BAB II
LANDASAN TEORI

2.1. Permainan

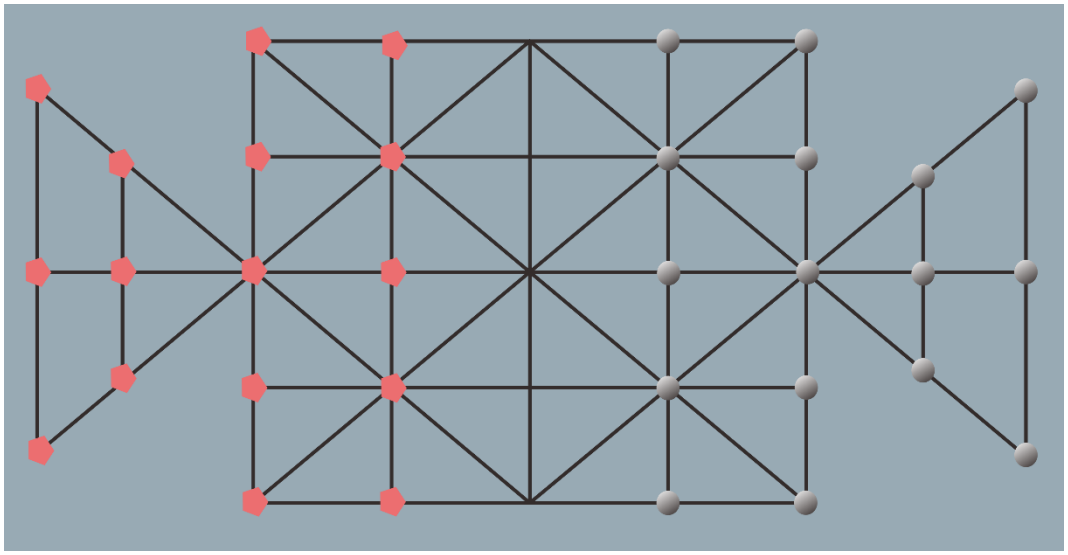
Istilah permainan dari kata dasar main. Menurut buku Kamus Besar Bahasa Indonesia, edisi kedua, terbitan Departemen Pendidikan dan Kebudayaan Balai Pustaka arti kata main adalah melakukan permainan untuk menyenangkan hati atau melakukan perbuatan untuk bersenang-senang baik menggunakan alat-alat tertentu atau tidak menggunakan alat. Jadi main adalah kata kerja, sedang permainan merupakan kata benda jadian untuk memberi sebutan pada sesuatu yang jika dilakukan dengan baik akan membuat senang hati si pelaku [6].

2.2. Permainan Dam Daman

Permainan dam daman termasuk permainan tradisional, permainan ini dimainkan oleh dua pemain, permainan ini dimainkan di papan atau karton yang telah digambar. Dalam permainan ini terdapat 16 bidak untuk setiap pemain.

Untuk memulai permainan dam daman, langkah-langkahnya sebagai berikut:

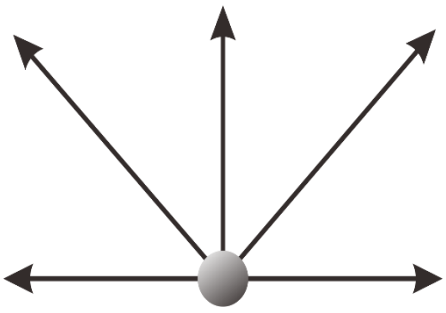
1. Pemain harus berjumlah 2 orang
2. Pemain harus mencari batu/kerikil untuk bidaknya tetapi setiap pemain tidak boleh sama
3. Jumlah bidak untuk setiap pemain ada 16 buah
4. Gambar papan permainan dam daman, seperti pada gambar 2.1
5. Susun setiap batu/kerikil tadi pada papan permainan seperti pada gambar 2.1
6. Lakukan suit untuk menentukan pemain giliran pertama.
7. Permainan berakhir apabila salah satu pemain sudah kehabisan bidak



Gambar 2.1 Papan Permainan Dam Daman

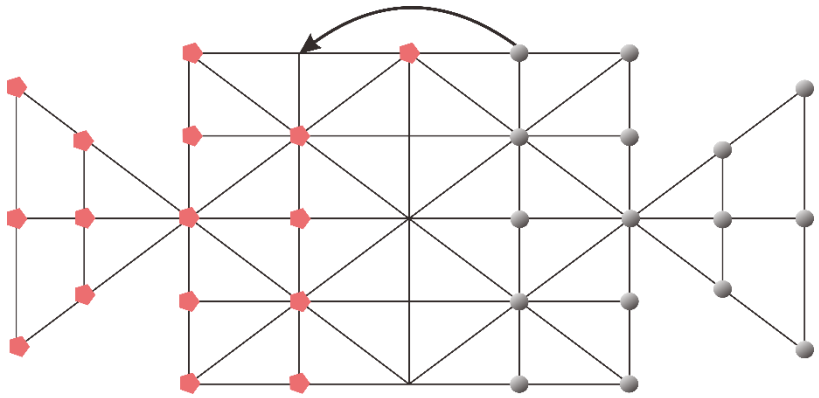
2.2.1. Langkah Bidak Pada Dam Daman

Langkah bidak pada permainan Damdaman terdiri dari dua langkah yang berbeda, yaitu langkah bidak biasa dan langkah bidak raja. Bidak biasa hanya bisa melakukan langkah ke samping, depan dan diagonal. Sedangkan bidak raja bisa bergerak bebas. Untuk bisa menjadi bidak raja harus melewati ujung yang berbentuk segitiga di daerah lawan atau bidak tersisa 3.



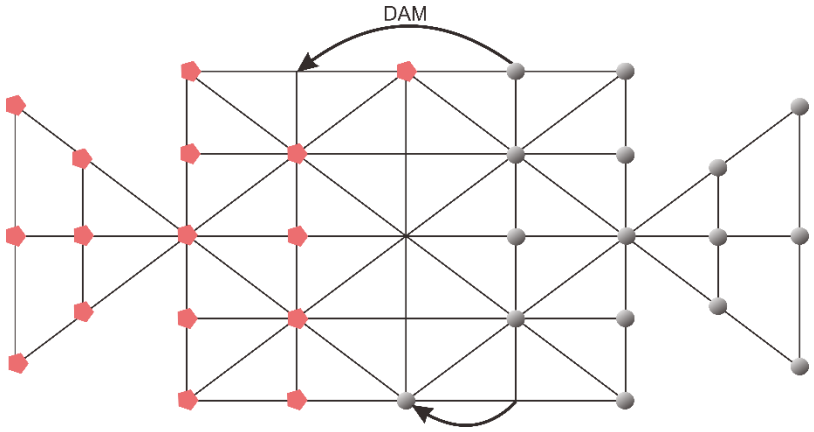
Gambar 2.2 Langkah Bidak

a. Untuk memakan bidak lawan dengan cara melewati bidak lawan.



Gambar 2.3 Cara Memakan Bidak

b. Jika ada kesempatan untuk memakan bidak lawan tapi tidak dilakukan, maka hal tersebut dinamakan dam. Dan lawan berhak mengambil 3 bidak secara bebas.



Gambar 2.4 Dam

c. Jika jumlah bidak tersisa 3 atau bidak melewati ujung dari daerah lawan, maka bidak tersebut dikatan sakti. Bidak yang sakti bisa bebas bergerak lebih dari satu langkah ke depan, ke belakang, ke samping ataupun bergerak diagonal.

2.3. *Artificial Intelligence*

Sebagian kalangan menerjemahkan *Artificial Intelligence* sebagai kecerdasan buatan, kecerdasan artifisial, intelijensia artifisial, atau intelijensia buatan. Istilah *Artificial Intelligence* sengaja tidak diterjemahkan ke dalam bahasa Indnesia karena istilah tersebut sudah sangat akrab bagi orang Indonesia. begitu juga dengan singkatannya yaitu AI, sudah sangat melekat di berbagai media ilmiah maupun non ilmiah.

Sturat Russel dan Peter Norvig mengelompokkan definisi AI, berbeda, ke dalam empat kategori, yaitu [7] :

1. *Thinking Humanly : the cognitive modeling approach*

Pendekatan ini dilakukan dengan dua cara sebagai berikut :

a. Melalui introspeksi : mencoba menangkap pemikiran-pemikiran kita sendiri pada saat kita terpikir. 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.
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 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.

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 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.

2.3.1. Teknik Pemecahan Masalah AI

1. *Searching*

Terdapat beberapa langkah dalam teknik *searching* atau pencarian ini.Langkah pertama yang harus dilakukan adalah mendefinisikan ruang masalah

untuk suatu masalah yang dihadapi. Ruang masalah ini dapat digambarkan sebagai himpunan keadaan (*state*) atau bisa juga sebagai

himpunan rute dari keadaan awal (*initial state*) menuju keadaan tujuan (*goal state*). Langkah kedua adalah mendefinisikan aturan produksi yang digunakan untuk mengubah suatu *state* ke *state* lainnya. Langkah terakhir adalah memilih metode pencarian yang tepat sehingga dapat

menemukan solusi terbaik dengan usaha yang minimal. Terdapat banyak metode pencarian yang telah diusulkan. Semua metode yang ada dapat dibedakan mendadi dua jenis, yaitu :

a. *Blind/Un-informed Search*

   1. *Breadth-First Search* (BFS)
   2. *Depth-First Search* (DFS)
   3. *Depth-Limited Search* (DLS)
   4. *Uniform Cost Search* (USC)
   5. *Iterative-Deepening Search* (IDS)
   6. *Bi-Directional Search* (BDS)

b. Metode Pencarian Heuristik

   1. *Generate-and-Test* (Bangkitkan-dan-Uji)
   2. *Hill Climbing* (Pendakian Bukit)
   3. *Simulated Annealing* (SA)
   4. *Best-First Search* (Pencarian Terbaik Terlebih Dahulu)
   5. *Greedy Best-First Search*

6. A* (A *star* atau A bintang)

7. *Iterative Deepening* A* (IDA*)

8. *Simplified Memory-Bounded* A* (SMA*)

9. *Bi-directional* A* (BDA*)

10. *Modified Bi-directional* A* (MBDA*)

11. *Dinamic Weighting* A* (DWA*)

2. *Reasoning*

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

3. *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.

4. *Learning*

Pada ketiga teknik sebelumnya, kita harus mengetahui aturan yang berlaku untuk sistem yang akan kita bangun. Tetapi pada masalah tertentu, terkadang kita tidak bisa mendefinisikan aturan secara benar dan lengkap karena data-data yang kita dapatkan tidak lengkap. Melalui suatu teknik yang disebut *learning*, kita bisa secara otomatis menemukan aturan yang diharapkan bisa berlaku umum untuk data – data yang belum kita ketahui [7].

2.4. Algoritma *Negamax*

Algoritma *negamax* merupakan bentuk sederhana dari algoritma minimax yang melakukan pencarian dengan menggunakan teknik algoritma DFS yang akan menelusuri setiap node untuk memperoleh hasil yang maksimum. Algoritma *negamax* identik dengan algoritma minimax untuk karakteristik kinerja. Meskipun sederhana untuk menerapkan dan cepat untuk mengeksekusi, itu skala dengan cara yang sama dengan pohon-pohon besar. Sebagian besar optimasi yang dapat diterapkan untuk *negamax*ing dapat dibuat untuk bekerja dengan ketat mendekati minimaxing [8].

Pseudocode algoritma dari *Negamax* adalah sebagai berikut:

```
1  def negamax(board, maxDepth, currentDepth):
2
3   # Check if we're done recursing
4   if board.isGameOver() or currentDepth == maxDepth:
5   return board.evaluate(), None
6
7   # Otherwise bubble up values from below
8
9   bestMove = None
10  bestScore = -INFINITY
11
12  # Go through each move
13  for move in board.getMoves():
14
15  newBoard = board.makeMove(move)
16
17  # Recurse
18  recursedScore, currentMove = negamax(newBoard,
19  maxDepth, currentDepth+1)
20  currentScore = -recursedScore
21
22  # Update the best score
23  if currentScore > bestScore:
24  bestScore = currentScore
25  bestMove = move
26
27  # Return the score and the best move
28  return bestScore, bestMove
```

Gambar 2.5 Algoritma *Negamax* [8]

2.5. *Alpha Beta Prunning*

Algoritma *negamax* efisien tapi memeriksa posisi papan lebih dari yang diperlukan. *Alpha Beta pruning* memungkinkan algoritma untuk mengabaikan bagian dari pohon yang tidak mengandung langkah yang terbaik . Hal ini terdiri dari dua jenis pemangkasan: *alpha* dan *beta* [8].

2.5.1. *Alpha prunning*

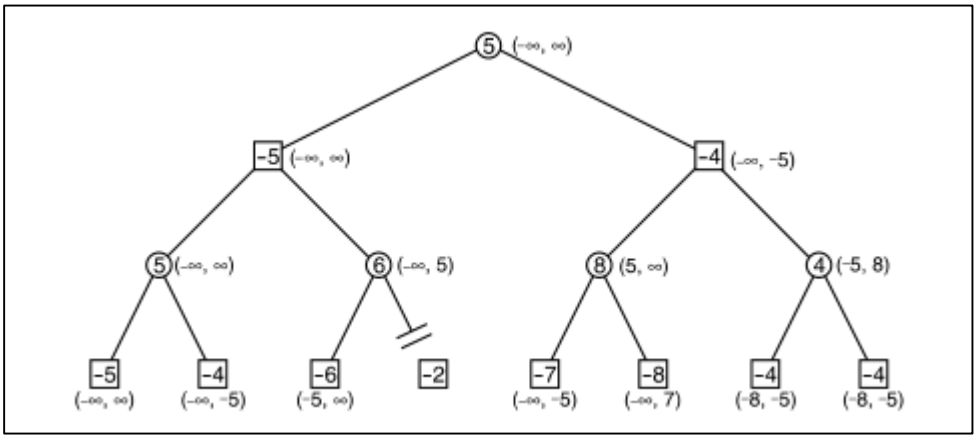
Untuk memangkas dengan cara ini, kita perlu melacak skor terbaik yang dapat dicapai. Faktanya, nilai ini membentuk batas bawah pada skor yang dicapai. Kemungkinan menemukan urutan yang lebih baik dari dari langkah dalam pencarian, tetapi kita tidak akan pernah menerima urutan langkah yang memberi kita nilai rendah. Ini terikat lebih rendah disebut nilai alpha (kadang-kadang, tapi jarang, ditulis sebagai huruf Yunani α), dan pemangkasan yang disebut *alpha pruning*. Dengan melacak nilai *alpha*, kita dapat menghindari mempertimbangkan langkah apapun dimana lawan memiliki kesempatan untuk membuatnya lebih buruk. Kita tidak perlu khawatir tentang berapa banyak langkah yang lebih buruk dari lawan yang bisa dibuatnya. kita sudah tahu bahwa kita tidak akan memberinya kesempatan [8].

2.5.2. *Beta Prunning*

Beta pruning bekerja dengan cara yang sama. Nilai beta (β lagi, jarang ditulis) melacak sebuah batas atas dengan harapan untuk mendapatkan nilai. Kami memperbarui nilai beta ketika kita menemukan urutan pergerakan yang lawan bisa untuk memaksa kita ke dalam.

Pada saat itu kami tahu tidak ada cara untuk mencetak lebih dari nilai beta, tapi mungkin ada beberap urutan belum ditemukan yang dapat digunakan untuk membatasi kita bahkan lebih jauh. Jika kita menemukan urutan dari gerakan bahwa skor lebih besar dari nilai beta, maka kita bisa mengabaikan itu, karena kita tahu kita akan tidak pernah diberi kesempatan. Alpha dan beta nilai bersama-sama memberikan jendela skor. Kita tidak akan pernah memilih untuk membuat gerakan yang skor kurang dari alpha, dan lawan kami tidak akan pernah membiarkan kita

membuat gerakan untuk mendapatkan nilai lebih dari beta. Jika cabang pohon ditemukan yang berada di luar nilai-nilai ini, maka cabang dapat dipangkas. Karena pergantian antara meminimalkan dan memaksimalkan untuk setiap pemain, hanya satu nilai yang perlu diperiksa pada setiap posisi papan. Pada posisi papan di mana itu adalah lawan untuk bermain, kita meminimalkan skor, sehingga hanya skor minimal dapat berubah dan kita hanya perlu memeriksa terhadap alpha. Jika itu adalah giliran kami untuk bermain, kita memaksimalkan nilai, dan sehingga hanya beta yang diperlukan [8].



Gambar 2.6 pohon pencarian algoritma *negamax* dengan *alpha beta pruning*

Berikut adalah *pseudocode* dari algoritma *negamax* dengan *alpa beta pruning*

```
def abNegamax(board, maxDepth, currentDepth, alpha, beta):

# Check if we're done recursing
if board.isGameOver() or currentDepth == maxDepth:
return board.evaluate(player), None

# Otherwise bubble up values from below

bestMove = None
bestScore = -INFINITY

# Go through each move
for move in board.getMoves():

newBoard = board.makeMove(move)

# Recurse
recursedScore, currentMove = abNegamax(newBoard,
maxDepth,
currentDepth+1
-beta,
-max(alpha, bestScore))
currentScore = -recursedScore

# Update the best score
if currentScore > bestScore:
bestScore = currentScore
bestMove = move

# If we're outside the bounds, then prune: exit immediately
if bestScore >= beta:
return bestScore, bestMove

return bestScore, bestMove
```

Gambar 2.7 Pseudocode algoritma *negamax* dengan *alpha beta pruning* [8]

2.6. Pemograman Berorientasi Objek

Metodologi berorientasi objek adalah suatu strategi pembangunan perangkat lunak yang mengorganisasikan perangkat lunak sebagai kumpulan objek yang berisi data dan operasi yang diberlakukan terhadapnya. Metodologi berorientasi objek merupakan suatu cara bagaimana sistem perangkat lunak dibangun melalui pendekatan objek secara sistematis. Metode berorientasi objek didasarkan pada penerapan prinsip-prinsip pengelolaan kompleksitas. Metode berorientasi objek meliputi rangkaian aktivitas analisis berorientasi objek, perancangan berorientasi objek, pemrograman berorientasi objek, dan pengujian berorientasi objek [9].

Pada saat ini, metode berorientasi objek banyak dipilih karena metodologi lama banyak menimbulkan masalah seperti adanya kesulitan pada saat mentransformasi hasil dari satu tahap pengembangan ke tahap berikutnya, misalnya pada metode pendekatan terstruktur jenis aplikasi yang dikembangkan saat ini berbeda dengan masa lalu. Aplikasi yang dikembangkan pada saat ini sangat beragam (aplikasi bisnis, *real-time*, *utility*, dan sebagainya) dengan platform yang berbeda-beda, sehingga membutuhkan tuntutan kebutuhan metodologi pengembangan yang dapat mengakomodasi ke semua jenis aplikasi tersebut.

Keuntungan menggunakan metodologi berorientasi objek adalah sebagai berikut :

1. Meningkatkan Produktivitas

   Karena kelas dan objek yang ditemukan dalam suatu masalah dapat dipakai ulang untuk masalah lainnya yang melibatkan objek tersebut (*reusable*).

2. Kecepatan Pengembangan

   Karena sistem yang dibangun dengan baik dan benar pada saat analisis dan perancangan akan menyebabkan berkurangnya kesalahan pada saat pengkodean.

3. Kemudahan Pemeliharaan

   Karena dengan model objek, pola-pola yang cenderung tetap dan stabil dapat dipisahkan dengan pola-pola yang mungkin sering berubah-ubah.

4. Adanya Konsistensi

   Karena sifat pewarisan dan penggunaan notasi yang sama pada saat analisis, perancangan maupun pengkodean.

5. Meningkatkan Kualitas Perangkat Lunak

   Karena pendekatan pengembangan lebih dekat dengan dunia nyatadan adanya konsistensi pada saat pengembangannya, perangkat lunak yang dihasilkan akan mampu memenuhi kebutuhan pemakai serta mempunyai sedikit kesalahan.

Saat ini sudah banyak bahasa pemrograman berorientasi objek. Banyak orang berfikir bahwa pemrograman berorientasi objek identik dengan bahasa java. Memang bahasa java merupakan bahasa yang paling konsisten dalam mengimplementasikan paradigma pemrograman berorientasi objek. Namun sebenarnya bahasa pemrograman yang mendukung pemrograman berorientasi objek tidak hanya bahasa java [9].

2.7. UML (*Unified Modeling Language*)

Pada perkembangan teknik pemrograman berorientasi objek, muncul sebuah standarisasi bahasa pemodelan untuk pembangunan perangkat lunak yang dibangun dengan menggunakan teknik pemrograman berorientasi objek, yaitu *Unified Modeling Language* (UML). UML muncul karena adanya kebutuhan pemodelan visual untuk menspesifikasikan, menggambarkan, membangun, dan dokumentasi dari sistem perangkat lunak. UML merupakan bahasa visual untuk pemodelan dan komunikasi mengenai sebuah sistem dengan menggunakan diagram dan teks-teks pendukung.

UML hanya berfungsi untuk melakukan pemodelan. Jadi penggunaan UML tidak terbatas pada metodologi tertentu, meskipun pada kenyataanya UML paling banyak digunakan pada metodologi berorientasi objek. Seperti banyak kita ketahui bahwa banyak hal di dunia informasi yang tidak dapat dibakukan, semua tergantung kebutuhan, lingkungan, dan konteksnya.

Begitu juga dengan perkembangan penggunaan UML bergantung pada level abstraksi penggunanya. Jadi, belum tentu pandangan yang berbeda dalam penggunaan UML adalah suatu yang salah, tapi perlu ditelaah dimanakah UML digunakan dan hal apa yang ingin divisualkan. Secara analogi jika dengan bahasa yang kita gunakan sehari-hari, belum tentu penyampaian bahasa dengan puisi adalah hal yang salah. Sistem informasi bukanlah ilmu pasti, maka jika ada banyak perbedaan dan interpretasi di dalam bidang sistem informasi merupakan hal yang sangat wajar [9].

2.7.1. *Use Case* Diagram

Secara grafik menggambarkan interaksi antara sistem, hal-hal diluar sistem,dan user. Dengan kata lain, mereka mengambarkan siapa yang akan memakai sistem dan dengan cara itu seorang user akan berinteraksi dengan sistem. *Use-case* merupakan urutan langkah yang secara tindakan saling terkait (Skenario), baik termotivasi maupun secara manual, untuk tujuan melengkapi satu tugas bisnis tunggal.

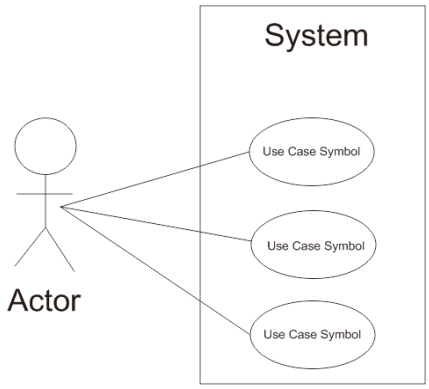
Notasi yang digunakan dalam *use-case* diagram antara lain :

1.*Actor*

Adalah posisi yang dimiliki user terhadap sistem. Yang menjadi *actor* dapat berupa manusia, *hardware* atau sistem lain yang berhubungan dengan sistem yng berjalan. Atau *actor* adalah sesuatu yang memerlukan interaksi dengan sistem untuk bertukar informasi.

2.*Use – case* simbol

Menggambarkan sekumpulan urutan, dimana setiap urutan mewakili intervensi antara *actor* dengan sistem. *Use-case symbol* mempresentasikan fungsionalitas sistem secara keseluruhan.



Gambar 2.8 Use Case Diagram

Deskripsi *Use-case*

Bagian umum kejadian *use-case* adalah deskripsi *use-case,* yaitu langkah demi langkah mulai dengan pelaku menginisialisasi use-case dan melanjutkan hingga akhir kejadian bisnis. Dimana dalam deskripsi *use-case* terdapat :

a. Nama *use-case* (*use-case*), merupakan nama dari suatu *use-case*.

b. Aktor (*actor*), pelaku yang berpartisipasi didalam suatu *use-case.*

c. Tujuan (*goal*), tujuan dari *use-case*.

d. Prakondisi (*precondition*), batasan pada keadaan sistem sebelum use-case dapat dieksekusi.

e. Ringkasan (*summary*), deskripsi ringkasan pendek yang berisi jumlah kalimat yang menunjukan secara garis besar tujuan *use-case* dan berbagai kegiatanya.

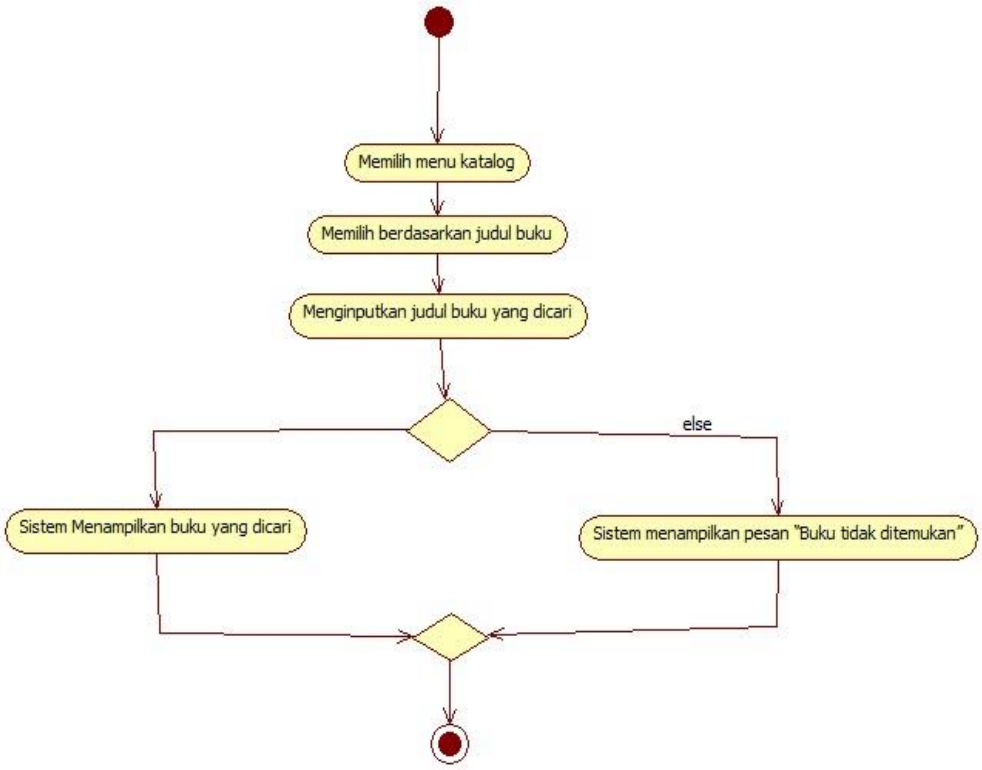
f. *Related use-case*, *use-case* yang saling berhubungan.

g. Langkah – langkah (*steps*), menjelaskan setiap langkah dari *use-case* dengan mengunakan dua kolom, dimana kolom sebelah kiri menunjukan aksi dari pelaku dan kolom sebelah kanan merupakan respon yang diberikan oleh sistem.

2.7.2. *Activity* Diagram

Activity diagram atau diagram aktivitas adalah sebuah diagram yang dapat digunakan untuk menggambarkan secara grafis aliran proses bisnis, langkah langkah sebuah *use-case* atau logika *behavior object*. Diagram ini serupa dengan *flowchart*, dimana secara grafis diagram ini menggambarkan aliran sekuential dengan *flowchart* dimana diagram ini menggambarkan mekanisme kegiatan paralel.

Oleh karena itu, diagram ini sangat berguna untuk memodelkan kegiatan yang akan dilakukan saat sebuah operasi dieksekusi dan untuk memodelkan hasil – hasil dari kegiatan ini, seperti memodelkan *event* yang menyebabkan *window* akan ditampilkan atau ditutup.



Gambar 2.9 *Activity* Diagram

Dibawah ini merupakan simbol-simbol yang ada pada diagram aktivitas :

a. *Initial node*

Lingkaran yang diisi penuh merupakan awal mulainya dari diagram ini. *Initial node* tidak harus ada, tetapi dengan mengunakanya membuat diagramnya lebih mudah untuk dibaca.

b. *Activity*

Segi empat bersudut tumpul menggambarkan kegiatan yang perlu dilakukan. Secara *activity* bisa secara fisik, seperti inspect *Forms*, atau secara elektronik, seperti *Display Create Student Screen*.

c. *Control Flow*

Panah yang ada di diagram, menggambarkan sasaran yang mengawali kegiatan.

d. *Fork*

Bar hitam dengan satu *flow* yang ke dalamnya dan beberapa *flow* lainnya meninggalkannya, merupakan bar sinkronissi dimana kegiatan dapat dilakukan secara paralel.

e. *Join*

Bar hitam dengan satu *flow* mengarah ke dalamnya dan ada satu *flow* yang meninggalkannya, merupakan bar sinkronisasi dimana beberapa aktivitas yang mempunyai tujuan yang sama dari keduannya digabung menjadi satu.

f. *Decicion*

Gambar sebuah wajik, menggambarkan sebuah kegiatan keputusan.

g. *Final Node*

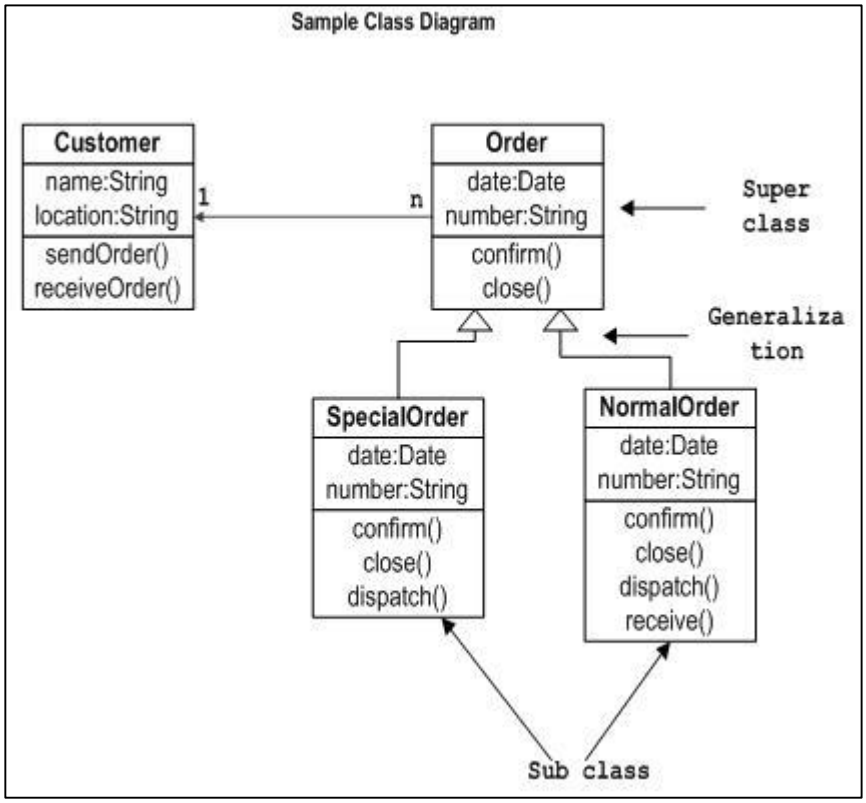
Gambar lingkaran yang diisi penuh dan ada batas lagi di luarnya, menggambarkan akhir dari sebuah proses.

2.7.3. *Class* Diagram

Class diagram merupakan salah satu diagram struktur statis yang menggambarkan struktur hubungan antar kelas. *Class* disgram digunakan untuk mensimulasikan objek-objek dalam dunia nyata ke dalam sistem yang akan dibangun. Notasi UML pada class diagram adalah sebuah persegi yang dibagi menjadi 3 area, yaitu nama kelas, atribut, dan operasi (*method*).

Dalam sistem yang akan dibangun, kelas-kelas akan dihubungkan secara konseptual. Hubungan ini disebut asosiasi.

Class diagram dapat juga menggambarkan keanekaragaman (*multiplicity*), yaitu jumlah objek dari suatu kelas yang berhubungan dengan sebuah objek dari kelas yang berasosiasi.



Gambar 2.10 Contoh *class* diagram

2.7.4. *Sequence* Diagram

Sequance diagram digunakan terutama untuk menunjukan interaksi antar objek dalam urutan sekuensial. *Sequance* diagram sangat berguna untuk mengkomunikasikan bagaimana objek-objek berinteraksi dalam suatu proses bisnis. Analisis sistem umumnya mengunakan *sequance* diagram untuk memperjelas *use case*.

Sequance diagram terdiri dari objek-objek yang digambarkan dengan sebuah persegi yang memiliki nama. Objek-objek tersebut diletakan diatas garis hidup tersebut, ada kotak kecil memanjang yang dinamakan aktivasi. Aktivasi mempresentasikan eksekusi dari operasi yang objek lakukan.

Suatu objek dapat bertukar pesan dengan objek lain. Pesan tersebut digambarkan sebagai sebuah panah dari garis hidup suatu objek ke objek lain.

Ada empat jenis pesan yang bisa digunakan, yakni pesan sederhana (*simple message*), pesan sinkron (*synchronous message*), pesan asinkron (*asynchronous*

message), dan pesan nilai pengembalian (*retrun value*). Pesan sederhana digunakan untuk memindahkan kontrol dari satu objek ke objek lain. Pesan sinkron digunakan apabila suatu objek harus menunggu jawaban dari objek lain (yang dipanggil) terhadap pesan tersebut sebelum melanjutkan proses lainnya. Pesan asinkron digunakan apabila suatu objek tidak perlu menunggu jawaban dari objek lain sebelum melanjutkan proses lainnya. Sequance diagram merepresentasikan waktu dan arah vertikal, dari atas ke bawah. Pesan yang lebih atas menandakan bahwa pesan tersebut terjadi terlebih dahulu.