BAB 2
LANDASAN TEORI

2.1 *Game*

2.1.1 Pengertian *Game*

Dalam kamus bahasa Indonesia "Game" diartikan sebagai permainan. Permainan adalah sesuatu yang dapat dimainkan dengan aturan tertentu sehingga ada yang menang dan ada yang kalah. Permainan dalam hal ini merujuk pada pengertian "kelincahan intelektual" (*intellectual playability*). Game juga bisa diartikan sebagai arena keputusan dan aksi pemainnya. Ada poin yang ingin dicapai pemainnya.

Game merupakan sebuah aktivitas, yang dilakukan dalam konteks meniru realita kehidupan dimana para pemainnya mencoba untuk setidaknya meraih sebuah pencapaian (*achievment*) dalam ruang lingkup aturan tertentu [3].

Game dapat mempunyai banyak arti tetapi mempunyai inti yang sama yaitu memecahkan masalah atau konflik yang di bangun dan mencapai tujuan tertentu .

2.2 *Game* Simulasi

Game simulasi adalah genre *game* yang mementingkan aspek realisme. Segala faktor di dalam *game* ini sangatlah diperhatikan, dengan tujuan agar menciptakan visualisasi yang semirip mungkin dengan dunia nyata [3]. Segala nilai, referensi, material dan faktor lain yang dibutuhkan dalam *game* ini didasarkan pada dunia nyata. Beberapa contoh dari *game* bergenre simulasi antara lain *Simcity 2013, Grand Turismo, Euro Truck Simulator*.

2.3 *Virtual Pet*

Virtual Pet adalah sebuah *game* simulasi yang merupakan kembangan dari *game* Tamagotchi yang pertama kali diproduksi oleh Bandai pada tahun 1996. Bandai merupakan sebuah perusahaan pembuat mainan ternama asal Jepang yang telah berhasil meluncurkan berbagai jenis mainan yang mendunia [7].

Inti dari permainan *Virtual Pet* ini adalah pemain ditugaskan untuk memelihara seekor hewan selama mungkin di dalam *game* dengan melakukan aksi-aksi yang disediakan layaknya memelihara hewan sungguhan seperti memberi makan, bermain, memandikan, dan memberi obat ketika sakit dan ada sebuah reaksi dari *game* berupa keadaan hewan ketika ada aksi yang dilakukan atau yang tidak dilakukan oleh pemain seperti senang, lapar, sedih, lemah, kotor, dan mati.

Permainan ini menuntut ketangkasan, perhitungan, strategi dalam memberikan aksi terhadap hewan dalam *game.* Sebab, permainan dikatakan selesai jika kondisi hewan dalam *game* dalam keadaan mati.

2.4 *Artificial Intelligence* (Kecerdasan Buatan)

Kecerdasan Buatan atau *Artificial Intelligence* (AI) merupakan bidang ilmu komputer yang mempunyai peran penting di era kini dan masa akan datang [2]. Bidang ini telah berkembang sangat pesat di 20 tahun terakhir seiring dengan pertumbuhan kebutuhan akan perangkat cerdas pada industri dan rumah tangga.

AI diartikan secara berbeda-beda tergantung pada sudut pandang para ahli masing-masing. Ada yang hanya focus pada logika berpikir manusia saja dan ada juga yang mendefinisikan AI lebih luas pada tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokkan definisi AI, yang diperoleh dari beberapa *textbook* berbeda, ke dalam empat kategori [5], yaitu :

A. Thinking humanly : *the cognitive modeling approach*

Pendekatan ini dilakukan dengan dua cara sebagai berikut :

1. Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita berpikir. Tetapi, seorang psikolog Barat mengatakan *"how do you know that you understand?"* Bagaimana anda tahu bahwa anda mengerti? Karena pada saat anda menyadari pemikiran anda. Sehingga definisi ini terkesan mengada-ngada dan tidak mungkin dilakukan.
2. Melalui eksperimen-eksperimen psikologi.

B. Acting humanly : *the Turing test approach*

Pada tahun 1950, Alan Turing merancang suatu ujian bagi komputer berintelijensia untuk menguji apakah komputer tersebut mampu mengelabui seorang manusia yang menginterogasinya melalui *teletype* (komunikasi berbasis teks jarak jauh). Jika *interrogator* tidak dapat membedakan yang d2nterogasi adalah manusia atau komputer, maka komputer berintelijensia tersebut lolos dari *Turning test*. Komputer tersebut perlu memiliki kemampuan : *Natural Language Processing, Knowledge Representation, Automated Reasoning, Machine Learning, Computer Vision, Robotics*. *Turing test* sengaja menghindari interaksi fisik antara *interrogator* dan komputer karena simulasi fisik manusia tidak memerlukan intelijensia.

C. Thinking rationally : *the laws of thought approach*

Terdapat dua masalah dalam pendekatan ini, yaitu :

1. Tidak mudah untuk membuat pengetahuan informal dan menyatakan pengetahuan tersebut ke dalam *formal term* yang diperlukan oleh notasi logika, khususnya ketika pengetahuan tersebut memiliki kepastian kurang dari 100%.
2. Terdapat perbedaan besar antara dapat memecahkan masalah “dalam prinsip” dan memecahkannya “dalam dunia nyata”.

D. Acting rationally : *the rational agent approach*

Membuat inferensi yang logis merupakan bagian dari suatu *rational agent.* Hal ini disebabkan satu-satunya cara untuk melakukan aksi secara rasional adalah dengan menalar secara logis. Dengan menalar secara logis, maka bisa didapatkan kesimpulan bahwa aksi yang diberikan akan mencapai tujuan atau tidak. Jika mencapai tujuan, maka *agent* dapat melakukan aksi berdasarkan kesimpulan tersebut.

Thinking humanly dan *acting humanly* adalah dua definisi dalam arti yang sangat luas. Sampai saat ini, pemikiran manusia yang diluar rasio, yakni reflex dan intuitif (berhubungan dengan perasaan), belum dapat ditirukan sepenuhnya

oleh komputer. Dengan demikian, kedua definisi ini dirasa kurang tepat untuk saat ini. Jika kita menggunakan definisi ini, maka banyak produk komputasi cerdas saat ini yang tidak layak disebut sebagai produk AI.

Definisi *thinking rationally* terasa lebih sempit daripada *acting rationally*. Oleh karena itu, definisi AI yang paling tepat untuk saat ini adalah *acting rationally* dengan pendekatan *rational agent*. Hal ini berdasarkan pemikiran bahwa komputer bisa melakukan penalaran secara logis dan juga bisa melakukan aksi secara rasional berdasarkan hasil penalaran tersebut.

2.4.1 *Fuzzy Logic* (Logika Fuzzy)

2.4.1.1 Pengertian

Fuzzy secara bahasa diartikan sebagai kabur atau samar-samar. Suatu nilai dapat bernilai besar atau salah secara bersamaan. Dalam *fuzzy* dikenal derajat keanggotaan yang memiliki rentang nilai 0 (nol) hingga 1(satu). Berbeda dengan himpunan tegas yang memiliki nilai 1 atau 0 (ya atau tidak).

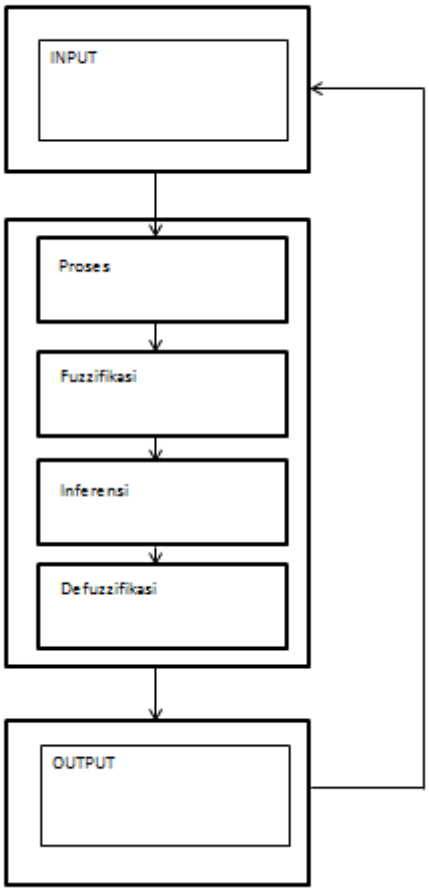
Algoritma *Fuzzy* merupakan suatu logika yang memiliki nilai kekaburan atau kesamaran (*fuzzyness*) antara benar atau salah. Dalam teori Algoritma *Fuzzy* satu nilai bias bernilai benar atau salah secara bersama. Namun berapa besar keberadaan dan kesalahan suatu tergantung pada bobot keanggotaan yang dimilikinya. Algoritma *Fuzzy* memiliki derajat keanggotaan dalam rentang 0 hingga 1. Berbeda dengan logika digital yang hanya memiliki dua nilai 1 atau 0. Algoritma *Fuzzy* digunakan untuk menterjemahkan suatu besaran yang diekspresikan menggunakan bahasa (*linguistic*), misalkan besaran kecepatan laju kendaraan yang diekspresikan dengan pelan, agak cepat, cepat, dan sangat cepat. Dan Algoritma *Fuzzy* menunjukan sejauh mana suatu nilai itu benar dan sejauh mana suatu nilai itu salah. Tidak seperti logika klasik (*crisp*) tegas, suatu nilai hanya mempunyai 2 kemungkinan yaitu merupakan suatu anggota himpunan atau tidak. Derajat keanggotaan 0 (nol) artinya nilai bukan merupakan anggota himpunan dan 1 (satu) berarti nilai tersebut adalah anggota himpunan.

Algoritma *Fuzzy* adalah suatu cara yang tepat untuk memetakan suatu ruang *input* kedalam suatu ruang *output*, mempunyai nilai kontinyu. *Fuzzy*

dinyatakan dalam derajat dari suatu keanggotaan dan derajat dari kebenaran. Oleh sebab itu sesuatu dapat dikatakan sebagian benar dan sebagian salah pada waktu yang sama.

Algoritma *Fuzzy* memungkinkan nilai keanggotaan antara 0 dan 1, tingkat keabuan dan juga hitam dan putih, dan dalam bentuk *linguistik*, konsep tidak pasti seperti "sedikit", "sedang" dan "banyak" [2].

Adapun alur yang terjadi dalam *fuzzy* secara garis besar adalah dapat dilihat pada gambar 2.1 .



Gambar 2.1 Alur Proses *Fuzzy*

Pertama dimasukan inputan yang kemudian akan diproses untuk mencari nilai bobot . Kemudian pada proses *fuzzifikasi* mencari aturan / rule yang berlaku setelah itu mengkomposisikan bobot yang didapatkan dari aturan . Lalu yang terakhir menghitung nilai keluarang dengan menggunakan metode yang berlaku yang kemudian akan menghasilkan output .

Tahap *fuzzifikasi* merupakan tahap terakhir pada proses *fuzzy logic* yang akan digunakan pada sistem yang akan dibangun. Dikarenakan tahap *fuzzifikasi* pada sistem ini akan menghasilkan *output* yang akan dijadikan referensi untuk proses pada algoritma *Naïve Bayes.*

2.4.1.2 Konsep Dasar Logika Fuzzy

Teori himpunan *fuzzy* diperkenalkan pertama kali oleh Lotfi A. Zadeh pada tahun 1965. Ada beberapa alasan mengapa orang menggunakan logika *fuzzy*, antara lain :

1. Konsep logika *fuzzy* mudah dimengerti. Konsep matematis yang mendasari penalaran *fuzzy* sangat sederhana dan mudah dimengerti.
2. Logika *fuzzy* sangat *fleksibel*.
3. Logika *fuzzy* memiliki toleransi terhadap data-data yang tidak tepat.
4. Logika *fuzzy* mampu memodelkan fungsi-fungsi *nonlinear* yang kompleks.
5. Logika *fuzzy* dapat membangun dan mengaplikasikan pengalaman-pengalaman para pakar secara langsung tanpa harus melalui proses pelatihan.
6. Logika *fuzzy* dapat bekerjasama dengan teknik-teknik kendali secara konvensional.
7. Logika *fuzzy* didasarkan pada bahasa alami.

Dalam logika *fuzzy* dikenal berhingga keadaan dari nilai “0” sampai ke nilai “1”. Logika *fuzzy* tidak hanya mengenal dua keadaan tetapi juga mengenal sejumlah keadaan yang berkisar dari keadaan salah sampai keadaan benar. Ada beberapa hal yang perlu diketahui dalam memahami sistem *fuzzy* ini, yaitu :

1. Variabel *Fuzzy*

   Variabel *Fuzzy* merupakan variabel yang hendak dibahas dalam suatu sistem *Fuzzy*.

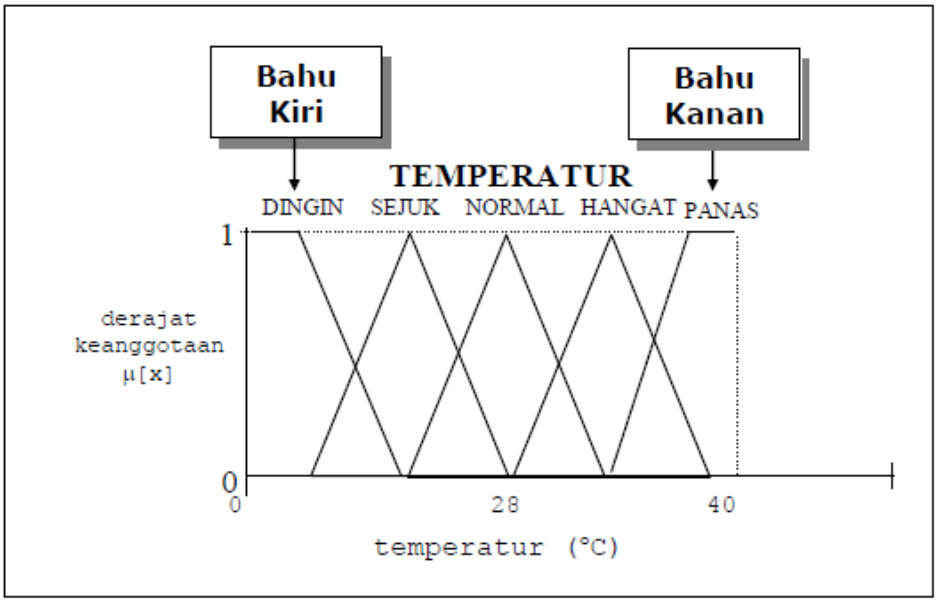
2. Himpunan *Fuzzy*

   Himpunan *Fuzzy* adalah himpunan yang tiap elemennya mempunyai derajat keanggotaan tertentu terhadap himpunannya.

   Himpunan *Fuzzy* memiliki dua atribut, yaitu :

   a. *Linguistik*, yaitu penamaan suatu grup yang mewakili suatu keadaan atau kondisi tertentu dengan menggunakan bahasa alami.
   b. *Numeris*, yaitu suatu nilai (angka) yang menunjukkan ukuran dari suatu variabel.

Contoh Himpunan *Fuzzy* : Misalkan Variabel temperatur, terbagi menjadi 5 himpunan *fuzzy*, yaitu: DINGIN, SEJUK, NORMAL, HANGAT, dan PANAS. Dapat dilihat pada gambar 2.2:



Gambar 2.2 Variabel Temperatur

Pada himpunan tegas (*crisp*), nilai keanggotaan suatu item x dalam satu himpunan A, yang sering ditulis dengan μA[x], memiliki 2 kemungkinan, yaitu:

a. Satu (1), yang berarti bahwa suatu item menjadi anggota dalam suatu himpunan, atau
b. Nol (0), yang berarti bahwa suatu item tidak menjadi anggota dalam suatu himpunan.

   Contoh :

   Jika diketahui:

   S = { 1, 2, 3, 4, 5, 6} adalah semesta pembicaraan.

   A = {1, 2, 3}

   B = {3, 4, 5}

   bisa dikatakan bahwa:

   Nilai keanggotaan 2 pada himpunan A, μA[2]= 1, karena 2 anggota .

Nilai keanggotaan 3 pada himpunan A, μA[3]= 1, karena 3 anggota A.

Nilai keanggotaan 4 pada himpunan A, μA[4]= 0, karena 4 bukan anggota A.

Nilai keanggotaan 2 pada himpunan B, μB[2]= 0, karena 2 anggota B.

Nilai keanggotaan 3 pada himpunan B, μB[3]= 1, karena 3 anggota B.

3. Semesta Pembicaraan

   Semesta pembicaraan adalah suatu keseluruhan nilai yang diperbolehkan untuk dioperasikan dalam suatu variabel *Fuzzy*. Nilai semesta pembicaraan dapat berupa bilangan positif atau bilangan negatif.

4. *Domain*

   Domain himpunan *Fuzzy* adalah keseluruhan nilai yang diijinkan dalam semesta pembicaraan dan boleh dioperasikan dalam suatu himpunan *Fuzzy*. Nilai *domain* dapat berupa bilangan positif maupun bilangan negatif.

2.4.1.3 Karakteristik Logika Fuzzy

Logika *Fuzzy* memiliki beberapa karakteristik yaitu himpunan *Fuzzy* dan fungsi keanggotaan [2].

1. Himpunan *Fuzzy*

Pada logika *boolean*, sebuah individu dipastikan sebagai anggota dari salah satu himpunan saja, sedangkan pada himpunan *fuzzy* sebuah individu dapat masuk pada dua himpunan yang berbeda. Seberapa besar eksistensinya dalam himpunan tersebut dapat dilihat pada nilai keanggotaannya.

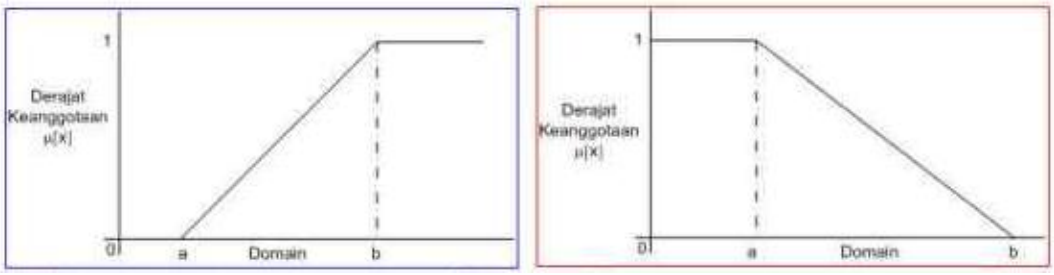
2. Fungsi Keanggotaan (*Membership Function*)

Fungsi keanggotaan adalah suatu kurva yang menunjukkan pemetaan titik-titik *input* data ke dalam nilai keanggotaannya (disebut juga dengan derajat keanggotaan) yang memiliki interval antara 0 sampai 1. Salah satu cara yang dapat digunakan untuk mendapatkan nilai keanggotaan adalah dengan melalui pendekatan fungsi. Derajat keanggotaan dalam himpunan (*degree of membership*) dilambangkan dengan μ .

Dalam sistem *fuzzy* banyak dikenal bermacam-macam fungsi keanggotaan (*membership function*) antara lain adalah fungsi keanggotaan linear , fungsi keanggotaan segitiga dan fungsi keanggotaan trapesium . Dalam *game* akan dibangun, fungsi keanggotaan yang akan digunakan adalah fungsi keanggotaan trapesium dan fungsi keanggotaan segitiga.

Representasi Kurva Trapesium

Kurva trapesium dinamakan demikian karena berbentuk trapesium. Dapat dilihat pada gambar 2.3 :



Gambar 2.3 Fungsi Keanggotan Kurva Trapesium

Fungsi keanggotaan trapesium :

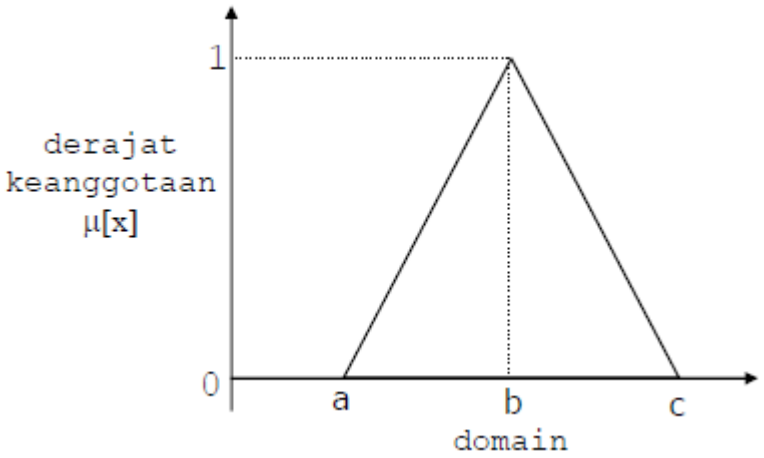
.. (2.1)

$$\mu[x] \begin{cases} 0; & x \leq a \\ \frac{(x-a)}{(b-a)}; & a < x < b \\ 1 & x \geq b \end{cases}$$

$$\mu[x] \begin{cases} 1; & x \leq a \\ \frac{(b-x)}{(b-a)}; & a < x < b \\ 0 & x \geq b \end{cases}$$

Representasi Kurva Segitiga

Kurva segitiga pada dasarnya merupakan gabungan antara dua garis dapat dilihat pada gambar 2.4 :



Gambar 2.4 Fungsi Keanggotan Kurva Segitiga

Fungsi keanggotaan Segitiga:

$$\mu[x] \begin{cases} 0; & x \leq a \; Or \; x \geq c \\ \frac{x-a}{b-a}; & a < x < b \\ \frac{c-x}{c-b}; & b < x < c \\ 1; x = b \end{cases} \quad .. (2.2)$$

2.4.1.4 Fuzzifikasi

Fuzzifikasi adalah pengubahan seluruh variabel *input/output* ke bentuk himpunan *fuzzy*. Rentang nilai variabel *input* dikelompokkan menjadi beberapa himpunan *fuzzy* dan tiap himpunan mempunyai derajat keanggotaan tertentu. *Fuzzifikasi* diperlukan untuk mengubah masukan tegas (*crisp)* yang bersifat bukan *fuzzy* kedalam bentuk himpunan *fuzzy* dari beberapa variabel *linguistik* masukan yang telah didefinisikan.

Bentuk *fuzzifikasi* yang dipakai pada sistem ini adalah bentuk segitiga dan

bentuk bahu. Bentuk *fuzzifikasi* menentukan derajat keanggotaan suatu nilai rentang *input/output*. Derajat keanggotaan himpunan *fuzzy* dihitung dengan menggunakan rumus fungsi keanggotaan dari segitiga *fuzzifikasi*.

Pada sistem ini, proses *fuzzifikasi* dibutuhkan untuk merubah nilai dari setiap variabel *fuzzy* kedalam fungsi keanggotaan untuk menentukan keadaan hewan yang akan dijadikan referensi untuk menentukan kondisi hewan pada proses algoritma *Naïve Bayes.*

2.4.2 *Naive Bayes*

2.4.2.1 Pengertian

Teori Probabilitas Bayesian digunakan untuk menghitung probabilitas terjadinya suatu peristiwa berdasarkan pengaruh yang didapat dari pengujian. Di dalam teori probabilitas dan statistika, teorema Bayes juga dikenal sebagai sebuah teori dengan dua penafsiran berbeda. Dalam penafsiran Bayes, teori ini menyatakan seberapa jauh derajat kepercayaan subjektif harus berubah secara rasional ketika ada petunjuk baru. Teori ini merupakan dasar dari statistika Bayes dan memiliki penerapan dalam sains, rekayasa, ilmu ekonomi, teori *games*, kedokteran dan hukum [2].

2.4.2.2 Teorema *Naive Bayes*

Naïve Bayes Classifier merupakan sebuah metoda klasifikasi yang berakar pada teorema Bayes. Ciri utama dari Naïve Bayes Classifier ini adalah asumsi yang sangat kuat (naif) akan independensi dari masing-masing kondisi kejadian. Sebelum menjelaskan Naïve Bayes Classifier ini, akan dijelaskan terlebih dahulu Teorema Bayes yang menjadi dasar dari metoda tersebut. Pada teorema Bayes, bila terdapat dua kejadian yang terpisah (misalkan A dan B), maka teorema Bayes dirumuskan sebagai berikut:

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)} \quad .. (2.3)$$

dengan ketentuan :

h	: Hypothesis
D	: Data
P(h)	: Probabilitas independen dari h
P(D)	: Probabilitas independen dari D
P(D|h)	: Probabilitas kondisional D jika diberikan h
P(h|D)	: Probabilitas kondisional h jika diberikan D

2.4.2.3 *Naive Bayes Classifier*

Naïve Bayes Classifier adalah metode klasifikasi yang didasarkan pada nilai probabilitas. Klasifikasi adalah proses untuk menemukan model atau fungsi yang menjelaskan atau membedakan konsep atau kelas data, dengan tujuan untuk dapat memperkirakan kelas dari suatu obyek. Oleh karena itu kelas yang ada tentulah lebih dari satu. Penentuan kelas dari suatu dokumen dilakukan dengan cara membandingkan nilai probabilitas suatu sampel berada di kelas yang satu dengan nilai probabilitas suatu sampel berada di kelas yang lain.

Sebagai contoh, ahli statistik ingin mengklasifikasikan sampel kucing ke dalam jenis kelaminnya. Oleh karena itu, terdapat dua kelas yaitu jantan dan betina. Suatu sampel kucing akan diklasifikasikan ke dalam satu kelas saja, entah itu jantan atau betina, dengan melihat petunjuk-petunjuk yang ada (misalnya berat badan, panjang ekor). Penentuan kelas yang cocok bagi suatu sampel dilakukan dengan cara membandingkan nilai probabilitas untuk masing-masing kelas, dan mengambil kelas dengan nilai probabilitas yang tinggi. Secara matematis klasifikasi dirumuskan sebagai berikut:

$$C_{NB} = argmax_{c \in C} P(c) \prod_{i=1}^{n} P(f_i|c) \quad .. (2.4)$$

Pada sistem yang akan dibangun, *Naïve Bayes Classifier* dibutuhkan untuk menentukan kondisi hewan yang akan ditampilkan yang didasarkan pada keadaan hewan yang telah dianalisis menggunakan *Fuzzy Logic.*

2.5 *Object Oriented Programming*

Object Oriented Programming (OOP) merupakan paradigma pemrograman yang berorientasikan kepada objek. Semua data dan fungsi di dalam paradigma ini dibungkus ke dalam kelas-kelas atau objek-objek. Kelebihan penggunaan model data berorientasi objek adalah memberikan fleksibilitas.

Kemudahan mengubah program, dan digunakan luas dalam teknik piranti lunak skala besar. Dengan menggunakan OOP maka dalam melakukan pemecahan suatu masalah tidak melihat bagaimana cara menyelesaikan suatu masalah tersebut (terstruktur) tetapi objek–objek apa yang dapat melakukan pemecahan masalah tersebut. Pemrograman berorientasi objek menekankan konsep berikut:

1. Kelas (*Class*) - kumpulan atas definisi data dan fungsi-fungsi dalam suatu unit untuk suatu tujuan tertentu. Sebagai contoh "*class of cat*" adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam *behavior* atau perilaku dari kucing. 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. Cara seperti ini akan menyederhanakan pemetaan dari masalah ke sebuah program ataupun sebaliknya.

2. Objek (*object*) - 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*) - 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*) - 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 (*Polimorphism*) - 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 pesan tersebut dikirim. Contohnya, bila sebuah burung menerima pesan "gerak cepat", dia akan menggerakan sayapnya dan terbang. Bila seekor serigala 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 tungal 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*) - 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) *behavior* mereka tanpa harus mengimplementasi ulang behavior tersebut (bahasa berbasis-objek tidak selalu memiliki inheritas) [6].

2.6 UML (*Unified Modeling Language*)

UML (*Unified Modeling Language*) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun system. *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 [2]. 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 merupakan dasar bagi perangkat (*tool*) desain berorientasi objek dari IBM.

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.

Berikut adalah berbagai macam diagram UML untuk memodelkan aplikasi berorientasi objek, yaitu [4]:

1. *Use Case Diagram*
2. *Sequence Diagram*
3. *Collaboration Diagram*
4. *State Machine Diagram*

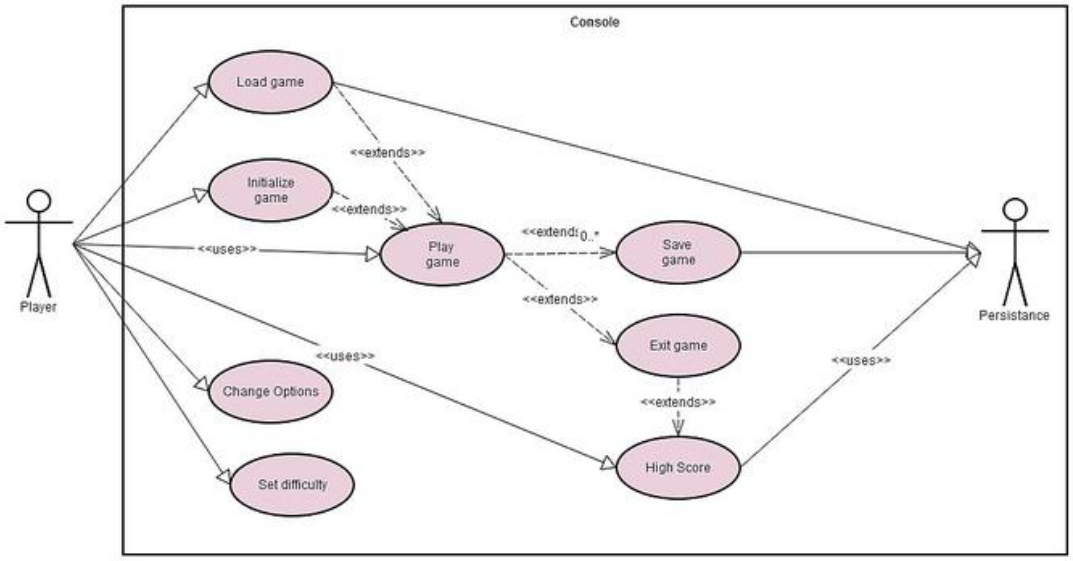
5. *Activity Diagram*
6. *Class Diagram*
7. *Object Diagram*
8. *Component Diagram*
9. *Deployment Diagram*

Pada sistem ini, diagram UML yang akan digunakan adalah:

1. *Use Case Diagram*

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

Contoh *Use Case Diagram* dapat dilihat pada gambar 2.5:

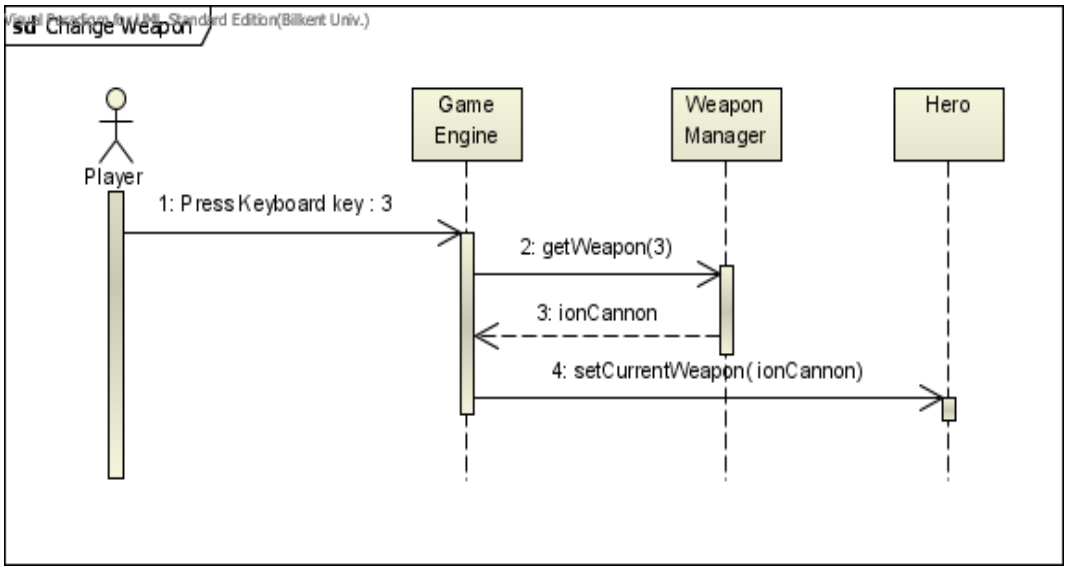


Gambar 2.5 *Use Case Diagram*

2. *Sequence Diagram*

Sequence diagram menggambarkan kelakuan objek pada *use case* dengan mendeskripsikan waktu hidup objek dan pesan yang dikirimkan dan diterima antar objek. Banyaknya *sequence diagram* yang harus digambarkan adalah minimal sebanyak pendefinisian *use case* yang memiiki proses sendiri.

Contoh *Sequence Diagram* dapat dilihat pada gambar 2.6:

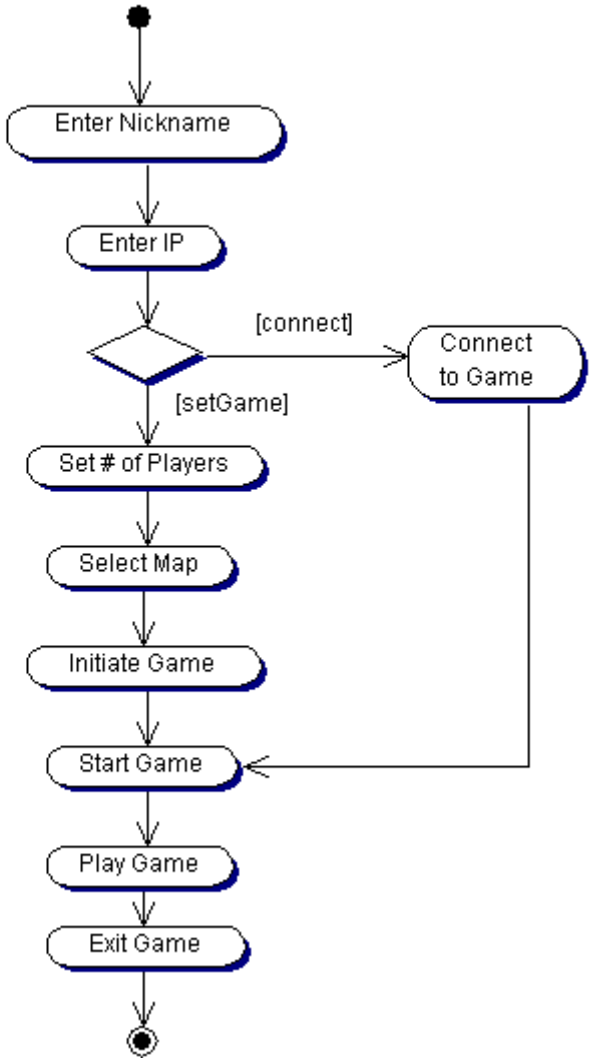


Gambar 2.6 *Sequence Diagram*

3. *Activity Diagram*

Pada dasarnya *Activity diagram* sering digunakan oleh *flowchart*. *Activity diagram* menggambarkan aliran kerja atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak.

Contoh *Activity Diagram* dapat dilihat pada gambar 2.7:

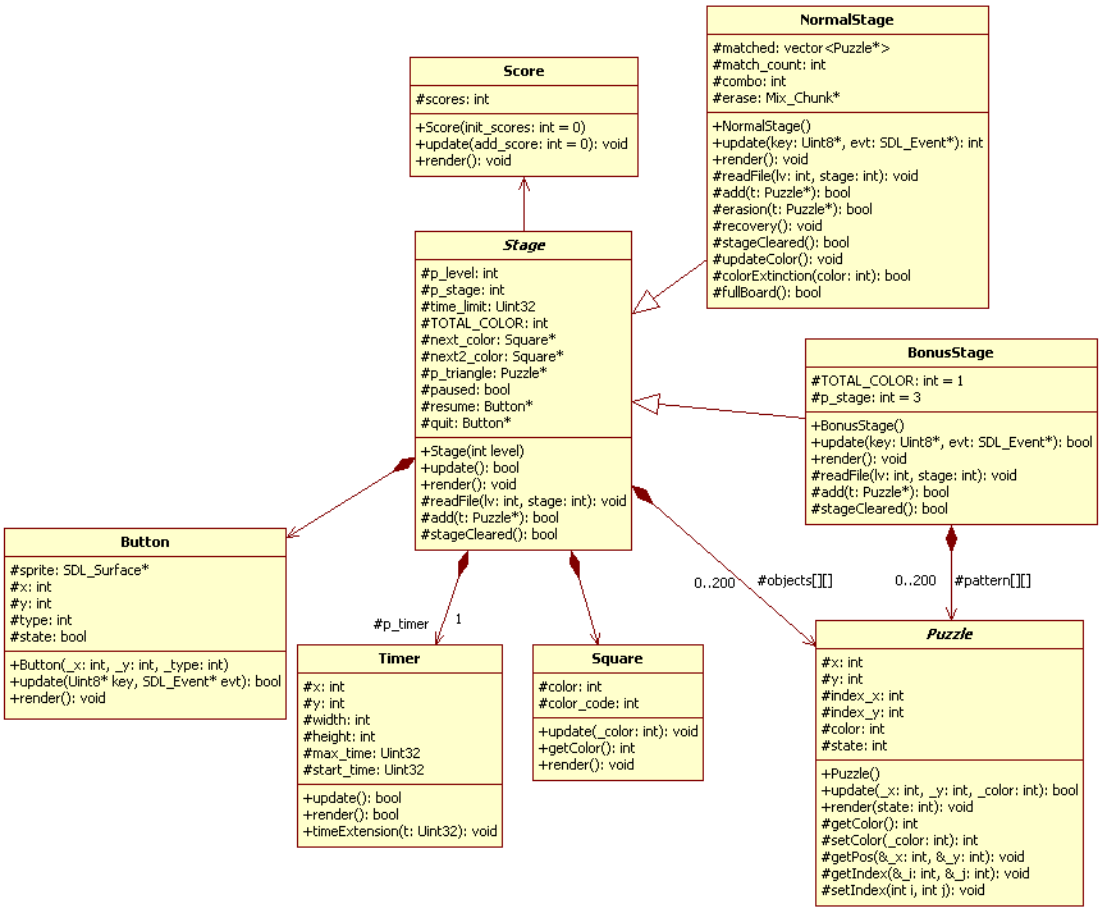


Gambar 2.7 *Activity Diagram*

4. *Class Diagram*

Class diagram adalah sebuah spesifikasi yang jika diinstansiasi akan menghasilkan sebuah objek dan merupakan inti dari pengembangan dan desain berorientasi objek. *Class diagram* menggambarkan keadaan (atribut/properti) suatu sistem, sekaligus menawarkan layanan untuk memanipulasi keadaan tersebut (metoda/fungsi). *Class diagram* menggambarkan struktur dan deskripsi *class*, *package* dan objek beserta hubungan satu sama lain seperti *containment*, pewarisan, asosiasi,dan lain-lain.

Contoh *Class Diagram* dapat dilihat pada gambar 2.8:



Gambar 2.8 *Class Diagram*

2.7 *Tools* yang Digunakan

2.7.1 Adobe Flash Professional CS6

Adobe Flash Professional CS6 merupakan salah satu program animasi 2D vektor yang sangat digemari oleh para animator. Dalam perkembangannya, program adobe flash professional melakukan banyak penyempurnaan pada setiap versinya. Program ini dapat menampilkan beberapa fitur baru yang membuat adobe flash semakin canggih untuk urusan animasi 2D berbasis vektor.

Berbagai aplikasi animasi 2D mulai dari animasi kartun, animasi interaktif, game, company profile, presentasi, video clip, animasi web dan aplikasi animasi lainnya sesuai kebutuhan dapat diolah. Dan kini flash didukung dengan kemampuannya dalam membuat animasi transformasi 3D, serta panel yang khusus untuk membuat animasi dengan ActionScript yang lebih mudah pemakaiannya.

2.7.2 ActionScript 3.0

ActionScript 3.0 merupakan bahasa pemrograman yang bekerja pada Adobe Flash, Flex, dan FlashDevelop. ActionScript 3.0 adalah bahasa pemrograman yang didasarkan pada ECMAScript, yaitu standar bahasa pemrograman yang dikembangkan oleh ECMA (European Computer Manufacturers Association). Dengan standar ini, AS3 mampu melakukan integrasi data yang cepat dengan berbagai bahasa pemrograman lainnya seperti JavaScript dan XML.

ActionScript 3.0 merupakan pengembangan terakhir dari ActionScript saat ini. Kehadirannya membawa perubahan yang signifikan, menjadikan AS3 sebagai bahasa pemrograman yang lebih kompleks, cepat, dan *powerful* dalam membuat *game* [8].

2.8 Pengujian *Black-Box*

Pengujian *Black-Box* merupakan teknik pengujian yang bertujuan menguji perangkat lunak dari segi spesifikasi fungsional tanpa menguji desain dan kode program. Pengujian dimaksudkan untuk mengetahui apakah fungsi-fungsi, masukan, dan keluaran dari perangkat lunak sesuai dengan spesifikasi yang dibutuhkan.

Pengujian *Black-Box* dilakukan dengan membuat kasus uji yang bersifat mencoba semua fungsi dengan memakai perangkat lunak apakah sesuai dengan spesifikasi yang dibutuhkan. Kasus uji yang dibuat untuk melakukan pengujian *Black-Box* harus dibuat dengan kasus benar dan kasus salah [4].

Pada sistem ini, pengujian *Black-Box* dilakukan untuk menguji reaksi dari sistem kepada pemain ketika pemain melakukan aksi terhadap karakter pada *Game.*

2.9 *Confusion Matrix*

Confusion Matrix adalah suatu metode yang digunakan untuk melakukan perhitungan akurasi pada konsep klasifikasi. Mengukur tingkat akurasi pada suatu klasifikasi adalah hal yang penting karena bisa digunakan untuk menilai akurasi prediksi dimasa mendatang yang bisa secara tidak langsung mengatakan tingkat konfedensi yang harus dimiliki dalam keluaran klasifikasi pada sistem yang dibangun [10]. Berikut adalah rumus yang digunakan untuk mengukur akurasi menggunakan *Confusion Matrix:*

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad .. (2.5)$$

Keterangan :

TP = Nilai prediksi negatif yang sesuai

FP = Nilai prediksi positif yang tidak sesuai

FN = Nilai prediksi negatif yang tidak sesuai

TN = Nilai prediksi positif yang sesuai