BAB 2
LANDASAN TEORI

2.1 *Game*

Permainan *video* adalah permainan yang menggunakan interkasi dengan antarmuka pengguna melalui gambar yang dihasilkan oleh piranti *video*. Sistem elektronik yang digunakan untuk menjalankan permainan *video* yang disebut dengan *platform* seperti *game console* dan *Personal Computer*.

Untuk melakukan perintah memanipulasi permainan *video* digunakan *game controller*, adapun *game controller* dapat berupa *joystick* yang terdiri dari satu tombol saja. Namun seiring perkembangan jaman, saat ini *game controller* tidak hanya *joystick,* melainkan dengan menggunakan sebuah *mouse* maupun *keyboard* dan juga penggunaan *joystick* yang memiliki beberapa tombol.

2.1.1 Sejarah Singkat *Game*

Pada tahun 1949, seorang insinyur muda bernama Ralph Baer mendapat tugas untuk membangun sebuah televisi. Ralph Baer bukan hanya membuat televisi yang biasa-biasa saja, tetapi sebuah televisi yang akan menjadi terbaik dari semua televisi. Ralph ingin menggabungkan beberapa jenis *game* kedalam televisi, tapi masih belum tahu jenis *game* yang akan digabungkan dan akan membutuhkan waktu 18 tahun untuk menjadi kenyataan[2]. Pada saat itu ada orang yang berbagi ilmu dalam *game* seperti William Higibotham yang berperan sebagai orang yang mendesain *game* tenis interaktif pada oscilloscpe, Steve Russel yang berperan sebagai orang yang menciptakan *space game* pada DEC PDP-1 *mainframe computer* dan Nolan Bushnell yang bermain *space game* dan bermimpi bahwa kedepannya dunia dipenuhi oleh *game* yang didukung dengan komputer.

Pada tahun 1958 William Higinbotham, seorang pegawai yang bekerja United States Goverment’s Brookhaven Natioal Laboratory. Dia menciptakan *game Tennis for Two* dengan menggunakan oscilloscope sebagai antarmuka[3]

dijalankan pada komputer analog dan dihubungkan ke osiloskop, permainan ini terlihat jelas dan cepat, yang mengejutkan adalah permainan ini tidak hanya *top-down perspective* seperti *Pong*, namun menampilkan dua sudut seperti lapangan tenis. Dua pemain harus melakukan pukulan terhadap bola secara bergantian dengan bola yang memantul ke tanah dan *net* searah gravitasi. Pada awal tahun 1962, seorang murid di MIT bernama Steve Russel menciptakan permainan *Spacewar* pada komputer DEC PDP-1. Tujuan dari permainan tersebut adalah setiap pemain melakukan *maneuver* pesawatnya ketika mencoba menembak pesawat pemain lain dengan menggunakan misil.

Pada akhir tahun 1971, perusahaan televisi yang bernama *Magnavox* akhirnya menandatangani kontrak dengan Sanders. Pada tahun 1972, *Magnavox* menunjukkan peralatan terbarunya yang diberi nama *Magnavox Odyssey*.

Pada tahun selanjutnya, berbagai perusahan pembuat permainan *video* berlomba-lomba untuk menciptakan perangkat dan permainan bagi para penggemar permainan *video* dan merupakan awal dari kebangkitan permainan *video* hingga saat ini.

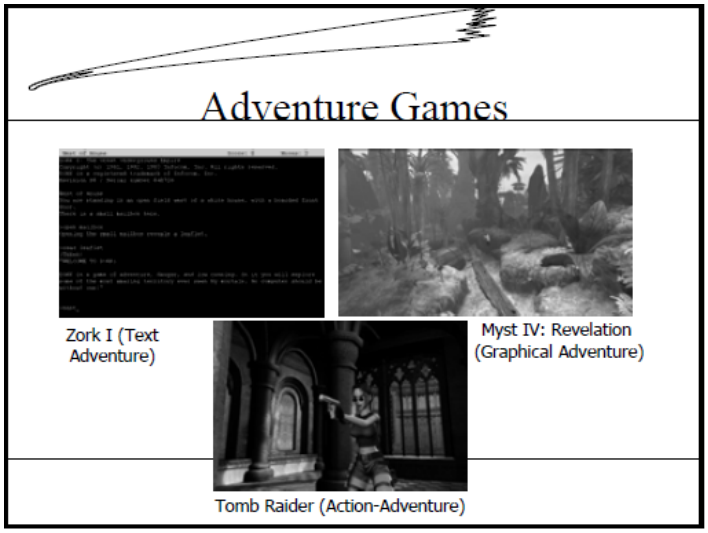
2.1.2 Jenis-jenis Game

Dalam permainan *video* terdapat beberapa jenis permainan yang dapat dimainkan, hal ini dibedakan berdasarkan cara bermain yang dapat dilakukan oleh para pemain[4].

A. ***Adventure Games***

Permainan berdasarkan/berbasiskan pada cerita, menggunakan teka-teki untuk menyelesaikan permainan sepanjang cerita.

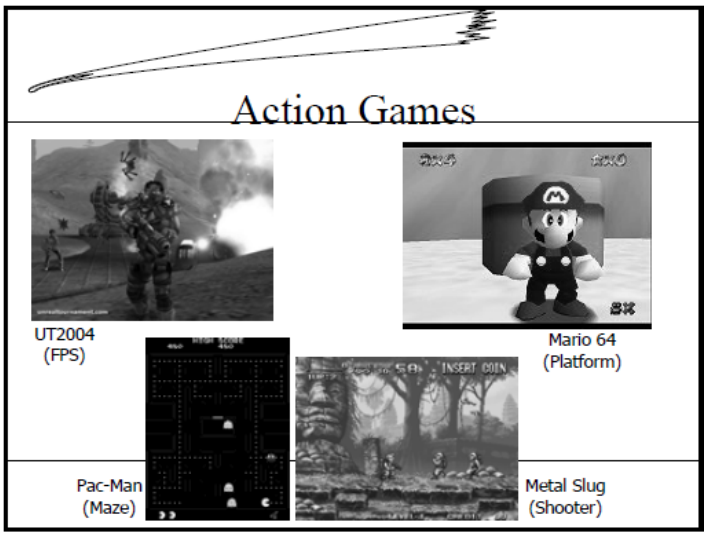
a. Bisa berupa *first, second,* atau *third person.*
b. Dapat berbasis teks (membutuhkan parser dari beberapa jenis) atau grafis (titik dan klik).
c. Umumya memiliki dunia atau ruang lingkup yang luas, sangat kompleks dengan berbagai karakter yang menarik serta plot yang baik.



Gambar 2.1 Contoh *Adventure Games* [4]

B. Action Games

Permainan *real-time* dengan membutuhkan respon atau reaksi yang cepat. *First Person Shooter* (FPS) seperti Quake dan Unreal Tournament, *Platform games* seperti Mario dan Sonic The Hedgehog, *Maze games* seperti Pac-man dan *Shooters* seperti Space Invaders dan Metal Slug. Musuh bisa berupa komputer yang di-*generate* atau manusia. Beberapa juga menambahkan petualangan, strategi, atau taktis elemen untuk pengalaman yang berbeda[4].

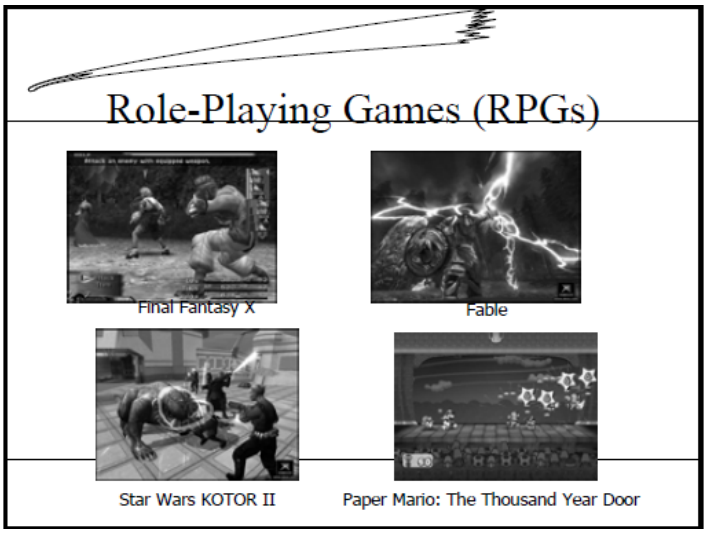


Gambar 2.2 Contoh *Action Games* [4]

C. ***Role-Playing Games (RPGs)***

Dalam RPG, pemain umumnya mengarahkan sekelompok *heroes*/pahlawan dalam serangkaian misi. Dunia atau ruang lingkupnya luas, pemain bisa mengkontrol setiap karakter, karakter diciptakan dengan kekuatan dan kemampuan, berperang adalah elemen yang penting untuk mendapatkan/mengumpulkan pengalaman, uang dan kekuatan. RPG memiliki fitur magis yang kompleks dan beragam ras karakter[4].

.



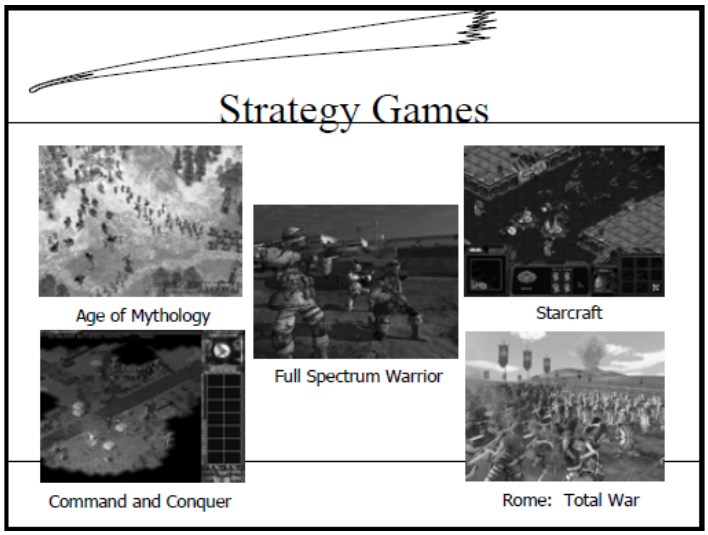
Gambar 2.3 Contoh *Role-Playing Shooter*[4]

D. ***Strategy Games***

Pemain mengelola beberapa sumber daya yang terbatas untuk mencapai tujuan yang telah ditentukan. Memanejemen sumber untuk menciptakan karakter dan bagaimana menempatkan karakter.

Dapat berupa *turn-based* atau *real-time*. Turn-based strategi memberi pemain waktu untuk berpikir dan mengimplementasikan keputusan. *Real-time* strategi (RTS) memiliki semua lawan berpikir dan bertindak pada waktu yang bersamaan. Lawan b

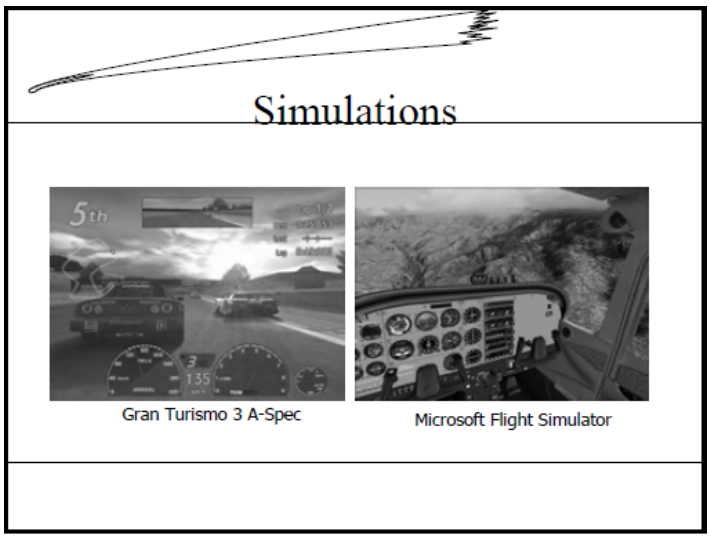
isa berupa komputer, manusia atau keduanya[4]



Gambar 2.4 Contoh *Strategy Game*[4]

E. ***Simulation* (*sims*) *Games***

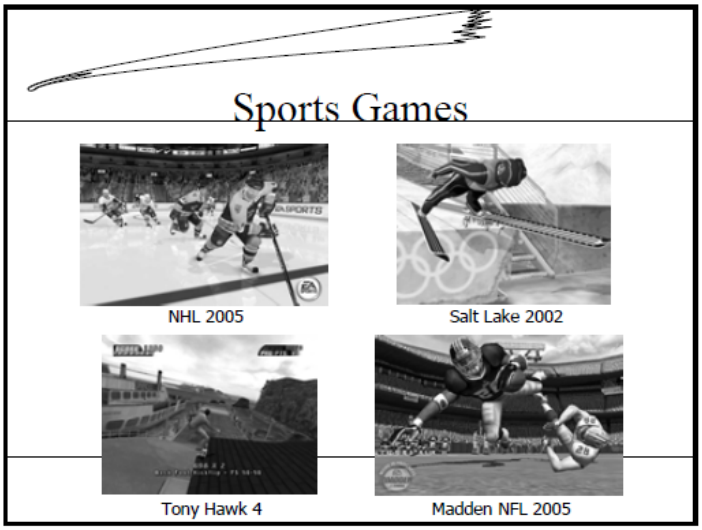
Simulation adalah jenis permainan dengan meniru dunia nyata dengan kondisi operasi dengan sangat baik. Membutuhkan waktu dan pembelajaran untuk mengenal seluk-beluk permainan. Kebanyakan dari game jenis ini berkaitan dengan sesuatu yang berhubungan dengan mesin seperti pesawat, tank, mobil dan sebagainya. Contohnya seperti simulasi pesawat terbang, *game* balap[4].



Gambar 2.5 Contoh *Simulation (sims) Game*[4]

F. ***Sports Games***

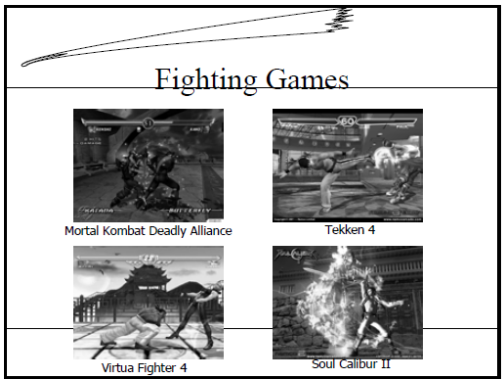
Berpatisipasi dalam acara kegiatan olahraga. Pemain (*gamer)* bisa sabagai pemain, pemilik, manejer atau pelatih. Konsep permainan bisa seperti pertandingan tunggal, seluruh musim atau seumur hidup dari sebuah tim[4].



Gambar 2.6 Contoh *Sport Game*[4]

G. ***Fighting Game***

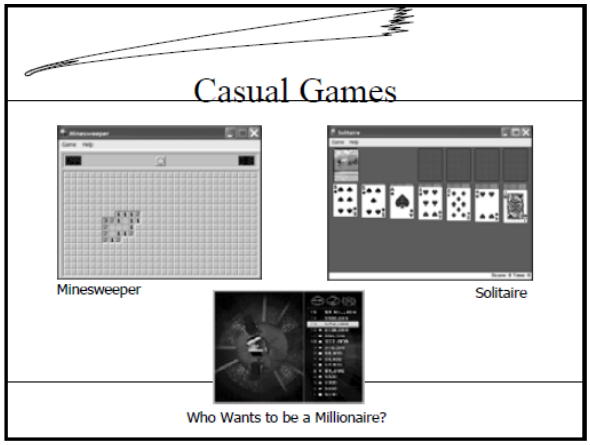
Permainan ini menggunakan tombol dan kombinasi untuk bergerak menyerang dan bertahan dari serangan. Pemain diberikan serangan dasar dan serangan balasan, serta kombinasi yang kompleks. Umumnya pertarungan berakhir hanya dalam beberapa menit saja, tetapi mungkin akan banyak putaran/ronde untuk menyelesaikan permainan ini[4].



Gambar 2.7 Contoh Fighting Game[4]

H. ***Casual Game***

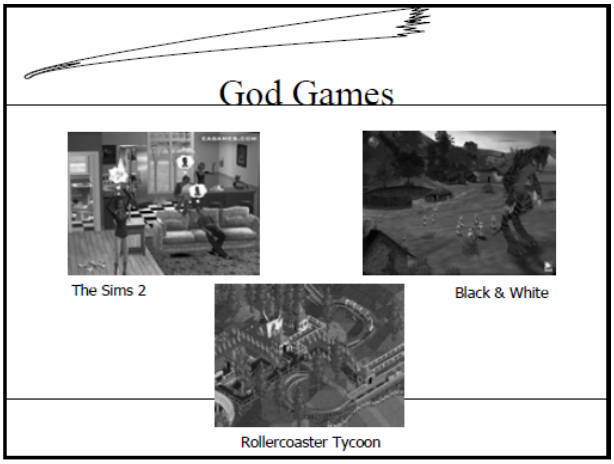
Casual game adalah jenis permainan yang mudah untuk dimainkan, sesi permainan yang singkat. Permainan ini banyak mengadaptasi dari permainan tradisional seperti catur, hearts dan solitaire. Termasuk juga dalam acara televisi seperti "*Who Wants To Be a Millionaire*?", Jeopardy dan Wheel of Fortune. Adapun permainan sederhana seperti *Minesweeper* [4].



Gambar 2.8 Contoh *Casual Game*[4]

I. ***God Game***

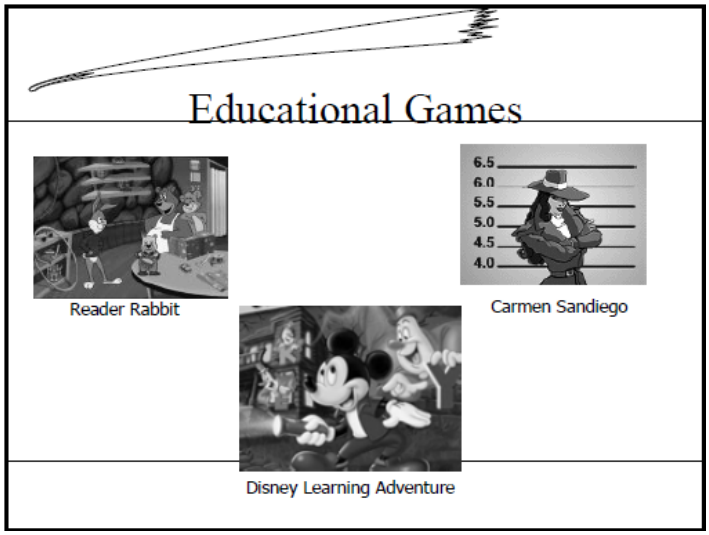
Permainan yang tidak memiliki tujuan, selain untuk mendorong pemain untuk bermain dengan mereka untuk melihat apa yang terjadi. Sering disebut juga dengan *software toys* [4].



Gambar 2.9 Contoh *God Game*[4]

J. ***Educational Game***

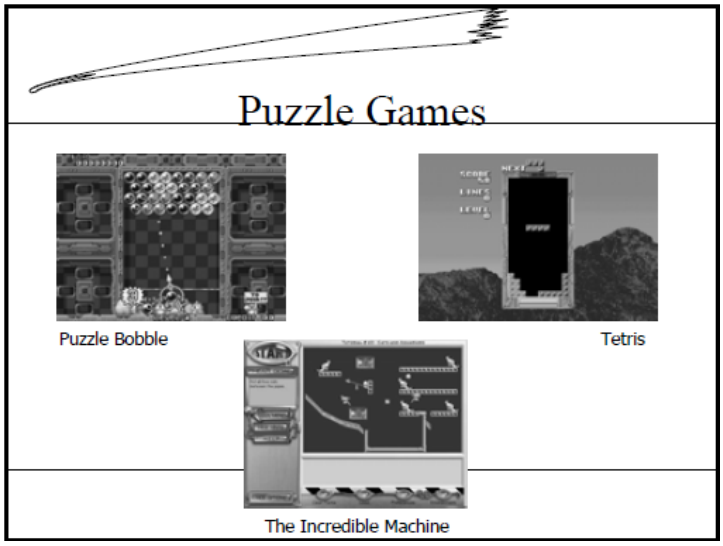
Dirancang untuk mengajar dan menghibur. Umumnya ditujukan kepada pamain yang relatif lebih muda dari kebanyakan produk *game*[4].



Gambar 2.10 Contoh *Educational Game*[4]

K. ***Puzzle Game***

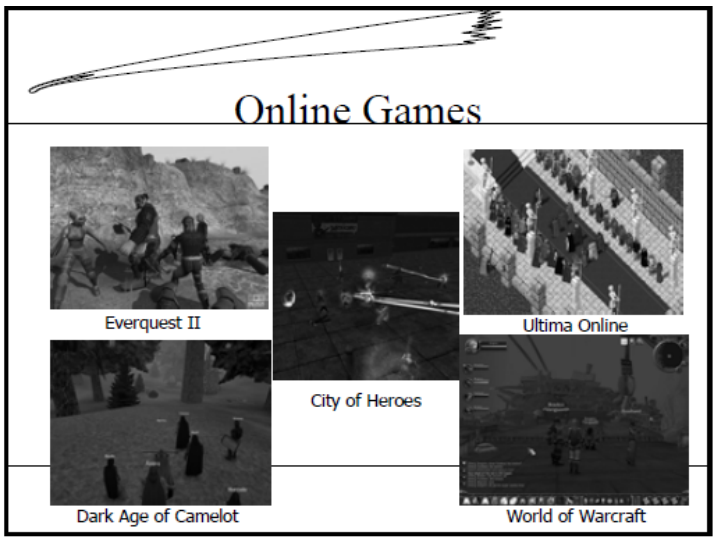
Permainan ini memberikan tantangan intelektual untuk menyelesaikan masalah. Teka-teki merupakan tujuan dari *game* ini dan tidak terdapat cerita di dalamnya[4].



Gambar 2.11 Contoh dari *Puzzle Game*[4]

L. ***Online Game***

Online Game dapat mencakup beberapa *genre* dan yang menjadi fitur utama adalah bermain dengan *multiplayer* dalam sebuah jarigan. Biasanya hanya terdapat 2-4 pemain, tetapi dapat melibat dengan puluhan, ratusan atau bahkan ribuan pemain. Dikenal sebagai Massively Multiplayer (Massively Multiplayer Online Role-Playing Game or MMORPG). Seringkali pemain berkomunikasi dalam permaian ini[4].



Gambar 2.12 Contoh *Online Game*[4]

2.1.3 Konsep *Game*

Menurut J. P. Flynt dan O. Salem dalam bukunya[5], terdapat beberapa syarat yang dibutuhkan dalam membangun sebuah aplikasi permainan (*game production*) sehingga aplikasi permainan yang dibangun memiliki karakteristik tersendiri, diantaranya diterapkannya *Audio* yang membuat sebuah aplikasi permainan semakin menarik saat dimainkan, sebuah algoritma yang menangani deteksi tubrukan (*collision detection*) antara objek yang satu dengan yang lainnya, 3D/2D komponen yang digunakan, aplikasi yang digunakan untuk melakukan *design* berbagai objek yang ada pada aplikasi permainan tersebut, dan lain-lain.

2.2 Kecerdasan Buatan *(Artificial Intelligence)*

2.2.1 Definisi AI

Sebagian kalangan menerjemahkan *Artificial Intelligence* sebagai kecerdasan buatan, kecerdasan artifisial, intelijensia artifisial, atau intelejensia buatan. Pada bukunya[6], Suyanto sengaja tidak menerjemahkan istilah *Artificial Intelligence* ke bahasa Indonesia karena istilah tersebut sudah sangat akrab bagi orang Indonesia. Begitu juga dengan singkatan istilah tersebut, yaitu AI, sudah sangat melekat di berbagai media ilmiah maupun non ilmiah.

Para ahli mendefinisikan AI secara berbeda-beda tergantung pada sudut pandang mereka masing-masing. Ada yang focus pada logika berpikir manusia saja, tetapi ada juga yang mendefinisikan AI secara lebih luas pada tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokkan definisi AI, yang diperoleh dari beberapa *textbook* berbeda, ke dalam empat kategori[6], 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 diinterogasi 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.

2.2.2 Teknik Pemecahan Masalah AI

Terdapat empat teknik dasar pemecahan masalah yang terdapat pada bidang *academic/traditional artificial intelligence* (AI)[6], diantaranya adalah sebagai berikut :

A. *Searching*

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

Metode-metode pencarian pada teknik *searching* diantaranya[6] :

1. *Blind/Un-informed Search*
   a. *Breadth-First Search* (BFS)
   b. *Depth-First Search* (DFS)
   c. *Depth-Limited Search* (DLS)
   d. *Uniform Cost Search* (USC)
   e. *Iterative-Deepening Search* (IDS)
   f. *Bi-Directional Search* (BDS)
2. Metode Pencarian Heuristik
   a. *Generate-and-Test*
   b. *Hill Climbing*
   c. *Simulated Annealing*
   d. *Best-First Search* (BFS)
   e. *Greedy Best-First Search*
   f. *A** (A *star*)

B. *Reasoning*

Teknik *reasoning* atau penalaran merupakan teknik penyelesaian masalah dengan cara merepresentasikan masalah ke dalam basis pengetahuan (*knowledge base)* menggunakan *logic* atau bahasa formal (bahasa yang dipahami komputer).

Teknik ini melakukan proses penalaran berdasarkan basis pengetahuannya untuk menemukan solusi.

C. *Planning*

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

D. *Learning*

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

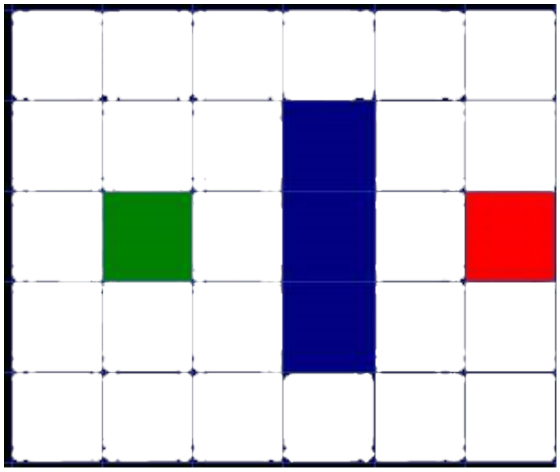
2.2.2.1 Algoritma A* (*A Star Algorithm*)

Algoritma A* (A *star*) merupakan algoritma *Best First Search* yang menggabungkan *Uniform Cost Search* dan *Greedy Best-First Search*[6]. Biaya yang diperhitungkan didapat dari biaya sebenarnya ditambah dengan biaya perkiraan, dalam notasi matematika dituliskan sebagai berikut :

$$f(n) = g(n) + h(n) \qquad (2.1)$$

Dengan perhitungan biaya seperti diatas, algoritma A* adalah *complete* dan *optimal*[6]. Untuk lebih jelasnya berikut merupakan gambaran dari penggunaan algoritma A* dalam pencarian rute terpendek dari *point* A menuju *point* B :

Asumsikan ada seseorang yang pergi dari *point* A menuju *point* B dan rute yang akan dilewati dihalangi sebuah dinding. Pada ilustrasi ini kotak berwarna hijau merupakan *point* A, kotak berwarna merah merupakan *point* B, dan kotak berwarna biru merupakan dinding yang menghalangi rute *point* A menuju *point* B[7].



Gambar 2.13 Ilustrasi awal Algoritma A*[7]

Hal pertama yang harus dilakukan adalah membuat sebuah *grid layout*, karena dalam pencarian rute pada algoritma A* akan lebih mudah apabila peta yang ada berbentuk kotak-kotak kecil, metode ini memudahkan pencarian dalam area yang didefinisikan sebagai array dua dimensi, setiap kotak memiliki status *walkable* dan *unwalkable*.

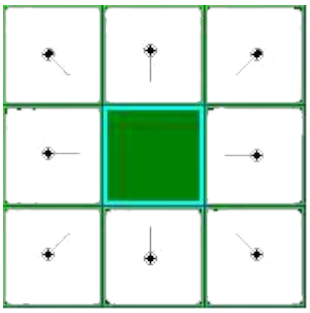
Pusat *point* pada algoritma A* dinamakan *node*, hal ini dikarenakan area yang dapat digunakan algoritma A* tidak hanya berbentuk kotak, akan tetapi dapat berbentuk yang lainnya, namun terkadang untuk memudahkan dalam perhitungan maka digunakanlah bentuk kotak.

Langkah-langkah yang dilakukan pada algoritma A* dalam melakukan pencarian rute terpendek dibagi menjadi beberapa tahap, yaitu :

A. Dimulai dari *point* A awal dan masukkan ke dalam kotak "*open list*". "*open list"* merupakan sebuah kotak yang berfungsi untuk menyimpan setiap *nod*, baik yang akan kita lewati maupun yang tidak kita lewati, pada dasarnya setiap kotak yang ada pada "*open list*" akan dilakukan perhitungan jarak untuk menentukan arah mana yang akan diambil menuju *point* B.
B. Lihat semua kotak yang memungkinkan untuk dilewati dari *point* awal, abaikan kotak yang terdapat penghalang seperti tembok, air, dan lain-lain. Masukan kotak-kotak tersebut ke dalam *open list*, dan untuk setiap kotak ini, simpan *point* A sebagai induk kotak.

C. Keluarkan kotak *point* awal dari *open list* kemudian masukkan ke dalam *closed list. Closed list* berfungsi untuk menyimpan kotak yang tidak akan pernah dipakai ataupun dilihat lagi.

Kotak berwarna hijau merupakan pusat dari *point* awal, pada sisinya dikelilingi oleh garis berwarna biru yang berarti bahwa kotak tersebut telah dimasukkan ke dalam *closed list*, dan untuk setiap garis disekeliling kotak awal menunjukkan setiap jalur yang dapat ditempuh dimulai dari kotak pada *point* awal.



Gambar 2.14 Ilustrasi kotak awal sebagai *closed list*[7]

Perhitungan yang terdapat pada algoritma A* untuk melakukan pencarian rute terpendek adalah dengan menggunakan rumus sebagai berikut :

$$F = G + H \quad (2.1)$$

dimana,

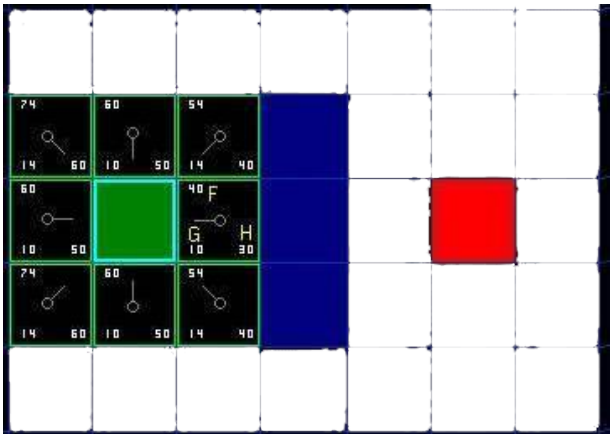
G : nilai yang dibutuhkan untuk menempuh jarak dari *point* awal menuju kotak selanjutnya

H : estimasi nilai yang dibutuhkan untuk menempuh jarak dari kotak selanjutnya untuk menempuh *point* tujuan.

Dalam proses perhitungannya dapat dimisalkan sebuah nilai untuk G baik yang terdapat pada kotak horizontal, vertikal, dan diagonal. Pada kasus ini dimisalkan nilai horizontal dan vertikal dari *point* awal untuk menuju kotak selanjutnya ialah 10, dan untuk nilai diagonalnya ialah 14.

Setiap kotak memiliki nilai F, G, dan H masing-masing, jadi perhitungan tersebut berlaku bagi setiap kotak yang berada di sekitar kotak induk, untuk nilai

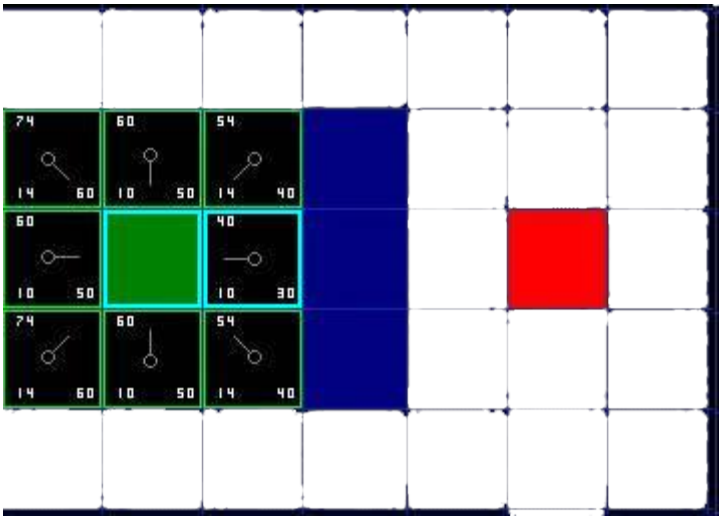
F ditulis di atas sebelah kiri, nilai G ditulis di bawah sebelah kiri, dan nilai H ditulis di bawah sebelah kanan.



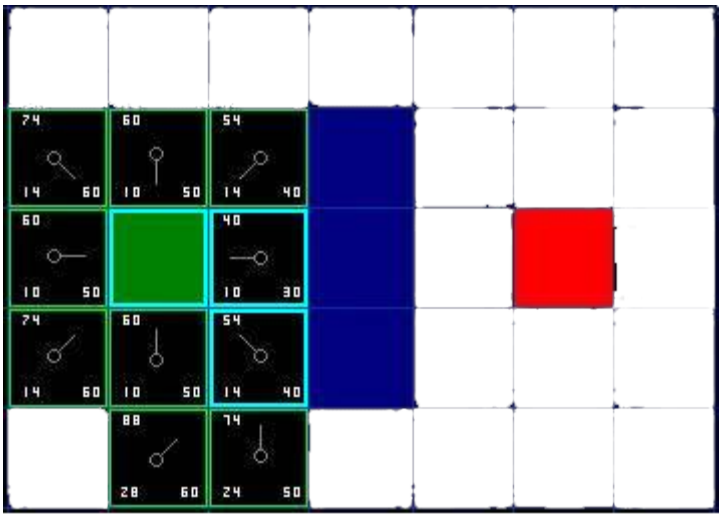
Gambar 2.15 Ilustrasi perhitungan F pada *open list*[7]

Langkah selanjutnya yang harus dilakukan adalah sebagai berikut :

D. Pilih nilai F yang terkecil dari setiap kotak yang telah dihitung nilai F nya, kemudian keluarkan dari *open list* dan masukkan ke dalam *closed list.*
E. Cek semua kotak yang berada di sekitar, abaikan kotak yang *unwalkable* dan kotak yang terdapat pada *closed list*, masukkan kotak yang belum ada di dalam *open list* dan jadikan kotak selanjutnya menjadi kotak induk untuk melakukan perhitungan ulang untuk setiap kotak disekitarnya.
F. Ketika sebuah kotak sekitar telah berada di dalam *open list*, cek apakah jalur kotak tersebut adalah yang terbaik dibanding yang jalur kotak yang lainnya. Di lain pihak, ketika nilai G untuk jalur yang baru itu paling kecil, ganti induk kotak pada kotak sekitarnya menjadi kotak yang telah terpilih, dan terakhir hitung ulang nilai F, G, dan H untuk kotak tersebut.

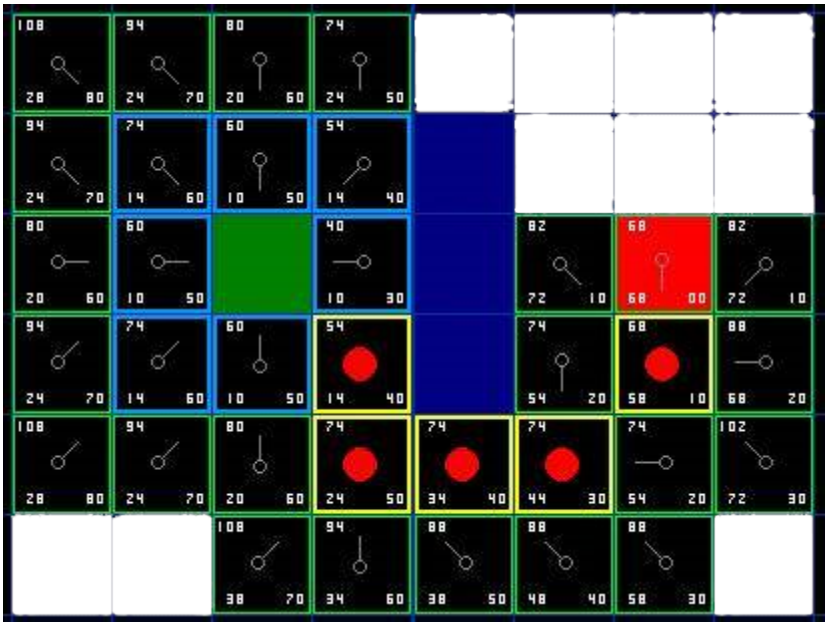


Gambar 2.16 Ilustrasi penentuan kotak selanjutnya[7]



Gambar 2.17 Ilustrasi perhitungan terhadap kotak *unwalkable*[7]

Setelah mengulangi setiap langkah yang ada di atas, perhitungan dapat dihentikan ketika *point* akhir atau yang disebut kotak target telah dimasukkan ke dalam *closed list*.



Gambar 2.18 Ilustrasi hasil akhir pencarian rute terpendek[7]

2.2.2.*2Waypoint*

Waypoint search merupakan pencarian yang efisien dan menghemat komsumsi pada CPU-*time*. Hampir semua *game* sekarang ini menggunakan metode *waypoint* sebagai *search*.

Pola gerakan yang paling dasar untuk karakter dalam permainan hanyalah sebuah garis lurus baik dalam bidang horisontal atau vertikal. Biasanya, karakter akan bergerak ke kiri dan kanan antara dua titik dan batas mungkin tepi layar atau sesuatu yang berhubungan dengan latar belakang peta. Contohnya dapat dilihat pada gerakan dasar dalam *Space Invaders*. Alien dimulai di sebelah kiri dan berbaris di layar ke kanan. Ketika mereka mencapai tepi layar, mereka membalikkan arah mereka dan berbaris ke kiri[15].

Karakter dalam permainan *platform* yang sering memiliki pola pergerakan yang sama dan mereka akan berpatroli *platform*, bergerak ke kiri atau kanan dan membalikkan arah saat mereka mencapai tepi. Jenis gerakan sangat mudah untuk menerapkan dan dibutuhkan sangat sedikit kode. Lima variabel yang diperlukan untuk karakter yang bergerak kiri dan kanan, dan ini adalah x, y, dx, x1 dan x2. Koordinat karakter disimpan dalam x, y dan arah yang diadakan di dx. Nilai ini

adalah jumlah piksel untuk bergerak setiap waktu dan dapat menjadi positif atau negatif, kecil atau besar. Dua variabel lainnya, x1 dan x2 adalah batas kiri dan kanan. Karakter ini dipindahkan dengan menambahkan dx ke koordinat x hingga mencapai baik x1 atau x2, maka dx menjadi-dx untuk membalikkan arah. Berikut adalah kode:

```
1   Global x = 320
2   Global y = 240
3   Global dx = 4
4   Global x1=200
5   Global x2=400
6
7   Repeat
8     Cls
9     MoveAlien()
10    Flip
11    Delay 100
12  Until KeyDown(1)
13  End
14  Function MoveAlien()
15    x=x+dx
16    If x<x1 Or x>x2 Then dx=-dx
17    DrawImage alien,x,y
18  End Function
```

Ini melintasi karakter jalur horisontal antara dua titik pada layar pada setiap kecepatan. Anda hanya perlu mengatur variabel pada awal daftar. Untuk membuatnya lebih menarik, posisi awal (x, y) dan kecepatan (dx) bisa nomor acak. Untuk melakukan langkah karakter vertikal, hanya mengubah dx ke dy dan x1 dan x2 untuk y1 dan y2. Kemudian tambahkan dy ke y mengkoordinasikan setiap kali karakter tersebut akan dipindahkan.

Banyak permainan beberapa karakter yang bergerak dalam garis horisontal atau vertikal dan Anda hanya perlu untuk menduplikasi kode di atas untuk masing-masing. Sebuah permainan yang memiliki enam karakter horisontal bergerak dan empat yang vertikal tidak lebih keras untuk menulis karena kode adalah sama. Dengan permainan *multi-level*, maka bisa menyimpan posisi awal

dari, arah karakter dan batas-batas sebagai daftar yang dibaca ketika tingkat dimasukkan.

2.3 *Object Oriented Programming* (OOP)

Object Oriented Programming (OOP) menerapkan sifat yang lebih modular agar setiap program dapat lebih mudah dikembangkan. Dalam OOP dibutuhkan memori lebih besar dibandingkan dengan program procedural (tradisional)[10]. Dua objek yang identik akan memerlukan dua area memori berbeda walaupun dari sisi data dan proses keduanya memiliki jumlah dan jenis yang sama. Hal ini disebabkan karena data dan proses pada kedua objek tersebut dipisahkan oleh komputer.

Secara garis besar yang menjadi ciri dari OOP adalah adanya proses abstraksi (*abstraction*), pengkapsulan (*encapsulation),* penurunan sifat (*inheritance*), dan polimorfisme (*polymorphism*) pada objek-objek yang dibentuk.

Object Oriented Programming (OOP) dibagi menjadi beberapa cirri utama[10], yaitu :

A. **Kelas**

Kelas (*class*) merupakan contoh abstrak dari sebuah objek yang telah terbentuk dari proses penyederhanaan, dengan kata lain kelas (*class*) merupakan cikal bakal dari objek (*object*), kemudian contoh nyata atau perwujudan dari sebuah objek dinamakan *instance*. Sehingga apabila kita mempunyai sebuah kelas manusia, maka beberapa *instances* (wujud nyata) dari kelas manusia adalah Prima, Aulia, Dewi, dan masih banyak yang lainnya.

Perbedaan antara kelas (*class*) dengan objek (*object*) dalam OOP dibagi menjadi dua[10], yaitu :

1. *Class* merupakan rancangan (*design*) dan *object* merupakan perwujudan dari suatu *class.*
2. *Class* bersifat abstrak sementara *object* bersifat konkrit (atau nyata).

B. Objek

Dalam kenyataannya, sebuah objek dalam OOP adalah sebuah persilangan yang berbagi-pakai (*share*) sejumlah ciri dari objek umum dengan fitur (*feature*) dari sebuah bentuk komputer[10].

Sebuah objek secara praktis pemrograman berorientasi objek bisa didefinisikan sebagai berikut :

1. Setiap objek dimiliki oleh kelas objek, sehingga sebuah objek tidak bisa hadir tanpa sebuah kelas yang mendefinisikannya. Dengan kata lain objek adalah wujud (*instance*) dari sebuah kelas.
2. Sebuah objek (dan kelas yang memuatnya) adalah sebuah pengkapsulan (*encapsulation*) yang memasukkan data dan operasi untuk pemrosesannya.
3. Atribut-atribut (*attributes*) objek membantu untuk menyimpan dan menjaga status objek. Atribut-atribut ini menentukan apa yang dengan mengenai objek. Methode objek adalah satu-satunya cara untuk mengakses data dan memodifikasi statusnya. Cara pengaksesan dan pemodifikasian data dilakukan dengan mengirimkan sebuah pesan ke objek tersebut.

C. Abstraksi

Abstraksi dapat didefinisikan sebagai suatu proses melakukan desain *class* dan menentukan data dan *method* yang akan dimiliki oleh sebuah *class*[10].

Sebuah *method* abstrak mendefinisikan sebuah antarmuka dalam kelas dasar dan meninggalkan implementasi pada kelas turunan. Kelas abstrak adalah sebuah kelas yang berisi satu atau beberapa *method* abstrak.

D. Pengkapsulan

Pengkapsulan (*encapsulation*) merupakan proses pembungkusan atau penyederhanaan dari beberapa data atau *method* menjadi sebuah objek (*object*) atau kelas (*class*)[10].

E. Pewarisan Sifat

Penurunan atau pewarisan sifat (*inheritance*) ini merupakan cirri utama dari OOP dimana sifat-sifat yang terdapat pada kelas induk (*base class*) akan dimiliki oleh kelas turunannya (*derived class*)[10]. Akan tetapi hal itu tentunya bergantung

juga pada *access specifier* (yaitu, *public* dan *private*) yang diberikan dalam proses penurunan kelas.

2.4 *Unified Modelling Language* (UML)

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.

2.4.1 Diagram UML

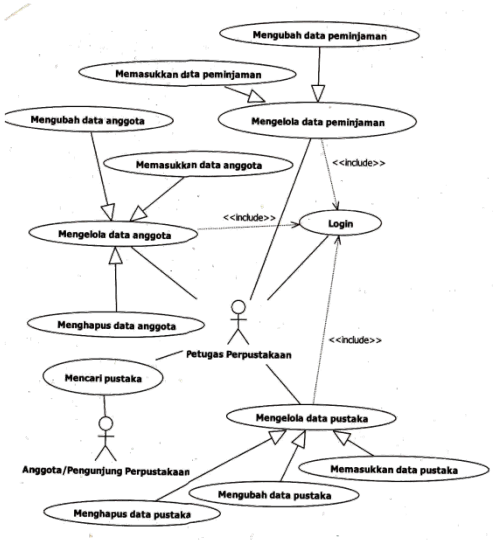
UML menggunakan berbagai macam diagram dengan fungsi masing-masing untuk menggambarkan setiap proses dari sistem berorientasi objek. Berikut merupakan beberapa diagram UML diantaranya[10] :

A. ***Use Case Diagram***

Use Case atau diagram *use case* merupakan pemodelan yang digunakan untuk menggambarkan kelakuan (*behavior)* dari sistem yang akan dibuat[10]. *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].

1. 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.
2. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesar antarunit atau aktor.



Gambar 2.13 Contoh dari *Use Case Diagram*[10]

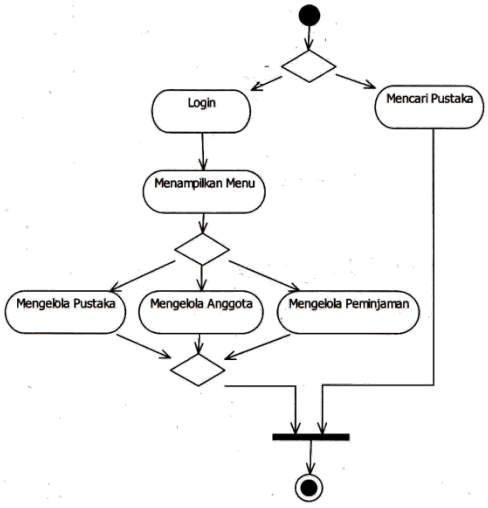
B. ***Activity Diagram***

Diagram aktivitas atau *activity diagram* adalah sebuah diagram yang menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis[10]. Dalam diagram aktivitas yang perlu diperhatikan adalah bahwa diagram aktivitas menggambarkan aktivitas sistem, bukan apa yang dilakukan aktor, jadi aktivitas yang dapat dilakukan oleh sistem.

Diagram aktivitas juga banyak digunakan untuk mendefinisikan hal-hal berikut[10] :

1. Rancangan proses bisnis di mana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
2. Urutan atau pengelompokan tampilan dari sistem/*user interface* di mana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.

3. Rancangan pengunjian di mana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.

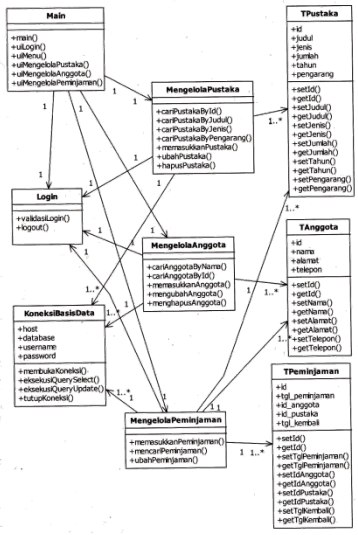


Gambar 2.20 Contoh dari *Activity Diagram*[10]

C. ***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[10].

1. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas
2. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas

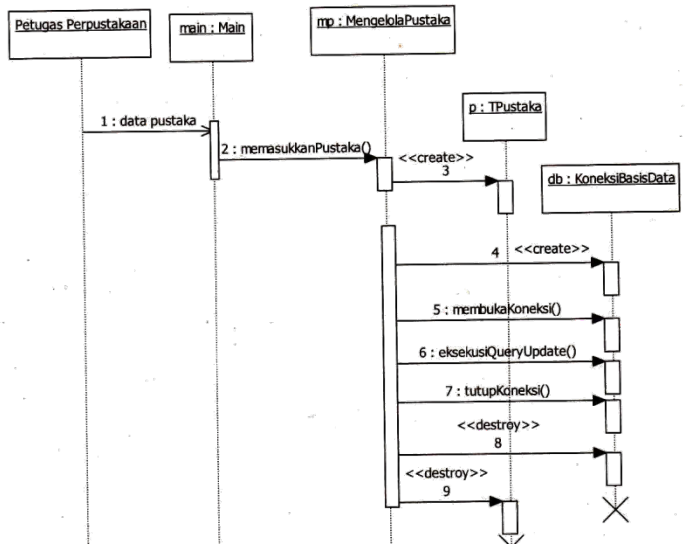


Gambar 2.20 Contoh dari *Class Diagram*[10]

D. ***Sequence Diagram***

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

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

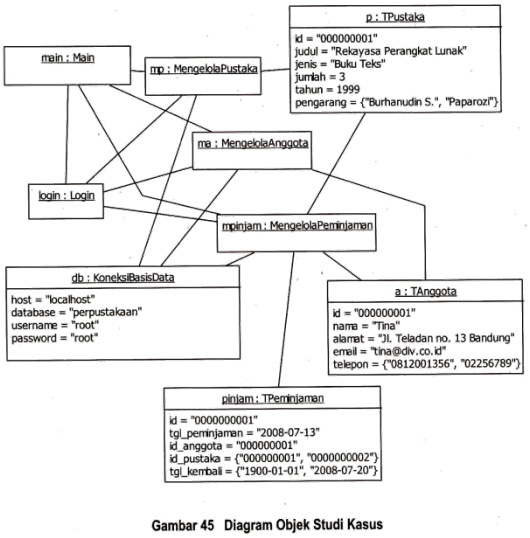


Gambar 2.21 Contoh dari *Sequence Diagram*[10]

E. ***Object Diagram***

Diagram objek menggambarkan struktur system dari segi penamaan objek dan jalannya objek dalam sistem[10]. Pada diagram objek harus dipastikan semua kelas yang sudah didefinisikan pada diagram kelas harus dipakai objeknya, karena jika tidak, pendefinisian kelas itu tidak dapat dipertanggungjawabkan.

Untuk apa mendefinisikan sebuah kelas sedangkan pada jalannya sistem, objeknya tidak pernah dipakai. Hubungan *link* pada diagram objek merupakan hubungan memakai dan dipakai di mana dua buah objek akan dihubungkan oleh *link* jika ada objek yang dipakai oleh objek lainnya.



Gambar 45 Diagram Objek Studi Kasus

Gambar 2.22 Contoh dari *Object Diagram*[10]

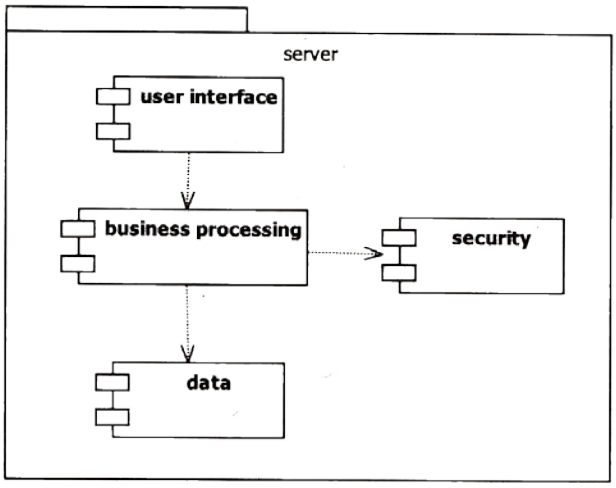
F. ***Component Diagram***

Diagram komponen dibuat untuk menunjukkan organisasi dan kebergantungan di antara kumpulan komponen dalam sebuah sistem. Diagram komponen focus pada komponen sistem yang dibutuhkan dan ada di dalam sistem[10]. Diagram komponen juga dapat digunakan untuk memodelkan hal-hal berikut[10] :

1. *Source code* program perangkat lunak
2. Komponen *executable* yang dilepas ke *user*
3. Basis data secara fisik
4. Sistem yang harus beradaptasi dengan sistem lain
5. *Framework* sistem

Adapun komponen-komponen dasar yang biasanya ada dalam suatu sistem adalah sebagai berikut[10] :

1. Komponen *user interface* yang menangani tampilan
2. Komponen *business processing* yang menangani fungsi-fungsi proses bisnis
3. Komponen *data* yang menangani manipulasi data
4. Komponen *security* yang menangani keamanan sistem



Gambar 2.23 Contoh dari *Component Diagram*[10]

G. ***Composite Structure Diagram***

Diagram ini dapat digunakan untuk menggambarkan struktur dari bagian-bagian yang saling terhubung maupun mendeskripsikan struktur pada saat berjalan (*runtime*) dari *instance* yang saling terhubung[10]. Contoh penggunaan diagram ini misalnya untuk menggambarkan deskripsi dari setiap bagian mesin yang saling terkait *router* pada jaringan komputer, dll.

H. ***Package Diagram***

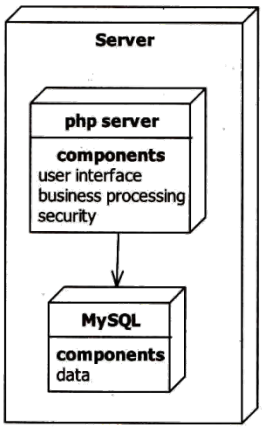
Package diagram menyediakan cara mengumpulkan elemen-elemen yang saling terkait dalam diagram UML[10].

I. ***Deployment Diagram***

Diagram deployment atau *deployment diagram* menunjukkan konfigurasi komponen dalam proses eksekusi aplikasi[10]. Diagram *deployment* juga dapat digunkan untuk memodelkan hal-hal berikut :

1. Sistem tambahan (*embedded system*) yang menggambarkan rancangan *device, node,* dan selanjutnya

2. Sistem *client/server*
3. Sistem terdistribusi murni
4. Rekayasa ulang aplikasi

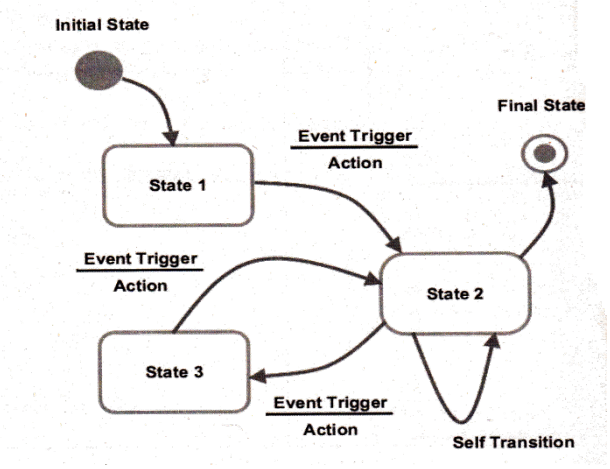


Gambar 2.24 Contoh dari *Deployment Diagram*[10]

J. ***State Machine Diagram***

Diagram mesin status digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem[10]. Perubahan tersebut digambarkan dalam suatu graf berarah. *State machine diagram* merupakan pengembangan dari diagram *Finite State Automata* dengan penambahan beberapa fitur dan konsep baru.

Diagram ini cocok digunakan untuk menggambarkan alur interaksi pengguna dengan sistem[10].



Gambar 2.25 Contoh dari *State Machine Diagram*[10]

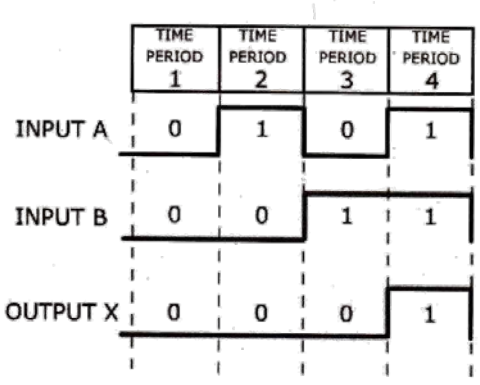
K. ***Communication Diagram***

Diagram komunikasi merupakan penyederhanaan dari diagram kolaborasi (*collaboration diagram*)[10]. Diagram ini menggambarkan interaksi antarobjek/bagian dalam bentuk urutan pengiriman pesan. Diagram komunikasi merepresentasikan informasi yang diperoleh dari Diagram Kelas, Diagram Sekuen, dan Diagram *Use Case* untuk mendeskripsikan gabungan antara struktur statis dan tingkah laku dinamis dari suatu sistem.

Diagram komunikasi mengelompokkan *message* pada kumpulan diagram sekuen menjadi sebuah diagram. Dalam diagram komunikasi yang dituliskan adalah operasi/metode yang dijalankan antara objek yang satu dan objek yang lainnya secara keseluruhan, oleh karena itu dapat diambil dari jalannya interaksi pada semua diagram sekuen. Penomoran metode dapat dilakukan berdasarkan urutan dijalankannya metode/operasi di antara objek yang satu dengan objek yang lainnya atau objek itu sendiri.

L. ***Timing Diagram***

Timing Diagram merupakan diagram yang focus pada penggambaran terkait batasan waktu. *Timing diagram* digunakan untuk menggambarkan tingkah laku sistem dalam periode waktu tertentu[10]. *Timing diagram* biasanya digunakan untuk mendeskripsikan operasi dari alat *digital* karena penggambaran secara visual akan lebih mudah dipahami daripada dengan kata-kata. Aliran waktu pada *timing diagram* dibaca dari kiri ke kanan.

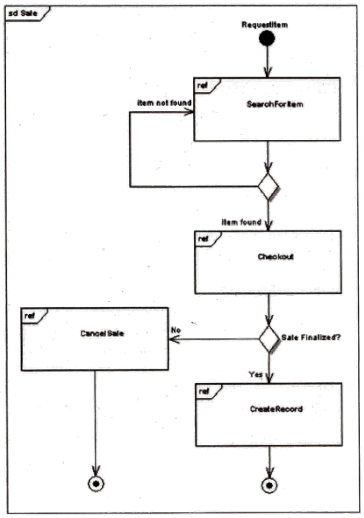


Gambar 2.26 Contoh dari *Timing Diagram*[10]

M. ***Iteraction Overview Diagram***

Iteraction overview diagram mirip dengan diagram aktivitas yang berfungsi untuk menggambarkan sekumpulan urutan aktivitas. *Iterraction overview diagram* adalah bentuk aktivitas diagram yang setiap titik merepresentasikan diagram interaksi[10]. Interaksi diagram dapat meliputi diagram sekuen, diagram komunikasi, *interaction overview diagram*, dan *timing diagram*.

Hampir semua notasi pada *interaction overview diagram* sama dengan notasi pada diagram aktivitas. Sebagai contoh *initial, final, decision, merge, fork,* dan *join nodes* sama seperti pada diagram aktivitas. Tambahan pada *interaction overview diagram* adalah *interaction accurrence* dan *interaction element*.



Gambar 2.14 Contoh dari *Iteraction Overview Diagram*[10]

2.5 Teknik Pengujian Perangkat Lunak

Pengujian perangkat lunak adalah elemen kritis dari jaminan kualitas perangkat lunak dan merepresentasikan kajian pokok dari spesifikasi, desain, dan pengkodean[11].

2.5.1 Pengujian *Black Box*

Menurut Roger S. Pressman[11], pengujian *black box* berfokus pada persyaratan fungsional perangkat lunak. Dengan demikian, pengujian *black box* menungkinkan perekayasa perangkat lunak mendapatkan serangkaian kondisi input yang sepenuhnya menggunakan semua persyaratan fungsional untuk suatu program. Pengujian *black box* bukan merupakan alternatif dari teknik *white box*, tetapi merupakan pendekatan komplementer yang kemungkinan besar mampu mengungkap kelas kesalahan daripada metode *white box*.

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

A. Fungsi-fungsi yang tidak benar atau hilang
B. Kesalahan dalam *interface*
C. Kesalahan dalam struktur data atau akses *database* eksternal
D. Kesalahan kinerja
E. Inisialisasi dan kesalahan terminasi

2.5.2 Pengujian *White Box*

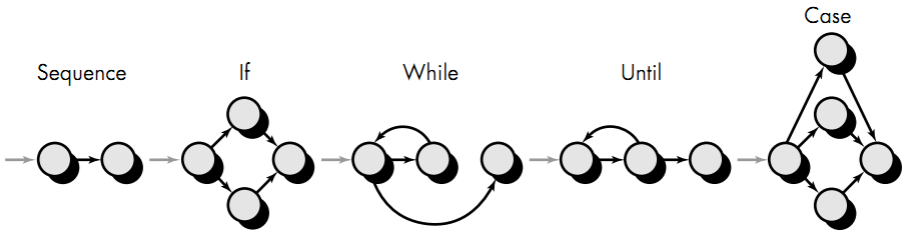
Menurut Roger S. Pressman[11], pengujian *white box*, yang kadang-kadang disebut pengujian *glass box*, adalah metode desain *test case* yang menggunakan struktur kontrol desain procedural untuk memperoleh *test case*. Dengan menggunakan metode pengujian *white box*, perekayasa sistem dapat melakukan *test case* sebagai berikut :

A. Memberikan jaminan bahwa semua jalur independen pada suatu modul telah digunakan paling tidak satu kali
B. Menggunakan semua keputusan logis pada sisi *true* dan *false*

C. Mengeksekusi semua *loop* pada batasan mereka dan pada batas operasional mereka

D. Menggunakan struktur data internal untuk menjamin validitasnya.

Pengujian *white box* yang berupa notasi diagram alir dapat dilihat pada gambar 2.20.



Gambar 2.28 Notasi Diagram Alir[11]

2.6 *Tools* yang Digunakan

Dalam pembuatan sebuah aplikasi tentu dibutuhkan satu atau beberapa *tools* yang dalam hal ini berupa *software* (perangkat lunak). Berikut merupakan beberapa perangkat lunak yang digunakan untuk membangun sebuah aplikasi permainan *The Tale of Hunter*.

2.6.1 *Java*

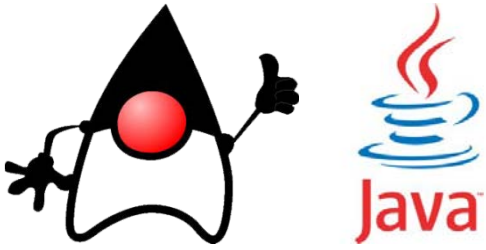
Java adalah salah satu bahasa pemrograman komputer yang berorientasi objek[12], yang diciptakan oleh satu tim dari perusahaan *Sun Microsystem*, perusahaan *workstation* UNIX (*Sparc*) yang cukup terkenal. *Java* diciptakan berdasarkan bahasa C++, dengan tujuan *platform independent* (dapat dijalankan pada berbagai jenis *hardware* tanpa kompilasi ulang), dengan slogan "*Write Once, Run Anywhere* (WORA), dan *Java* pada hakikatnya lebih sederhana dan memakai objek secara murni dibanding dengan bahasa pemrograman C++.

Asal usul *java* dimulai pada tahun 1991 ketika *Sun Microsystem* melakukan penelitian terhadap berbagai produk elektronika. James Gosling (kepala proyek penelitian) beserta Patrick Naughton ditugaskan untuk merancang perangkat lunak aplikasi yang independen, tidak bergantung pada jenis perangkat keras, agar bisa dipakai pada berbagai peralatan elektronik itu. James Gosling

kemudian memakai bahasa pemrograman C++ untuk menulis beberapa aplikasi untuk peralatan mikro, namun ternyata dia mengalami banyak kesulitan. Setiap kali sebuah peralatan mikro menggunakan jenis *microchip* yang berbeda, program yang digunakannya harus dikompilasi ulang. Berdasarkan hal itu kemudian munculah ide dari James Gosling untuk menciptakan sebuah bahasa baru, yang ditulis berdasarkan C++ namun dengan beberapa perbaikan. Bahasa ini kemudian disebut *Oak* (nama pohon yang terlihat dari jendela ruangan kerja James Gosling), yang digunakan untuk membuat perangkat lunak cerdas bagi peralatan elektronika buatan *Sun Mirosystem.*

Pada tahun 1994, ketika Web mulai popiler nama bahasa *Oak* kemudian diganti menjadi *Java* (salah satu sebabnya ternyata nama *Oak* telah terlebih dahulu dipakai untuk hal lain). Menurut gosip, Gosling bingung mencari nama untuk bahasa baru tersebut karena setiap nama yang dipikirkannya untuk nama bahasa pemrogramannya ternyata sudah ada yang memakai, sampai suatu ketika dia mampir di sebuah kafe untuk minum kopi, dan ia meminta "*Java*". Timbullah idenya untuk memakai nama *Java*. Menurut Gosling nama ini cukup *cool*.

Java kemudian dipakai untuk membuat sebuah browser yang bernama *WebRunner*. Setelah berhasil maka browser ini kemudian dinamakan *HotJava* dengan simbol secangkir kopi panas.



Gambar 2.29 Maskot *Java* (Duke) dan *HotJava*[12]

2.6.2 *Greenfoot*

Greenfoot merupakan *tools* yang berorientasi objek dengan menggunakan bahasa pemrograman java [13]. Membuat *actors* yang berada di dalam *world* untuk membangun permainan, simulasi dan program grafis lainnya. Adapun *Interface* dari Greenfoot yang visual dan interaktif. *Tools* visualisasi dan interaksi dibangun kedalam *environment.*

Interface-nya terdapat IDE yang lengkap meliputi *project management*, *auto-completion*, *syntax highlighting* dan beberapa tools seperti kebanyakan IDE yang lain sehingga memungkinkan untuk mempublikasikannya. Greenfoot digunakan oleh ribuan institusi di seluruh dunia. Greenfoot juga sangat mudah untuk ditransisi ke *environment* lain, seperti BlueJ dan beberapa *professional* IDEs