algoritma A*

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*.

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 ketiga 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* adalah optimal. Tanpa ada batasan waktu dan memori, A* adalah *complete* (selalu menemukan solusi jika solusinya ada). Pada kasus tersebut A* 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* akan menghadapi masalah waktu proses dan memori yang dibutuhkan. Untuk menyelesaikan kedua masalah tersebut, telah diusulkan berbagai variasi A* dengan karakteristik yang sesuai untuk permasalahan tertentu [4].

2.7 Pemrograman Berorientasi Objek

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 [6]:

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*.

2.7.1 Analisis dan Desain Berorientasi Objek *(Object-Oriented Analysis and Design Process)*

Pemrograman berorientasi objek bekerja dengan baik ketika dibarengi dengan *object-oriented analysis and design process (OOAD).* Jika membuat program berorientasi objek tanpa OOAD, ibarat membangun rumah tanpa terlebih dahulu menganalisa apa saja yang dibutuhkan oleh rumah itu, tanpa perencanaan tanpa *blueprint*, tanpa menganalisis ruangan apa saja yang diperlukan, berapa besar rumah yang akan dibangun dan sebagainya.

2.7.2 Objek *(Object)*

Orientasi objek merupakan teknik dalam menyelesaikan masalah yang kerap muncul dalam pengembangan perangkat lunak. Teknik ini merupakan titik kulminasi dalam menemukan cara yang efektif dalam membangun sistem dan menjadi metode yang paling banyak dipakai oleh para pengembang perangkat lunak saat ini. Orientasi objek merupakan teknik pemodelan sistem riil yang berbasis objek. Inti dari konsep ini adalah objek yang merupakan model dari sistem nyata.

Objek adalah entitas yang memiliki atribut, karakter dan kadangkala disertai kondisi. Objek merepresentasikan sesuatu sistem nyata seperti siswa, sistem kontrol permukaan sayap pesawat, sensor atau mesin. Objek juga merepresentasikan sesuatu dalam bentuk konsep seperti nasabah bank, merek dagang, pernikahan atau sekedar *listing*. Bahkan bisa juga mengatakan visualisasi seperti, bentuk huruf, histogram, poligon, garis atau lingkaran. Semuanya memiliki fitur atribut (untuk data), *behavior* (*operation* atau *method*), keadaan (memori), identitas dan tanggung jawab. Proses menjabarkan sistem nyata menjadi objek dinamakan abstraksi *(abstraction)*. Abstraksi mengeliminir aspek yang tidak perlu dalam suatu objek.

2.7.3 Kelas *(Class)*

Kelas adalah penggambaran satu set objek yang memiliki atribut dan *behaviour* yang sama. Kelas mirip tipe data pada pemrograman non objek, tapi

lebih komprehensif karena terdapat struktur sekaligus karakteristiknya. Programmer dapat membentuk kelas baru yang lebih spesifik dari kelas *general*-nya. Kelas dan objek merupakan jantung dari pemrograman berorientasi objek. Untuk menghasilkan program jenis ini sangat penting untuk selalu berfikir dalam bentuk objek.

2.7.4 Pembungkusan *(Encapsulation)*

Pembungkusan sebagai penggabungan potongan-potongan informasi dan perilaku-perilaku spesifik yang bekerja pada informasi tersebut, kemudian mengemasnya menjadi apa yang disebut sebagai objek. Dalam perbankan dikenal objek rekening yang memiliki perilaku-perilaku misalnya buka, tutup, penarikan, penyimpanan, ubah nama, ubah alamat, dan sebagainya. Akibatnya, perubahan-perubahan pada sistem perbankan yang berkaitan dengan rekening-rekening dapat secara sederhana diimplementasikan satu kali saja pada objek rekening. Keuntungan lainnya adalah membatasi efek-efek perubahan pada sistem. Misalnya, saat manajemen bank menentukan jika seseorang memiliki rekening pinjaman di bank yang bersangkutan, rekening pinjaman itu harus dapat juga digunakan sebagai sarana bagi penarikan rekening.

2.7.5 Pewarisan *(Inheritance)* dan Generalisasi/Spesialisasi

Konsep dimana metode dan atau atribut yang ditentukan di dalam sebuah objek kelas dapat diwariskan atau digunakan lagi atau digunakan lagi oleh objek kelas lainnya. Sedangkan generalisasi/spesialisasi merupakan teknik dimana atribut dan perilaku yang umum pada beberapa tipe kelas objek, dikelompokkan (atau diabstraksi) ke dalam kelasnya sendiri (dinamakan *supertype*). Atribut dan metode kelas objek *supertype* kemudian diwariskan oleh kelas objek tersebut (dinamakan *subtype*).

2.7.6 *Polimorfisme*

Polimorfisme berarti suatu fungsionalitas yang diimplementasikan dengan berbagai cara yang berbeda. Pada terminologi berorientasi objek, ini berarti dapat

memiliki berbagai implementasi untuk sebagian fungsionalitas tertentu. Sebagai contoh, misalkan pengembang akan mengembangkan sistem berbasis grafis. Saat pengguna mau menggambar sesuatu, misalnya garis atau lingkaran, sistem akan memunculkan perintah gambar. Sistem akan mengenali berbagai bentuk gambar, masing-masing dengan perilakunya sendiri-sendiri. Manfaat dari *polimorfisme* adalah kemudahan pemeliharaannya. Jika perlu menambahkan gambar baru (misalnya segitiga) maka cukup menambahkan fungsi baru (fungsi menggambar segitiga) sedangkan fungsi umumnya (fungsi gambar) tidak mengalami perubahan.

2.8 *UML (Unified Modeling Language)*

UML adalah keluarga notasi grafis yang didukung oleh meta-model tunggal, yang membantu pendeskripsian dan desain sistem perangkat lunak, khususnya sistem yang dibangun menggunakan pemrograman berorientasi objek. Definisi ini merupakan definisi sederhana. Pada kenyataannya, pendapat orang-orang tentang UML berbeda satu sama lain. Hal ini dikarenakan oleh sejarahnya sendiri dan oleh perbedaan persepsi tentang apa yang membuat sebuah proses rancang bangun perangkat lunak efektif.

UML merupakan standar yang relatif terbuka yang dikontrol oleh O*bject Management Group (OMG),* sebuah konsorsium terbuka yang terdiri dari banyak perusahaan. OMG dibentuk untuk membuat standar-standar yang mendukung interoperabilitas, khususnya interoperabilitas sistem berorientasi objek. OMG mungkin lebih dikenal dengan standar-standar *COBRA (Common Object Request Broker Architecture).*

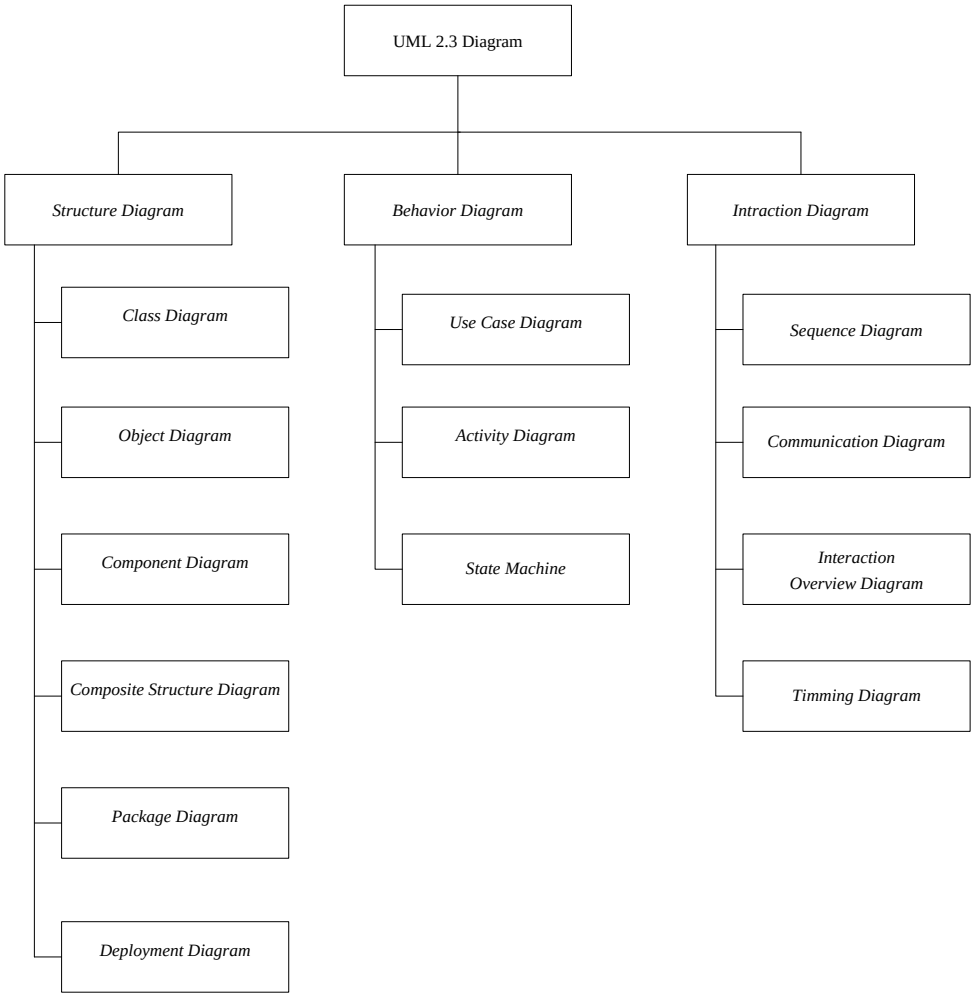
UML lahir dari penggabungan banyak bahasa pemodelan grafis berorientasi objek yang berkembang pesat pada akhir 1980-an dan awal 1990-an. Sejak kehadirannya pada tahun 1997, UML menggantikan menara Babel yang telah menjadi sejarah. UML merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

Bagian-bagian utama dari UML adalah *view*, *diagram*, *model element*, dan *general mechanism* [7]. Diagram berbentuk grafik yang menunjukkan simbol

elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu view tertentu dan ketika digambarkan biasanya.

2.9 Diagram UML

Pada UML 2.3 terdiri dari 13 macam diagram yang dikelompokkan dalam 3 kategori. Pembagian kategori dan macam-macam diagram tersebut dapat dilihat pada gambar Gambar 2.7.



Gambar 2.7 Diagram UML [10]

Berikut ini penjelasan singkat dari pembagian kategori tersebut.

1. *Structure diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan suatu struktur statis dari sistem yang dimodelkan.

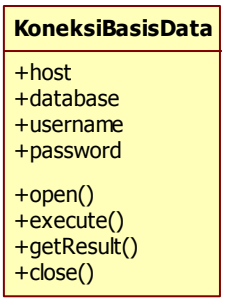
2. *Behavior diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan kelakuan sistem atau rangkain perubahan yang terjadi pada sebuah sistem.
3. *Interaction diagrams* yaitu kumpulan diagram yang digunakan untuk menggambarkan interaksi sistem dengan sistem lain maupun interaksi antarsubsistem pada suatu sistem.

2.9.1 Diagram Kelas *(Class Diagram)*

Diagram kelas menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi.

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas.
2. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

Diagram kelas dibuat agar pembuat program atau *programmer* membuat kelas-kelas sesuai rancangan di dalam diagram kelas agar antara dokumentasi perancangan dan perangkat lunak sinkron. Berikut adalah contoh dari diagram kelas.



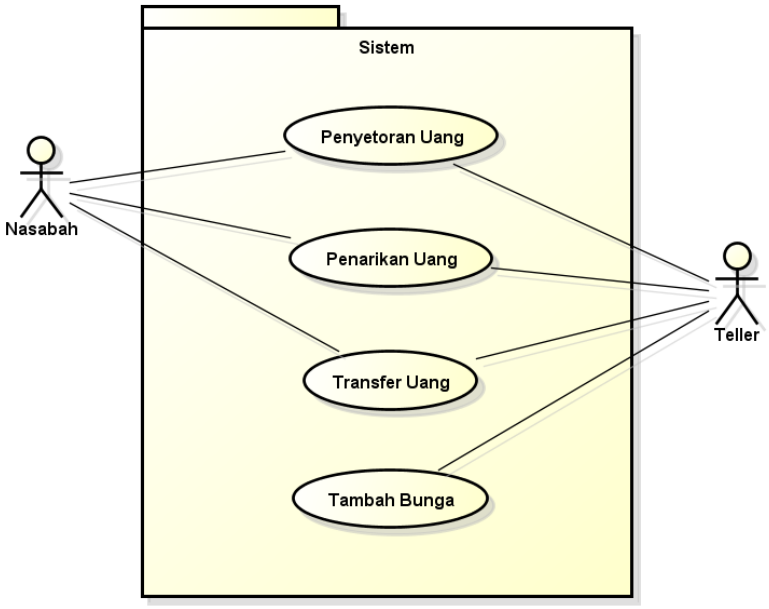
Gambar 2.8 Contoh *Class Diagram* [10]

2.9.2 Diagram *Use Case*

Diagram *use case* merupakan pemodelan untuk kelakuan *(behaviour)* sistem informasi yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam

sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case*.

1. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
2. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.

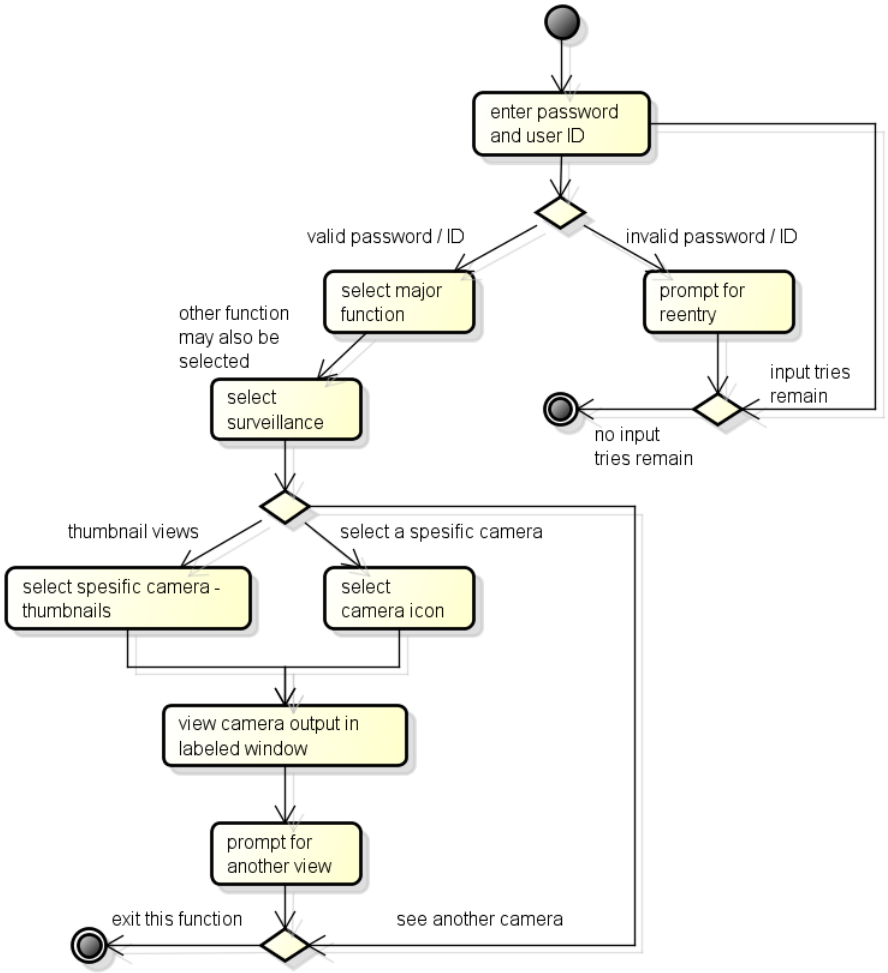


Gambar 2.9 Contoh *Use Case Diagram* [3]

2.9.3 Diagram Aktivitas *(Activity Diagram)*

Diagram aktivitas menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak. Yang perlu diperhatikan disini adalah bahwa diagram aktivitas menggambarkan aktivitas sistem bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem. Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut:

1. Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan atau pengelompokkan tampilan dari sistem/*user interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.
3. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.
4. Rancangan menu yang ditampilkan pada perangkat lunak.

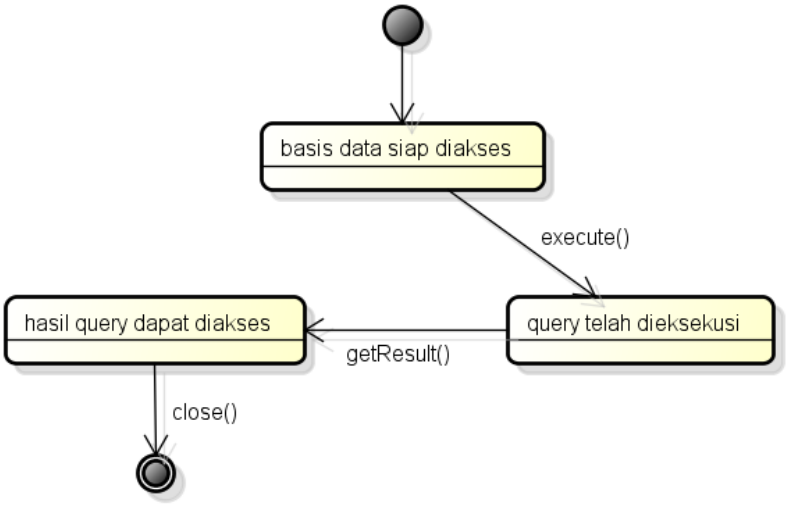


Gambar 2.10 Contoh *Activity Diagram* [13]

2.9.4 Diagram *State Machine*

State machine diagram atau *statechart diagram* atau dalam bahasa Indonesia disebut diagram mesin status atau sering juga disebut diagram status

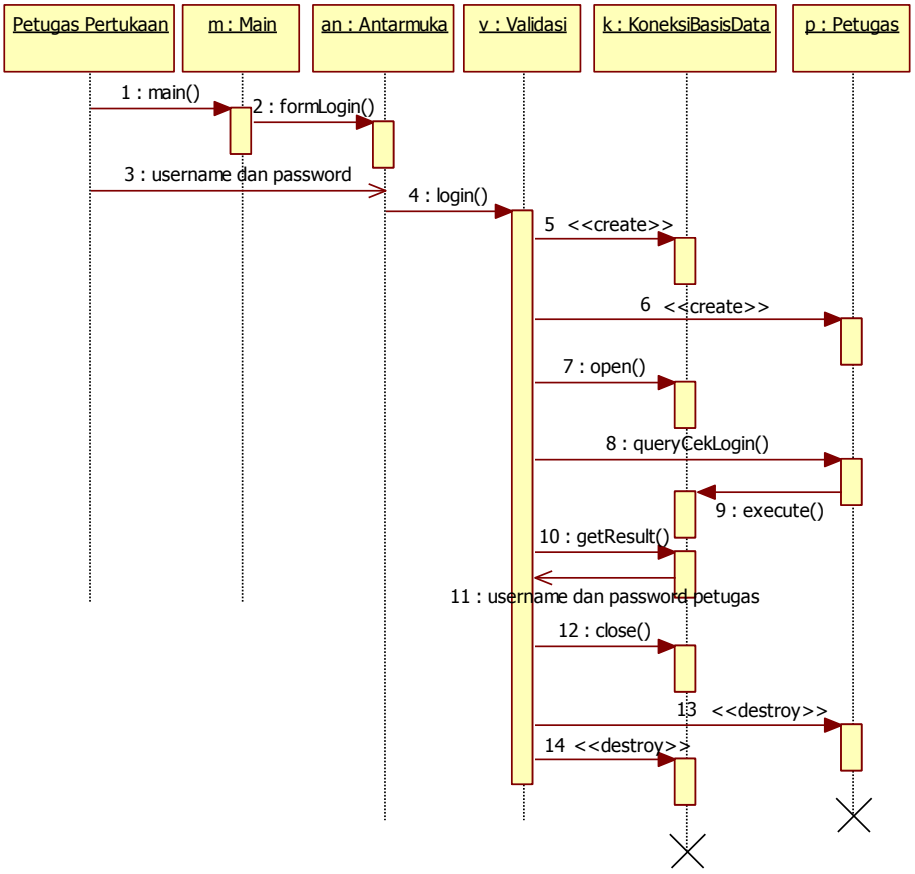
digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem atau objek. Jika diagram sekuen digunakan untuk interaksi antar objek maka diagram status digunakan untuk interaksi di dalam sebuah objek. Perubahan tersebut digambarkan dalam suatu graf berarah.



Gambar 2.11 Contoh *Statechart Diagram* [10]

2.9.5 Diagram Sekuen *(Sequence Diagram)*

Diagram sekuen menggambarkan kelakuan pada objek *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambar diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Membuat diagram sekuen juga dibutuhkan untuk melihat skenario yang ada pada *use case*. Banyaknya diagram sekuen yang harus digambar adalah minimal sebanyak pendefinisian *use case* yang memiliki proses sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram sekuen sehingga semakin banyak *use case* yang didefinisikan maka diagram sekuen yang harus dibuat juga semakin banyak.



Gambar 2.12 Contoh *Sequence Diagram* [10]

2.10 Unity

Unity Game Engine adalah *software* atau *game engine* yang digunakan untuk membuat *video game* berbasis dua atau tiga dimensi dan dapat digunakan secara gratis. Selain untuk membuat *game*, *unity* juga dapat digunakan untuk membuat konten yang interaktif lainnya seperti, visual arsitektur dan *real-time* 3D animasi.

Unity adalah sebuah sebuah *tool* yang terintegrasi untuk membuat *game*, arsitektur bangunan dan simulasi. *Unity* bisa digunakan untuk *game*s PC dan *games online*. Untuk *games online* diperlukan sebuah plugin, yaitu *Unity Web Player*, yang sama halnya dengan *flash player* pada *browser*. Bahasa pemrograman yang digunakan bermacam-macam, mulai dari *javascript*, C#, dan *boo*.

Unity tidak bisa melakukan desain atau *modelling*, dikarenakan *unity* bukan

merupakan *tools* untuk mendesain. Banyak hal yang bisa di lakukan di unity, ada fitur *audio reverb zone , particle effect , sky box* untuk menambahkan langit, dan masih banyak lagi, dan juga bisa langsung *edit texture* dari *editor* seperti *photoshop* dll.

Features (Scripting) di dalam *unity* adalah sebagai berikut (Unity Technologies, 2013):

1. Mendukung 3 bahasa pemrograman, *JavaScript*, C#, dan *Boo*.
2. *Flexible and EasyMoving, rotating,* dan *scaling objects* hanya perlu sebaris kode. Begitu juga dengan *duplicating, removing,* dan *changing properties.*
3. *Multi Platform Game* bisa di deploy di PC, Mac, Wii, iPhone, iPad dan *browser, android*.
4. *Visual Properties Variables* yang di definisikan dengan *scripts* ditampilkan pada editor. Bisa digeser, di *drag* and *drop*, bisa memilih warna dengan *color picker.*
5. Berbasis .NET, penjalanan program dilakukan dengan *Open Source* .NET *platform, Mono.*

2.11 Bahasa Pemrograman C#

C# merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. C# adalah Java versi Microsoft, sebuah bahasa multi flatform yang didesain untuk bisa berjalan di berbagai mesin. C# adalah pemrograman berorientasi Object (OOP). C# memiliki kekuatan bahasa C++ dan portabilitas seperti Java. Fitur-fitur yang diambilnya dari bahasa C++ dan Java adalah desain berorientasi objek, seperti *garbage collection, reflection,* akar kelas *(root class),* dan juga penyederhanaan terhadap pewarisan jamak *(multiple inheritance).*

Bahasa pemrograman C# dibuat sebagai bahasa pemrograman yang bersifat *general-purpose* (untuk tujuan jamak), berorientasi objek, modern, dan sederhana. C# ditujukan agar cocok digunakan untuk menulis program aplikasi baik dalam sistem klien-server (*hosted system*) maupun sistem embedded *(embedded system),* mulai dari program aplikasi yang sangat besar yang menggunakan sistem operasi yang canggih hingga kepada program aplikasi yang sangat kecil.

Meskipun aplikasi C# ditujukan agar bersifat 'ekonomis' dalam hal kebutuhan pemrosesan dan memori komputer, bahasa C# tidak ditujukan untuk bersaing secara langsung dengan kinerja dan ukuran program aplikasi yang dibuat dengan menggunakan bahasa pemrograman C.

2.12 Teknik Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah proses menelusuri dan mempelajari sebuah program dalam rangka menemukan kesalahan pada perangkat lunak sebelum diserahkan kepada *end user*.

2.12.1 Pengujian *Black Box*

Pengujian *black box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Pengujian *black box* bukan merupakan alternatif dari teknik *white box*, tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode *white box*.

Pengujian *black box* akan menemukan kesalahan dengan kategori sebagai berikut:

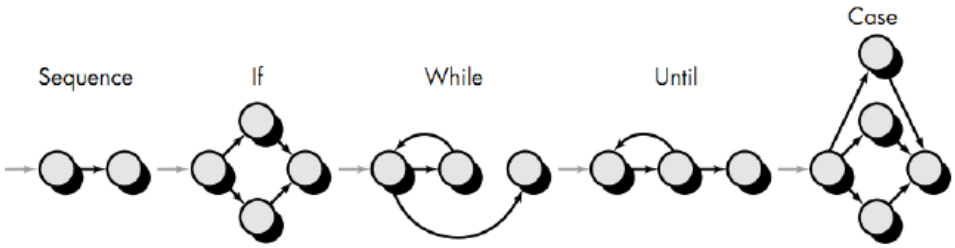
1. Fungsi tidak benar atau hilang
2. Kesalahan antar muka
3. Kesalahan pada struktur data (pengaksesan basis data)
4. Kesalahan inisialisasi dan akhir program
5. Kesalahan performasi

2.12.2 Pengujian *White Box*

Pengujian *white box*, yang kadang – kadang disebut pengujian *glass box*, adalah metode desain *test case* yang menggunakan struktur kontrol desain procedural untuk memperoleh *test case*. Dengan demikian metode pengujian *white box*, perekayasa sistem dapat melakukan *test case* sebagai berikut:

1. Memberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan paling tidak satu kali.
2. Menggunakan semua keputusan logis pada sisi benar dan salah.
3. Mengeksekusi semua *loop* pada batasan dan pada batas operasional
4. Menggunakan struktur dan internal untuk menjamin validitasnya.

Pengujian *white box* yang berupa notasi diagram alir dapat dilihat pada gambar berikut.



Gambar 2.13 Notasi Diagram Alir

2.13 Skala Guttman

Skala Guttman dikembangkan oleh Louis Guttman. Skala ini mempunyai ciri penting, yaitu merupakan skala kumulatif dan mengukur satu dimensi saja dari satu variabel yang multi dimensi, sehingga skala ini termasuk mempunyai sifat undimensional. Skala Guttman yang disebut juga metode *scalogram* atau analisa skala (*scale analysis*) sangat baik untuk meyakinkan peneliti tentang kesatuan dimensi dari sikap atau sifat yang diteliti, yang sering disebut isi universal (*universe of content*) atau atribut universal (*universe attribute*).

Skala Guttman menginginkan jawaban tegas, seperti jawaban benar – salah, ya – tidak, pernah – tidak pernah, positif – negative, tinggi – rendah, baik – buruk, dan seterusnya. Pada skala guttman hanya ada dua interval, yaitu setuju dan tidak setuju. Skala Guttman dapat dibuat dalam bentuk pilihan ganda maupun daftar *checklist*. Untuk jawaban positif seperti benar, ya, tinggi, baik, dan semacamnya diberi skor 1, sedangkan untuk jawaban negative seperti salah, tidak, rendah, buruk, dan semacamnya diberi skor 0.

Kelemahan pokok dari Skala Guttman, yaitu:

1. Skala ini bisa jadi tidak mungkin menjadi dasar yang efektif baik untuk mengukur sikap terhadap objek yang kompleks ataupun untuk membuat prediksi tentang perilaku objek tersebut.
2. Satu skala bisa saja mempunyai dimensi tunggal untuk satu kelompok tetapi ganda untuk kelompok lain, ataupun berdimensi satu untuk satu waktu dan mempunyai dimensi ganda untuk waktu yang lain.