BAB 2
LANDASAN TEORI

1.1. Permainan

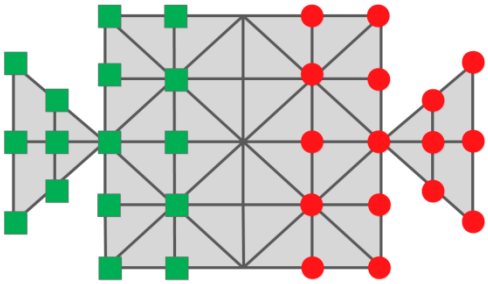
Istilah permainan dari kata dasar main. Menurut buku Kamus Besar Bahasa Indonesia, edisi kedua, terbitan Departemen Pendidikan dan Kebudayaan Balai Pustaka arti kata main adalah melakukan permainan untuk menyenangkan hati atau melakukan perbuatan untuk bersenang-senang baik menggunakan alat-alat tertentu 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 [**10**].

1.2. Damdaman

Permainan damdaman 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:

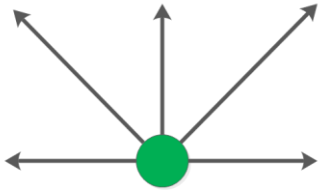
1. Pemain harus berjumlah 2 orang
2. Pemain harus mencari batu/kerikil untuk bidaknya tetapi setiap pemain tidak boleh sama
3. Jumlah bidak untuk setiap pemain ada 16 buah
4. Gambar papan permainan dam daman, seperti pada gambar 2.1
5. Susun setiap batu/kerikil tadi pada papan permainan seperti pada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



Gambar 2. 1 Bentuk papan permainan damdaman

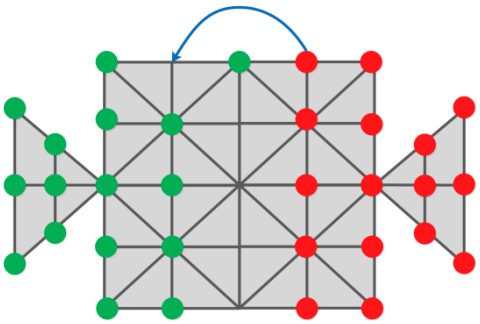
1.2.1. Langkah bidak

Langkah bidak pada permainan Damdaman terdiri dari dua langkah yang berbeda, yaitu langkah bidak biasa dan langkah bidak raja/satria. 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.



Gambar 2. 2 Langkah valid sebuah bidak

Sedangkan untuk memakan bidak lawan, ialah jika terdapat koordinat kosong dibelakang bidak lawan. Seperti terlihat pada gambar 2.3.



Gambar 2. 3 Langkah memakan bidak lawan

Jika kesempatan untuk memakan bidak lawan tersebut tidak diambil, atau pemain justru mengambil langkah lain, maka pemain lawan dapat mengambil 3 bidak pemain secara bebas/acak. Aturan ini disebut sebagai ‘Dam’.

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

1.3. Kecerdasan Buatan

Kecerdasan Buatan (*Artificial Intelligence*) merupakan sebuah studi tentang bagaimana membuat komputer melakukan hal-hal yang pada saat ini dapat

dilakukan oleh manusia [**11**]. Dengan kata lain komputer dapat memiliki kecerdasan layaknya manusia dalam menjalankan tugasnya.

Terdapat tiga tujuan dari kecerdasan buatan, yaitu membuat komputer menjadi cerdas, mengerti kecerdasan, dan membuat mesin lebih berguna. Yang dimaksud dengan kecerdasan adalah kemampuan belajar dan mengerti dari pengalaman, dapat menanggapi situasi yang baru, dan menggunakan penalaran dalam pengambilan keputusan dan penyelesaian masalah.

Teknik pemecahan masalah pada kecerdasan buatan terbagi kedalam empat bagian, yaitu :

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 *Blind/un-informed* dan *informed/heuristic* seperti terlihat pada tabel 2.1.

Tabel 2. 1 Contoh *Blind/Un-Informed* dan *Informed/Heuristic search*

Blind/Un-informed Search	*a. Informed Search/Heuristic Search*
1. *Breadth-First Search (BFS)* 2. *Depth-First Search (DFS)* 3. *Depth-Limited Search (DLS)*	1. *Generate-and-Test* (pembangkitan dan pengujian) 2. *Hill Climbing (Pendakian Bukit)* 3. *Simulated Annealing (SA)* 4. *Best-First Search*

2. Reasoning

Teknik *reasoning* atau penalaran merupakan teknik penyelesaian masalah dengan cara mempresentasikan masalah kedalam basis pengetahuan (*knowledge base*) menggunakan logika 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 [**12**].

1.4. Algoritma *Monte Carlo Tree Search*

Algoritma *Monte Carlo Tree Search* merupakan algoritma pencarian *heuristik Best First Search* yang umum digunakan sebagai metode pemilihan langkah optimal. Metode ini menggunakan pencarian dalam bentuk pohon pencarian dan juga menggunakan simulasi kemungkinan yang dipilih secara acak dengan metode *Monte Carlo* [**13**]. MCTS sudah diterapkan dan memberi hasil yang menjanjikan pada berbagai permainan papan 1 lawan 1, diantaranya *Go, Othelo, connect4* dan *Checkers.*

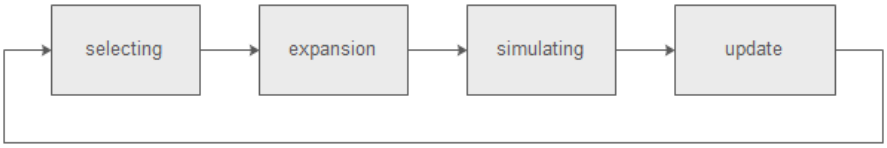
Didasarkan pada keberhasilannya pada berbagai permainan papan tersebut, pada penelitian ini algoritma *Monte Carlo Tree search* digunakan untuk mencari langkah optimal pada catur, dengan perbandingan algoritma *Minimax* dan *Alpha*

beta. Setelah proses pengujian dapat disimpulkan bahwa algoritma MCTS masih belum optimal dalam permainan catur dibandingkan algoritma lain [**6**].

Proses dalam pengambilan keputusan oleh algoritma MCTS yang memiliki empat tahap yaitu:

1. *Selection,* algoritma akan membuat pohon pencarian berdasarkan langkah yang valid dalam permainan. Dari *node* pohon pencarian yang ada sementara, Selanjutnya akan dipilih node mana yang merupakan langkah terbaik dengan metode *Best-First Search*.
2. *Expanding the Tree*, Langkah ini diambil ketika menemui keadaan/*state* yang belum diketahui pada pohon pencarian/jika simulasi belum menemukan nilai node level selanjutnya, Maka akan dibuat *node* baru dari *state* tersebut. *Node-node* baru tersebut adalah kemungkinan langkah bidak yang dapat digunakan.
3. *Simulation*, melakukan simulasi langkah permainan secara acak dan sesuai dengan aturan permainan untuk mencari langkah optimal. Setiap langkah akan diberi aturan bobot agar proses ini akan cenderung kepada langkah yang terbaik.
4. *Update*, setelah sampai pada akhir dari simulasi permainan, proses ini akan memperbaharui nilai dan jumlah kunjungan dari setiap langkah yang dilewati kepada *root*. Kembali ke langkah satu jika *state* akhir pemainan belum tercapai [**14**]

Akan dilakukan perulangan langkah hingga proses pemilihan langkah dengan algoritma UCT mencapai *state* akhir dari permainan. Berikut bagan alir dari algoritma MCTS :



Gambar 2. 4 Bagan alir algoritma MCTS

1.5. Algoritma *Upper Confidence Bound to Tree (UCT)*

Merupakan algoritma varian dari Monte Carlo Tree Search, pada algoritma ini MCTS digabungkan dengan algoritma *Upper Confidence Bound* (UCB) pada metode penyeleksian node pohon pencariannya. UCB merupakan salah satu algoritma yang digunakan untuk menyelesaikan *multi-armed bandit problem*. Yaitu dimana suatu permasalahan memiliki banyak pilihan solusi,dan tiap solusi yang diambil memiliki *reward* yang berbeda. Maka algoritma ini tujuannya ialah mendapatkan *reward* semaksimal mungkin dari setiap langkah/pemilihan *bandit arm* yang diambil [**15**].

Pada Algoritma UCT, setiap node yang telah diketahui akan disimpan *cost*-nya, lalu algoritma akan membuat sampel acak namun berdasar pada aturan permainan, jika jumlah sampel sedikit, maka kemungkinan rata-rata pada sampel akan kurang akurat. Karena itu UCT akan melakukan pemilihan *node* secara optimistik, UCT akan memilih langkah yang memiliki nilai *confidence* tertinggi (*Upper Confidence Bound*) yaitu nilai kemenangan tertinggi dari sampel yang diketahui [**8**]. Pada proses pemilihan ini akan digunakan algoritma *Upper Confidence Bound to Tree* sebagai aturan pemilihan. Rumusnya ialah sebagai berikut:

$$\frac{Q(v')}{N(v')} + \sqrt{\frac{\ln\ (N\ (v))}{5\ \times N(v')}} = \ UCT \qquad (2.1)$$

Dengan keterangan notasi sebagai berikut:

Q = nilai kemenangan

N = jumlah visit

v = node parent

v’ = node anak

Q(v’) didapatkan dari evaluasi sebuah *node*, jika *node* menguntungkan berdasar aturan permainan maka *node* akan diberi nilai 1, nilai akan bertambah 1 jika anak dari *node* tersebut juga merupakan langkah menguntungkan, sehingga semakin tinggi nilai ini, maka akan semakin sering *node* ini diprioritaskan dalalm pencarian. Nilai *N(v’)* menunjukan berapa kali node tersebut pernah dilewati untuk dievaluasi oleh algoritma UCT, sedangkan *N(v)* adalah jumlah visit dari

parent *node*-nya. Kedua nilai tersebut akan bertambah jika suatu *child node* dilewati/dievaluasi.

Algoritma UCT akan mencari nilai UCT tertinggi sebagai solusi pencariannya. Dengan mencari nilai UCT tertinggi, dari *node* anak, maka didapat *node* yang akan dijadikan langkah optimal untuk pemain komputer.

1.6. Pemrograman 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.

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 [**16**].

1.7. HTML

Hypertext Markup Language merupakan format teks dimana halaman *web* tertulis. *Hypertext* ialah metode pengorganisasian dimana sumber data dan saling dihubungkan oleh *link/hyperlink* yang dapat digunakan pengguna untuk dapat berpindah dari suatu informasi ke informasi lainnya. *Hypertext* sangat cocok digunakan pada *Web* karena pengguna tidak perlu mengetahui alamat setiap informasi yang dibutuhkan, yang pengguna butuhkan ialah bagaimana mengaktifkan *Hypertext* tersebut. *Markup Language* ialah bahasa yang merupakan konten dan struktur dari dokumen dengan identifikasi, *tagging*, pada setiap elemen berbeda pada dokumen, misalnya *captions*, *heading* dan lainnya. Bagian-bagian tersebut dapat diber *tag* sehingga menjadi elemen-elemen yang berbeda menggunakan *Markup Language*. Kedua deskrpsi tersebut menjadikan HTML merupakan bahasa yang mendukung *hypertext* dan *tagging* pada elemen yang berbeda pada sebuah halaman *web* [**17**].

1.8. Javascript

JavaScript adalah bahasa skrip yang populer di internet dan dapat bekerja di

sebagian besar penjelajah *web* populer seperti *Internet Explorer* (IE), Mozilla Firefox, Netscape dan Opera. Kode *JavaScript* dapat disisipkan dalam halaman *web* menggunakan *tag* '<script>'.

JavaScript pertama kali dikembangkan oleh *Brendan Eich* dari *Netscape* di bawah nama *Mocha*, yang nantinya namanya diganti menjadi *LiveScript*, dan akhirnya menjadi *JavaScript*.

Navigator sebelumnya telah mendukung *Java* untuk lebih bisa dimanfaatkan para *programmer* yang non-Java. Maka dikembangkanlah bahasa pemrograman bernama *LiveScript* untuk mengakomodasi hal tersebut. Bahasa pemrograman inilah yang akhirnya berkembang dan diberi nama *JavaScript*, walaupun tidak ada hubungan bahasa antara *Java* dengan *JavaScript*.

JavaScript bisa digunakan untuk banyak tujuan, misalnya untuk membuat efek *rollover* baik di gambar maupun teks, dan yang penting juga adalah untuk membuat AJAX. *JavaScript* adalah bahasa yang digunakan untuk AJAX.

Kelebihan bahasa pemrograman Javascript diantaranya adalah:

1. Ukuran *file* kecil

Script dari *Javascript* memiliki ukuran yang kecil sehingga ketika *web* yang memiliki *Javascript* ditampilkan di *browser* maka akses tampilannya akan lebih cepat dibandingkan ketika *browser* membuka suatu *web* yang memiliki *script Java*. Hal ini juga sangat berkepentingan dengan daya kerja *server*. Semakin kecil *space* suatu *web* yang disimpan dalam suatu *server* maka daya kerja *server* ketika digunakan oleh user di *internet* akan tidak terlalu berat, selain itu sifat *Javascript client side* yang tidak perlu lagi di olah oleh *server* ketika *browser* memanggil *web* dari sebuah *server*.

2. Mudah dipelajari

Javascript merupakan bahasa semi pemograman yang merupakan gabungan antara bahasa pemograman *Java* dengan bahasa kode HTML sehingga disebut bahasa *hybrid*. Walaupun javascript merupakan turunan dari *Java* namun *Javascript* tidak memiliki aturan yang serumit *Java*.

3. Terbuka (*Open Source*)

Javascript tidak terikat oleh *hardware* maupun *software* tertentu bahkan sistem operasi seperti *Windows* maupun UNIX. Karena ia bersifat terbuka, maka ia dapat dibuat maupun di baca di semua jenis komputer.

Sedangkan beberapa kekurangannya adalah:

1. Script tidak terenkripsi

Karena *Javascript* bersifat *client side*, maka *script* yang kita buat di *text editor* dan telah dijadikan *web* di *server*, ketika *user merequest web* dari *server* tersebut maka sintak *Javascript* akan langsung ditampilkan di *browser*. *User* bisa melihat dan menirunya dari *source*-nya.

2. Kemampuan terbatas

Walaupun *Javascript* mampu membuat bentuk *web* menjadi interaktif dan dinamis, namun *Javascript* tidak mampu membuat program aplikasi sendiri seperti *Java*.

3. Keterbatasan Objek

Javascript tidak mampu membuat kelas-kelas yang bisa menampung objek-objek tambahan seperti *Java* karena *Javascript* teleh memiliki objek yang *built-in* pada sturktur bahasanya [**18**].

1.9. Text Editor

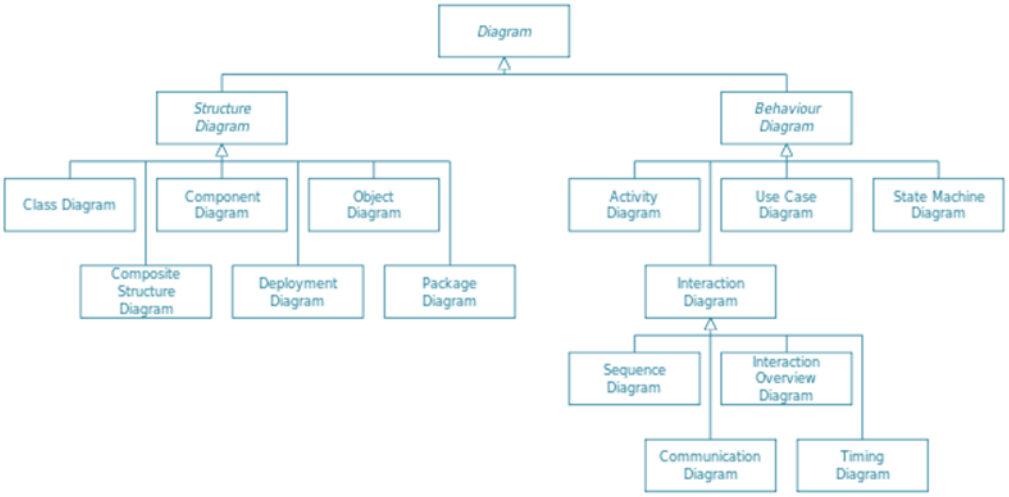
Text Editor adalah jenis program yang digunakan untuk mengolah *file* teks biasa. *Text Editor* sering disediakan dengan sistem operasi atau paket pengembangan perangkat lunak, dan dapat digunakan untuk mengubah file konfigurasi dan kode sumber bahasa pemrograman.

Text Editor, umumnya menggunakan teks tanpa format, selain untuk memudahkan pembacaan, juga karena teks ini sering digunakan sebagai kode sumber sebuah program. Halaman *web* misalnya, halaman tersebut kenyataanya merupakan teks tanpa format yang diberi tag HTML sehingga dapat dibaca oleh pengguna sebagai halaman yang berisi elemen-elemen non-teks seperti gambar atau tabel.

1.10. Unified Model Language

Unified modeling language (UML) adalah bahasa pemodelan standar untuk pembangunan perangkat lunak ataupun sistem yang dibangun dengan pendekatan *object-oriented.* Dalam merancang sistem, model memiliki peranan penting yaitu untuk mengelola kompleksitas. Pemodelan membantu untuk fokus menghubungkan aspek penting dari perancangan sistem. Sebuah model adalah sebuah abstraksi dari benda nyata. Model yang dibangun adalah penyederhanaan dari sistem yang dibangun. Memberikan maksud dan kelangsungan hidup sistem untuk dipahami, dievaluasi, dikritisasi bahkan lebih cepat dibandingkan harus langsung ke sistem sebenarnya.

UML itu sendiri sudah mengalami beberapa generasi, dimulai dari UML 1.0, 1.5, dan yang terakhir 2. Versi pertama dari UML memberikan komunikasi rancangan secara jelas, menyampaikan esensi dari suat perancangan, bahkan menangkap peta kebutuhan fungsional dari solusi perangkat lunak. Bagaimanapun, dunia berubah secara fundamental dengan pengenalan pemodelan sistem, dibanding hanya pemodelan perangkat lunak. Dari perubahan itulah yang membuat versi UML itu sendiri berkembang hingga kini ialah versi 2. Secara keseluruhan ada 13 diagram yang dapat dimanfaatkan dari UML versi 2 ini.



Gambar 2. 5 Jenis diagram pada pemodelan UML

Pada penelitian ini akan diterapkan 4 diagram UML, yaitu *use case diagram, activity diagram, class diagram* dan *sequence diagram* untuk menjelaskan struktur aplikasi dan *behaviour* aplikasi [**19**]. Berikut penjelasan dari tiap-tiap diagram yag digunakan:

1.10.1. Use Case Diagram

Use case biasa digunakan untuk memodelkan suatu fungsi yang disediakan oleh sistem, didalam use case juga memiliki batasan didalam sistem dengan diberi suatu label nama *use case* tersebut.

Use case juga memiliki karakteristik antara lain :

a. *Use case* merupakan interaksi antara sistem dengan *actor*, yaitu tindakan yang akan dilakukan sistem.
b. *Use case* itu terdapat *actor* yang dimana juga bisa melibatkan *actor* lain didalam sistem.
c. *Use case* mendefinisikan tindakan yang akan dilakukan didalam sistem.
d. *Use case class* tentunya memiliki objek *use case* yang dikenal dengan skenario.

Simbol-simbol yang ada di dalam *use case diagram*, antara lain:

1. *Actor*

Merupakan seseorang atau pelaku yang akan menggunakan suatu sistem. Sebuah actor memberikan inputan informasi ke sistem, menerima informasi, dan sebaliknya. *Actor* hanya berinteraksi dengan use case. *Actor* biasanya disimbolkan dengan *Stick Man*.

2. *Use Case*

Merupakan fungsionalitas dari suatu sistem, sehingga user mengerti kegunaan dari sistem tersebut di kembangkan atau dibangun nantinya seperi apa. cara menentukan use case dalam suatu sistem adalah :

a. lihat pola perilaku dari perangkat lunak aplikasi
b. lihat gambaran dari tugas sebuah *actor*
c. lihat apa yang akan dikerjakan oleh suatu perangkat lunak, bukan melihat bagaimana cara mengerjakannya.

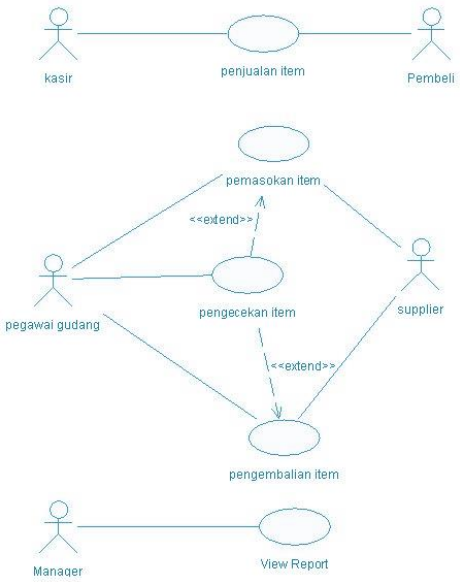
3. Relasi

Merupakan hubungan antara actor dengan *use case* didalam sistem. Ada beberapa relasi yang terdapat didalam *use case diagram*:

a. *Association,* berfungsi menghubungkan antar elemen.
b. *Generalization,* merupakan inheritance, dimana suatu elemen dapat juga merupakan spesialisasi dari elemen lainnya, contoh : staf akuntan dan staf kreatif digeneralisasi menjadi member staf.
c. *Dependency,* merupakan elemen yang ketergantungan dengan elemen lainnya
d. *Aggregation,* merupakan tipe gabungan yang menunjukkan bahwa elemen berisi atau terdiri dari elemen-elemen lainnya. contoh : mobil terdiri dari ban, setir, dan lain-lain

Berikut ini akan dijelaskan tipe-tipe relasi yang umum yang ada didalam *use case diagram* :

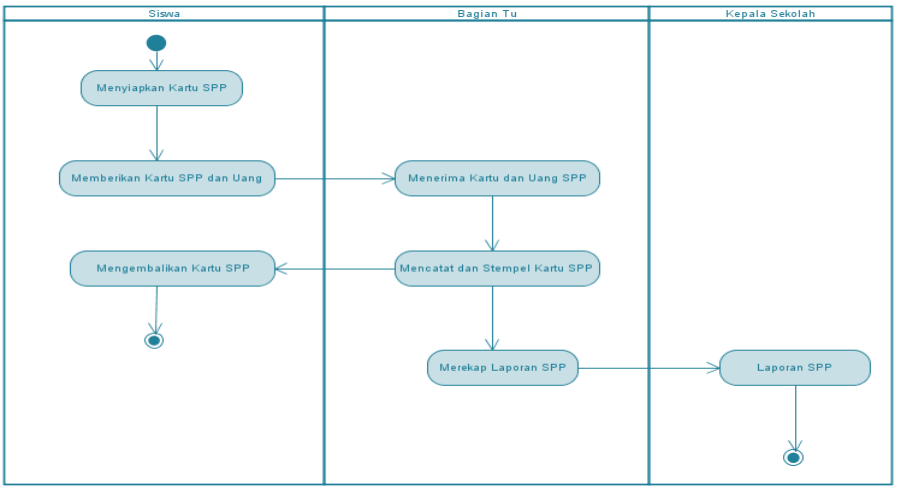
A. *Include*, merupakan aktivitas yang harus dipenuhi untuk terjadi sebuah *event*, dimana kondisi ini sebuah *use case* adalah bagian dari *use case* lainnya.
B. *Extends*, merupakan aktivitas yang hanya berjalan di bawah kondisi tertentu.



Gambar 2. 6 Contoh diagram *Use Case*

1.10.2. Activity Diagram

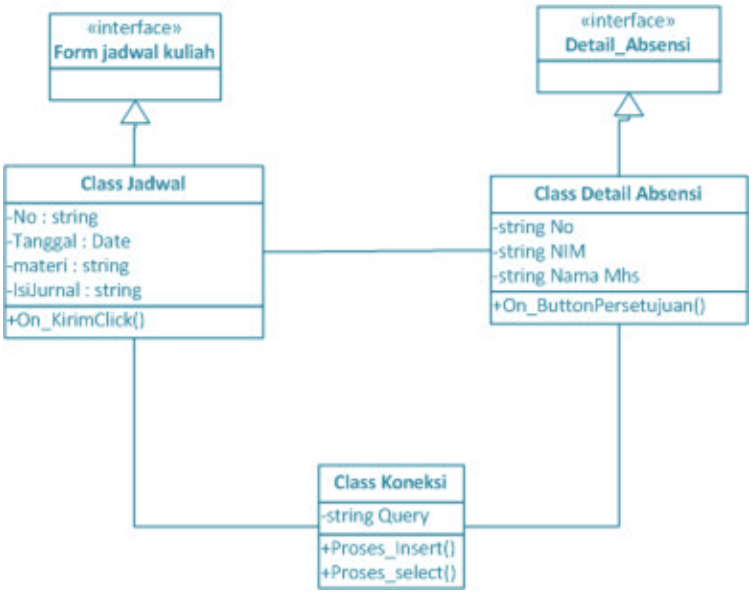
Diagram ini memperlihatkan aliaran dari suatu aktifitas ke aktifitas lainnya dalam suatu sistem. Diagram ini terutama penting dalam pemodelan fungsi-fungsi dalam suatu sistem dan memberi tekanan pada aliran kendali antar objek.



Gambar 2. 7 Contoh *Activity diagram*

1.10.3. Class Diagram

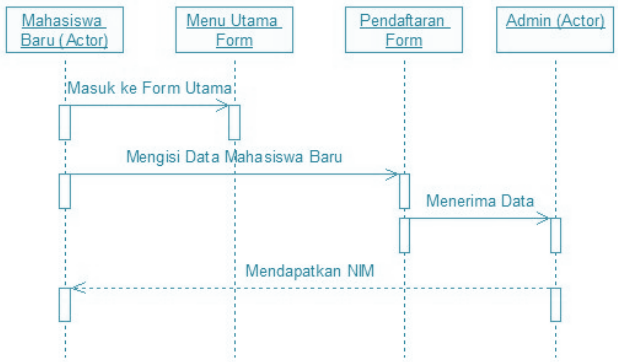
Diagram ini memperlihatkan himpunan kelas-kelas, antarmuka-antarmuka, kolaborasi-kolaborasi dan relasi-relasi antar objek.



Gambar 2. 8 Contoh class diagram

1.10.4. Sequence Diagram

Diagram ini memperlihatkan interaksi yang menekankan pada pengiriman pesan *(message)* dalam suatu waktu tertentu. Biasanya yang ditampilkan ialah objek kelas, pesan, nama kelas.



Gambar 2. 9 Contoh *sequence diagram*

1.11. Pengujian *Black box*

Pengujian black box merupakan pendekatan komplementer dari teknik white box, karena pengujian black box diharapkan mampu mengungkap kelas kesalahan yang lebih luas dibandingkan teknik white box. Pengujian black box berfokus pada pengujian persyaratan fungsional perangkat lunak, untuk mendapatkan serangkaian kondisi input yang sesuai dengan persyaratan fungsional suatu program.

Pengujian black box adalah pengujian aspek fundamental sistem tanpa memperhatikan struktur logika internal perangkat lunak. Metode ini digunakan untuk mengetahui apakah perangkat lunak berfungsi dengan benar. Pengujian black box merupakan metode perancangan data uji yang didasarkan pada spesifikasi perangkat lunak. Data uji dibangkitkan, dieksekusi pada perangkat lunak dan kemudian keluaran dari perangkat lunak dicek apakah telah sesuai dengan yang diharapkan. Pengujian black box berusaha menemukan kesalahan dalam kategori [**9**]:

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 terminasi.