BAB 2
TINJAUAN PUSTAKA

2.1 Lampu Lalu Lintas

Lampu lalu lintas (menurut UU *no*. 22/2009 tentang Lalu lintas dan Angkutan Jalan: alat pemberi isyarat lalu lintas atau APILL) adalah lampu yang mengendalikan arus lalu lintas yang terpasang di persimpangan jalan, tempat penyeberangan pejalan kaki *(zebra cross),* dan tempat arus lalu lintas lainnya. Lampu ini yang menandakan kapan kendaraan harus berjalan dan berhenti secara bergantian dari berbagai arah. Pengaturan lalu lintas di persimpangan jalan dimaksudkan untuk mengatur pergerakan kendaraan pada masing-masing kelompok pergerakan kendaraan agar dapat bergerak secara bergantian sehingga tidak saling mengganggu antar-arus yang ada.

Lampu lalu lintas telah diadopsi di hampir semua kota di dunia ini. Lampu ini menggunakan warna yang diakui secara universal; untuk menandakan berhenti adalah warna merah, hati-hati yang ditandai dengan warna kuning, dan hijau yang berarti dapat berjalan.

2.1.1 Jenis Lampu Lalu Lintas

Lampu lalu lintas memiliki pengaturan agar dapat diperintah dalam perubahan tanda fase berhenti, hati-hati, dan fase hijau yang menandakan berjalan. Berikut penjelasan jenis lampu lalu lintas berdasarkan cara pengoperasianya :

1. *Fixed Time Control*

   Lampu lalu lintas yang pengoperasiaannya menggunakan waktu yang tepat dan tidak mengalami perubahan. Durasi waktu yang digunakan biasanya berkisar antara 35 detik hingga 120 detik.

2. *Dynamic Control*

   Lampu lalu lintas yang pengoperasiaannya dengan pengaturan waktu tertentu dan mengalami perubahan dari waktu ke waktu sesuai dengan kedatangan kendaraan dari berbagai persimpangan.

3. *Coordinated control*

Pengoperasian lampu lalu lintas yang pemasangannya mempertimbangkan beberapa persimpangan yang terdapat pada arah tertentu. Pada sistem terkoordinasi ini memungkinkan untuk pengemudi dalam melakukan perjalanan jauh tanpa menghadapi lampu merah. Koordinasi ini dapat dilakukan dengan mudah hanya pada jalan satu arah dengan tingkatan kepadatan kendaraan yang cukup konstan. Lampu lalu lintas terkadang dikendalikan secara terpusat oleh monitor atau komputer untuk memungkinkan dapat dikoordinasikan secara *real time* untuk menangani perubahan pola lalu lintas.

2.2 *Game*

Game merupakan salah satu media hiburan yang paling popular untuk semua kalangan usia. Sejak pertama kali ditemukan sampai saat sekarang, tek*no*logi *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 berasal dari kata bahasa inggris yang memiliki arti dasar Permainan. Permainan dalam hal ini merujuk pada pengertian "kelincahan intelektual" (*intellectual playability*). *Game* juga bisa diartikan sebagai arena keputusan dan aksi pemainnya. Ada target-target yang ingin dicapai pemainnya. Kelincahan intelektual, pada tingkat tertentu, merupakan ukuran sejauh mana *game* itu menarik untuk dimainkan secara maksimal. *Game* merupakan sebuah bentuk seni dimana penggunanya, disebut dengan pemain (*player*), diharuskan membuat keputusan-keputusan dengan tujuan untuk mengelola sumber daya yang diperoleh dari kesempatan-kesempatan bermain (token) miliknya, untuk mencapai suatu tujuan tertentu.

Video game adalah bentuk *game* yang interaksi utamanya melibatkan media *video* (dan biasanya melibatkan *audio*). Berdasarkan representasi visualnya, *game* dapat dibedakan menjadi 2 jenis, yaitu *game* 2 dimensi (2D) dan 3 dimensi (3D). *Game* 2D adalah *game*, yang secara matematis, hanya melibatkan 2 elemen koordinat kartesius, yaitu x dan y, sehingga konsep kamera pada *game* 2D hanya menentukan "gambar" yang dimana pada *game* dapat dilihat oleh pemain. Sedangkan *game* 3D adalah *game* yang selain melibatkan elemen xdan y, juga melibatkan elemen z pada perhitungannya, sehingga konsep kamera pada *game* 3D benar-benar menyerupai konsep kamera pada kehidupan nyata, yaitu selain digeser (seperti pada *game* 2D), juga dapat diputar dengan sumbu tertentu.

Menariknya akan tantangan maupun visualisasi yang ada pada *game* sering mengakibatkan ketergantungan terhadap *game* tersebut (*addicted)*. Ini merupakan salah satu akibat buruk yang disebabkan oleh *game* (*game addicted)*.

2.2.2 Sejarah *Game* Komputer

Game komputer adalah sebuah permainan interaktif yang dijalankan menggunakan komputer. Program komputer menerima input dari si pemain melalui pengendali dan menampilkan lingkungan buatan melalui TV atau layar monitor. Sejarah tek*no*logi *game* komputer secara langsung berhubungan dengan perkembangan komputer itu sendiri.

Game komputer generasi pertama, tahun 1952, adalah sebuah permainan OXO (tic-tac-toe) dalam versi grafik yang diciptakan oleh seorang mahasiswa Universitas Cambridge bernama A.S. Gouglas. *Game* ini diprogram dengan memakai komputer EDVAC vacuum tube yang memiliki layer berupa *cathode ray tube* (CRT). Pada tahun 1958. William Haginbotham menciptakan *video game* pertamanya, dan *game* komputer pertama yang benar-benar diciptakan menggunakan komputer itu sendiri adalah Spacewar oleh Steve Russel yang terinspirasi dari kisah fiksi ilmiah berjudul Skylark karangan Edward E Smith. Steve Russel bekerja menggunakan sebuah komputer mainframe bernama MIT PDP-1 yang biasa dipakai untuk perhitungan *statistic* pada tahun 1961. Saat itu di

tahun 60-an, komputer adalah barang sangat mewah. Komputer umumnya dipakai untuk kepentingan riset dalam dunia militer.

Game generasi kedua, di tahun 1971, *No*lan Bushnell bersama Ted Dabney menciptakan *game* ber*genre arcade* yang pertama yang diberi judul Komputer Space, yang didasari oleh Spacewar. Tahun 1972, *No*lan dan Ted memulai Atari komputer. Mereka mengembangkan *game* berjudul Pong yaitu *game* pertama yang tersedia untuk publik. Karena sebelumnya *game* hanya ada didalam komputer mainframe yang digunakan untuk kesenangan sendiri. Asal usul Pong dimulai saat *No*lan ingin membuat *game* sederhana dan mudah dimengerti. Menggunakan *memory* dan *micro processor* kelas rendah, dengan kemampuan grafis yang sederhana, akhirnya dia membuat versi elektronik dari permainan ping pong yang kemudian menjadi Pong. Pong kemudian berevolusi menjadi sebuah *game* bernama Breakout. *Game* itu diciptakan oleh Steve Jobs untuk Atari dan bersama Steve Wozniak mulai berpikir untuk menciptakan sebuah PC. Mereka meminjam semua peralatan yang dipakai dalam proyek Breakout untuk membuat sebuah prototype bernama Apple I. dan itu adalah cikal bakal komputer Apple Macintosh yang ada sekarang.

Pada tahun 1980, Atari mengeluarkan *game* berjudul Asteroid dan Lunar Lander. Kedua *game* tersebut adalah *game* pertama yang didaftarkan kepada perusahaan untuk mendapatkan hak cipta yang paten. Asteroid merupakan *game* yang penuh i*no*vasi baru dalam grafisnya, dengan menggunakan metode raster, *game* ini merupakan *grafis vector line* seperti yang ada pada *osciloscope*. Pengenalan Atari Video komputer *System* (Atari 2600) dengan CPU biasa dan slot untuk kasetnya, yang menjadi suatu era baru dalam dunia *game*. Di tahun 1980 itu juga menunjukkan adanya peningkatan penjualan PC yang biasa dipakai untuk *game*.

Kini komputer bersaing dengan mesin *console* seperti PS atau Xbox. Komputer tidak hanya sebagai mesin untuk kepentingan bisnis semata, tapi juga untuk hiburan seperti *game*.

2.2.3 Jenis "*Platform*"/ Alat yang digunakan

Beberapa alat yang digunakan untuk menjalankan suatu *game*, diantaranya adalah :

1. *Arcade Games*

   Game yang dijalankan pada mesin dengan *input* dan *output audio visual* yang telah terintegrasi dan tersedia ditempat-tempat umum, seperti mal, bandara dan sebagainya. Contoh permainan ding-dong.

2. *PC Games*

   Game yang dimainkan pada PC *(Personal Computer)* yang memiliki kelebihan yaitu memiliki tampilan antarmuka yang baik, *output visual* kualitas tinggi karena layar computer biasanya memiliki resolusi yang jauh lebih tinggi dibandingkan dengan layar televisi biasa.

3. *Console Games*

   Game yang dijalankan pada suatu mesin spesifik yang biasanya dimainkan dirumah, seperti Playstation, Microsoft Xbox 360, Nintendo Wii, Sega dan sebagainya.

4. *Mobile Games*

   Game yang dapat dimainkan khusus untuk *mobile phone* atau PDA.

5. *Handheld Games*

   Game yang dimainkan di *console* khusus video *game* yang dapat dibawa kemana-mana, contoh *Game*Boy, Nintendo DS dan Sony PSP.

2.2.4 Jenis-jenis *Game* Komputer

Berdasarkan *genre* pemainannya, *game* dapat dibagi menjadi beberapa *genre*, yaitu [3]:

1. RPG *(Role Playing Game)*

   RPG adalah salah satu *game* yang mengandung unsur *experience* atau *leveling* dalam *game*play nya. Biasanya dalam *game* ini pemain memiliki kebebasan untuk menjelajah dunia *game* tersebut, dan kadang kala dalam beberapa *game*, pemain dapat menentukan ending dari *game* tersebut.

2. FPS *(First Person Shooting)*

FPS adalah *game* tembak menembak yang memiliki ciri utamanya adalah penggunaan sudut pandang orang pertama yang membuat pemain terlihat berada dibelakang senjata.

3. *Third Person Shooter*

TPS adalah *game* yg mirip dengan FPS yaitu memiliki *game*play tembak menembak hanya saja sudut pandang yg digunakan dalam *game* inia dalah orang ketiga.

4. *Strategy*

Strategy adalah *genre game* yg memiliki *gameplay* untuk mengatur suatu unit atau pasukan untuk menyerang markas musuh dalam rangka memenangkan permainan. biasanya di dalam *game strategy*, pemain dituntut untuk mencari gold untuk membiayai pasukan yang dimiliki. *Game*s *Strategy* terbagi menjadi dua :

a. RTS (*Real Time Strategy*)

Pada *game* ini, kita dapat mengendalikan pasukan secara langsung, dari mencari sumber daya, hingga menghancurkan musuh. Semua pertempuran ini dapat kita saksikan secara langsung. Contoh : Warcraft 3.

b. TBS (*Turn Based Strategy*)

Sistem nya seperti *Turn Based* RPG, tetapi disini selain mengendalikan karakter utama, pemain mengendalikan pasukan dan kota untuk dapat memenangkan pertarungan. Contoh : Civilization.

5. *Simulation*

Merupakan *genre* yang mementingkan realisme. Segala faktor pada *game* ini sangat diperhatikan agar mirip berdasarkan dunia nyata. Segala nilai, material, referensi, dan faktor lainnya adalah berdasarkan dunia nyata. Cara memainkannya juga berbeda, karena biasanya kontrol yang dimiliki cukup rumit.

6. *Tycoon*

Tycoon adalah *game* yang menjadikan pemain sebagai seorang businessman yang akan mengembangkan sesuatu Property untuk dikembangkan hingga laku di pasaran.

7. *Racing*

   Racing adalah *game* berjenis balapan yang memungkinkan pemain untuk mengendalikan sebuah kendaraan untuk memenangkan sebuah balapan.

8. *Action Adventure*

   Action Adventure adalah *game* berupa pertualangan salah seorang karakter yangg penuh aksi yangg akan terus ada hingga *game* tersebut tamat.

9. *Arcade*

   Arcade *game* adalah *genre game* yang tidak terfokus pada alur cerita permainan, melainkan hanya dimainkan *"just for fun"* atau untuk kejar-mengejar *point /highscore.*

10. *Fighting Game*

    Fighting adalah *genre game* bertarung. Seperti dalam arcade, pemain dapat mengeluarkan jurus-jurus ampuh dalam pertarungannya. *Genre* fighting biasanya *one on one* dalam sebuah arena yang sempit.

11. *Sports*

    Adalah *genre* bertema permainan olahraga. Sistem permainan akan berbeda-beda tergantung jenis olahraga yang menjadi tema *game* tersebut.

2.3 *Game* Simulasi

Game simulasi dibuat dengan tujuan menempatkan pemain sebagai pengatur dari aktivitas tertentu dan mencoba membuatnya serealistis mungkin.

Dalam artian yang sempit, semua *game* elektronik merupakan simulasi, karena umumnya *game* tidak bisa ada tanpa simulasi aspek kehidupan nyata. Namun, *genre* simulasi hanya mencakup permainan yang berfokus (seluruhnya atau sebagian) pada meniru kegiatan kehidupan nyata. Ini tidak berarti bahwa *game* simulasi harus benar-benar realistis, misalnya, *game* simulasi pertempuran ruang angkasa yang mengsimulasikan suatu kegiatan yang tidak (belum) diketahui oleh manusia. Fitur yang membedakan *game* simulasi adalah penekanan pada realisme dan detail sebanding dengan yang ditemui dalam kehidupan nyata.

Kebanyakan *game* simulasi tidak memiliki alur cerita, karena *game* simulasi berkonsentrasi pada penjelasan kegiatan umum, bukan situasi yang konkret. sebagai pengecualian adalah *game* Wing Comander, yang menggabungkan ruang simulasi tempur dengan narasi berkesinambungan.

Tema simulasi sangat bervariasi, secara teoritis, mereka hanya dibatasi oleh jumlah kegiatan yang dikenal manusia. Tema simulasi meliputi, misalnya, jaringan lalu lintas, perawatan medis, romance, musik, dan banyak lainnya.

Game simulasi dapat dibagi menjadi lima kelompok utama :

1. *Managerial simulations*, menempatkan pemain dalam posisi seorang manajer atau peran serupa.
2. *Construction simulations*, umumnya melibatkan proses pembangunan, misalnya pembangunan kota.
3. *Life simulations*, berhubungan dengan kehidupan seseorang atau kehidupan manusia pada umumnya.
4. *Professional/Social simulations*, dedikasikan untuk kegiatan sosial atau profesi tertentu.
5. *Vehicle simulations*, memungkinkan pemain untuk merasakan mengendarai sebuah kendaraan, misalnya, pesawat, kapal, tank.

Simulasi juga dapat digunakan sebagai pengubah untuk *genre* lain, misalnya strategi (penciptaan kembali pertempuran historis), racing/mengemudi (penanganan kendaraan, tuning, perbaikan), atau olahraga (tim manajemen, pelatihan, dan perdagangan).

Game simulasi sistem lampu lalu lintas yang akan dibangun merupakan jenis *game* simulasi *Managerial* dan *Professional*. Hal ini dikarenakan dalam *game*play permainanya pemain akan berperan sebagai pengendali lampu sistem lampu lalu lintas yang juga merupakan sebuah profesi kerja.

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.

2.4.1 *Game* AI

AI pada *game* modern saat ini memiliki 3 kebutuhan dasar, yaitu, kemampuan untuk menggerakkan karakter, kemampuan untuk membuat keputusan kemana harus bergerak, dan kemampuan untuk bepikir taktis atau strategis.

1. *Movement*

   Movement merupakan algoritma yang bertujuan untuk membuat keputusan dalam bentuk pergerakan. Misalnya pergerakan karakter untuk dapat menghindari penghalang agar dapat menuju sebuah ruangan. Pada *game* The sims ketika karakter ingin makan, maka AI movement akan menggerakan karakter ke kursi yang terdapat didekatnya.

2. *Decision making*

Decision making merupakan AI yang bertujuan untuk menentukan apa yang harus dilakukan selanjutnya oleh sebuah karakter. Pada umumnya sebuah karakter memiliki beberapa tingkah laku. Misalya, berdiri, berlari, diam dan lainya, sehingga decision making bertujuan agar dapat menentukan tingkah laku apa yang sesuai pada saat momen tertentu terjadi.

3. *Strategy*

   Strategy merupakan AI yang biasanya digunakan dalam mengkoordinasi sebuah kelompok pada *game*. Pada dasarnya AI ini tidak hanya mengendalikan satu karakter namun mengendalikan keputusan pada aksi sebuah kelompok karakter.

2.4.2 Algoritma A* *(A Star)*

Algoritma ini berfungsi untuk penentuan jalur terpendek, pada *game* algoritma ini dapat digabungkan dengan movement AI agar karakter dapat bergerak sesuai jalur terpendek yang telah diperhitungkan.

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 *no*tasi 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* 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.

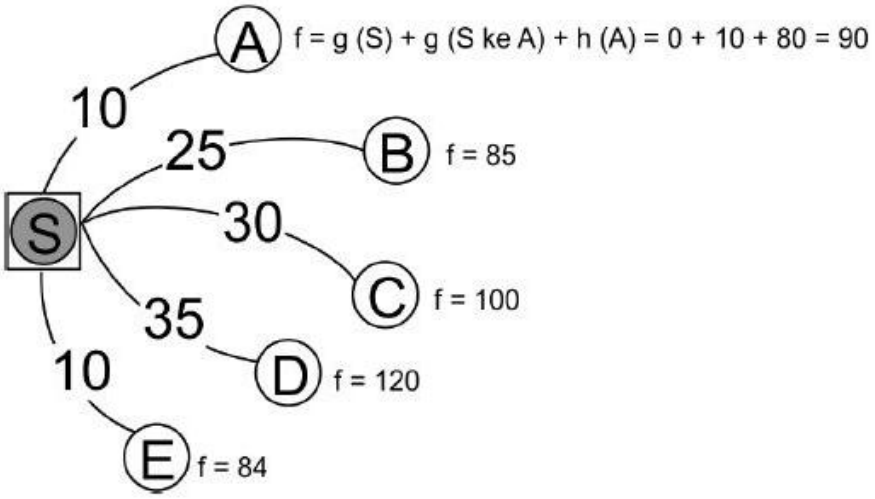
Pada ketiga kondisi tersebut diberikan penanganan yang berbeda-beda. Jika suksesor sudah berada di OPEN, maka dilakukan pengecekan apakah perlu pengubahan parent atau tidak tergantung pada nilai g nya melalui parent lama atau parent baru. Jika melalui parent baru memberikan nilai g yang lebih kecil, 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 *no*de).

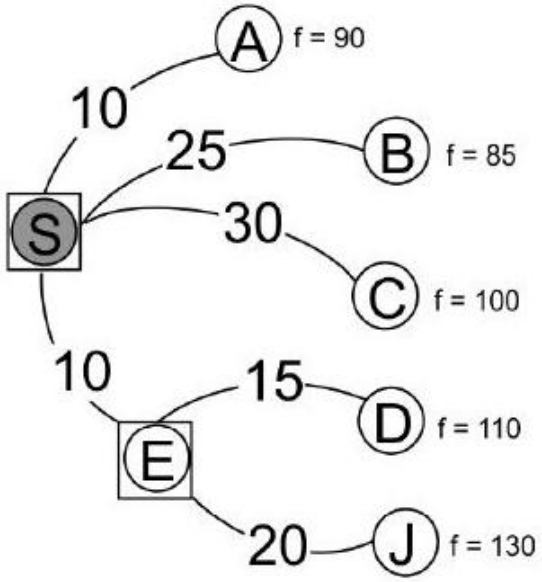
Jika suksesor sudah pernah berada di CLOSED, maka akan dilakukan pengecekan apakah perlu pengubahan parent atau tidak. Jika ya, maka dilakukan perbaruan nilai g dan f pada suksesor tersebut serta pada semua "anak cucunya" yang sudah pernah berada di OPEN. Dengan perbaruan ini, maka anak cucunya tersebut memiliki kesempatan lebih besar untuk terpilih sebagai simpul terbaik (best *no*de).

Jika suksesor tidak berada di OPEN maupun CLOSED, maka suksesor tersebut dimasukkan ke dalam OPEN. Tambahkan suksesor tersebut sebagai suksesornya best *no*de. Hitung biaya suksesor tersebut dengan rumus f = g + h.

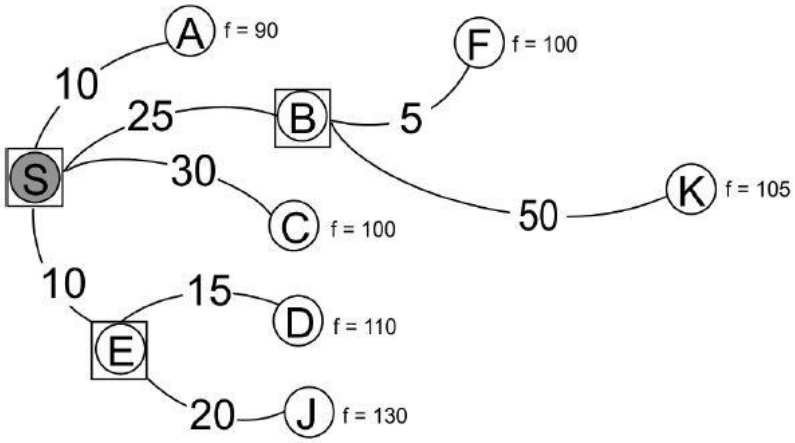
Untuk memperjelas pemahaman mengenai algoritma A* (*A Star*), maka berikut ini adalah langkah-langkah pencarian rute berdasarkan algoritma ini yang dapat dilihat pada gambar 2.1, gambar 2.2, gambar 2.3, gambar 2.4, gambar 2.5 dan gambar 2.6 :



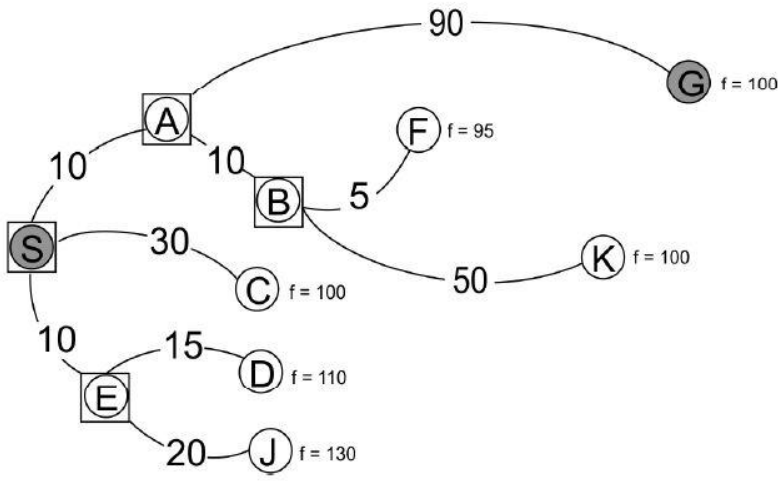
Gambar 2.1 Langkah 1 pencarian rute dengan algoritma A* (A Star)



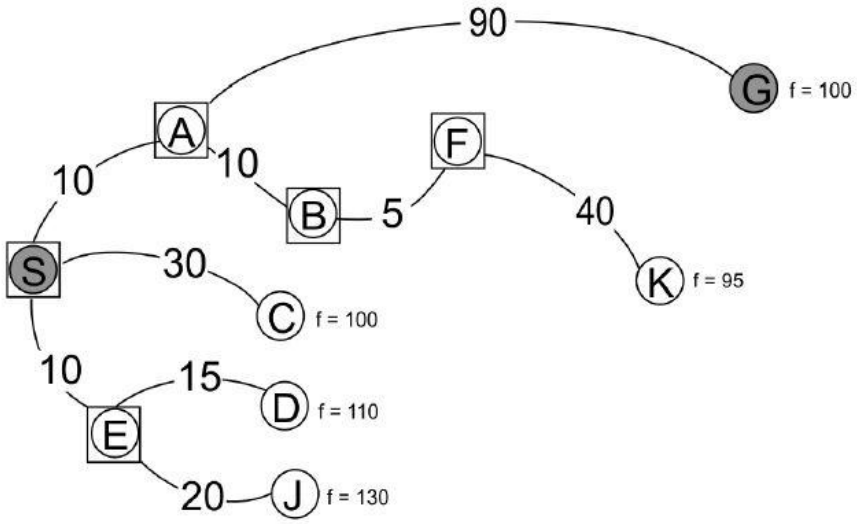
Gambar 2.2 Langkah 2 pencarian rute dengan algoritma A* (A Star)



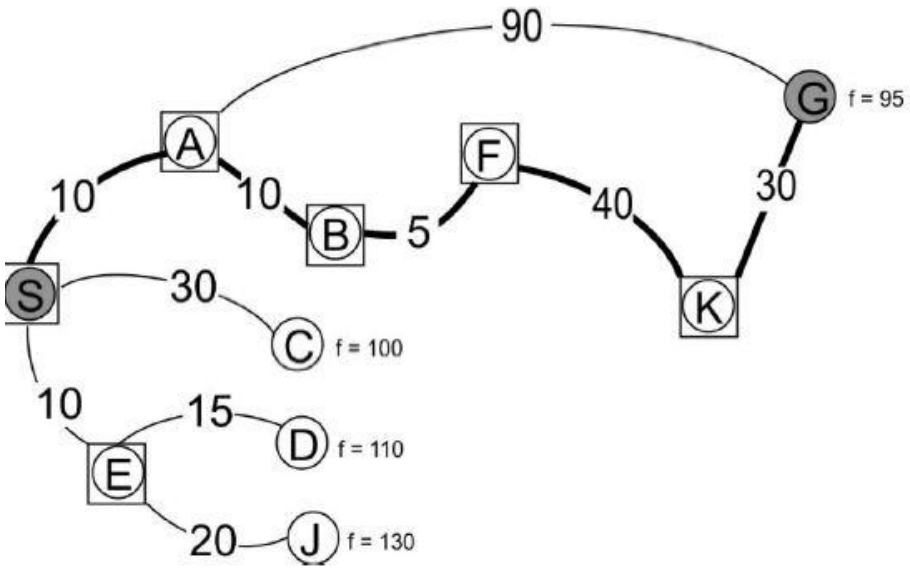
Gambar 2.3 Langkah 3 pencarian rute dengan algoritma A* (A Star)



Gambar 2.4 Langkah 4 pencarian rute dengan algoritma A* (A Star)



Gambar 2.5 Langkah 5 pencarian rute dengan algoritma A* (A Star)



Gambar 2.6 Langkah 6 pencarian rute dengan algoritma A* (A Star)

Simpul dengan tanda kotak menyatakan bahwa simpul tersebut berada di CLOSED (pernah terpilih sebagai Best*No*de) 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 Best*No*de 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 Best*No*de 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 Best*No*de 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 Best*No*de 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 Best*No*de 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 Best*No*de 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 Best*No*de. Karena Best*No*de 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.

2.5 OOP *(Object Oriented Programming)*

OOP adalah teknik memodelkan suatu sistem dunia nyata dalam perangkat lunak berdasarkan objek. Objek tersebut adalah konsep inti. Suatu objek adalah sebuah perangkat lunak entitas atau konsep model dunia nyata.

Ketika sebuah program berjalan, objek individu biasanya tidak dapat berdiri sendiri. Mereka termasuk ke sebuah koleksi dari objek lain yang serupa yang merupakan anggota dari grup yang sama, atau *class*. Sebuah program akan terbentuk dari banyak *class* yang berbeda, setiap *class* terbentuk dari objek-objek yang serupa.

Beberapa sistem perangkat lunak orientasi objek akan memiliki sifat berikut:

1. Abtraksi dengan objek.

   Abtraksi adalah mekanisme yang memungkinkan kompleks, situasi dunia nyata dapat diwakili menggunakan model yang disederhanakan. Orientasi objek abstrak dunia nyata didasarkan pada objek dan interkasi antar objek lainnya.

2. Enkapsulasi *class*.

   Enkapsulasi adalah proses menyembunyikan semua bagian rinci sebuah objek dari dunia luar.

3. Interaksi lewat pesan.

   Untuk memenuhi suatu perintah, objek butuh berinteraksi dengan objek lain. Interaksi dapat antara objek di *class* yang sama, atau objek di *class* lain. Interaksi ini ditangani dengan cara mengirimkan pesan ke objek lain untuk melewati informasi atau meminta aksi.

4. Masa hidup objek.

   Semua objek memiliki masa hidup. Mereka dibuat dan diinisialisasi sebagaimana mereka dibutuhkan pada saat program dijalankan, hidup dan membawa keluar fungsi mereka, dan akhirnya dihancurkan. Sementara mereka ada, mereka mempertahankan identitas mereka sendiri dan kondisi. Banyak objek yaitu contoh dari *class* yang sama bisa hidup pada waktu tertentu. Setiap objek memiliki atribut yang berbeda dari yang lain yaitu contoh objek dari *class* yang sama.

5. *Hirarki class*

Dalam desain orientasi objek, *class* objek tersebut diatur ke dalam hirarki yang memodelkan dan menggambarkan hubungan antar *class*. Hubungan yang paling sederhana adalah sebuah asosiasi.

6. *Polimorfisme*

Polimorfisme adalah karakteristik sistem orientasi objek. Ketika warisan digunakan untuk memperpanjang *class* umum ke *class* yang lebih khusus, biasanya akan mencakup memperluas beberapa perilaku dari *class* umum. *Class* khusus sering akan menerapkan perilaku yang agak berbeda dengan *class* umum, tetapi nama yang digunakan mendefinisikan perilaku akan menjadi sama. Adalah penting bahwa contoh yang diberikan dari sebuah objek menggunakan perilaku yang benar, dan *property* polimorfisme memungkinkan ini terjadi secara otomatis dan mulus. Polimorfisme sebenarnya lebih mudah digunakan daripada dijelaskan.

2.6 UML *(Unified Modeling Language)*

UML adalah penerus gelombang berorientasi objek alnalisis dan desain (OOA & D) metode yang muncul di akhir ‘80-an dan ’90-an awal. Hal yang paling langsung menyatukan metode Booch, Rumbaugh (OMT), dan Jacobson, tapi jangkauan lebih luas dari itu. UML pergi melalui proses standarisasi dengan OMG (*Obyek Manajement Group*) dan sekarang menjadi standar OMG.

UML disebut bahasa permodelan, bukan metode. Sebagian besar metode terdiri, setidaknya pada prinsipnya, kedua model sebuah bahasa dan proses. Bahasa pemodelan adalah *no*tasi (terutama grafis) bahwa metode digunakan untuk mengekspresikan desain. Proses ini menyarankan mereka apa langkah yang harus diambil dalam melakukan desain.

Bagian proses dalam buku banyak metode yang agak samar. Selain itu, kebanyakan orang, ketika mereka mengatakan bahwa mereka menggunakan metode, menggunakan bahasa pemodelan, tapi jarang mengikuti proses. Jadi dalam banyak hal pemodelan bahasa adalah bagian paling penting dari meode ini. Hal ini tentu bagian penting untuk komunikasi. Jika anda ingin mendiskusikan

desain anda dengan seseorang, itu adalah bahasa pemodelan yang anda berdua perlu pahami, bukan proses yang digunakan untuk sampai ke desain tersebut.

2.7 *Diagram* UML

Menggunakan berbagai macam *diagram* dengan fungsi masing-masing untuk menggambarkan setiap proses dari sistem berorientasi objek. Berikut merupakan beberapa *diagram* UML diantaranya:

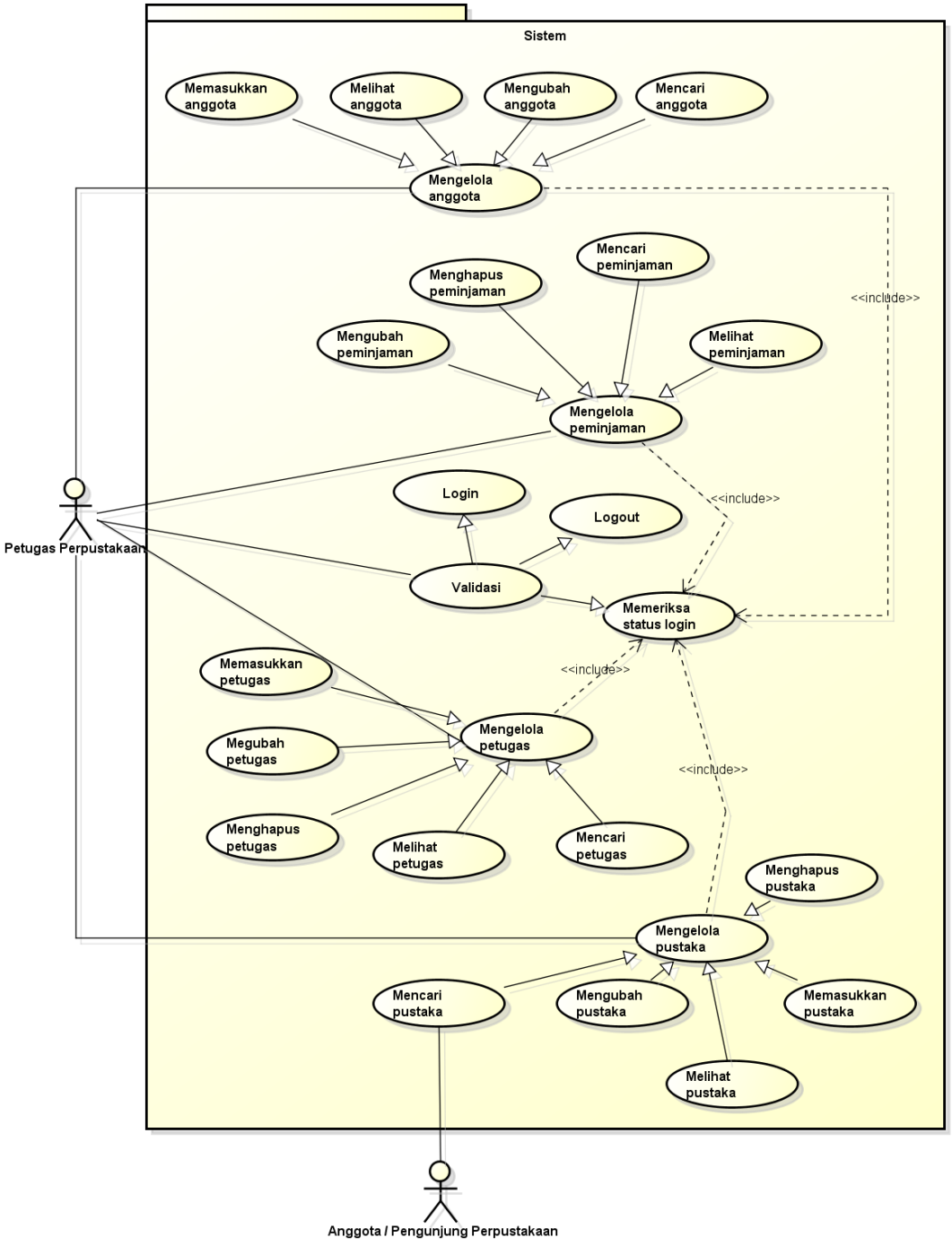
2.7.1 *Use Case Diagram*

Use Case atau *diagram use case* merupakan pemodelan yang digunakan untuk menggambarkan kelakuan (*behavior)* dari sistem yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut.

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 yang akan dibuat diluar sistem yang akan dibuat itu sendiri, jadi walaupun simbol dari actor adalah gambar orang, tapi actor belum tentu merupakan orang.
2. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antarunit atau aktor.

Contoh dari *use case diagram* dapat dilihat pada Gambar 2.7.



Gambar 2.7 Contoh dari *Use Case Diagram*

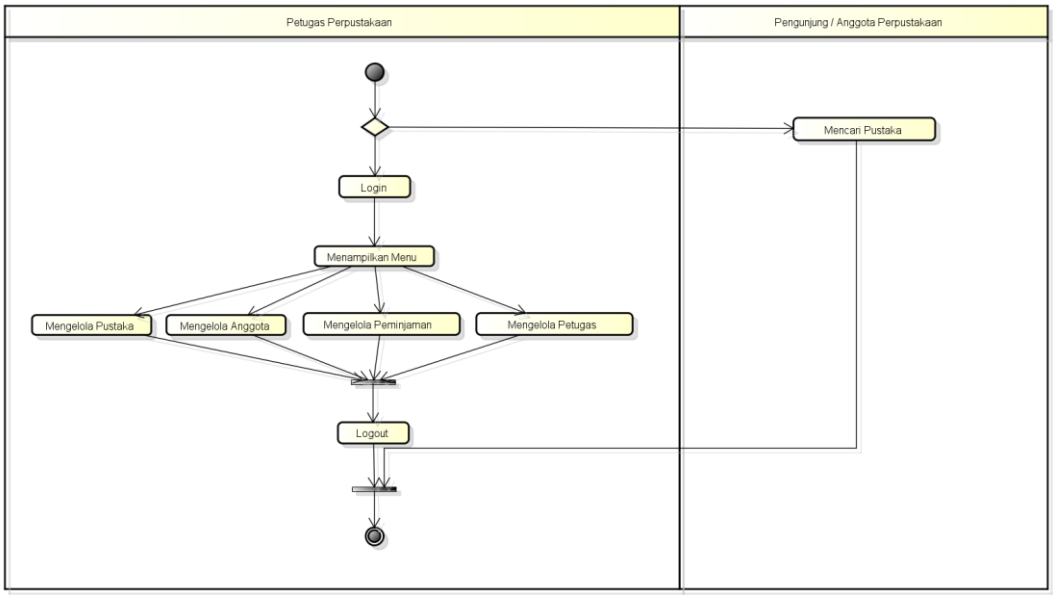
2.7.2 *Activity Diagram*

Diagram aktivitas atau *activity diagram* adalah sebuah *diagram* yang menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis. Dalam *diagram* aktivitas yang perlu diperhatikan 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 di mana setiap urutan aktivitas yang digambarkan merupakan proses bisnis system yang didefinisikan
2. Urutan atau pengelompokan tampilan dari sistem/user interface di mana setiap aktivitas dianggap memiliki sebuah rancangan antaruka tampilan.
3. Rancangan pengunjian di mana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.

Contoh dari *activity diagram* dapat dilihat pada gambar 2.8.



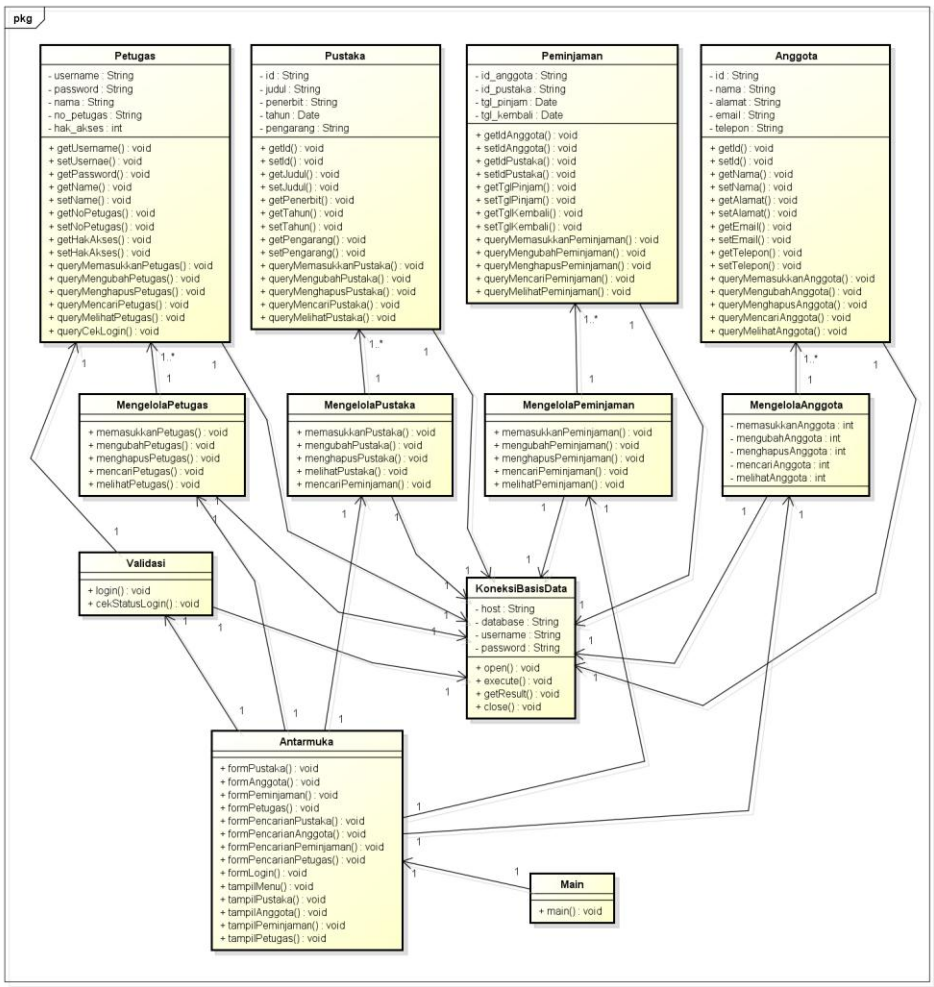
Gambar 2.8 Contoh dari *Activity Diagram*

2.7.3 *Class Diagram*

Diagram kelas atau *class diagram* 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

Contoh dari *class diagram* dapat dilihat pada gambar 2.9.



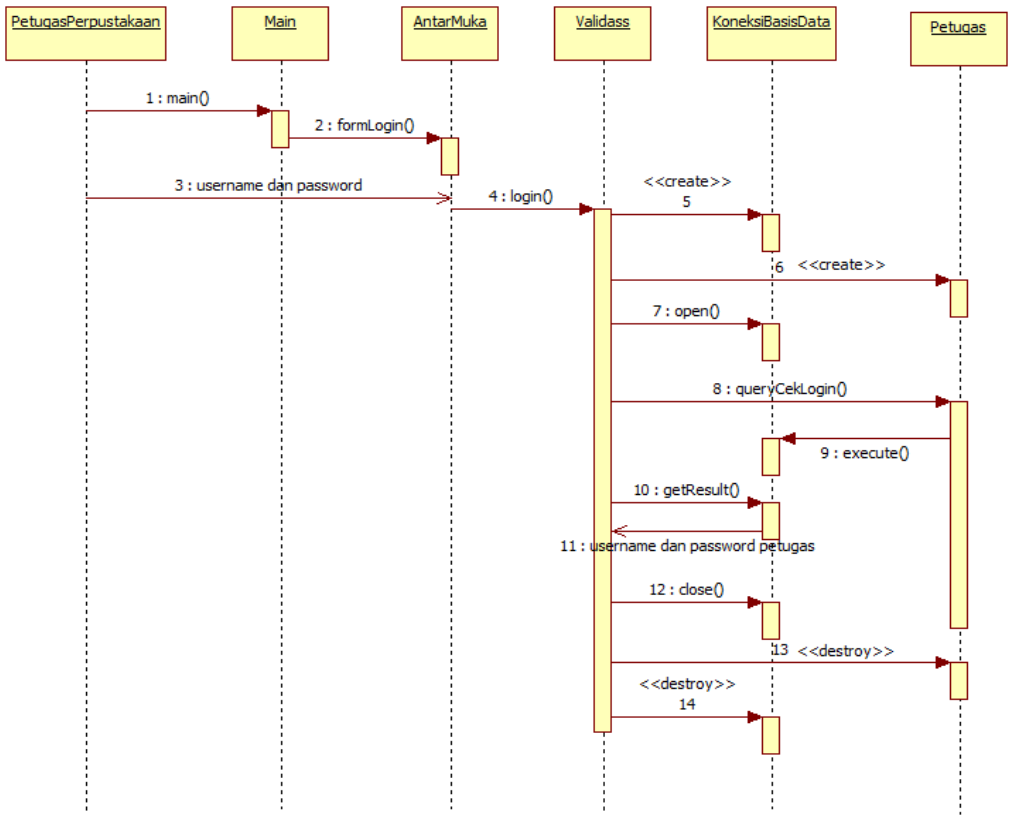
Gambar 2.9 Contoh dari *Class Diagram*

2.7.4 *Sequence Diagram*

Diagram sequence adalah *diagram* yang menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antarobjek. Oleh karena itu untuk menggambarkan *diagram* sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang d2nstansiasi menjadi objek itu.

Banyaknya *diagram* sekuen yang harus digambarkan adalah sebanyak pendefinisian *use case* yang memiliki prose situ 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.

Contoh dari *sequence diagram* dapat dilihat pada gambar 2.10.



Gambar 2.10 Contoh dari *Sequence Diagram*

2.8 Teknik Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean [8].

2.8.1 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-fungsi yang tidak benar atau hilang
2. Kesalahan dalam interface
3. Kesalahan dalam struktur data atau akses *database* eksternal
4. Kesalahan kinerja
5. Inisialisasi dan kesalahan terminasi

2.8.2 Pengujian *White Box*

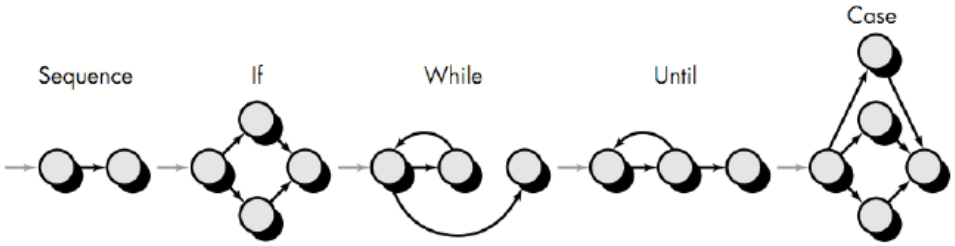
Pengujian *white box*, yang kadang-kadang disebut pengujian *glass box*, adalah metode desain *test case* yang menggunakan struktur kontrol desain procedural untuk memperoleh *test case*. Dengan menggunakan metode pengujian *white box*, perekayasa sistem dapat melakukan *test case* sebagai berikut :

1. Memberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan paling tidak satu kali
2. Menggunakan semua keputusan logis pada sisi *true* dan *false*
3. Mengeksekusi semua *loop* pada batasan mereka dan pada batas operasional mereka

4. Menggunakan struktur data internal untuk menjamin validitasnya.

5.

Pengujian *white box* yang berupa *no*tasi *diagram* alir dapat dilihat pada gambar 2.11.



Gambar 2.11 Notasi *Diagram* Alir

2.9 Unity 3D

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 Tech*n*ologies, 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.*