BAB 2

TINJAUAN PUSTAKA

2.1 *Game*

Video game merupakan salah satu media hiburan yang sangat populer untuk semua orang, baik itu tua maupun muda. Teknologi *game* telah mengalami kemajuan yang sangat pesat. Terbukti hal ini ditandai dengan berkembangnya jenis, produk dan alat yang beragam untuk berinteraksi dengan *game*.

2.1.1 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. Pada awalnya, *game* identik dengan permainan anak-anak. Kita selalu berpikir *game* merupakan suatu kegiatan yang dilakukan oleh anak-anak yang dapat menyenangkan hati mereka. Dengan kata lain, segala bentuk kegiatan yang memerlukan pemikiran, kelincahan intelektual dan pencapaian terhadap target tertentu dapat dikatakan sebagai *game* .

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" mana

pada *game* yang dapat dilihat oleh pemain. Sedangkan *game* 3D adalah *game* yang selain melibatkan elemen x dan 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.

2.1.2 Sejarah Singkat Perkembangan *Game*

Video game pertama diciptakan oleh dua orang utama, yaitu William Higinbotham dan Steve Russell[7]. Saat itu William Higinbotham berperan sebagai orang pertama yang mendesain dan mengimplementasikan sebuah *video game*, dan Steve Russell berperan sebagai orang pertama yang menciptakan sebuah *game* yang kemudian menginspirasikan bermilyaran *dollar* industri *video game*.

Berdasarkan sejarah, yang pertama kali menemukan *video game* ialah *United States Department of Enery*. Khususnya ialah seorang pria bernama William Higinbotham yang menjadi kepala pada bagian *Instrumentation division for Brookhaven National Laboratory*. Beforen Brookhaven, William pernah bekerja sebelumnya di *Manhattan Project* dan telah menyaksikan ledakan atom pertama. Namun, pada tahun 1950-an, masyarakat khawatir akan kekuatan atom, dan Brookhaven mencoba untuk menampilkannya untuk memasyarakatkan penelitian yang dilakukan pada acara kunjungan tahunan. Ratusan orang berkunjung ke laboratorium setiap musim gugur untuk melihat berbagai penelitian yang sedang dilakukan disana. Pada tahun 1958, William mendapatkan ilham bagaimana membuat pengunjung agar tidak bosan pada setiap kunjungan tahunan

ke laboratoriumnya, yaitu dengan cara membuat tampilan interaktif. Tampilan ini menjadi *video tennis game.*

Generasi ketiga dimulai pada tahun 1983 dengan dipasarkannya *Japan Family Computer* atau lebih dikenal dengan nama FAMICOM (kemudian dikenal sebagai *Nintendo Entertainment System* di seluruh dunia). FAMICOM juga termasuk sistem 16-bit seperti *Mega Drive* / *Genesis*. Di era ini pulalah terjadi perang konsol *game* yang pertama antara perusahaan konsol Nintendo dengan SEGA.

Generasi keenam ini ditandai dengan munculnya konsol-konsol *game next-generation* dari masing-masing perusahaan seperti SONY, SEGA, Nintendo serta munculnya satu lagi konsol *game* baru yang diluncurkan oleh Microsoft yang diberi nama XBOX. Perang konsol game ini akhirnya mengakibatkan jatuhnya perusahaan konsol SEGA yang tidak dapat lagi meneruskan konsol generasi selanjutnya (*Dreamcast*) mereka dan lebih memilih untuk berkonsentrasi dibidang pembuatan game konsol.

Dikarenakan semakin canggihnya teknologi di bidang teknologi maka kemudian 3 perusahaan konsol terbesar (Sony, Nintendo, dan Microsoft) mengeluarkan kembali konsol *next-generation* mereka dengan keluaran terbaru. Sony mengeluarkan konsol selanjutnya yang diberi nama PS3 (Playstation 3), lalu Nintendo dengan Nintendo Wii kemudian Microsoft dengan Xbox 360. Pada generasi ini semakin dikembangkan sistem permainan *online* atau permainan yang melibatkan banyak pemain yang terhubung dengan antar konsol dan semakin ditinggalkannya permainan *single player*.

2.1.3 Jenis-jenis *Game*

Berikut adalah jenis-jenis *game* antara lain[4] :

1. *Game Arcade*

   Biasanya berada ditempat-tempat khusus dan memiliki *box* atau mesin yang memang khusus di desain untuk jenis *video game* tertentu, Bahkan pemain bisa lebih merasa masuk ke dalam *game* yang dimainkan seperti pistol, kursi khusus, sensor gerakan dan stir mobil.

2. *Game PC*

   Video game yang dimainkan pada *Personal Computer* yang memiliki keunggulan dalam tampilan antar muka yang sangat baik, *Output visual* kualitas tinggi karena layar komputer mempunyai resolusi lebih tinggi dibandingkan layar televisi. Kekurangan pada *game PC* adalah spesifikasi komputer sangat beragam hingga memungkinkan beberapa *game* dapat ditampilkan dengan baik pada satu komputer tetapi tidak berjalan baik pada komputer yang lainnya.

3. *Game* Konsol

   Video game yang dimainkan pada suatu mesin tertentu misalnya Xbox, Sony Playstation, Sega genesis, Gameboy Advance dan sebagainya.

4. *Game Mobile*

   Game konsol atau *game mobile phone* yang dapat dibawa kemana-mana.

5. *Game Online*

   Game yang hanya dapat dimainkan secara *online* melalui LAN atau internet.

2.1.4 *Genre Game*

Setiap *game* memiliki *genre* atau tipenya, baik dalam satu kategori *genre* maupun beberapa *genre* dalam satu *game*. berikut ini adalah tipe-tipe *game*[4]:

1. *Action*

   Game ini dikenali dari aksi didalamnya yang berupa pergerakan, *attack*, reaksi dan pergerakan-pergerakan lainnya. Yang lebih ditekankan pada *game* jenis ini adalah aksi didalamnya, bukan dari sudut pandang cerita.

2. *Adventure*

   Dalam *game* tipe ini, *player* akan menyelesaikan misi yang diberikan sepanjang petualangan di dalam *game*. *Player* memulai permainan dengan inventory yang terbatas, senjata dan makanan, kemudian dikirim untuk melakukan suatu *quest*, dan dalam perjalanannya ada tugas yang harus diselesaikan.

3. *Casual Game*

   Yang termasuk dalam *casual game* adalah *board game*, *card game* dan *game shows*. *Game* ini bertujuan untuk menghibur *player*, dan selain itu dapat juga digunakan untuk melatih, mempelajari dan meningkatkan pengetahuan dan *level skill gamer*

4. *Educational Game*

   Game edukasi memberikan penekanan pada unsur pendidikan dan pembelajaran. Contoh *game* edukasi yang sederhana dapat berupa pertanyaan pilihan ganda, isian maupun essai.

5. *Role-Playing Game (RPG)*

   Pada *game* bertipe ini memiliki dunia yang luas untuk dieksplorasi oleh *player*, baik untuk mencari harta karun, objek dan status (*leveling*), sambil menghancurkan monster dan objek yang menghalaginya. Dalam RPG ada tujuan yang harus dicapai player untuk menyelesaikan permainan.

6. *Strategy*

   Dalam *game* tipe ini dibutuhkan pemikiran dan perencanaan yang benar untuk dapat memenangkan permainan dan biasanya menggunakan sudut pandang *overhead* sehingga *player* dapat melihat seluruh area permainan.

7. *Simulation*

   Simulation game atau *sims* adalah suatu bentuk permainan *game* yang menggambarkan situasi dunia nyata, baik dalam rupa simulasi *driving* maupun dalam hubungannya dengan lingkungan interaksi antara manusia.

8. *Puzzle*

   Puzzle game memiliki tujuan yang sangat sederhana untuk menyelesaikan *puzzle* yang diberikan.

2.1.5 *Game Design*

Game Design adalah suatu aksi untuk menentukan seperti apakah suatu *game* seharusnya akan dibuat. Menurut Rolling dan Adams dalam bukunya *Game Design* merancang *game* adalah proses dari[8] :

a. Membayangkan permainan
b. Menentukan cara permainan tersebut bekerja
c. Menjelaskan unsur – unsur yang membentuk permainan tersebut

d. Mengirim informasi kepada kelompok yang akan membuat *game* tersebut.

Game design dibagi menjadi 3 bagian, yaitu : *core mechanics*, cerita dan interaktivitas[8].

1. *Core Mechanics*

   Aturan yang mendefinisikan operasi dari dunia permainan membentuk mekanisme inti permainan, atau dasar *gameplay*. Mekanisme inti merupakan terjemahan dari visi perancang ke dalam sekumpulan peraturan konsisten yang dapat diinterpretasikan oleh komputer atau lebih tepatnya aturan-aturan yang dapat dinterpretasikan oleh orang yang menulis *software* yang dapat diinterpretasikan oleh komputer.

2. Cerita dan Narasi

   Semua permainan *game* menceritakan cerita. Kompleksitas dan kedalaman dari cerita tersebut tergantung dari permainannya. Narasi berarti bagian dari cerita yang diceritakan.

3. Interaktivitas

   Interaktivitas adalah cara bagi pemain untuk melihat, mendengar dan bereaksi dalam dunia permainan. interaktivitas mencakup banyak topic berbeda : grafis, suara, antarmuka pengguna dan segala sesuatu yang dapat merepresentasikan pengalaman bermain bagi *user*.

Tujuan utama dari membuat *game design* dokumen adalah untuk menarik minat pembaca dan untuk menginformasikan kepada pembaca mengenai gambaran keseluruhan *game* yang akan dibuat sehingga nantinya gambaran-gambaran ini dapat dikumpulkan menjadi suatu *requirement.*

2.2 AI (*Artficial Intelligence*)

Kecerdasan buatan sering disebut juga dengan *Artificial Intelligence* (AI). AI merupakan salah satu bagian ilmu komputer yang mempelajari tentang bagaimana caranya agar komputer dapat melakukan pekerjaan seperti yang dapat dilakukan oleh manusia. Pada awal diciptakannya, komputer hanya difungsikan sebagai alat hitung saja. Namun seiring dengan perkembangan zaman, teknologi komputer semakin ditingkatkan dan peran komputer semakin mendominasi kehidupan umat manusia. Komputer tidak lagi hanya digunakan sebagai alat hitung, melainkan komputer diharapkan dapat diberdayakan untuk mengerjakan hal-hal yang dapat memudahkan pekerjaan manusia.

Agar komputer bisa bertindak seperti apa yang dapat dilakukan manusia, maka komputer juga harus diberi bekal pengetahuan dan mempunyai kemampuan untuk menalar. Untuk itu AI akan mencoba untuk memberikan beberapa metode untuk membekali komputer agar komputer bisa menjadi mesin yang pintar.

Dari beberapa perspektif, AI dapat dipandang sebagai berikut[3]:

1. Dari perspektif kecerdasan, AI adalah bagaimana membuat mesin yang cerdas dan dapat melakukan hal-hal yang sebelumnya hanya dapat dilakukan manusia.

2. Dari perspektif bisnis, AI adalah sekelompok alat bantu (*tools*) yang berdayaguna dan metodologi yang menggunakan alat-alat bantu tersebut untuk menyelesaikan masalah-masalah bisnis.
3. Dari perspektif pemrograman, AI meliputi studi tentang pemrograman simbolik, pemecahan masalah, dan proses pencarian (*search*).
4. Dari perspektif penelitian, riset tentang AI dimulai pada awal tahun 1960-an, percobaan pertama adalah membuat program permainan catur, membuktikan teori, dan *general problem solving.*

AI seperti bidang ilmu lainnya juga memiliki sejumlah sub-disiplin ilmu yang sering digunakan untuk pendekatan yang esensial bagi penyelesaian suatu masalah dan dengan aplikasi bidang AI yang berbeda. Bidang-bidang tugas AI dibagi menjadi tiga kelompok yaitu[3] :

a. *Mundane Task*

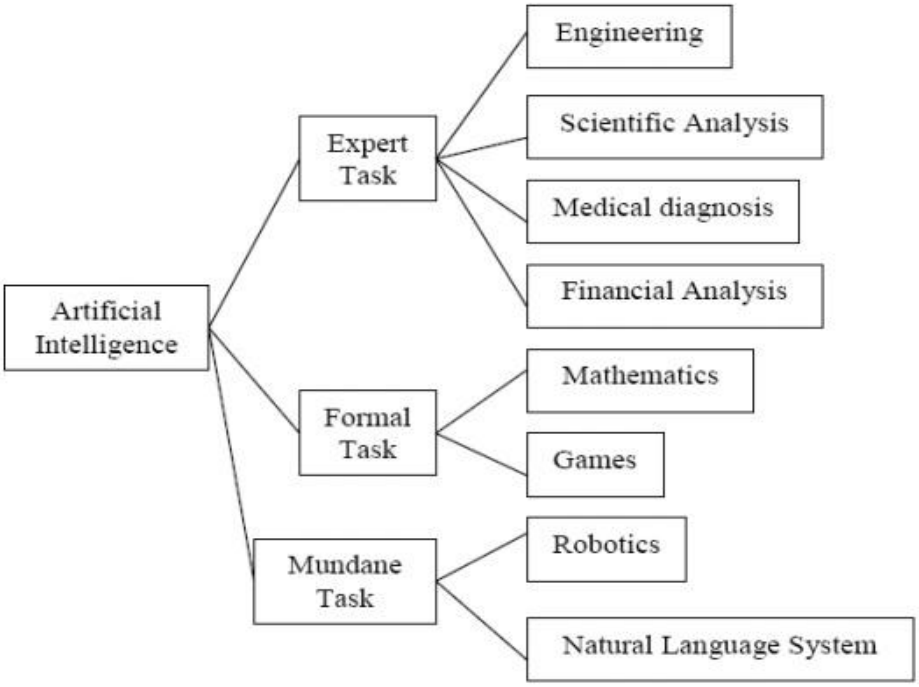
   Secara harfiah, arti mundane adalah keduniaan. Di sini, AI digunakan untuk melakukan hal-hal yang sifatnya duniawi atau melakukan kegiatan yang dapat membantu manusia.

b. *Formal task*

   AI digunakan untuk melakukan tugas-tugas formal yang selama ini manusia biasa lakukan dengan lebih baik.

c. *Expert task*

   AI dibentuk berdasarkan pengalaman dan pengetahuan yang dimiliki oleh para ahli. Penggunaan ini dapat membantu para ahli untuk menyampaikan ilmu-ilmu yang mereka miliki.



Gambar 2.1 Bidang-bidang tugas (task domains) dari AI[3].

2.2.1 Teknik Pemecahan Masalah AI

Terdapat empat teknik dasar pemecahan masalah yang terdapat pada bidang *academic/traditional artificial intelegence* diantaranya adalah[12]:

a. *Searching*

Pada teknik *searching* atau pencarian ini terdiri dari beberapa langkah untuk merealisasikannya. Langkah pertama adalah mendefinisikan ruang masalah untuk suatu masalah yang dihadapi. Langkah kedua adalah mendefinisikan aturan produksi yang digunakan untuk mengubah suatu keadaan ke keadaan lainnya. Langkah terakhir adalah memilih metode pencarian yang tepat sehingga dapat menemukan solusi terbaik.

b. *Reasoning*

Teknik *reasoning* atau penalaran merupakan teknik penyelesaian masalah dengan cara merepresentasikan masalah ke dalam basis pengetahuan menggunakan *logic* atau bahasa formal (bahasa yang dipahami komputer). Teknik ini melakukan proses penalaran berdasarkan basis pengetahuannya untuk menemukan solusi.

c. *Planning*

Planning adalah suatu metode penyelesaian masalah dengan cara memecah masalah ke dalam sub-sub masalah yang lebih kecil, menyelesaikan sub-sub masalah satu demi satu, kemudian menggabungkan solusi-solusi dari sub-sub masalah tersebut menjadi sebuah solusi lengkap dengan tetap mengingat dan menangani interaksi yang terdapat pada sub-sub masalah tersebut.

d. *Learning*

Pada ketiga teknik sebelumnya, seseorang harus mengetahui aturan yang berlaku untuk sistem yang akan dibangunnya. Tetapi, pada masalah tertentu terkadang aturan tidak bisa didefinisikan secara benar ataupun lengkah. Hal tersebut mungkin dikarenakan data-data yang didapat tidak lengkap. Melalui teknik yang disebut *learning* ini, secara otomatis aturan yang diharapkan bisa berlaku umum untuk data-data yang belum pasti diketahui dapat ditemukan.

2.2.2 *Game Artificial Intelligence*

Dalam ruang lingkup permainan AI yang diterapkan sedikit berbeda dengan traditional AI. Pada permainan *game* AI dikenal juga dengan sebutan *game* AI. Menurut penerapannya pada entitas permainan AI dibedakan menjadi tiga, yaitu :

a. *Combat* AI

AI ini diterapkan pada musuh dalam area permainan. penerapannya dimaksudkan agar musuh memiliki kemampuan untuk dapat melawan pemain lain. Sebagai contoh pada permainan *arcade-shooter* sang musuh harus mengetahui kapan ia harus menembak, menghindar, dan bersembunyi.

b. *Non-Combat* AI

AI ini diterapkan pada entitas lain dalam arena permainan selain pemain baik itu yang berperan sebagai musuh ataupun bukan. AI ini dapat kita lihat pada permainan berjenis RPG. Dimana dalam permainan jenis tersebut banyak ditemui entitas-entitas yang memiliki kegiatannya masing-masing dalam suatu arena permainan.

c. *Analitycal* AI

AI ini diterapkan dalam permainan permainan yang berjenis simulasi. Hal tersebut dikarenakan AI ini memiliki tingkat kerumitan yang tinggi . sebagai contoh dalam simulasi pesawat terbang, simulasi balap mobil yang membutuhkan AI tersendiri.

2.2.3 Algoritma Pencarian

Metode pencarian dikatakan penting untuk menyelesaikan permasalahan karena setiap *state* (keadaan) menggambarkan langkah-langkah untuk menyelesaikan permasalahan. Metode pencarian dikatakan penting untuk perencanaan karena dalam sebuah permainan akan menentukan apa yang harus dilakukan, dimana setiap *state* menggambarkan kemungkinan posisi pada suatu saat. Metode pencarian adalah bagian dari kesimpulan, dimana setiap *state* menggambarkan hipotesis dalam sebuah rangkaian deduktif.

Permasalahan pencarian dapat diselesaikan dengan beberapa metode yaitu:

1. Metode pencarian yang pertama adalah metode yang sederhana yang hanya berusaha mencari kemungkinan penyelesaian yang disebut juga pencarian buta.
2. Metode yang lebih kompleks yang akan mencari jarak terpendek. Metode ini adalah *British Museum Procedure*, *Branch and Bound*, *Dynamic Programming*, *Best First Search*, *Greedy Search*, A* (A Star) *Search*, dan *Hill Climbing Search*. Metode-metode ini digunakan pada saat harga perjalanan untuk mencari kemungkinan menjadi perhitungan. Beberapa *procedure*/metode yang kita terapkan saat berhadapan dengan musuh. Prosedur ini adalah *minimax search* dan *alpha-beta pruning*. Metode ini banyak digunakan pada program-program seperti catur dan sebagainya.

2.2.4 Algoritma A*

Algoritma A* (*A Star*) adalah algoritma pencarian yang merupakan pengembangan dari algoritma *Best First Search* (BFS). Seperti halnya pada BFS, untuk menemukan solusi, A* juga 'dituntun' oleh fungsi heuristik, yang menentukan urutan titik mana yang akan dikunjungi terlebih dahulu. Heuristik merupakan penilai yang memberi harga pada tiap verteks yang memandu A* mendapatkan solusi yang diinginkan[1].

Algoritma ini pertama kali diperkenalkan pada 1968 oleh Peter Hart, Nils Nilsson, dan Bertram Raphael Dalam tulisan mereka, algoritma ini dinamakan algoritma A. Dengan penggunaan fungsi heuristik yang tepat pada algoritma ini yang dapat memberikan hasil yang optimal, maka algoritma inipun disebut A*.

Dengan fungsi heuristik Algoritma ini membangkitkan verteks yang paling mendekati solusi. Verteks ini kemudian disimpan suksesornya ke dalam *list* sesuai dengan urutan yang paling mendekati solusi terbaik. Kemudian, verteks pertama pada *list* diambil, dibangkitkan suksesornya dan kemudian suksesor ini disimpan ke dalam *list* sesuai dengan urutan yang terbaik untuk solusi. *List* verteks ini disebut dengan verteks terbuka *(open node)* [14].

Pencarian menggunakan algoritma A* mempunyai prinsip yang sama dengan algoritma BFS, hanya saja dengan dua faktor tambahan.

1. Setiap sisi mempunyai "*cost*" yang berbeda-beda, seberapa besar *cost* untuk pergi dari satu simpul ke simpul yang lain.
2. *Cost* dari setiap simpul ke simpul tujuan bisa diperkirakan. Ini membantu pencarian, sehingga lebih kecil kemungkinan kita mencari ke arah yang salah.

Cost untuk setiap simpul tidak harus berupa jarak. *Cost* bisa saja berupa waktu bila kita ingin mencari jalan dengan waktu tercepat untuk dilalui. Sebagai contoh, bila kita berkendaraan melewati jalan biasa bisa saja merupakan jarak terdekat, tetapi melewati jalan tol biasanya memakan waktu lebih sedikit.

Cost antara simpul adalah jaraknya, dan perkiraan *cost* dari suatu simpul ke simpul tujuan adalah penjumlahan jarak dari simpul tersebut ke simpul tujuan. Atau agar lebih mudahnya bisa ditunjukkan seperti berikut ini.

$$f(n) = g(n) + h(n)$$

Persamaan II.1. Algoritma A* [14].

dengan :

$f(n)$ = fungsi evaluasi

$g(n)$ = biaya *(cost)* yang sudah dikeluarkan dari keadaan sampai keadaan n

$h(n)$ = estimasi biaya untuk sampai pada suatu tujuan mulai dari n

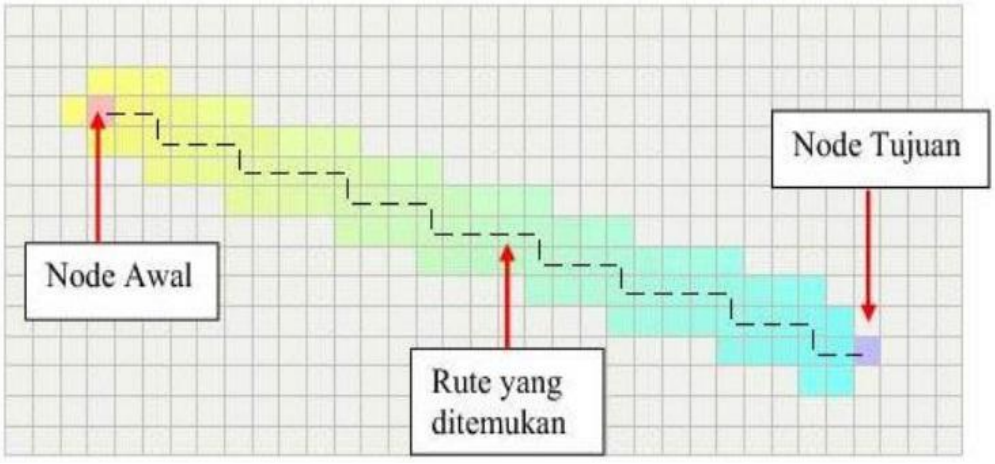
Node dengan nilai terendah merupakan solusi terbaik untuk diperiksa pertama kali pada $g(n) + h(n)$. Dengan fungsi *heuristic* yang memenuhi kondisi tersebut, maka pencarian dengan algoritma A* dapat optimal [14].

Metode A* dapat melakukan *backtracking* jika jalur yang ditempuh ternyata salah. Metode A* dapat melakukannya karena menyimpan jejak / *track* yang mungkin sebagai jalur yang optimal. Sebagai contoh, jika kita sedang menuju suatu kota dan sampai pada persimpangan jalan, dan memutuskan untuk belok kiri daripada ke kanan, dan ternyata bila jalan yang dipilih ternyata salah,

kita akan kembali ke persimpangan dan mengambil jalan satunya. Itulah yang dilakukan metode A* ini.

2.2.5 Fungsi *Heuristic*

BFS dan A* sebagai algoritma pencarian yang menggunakan fungsi *heuristic* untuk menuntun pencarian rute, khususnya dalam hal pengembangan dan pemeriksaan *node-node* pada peta. Dalam aplikasi ini, fungsi *heuristic* yang dipakai untuk pencarian rute mengisi nilai/notasi h pada algoritma BFS dan A*. Ada beberapa fungsi *heuristic* umum yang bisa dipakai untuk algoritma BFS dan A* ini. Salah satunya adalah yang dikenal dengan istilah "*Manhattan Distance*". Fungsi *heuristic* ini digunakan untuk kasus dimana pergerakan pada peta hanya lurus (horizontal atau vertikal), tidak diperbolehkan pergerakan diagonal[14].



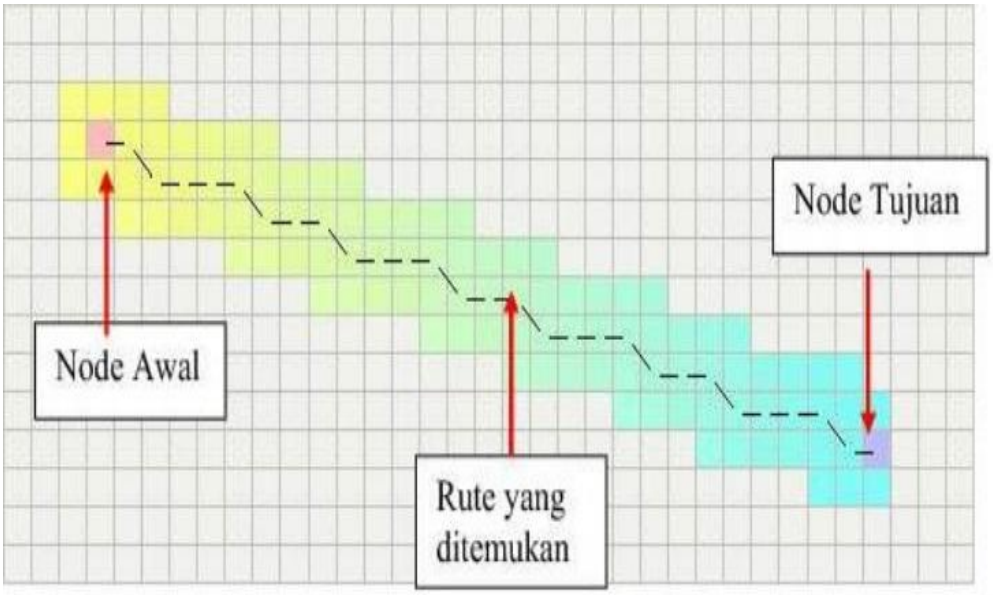
Gambar 2.2 Rute dengan langkah diagonal tidak diperbolehkan[14]

Perhitungan nilai *heuristic* untuk *node* ke-n menggunakan *Manhattan Distance* adalah sebagai berikut :

$$h(n) = (abs(n.x - goal.x) + abs(n.y - goal.y))$$

Dimana h(n) adalah nilai *heuristic* untuk node n, dan *goal* adalah node tujuan.

Jika pergerakan diagonal pada peta diperbolehkan, maka digunakan fungsi *heuristic* selain *Manhattan Distance*. Untuk mendekati kenyataan, *cost* untuk perpindahan *node* secara diagonal dan orthogonal dibedakan. *Cost* diagonal adalah 1,4 kali *cost* perpindahan secara orthogonal.



Gambar 2.3 Rute dengan langkah diagonal diperbolehkan[14]

Maka fungsi heuristic yang digunakan adalah sebagai berikut:

```
h_diagonal(n) = min(abs(n.x - goal.x) + abs(n.y – goal.y))
h_orthogonal(n) = (abs(n.x - goal.x) + abs(n.y – goal.y))
h(n) = h_diagonal(n) + (h_orthogonal (n) – (2 * h_diagonal(n)))
```

Dimana h_diagonal(n) adalah banyaknya langkah diagonal yang bisa diambil untuk mencapai goal dari node n. h_orthogonal adalah banyaknya langkah lurus yang bisa diambil untuk mencapai goal dari node n.

Nilai *heuristic* kemudian diperoleh dari h_diagonal(n) ditambah dengan selisih h_orthogonal(n) dengan dua kali h_diagonal(n). Dengan kata lain, jumlah langkah diagonal kali cost diagonal ditambah jumlah langkah lurus yang masih bisa diambil dikali cost pergerakan lurus. [14].

2.2.6 *Pathfinding*

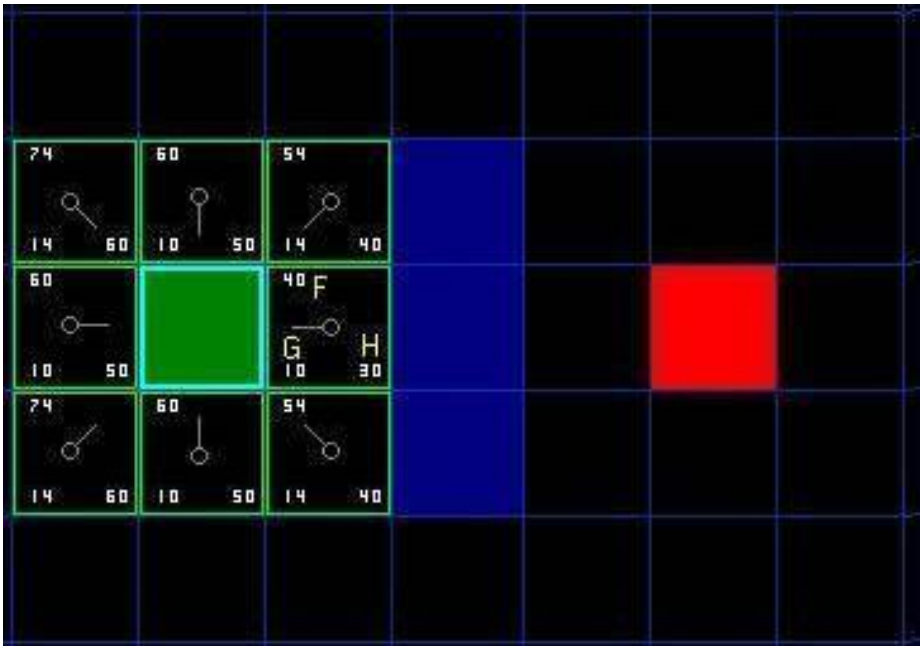
Pathfinding (pencarian jalan/*rute*) adalah salah satu bidang penerapan yang sering ditangani oleh kecerdasan buatan khususnya dengan menggunakan algoritma pencarian. Penerapan yang dapat dilakukan dengan *pathfinding* antara lain adalah pencarian rute dalam suatu *game* dan pencarian jalan/rute pada suatu peta. Algoritma pencarian yang dipakai harus dapat mengenali jalan dan elemen peta yang tidak dapat dilewati.

Bagaimana *Pathfinding* ini bekerja? Pertama-tama peta permainan harus diolah atau diproses terlebih dahulu sebelum algoritma A * dapat bekerja. Hal ini akan melibatkan pemutusan peta ke titik yang berbeda atau lokasi, yang disebut *node*. *Node* ini digunakan untuk mencatat kemajuan pencarian. Selain memegang peta setiap *node* memiliki tiga atribut lainnya yaitu fungsi , tujuan dan heuristik

yang umum dikenal sebagai f, g,dan h. Nilai yang berbeda dapat diberikan ke jalur antara *node*, biasanya nilai-nilai ini akan mewakili jarak antara *node*. Atribut g, h, dan f didefinisikan sebagai berikut:

1. g adalah biaya yang didapatkan dari *node* awal ke *node* saat ini yaitu jumlah dari semua nilai di jalan antara awal dan *node* saat ini.
2. h singkatan heuristik yang merupakan perkiraan biaya dari *node* saat ini ke tujuan *node* (biasanya jarak garis lurus dari *node* ini ke tujuan).
3. f adalah jumlah dari g dan h dan merupakan estimasi terbaik dari biaya jalan akan melalui *node* saat ini. Intinya semakin rendah nilai f yang akan lebih efisien.

Untuk lebih jelasnya ilustrasi *pathfinding* yang digunakan musuh untuk mengejar pemain menggunakan Algoritma A* dapat dilihat pada gambar dibawah ini[6] :



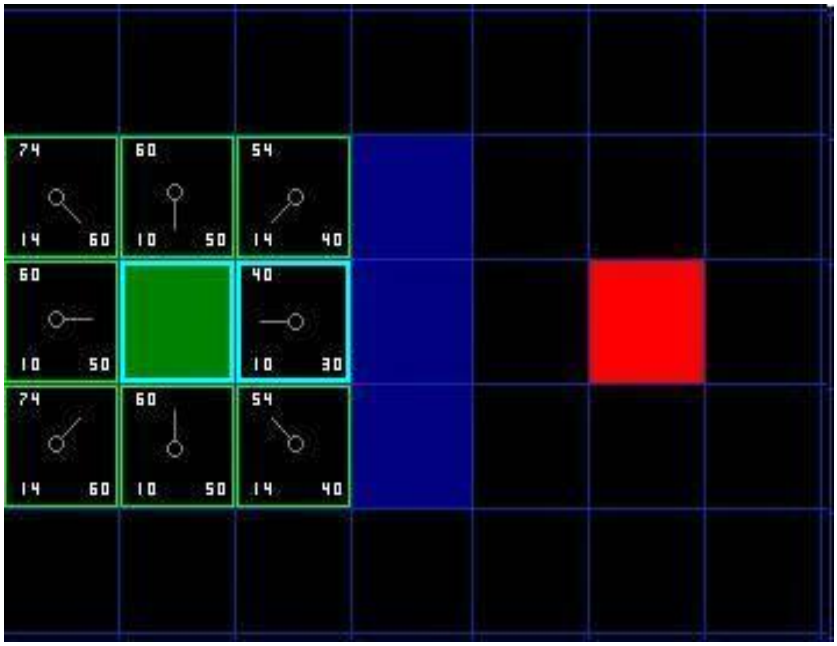
Gambar 2.4 *Pathfinding* Algoritma A*[13]

Pada gambar 2.4 diatas kotak berwarna hijau adalah *node* awal yang di ilustrasikan sebagai posisi awal musuh sedangkan kotak berwarna merah adalah tujuan yang merupakan posisi pemain. Tiga buah kotak biru merupakan penghalang yang tidak dapat dilewati oleh musuh. Angka-angka dalam kotak yang berada pada kiri atas adalah nilai f dan yang berada di kanan bawah adalah nilai h. gambar lingkaran ditengah kotak menunjukkan *parent* kotak tersebut. Untuk pergerakan kearah kanan vertikal *cost*-nya adalah sepuluh, sedangkan pergerakan diagonal kanan memakan *cost* sebesar empat belas. Angka pada kanan bawah adalah h(n) yaitu harga estimasi dihitung dari posisi akhir. Simpul di sebelah kanan simpul awal memiliki h(n) = 30. Perhatikan pula bahwa jarak antara simpul tersebut dengan simpul tujuan adalah 3 kotak horizontal. Sehingga harganya adalah 3 kali harga pergerakan horizontal, yaitu 3*10 = 30. Sedangkan h(n) di simpul sebelah kanan bawah adalah 40. Perhatikan bahwa jarak antara simpul ini dengan simpul tujuan adalah 3 kotak horizontal dan 1 kotak vertikal. Sehingga harga h(n)-nya adalah 3*10 + 1*10 = 40. Nilai yang tertera di bagian kiri-atas adalah f(n) = g(n) + h(n). Panah pada bagian tengah dari kotak menunjuk pada simpul yang menjadi *parent* dari simpul tersebut. Kedelapan simpul di atas kemudian dimasukkan ke *list* terbuka / *OPEN LIST*[6].

Selanjutnya kita akan memilih simpul dengan nilai f(n) terkecil dari *list* terbuka. Terhadap simpul yang dipilih kita lakukan :

1. Mengeluarkan simpul tersebut dari *list* terbuka dan memasukkan ke *list* tertutup

2. Cek semua simpul yang terhubung langsung dengan simpul yang dipilih. Masukkan ke dalam *list* terbuka apabila simpul yang baru belum ada pada *list* terbuka serta men-*set parent* dari simpul baru tersebut ke simpul yang dipilih
3. Apabila simpul yang dicek sudah terdapat pada *list* terbuka, cek nilai g(n) nya. Apabila nilai g(simpul dipilih) + harga untuk bergerak ke simpul dipilih < g(simpul dicek), maka ubah *parent* dari simpul yang dicek ke simpul yang dipilih. Jika tidak, jangan lakukan apa-apa.

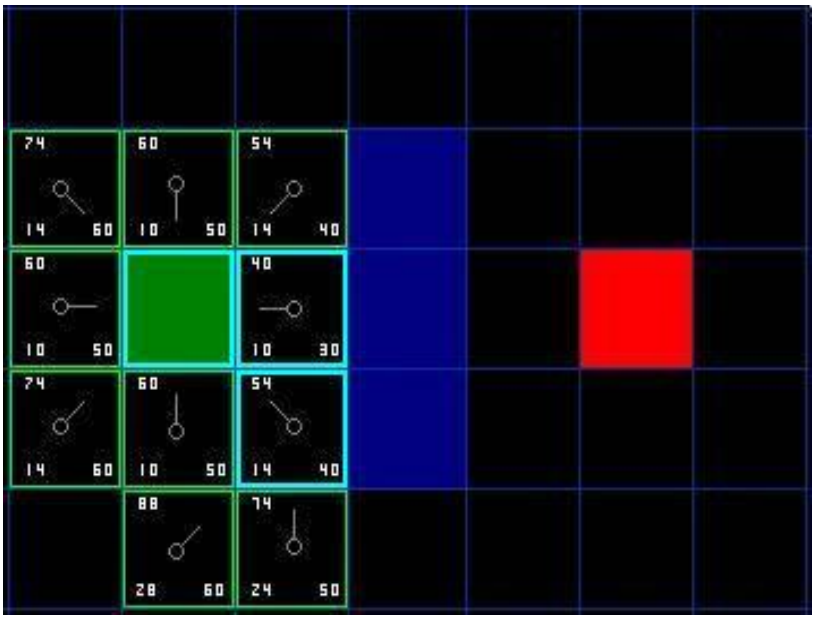


Gambar 2.5 Pathfinding Algoritma A* [11]

Saat ini, simpul yang terpilih adalah simpul di sebelah kanan simpul awal. Kemudian seluruh simpul yang terhubung langsung dicek. Pada kasus simpul di bawah simpul terpilih dicek, kita menemukan bahwa simpul tersebut sudah berada di dalam *list* terbuka. Karena itu, kita akan membandingkan g(n)

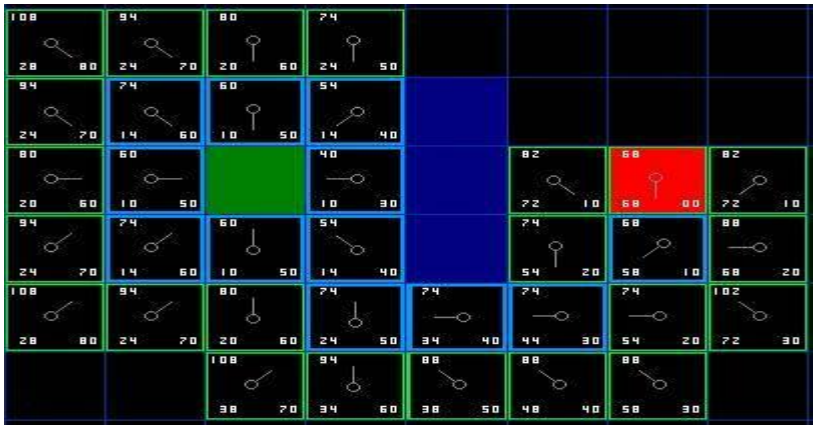
simpul terpilih ditambah harga untuk pergerakan secara vertikal (10) dengan g(n) dari simpul yang dicek. Karena g(simpul terpilih) = 10, harga untuk bergerak vertikal = 10, dan g(simpul dicek) = 14, maka hubungan yang didapatkan adalah 10+10 > 14. Karena nilai g(simpul dicek) lebih kecil, maka kita tidak men-*set* *parent* simpul dicek menjadi simpul terpilih.

Dengan menggunakan cara yang sama, kita mendapatkan bahwa simpul terpilih yang baru adalah simpul di bawah dari simpul terpilih sebelumnya.



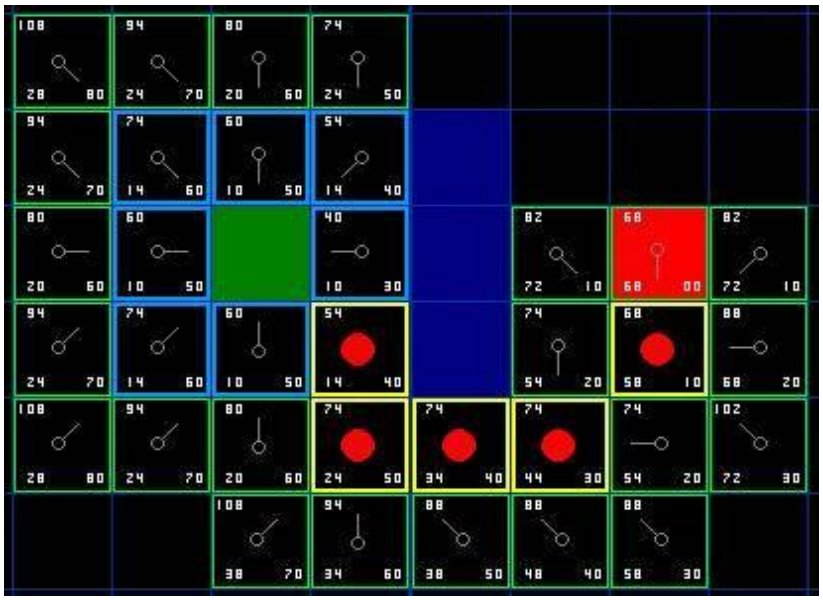
Gambar 2.6 *Pathfinding* Algoritma A* [11]

Simpul yang terpilih kembali melakukan pengecekan terhadap semua simpul yang terhubung. Di sini kita definisikan bahwa simpul tidak dapat terhubung secara diagonal apabila berada di dekat penghalang. Apabila kita mengulangi cara ini terus menerus, kita akan mendapatkan hasil berikut :



Gambar 2.7 *Pathfinding* Algoritma A* [13]

Selanjutnya kita hanya perlu menelusuri *parent* dari setiap simpul dimulai dari simpul tujuan, Seperti yang ditunjukkan gambar 2.8 dibawah ini :



Gambar 2.8 *Pathfinding* Algoritma A* [13]

Pada gambar 2.8 rute yang ditemukan adalah bujur sangkar berbingkai kuning dengan lingkaran yang berwarna merah. Maka dengan demikian musuh dapat menemukan rute terpendek ke tujuan yaitu mengejar pemain.

2.3 Interaksi Manusia dan Komputer

Interaksi Manusia dan Komputer (IMK) atau *Human-Computer Interaction* (HCI) adalah disiplin ilmu yang berhubungan dengan perancangan, evaluasi, dan implementasi sistem komputer interaktif untuk digunakan oleh manusia, serta studi fenomena-fenomena besar yang berhubungan dengannya[9].

Dalam bukunya berjudul Designing the User Interface-Strategies for Effective Human-Computer Interaction, Shneiderman mengemukakan ada lima faktor manusia terukur yang dapat dijadikan sebagai pusat evaluasi[9], yaitu :

1. Waktu belajar

   Berapa lama waktu yang dibutuhkan oleh *user* untuk mempelajari cara relevan untuk melakukan suatu tugas.

2. Kecepatan kinerja

   Berapa lama waktu yang diperlukan untuk mengerjakan suatu tugas.

3. Tingkat kesalahan

   Berapa banyak kesalahan dan kesalahan-kesalahan seperti apa yang bisa terjadi saat *user* mengerjakan tugas-tugas tersebut.

4. Daya ingat

   Bagaimana kemampuan pemakai mempertahankan pengetahuannya setelah jangka waktu tertentu.

5. Kepuasan subjektif

   Bagaimana kepuasan *user* terhadap berbagai aspek dari sistem.

Selain itu Shneiderman juga mengemukakan 8 (delapan) aturan yang dapat digunakan sebagai petunjuk dasar yang baik untuk merancang suatu *user interface*. Delapan aturan ini disebut dengan *Eight Golden Rules of Interface Design*, yaitu[9]:

1. Berusaha untuk Konsisten

   Urutan konsisten diperlukan pada situasi yang sama. Konsistensi harus digunakan pada *prompt*, menu, layar bantuan, warna tampilan, kapitalisasi, huruf, dan sebagainya.

2. Memungkinkan pengguna untuk menggunakan *shortcut*

   Ada kebutuhan dari pengguna yang sudah ahli untuk meningkatkan kecepatan interaksi, sehingga diperlukan singkatan, tombol fungsi, perintah tersembunyi, dan fasilitas makro.

3. Memberikan umpan balik yang informatif

   Untuk setiap tindakan operator, sebaiknya disertakan suatu sistem umpan balik (*feedback*). Untuk tindakan yang sering dilakukan dan tidak terlalu penting, dapat diberikan umpan balik yang sederhana. Tetapi ketika tindakan merupakan hal yang penting, maka umpan balik sebaiknya lebih substansial. Misalnya muncul suatu suara ketika salah menekan tombol pada waktu input data atau muncul pesan kesalahannya.

4. Merancang dialog untuk menghasilkan suatu penutupan

   Urutan tindakan sebaiknya diorganisir dalam suatu kelompok dengan bagian

awal, tengah, dan akhir. Umpan balik yang informatif akan memberikan indikasi bahwa cara yang dilakukan sudah benar dan dapat mempersiapkan kelompok tindakan berikutnya.

5. Memberikan penanganan kesalahan yang sederhana

   Sedapat mungkin sistem dirancang sehingga pengguna tidak dapat melakukan kesalahan fatal. Jika kesalahan terjadi, sistem dapat mendeteksi kesalahan dengan cepat dan memberikan mekanisme yang sederhana dan mudah dipahami untuk penanganan kesalahan.

6. Memungkinkan pembalikan aksi yang mudah

   Hal ini dapat mengurangi kekuatiran pengguna karena pengguna mengetahui kesalahan yang dilakukan dapat dibatalkan, sehingga pengguna tidak takut untuk mengekplorasi pilihan-pilihan lain yang belum biasa digunakan.

7. Mendukung tempat pengendali internal (*internal locus of control*)

   Pengguna ingin menjadi pengontrol sistem dan sistem akan merespon tindakan yang dilakukan pengguna daripada pengguna merasa bahwa sistem mengontrol pengguna. Sebaiknya sistem dirancang sedemikan rupa sehingga pengguna menjadi inisiator daripada responden.

8. Mengurangi beban ingatan jangka pendek

9. Keterbatasan ingatan manusia membutuhkan tampilan yang sederhana atau banyak tampilan halaman yang sebaiknya disatukan, serta diberikan cukup waktu pelatihan untuk kode, *mnemonic*, dan urutan tindakan.

2.4 Rekayasa Perangkat Lunak

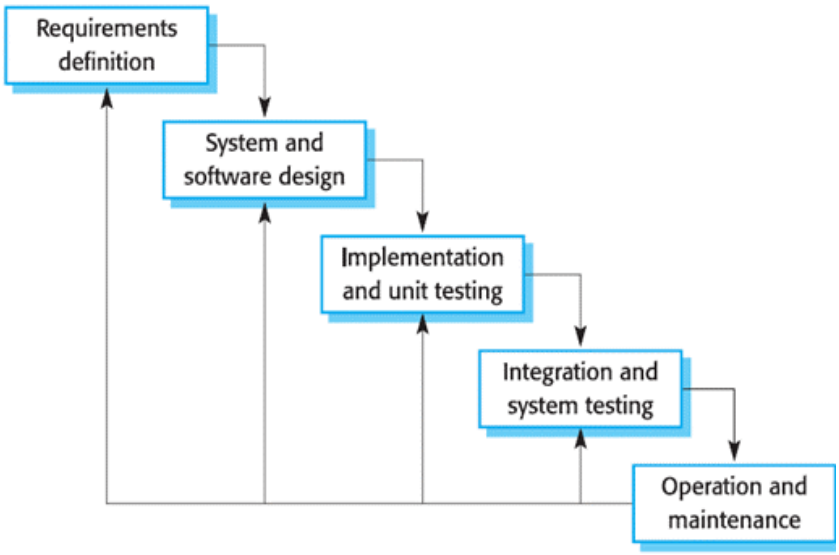
Rekayasa perangkat lunak adalah sebuah teknologi rancangan yang memiliki beberapa lapisan. Dasar dari rekayasa perangkat lunak adalah fokus terhadap kualitas. Setiap pendekatan rekayasa perangkat lunak harus didasari pada komitmen sebuah organisasi terhadap kualitas[5].

Fondasi dari rekayasa perangkat lunak adalah lapisan proses. Proses dari rekayasa perangkat lunak adalah pelekat yang menyatukan semua lapisan teknologi secara utuh dan memberikan pengembangan perangkat lunak komputer secara rasional dan tepat waktu. Prose mendefinisikan sebuah kerangka kerja harus ditetapkan agar rekayasa perangkat lunak menjadi efektif. Prose perangkat lunak membentuk dasar dari pengendalian proyek perangkat lunak dan menetapkan konteks dimana cara-cara teknikal diterapkan, hasil kerja(model-model, dokumentasi, data, laporan, dan lain-lain) dihasilkan.

Ada banyak model pengembangan perangkat lunak, antara lain The *Waterfall Model*, *Joint Application Development* (JAD), *Information Engineering* (IE*), Rapid Application Development* (RAD) termasuk di dalamnya *Prototyping*, *Unified Process* (UP), *Structural Analysis and Design* (SAD) dan *Framework for the Application of System Thinking* (FAST). Salah satu yang akan dijelaskan yaitu tentang model *waterfall*.[5].

Model *waterfall* sebenarnya adalah "*linear sequential model*". Model ini sering disebut dengan "*classic life cycle*" atau model *waterfall*. Model ini adalah model yang muncul pertama kali yaitu sekitar tahun 1970 sehingga sering dianggap kuno, tetapi merupakan model yang paling banyak dipakai

didalam *Software Engineering* (SE). Model ini melakukan pendekatan secara sistematis dan urut mulai dari level kebutuhan sistem lalu menuju ke tahap analisis, desain, *coding*, *testing* / *verification*, dan *maintenance*. Disebut dengan *waterfall* karena tahap demi tahap yang dilalui harus menunggu selesainya tahap sebelumnya dan berjalan berurutan. Sebagai contoh tahap desain harus menunggu selesainya tahap sebelumnya yaitu tahap *requirement.*



Gambar 2.9 Model *Waterfall*[11]

Gambar 2.9 merupakan model *Waterfall*. Berikut adalah penjelasan dari tahap-tahap yang dilakukan di dalam model ini:

a. *Requirements analysis and definition*

   Tahap ini Mengumpulkan kebutuhan secara lengkap kemudian kemudian dianalisis dan didefinisikan kebutuhan yang harus dipenuhi oleh program yang akan dibangun. Fase ini harus

dikerjakan secara lengkap untuk bisa menghasilkan desain yang lengkap..

b. *System and software design*

Tahap ini merupakan kegiatan Tahap ini merupakan kegiatan mengumpulkan kebutuhan secara lengkap kemudian dianalisis dan didefinisikan kebutuhan yang harus dipenuhi oleh aplikasi yang akan dibangun. Tahap ini harus dikerjakan secara lengkap untuk bisa menghasilkan desain yang lengkap. Pada tahap ini juga dilakukan analisis algoritma yang akan dipakai dalam aplikasi *game Death Castle.*

c. *Implementation and unit testing*

Desain program diterjemahkan ke dalam kode-kode dengan menggunakan bahasa pemrograman yang sudah ditentukan. Program yang dibangun langsung diuji baik secara unit.

d. *Integration and system testing*

Penyatuan unit-unit program kemudian diuji secara keseluruhan (*system testing*).

e. *Operation and maintenance*

Mengoperasikan program dilingkungannya dan melakukan pemeliharaan, seperti penyesuaian atau perubahan karena adaptasi dengan situasi sebenarnya[5].

2.5 OOP (*Object Oriented Programming*)

OOP atau dikenal dengan Pemrograman Berorientasi Objek merupakan pemrograman yang semua data dan fungsi di dalam paradigma ini dibungkus kedalam kelas-kelas atau objek-objek.

Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Lebih jauh lagi, pendukung OOP mengklaim bahwa OOP lebih mudah dipelajari dibandingkan dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah kita tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh anggap kita memiliki sebuah departemen yang memiliki manager, sekretaris, petugas administrasi data dan lainnya. Misal manager tersebut ingin memperoleh data dari bag administrasi maka manager tersebut tidak harus mengambilnya langsung tetapi dapat menyuruh petugas bagian administrasi untuk mengambilnya. Pada kasus tersebut seorang manager tidak harus mengetahui bagaimana cara mengambil data tersebut tetapi manager bisa mendapatkan data tersebut melalui objek petugas administrasi. Jadi untuk menyelesaikan suatu masalah dengan kolaborasi antar objek-objek yang ada karena setiap objek memiliki deskripsi tugasnya sendiri [2]. Pemrograman orientasi-objek menekankan konsep berikut:

1. Kelas (*Class*) adalah kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebuah class adalah dasar dari modularitas dan struktur dalam pemrograman berorientasi objek. Sebuah *class* secara tipikal sebaiknya dapat dikenali oleh seorang *non-programmer* sekalipun terkait dengan domain permasalahan yang ada, dan kode yang terdapat dalam sebuah *class* sebaiknya (relatif) bersifat mandiri dan independen (sebagaimana kode tersebut digunakan jika tidak menggunakan OOP). Dengan modularitas, struktur dari sebuah program akan terkait dengan aspek-aspek dalam masalah yang akan diselesaikan melalui program tersebut.
2. Objek (*Object*) adalah membungkus data dan fungsi bersama menjadi suatu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.
3. Abstraksi (*Abstract*) adalah Kemampuan sebuah program untuk melewati aspek informasi yang diproses olehnya, yaitu kemampuan untuk memfokus pada inti. Setiap objek dalam sistem melayani sebagai model dari "pelaku" abstrak yang dapat melakukan kerja, laporan dan perubahan keadaannya, dan berkomunikasi dengan objek lainnya dalam sistem, tanpa mengungkapkan bagaimana kelebihan ini diterapkan. Proses, fungsi atau metode dapat juga dibuat abstrak, dan beberapa teknik digunakan untuk mengembangkan sebuah pengabstrakan.
4. Enkapsulasi (*Encapsulation*) adalah Memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak; hanya metode dalam objek tersebut yang diberi ijin untuk

mengakses keadaannya. Setiap objek mengakses *interface* yang menyebutkan bagaimana objek lainnya dapat berinteraksi dengannya. Objek lainnya tidak akan mengetahui dan tergantung kepada representasi dalam objek tersebut.

5. Polimorfisme (*Polimorfism*) melalui pengiriman pesan. Tidak bergantung kepada pemanggilan subrutin, bahasa orientasi objek dapat mengirim pesan, metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesa tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor singa menerima pesan yang sama, dia akan menggerakkan kakinya dan berlari. Keduanya menjawab sebuah pesan yang sama, namun yang sesuai dengan kemampuan hewan tersebut. Ini disebut polimorfisme karena sebuah variabel tunggal dalam program dapat memegang berbagai jenis objek yang berbeda selagi program berjalan, dan teks program yang sama dapat memanggil beberapa metode yang berbeda di saat yang berbeda dalam pemanggilan yang sama. Hal ini berlawanan dengan bahasa fungsional yang mencapai polimorfisme melalui penggunaan fungsi kelas-pertama.
6. Inheritas (*Inheritance*) adalah Mengatur polimorfisme dan enkapsulasi dengan mengijinkan objek didefinisikan dan diciptakan dengan jenis khusus dari objek yang sudah ada, objek-objek ini dapat membagi (dan memperluas) perilaku mereka tanpa haru mengimplementasi ulang perilaku tersebut (bahasa berbasis-objek tidak selalu memiliki inheritas).

2.6 UML (*Unified Modeling Language*)

Unified Modeling Language (UML) adalah himpunan struktur dan teknik untuk pemodelan desain program berorientasi objek (OOP) serta aplikasinya. UML adalah metodologi untuk mengembangkan sistem OOP dan sekelompok perangkat *tool* untuk mendukung pengembangan sistem tersebut. UML mulai diperkenalkan oleh *Object Management Group*, sebuah organisasi yang telah mengembangkan model, teknologi, dan standar OOP sejak tahun 1980-an. Sekarang UML sudah mulai banyak digunakan oleh para praktisi OOP.

UML adalah suatu bahasa yang digunakan untuk menentukan, memvisualisasikan, membangun, dan mendokumentasikan suatu sistem informasi. UML dikembangkan sebagai suatu alat untuk analisis dan desain berorientasi objek oleh Grady Booch, Jim Rumbaugh, dan Ivar Jacobson. Namun demikian UML dapat digunakan untuk memahami dan mendokumentasikan setiap sistem informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan sistem.

a. Kegunaan UML

Adapun kegunaan UML menurut Fowler [2] adalah sebagai berikut :

1. UML sebagai bahasa visualisasi digunakan untuk merancang suatu model yang dapat dibaca oleh banyak orang dengan pengertian yang sama.

2. UML sebagai bahasa pendefinisian digunakan untuk mendefinisikan dengan rinci seluruh hasil analisis, desain, dan implementasi yang harus dilakukan dalam pengembangan sistem.
3. UML sebagai bahasa perancangan digunakan untuk merancang model yang dapat dikembangkan oleh bahasa pemrograman yang berbeda-beda.
4. UML sebagai bahasa dokumentasi digunakan untuk mendokumentasikan arsitektur sistem beserta perinciannya, unsur-unsur yang dibutuhkan dalam pengembangannya, serta perencanaan dan implemantasi proyek secara keseluruhan dengan simbol-simbol yang mudah dimengerti.

b. Struktur UML

Adapun unsur-unsur pada UML terdiri dari :

1. Objek, yang merupakan abstraksi dari elemen-elemen model.
2. Hubungan, yang menjadi alat penyatu bagi objek-objek yang ada.
3. Diagram, yang mengelompokkan objek-objek dan hubungannya dalam kelompok-kelompok yang mudah dibaca.

Selain memiliki unsur-unsur, UML juga memiliki aturan-aturan yang berfungsi untuk mengatur bentuk sebuah model yang baik. Sebuah model dinyatakan telah dirancang dengan baik jika model tersebut konsisten secara semantik dan selaras dengan model-model lain yang terkait.

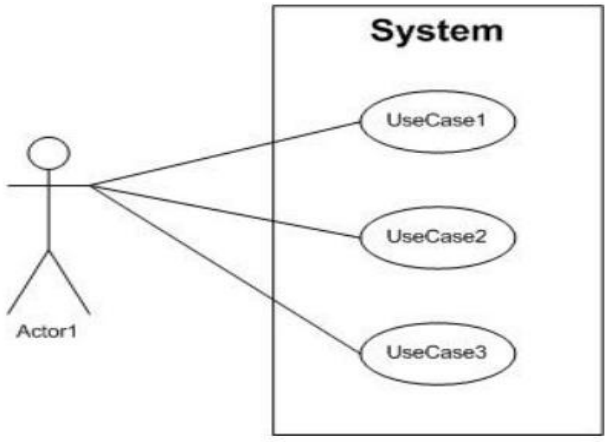
c. Tipe Diagram UML

UML Sebuah diagram merupakan presentasi grafis dari suatu set elemen, paling sering diterjemahkan sebagai *graph* terhubung dari simpul(benda) dan lengkungan (hubungan).[2]

Adapun tipe dari diagram UML adalah sebagai berikut :

1. *Use Case Diagram*

   Diagram yang menunjukkan suatu set penggunaan kasus beserta aktornya (suatu kelas spesial), beserta hubungannya. *Use case* diagram mengalamatkan suatu pandangan kasus statis pada suatu sistem. Diagram ini sangat penting dalam menyusun dan membuat model tingkah laku (*behaviour*) dari suatu sistem.

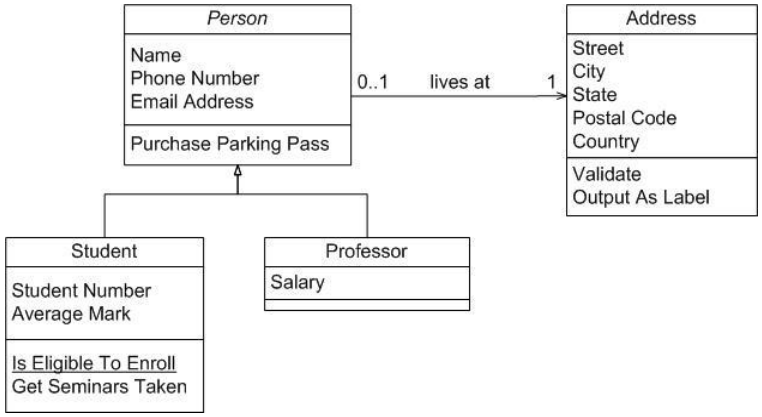


Gambar 2.10 Use Case Diagram [2].

2. *Class Diagram*

   Diagram ini menunjukkan suatu set kelas, antarmuka dan kolaborasi, serta hubungannya. Diagram ini adalah diagram yang umum ditemukan pada sistem berbasis objek. *Class diagram* mengalamatkan desain yang statis dari suatu sistem. *Class*

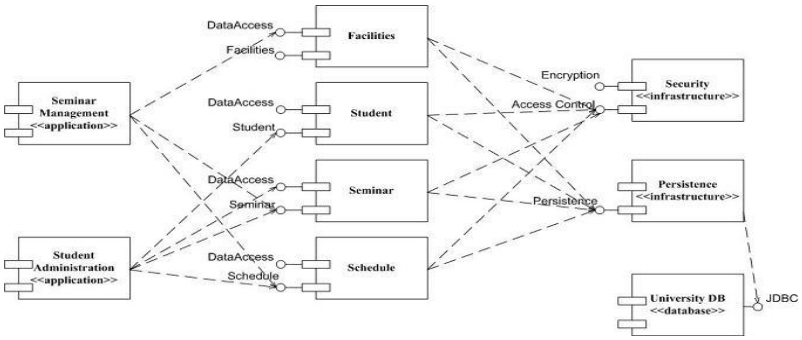
diagram yang berisi kelas aktif, mengalamatkan proses statis dari suatu sistem.



Gambar 2.11 *Class Diagram* [2].

3. *Component Diagram*

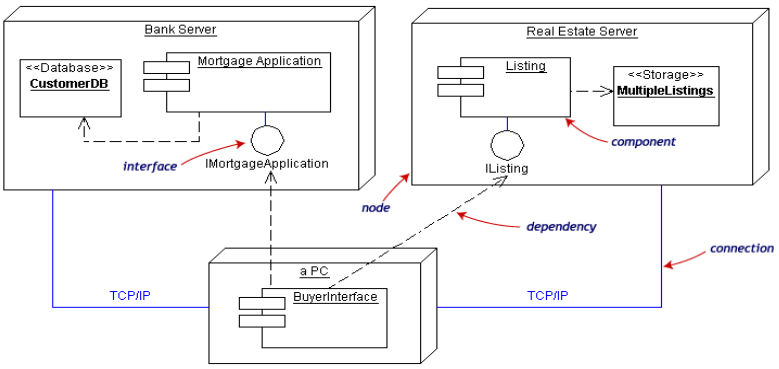
Component diagram menggambarkan struktur dan hubungan antar komponen piranti lunak, termasuk ketergantungan (*dependency*) di antaranya. Komponen piranti lunak adalah modul berisi *code*, baik berisi *source code* maupun *binary code*, baik *library* ataupun *executable*, baik yang muncul pada *compile time*, *link time*, maupun *run time*. Umumnya komponen terbentuk dari beberapa *class* atau *package*, tapi dapat juga dari komponen-komponen yang lebih kecil. Komponen dapat juga berupa *interface*, yaitu kumpulan layanan yang disediakan sebuah komponen untuk komponen lain.



Gambar 2.12 *Component Diagram* [2].

4. *Deployment Diagram*

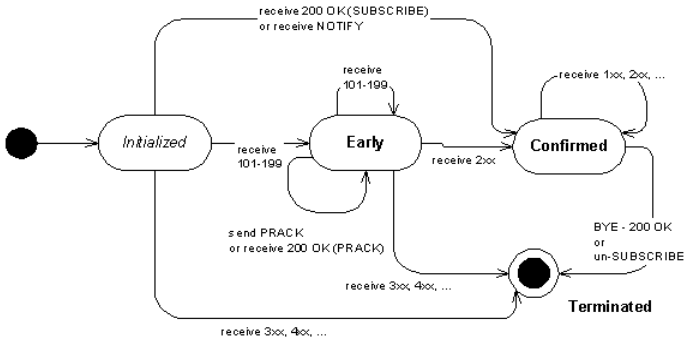
Deployment/physical diagram menggambarkan detail bagaimana komponen di-*deploy* dalam infrastruktur sistem, di mana komponen akan terletak (pada mesin, *server* atau piranti keras apapun), bagaimana kemampuan jaringan pada lokasi tersebut, spesifikasi *server*, dan hal-hal lain yang bersifat fisikal. Sebuah node adalah *server*, *workstation*, atau piranti keras lain yang digunakan untuk men-*deploy* komponen dalam lingkungan sebenarnya [2].



Gambar 2.13 *Deployment Diagram* [2].

5. *State Diagram*

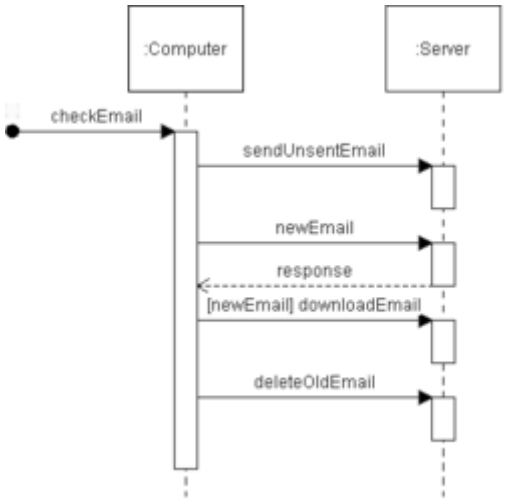
Statechart diagram menggambarkan transisi dan perubahan keadaan (dari satu *state* ke *state* lainnya) suatu objek pada sistem sebagai akibat dari stimulus yang diterima. Pada umumnya *statechart* diagram menggambarkan *class* tertentu (satu *class* dapat memiliki lebih dari satu *statechart* diagram).



Gambar 2.14 *State Chart Diagram* [2].

6. *Sequence Diagram*

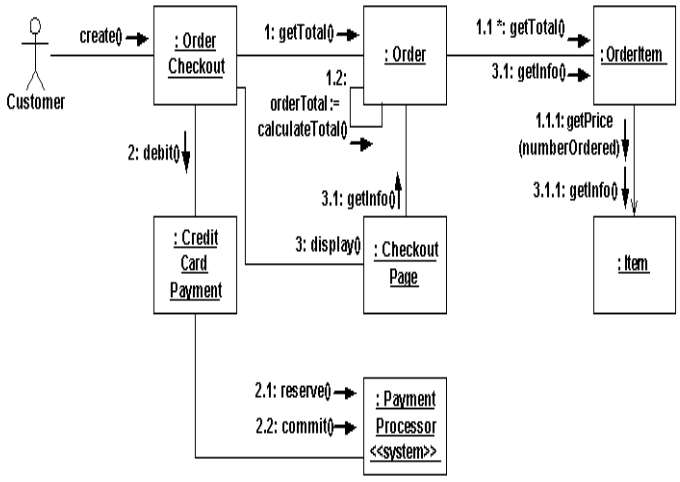
Sequence diagram menggambarkan interaksi antar objek di dalam dan di sekitar sistem (termasuk pengguna, display, dan sebagainya) berupa message yang digambarkan terhadap waktu. Sequence diagram terdiri atar dimensi vertikal (waktu) dan dimensi horizontal (objek-objek yang terkait). *Sequence* diagram biasa digunakan untuk menggambarkan skenario atau rangkaian langkah-langkah yang dilakukan sebagai respons dari sebuah event untuk menghasilkan output tertentu.



Gambar 2.15 *Sequence Diagram* [2].

7. Collaboration Diagram

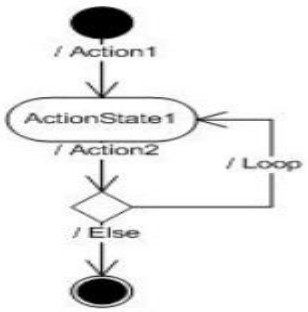
Collaboration diagram juga menggambarkan interaksi antar objek seperti *sequence* diagram, tetapi lebih menekankan pada peran masing-masing objek dan bukan pada waktu penyampaian *message*. Setiap message memiliki *sequence number*, di mana *message* dari level tertinggi memiliki nomor 1. *Messages* dari level yang sama memiliki prefiks yang sama.



Gambar 2.16 *Collaboration Diagram* [2].

8. *Activity Diagram*

Diagram ini adalah jenis spesial dari *statechart* diagram yang menunjukkan alur dari aktifitas ke aktifitas lain dalam suatu sistem. *Activity* diagram mengalamatkan secara dinamis suatu sistem. Diagram ini sangat penting dalam membuat model fungsi dari suatu sistem dan menekankan pada alur control diantara objek - objek yang bersangkutan.



Gambar 2.17 *Activity Diagram* [2].

2.7 Pembangun Perangkat Lunak

2.7.1 Bahasa Pemrograman C#

C# (dibaca: C sharp) merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti Java, Delphi, Visual Basic, dan lain-lain) dengan beberapa penyederhanaan. Menurut standar ECMA-334 C# *Language Specification*, nama C# terdiri atas sebuah huruf Latin C

(U+0043) yang diikuti oleh tanda pagar yang menandakan angka # (U+0023). Tanda pagar # yang digunakan memang bukan tanda kres dalam seni musik (U+266F), dan tanda pagar # (U+0023) tersebut digunakan karena karakter kres dalam seni musik tidak terdapat di dalam keyboard standar

2.7.2 Microsoft Visual C# 2010 Express

Microsoft Visual Studio Express adalah kumpulan *freeware* lingkungan pengembangan terpadu (IDE) yang dikembangkan oleh Microsoft yang merupakan versi ringan dari Microsoft Visual Studio. Gagasan edisi ekspres menurut Microsoft adalah untuk menyediakan efisiensi, mudah digunakan dan mudah dipelajari bagi pengguna IDE selain pengembang perangkat lunak profesional, yaitu seperti penggemar dan mahasiswa.

2.7.3 Framework XNA

XNA merupakan Framework yang dibuat oleh Microsoft khusus untuk pengembang *game* baik PC, XBOX 360, maupun *device* lain seperti Zune dan Windows Phone 7. Kata "XNA" tidak memiliki arti. XNA adalah satu *game* framework yang fiturnya paling banyak dan sudah banyak digunakan oleh para *game developer* dunia. Selain fiturnya yang sudah cukup lengkap dan sangat membantu para *game programmer*, XNA juga *cross-platform* framework, sehingga *game* untuk PC hanya cukup dikompilasi ulang untuk ditaruh di platform lain, misalnya XBOX 360, Zune dan Windows Phone 7. Microsoft XNA Framework adalah bagian dari XNA Game Studio, yaitu tools tambahan pada Microsoft Visual Studio untuk membuat *game* menggunakan XNA. Dengan

XNA kita tak perlu membuat *game* dari awal (*coding* arsitektur). Jadi, sangat mudah tinggal memanfaatkan fungsi dan kelas yang disediakan. Selain itu support XNA sangat besar baik *intern* maupun *extern* Microsoft.