# BAB 2
# LANDASAN TEORI

## 2.1 Galah

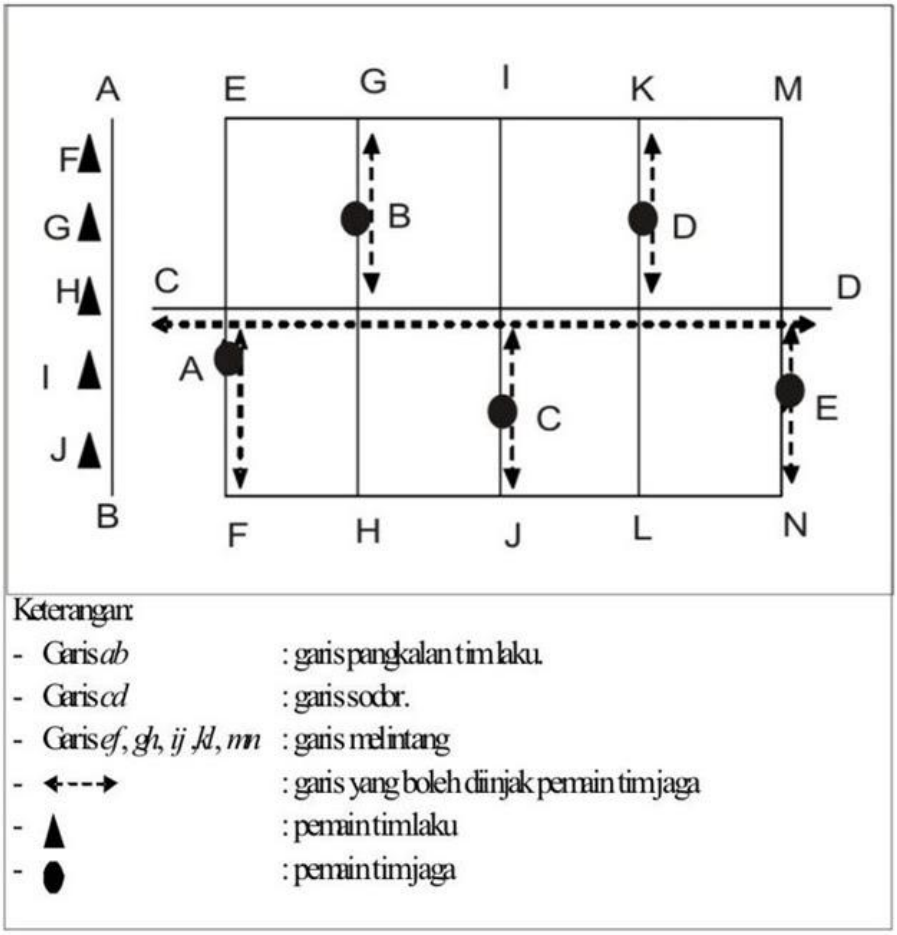
Galah asin, galasin, atau gobak sodor adalah sejenis permainan daerah dari Indonesia. Permainan ini adalah sebuah permainan grup yang terdiri dari dua grup, di mana masing-masing tim terdiri dari 3-5 orang. Inti permainannya adalah menghadang lawan agar tidak bisa lolos melewati garis ke baris terakhir secara bolak-balik, dan untuk meraih kemenangan seluruh anggota grup harus secara lengkap melakukan proses bolak-balik dalam area lapangan.

Permainan ini biasanya dimainkan di lapangan bulu tangkis dengan acuan garis-garis yang ada atau bisa juga dengan menggunakan lapangan segiempat dengan ukuran 9 x 4 m yang dibagi menjadi 6 bagian. Garis batas dari setiap bagian biasanya diberi tanda dengan kapur. Anggota grup yang mendapat giliran untuk menjaga lapangan ini terbagi dua, yaitu anggota grup yang menjaga garis batas horisontal dan garis batas vertikal.

Bagi anggota grup yang mendapatkan tugas untuk menjaga garis batas horisontal, maka mereka akan berusaha untuk menghalangi lawan mereka yang juga berusaha untuk melewati garis batas yang sudah ditentukan sebagai garis batas bebas. Bagi anggota grup yang mendapatkan tugas untuk menjaga garis batas vertikal (umumnya hanya satu orang), maka orang ini mempunyai akses untuk keseluruhan garis batas vertikal yang terletak di tengah lapangan [5].

### 2.1.1 Cara Bermain Galah

Lapangan akan dibagi-bagi menjadi beberapa bagian secara melintang seperti *level* dalam permainan dan secara membujur yang akan membelah daerah permainan menjadi dua. Sebagai jalur yang akan dilalui oleh sang “selodor” akan dijelaskan seperti pada gambar 2.1.




**Gambar 2.1 Ilustrasi Bermain Galah**

Pemain yang berjaga akan mengisi baris-baris melintang di lapangan seperti berlapis-lapis. Kita biasanya mengisi strategi bahwa lapisan terakhir dan paling awal akan diisi oleh orang yang paling panjang tangannya. Ini karena dua lapisan itu adalah yang paling penting. Lapisan terakhir menentukan apakah lawan dapat keluar dari lapangan dan lapisan pertama menentukan apakah lawan dapat masuk ke lapangan. Jumlah *level* atau garis melintang akan sama dengan jumlah pemain disatu tim. Biasanya antara 3-6 orang. Kalau makin banyak. Maka lapangannya akan semakin besar.

Satu orang spesial dari tim yang berjaga disebut “selodor” dia membelah lapangan permainan jadi dua bisa bergerak dari ujung ke ujung. Mengunci lawan

dalam satu kotak adalah salah satu kegunaan utamanya. Dia juga bisa secara tiba-tiba menyerang pemain yang sedang bengong. Posisi selodor ini adalah posisi terpenting ketiga setelah penjaga depan dan belakang.

Pemain yang tidak berjaga akan berusaha untuk masuk ke dalam lapangan, lalu melewati semua *level* dan keluar dari ujung satunya lagi. Dan kemudian berusaha kembali untuk bisa melakukan asin alias gol. Saat salah satu pemain lawan dapat melakukan gol itu. Maka satu babak selesai dan yang tidak berjaga menang mendapatkan nilai dan kemudian permainan diulang lagi tanpa ada perubahan posisi berjaga.

Pemain yang tidak berjaga kalah bila ada salah satu anggotanya tersentuh oleh yang berjaga atau bila salah satu pemainnya keluar dari lapangan sebelah kiri atau kanan. Bila satu kaki saja yang keluar itu tak apa tapi bila dua-duanya berarti dia keluar. Saat pemain yang tidak berjaga kalah maka posisi akan berganti, yang tidak berjaga menjadi jaga dan sebaliknya dan permainan diulang [5].

### 2.1.2 Manfaat Bermain Galah

Permainan ini mempunyai manfaat yang begitu penting untuk perkembangan keterampilan gerak dasar. Diantaranya yaitu:

1. Mengembangkan keterampilan gerak dasar berlari dan reaksi
2. Mengembangkan sikap sosial yang dimiliki anak untuk menyelamatkan temannya dari garis lawan.
3. Dapat melatih kecermatan anak dalam menyelesaikan suatu masalah [5].

## 2.2 Permainan

Permainan merupakan sebuah aktivitas rekreasi dengan tujuan bersenang-senang ataupun untuk mengisi waktu luang. Permainan biasanya dilakukan secara sendiri-sendiri ataupun secara kelompok.

### 2.2.1 Permainan Tradisional

Permainan tradisional adalah bentuk kegiatan permainan dan atau olahraga yang berkembang dari suatu kebiasaan masyarakat tertentu. Pada perkembangan selanjutnya permainan tradisional sering dijadikan sebagai jenis permainan yang memiliki ciri kedaerahan asli serta disesuaikan dengan tradisi budaya setempat.

Kegiatannya dilakukan baik secara rutin maupun sekali-kali dengan maksud untuk mencari hiburan dan mengisi waktu luang setelah terlepas dari aktivitas rutin seperti bekerja mencari nafkah, sekolah [6].

### 2.2.2 Pengertian *Game*

Dalam bagian ini akan dibahas pengertian *game*, klasifikasi *game* berdasarkan *platform*, berdasarkan *genre*, berdasarkan sudut pandang permainan (*game view point*) dan berdasarkan pembatasan usia dari pemainnya serta komponen penyusun dari *game*.

Pengertian *game*

*Game* merupakan salah satu media hiburan yang paling populer untuk semua kalangan usia. *Game* dapat dimainkan dengan aturan tertentu sehigga ada yang menang dan ada yang kalah, biasanya dalam konteks tidak serius atau dengan tujuan *refreshing*. Pengertian *game* menurut beberapa ahli:

1. Menurut David Parlett, *Game* adalah sesuatu yang memiliki "akhir dan cara mencapainya": artinya ada tujuan, hasil dan serangkaian peraturan untuk mencapai keduanya
2. Menurut Roger Caillois seorang sosiolog Perancis, dalam bukunya yang berjudul Les jeux et les hommes menyatakan *game* adalah aktivitas yang mencakup karakteristik berikut:
   a. *Fun* (bebas bermain adalah pilihan bukan kewajiban).
   b. *Separate* (terpisah).
   c. *Uncertain, non-productive, governed by rules* (ada aturan).
   d. *Fictitious* (pura-pura).

3. Menurut Clark C.Abt, *Game* adalah kegiatan yang melibatkan keputusan pemain, berupaya mencapai tujuan dengan dibatasi oleh konteks tertentu (misalnya, dibatasi oleh peraturan)
4. Menurut Agustinus Nilwan dalam bukunya "Pemrograman Animasi dan *Game Profesional*" terbitan Elex Media Komputindo, *game* merupakan permainan komputer yang dibuat dengan teknik dan metode animasi. Jika ingin mendalami pengunaan animasi haruslah memahami pembuatan *game*, atau jika ingin membuat *game* maka haruslah memahami teknik dan metode animasi, sebab keduanya saling berkaitan [7].

Jadi *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. *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.

### 2.2.3 Klasifikasi *Game*

*Game* bisa diklasifikasikan kedalam beberapa hal, diantaranya klasifikasi berdasarkan *platform*, berdasarkan *genre*, berdasarkan sudut pandang permainan (*game view point*) dan berdasarkan pembatasan usia dari pemainnya.

#### 2.2.3.1 Berdasarkan *Platform*

Tiga kategori utama *platform* pada *game* adalah *personal computer* (PC), *console*, dan *mobile* [8].

1. *Personal Computer (PC)*

   *PC game* adalah *game* yang dibuat untuk computer baik berbasis Windows, Mac, ataupun Linux. *PC* menyediakan kekuatan grafis dan pemrosesan yang

kuat yang memungkinkan pengembang untuk membuat *game* yang mutakhir. Tetapi kelemahan *game PC* adalah mahal karena pengguna harus menggunakan *hardware* yang *up-to-date* untuk memainkan *game PC* dengan baik. Selain itu, kelemahan *game PC* bagi pengembang adalah banyaknya konfigurasi *PC* membuat sulit bagi pengembang untuk memastikan *game* tersebut berjalan dengan benar pada semua *setup PC*.

2. *Console*

   *Console* adalah *hardware* yang dibuat oleh pihak ketiga seperti *Sony*, *Microsoft*, dan *Nintendo*. *Console* terhubung ke televisi dan tujuan utamanya adalah untuk bermain *game*. *Game Console* sangat menarik bagi *game* pengembang karena mereka hanya perlu memikirkan satu konfigurasi *hardware* ketika membuat *software* untuk konsol. Sangat kontras dengan *PC* yang memiliki opsi konfigurasi yang tak terbatas.

3. *Mobile*

   *Mobile platform* terdiri atas sesuatu yang *portable* dan bisa digenggam, termasuk ponsel, PDA, iPods, dan *handheld game* seperti *Nintendo* DSi atau *Sony* PSP. *Game mobile* memiliki kontrol yang sederhana jika dibandingkan dengan PC.

#### 2.2.3.2 Berdasarkan *Genre*

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

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. *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.

3. *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.

4. *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.

5. *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.

6. *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.

7. *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.

8. *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.

9. *Online Game*

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

#### 2.2.3.3 Berdasarkan Sudut Pandang Pemain

Sebuah *game* biasanya mempunyai sudut pandang permainan tersendiri disesuaikan berdasarkan *genre game* yang diambil. Berikut beberapa macam sudut pandang permainan yang biasa digunakan [9]:

1. *Side Scrolling*

   Sudut pandang permainan yang terlihat dari samping dan memungkinkan karakter utama untuk bergerak dari kiri ke kanan serta memungkinkan *background* pada *game* seolah-olah bergeser mengikuti pergerakan karakter utama.

2. *Top Down*

   Sudut pandang permainan yang memungkinkan karakter utama bermanuver ke empat arah namun cara permainannya sendiri bergeser dari bawah ke atas, dan biasanya *game* yang menggunakan sudut pandang permainan jenis ini adalah *shooter game*.

3. *Isometric*

   Sudut pandang permainan yang memungkinkan permainan terlihat diantara sisi *Side Scrolling* dan juga *Top Down*, dan biasanya diterapkan pada *game* dengan *genre* RTS (*Real Time Strategy*).

**2.2.3.4 Berdasarkan Pembatasan Usia Pemain**

Menurut ESRB (*Entertainment Software Rating Board*) ada 7 pembatasan usia dari pemainnya adalah sebagai berikut [16]:

1. *Early Childhood (3+),* memiliki isi yang mungkin cocok untuk usia 3 keatas. Tidak mengandung materi yang tidak pantas seperti adegan kekerasan.
2. *Everyone (6+),* memiliki konten yang mungkin cocok untuk usia 6 keatas. Judul-judul dalam kategori ini mungkin berisi sedikit kartun, fantasi atau kekerasan ringan dan tidak jarang menggunakan bahasa yang ringan.
3. *Everyone (10+),* memiliki konten yang mungkin cocok untuk usia 10 keatas. Judul-judul dalam kategori ini mungkin berisi lebih banyak kartun, fantasi atau kekerasan ringan, bahasa ringan dan juga minimal tema sugestif.
4. *Teen (13+),* memiliki konten yang mungkin cocok untuk usia 13 keatas. Judul-judul dalam kategori ini mungkin mengandung kekerasan, tema sugestif, humor kasar, sedikit darah, simulasi perjudian, dan tidak jarang menggunakan bahasa yang kuat.
5. *Mature (17+),* memiliki konten yang mungkin cocok untuk orang usia 17 dan lebih tua. Judul-judul dalam kategori ini mungkin berisi intens kekerasan, darah, konten seksual dan bahasa yang kuat.
6. *Adults Only (18+)* memiliki konten yang hanya boleh dimainkan oleh orang-orang 18 tahun keatas. Judul-judul dalam kategori ini mungkin termasuk adegan yang intens, kekerasan dan juga mengandung konten seksual.
7. RP *(Rating Pending)* telah disampaikan kepada ESRB dan sedang menunggu penilaian akhir. (Simbol ini hanya muncul dalam iklan sebelum permainan dibebaskan).

### 2.2.4 Komponen *Game*

Di dalam sebuah *game*, terdapat komponen-komponen yang menyusunnya [12] antara lain:

1. Grafik

   Grafik merupakan suatu hal yang penting dalam *game*, dimana grafik ini akan membuat *game* menjadi daya tarik user untuk memainkan *game*. Pada awalnya *game* menggunakan grafik monokrom dengan layar hitam putih sampai sekarang sudah menggunakan teknologi 3D, semakin bagus grafik maka *game* tersebut semakin seperti dalam dunia nyata.

2. Suara

   Suara menjadi komponen *game* karena dengan adanya suara pada *game* maka *game* tersebut akan menarik sehingga kita dapat merasakan kondisi dalam *game* tersebut. Selain itu, dengan adanya suara, kita tidak bosan bermain *game*.

3. *Artificial Inteligence* (Kecerdasan Buatan)

   Kecerdasan buatan atau AI (artificial Inteligence) menjadi komponen *game* karena dengan hal tersebut *game* menjadi lebih menantang. Dengan menggunakan AI, *game* dapat berpikir untuk mengalahkan pemain sehingga membuat *game* lebih seru.

4. *Story Line*

*Story line game* menjadi komponen *game* karena dengan adanya *story line* dalam *game*, si pemain seolah-olah bermain sesuai dengan petunjuk yang ada di *game* tersebut, sehingga dapat dikatakan *game* tersebut memiliki cerita sehingga membuat pemain lebih asik memainkan *game* tersebut.

## 2.3 Kecerdasan Buatan (*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. Berikut adalah beberapa definisi kecerdasan buatan yang telah didefinisikan oleh beberapa ahli.

Alan Turing, ahli matematika berkebangsaan Inggris yang dijuluki bapak *computer modern* dan pembongkar sandi Nazi dalam Perang Dunia II 1950, menetapkan definisi *Artificial Intelligence*. "Jika komputer tidak dapat dibedakan dengan manusia saat berbincang melalui terminal komputer, maka bisa dikatakan komputer itu cerdas, mempunyai kecerdasan".

Misalnya Anda mengunjungi sebuah situs agen penjualan barang antik. Di layar komputer muncul wajah seorang wanita yang sangat cantik. Sayangnya wajah wanita itu hanyalah ciptaan komputer belaka. Uniknya, ia mampu bercakap-cakap dengan Anda untuk melayani penjualan barang antik dan Anda tidak menyadarinya, layaknya Anda berbicara dengan staf wanita sesungguhnya di *counter* agen penjualan. Kalau ini terjadi, maka pelayanan penjualan barang antic dapat dilakukan 100% *online*, dengan akurasi yang sangat tinggi, terutama dari konsistensi, keramahan, kecepatan, dan akurasi pelayanan dijamin memuaskan pelanggan. Lain kalau kita menggunakan staf manusia asli yang konsistensinya tidak bias akurat karena terpengaruh kondisi fisik dan emosi saat itu [10].

John McCarthy dari Stanford mendefinisikan kecerdasan sebagai "kemampuan untuk mencapai sukses dalam menyelesaikan suatu permasalahan" [10].

Herbert Alexander Simon (June 15- February 9, 2001) [10]:

"Kecerdasan buatan (*artificial intelligent*) merupakan kawasan penelitian, aplikasi, dan instruksi yang terkait dengan pemrograman komputer untuk melakukan sesuatu hal yang dalam pandangan manusia adalah cerdas".

Rich and Knight (1991) [10]:

"Kecerdasan buatan (AI) merupakan sebuah studi tentang bagaimana membuat komputer melakukan hal-hal yang pada saat itu dapat dilakukan lebih baik oleh manusia".

Encyclopedia Britannica:

"Kecerdasan buatan merupakan cabang dari ilmu komputer yang dalam merepresentasi pengetahuan lebih banyak menggunakan bentuk symbol-simbol daripada bilangan dan memproses informasi berdasarkan metode heuristis atau dengan berdasarkan sejumlah aturan" [10].

Menurut Winston dan Prendergast [10], 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, makan kecerdasan buatan menawarkan media maupun uji teori tentang kecerdasan. Teori-teori ini nantinya dapat dinyatakan dalam bahasa pemrograman dan eksekusinya dapat dibuktikan pada komputer nyata.

Program konvensional hanya dapat menyelesaikan persoalan yang deprogram secara spesifik. Jika ada informasi baru, sebuah program konvensional harus di ubah untuk menyesuaikan diri dengan informasi baru tersebut. Hal ini tidak hanya menyebabkan boros waktu, namun juga dapat menyebabkan terjadinya *error*. Sebaliknya, kecerdasan buatan memungkinkan komputer untuk berpikir 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.

Dari sini dapat dikatakan bahwa:

Kecerdasan adalah pemilik pengetahuan, pengalaman, dan penalaran untuk membuat keputusan dan mengambil tindakan. Jadi, agar mesin bias cerdas (bertindak seperti manusia) maka harus diberi bekal pengetahuan dan diberi kemampuan untuk menalar [10].

Stuart Russel dan Peter Norvig mengelompokan definisi AI, yang diperoleh dari beberapa *textbook* berbeda, ke dalam empat kategori, yaitu:

1. *Thinking Humanly*: *the cognitive modeling approach*
    a. Melalui intropeksi: 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, ternyata pemikiran tersebut sudah lewat dan digantikan kesadaran anda. Sehingga definisi ini terkesan mengada-ada dan tidak mungkin dilakukan.

b. Melalui eksperimen-eksperimen psikologi.

2. *Acting Humanly*: *the Turing test approach*

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

3. *Thinking Rationally*: *the laws of thought approach*

   Terdapat dua masalah dalam pendekatan ini, yaitu:

   a. 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%.

   b. Terdapat perbedaan besar antara dapat memecahkan masalah "dalam prinsip" dan memecahkannya "dalam dunia nyata".

4. *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 bias 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 di luar rasio, yakni *reflex* dan *intuitif* (berhubungan dengan perasaan), belum dapat ditirukan sepenuhnya oleh *computer*. Dengan demikian, kedua definisi ini dirasa kurang tepat untuk saat ini. Jika kita menggunakan definisi ini, maka banyak produk komputasu cerdas saat ini yang tidak layak disebut sebagai produk AI.

Definisi *thinking rationally* terasa lebih sempit daripada *acting rationally* dengan pendekatan *rational agent*. Hal ini berdasarkan pemikiran bahwa *computer* bisa melakukan penalaran secara logis dan juga bisa melakukan aksi secara rasional berdasarkan hasil penalaran tersebut [11].

### 2.3.1 Algoritma *Greedy*

Algoritma *greedy* merupakan algoritma yang membentuk solusi langkah per langkah. 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.

Secara harfiah *greedy* artinya rakus atau tamak, sifat yang berkonotasi negatif. Sifat ini akan mengambil sebanyak mungkin atau mengambil yang paling bagus atau yang paling mahal [12].

#### 2.3.1.1 Prinsip Utama Algoritma *Greedy*

Prinsip utama algoritma *greedy* adalah "*take what you can get now!*". Maksud dari prinsip tersebut ialah bahwa algoritma *greedy* merupakan metode yang paling populer untuk memecahkan persoalan optimasi. Persoalan optimasi (*optimization problems*), persoalan mencari solusi optimum. Algoritma *greedy* membentuk solusi langkah per langkah (*step by step*).

Pada setiap langkah, terdapat banyak pilihan yang perlu dieksplorasi. Oleh karena itu, pada setiap langkah harus dibuat keputusan yang terbaik dalam menentukan pilihan. Pada setiap langkah, kita membuat pilihan optimum lokal (*local optimum*). Dengan harapan bahwa langkah sisanya mengarah ke solusi optimum global (*global optimum*). Algoritma *greedy* adalah algoritma yang memecahkan masalah dengan cara langkah per langkah, yaitu sebagai berikut:

Mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip "*take what you can get now!*").

Berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Optimum global belum tentu menjadi solusi optimum (terbaik). Alasannya, pertama algoritma *greedy* tidak beroperasi menyeluruh terhadap semua alternatif solusi yang ada sebagaimana pada metode *exhaustivesearch*. Kedua, terdapat beberapa fungsi seleksi yang berbeda sehingga kita harus memilih fungsi yang tepat jika kita ingin algoritma tersebut mendapat solusi optimal. Jadi pada sebagian masalah, algoritma *greedy* tidak selalu memberikan hasil yang optimal [12].

#### 2.3.1.2 Elemen Algoritma *Greedy*

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

1. Himpunan Kandidat

   Himpunan yang berisi elemen pembentuk solusi.

2. Himpunan Solusi

   Himpunan yang terpilih sebagai solusi persoalan.

3. Fungsi Seleksi

   Fungsi yang memilih kandidat yang paling mungkin untuk mencapai solusi optimal.

4. Fungsi Kelayakan

   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

   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

   Fungsi yang mengoptimalkan solusi [12].

#### 2.3.1.3 Skema Umum Algoritma Greedy

Misal kita mengasumsikan elemen algoritma *greedy* sebagai berikut:

a. Himpunan Kandidat = C,
b. Himpunan Solusi = S,
c. Fungsi Seleksi = *select*(),
d. Fungsi Kelayakan = *feasible*(),
e. Fungsi Solusi = *solution*(), dan
f. Fungsi Obyektif = *objective*().

Skema umum dari algoritma greedy dapat kita tuliskan:

1. Inisialisasi S dengan kosong.
2. Pilih sebuah kandidat dari C (dengan *select*()).
3. Kurangi C dengan kandidat yang telah terpilih di atas.
4. Periksa apakah kandidat yang dipilih tersebut bersama-sama dengan S membentuk solusi yang layak (dengan *feasible*()). Jika ya, masukkan kandidat ke S; jika tidak buang kandidat tersebut dan tidak perlu ditelaah lagi.
5. Periksa apakah S yang sudah terbentuk telah memberikan solusi yang lengkap (dengan *solution*()). Jika ya, berhenti; jika tidak, ulangi dari langkah 2 [12].

## 2.4 *OOP* (*Object Oriented Programming*)

*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 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 bag 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.

Pemrograman orientasi-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 dog*' adalah suatu unit yang terdiri atas definisi-definisi data dan fungsi-fungsi yang menunjuk pada berbagai macam perilaku/turunan dari anjing. 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** *(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 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) perilaku mereka tanpa haru mengimplementasi ulang perilaku tersebut (bahasa berbasis-objek tidak selalu memiliki inheritas) [13].

## 2.5 *UML (Unified Modeling Language)*

*UML* (*Unified Modeling Language*) adalah bahasa spesifikasi standar untuk mendokumentasikan, menspesifikasikan, dan membangun sistem. *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 system 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* [14].




**Gambar 2.2 Logo *UML***

*UML* 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 system informasi. Penggunaan *UML* dalam industri terus meningkat, ini merupakan standar terbuka yang menjadikannya sebagai bahasa pemodelan yang umumdalam industri peranti lunak dan pengembangan sistem.

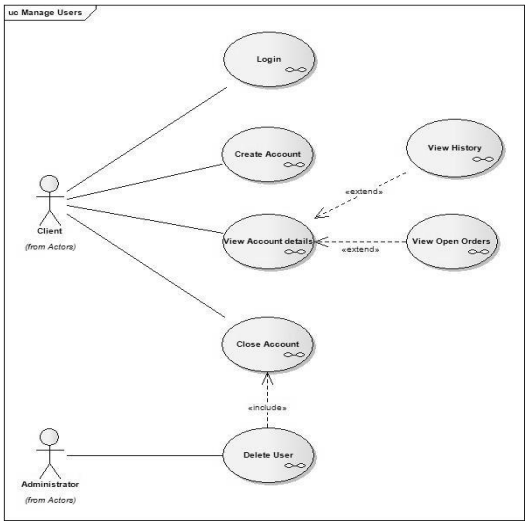
### 2.5.1 Bagian-bagian *UML*

Bagian-bagian utama dari *UML* adalah *view*, *diagram*, *model element*, dan *general mechanism*. *Diagram* berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu *view* tertentu dan ketika digambarkan biasanya.

1. *Use Case Diagram*

*Use case* adalah abstraksi dari interaksi antara sistem dan *actor*. *Use case* bekerja dengan cara mendeskripsikan tipe interaksi antara *user* sebuah sistem dengan sistemnya sendiri melalui sebuah cerita bagaimana sebuah sistem dipakai. *Use case*

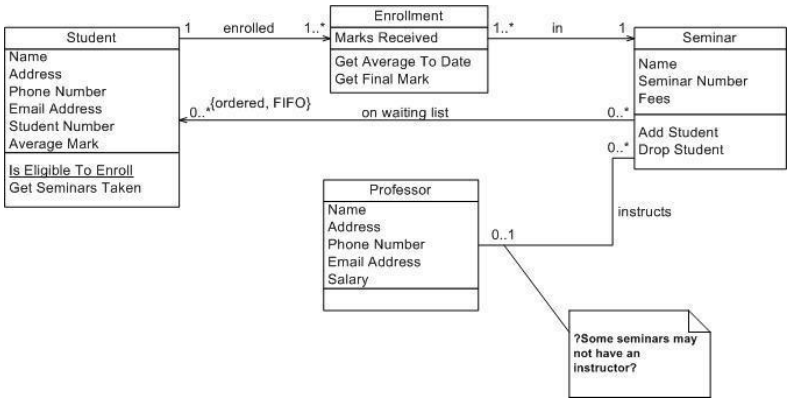
merupakan konstruksi untuk mendeskripsikan bagaimana sistem akan terlihat di mata *user*. Sedangkan *use case diagram* memfasilitasi komunikasi diantara analis dan pengguna serta antara analis dan *client*.




**Gambar 2.3 *Use Case Diagram***

2. *Class Diagram*

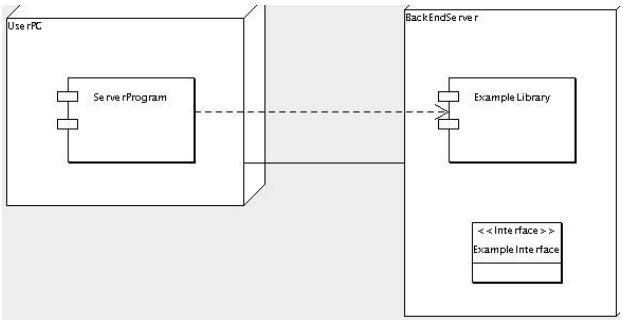
*Class* adalah deskripsi kelompok obyek-obyek dengan properti, perilaku (operasi) dan relasi yang sama. Sehingga dengan adanya *class diagram* dapat memberikan pandangan global atas sebuah sistem. Hal tersebut tercermin dari *class-class* yang ada dan relasinya satu dengan yang lainnya. Sebuah sistem biasanya mempunyai beberapa *class diagram*. *Class diagram* sangat membantu dalam visualisasi struktur kelas dari suatu sistem. *Class diagram* dapat dilihat pada gambar 2.4:




**Gambar 2.4 Logo *Class Diagram***

3. *Component Diagram*

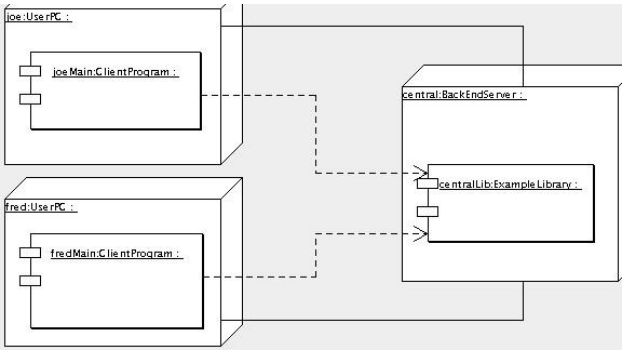
*Component software* merupakan bagian fisik dari sebuah sistem, karena menetap di komputer tidak berada di benak para analis. Komponent merupakan implementasi *software* dari sebuah atau lebih *class*. Komponent dapat berupa *source code*, komponent biner, atau *executable component*. Sebuah komponent berisi informasi tentang *logic class* atau *class* yang diimplementasikan sehingga membuat pemetaan dari *logical view* ke *component view*. Sehingga *component diagram* merepresentasikan dunia riil yaitu *component software* yang mengandung *component*, interface dan relationship. *Component diagram* dapat dilihat pada gambar 2.5.




**Gambar 2.5 *Component Diagram***

4. *Deployment Diagram*

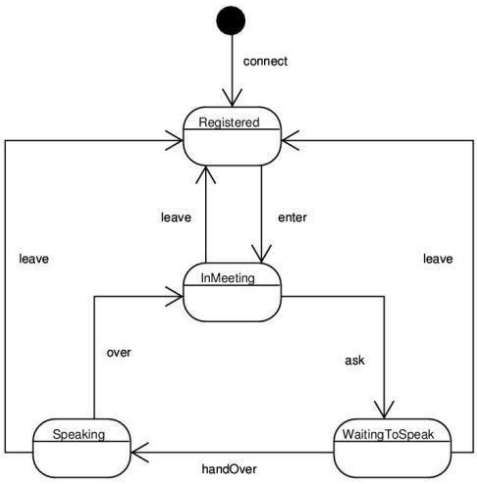
Menggambarkan tata letak sebuah sistem secara fisik, menampakkan bagian-bagian *software* yang berjalan pada bagian-bagian *hardware*, menunjukkan hubungan komputer dengan perangkat (*nodes*) satu sama lain dan jenis hubungannya. Di dalam *nodes*, *executable component* dan *object* yang dialokasikan untuk memperlihatkan unit perangkat lunak yang dieksekusi oleh *node* tertentu dan ketergantungan komponen.




**Gambar 2.6 *Deployment Diagram***

5. *State Diagram*

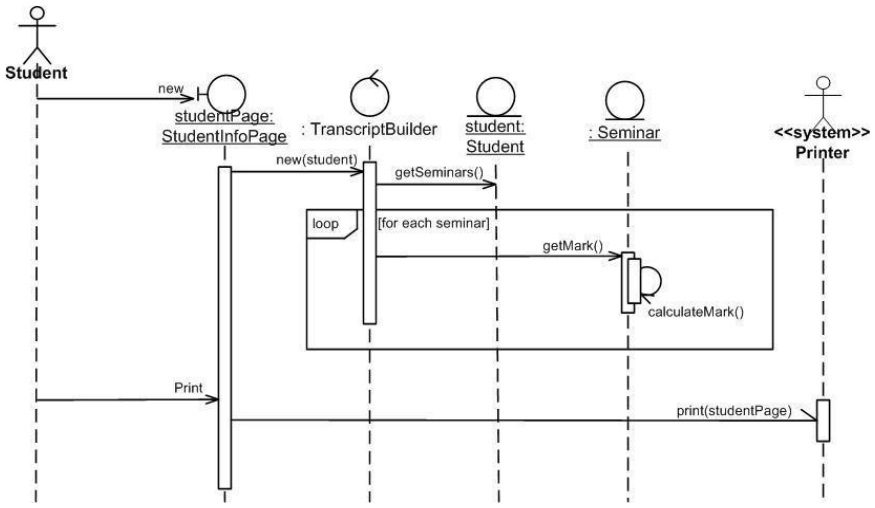
Menggambarkan semua *state* (kondisi) yang dimiliki oleh suatu *object* dari suatu *class* dan keadaan yang menyebabkan *state* berubah. Kejadian dapat berupa *object* lain yang mengirim pesan. *State class* tidak digambarkan untuk semua *class*, hanya yang mempunyai sejumlah *state* yang terdefinisi dengan baik dan kondisi *class* berubah oleh *state* yang berbeda.




**Gambar 2.7 *State Diagram***

6. *Sequence Diagram*

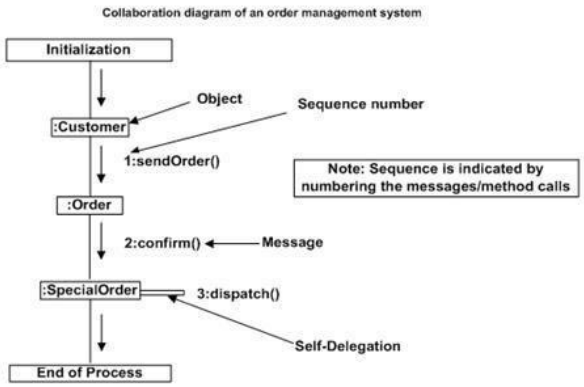
*Sequence Diagram* digunakan untuk menggambarkan perilaku pada sebuah *scenario*. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara *object* juga interaksi antara *object*, sesuatu yang terjadi pada titik tertentu dalam eksekusi sistem.




**Gambar 2.8 *Sequence Diagram***

7. *Collaboration Diagram*

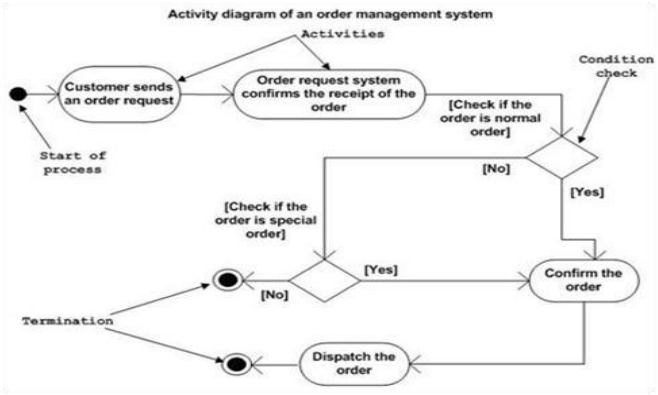
Menggambarkan kolaborasi dinamis seperti *sequence diagram*. Dalam menunjukkan pertukaran pesan, *collaboration diagrams* menggambarkan *object* dan hubungannya (mengacu ke konteks). Jika penekannya pada waktu atau urutan gunakan *sequence diagrams*, tapi jika penekanannya pada konteks gunakan *collaboration diagram*. *Collaboration diagram* dapat dilihat pada gambar 2.9.




**Gambar 2.9 *Collaboration Diagram***

8. *Activity Diagram*

Menggambarkan rangkaian aliran dari aktivitas, digunakan untuk mendeskripsikan aktifitas yang dibentuk dalam suatu operasi sehingga dapat juga digunakan untuk aktivitas lainnya seperti *use case* atau interaksi.




**Gambar 2.10** ***Activity Diagram***

## 2.6 *Tools* Yang Digunakan

Tools yang digunakan dalam Pembangunan Aplikasi *Game* Permainan Tradisional Galah diantaranya adalah HTML5, Construct 2, Adobe Photoshop CS.

### 2.6.1 HTML5

HTML5 adalah bahasa pemrograman yang digunakan untuk membuat struktur dan menampilkan konten dari sebuah website. Angka 5 mengartikan revisi ke-5 dari bahasa pemrograman web standar HTML yang diciptakan pada tahun 1989. HTML5 dikembangkan oleh WHATMG (*Web Hypertext Application Technology Working Group*).

HTML5 menawarkan beberapa fitur menarik yang tidak didukung oleh HTML sebelumnya. Beberapa fitur yang tersedia pada HTML5 antara lain: *Canvas, Header, Footer, Time, Audio, Video* [17].

### 2.6.2 Construct 2

Construct 2 adalah sebuah *tool* berbasis HTML5 untuk menciptakan sebuah permainan. Dengan *tool* Construct 2 memungkinkan siapa saja membuat *game* tanpa harus memiliki pengalaman pemrograman. Dikembangkan oleh Scirra Ltd, Hal ini ditujukan terutama untuk para *non-programmer* yang ingin menciptakan sebuah

game secara *drag and drop* menggunakan editor visual dan berbasis sistem logika perilaku [18].

### 2.6.3 Adobe Photoshop

Adobe Photoshop, atau biasa disebut Photoshop, adalah perangkat lunak editor citra buatan Adobe Systems yang dikhususkan untuk pengeditan foto/gambar dan pembuatan efek. Perangkat lunak ini banyak digunakan oleh fotografer digital dan perusahaan iklan sehingga dianggap sebagai pemimpin pasar (market leader) untuk perangkat lunak pengolah gambar/ foto bersama Adobe Acrobat, dianggap sebagai produk terbaik yang pernah diproduksi oleh Adobe *Systems*. Versi kedelapan aplikasi ini disebut dengan nama Photoshop CS (*Creative Suite*), versi sembilan disebut Adobe Photoshop CS2, versi sepuluh disebut Adobe Photoshop CS3 , versi kesebelas adalah Adobe Photoshop CS4, versi keduabelas adalah Adobe Photoshop CS5 , dan versi yang terakhir (ketigabelas) adalah Adobe Photoshop CS6.

Photoshop tersedia untuk Microsoft Windows, Mac OS X, dan Mac OS. Versi 9 ke atas juga dapat digunakan oleh sistem operasi lain seperti Linux dengan bantuan perangkat lunak tertentu seperti *CrossOver*. Beberapa *tools* yang ada di Photoshop antara lain:

1. *History Brush Tool*.
2. *Eraser Tool.*
3. *Path Selection Tool.*
4. *Direct Selection tool.*
5. *Pen Tool.*
6. *Shape Tool.*
7. *Brush Tool.*
8. *Audio Annotation Tool.*
9. *Eyedropper Tool.*
10. *Measure Tool* [16].

## 2.7 Pengujian Perangkat Lunak

Pengujian perangkat lunak merupakan elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean. Pentingnya pengujian perangkat lunak dan implikasinya yang mengacu pada kualitas perangkat lunak tidak dapat terlalu ditekan karena melibatkan sederetan aktivitas produksi di mana peluang terjadinya kesalahan manusia sangat besar dan arena ketidakmampuan manusia untuk melakukan dan berkomunikasi dengan sempurna maka pengembangan perangkat lunak diiringi dengan aktivitas jaminan kualitas.

Meningkatnya visibilitas (kemampuan) perangkat lunak sebagai suatu elemen sistem dan "biaya" yang muncul akibat kegagalan perangkat lunak, memotivasi dilakukannya perencanaan yang baik melalui pengujian yang teliti. Pada dasarnya, pengujian merupakan satu langkah dalam proses rekayasa perangkat lunak yang dapat dianggap sebagai hal yang merusak daripada membangun.

### 2.7.1 Pengujian *White Box*

Pengujian w*hite box* adalah pengujian yang meramalkan cara kerja perangkat lunak secara rinci, karenanya logikal path (jalur logika) perangkat lunak akan di-test dengan menyediakan test case yang akan mengerjakan kumpulan kondisi atau pengulangan secara spesifik. Secara sekilas dapat diambil kesimpulan *white box* testing merupakan petunjuk untuk mendapatkan program yang benar secara 100%.

Pengujian *white box* merupakan salah satu metode perancangan *test case* yang menggunakan struktur kontrol dari perancangan prosedural untuk mendapatkan *test case*. Dengan rnenggunakan metode *white box*, analis sistem akan dapat memperoleh *test case* yang:

1. Menjamin seluruh *independent path* di dalam modul yang dikerjakan sekurang-kurangnya sekali.
2. Mengerjakan seluruh keputusan logika.
3. Mengerjakan seluruh *loop* yang sesuai dengan batasannya.

4. Mengerjakan seluruh struktur data internal yang menjamin validitas.

#### 2.7.1.1 Uji Coba Basis *Path*

Ujicoba basis path adalah teknik uji coba *white box* yang diusulkan Tom Mc Cabe. Metode ini memungkinkan perancang *test case* mendapatkan ukuran kekompleksan logical dari perancangan prosedural dan menggunkan ukuran ini sebagai petunjuk untuk mendefinisikan basis set dari jalur pengerjaan. *Test case* yang didapat digunakan untuk mengerjakan basis set yang menjamin pengerjaan setiap perintah minimal satu kali selama uji coba.

Terdapat beberapa proses yang harus di lakukan dalam uji coba basis path yaitu diantaranya:

1. Notasi Diagram Alir

Sebelum metode basis *path* diperkenalkan, terlebih dahulu akan dijelaskan mengenai notasi sederhana dalam bentuk diagram alir (grafik alir). Diagram alir menggambarkan aliran kontrol logika yang menggunakan notasi.

2. Kompleksitas Siklomatis

Kompleksitas siklomatis adalah metriks perangkat lunak yang memberikan pengukuran kuantitatif terhadap kompleksitas logis suatu program. Bila metriks ini digunakan dalam konteks metode pengujian basis *path*, maka nilai yang terhitung untuk kompleksitas siklomatis menentukan jumlah jalur independen dalam basis set suatu pemrograman memberi batas atas bagi jumlah pengujian yang harus dilakukan untuk memastikan bahwa semua statemen telah dieksekusi sedikitnya satu kali.

Jalur independen adalah jalur yang memlalui progarm yang mengintroduksi sedikitnya satu rangkaian statemen proses baru atau suatu kondisi baru. Bila dinyatakan dengan terminologi grafik alir, jalur independen harus bergerak sepanjang paling tidak satu *edge* yang tidak dilewatkan sebelum jalur tersebut ditentukan.

*3.* Melakukan *Test Case*

Metode uji coba basis *path* juga dapat diterapkan pada perancangan prosedural rinci atau program sumber. Pada bagian ini akan dijelaskan langkah-langkah uji coba basis path.

4. Matriks Grafis

Prosedur untuk mendaptakan grafik alir dan menentukan serangkaian basis *path*, cocok dengan mekanisasi. Untuk mengembangkan peranti perangkat lunak yang membantu pengujian basis *path*, struktur data yang disebut matriks grafis dapat sangat berguna.

Matriks garfis adalah matriks bujur sangkar yang ukurannya sama dengan jumlah simpul pada grafik alir. Masing-masing baris dan kolom sesuai dengan simpul yang diidentifikasikan dan *entry* matriks sesuai dengan edge diantara simpul.

### 2.7.2 Pengujian *Black Box*

Pengujian *black box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black box* memungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua peryaratan fungsional untuk suatu program. Pengujian *black box* bukan merupakan alternatif dari teknik *white box* tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kesalahan daripada metode *white box*.

Pengujian *black box* berusaha menemukan kesalahan dalam kategori sebagai berikut:

1. Fungsi-fungsi yang tidak benar atau hilang.
2. Kesalahan *interface*.
3. Kesalahan dalam struktur data atau akses *database* eksternal.
4. Kesalahan kinerja.
5. Inisialisasi dan kesalahan terminal.

Tidak seperti pengujian *white box* yang dilakukan pada awal proses pengujian, pengujian black box cenderung diaplikasikan selama tahap akhir pengujian. Karena

pengujian *black Box* memperhatikan struktur kontrol, maka perhatian berfokus pada domain informasi. Pengujian didesain untuk menjawab pertanyaan-pertanyaan berikut:

a. Bagaimana validitas fungsional diuji?
b. Kelas input apa yang akan membuat test case menjadi baik?
c. Apakah sistem sangat sensitif terhadap harga input tertentu?
d. Bagaimana batasan dari suatu data diisolasi?
e. Kecepatan data apa dan *volume* data apa yang dapat ditolerir oleh *system*?
f. Apa pengaruh kombinasi tertentu dari data terdahap operasi sistem?

Dengan mengaplikasikan teknik *black box*, maka dapat menarik serangkaian *test case* yang memenuhi kriteria berikut ini:

1. *Test case* yang mengurangi, dengan harga lebih dari saatu, jumlah test case tambahan yang harus didesain untuk mencapai pengujian yang dapat dipertanggungjawabkan.
2. *Test case* yang memberi tahu sesuatu mengenai kehadiran atau ketidakhadiran kelas kesalahan daripada memberi tahu kesalahan yang berhubungan hanya dengan pengujian spesifik yang ada.

### 2.7.3 Pengujian *Alpha* dan *Beta*

Apabila Perangkat Lunak dibuat untuk pelanggan maka dapat dilakukan *acceptance test* sehingga memungkinkan pelanggan untuk memvalidasi seluruh keperluan. *Test* ini dilakukan karena memungkinkan pelanggan menemukan kesalahan yang lebih rinci dan membiasakan pelanggan memahami perangkat lunak yang telah dibuat.

#### 2.7.3.1 Pengujian *Alpha*

Dilakukan pada sisi pengembang oleh seorang pelanggan. Pengujian ini dilakukan terhadap perangkat lunak untuk memastikan bahwa perangkat lunak dapat

berjalan dengan benar sesuai dengan kebutuhan dan tujuan yang diharapkan. Pengujian *Alpha* berfokus pada persyaratan fungsional dari perangkat lunak.

#### 2.7.3.2 Pengujian *Beta*

Pengujian beta merupakan pengujian yang dilakukan secara objektif, Dimana pengujian dulakukan secara langsung terhadap pengguna, biasanya menggunakan kuisioner mengenai tanggapan pengguna atas perangkat lunak yang telah dibangun. Metode penilaian pengujuan yang digunakan adalah metode kuantitatif berdasarkan data dari pengguna.

### 2.7.4 Skala *Likert*

Skala *Likert* adalah suatu skala psikometrik yang umum digunakan dalam kuesioner. Skala *Likert* merupakan skala yang paling banyak digunakan dalam riset berupa *survey* seperti mengukur sikap, pendapat, dan persepsi seseorang atau sekelompok orang tentang fenomena sosial. Dengan Skala *Likert*, variabel yang akan diukur dijabarkan menjadi indikator variabel. Kemudian indikator tersebut dijadikan sebagai titik tolak untuk menyusun item-item instrumen yang dapat berupa pertanyaan atau pernyataan. Nama skala ini diambil dari nama Rensis Likert, yang menerbitkan suatu laporan yang menjelaskan penggunaannya. Sewaktu menanggapi pertanyaan dalam skala *Likert*, responden menentukan tingkat persetujuan mereka terhadap suatu pernyataan dengan memilih salah satu dari pilihan yang tersedia. Biasanya disediakan lima pilihan skala dengan format seperti:

1. Sangat tidak setuju
2. Tidak setuju
3. Netral
4. Setuju
5. Sangat setuju

Penulis kuesioner harus memutuskan apakah memasukkan titik tengah atau tidak sesuai dengan pernyataan yang diberikan kepada responden. Meskipun

penggunaan respon kategori tengah tidak mempengaruhi reliabilitas dan validitas dalam penelitian ini, namun direkomendasikan bahwa penilaian. Pengembang kuesioner untuk memasukkan alternatif tengah. Ahli lain bahwa menyediakan kategori tengah memungkinkan responden untuk menunjukkan respon yang netral dan lebih diskriminatif dalam respon mereka, membuat nilai skala yang lebih handal dan skala yang lebih disukai oleh responden.

Selain pilihan dengan lima skala seperti contoh di atas, kadang digunakan juga skala dengan tujuh atau sembilan tingkat. Suatu studi empiris menemukan bahwa beberapa karakteristik statistik hasil kuisioner dengan berbagai jumlah pilihan tersebut ternyata sangat mirip.

Prosedur dalam membuat skala *Likert* adalah sebagai berikut:

1. Peneliti mengumpulkan item-item yang cukup banyak, relevan dengan masalah yang sedang diteliti, dan terdiri dari item yang cukup jelas disukai dan tidak disukai.
2. Kemudian item-item itu dicoba kepada sekelompok responden yang cukup representatif dari populasi yang ingin diteliti.
3. Responden di atas diminta untuk mengecek tiap item, apakah ia menyenangi (+) atau tidak menyukainya (-). Respon tersebut dikumpulkan dan jawaban yang memberikan indikasi menyenangi diberi skor tertinggi. Tidak ada masalah untuk memberikan angka 5 untuk yang tertinggi dan skor 1 untuk yang terendah atau sebaliknya. Yang penting adalah konsistensi dari arah sikap yang diperlihatkan. Demikian juga apakah jawaban “setuju” atau “tidak setuju” disebut yang disenangi, tergantung dari isi pertanyaan dan isi dari item-item yang disusun.
4. Total skor dari masing-masing individu adalah penjumlahan dari skor masing-masing item dari individu tersebut.
5. Respon dianalisis untuk mengetahui item-item mana yang sangat nyata batasan antara skor tinggi dan skor rendah dalam skala total. Misalnya, responden pada *upper* 25% dan *lower* 25% dianalisis untuk melihat sampai berapa jauh tiap

item dalam kelompok ini berbeda. Item-item yang tidak menunjukkan beda yang nyata, apakah masuk dalam skor tinggi atau rendah juga dibuang untuk mempertahankan konsistensi internal dari pertanyaan.