BAB II

LANDASAN TEORI

II.1 Permainan (*Game*)

Game adalah permainan elektronik yang menggunakan media komputer maupun mesin konsol seperti PlayStation atau Xbox sudah menjamur dan terkenal dimana-mana. *Game* mulai popular ke seluruh dunia dimulai dengan populernya berbagai jenis *game* di Amerika di era 70'an dan mulai berkembang ke berbagai Negara di awal tahun 80'an.

Game pada awalnya dibuat untuk sekedar permainan sederhana yang menguji ketangkasan ataupun kecepatan reaksi pemain, dalam perkembangannya *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 dihidupkan didalam berbagai mesin *game* (catur, monopoly dan bilyard). Berbagai *genre* terus muncul, sebagaian adalah perbaikan dari *genre* yang lama, sebagian lagi adalah *genre* baru dan sebagian adalah kombinasi antar *genre*. Semua ini membuat *game* semakin variatif dan semakin banyak produk *game* yang muncul [1].

II.1.1 Klasifikasi Permainan

Sejarah *game* tidak lepas dengan namanya *platform*, dari yang awalnya hanya mesin *Arcade* (mesin ding dong) hingga sampai sekarang ini yang sudah mencapai era *Mobile Games*. Untuk lebih jelasnya, berikut ini adalah klasifikasi *game* berdasarkan jenis *Platform* yang di gunakan: [1]

1. *Arcade Games* yaitu yang sering di sebut ding-dong di Indonesia, biasanya berada di tempat khusus dan memiliki boks atau mesin yang memang khusus di desain untuk jenis *video games* tertentu dan tidak jarang bahkan

memiliki fitur yang dapat membuat pemainnya lebih merasa “masuk” dan menikmati. Seperti pistol, kursi khusus, sensor gerakan, sensor injakkan dan stir mobil (beserta transmisinya).

2. *PC Games* (*personal computer*) yaitu *video game* yang dimainkan menggunakan *Personal Computers*.
3. *Console Games* yaitu *video game* yang dimainkan menggunakan konsol tertentu seperti *playstation XBOX, Nintendo*, dll.
4. *Handheld Game* yaitu *game* yang dimainkan di konsol khusus *video game* yang dapat dibawa kemana-mana, contoh *nintendo DS dan Sony PSP*.
5. *Mobile Games* yaitu *game* yang biasa di mainkan khusus untuk *handphone atau PDA.*

Klasifikasi permainan berdasarkan pembatasan usia dari pemainnya, menurut ***ESRB*** (***Entertainment Software Rating Board***) adalah sebagai berikut: [1]

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. Mature (17+) memiliki konten yang mungkin cocok untuk orang usia 17 dan lebih tua. Judul judul dalam kategori ini mungkin berisi intens kekerasan, darah, konten seksual dan bahasa yang kuat.
6. ***Adults Only (18+)*** memiliki konten yang hanya boleh dimainkan oleh orang-orang dengan usia 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).

II.1.2 Sudut Pandang Permainan

Sebuah *game* biasanya mempunyai sudut pandang permainan tersendiri disesuaikan berdasarkan *genre game* yang diambil. Berikut beberapa macam sudut pandang permainan yang biasa digunakan: [1]

1. *Side Scrolling*

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*

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*

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*).

II.1.3 Jenis Permainan

Game dari tahun-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 *genre*. Pada dasarnya, *genre-genre game* yang beredar di pasaran dapat dikelompokkan sebagai berikut: [1]

a. *Advergames*

Advergames adalah permainan yang dikembangkan untuk tujuan periklanan.

b. *Action Game*

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

c. *Adventure Game*

Adventure game adalah tipe *game* yang umumnya membuat pemain harus berjalan mengelilingi suatu tempat yang terkondisi, seperti sebuah istana, gua yang berkelok, dan planet yang jauh. Pemain melakukan navigasi suatu area mencari pesan-pesan rahasia, memperoleh obyek yang memiliki kemampuan yang bervariasi, bertempur dengan musuh, dan lain-lain. Untuk membuat game ini, diperlukan perencanaan yang akurat sehingga memiliki alur cerita yang menarik bagi pemain, contohnya: *Tomb Raider, Prince of Persia, Hitman*, dll.

d. *Fighting Game*

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

e. FPS *(First Person Shooter)*

FPS (*First Person Shooter*) adalah tipe *game* yang menggunakan sudut pandang orang pertama untuk membidik atau membunuh musuh, sehingga kita hanya melihat tangannya saja dan tidak melihat tubuh karakter yang dimainkan, contohnya: *Call of Duty, Half Life, Crysis*, dll.

f. RPG *(Role Playing Game)*

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

g. *Platform Game*

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

h. *Puzzle Game*

Puzzle game adalah tipe *game* yang umumnya membuat pemain menggunakan kemampuan berpikirnya sebagai pengganti keterampilan reaksi yang cepat karena terdapat rahasia yang perlu dipecahkan. *Game* ini lebih bersifat statis dibanding *action game*. Pembuatan *game*

tipe ini seringkali ditunjang dengan AI (*Artificial Intelligence*), contohnya: *7 Wonders of The Ancient World.*

i. *Racing Game*

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

j. *Simulation Game*

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

k. *Sandbox Game*

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

l. RTS *(Real Time Strategy)*

RTS (*Real Time Strategy*) adalah tipe *game* yang mengharuskan pemainnya menggunakan taktik dan strategi untuk jeli dalam melihat setiap peluang, kelemahan musuh dan bijaksana dalam menggunakan sumber daya yang ada, contohnya: *Command and Conquer*.

m. *Sport Game*

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

II.2 Kecerdasan Buatan/AI (*Artificial Intelligence*)

AI adalah ilmu rekayasa yang membuat mesin mempunyai intelegensi tertentu khususnya program komputer yang ‘cerdas’. Intelegensi merupakan bagian kemampuan komputasi untuk mencapai tujuan di dalam dunia. Ada bermacam jenis dan derajat intelegensi untuk manusia, binatang, dan beberapa mesin.

AI berhubungan dengan penggunaan komputer untuk melaksanakan tugas atau menyelesaikan masalah yang memerlukan kecerdasan atau kepintaran. Pada umumnya, komputer memerlukan aplikasi atau *software* untuk melaksanakan tugas atau menyelesaikan masalah. Kebanyakan tugas atau masalah yang dikontrol oleh komputer adalah berbentuk rutin (berdasarkan memori/ingatan atau perhitungan) walaupun mungkin rumit dari segi perhitungan. Sekiranya pelaksanaan tugas atau penyelesaian masalah itu memerlukan langkah-langkah yang tidak rutin, yang biasanya memerlukan pemikiran yaitu bukan saja menggunakan ingatan atau rumusan/formula, maka aplikasi atau software yang berasaskan teknologi AI diperlukan. Intelegensi perlu dilihat dari berbagai sudut pandang, termasuk pemahaman, inferensi (*inferencing/reasoning*), daya penjelasan, pembelajaran, komunikasi dan sebagainya. Seseorang atau sebuah mesin yang pintar perlu memiliki beberapa sifat ini. Sebuah mesin tidak mungkin pintar tanpa program yang bersifat pintar atau cerdas [7].

II.2.1 Karakteristik Kecerdasan Buatan

Karakteristik ini didasarkan pada pandangan bahwa AI ada 4 kategori yaitu: [7]

1. **Sistem yang dapat berpikir seperti manusia**

   Sistem yang dapat beraksi seperti manusia. Contoh sistem ini dikembangkan dan dipresentasikan Alan Turing pada 1950 yang kita kenal dengan *Turing Test*. Untuk lolos dari system ini diperlukan pemrosesan bahasa alami (*Natural Language Processing*), representasi pengetahuan (*Knowledge Representasion*), alasan terotomatisasi (*Automated Reasoning*), dan mesin yang dapat belajar (*Machine Learning*).

2. Sistem yang dapat beraksi seperti manusia

Sistem yang dapat berpikir seperti manusia. Sistem ini menggunakan teori pikiran presisi untuk diekspresikan sebagai program komputer. Sistem Newell & Simon's GPS (*General Problem Solver*) mencari untuk penyelesaian masalah "jalan yang dilakukan manusia". Sistem ini menggunakan ilmu kognitif yaitu gabungan antara model komputer AI dan teknik psikologi.

3. Sistem yang dapat berpikir secara rasional

Sistem yang dapat berpikir secara rasional. Sistem ini menggunakan Logika yaitu dimulai dari pengetahuan dalam istilah formal, kemudian memberikan premis yang benar, dan akhirnya didapatkan kesimpulan yang benar.

4. Sistem yang dapat beraksi secara rasional

Sistem yang dapat beraksi secara rasional. Sistem ini merupakan Agen yang kadang-kadang perlu menggunakan perasaan dan dapat bertindak dengan baik. Rasional agen beraksi untuk mencapai tujuan dan memberikan kepercayaan. Lebih umum daripada berpikir secara rasional, karena inferensi yang benar tidak selalu mungkin, tetapi aksi selalu dibutuhkan. Selain itu, sistem agen ini lebih disetujui untuk pengembanan ilmu daripada pendekatan berdasarkan pada pikiran manusia atau tingkah laku manusia.

II.2.2 Kecerdasan Buatan yang Digunakan

Algoritma A* (A *Star*) 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. Dengan perhitungan biaya seperti ini, algoritma A* (A *Star*) adalah *complete* dan optimal. Dalam notasi matematika dituliskan dengan rumus :

$$\mathbf{F(n) = G(n) + H(n).} \quad \mathbf{(II.1)}$$

Sama dengan algoritma dasar *Best First Search*, algoritma A* (A *Star*) ini juga menggunakan *OPEN* dan *CLOSED*. Terdapat tiga kondisi bagi setiap suksesor yang dibangkitkan, yaitu : sudah berada di *OPEN*, sudah berada di *CLOSED*, dan 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 (2.1). [7]. Berikut adalah *Pseudocode* algoritma A* dapat dilihat di gambar II.1

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

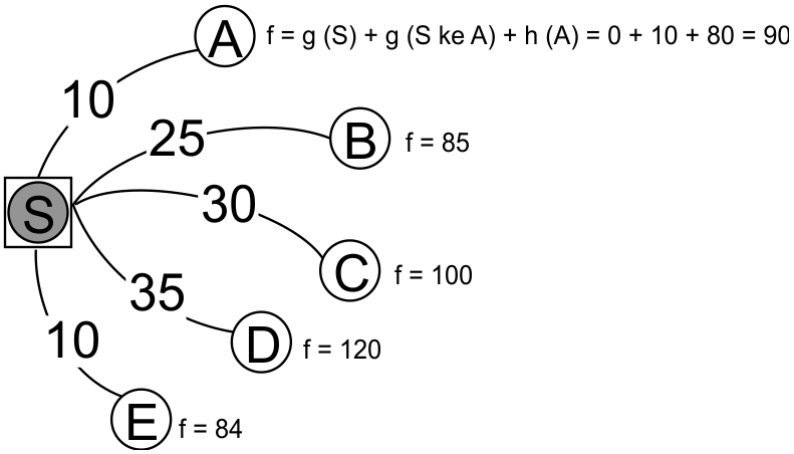
loop sampai goal ditemukan atau sampai tidak ada simpul didalam OPEN
        if OPEN = kosong then
                Gagal
        else
```

```
        BestNode = simpul yang ada di OPEN dengan f minimal
        Pindahkan simpul terbaik tersebut dari OPEN ke CLOSED
if BestNode = goal then
        Sukses
else
        Bangkitkan semua suksesor BestNode tapi jangan buat pointer
        Untuk setiap suksesor kerjakan :
        Hitung g (suksesor) = g (BestNode) + actual cost (dari BestNode
        ke suksesor)
{Periksa suksesor}
if suksesor ada di OPEN then {sudah pernah dibangkitkan tapi
belum diproses}
        OLD = simpul di OPEN yang sama dengan suksesor tersebut
        Tambahkan OLD sebagai suksesor BestNode
        Buat pointer dari OLD ke BestNode
        Bandingkan nilai g (OLD) dengan g (suksesor)
        if g (OLD) lebih baik then
                Ubah parent OLD ke BestNode
                Ubah nilai g dan f yang ada pada OLD
        end
else
        if suksesor ada di CLOSED then {sudah pernah dibangkitkan
        tapi belum diproses}
                OLD = simpul di CLOSED yang sama dengan suksesor
                tersebut
                Tambahkan OLD sebagai suksesor BestNode
                Bandingkan nilai g (OLD) dengan g (suksesor)
                        if g (OLD) lebih baik then
                        Ubah parent OLD ke BestNode
                        Ubah nilai g dan f yang ada pada OLD
                        Propagasi untuk semua suksesor OLD dengan
```

```
                    penulusuran DFS dengan aturan loop sampai simpul
                    suksesor tidak ada di OPEN atau simpul tidak punya
                    suksesor
                        if suksesor ada di OPEN then
                            Propagasi diteruskan
                        else
                        if nilai g via suksesor lebih baik then
                            Propagasi diteruskan
                        else
                            Propagasi dihentikan
                        end
                    end
                end
            else {suksesor tidak ada di OPEN maupun CLOSED}
                Masukkan suksesor ke OPEN
                Tambahkan suksesor tersebut sebagai suksesornya
                BestNode
                Hitung f = g (suksesor) + h (suksesor)
            end
        end
end
```

Gambar II.1 Notasi Pseudo-code Algoritma A*

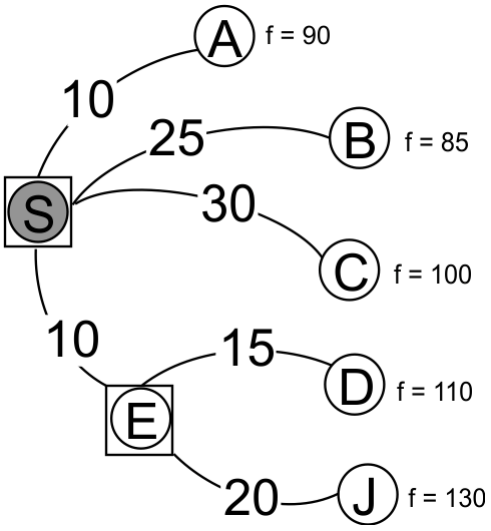
Untuk lebih memperjelas pemahaman mengenai algoritma A*, berikut adalah contoh langkah-langkah pencarian rute terpendek menggunakan algoritma A* akan dijelaskan di gambar II.2.



Gambar II.2 Langkah 1 pencarian rute terpendek menggunakan algoritma A*

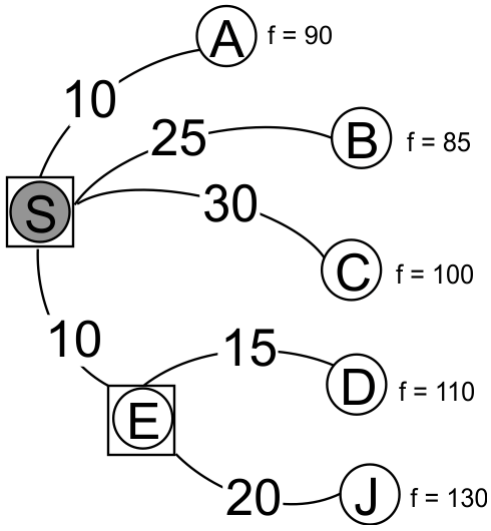
Simpul dengan tanda kotak menyatakan bahwa simpul tersebut berada di *CLOSED* (pernah terpilih sebagai *BestNode*) sedangkan simpul tanpa tanda kotak menyatakan simpul yang ada berada di *OPEN*.

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 pencarian rute menggunakan algoritma A* akan dijelaskan di gambar II.3



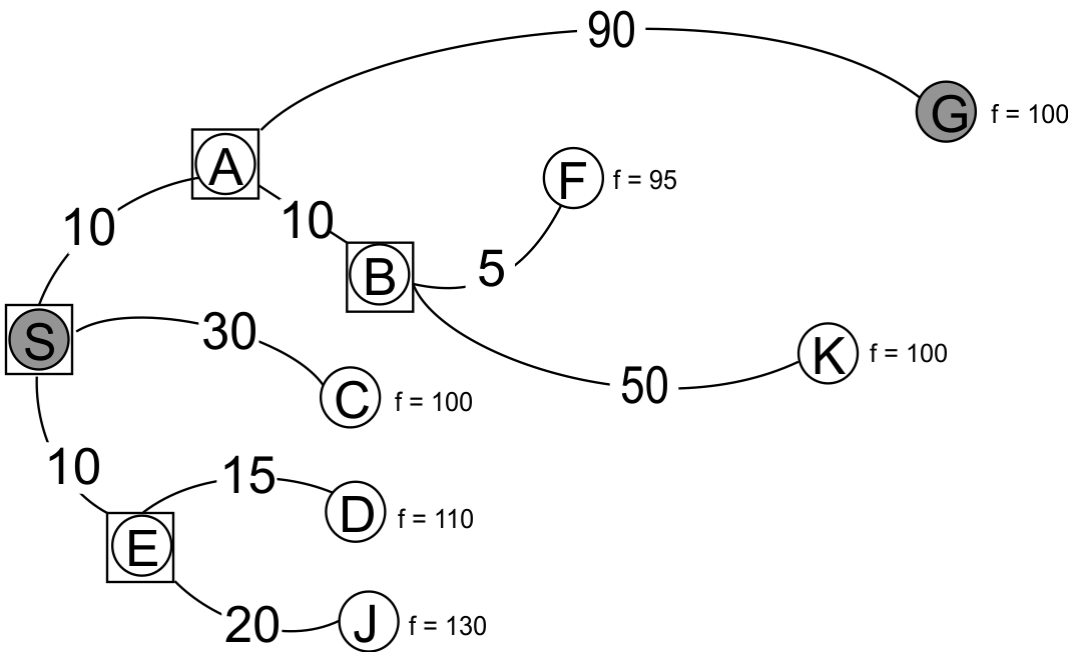
Gambar II.3 Langkah 2 pencarian rute terpendek menggunakan algoritma A*

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 pencarian rute menggunakan algoritma A* akan dijelaskan di gambar II.4



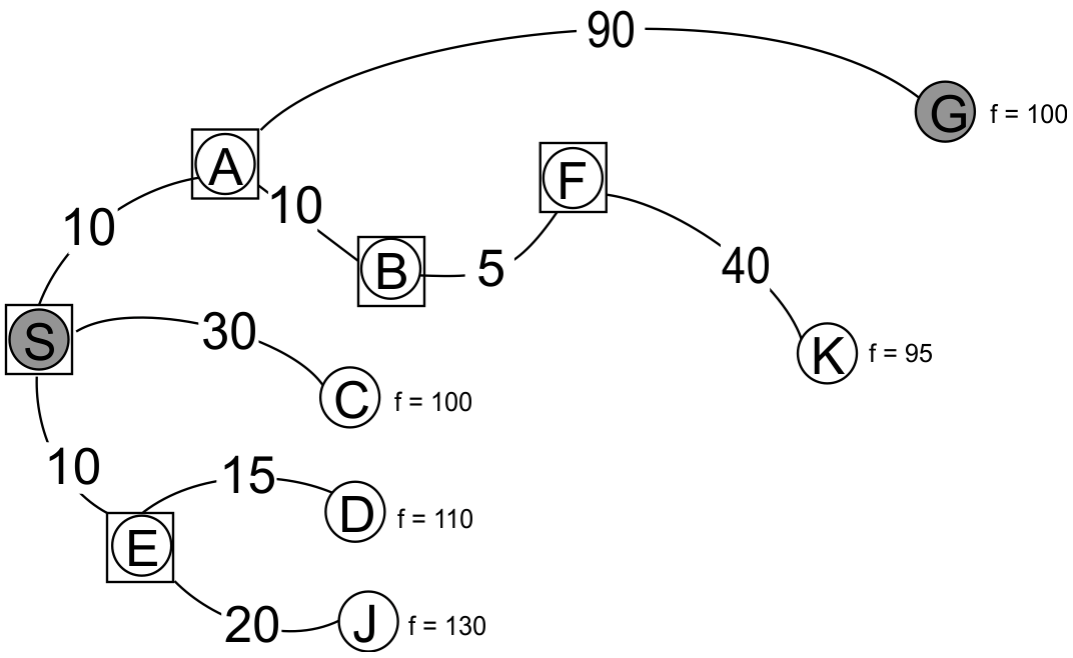
Gambar II.4 Langkah 3 pencarian rute terpendek menggunakan algoritma A*

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 pencarian rute menggunakan algoritma A* akan dijelaskan di gambar II.5



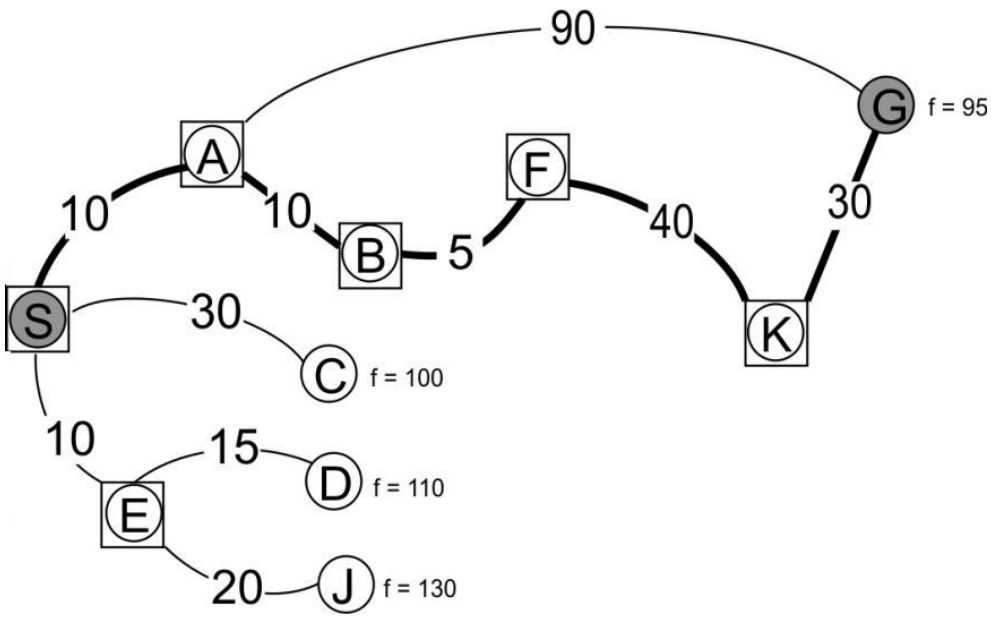
Gambar II.5 Langkah 4 pencarian rute terpendek menggunakan algoritma A*

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 pencarian rute menggunakan algoritma A* akan dijelaskan di gambar II.6



Gambar II.6 Langkah 5 pencarian rute terpendek menggunakan algoritma 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 pencarian rute menggunakan algoritma A* akan dijelaskan di gambar II.7



Gambar II.7 Langkah 6 pencarian rute terpendek menggunakan algoritma A*

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. 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*). [3]

II.3 *Tools* yang Digunakan

Pada sub bab ini akan dijelaskan *tools* yang akan digunakan dalam penelitian tugas akhir ini.

II.3.1 OOP (*Object Oriented Programming*)

Pemrograman berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya. Pemrograman berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Pemrograman berorientasi objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Pemrograman berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek dan pengujian berorientasi objek.

Pada saat ini, pemrograman berorientasi objek banyak dipilih karena metodologi lama banyak menimbulkan masalah seperti adanya kesulitan pada saat transformasi hasil dari satu tahap pengembangan ke tahap berikutnya, misalnya pada metode pendekatan terstruktur, jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan saat ini sangat beragam (aplikasi bisnis, *real-time*, *utility*, dan sebagainya) dengan *platform* yang berbeda-beda, sehingga menimbulkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi tersebut.

Keuntungan menggunakan pemrograman berorientasi objek adalah sebagai berikut: [8]

1. Meningkatkan produktivitas
2. Kecepatan pengembangan
3. Kemudahan pemeliharaan
4. Adanya konsistensi
5. Meningkatkan kualitas perangkat lunak

Berikut ini beberapa bahasa pemrograman yang sudah menggunakan konsep pemrograman berbasis objek adalah:

1. C++
2. Visual C++
3. Visual Basic
4. C#
5. PHP
6. Java

II.3.2 UML (*Unified Modeling Language*)

UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah system dengan menggunakan diagram dan teks-teks pendukung. UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataannya UML paling banyak digunakan pada metodologi berorientasi objek.

UML menyediakan bermacam-macam diagram yang dapat digunakan untuk memodelkan aplikasi berorientasi objek yang dibuat, yaitu: [8]

1. ***Class Diagram*** untuk menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun system.
2. ***Object Diagram*** untuk menggambarkan struktur sistem dari segi penamaan objek dan jalannya objek dalam system.
3. ***Component Diagram*** untuk menunjukkan organisasi dan ketergantungan di antara kumpulan komponen dalam sebuah sistem.
4. ***Composite Structure Diagram*** untuk menggambarkan struktur dari bagian-bagian yang saling terhubug maupun mendeskripsikan struktur pada saat berjalan (*runtime*) dari *instance* yang saling terhubung,
5. ***Package Diagram*** untuk mengumpulkan elemen-elemen yang saling terkait dalam diagram UML.
6. ***Deployment Diagram*** menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi.

7. ***Use Case Diagram*** untuk memodelkan kelakuan (*behavior)* sistem yang akan dibuat.
8. ***Activity Diagram*** menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis.
9. ***State Machine Diagram*** untuk menggambarkan perubahan status atau transisi status sebuah mesin atau sistem.
10. ***Sequence Diagram*** menggambarkan kelakuan objek pada use case dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antarobjek.
11. ***Communication Diagram*** menggambarkan interaksi antarobjek/bagian dalam bentuk urutan pengiriman pesan.
12. ***Timing Diagram*** merupakan diagram yang fokus pada penggambaran terkait batasan waktu.
13. ***Interaction Overview Diagram*** untuk menggambarkan sekumpulan urutan aktivitas.

II.3.3 Bahasa Pemrograman Java

Java adalah merek dagang Sun Microsystems yang merujuk pada bahasa pemrograman yang dikembangkan oleh Sun Microsystems yang versi alfa dan versi beta-nya dirilis tahun 1995. Bahasa pemrograman Java digunakan untuk membuat program yang dapat didistribusikan melalui jaringan komputer. Secara generik, nama Java merujuk pada seperangkat *tool* untuk membuat dan mengimplementasikan program dengan menggunakan bahasa pemrograman Java.

Sun sendiri mendeskripsikan Java sebagai bahasa pemrograman yang "sederhana, berorientasi-objek, bertipe *static*, terkompilasi, *multi-threaded*, *garbage collected*, netral arsitektur, *robust*, aman, dan dapat diperluas". Rincian sifat-sifat yang dideskripsikan itu dapat dipaparkan sebagai berikut: [6]

1. Sederhana

Pengembang java banyak membuang fitur-fitur yang tidak diperlukan seperti yang dimiliki oleh bahasa pemrograman tingkat tinggi lainnya. Sebagai contoh, java tidak mendukung pointer matematis, *casting* tipe implisit, struktur

atau *unions*, operator *overloading*, template, *header filer*, maupun pewarisan majemuk (*multiple inheritance*).

2. Berorientasi-objek

Seperti halnya C++, Java menggunakan kelas-kelas untuk mengelola kode ke dalam modul-modul logika. Pada saat *runtime*, program menciptakan objek dari kelas-kelas yang ada. Kelas-kelas Java dapat merupakan warisan dari kelas lain, tetapi pewarisan majemuk di mana suatu kelas mewariskan metode-metode dan *field* yang berasal lebih dari satu kelas tidak diperkenankan.

3. Bertipe *static*

Semua objek yang digunakan dalam program harus dideklarasikan sebelum digunakan. Cara ini memungkinkan kompilator Java melaporkan dan melokalisir terjadinya konflik-konflik tipe.

4. Terkompilasi

Sebelum dapat menjalankan program yang ditulis dengan bahasa pemrogaman java, program harus dikompilasi oleh kompilator java. Hasil kompilasinya berupa file kode byte (*byte-code*), yang mirip dengan file kode mesin, yang dapat dieksekusi di bawah system operasi apapun asal memiliki *interpreter* java. *Interpreter* membaca file kode *byte* dan menterjemahkannya perintah kode *byte* itu menjadi perintah bahasa mesin yang dapat dieksekusi langsung oleh mesin yang menjalankan program Java. Jadi dapatlah dikatakan bahwa Java merupakan bahasa yang dikompilasi sekaligus diinterpretasi.

5. *Multi-threaded*

Program java dapat berisi eksekusi *multiple threads*, yang memungkinkan program mengulas citra pada layar pada satu *thread*, sementara tetap menerima masukan dari *keyboard* pada *thread* utamanya. Semua aplikasi memiliki paling tidak satu *thread* yang menggambarkan alur eksekusi utama program.

6. *Garbage Collected* (Pembuang Sampah)

Program Java menyingkirkan sendiri ‘sampah-sampah’ yang tidak berguna, artinya program tidak perlu menghapus objek-objek yang

dialokasikannya di memori. Ini menjadikan pemrogram bebas dari masalah-masalah pengelolaan memori.

7. Bebas arsitektur

Pada dasarnya, Java tidak dirancang untuk prosesor ataupun system operasi tertentu. Ia dirancang untuk bekerja pada berbagai arsitektur prosesor dan berbagai sistem operasi. Oleh karena itu, ia bekerja dengan interpreter Java untuk mengubah program Java menjadi *byte-code* sebelum dijalankan oleh mesin tertentu.

8. Tangguh (*Robust*)

Karena interpreter Java memeriksa seluruh akses sistem yang dilakukan program, maka program Java tidak akan membuat sistem menjadi *crash*. Bila terjadi masalah serius, program Java membuat pengecualian (*exception*). *Exception* ini dapat ditangani dan dikelola oleh program tanpa berisiko memacetkan sistem.

9. Aman

Sistem java tidak hanya memverifikasi seluruh akses ke memori tetapi juga menjamin bahwa tidak ada virus yang ‘membonceng’ pada *applet* yang sedang berjalan. Karena java tidak mendukung pointer, program tidak dapat memperoleh akses ke area system dimana ia tidak mendapatkan otorisasi.

10. Dapat diperluas

Program Java mendukung metode *native* yakni fungsi-fungsi yang ditulis dalam bahasa lain, Biasanya C++. Dukungan terhadap metode *native* ini memungkinkan pemrogram menulis fungsi yang bisa dieksekusi secara lebih cepat daripada fungsi yang sama yang ditulisdalam Java. Metode *native* dikaitkan secara dinamis (*Dynamically linked*) ke program Java, yakni dikaitkan dengan program saat *runtime*. Dalam perkembangannya, bila java berkembang lebih jauh dan mampu mengeksekusi fungsi-fungsi lebih cepat dari bahasa lain, metode *native* mungkin tidak diperlukan lagi.

II.3.4 Greenfoot

Greenfoot merupakan *software* pembelajaran pemrograman berorientasi objek, dengan berbasis pemrograman java yang memungkinkan kita untuk membuat *game* 2 dimensi. *Greenfoot* diprakarsai oleh Michael Kolling pada tahun 2003. *Prototype* pertama dibangun oleh Michael Kolling dan Paul 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 menginstalasi program *java* 5 (JDK 1.5) atau program *java* terbaru. Didalam *Greenfoot* terdapat *project* yang dinamakan *scenario*. Dalam *project* tersebut, kita dapat membuat dua macam tipe *class* dari *library Greenfoot*, yaitu *class world* dan *class actor*. *World* merupakan latar dari game yang akan dibuat. *Actor* adalah kelas yang berfungsi sebagai karakter pada game yang akan dibuat [5].

II.3.5 Pengolahan Data

Pengolahan data secara sederhana diartikan sebagai proses mengartikan data-data lapangan sesuai dengan tujuan, rancangan, dan sifat penelitian. Misalnya dalam rancangan penelitian kuantitatif, maka angka-angka yang diperoleh melalui alat pengumpul data tersebut harus diolah secara kuantitatif, baik melalui pengolahan statistik inferensial maupun statistik deskriptif. Lain halnya dalam rancangan penelitian kualitatif, maka pengolahan data menggunakan teknik non statitistik, mengingat data-data lapangan diperoleh dalam bentuk narasi atau kata-kata, bukan angka-angka. Mengingat data lapangan disajikan dalam bentuk narasi kata-kata, maka pengolahan datanya tidak bisa dikuantifikasikan.

Perbedaan ini harus dipahami oleh peneliti atau siapapun yang melakukan penelitian, sehingga penyajian data dan analisis kesimpulan penelitian relevan dengan sifat atau jenis data dan prosedur pengolahan data yang akan digunakan. Di atas dikatakan bahwa pengolahan data diartikan sebagai proses mengartikan data lapangan, yang berarti supaya data lapangan yang diperoleh melalui alat

pengumpul data dapat dimaknai, baik secara kuantitatif maupun kualitatif, sehingga proses penarikan kesimpulan penelitian dapat dilaksanakan. Dengan demikian, pengolahan data tersebut dalam kaitannya dengan praktek pendidikan adalah sebagai upaya untuk memaknai data atau fakta menjadi makna

Dalam setiap kegiatan penelitian selalu ada kegiatan pengumpulan data. Metode pengumpulan data dalam penelitian dalam penelitian meliputi: [12]

1. **Observasi nonpartisipan**

   Pada metode ini peneliti hanya mengamati, mencatat apa yang terjadi. Metode ini banyak digunakan untuk mengkaji perilaku pemustaka di perpustakaan.

2. **Kuesioner**

   Kuesioner adalah pertanyaan terstruktur yang diisi sendiri oleh responden atau diisi oleh pewawancara yang membacakan pertanyaan dan kemudian mencatat jawaban yang diberikan. Pertanyaan yang diberikan pada kuesioner ini adalah pertanyaan menyangkut fakta dan pendapat responden, sedangkan kuesioner yang digunakan pada penelitian ini adalah kuesioner tertutup, dimana responden diminta menjawab pertanyaan dan menjawab dengan memilih dari sejumlah alternatif.

3. ***Pre Test* dan *Post Test***

   Pre test merupakan suatu bentuk pertanyaan yang digunakan untuk menguji tingkatan pengetahuan seseorang terhadap materi yang akan disampaikan. Manfaat dari *pre test* adalah untuk mengetahui kemampuan awal seseorang mengenai materi yang akan disampaikan. Sedangkan *post test* merupakan suatu bentuk pertanyaan yang diberikan setelah materi disampaikan. Manfaat dari *post test* adalah untuk memperoleh gambaran tentang kemampuan setelah berakhirnya penyampaian materi. Hasil *post test* ini akan dibandingkan dengan hasil *pre test* sehingga akan diketahui seberapa jauh efek dari penyampaian materi yang dilakukan, disamping sekaligus dapat diketahui bagian dari bahan penyampaian materi yang masih belum dipahami oleh responden.

II.3.5.1 Skala Likert

Skala Likert adalah skala yang dapat dipergunakan untuk mengukur sikap, pendapat, dan persepsi seseorang atau sekelompok orang mengenai suatu gejala atau fenomena pendidikan. Dalam skala Likert terdapat dua bentuk pernyataan yaitu pernyataan positif yang berfungsi untuk mengukur sikap positif dan pernyataan negatif yang berfungsi untuk mengukur sikap negatif.

Skala Likert dalam penelitian mempunyai gradasi nilai dari sangat positif sampai sangat negatif. Lalu hasil data kuesioner tersebut diolah dengan rumus sebagai berikut:

$$\mathbf{Y = P/Q*100\%} \qquad \mathbf{(II.2)}$$

Keterangan :

Y = Persentase

P = Bobot setiap pilihan * jawaban

Q = Total bobot maksimal

II.3.5.2 Pengujian Kolmogorov-Smirnov

Uji Kolmogorov Smirnov merupakan pengujian normalitas yang banyak dipakai, salah satu kelebihan dari pengujian ini adalah sederhana dan tidak menimbulkan perbedaan persepsi di antara satu pengamat dengan pengamat yang lain, yang sering terjadi pada uji normalitas dengan menggunakan grafik. Konsep dasar dari uji normalitas Kolmogorov Smirnov adalah dengan membandingkan distribusi data (yang akan diuji normalitasnya) dengan distribusi normal baku. Distribusi normal baku adalah data yang telah ditransformasikan ke dalam bentk Z-Score dan diasumsikan normal. Jadi sebenarnya uji Kolmogorov-Smirnov beda data antara yang diuji normalitasnya dengan data normal baku. Seperti pada uji beda biasa, jika signifikasi diatas 0,05 maka tidak terjadi perbedaan yang signifikan, dan begitu juga sebaliknya [14].

II.3.5.3 Pengujian Wilcoxon

Uji Wilcoxon pertama kali diperkenalkan oleh Frank Wilcoxon pada tahun 1945, Uji Wilcoxon merupakan pengembangan dari uji t dengan ketelitian hasil analisis Wilcoxon dibandingkan Uji t adalah tidak hanya dapat menunjuk arah perbedaan tetapi juga dapat menunjukkan perbedaan antar kelompok-kelompok yang dibandingkan. Uji peringkat bertanda Wilcoxon digunakan jika besaran maupun arah perbedaan relevan untuk menentukan apakah terdapat perbedaan yang sesungguhnya antara data yang satu dengan data yang lainnya. Uji peringkat bertanda Wilcoxon tidak hanya memanfaatkan informasi tentang arah tetapi juga besarnya perbedaan pasangan nilai itu [14].