BAB 2
LANDASAN TEORI

2.1. *Game*

Game adalah sesuatu yang dapat dimainkan dengan aturan tertentu sehingga ada yang menang dan ada yang kalah, biasanya dalam konteks tidak serius atau dengan tujuan *refreshing*. Suatu cara belajar yang digunakan dalam menganalisa interaksi antara sejumlah pemain maupun perorangan yang menunjukkan strategi-strategi yang rasional.

"Permainan terdiri atas sekumpulan peraturan yang membangun situasi bersaing dari dua sampai beberapa orang atau kelompok dengan memilih strategi yang dibangun untuk memaksimalkan kemenangan sendiri atau pun untuk meminimalkan kemenangan lawan. Peraturan-peraturan menentukan kemungkinan tindakan untuk setiap pemain, sejumlah keterangan diterima setiap pemain sebagai kemajuan bermain, dan sejumlah kemenangan atau kekalahan dalam berbagai situasi " [6].

2.1.1. Jenis-jenis *Game*

Ada beberapa jenis platform di dunia *game* yang selalu dipilih oleh pengguna *game*s , yaitu [8] :

1. *Arcade games*, yaitu yang sering disebut ding-dong di Indonesia, biasanya berada di daerah / tempat khusus dan memiliki box atau mesin yang memang khusus di design untuk jenis video *game*s 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 tentunya).

2. PC *Games* , yaitu video *game* yang dimainkan menggunakan *Personal Computers.*
3. *Console games*, yaitu video *game*s yang dimainkan menggunakan console tertentu, seperti Playstation 2, Playstation 3, XBOX 360, dan Nintendo Wii.
4. *Handheld games*, yaitu yang dimainkan di *console* khusus video *game* yang dapat dibawa kemana-mana, contoh Nintendo DS dan Sony PSP.
5. *Mobile games*, yaitu yang dapat dimainkan atau khusus untuk *mobile phone* atau PDA.

2.1.2. *Genre Game*

Game dibagi atas beberapa *genre*, diantaranya yaitu [8] :

1. *Action Shooting* (tembak – menembak). Menembak , memukul , bisa juga tusuktusukan, tergantung cerita dan tokoh di dalamnya, video *game* jenis ini sangat memerlukan kecepatan refleks, koordinasi mata-tangan, juga timing, inti dari *game* jenis ini adalah tembak, tembak dan tembak.
   Contoh : CS (*Counter Strike*) dan Crysis.
2. *Fighting* (pertarungan). Ada yang mengelompokan video *game fighting* di bagian Aksi, namun penulis berpendapat berbeda, jenis ini memang memerlukan kecepatan refleks dan koordinasi mata-tangan, tetapi inti dari *game* ini adalah penguasaan jurus (hafal caranya dan lancar mengeksekusinya), pengenalan karakter dan *timing* sangatlah penting, iya, combo-pun menjadi esensial untuk mengalahkan lawan secepat mungkin.
   Contoh : Mortal Kombat dan Tekken.
3. *Adventure* (Petualangan). Memasuki gua bawah tanah, melompati bebatuan di antara lahar, bergelayutan dari pohon satu ke pohon lain, bergulat dengan ular sambil mencari kunci untuk membuka pintu kuil legendaris, atau sekedar mencari telepon umum untuk mendapatkan misi berikutnya, itulah beberapa

dari banyak hal yang karakter pemain harus lakukan dan lalui dalam video *game* jenis ini.

Contoh : Kings Quest, dan Space Quest.

4. *Strategy* (strategi). Video *game* strategi biasanya memberikan pemain atas kendali tidak hanya satu orang tapi minimal sekelompok orang dengan berbagai jenis tipe kemampuan, sampai kendaraan, bahkan hingga pembangunan berbagai bangunan, pabrik dan pusal pelatihan tempur, tergantung dari tema ceritanya. Kebanyakan *game* stategi adalah *game* perang. Contoh : Warcraft.
5. *Simulation* (Simulasi). Video *game* jenis ini seringkali menggambarkan dunia di dalamnya sedekat mungkin dengan dunia nyata dan memperhatikan dengan detil berbagai faktor. Dari mencari jodoh dan pekerjaan, membangun rumah, gedung hingga kota, mengatur pajak dan dana kota hingga keputusan memecat atau menambah karyawan. Dunia kehidupan rumah tangga sampai bisnis membangun konglomerasi, dari jualan limun pinggir jalan hingga membangun laboratorium *cloning*. Video *Game* jenis ini membuat pemain harus berpikir untuk mendirikan, membangun dan mengatasi masalah dengan menggunakan dana yang terbatas.
6. *Puzzle* (teka-teki). Video *game* jenis ini sesuai namanya berintikan mengenai pemecahan teka-teki, baik itu menyusun balok, menyamakan warna bola, memecahkan perhitungan matematika, melewati labirin, sampai mendorong-dorong kota masuk ke tempat yang seharusnya, itu semua termasuk dalam jenis ini. Sering pula *game* jenis ini adalah juga unsur *game* dalam video *game* petualangan maupun *game* edukasi
7. *Sport game* (Olahraga). *Game* ini merupakan adaptasi dari kenyataan, membutuhkan kelincahan dan juga strategi dalam memainkannya. *Game* 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.

8. *RPG (Role Playing Game).* Video *game* jenis ini sesuai dengan terjemahannya, bermain peran, memiliki penekanan pada tokoh/peran perwakilan pemain di dalam *game*, yang biasanya adalah tokoh utamanya, dimana seiring kita memainkannya, karakter tersebut dapat berubah dan berkembang ke arah yang diinginkan pemain dalam berbagai parameter yang biasanya ditentukan dengan naiknya level, baik dari status kepintaran, kecepatan dan kekuatan karakter, senjata yang semakin sakti, ataupun jumlah teman maupun mahluk peliharaan.
9. *Education* (edukasi). *Game* edukasi merupakan paket software yang menciptakan kemampuan pada lingkungan *game* yang diberikan sebagai alat bantu untuk memotivasi atau membantu siswa untuk melalui prosedur *game* secara teliti untuk mengembangkan kemampuannya. *Developer* yang membuatnya, harus memperhitungkan berbagai hal agar *game* ini benar-benar dapat mendidik, menambah pengetahuan dan meningkatkan ketrampilan yang memainkannya. Target segmentasi pemain harus pula disesuaikan dengan tingkat kesulitan dan *design* visual ataupun animasinya.

2.1.3. *Game* Edukasi

Sesuai dengan arti bahasa Indonesia, *Game* berarti permainan, sedangkan edukasi adalah pendidikan.

Game edukasi adalah salah satu genre game yang digunakan untuk memberikan pengajaran / menambah pengetahuan penggunanya melalui suatu media unik dan menarik. Ada beberapa kriteria game edukasi yang ideal yaitu:

1. Rasa ingin tahu, fantasi dan kontrol pengguna
   a. Motivasi intrinsik, menyenangkan untuk digunakan. Pengguna memiliki kontrol terhadap permainan.
   b. Integrasi antara materi edukasi dan aspek fantasi permainan
   c. Mendorong keingintahuan, pengguna dapat melakukan eksplorasi bebas, permainan mengandung rahasia tersembunyi.
   d. Keberhasilan ditentukan oleh pengetahuan, bukan kebetulan.
   e. Simulasi realistik dunia.

f. Materi edukasi disesuaikan dengan materi dunia nyata.

2. Tantangan

a. Tantangan diberikan secara terus menerus tapi disesuaikan dengan tingkat pemahaman pemain.

b. Menyediakan hint dan instruksi untuk membantu pengguna.

3. Pengguna anak-anak dan yang berkebutuhan khusus

a. Gambar, objek, layar tertata rapi.

b. Permainannya sendiri merupakan aktivitas yang penting (*play for the sake of play*).

c. Menginspirasi anak, bahkan setelah komputer dimatikan.

2.2. *Artificial Intelligence*

Kecerdasan buatan sering disebut juga dengan *Artificial Intelligence* (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.

Manusia bisa menjadi pandai dalam menyelesaikan segala permasalahan di dunia ini karena manusia mempunyai pengetahuan dan pengalaman. Pengetahuan dapat diperoleh dari proses belajar. Semakin banyak bekal pengetahuan yang dimiliki oleh seseorang tentu saja diharapkan akan lebih baik dalam menyelesaikan permasalahan. Namun bekal pengetahuan saja tidak cukup, manusia juga diberi akal untuk melakukan penalaran, mengambil kesimpulan berdasarkan pengetahuan dan pengalaman yang mereka miliki. Tanpa memiliki kemampuan menalar yang baik, manusia dengan segudang pengalaman dan pengetahuan tidak akan dapat menyelesaikan masalah dengan baik, demikian pula

dengan kemampuan menalar yang sangat baik, namun tanpa bekal pengetahuan dan pengalaman yang memadai, manusia juga tidak akan bisa menyelesaikan masalah dengan baik.

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 dengan kedua komponen tersebut agar komputer bisa menjadi mesin yang pintar [4] .

2.2.1. Algoritma *Greedy*

Algoritma *Greedy* adalah algoritma yang untuk membentuk solusi langkah perlangkah. Pada setiap langkah tersebut akan dipilih keputusan yang paling optimal. Keputusan tersebut tidak perlu memperhatikan keputusan selanjutnya yang akan diambil, dan keputusan tersebut tidak dapat diubah lagi pada langkah selanjutnya [2].

2.2.2. Prinsip Utama Algoritma *Greedy*

Prinsip utama algoritma *greedy* adalah "*take what you can get now*". Maksud dari prinsip tersebut adalah pada setiap langkah dalam algoritma *greedy*, diambil keputusan yang paling optimal untuk langkah tersebut tanpa memperhatikan konsekuensi pada langkah selanjutnya. Solusi tersebut disebut dengan optimum lokal. Kemudian saat pengambilan nilai optimum lokal pada setiap langkah, diharapkan tercapai optimum global, yaitu tercapainya solusi optimum yang melibatkan keseluruhan langkah dari awal sampai akhir.

2.2.3. Elemen Algoritma *Greedy*

Elemen-elemen yang digunakan dalam penerapan algoritma *greedy* antara lain :

1. Himpunan Kandidat, adalah himpunan yang berisi elemen pembentuk solusi.
2. Himpunan Solusi, adalah himpunan yang terpilih sebagai solusi persoalan.

3. Fungsi Seleksi, adalah fungsi yang memilih kandidat yang paling mungkin untuk mencapai solusi optimal.
4. Fungsi Kelayakan, adalah fungsi yang memeriksa apakah suatu kandidat yang dipilih dapat memberikan solusi yang layak. Maksudnya yaitu apakah kandidat tersebut bersama dengan himpunan solusi yang sudah terbentuk tidak melanggar kendala yang ada.
5. Fungsi Solusi, adalah fungsi yang mengembalikan nilai boolean. *True* jika himpunan solusi yang sudah tebentuk merupakan solusi yang lengkap; *False* jika himpunan solusi belum lengkap.
6. Fungsi Objektif, adalah fungsi yang mengoptimalkan solusi.

2.2.4. Metode *Path finding*

Path finding merupakan metode yang sangat dibutuhkan pada berbagai *game*, terutama *game* 3d. *Path finding* digunakan untuk menentukan arah pergerakan suatu objek dari satu tempat ke tempat lain berdasarkan keadaan peta dan objek lainnya. Dalam pemecahan *path finding* akan dibutuhkan algoritma yang dapat dengan cepat memproses dan menghasilkan arah yang terpendek untuk mencapai suatu lokasi target. Algoritma yang digunakan untuk *path finding* sudah banyak yang ditemukan. Misalnya *Bellman–Ford, Dijkstra, Floyd–Warshall*, dan A *star* [2].

2.2.4.1. Algoritma A* (A *star*)

Algoritma A *star* merupakan perbaikan dari metode BFS (*Best First Search*) dengan memodifikasi fungsi heuristiknya. A *star* akan meminimumkan total biaya lintasan yang terdapat di metode BFS. Pada kondisi yang tepat, A *star* akan memberikan solusi yang terbaik dalam waktu yang optimal. Pada pencarian *rute* kasus sederhana, dimana tidak terdapat halangan pada peta, A *star* bekerja secepat dan seefisien BFS. Pada kasus peta dengan halangan, A *star* dapat menemukan solusi *rute* tanpa terjebak oleh halangan yang ada [2].

Beberapa terminologi dasar yang terdapat pada algoritma A *star* adalah *starting point*, simpul (*nodes*), A, *open list*, *closed list*, harga (*cost*), halangan (*unwalkable*). *Starting point* adalah sebuah terminologi untuk posisi awal sebuah benda. A adalah *node* yang sedang dijalankan dalam algortima pencarian jalan terpendek. *Node* adalah petak-petak kecil sebagai representasi dari area *path finding*. Bentuknya dapat berupa persegi, lingkaran, maupun segitiga. *Open list* adalah tempat menyimpan data *node* yang mungkin diakses dari *starting point* maupun simpul yang sedang dijalankan. *Closed list* adalah tempat menyimpan data simpul sebelum A yang juga merupakan bagian dari jalur terpendek yang telah berhasil didapatkan.

Harga (F) adalah nilai yang diperoleh dari penjumlahan nilai G, jumlah nilai tiap simpul dalam jalur terpendek dari *starting point* ke A, dan H, jumlah nilai perkiraan dari sebuah simpul ke simpul tujuan. Simpul tujuan adalah simpul yang akan dituju. Rintangan adalah sebuah atribut yang menyatakan bahwa sebuah simpul tidak dapat dilalui oleh A.

A *star* memiliki 2 fungsi utama dalam menentukan solusi terbaik. Fungsi pertama disebut sebagai g(n) merupakan fungsi yang digunakan untuk menghitung total *cost* yang dibutuhkan dari *starting point* menuju *node* tertentu. Fungsi kedua yang biasa disebut sebagai h(n) merupakan fungsi perkiraan total *cost* yang diperkirakan dari suatu *node* ke *node* akhir.

Pada A *star*, setiap *node* dari *node* awal ditelusuri kemudian dihitung *cost* dari tiap-tiap *node* dan dimasukkan ke tabel prioritas. *Node* dengan *cost* paling rendah akan diberikan tingkat prioritas paling tinggi. Kemudian pencarian dilanjutkan pada *node* dengan nilai prioritas tertinggi pada tabel.

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

dengan :

n = posisi koordinat *node*
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

Nilai F adalah *cost* perkiraan suatu *node* yang teridentifikasi. Nilai F merupakan hasil dari f(n). Nilai G hasil dari fungsi g(n), adalah banyaknya langkah yang diperlukan untuk menuju ke *node* sekarang. Setiap *node (node)* harus memiliki informasi nilai h(n), yaitu estimasi harga *node* tersebut dihitung dari *node* tujuan yang hasilnya menjadi nilai H.

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

2.2.4.2. Fungsi Heuristik *Manhattan Distance*

Heuristik adalah sebuah teknik yang mengembangkan efisiensi dalam proses pencarian, namun dengan kemungkinan mengorbankan kelengkapan (*completeness*). Fungsi Heuristik digunakan untuk mengevaluasi keadaan-keadaan problema individual dan menentukan seberapa jauh hal tersebut dapat digunakan untuk mendapatkan solusi yang diinginkan [10].

A *star* sebagai algoritma pencarian yang menggunakan fungsi heuristik untuk pencarian rute *node-node* pada peta. Ada beberapa fungsi heuristik umum yang bisa dipakai untuk algoritma A *star* ini, salah satunya adalah *Manhattan distance.* Rumus dari heuristik *Manhattan distance* adalah sebagai berikut :

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

Dimana h(n) adalah sebuah fungsi heuristik. D (*distance*) adalah jarak dari *note* awal (*starting point*) menuju *note* tujuan (*goal*).

2.3. Aplikasi Pembangun perangkat lunak

Dalam membangun aplikasi *game puzzle and the solver ini p*erangkat lunak yang digunakan adalah sebagai berikut :

2.3.1. Unity 3D

Unity 3d adalah sebuah *software development* yang terintegrasi untuk menciptakan video *game* atau konten lainnya seperti visualisasi arsitektur atau *real-time* animasi 3D. Unity 3D dapat digunakan pada microsoft Windows dan MAC OS X, dan permainan yang dihasilkan dapat dijalankan pada Windows, MAC, Xbox 360, OlayStation 3, Wii, iPad, iPhone, Android dan Linux. Unity 3D juga dapat menghasilkan permainan untuk *browser* dengan menggunakan *plugin* Unity Web Player. Unity 3D juga memiliki kemampuan untuk mengekspor permainan yang dibangun untuk fungsionalitas Adobe Flash 3D.

2.3.2. Microsoft Visual Studio 2010

Microsoft Visual Studio merupakan sebuah perangkat lunak lengkap (*suite*) yang dapat digunakan untuk melakukan pengembangan aplikasi, baik itu aplikasi bisnis, aplikasi personal, ataupun komponen aplikasinya, dalam bentuk aplikasi *console*, aplikasi Windows, ataupun aplikasi Web. Visual Studio mencakup *kompiler*, SDK, Integrated Development Environment (IDE), dan dokumentasi (umumnya berupa MSDN Library). Kompiler yang dimasukkan ke dalam paket Visual Studio antara lain Visual C++, Visual C#, Visual Basic, Visual Basic .NET, Visual InterDev, Visual J++, Visual F#, Visual FoxPro, dan Visual SourceSafe. Microsoft Visual Studio dapat digunakan untuk mengembangkan aplikasi dalam *native code* (dalam bentuk bahasa mesin yang berjalan di atas Windows) ataupun *managed code* (dalam bentuk Microsoft *Intermediate Language* di atas .NET Framework). Selain itu, Visual Studio juga dapat digunakan untuk mengembangkan aplikasi Silverlight, aplikasi Windows *Mobile* (yang berjalan di atas .NET Compact Framework).

Visual Studio kini telah menginjak versi Visual Studio 2010, yang ditujukan untuk *platform* Microsoft .NET Framework 4.0. Versi sebelumnya Visual Studio 2008 yang ditujukan untuk *platform* Microsoft .NET Framework 3.5. Visual Studio 2005 ditujukan untuk *platform* .NET Framework 2.0 dan 3.0. Visual Studio 2003 ditujukan untuk .NET Framework 1.1, dan Visual Studio 2002

ditujukan untuk .NET Framework 1.0. Versi-versi tersebut di atas kini dikenal dengan sebutan Visual Studio .NET, karena memang membutuhkan Microsoft .NET Framework. Sementara itu, sebelum muncul Visual Studio .NET, terdapat Microsoft Visual Studio 6.0 (VS1998).

2.3.3. 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.

2.3.4. Autodesk MAYA 2011

Maya pada mulanya merupakan generasi animasi produk yang sedang dikembangkan di Research Alias, Inc berdasarkan kode dari The Visualizer Advanced, PowerAnimator dan Alias Sketch!. Kode adalah *porting* ke IRIX dan fitur animasi yang ditambahkan. Nama kode untuk proyek ini *port* adalah Maya. Walt Disney bekerja sama erat dengan pengembangan Maya selama produksi dari Dinosaurus. Disney meminta agar *User interface* dari aplikasi menjadi disesuaikan sehingga alur kerja pribadi bisa dibuat. Ini adalah pengaruh tertentu dalam arsitektur terbuka Maya, dan sebagian bertanggung jawab untuk itu menjadi begitu populer di industri.

Setelah Silicon Graphics Inc mengakuisisi kedua Alias Wavefront dan Teknologi, Inc, generasi teknologi Wavefront itu (waktu itu dalam

pengembangan) dilebur ke dalam Maya. Akuisisi SGI merupakan tanggapan terhadap Microsoft *Corporation* memperoleh Softimage, Co. Anak perusahaan yang dimiliki sepenuhnya baru bernama "Alias | Wavefront" .

Pada awal pembangunan, Maya dimulai dengan Tcl sebagai bahasa *scripting*, dalam rangka untuk meningkatkan kesamaannya dengan bahasa shell Unix. Tapi setelah merger dengan Wavefront Sophia, bahasa *scripting* dalam Dynamation Wavefront, dipilih sebagai dasar MEL (*Maya embedded language*).

Maya 1.0 dirilis pada Februari 1998. Setelah serangkaian akuisisi, Maya dibeli oleh Autodesk pada tahun 2005. Di bawah nama perusahaan induk baru, diubah namanya menjadi Autodesk Maya. Namun, nama "Maya" terus menjadi nama yang dominan digunakan untuk produk.

2.3.5. Google SketchUp

SketchUp ini dikembangkan oleh perusahaan *startup* @ Last Software, Boulder, Colorado yang dibentuk pada tahun 1999. SketchUp pertama kali dirilis pada bulan Agustus 2000 sebagai tujuan umum alat pembuatan konten 3D. Aplikasi ini memenangkan penghargaan *Community Choice Award* di sebuah pameran pada tahun 2000. Kunci keberhasilan awal adalah masa belajar yang lebih pendek daripada alat 3D lainnya. Pada tanggal 14 Maret 2006, Google mengakuisisi @ Last Software, karena Google tertarik buat plugin untuk Google Earth.

Pada 9 Januari 2007, SketchUp 6 dirilis, yang menampilkan alat-alat baru serta versi beta Google SketchUp Layout. Vektor 2D Layout termasuk peralatan, serta alat-alat tata letak halaman dimaksudkan untuk memudahkan bagi para profesional untuk membuat presentasi tanpa berkerjasama dengan pihak ketiga program presentasi. Pada tanggal 9 Februari 2007, sebuah pembaruan dirilis. Ini mengoreksi beberapa bug, tetapi tidak membawa fitur baru.

Pada 17 November 2008, SketchUp 7 sudah diluncurkan, dengan kemudahan penggunaan, integrasi SketchUp's Komponen *Browser* dengan Google

3D Warehouse, Layout 2 komponen dinamis yang merespon tepat untuk scaling dan peningkatan kinerja API Ruby.

Google SketchUp

Pada tanggal 27 April 2006, Google mengumumkan Google SketchUp, yang bebas-download versi SketchUp. Versi gratis ini beda dengan versi SketchUp *Pro*, tetapi terpadu mencakup alat untuk meng-*upload* konten ke Google Earth dan Google 3D Warehouse, repositori model dibuat dalam SketchUp. Mereka juga menambahkan kotak peralatan baru di mana Anda dapat berjalan, melihat segala sesuatu dari sudut pandang seseorang, label untuk model, melihat-lihat alat, dan "Setiap poligon" bentuk alat. Sedangkan versi gratis dari Google Sketchup 3D hanya dapat mengekspor ke SKP dan Google Earth. Kmz format file, versi Pro bisa mengekspor dan memasukkan .3 ds,. Dae,. DWG,. DXF,. Fbx,. Obj,. XSI , dan. wrl format file. Google SketchUp juga dapat menyimpan "*screenshot*" dari model. Bmp,. Png,. Jpg,. Tif, dengan versi *Pro* juga mendukung. Pdf,. Eps,. Epx,. DWG, dan. DXF. Informasi lokasi GPS selalu disimpan dalam file KMZ. Bangunan desain sendiri dapat disimpan dalam SKP.

2.4. Analisis sistem Berorientasi Objek

Analisis dan desain berorientasi objek adalah cara baru dalam memikirkan suatu masalah dengan menggunkan model yang dibuat menurut konsep sekitar dunia nyata. Dasar pembuatan adalah objek, yang merupakan kombinasi antara struktur data dab perilaku dalam suatu entitas. Model berorientasi objek bermanfaat untuk memahami masalah, komunikasi dengan ahli aplikasi, pemodelan suatu organisasi, meyiapkan dokumentasi serta perancangan program dan basis data. Pertama-tama suatu model analisis dibuat untuk menggambarkan aspek dasar dari domain aplikasi, dimana model tersebut berisi objek yang terdapat dalam domain aplikasi termasuk deskripsi dari keterangan objek dan perilakunya.

2.4.1. OOP (Object Oriented Programming)

Pemrograman berorientasi objek (*object-oriented programming* disingkat OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus dalam kelas-kelas atau objek-objek. Bandingkan dengan logika pemrograman terstruktur. Setiap objek dapat menerima pesan, memproses data, dan mengirim pesan ke objek lainnya, Model data berorientasi objek dikatakan dapat memberi fleksibilitas yang lebih, kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar.

Konsep dasar dari Pemrograman Berorientasi Objek menekankan konsep berikut:

1. *Class* - 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 *object*. 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.
2. Objek - 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. Abstrak - 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 - Memastikan pengguna sebuah objek tidak dapat mengganti keadaan dalam dari sebuah objek dengan cara yang tidak layak hanya metode dalam objek tersebut yang diberi izin 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 - 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.
6. Inheritas (*Inheritance*) - 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).

2.4.2. UML (Unified Modeling Language)

UML (*Unified Modeling Language*) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun system (Flowler, 2006). *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 *tools* 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

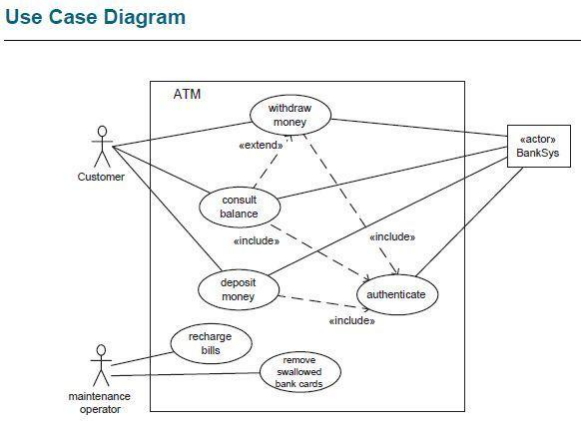
informasi. Penggunaan UML dalam industri terus meningkat. Ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umum dalam industri peranti lunak dan pengembangan *system*.

Diagram-diagram dalam UML (*Unified Modeling Language*)

Beberapa litelatur menyebutkan bahwa *UML* menyediakn Sembilan jenis diagram, yang lain menyebutkan delapan karena ada beberapa diagram yang digabung, misalnya diagram komunikasi, diagram urutan dan diagram pewaktuan digabung menjadi diagram interaksi. Namun demikian model-model itu dapat dikelompokan berdasarkan sifatnya yaitu statis atau dinamis. Jenis diagram itu antara lain:

1. ***Use Case Diagram***

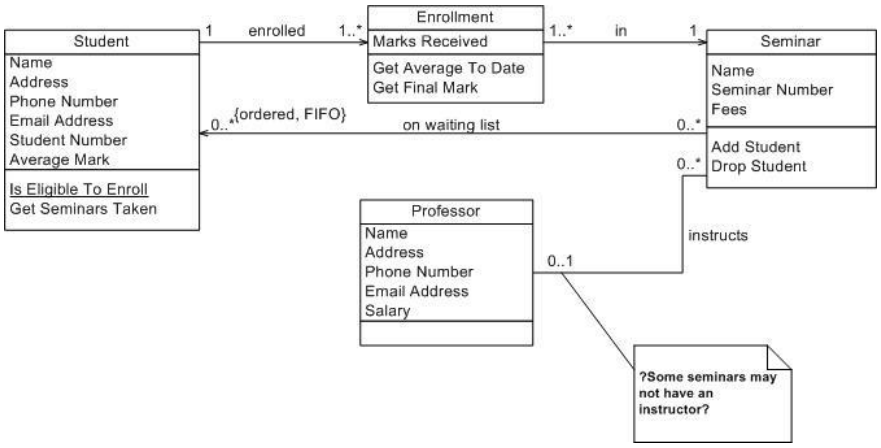
Bersifat statis. Diagram ini memperlihatkan himpunan *use-case* dan aktor-aktor. Diagram ini terutama sangat penting untuk mengorganisasi dan memodelkan perilaku suatu sistem yang dibutuhkan serta diharapkan pengguna. Dapat dilihat pada gambar 2.1 [9].



Gambar 2. 1 *Usecase diagram*

2. ***Class Diagram***

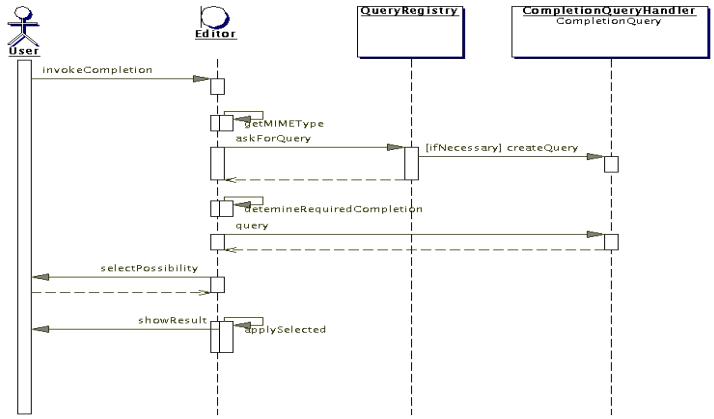
Bersifat statis. Diagram ini memperlihatkan himpunan kelas-kelas, antarmuka-antarmuka, kolaborasi-kolaborasi, serta relasi-relasi. Diagram ini umum dijumpai pada pemodelan sistem berorientasi objek. Meskipun bersifat statis, sering pula diagram kelas memuat kelas-kelas aktif. Dapat digambarkan pada Gambar 2.2 [9].



Gambar 2. 2 ***Class diagram***

3. ***Sequance Diagram***

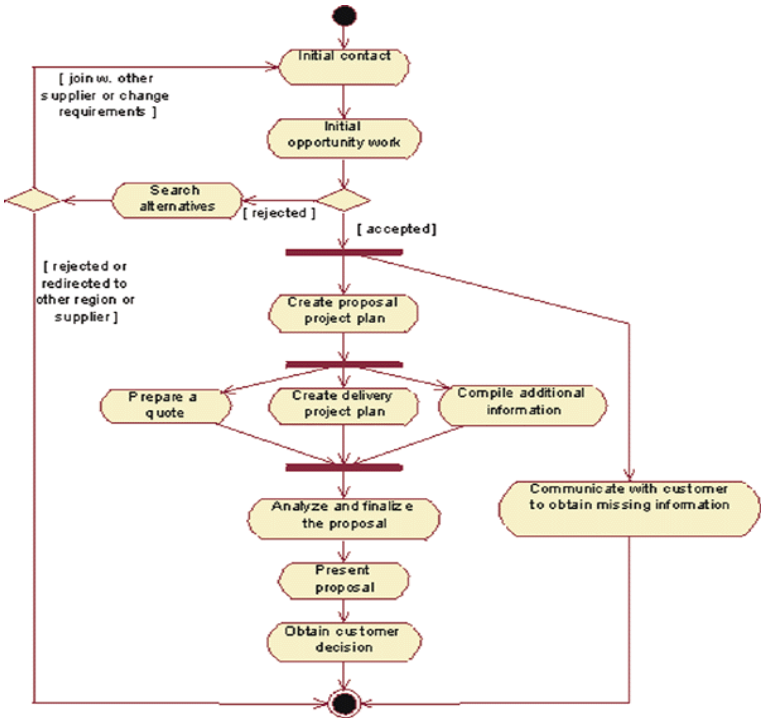
Bersifat dinamis. Diagram interaksi yang menekankan pada pengiriman pesan dalam suatu waktu tertentu. Dapat digambarkan pada Gambar 2.3 [9].



Gambar 2. 3 ***Sequence diagram***

4. *Activity Diagram*

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktivitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktivitas laiinya seperti *Use Case* atau interaksi. Dapat digambarkan pada Gambar 2.4 [9].



Gambar 2. 4 ***Activity diagram***