BAB 2
LANDASAN TEORI

2.1 Kecerdasan Buatan AI *(Artificial Intelligence)*

Kecerdasan buatan berasal dari bahasa inggris "*Artificial Intelligence"* atau disingkat AI, yaitu *intelligence* adalah kata sifat yang berarti cerdas, sedangkan *artificial* artinya buatan Kecerdasan buatan yang dimaksud di sini merujuk pada mesin yang mampu berfikir, menimbang tindakan yang akan diambil, dan mampu mengambil keputusan seperti yang dilakukan oleh manusia.

Menurut Winston dan Prendergast (1984), tujuan dari kecerdasan buatan adalah :

1. Membuat mesin menjadi lebih pintar (tujuan utama)
2. Memahami apa itu kecerdasan (tujuan ilmiah)
3. Membuat mesin lebih bermanfaat (tujuan *entrepreneurial*)

Berdasarkan definisi ini, maka kecerdasan buatan menawarkan media maupun uji teori tentang kecerdasan. Teori-teori ini nantinya dapat dinyatakan dalam bahasa pemograman dan eksekusinya dapat dibuktikan pada komputer nyata.

Pemograman konvesional hanya dapat menyelesaikan persoalan yang diprogram secara spesifik. Jika ada informasi baru, sebuah program konvesional harus diubah untuk menyesuaikan diri dengan informasi baru tersebut. Hal ini tidak hanya menyebabkan boros waktu, namun juga dapat menyebabkan terjadinya eror. Sebaliknya, kecerdasan buatan memungkinkan komputer untuk berfikir atau menalar dan menirukan proses belajar manusia sehingga informasi baru dapat diserap sebagai pengetahuan, pengalaman, dan proses pembelajaran serta dapat digunakan sebagai acuan di masa-masa yang akan datang [9].

2.2 Logika *Fuzzy (Fuzzy Logic)*

Dalam bagian ini akan dibahas pengertian logika *fuzzy*, dasar – dasar logika *fuzzy*.

2.2.1 Pengertian Logika *Fuzzy*

Konsep tentang logika *fuzzy* diperkenalkan oleh Prof. Loyfi Zadeh pada 1962. Logika *fuzzy* adalah metodologi sistem kontrol pemecahan masalah, yang cocok untuk diimplementasikan pada sistem, mulai dari sistem yang sederhana, sistem kecil, *embedded system*, jaringan PC, *multi-chanel* atau *workstation* berbasis akuisisi data, dan sistem kontrol. Metodologi ini dapat diterapkan pada perangkat keras, perangkat lunak, atau kombinasi keduanya. Dalam logika klasik dinyatakan bahwa segala sesuatu bersifat biner, yang artinya adalah hanya mempunyai dua kemungkinan, "Ya atau Tidak", "Benar atau Salah", "Baik atau Buruk", dan lain-lain. Oleh karena itu, semua dapat mempunyai nilai keanggotaan 0 atau 1. Akan tetapi, dalam logika *fuzzy* kemungkinan nilai keanggotaan berada di antara 0 dan 1. Artinya, biasa saja suatu keadaan mempunyai dua nilai "Ya dan Tidak", "Benar dan Salah", "Baik dan Buruk" secara bersamaan. Namun bisa nilainya tergantung pada bobot keanggotaan yang dimilikinya. Logika *fuzzy* dapat digunakan di berbagai bidang, seperti pada sistem diagnosis penyakit (dalam bidang kedokteran), pemodelan sistem pemasaran, riset operasi (dalam bidang ekonomi), kendali kualitas air, prediksi adanya gempa bumi, klasifikasi dan pencocokan pola (dalam bidang teknik) [9].

2.2.2 Dasar – dasar Logika *Fuzzy*

Untuk memahami logika *fuzzy*, sebelumnya perhatikan dahulu tentang konsep himpunan *fuzzy*. Himpunan *fuzzy* memiliki 2 atribut [9], yaitu :

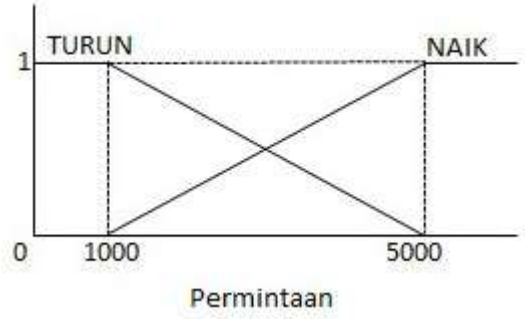
1. Linguistik, yaitu nama suatu kelompok yang mewakili suatu keadaan tertentu dengan menggunakan bahasa alami, misalnya DINGIN, SEJUK, PANAS mewakili variabel temperatur. Contoh lain misalnya MUDA, PAROBAYA, TUA, mewakili variabel umur.
2. Numeris, yaitu suatu nilai yang menunjukan ukuran dari suatu variabel, misalnya 10, 35, 40, dan sebagainya.

Di samping itu, ada beberapa hal yang harus dipahami dalam memahami logika *fuzzy*, yaitu :

1. Variabel *fuzzy*, yaitu variabel yang akan dibahas dalam suatu sistem *fuzzy*. Contoh : penghasilan, temperatur, permintaan, umur, dan sebagainya.

2. Himpunan *fuzzy*, yaitu suatu kelompok yang mewakili suatu keadaan tertentu dalam suatu variabel *fuzzy*.

Variabel permintaan, terbagi menjadi 2 himpunan *fuzzy*, yaitu NAIK dan TURUN. Gambar 2.1 merupakan gambar himpunan naik dan himpunan turun.



Gambar 2.1 Variabel permintaan terbagi menjadi 2 himpunan *fuzzy*, yaitu himpunan NAIK dan himpunan TURUN

3. Semesta pembicaraan, yaitu seluruh nilai yang diizinkan untuk dioperasikan dalam suatu variabel *fuzzy*.

   Contoh : Semesta pembicaraan untuk variabel permintaan : [0+∞)

   Semesta pembicaraan untuk variabel temperatur : [-10 90]

4. Domain himpunan *fuzzy*, yaitu seluruh nilai yang diizinkan dalam semesta pembicaraan dan boleh dioperasikan dalam suatu himpunan *fuzzy*. Pada Gambar 2.1 di atas domain untuk himpunan TURUN dan himpunan NAIK masing-masing adalah :

   Domain himpunan TURUN = [0 5000]

   Domain himpunan NAIK = [1000 +∞)

2.2.3 Metode Mamdani

Metode mamdani paling sering digunakan dalam aplikasi – aplikasi karena strukturnya yang sederhana, yaitu menggunakan operasi MIN – MAX atau MAX – PRODUCT. Untuk mendapatkan output, diperlukan 4 tahapan berikut [14] :

1. *Fuzzyfikasi*

   Artinya himpunan *fuzzy* dan penentuan derajat keanggotaan dari *crisp input* pada sebuah himpunan *fuzzy*.

2. Pembentukan basis pengetahuan *fuzzy* (rule dalam bentuk IF…THEN)

3. Aplikasi fungsi implikasi menggunakan fungsi MIN dan komposisi antar rule menggunakan fungsi MAX (menghasilkan himpunan *fuzzy* baru)
4. Defuzzyfikasi menggunakan metode *centroid.*

   Defuzyfikasi pada komposisi aturan mamdani dengan metode *centroid*. Dimana pada metode ini, solusi *scrip* diperoleh dengan cara mengambil titik pusat daerah *fuzzy*. Secara umum dirumuskan untuk variable kontinu :

$$z* = \frac{\int_z z\mu(z)dz}{\int_z \mu(z)dz}$$

Ada dua keuntungan menggunakan metode *centroid,* yaitu :

1. Nilai defuzzyfikasi akan bergerak secara halus sehingga perubahan dari suatu himpunan *fuzzy* juga akan berjalan dengan halus.
2. Lebih mudah dalam perhitungan.

2.3 *Game*

Dalam bagian ini akan dibahas pengertian *game*, pengertian simulasi *game*, jenis – jenis simulasi, dan klasifikasi *game* berdasarkan *genre*.

2.3.1 Pengertian *Game*

Game adalah kegiatan yang memiliki tujuan untuk dicapai dan dibatasi dengan peraturan bagi para pemainnya yang bersifat menyenangkan dan untuk membuat *game* maka haruslah memahami teknik dan metode animasi [7]. *Game* bertujuan untuk menghibur, biasanya *game* banyak disukai oleh anak – anak hingga orang dewasa. *Game* sebenarnya penting untuk perkembangan otak, untuk meningkatkan konsentrasi dan melatih untuk memecahkan masalah dengan tepat dan cepat karena dalam *game* terdapat berbagai konflik atau masalah yang menuntut untuk dilesaikan dengan cepat dan tepat.

Game merupakan salah satu teknologi yang perkembangannya sangat pesat pada masa kini di mulai dari *game* sederhana seperti permainan kartu, catur sampai permainan yang di mainkan di beberapa sarana atau perangkat teknologi seperti Playstation 3, Xbox 360, Nintendo Wii, PSP, Nintendo DS, maupun PC dari yang berbasis individu ataupun *multiplayer* yang lebih kompleks.

2.3.2 Pengertian Simulasi *Game*

Simulasi merupakan *game* yang mensimulasikan suatu kedaan rill. Contohnya simulasi pengendalian pesawat terbang pada *game Microsoft Flight Simulator,* simulasi kehidupan sehari-hari pada *game* The Sims, dan simulasi seorang manjer tim sepakbola pada *game Champions Manager* [1]**.**

2.3.3 Jenis – jenis Simulasi

Simulasi terdiri dari beberapa jenis [6], diantaranya :

1. *Sosiodrama*.

   Sosiodrama adalah metode pembelajaran bermain peran untuk memecahkan masalah-masalah yang berkaitan dengan fenomena sosial, permasalahan yang menyangkut hubungan antara manusia seperti masalah kenakalan remaja, narkoba, gambaran keluarga otoriter, dan lain sebagainya. Sosiodrama digunakan utnuk memberikan pemahaman dan penghayatan akan masalah-masalah sosial serta mengembangkan kemampuan siswa untuk memecahkannya.

2. *Psikodrama.*

   Psikodrama adalah metode pembelajaran dengan bermain peran yang bertitik tolak daripermasalahan-permasalahan psikologis. Psikodrama biasanya digunakan untuk terapi, yaitu agar siswa memperolehpemahaman yang lebih baik tentang dirinya, menemukan konsep diri, menyatakan reaksi terhadap tekanan-tekanan yang dialaminya.

3. *Role Playing.*

   Role playing atau bermain peran adalah metode pembelajaran sebagai bagian dari simulasi yang diarahkan untuk mengkreasi peristiwa-pristiwa actual, atau kejadian-kejadian yang mungkin muncul pada masa mendatang. Topik yang dapat diangkat untuk role playing misalnya memainkan peran sebagai juru kampanye suatu partai atau gambaran keadaan yang mungkinmuncul pada abad teknologi informasi.

4. *Peer Teaching.*

   Peer teaching merupakan latihan mengajar yang dilakukan oleh siswa kepada teman-teman calon guru. Selain itu peer teaching merupakan

kegiatan pembelajaran yang dilakukan seorang siswa kepada siswa lainnya dan salah satu siswa itu lebih memahami materi pembelajaran.

2.3.4 Klasifikasi *Game* berdasarkan Genre

Menurut buku "*Fundamentals of Game Design*", *genre game* bisa dibagi menjadi 9 [8], yaitu :

1. *Action*

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

2. *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.*

3. *Strategy*

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

4. *Role Playing Game (RPG)*

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

5. *Sports*

   Sports game mensimulasikan berbagai aspek dari olahraga atletik nyata maupun imajiner, apakah itu memainkan pertandingan, me-*manage* tim dan karir, atau keduanya. Salah satu contoh *game* jenis ini adalah *Pro Evolution Soccer* 2012 (PES 2012), dimana pemain bisa memainkan pertandingan,

menjadi manajer tim, maupun menjadi pemain dan mengembangkan karirnya sendiri.

6. *Vechicle Simulation*

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

7. *Construction and Management Simulation*

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

8. *Adventure/ Platform*

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

9. *Artificial Life and Puzzle Game*

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

10. *Online Game*

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

2.4 OOP (*Object* Oriented Programing)

OOP (*Object Oriented Programming*) atau yang dikenal dengan Pemrograman Berorientasi Objek merupakan paradigma pemrograman yang

berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus ke dalam 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 bagi pemula dibanding dengan pendekatan sebelumnya, dan pendekatan OOP lebih mudah dikembangkan dan dirawat.

Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah tidak harus melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek-objek apa yang dapat melakukan pemecahan masalah tersebut. Sebagai contoh suatu departemen yang memiliki *manager*, sekretaris, petugas administrasi data dan lainnya. Misal manager tersebut ingin memperoleh data dari bagian 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 [5].

2.4.1 Istilah-istilah OOP

Istilah OOP sudah sering didengar dalam ilmu pemrograman. Berikut ini istilah-istilah yang sering kali terdengar dalam teknik pemrograman OOP :

1. *Object*

   Objek memiliki atribut sebagai status (*state*) dan tingkah laku sebagai *behavior*. Di dalam OOP, state disimpan pada variabel dan tingkah laku disimpan pada method. Dalam bahasa teoretis OOP, Objek berfungsi untuk membungkus data dan fungsi bersama menjadi satu unit dalam sebuah program komputer. Objek merupakan dasar dari modularitas dan struktur dalam sebuah program komputer berorientasi objek.

2. *Class*

   Class adalah suatu frame yang merupakan definisi yang memuat data dan metod pengolah data. *Class* juga dapat diartikan sebagai tempat untuk membuat obyek. Di dalam *class* dideklarasikan variable dan method yang dimiliki oleh obyek. Proses pembuatan obyek dari sebuah *class* disebut dengan instantiation. Jadi obyek merupakan hasil instansiasi dari *class*. Obyek disebut juga dengan instance. *Class* memiliki anggota yang disebut Anggota *Class* (*class member*) yaitu atribut dan *method*.

3. *Attributes*

   Atribut adalah data yang membedakan antara objek satu dengan yang lainnya. Contoh Objek : Volcano Robot (*a volcanic exploration vehicle*), mempunyai atribut sebagai berikut :

   Status → exploring, moving, returning home

   Speed → in miles per hour

   Temperature → in Fahrenheit degrees

   Dalam *class*, atribut sering disebut sebagai variabel. Atribut dibedakan menjadi dua jenis yaitu *Instance Variable* dan *Class Variable*. *Instance variable* adalah atribut untuk tiap objek dari kelas yang sama. Tiap objek mempunyai dan menyimpan nilai atributnya sendiri. Jadi, tiap objek dari *class* yang sama boleh mempunyai nilai yang sama atau berbeda. *Class Variable* adalah atribut untuk semua objek yang dibuat dari *class* yang sama. Semua objek mempunyai nilai atribut yang sama. Jadi semua objek dari *class* yang sama mempunyai hanya satu nilai yang *value* nya sama.

   a. *Behavior*

      Behavior/tingkah laku adalah hal-hal yang bisa dilakukan oleh objek dari suatu *class*. *Behavior* dapat digunakan untuk mengubah nilai atribut suatu objek, menerima informasi dari objek lain, dan mengirim informasi ke objek lain untuk melakukan suatu tugas (*task*).

   b. Abstraksi

      Abstraksi 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.

2.4.2 Konsep-konsep OOP

Konsep dasar dari Pemrograman Berorientasi Objek Pemrograman orientasi-objek menekankan konsep berikut:

1. Enkapsulasi

   Enkapsulasi berfungsi untuk 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.

2. Pewarisan *(Inheritance)*

   Pewarisan merupakan pewarisan atribut dan method dari sebuah *class* ke *class* lainnya. *Class* yang mewarisi disebut *superclass* dan *Class* yang diwarisi disebut *subclass*. *Subclass* bisa berlaku sebagai *superclass* bagi *class* lainya, disebut sebagai *multilevel inheritance*. Prinsip dasar *inheritance* yaitu persamaan-persamaan yang dimiliki oleh beberapa kelas dapat digabungkan dalam sebuah *class* induk sehingga setiap kelas yang diturunkannya memuat hal-hal yang spesifik untuk kelas yang bersangkutan. Keuntungan pewarisan:

   a. *Subclass* menyediakan *state/behaviour* yang spesifik yang membedakan dengan *superclass*, sehingga memungkinkan *programmer* untuk menggunakan ulang *source code* dari *superclass* yang telah ada.

   b. *Programmer* dapat mendefinisikan *superclass* khusus yang bersifat generik, yang disebut *abstract class* (abstraksi), untuk mendefinisikan *class* dengan tingkah laku dan *state* secara umum.

3. Polimorfisme

   Poolimorfisme adalah kemampuan suatu obyek untuk mempunyai lebih dari satu bentuk. Polimorfisme tidak bergantung kepada pemanggilan subrutin. Metode tertentu yang berhubungan dengan sebuah pengiriman pesan tergantung kepada objek tertentu di mana pesan tersebut dikirim. 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.

2.5 UML *(Unified Modeling Language)*

Pada perkembangan teknik pemrograman berorientasi objek, muncul sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modelling Language* (UML). Adapun pengertian dari UML adalah salah satu standar bahasa yang banyak digunakan di dunia industri untuk mendefinisikan *requirement*, membuat analisis dan desain, serta menggambarkan arsitektur dalam pemrograman berorientasi objek [10].

UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. Dalam hal ini UML merupakan suatu bahasa visual untuk melakukan pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.

UML menggunakan berbagai macam diagram dengan fungsi masing-masing untuk menggambarkan setiap proses dari sistem berorientasi objek. UML memiliki 13 jenis diagram. Dalam pembangunan *game* ini dipakai 4 diagram UML diantaranya :

1. *Use Case Diagram*

Use Case atau diagram *use case* merupakan pemodelan yang digunakan untuk menggambarkan kelakuan (*behavior*) dari sistem yang akan dibuat. *Use case* mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem

yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem dan siapa saja yang berhak menggunakan fungsi-fungsi tersebut. Syarat penamaan pada *use case* adalah nama didefinisikan sesimpel mungkin dan dapat dipahami. Ada dua hal utama pada *use case* yaitu pendefinisian apa yang disebut aktor dan *use case* [10].

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

Sebuah *use case* dapat meninclude fungsionalitas *use case* lain sebagai bagian dari proses dalam dirinya. Secara umum diasumsikan bahwa *use case* yang di-*include* akan dipanggil setiap kali *use case* yang me-*include* dieksekusi secara normal. *Use case* dapat di-*include* oleh lebih dari satu *use case* lain, sehingga duplikasi fungsionalitas dapat dihindari dengan cara menarik keluar fungsionalitas yang *common*. *Use case* juga dapat me-e*xtend use case* lain dengan *behaviour*-nya sendiri. Sementara hubungan generalisasi antar *use case* menunjukkan bahwa *use case* yang satu merupakan spesialisasi dari yang lain. Contoh dari *use case diagram* dapat dilihat pada gambar berikut.

2. *Activity Diagram*

Activity diagram menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, dan bagaimana mereka berakhir. *Activity diagram* juga dapat menggambarkan proses paralel yang mungkin terjadi pada beberapa eksekusi.

Activity diagram merupakan *state diagram* khusus, di mana sebagian besar *state* adalah *action* dan sebagian besar transisi di-*trigger* oleh selesainya *state* sebelumnya (*internal processing*). Oleh karena itu *activity diagram* tidak menggambarkan *behaviour* internal sebuah sistem (dan interaksi antar subsistem) secara eksak, tetapi lebih menggambarkan proses-proses dan jalur-jalur aktivitas dari level atas secara umum.

Sebuah aktivitas dapat direalisasikan oleh satu *use case* atau lebih. Aktivitas menggambarkan proses yang berjalan, sementara *use case* menggambarkan bagaimana aktor menggunakan sistem untuk melakukan aktivitas. Sama seperti *state*, standar UML menggunakan segiempat dengan sudut membulat untuk menggambarkan aktivitas. *Decision* digunakan untuk menggambarkan *behaviour* pada kondisi tertentu. Untuk mengilustrasikan proses-proses paralel (*fork* dan *join*) digunakan titik sinkronisasi yang dapat berupa titik, garis horizontal atau vertikal [10].

3. *Sequence Diagram*

Diagram *sequence* adalah diagram yang menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek [10]. Oleh karena itu untuk menggambarkan diagram *sequence* maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu.

Banyaknya diagram *sequence* yang harus digambarkan adalah sebanyak pendefinisian *use case* yang memiliki proses itu sendiri atau yang penting semua *use case* yang telah didefinisikan interaksi jalannya pesan sudah dicakup pada diagram *sequence* sehingga semakin banyak *use case* yang didefinisikan maka diagram *sequence* yang harus dibuat juga semakin banyak.

4. *Class Diagram*

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

b. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas

c. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas

2.6 *Tools* Yang Digunakan

Tools yang digunakan untuk mengimplementasikan algoritma logika *fuzzy* pada *game* simulasi di antaranya adalah *Java*, Greenfoot dan Fuzzy logic toolbox.

2.6.1 *Java*

Java adalah bahasa pemrograman yang disusun oleh James Gosling yang dibantu oleh rekan-rekannya seperti Patrick Naugton, Chris Warth, Ed Frank, dan Sheridan di suatu perusahaan perangkat lunak yang bernama Susn Microsystems, pada tahun 1991. Bahasa pemrograman ini mula-mula diinisialisasi dengan nama "Oak", namun oada tahun 1995 diganti namanya menjadi "Java".

Alasan utama pembentukan bahasa Java adalah untuk membuat aplikasi-aplikasi yang dapat diletakkan diberbagai macam perangkat elektronik, seperti microwave oven dan remote control, sehingga Java harus bersifat portabel atau yang sering disebut dengan *platform independent* (tidak bergantung pada platform). Itulah yang menyebabkan dalam dunia pemrograman Java, dikenal adanya istilah '*write once, run everywhere'*, yang berarti kode program hanya ditulis sekali, namun dapat dijalankan di bawah platform manapun, tanpa harus melakukan perubahan kode program [11].

2.6.2 Greenfoot

Greenfoot merupakan sebuah IDE berbasis java yang dikhususkan untuk pembuatan *game* sederhana. Greenfoot diprakarsai oleh Michael Kolliung pada tahun 2003. *Prototype* yang pertama dibangun oleh Poul Henriksson dan Michael Kolling pada tahun 2003/2004. semenjak tahun 2005 pembuatan greenfoot ini dilanjutkan dengan melibatkan anggota kelompok dari University of Kent dan Deakin University. Pada awalnya, greenfoot bertujuan untuk menarik minat anak-anak untuk belajar pemrograman, greenfoot ini cocok untuk anak usia 13 tahun ke atas. Cara penggunaannya pun cukup mudah. Selain berbentuk teks seperti editor lainnya, pada greenfoot ini terdapat perangkat alat bantu seperti *class browser*, *editor* dan *compiler*. Greenfoot juga mendukung bahasa Java secara utuh. Dengan adanya bantuan perangkat-perangkat tadi, konsep OOP pada greenfoot lebih mudah dimengerti. Didalam greenfoot terdapat *project* yang dinamakan *scenario*. Di dalam *project* tersebut, kita bisa membuat dua macam tipe *class* dari *library* greenfoot. Yaitu *class world* dan *actor*. World merupakan latar dari *game* yang akan dibuat. Untuk membuat *class world*, caranya adalah dengan mengklik *new sub class* di bagian world. Maka akan mucul sebuah kotak dialog. Kemudian kita

diminta untuk memberi nama untuk *class world* tersebut. Setelah itu kita dapat memilih gambar sebagai latar belakang *class world* yang akan kita buat. Di dalam greenfoot sendiri telah tersedia beberapa gambar untuk latar belakang, seperti gambar tanah, pasir, batu, air, dll. *Class actor* adalah sebuah kelas yang berfungsi sebagai karakter pada *game* yang akan kita buat. Untuk membuat *actor* caranya adalah dengan meng-klik icon aktor, kemudian pilih *new subclass*. caranya hampir mirip dengan pembuatan *class world*. Pada *class actor* juga telah disediakan beberapa gambar, sepertii gambar *animal*, *people*, *symbol*, All [12].

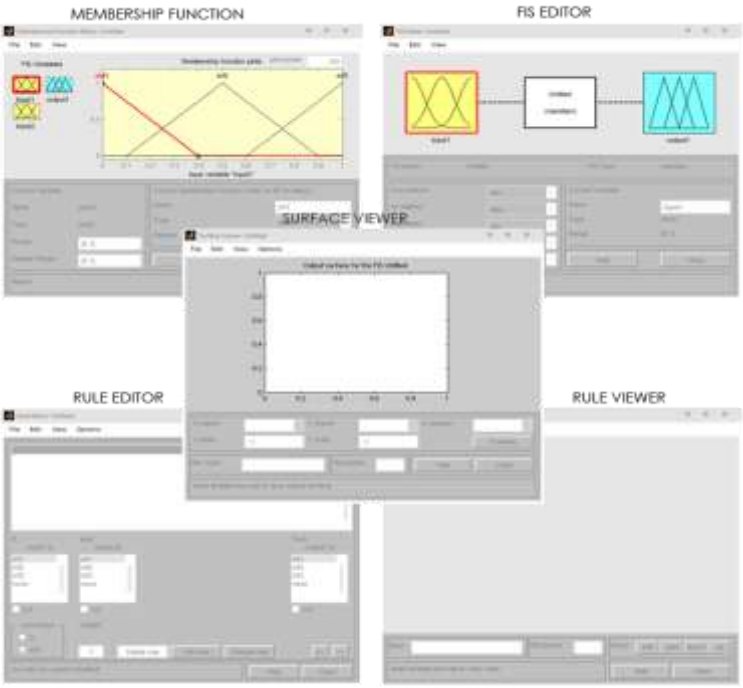
2.6.3 *Fuzzy Logic Toolbox*

Fuzzy Logic Toolbox adalah fasilitas yang tersedia dalam program matlab untuk membangun suatu sistem *Fuzzy* logic. *Fuzzy* Logic Toolbox memberikan fasilitas Graphical User Interface (GUI) untuk mempermudah dalam membangun suatu sistem *Fuzzy*. pada saat membangun sistem *Fuzzy* dengan *Fuzzy* Logic Toolbox, maka pengguna harus melakukan editing yang meliputi : jumlah masukan atau keluaran, klasifikasi masukan atau keluaran, jenis MF (membership Function) masukan atau keluaran, rule, metode De*Fuzzy*fikasi.

Ada 5 GUI yang dapat dipergunakan untuk membangun, mengedit, dan mengobservasi sistem penlaran, yaitu :

1. *Fuzzy* Inference System (FIS) Editor
2. Membership Function Editor
3. Rule Editor
4. Rule Viewer
5. Surface Viewer

Dengan memanfaatkan tigas fasilitas GUI seperti, Fis Editor, MF Editor dan Rule Editor pengguna dapat membaca dan memodifikasi data tentang sistem *Fuzzy* yang dibangun. Untuk lebih jelasnya bisa dilihat pada Gambar 2.2.



Gambar 2.2 *Fuzzy* Logic Toolbox Matlab

2.7 Metode Pengujian Sistem

Metode pengujian sistem untuk mengetahui efektifitas dari *software* yang digunakan selain memberikan kesempatan kepada pengguna untuk mengoperasikan dan melakukan pengecekan terhadap laporan yang dihasilkan melalui *software*. Metode pengujian sistem terdiri dari Pengujian *Whitebox* dan Pengujian *Blackbox* [13].

2.7.1 Pengujian *Whitebox*

Pengujian *whitebox* dilakukan untuk menguji prosedur-prosedur yang ada. Lintasan lojik yang dilalui oleh setiap bagian prosedur diuji dengan memberikan kondisi/loop spesifik. Pengujian *whitebox* menjamin pengujian terhadap semua lintasan yang tidak bergantungan minimal satu kali, mencoba semua keputusan lojik dari sisi '*true*' dan '*false*', eksekusi semua *loop* dalam batasan kondisi dan batasan operasionalnya dan pengujian validasi data internal. Konsep Pengujian Basis *Path* sebagai berikut*:*

1. Merupakan bagian dari pengujian *whitebox* dalam hal pengujian prosedur-prosedur.

2. Mempergunakan notasi aliran *graph* (*node, link* untuk merepresentasikan *sequence, if, while, until* dan sebagainya).
3. Konsep kompleksitas *cyclomatic* antara lain cara perhitungan daerah tertutup pada graph planar dimana dapat menghubungkan batas atas jumlah pengujian yang harus direncanakan dan dieksekusi untuk menjamin pengujian seluruh *statement* program.
4. Memunculkan kasus-kasus yang akan diuji dengan membuat daftar lintasan kasus pengujian berdasarkan kompleksitas dan *cyclomatic* yang didapat.
5. Membuat alat bantu graph matriks yang membantu pengawasan pengujian.

2.7.2 Pengujian *Blackbox*

Pengujian yang dilakukan untuk antarmuka perangkat lunak, pengujian ini dilakukan untuk memperlihatkan bahwa fungsi-fungsi bekerja dengan baik dalam arti masukan yang diterima dengan benar dan keluaran yang dihasilkan benar-benar tepat, pengintegrasian dari eksternal data berjalan dengan baik.

Metode pengujian *blackbox* memfokuskan pada *requirement* fungsi dari perangkat lunak, pengujian ini merupakan komplenetari dari pengujian *whitebox*. Pengujian *whitebox* dilakukan terlebih dahulu pada proses pengujian, sedangkan pengujian *blackbox* dilakukan pada tahap akhir dari pengujian perangkat lunak. Proses yang terdapat dalam proses pengujian *blackbox* antara lain sebagai berikut:

1. Pembagian kelas data untuk pengujian setiap kasus yang muncul pada pengujian *whitebox*.
2. Analisis batasan nilai yang berlaku untuk setiap data.