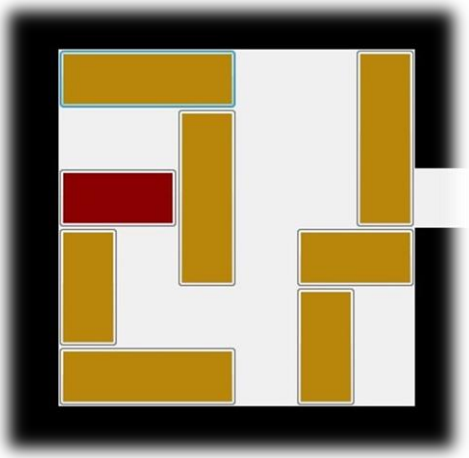
BAB 2
LANDASAN TEORI

2.1 Game Unblockme

Game Unblockme, Game ini sangat sederhana dan simple karena pemain hanya perlu memindahkan box/balok kayu kelubang atau ke tempatnya. Namun dalam permainannya dibutuhkan suatu pemikiran dan strategi karena banyak terdapat penghalang dan aturan dalam permainan. Sehingga pemain harus benar-benar memikirkan langkah yang jitu untuk menyelesaikan permainan . Game Unblokme merupakan salah satu game yang sistematis dengan ruang lingkup masalah yang jelas dimana dalam penyelesaian game ini pemain harus fokus untuk mencapai kondisi akhir (goal). Sehingga dengan fokus dalam pencarian kemungkinan langkah yang tepat, game ini dapat diselesaikan. Papan dan board pada game unblockme bisa dilihat pada gambar berikut :



Gambar 2.1 *Game Unblockme*

2.1.1 Aturan Permainan Unblockme

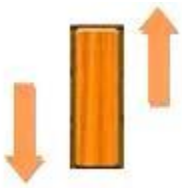
Aturan permainan secara umum adalah membuat balok yang berwarna merah keluar dari papan Permainan. Pemain hanya perlu menggulirkan balok-balok lain agar tidak menghalangi jalan balock merah keluar dari board. Balok terdiri atas empat macam, balok vertikal kecil, vertikal besar, horizontal kecil dan horizontal besar.

1. Jika balok tersebut berbentuk horizontal maka hanya bisa digeser ke kanan atau ke kiri. Pergerakan balok tersebut bisa dilihat pada gambar berikut :



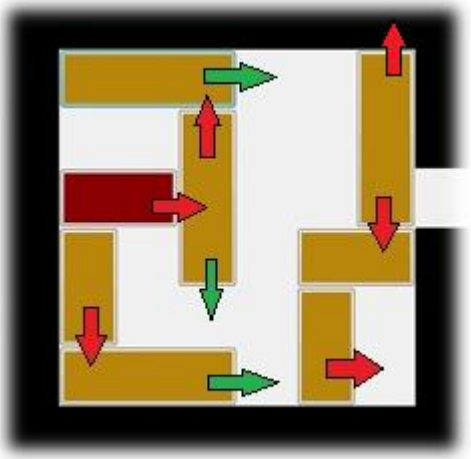
Gambar 2.2 *Balok bergerak Horizontal.*

2. Jika balok tersebut berbetuk vertikal maka hanya bisa digeser ke atas atau kebawah. Pergerakan balok tersebut bisa dilihat pada gambar berikut :



Gambar 2.3 *Balok Bergerak Secara Vertikal.*

3. Aturan-aturan pada pergerakan atau pergeseran balok pada kotak bisa di lihat pada gambar berikut :



Gambar 2.8 Contoh gerakan valid dalam panah hijau,invalid dalam panah merah.

Pada gambar 2.8 aturan pergerakan balok di jelaskan dengan tanda panah berwarna hijau dan merah. Tanda panah berwarna hijau berarti valid dalam artian bisa di gerakan atau arah pergeraknnya benar dan tanda panah yang berwarna merah berarti invalid dalam artian balok yang ditandai dengan panah berwarna merah tidak bisa degerakan dengan alasan balok tersebut terhalang dinding atau balok lain.

2.2 Algoritma

Algoritma adalah salah satu cabang ilmu komputer yang membahas prosedur penyelesaian suatu permasalahan. Algoritma adalah urutan langkah-langkah dalam menentukan suatu masalah. Algoritma juga dapat didefinisikan dengan deretan langkah komputasi yang mentransformasikan masukan (*input*) menjadi keluaran (*output*). Dapat disimpulkan algoritma adalah serangkaian urutan langkah-langkah atau prosedur untuk menyelesaikan suatu masalah dengan memproses nilai keluaran. Algoritma dapat ditulis dalam bentuk *pseudocode* ataupun *flowchart. Pseudocode* berasal dari kata *pseudo* yang artinya mirip atau menyerupai dan *code* berarti kode,

sehingga *pseudocode* adalah kode yang menyerupai kode program sebenarnya. Sedangkan *flowchart* merupakan bentuk gambar atau diagram yang mempunyai aliran satu atau dua arah secara skuensial [4].

2.2.1 Definisi Algoritma

Algoritma adalah urutan langkah-langkah logis penyelesaian yang disusun secara matematis dan logis. Langkah-langkah dalam algoritma harus logis dan harus dapat ditentukan bernilai salah atau benar. Dalam beberapa konteks, algoritma adalah spesifikasi urutan langkah untuk melakukan pekerjaan tertentu. Pertimbangan dalam pemilihan algoritma adalah[7] :

1. Algoritma haruslah benar, artinya algoritma akan memberikan keluaran yang dikehendaki dari sejumlah masukan yang diberikan. Tidak peduli sebagus apapun algoritma, apabila menghasilkan keluaran yang bernilai salah maka algoritma tersebut bukanlah algoritma yang baik.
2. Seberapa baik algoritma yang diterapkan, artinya algoritma yang baik harus mampu memberikan hasil yang sedekat mungkin dengan nilai sebenarnya. Dalam hal ini, hal terpenting pada algoritma untuk menyelesaikan masalah yang memerlukan aproksimasi hasil (hasil yang berupa pendekatan).
3. Efisiensi algoritma, artinya efisiensi algoritma dapat ditinjau dari dua hal yaitu efisiensi waktu dan memori.

2.2.2 Sejarah Algoritma

Kata algoritma berasal dari latinisasi nama seorang ahli matematika dari Uzbekistan Al Khawārizmi. Dahulu, orang hanya mengenal kata *algorism* yang berarti proses menghitung dengan angka arab. Para ahli bahasa berusaha menemukan asal kata *algorism,* dan akhirnya para ahli sejarah matematika berhasil menemukan asal kata yang berasal dari penulis buku arab yang terkenal yaitu Abu Ja'far Muhammad Ibnu Musa Al Khawārizmi. Al Khawārizmi dibaca orang barat menjadi *algorism.* Al Khawārizmi menulis kitab *Al Jabar Wal-Muqabala* yang artinya "Buku Pemograman Dan Pengurangan" (*The Book of Restoration And Reducation*). Perubahan kata dari *algorism*

menjadi kata *algoritm* muncul dikarenakan kata *algorism* sering dikelirukan dengan *arithmetic,* sehingga akhiran -*sm* berubah menjadi –*thm.* Seiring perkembangan, lambat laun kata *algorithm* berangsur-angsur dipakai sebagai metode perhitungan (komputasi) secara umum sehingga kehilangan makna kata aslinya. Dalam bahasa Indonesia, kata *algorithm* diserap menjadi algoritma .

2.2.3 Algoritma Backtracking

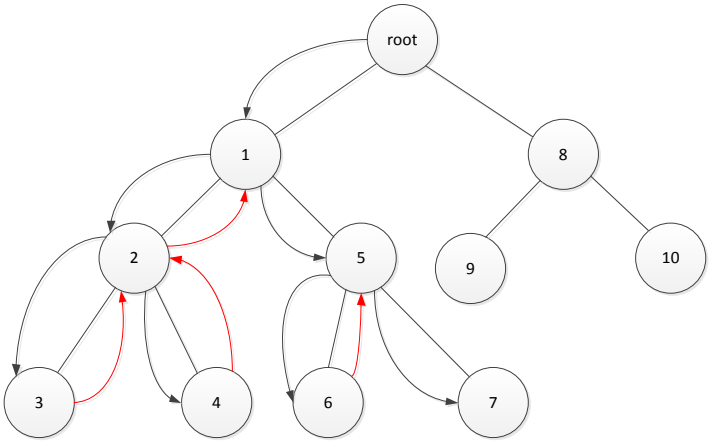
Algoritma *backtracking* pertama kali diperkenalkan oleh D.H. Lehmer pada tahun 1950. Dalam perkembangannya, beberapa ahli seperti RJ Walker, Golomb, dan Baumert menyajikan uraian umum tentang algoritma *backtracking* dan penerapannya dalam berbagai persoalan dan aplikasi. Algoritma *backtracking* merupakan salah satu metode pemecahan masalah yang termasuk dalam strategi yang berbasis pencarian pada ruang status [8].

Algoritma *backtracking* bekerja secara rekursif dan melakukan pencarian solusi persoalan secara sistematis pada semua kemungkinan solusi yang ada. Algoritma ini berbasis *Depth-First Search* (DFS) dalam pencarian solusi dengan melakukan penelusuran suatu struktur berbentuk pohon berakar secara *preorder.* Proses ini dicirikan dengan ekspansi simpul terdalam lebih dahulu sampai tidak ditemukan lagi suksesor dari suatu simpul.

Mekanisme penyelesaian dengan menggunakan algoritma *backtracking* berprinsip pada metode rekursif. Untuk menyelesaikan keseluruhan masalah, dibutuhkan sebuah solusi untuk permasalahan pertama kemudian permasalahan-permasalahan lainnya akan dicoba untuk diselesaikan secara rekursif berdasarkan solusi pertama. Apabila pada kemungkinan solusi yang dicari tidak ditemukan, maka dilakukan *backtrack* untuk menguji kemungkinan solusi selanjutnya. Proses *backtrack* akan selesai ketika tidak ada lagi solusi yang mungkin untuk diselesaikan pada permasalahan awal.

2.2.4 DFS (*Depth-First Seacrh)*

DFS (Depth-First-Search) adalah salah satu algoritma penelusuran struktur graf / pohon berdasarkan kedalaman. Simpul ditelusuri dari root kemudian ke salah satu simpul anaknya (misalnya prioritas penelusuran berdasarkan anak pertama [simpul sebelah kiri]), maka penelusuran dilakukan terus melalui simpul anak pertama dari simpul anak pertama level sebelumnya hingga mencapai level terdalam. Setelah sampai di level terdalam, penelusuran akan kembali ke 1 level sebelumnya untuk menelusuri simpul anak kedua pada pohon biner [simpul sebelah kanan] lalu kembali ke langkah sebelumnya dengan menelusuri simpul anak pertama lagi sampai level terdalam dan seterusnya. berikut ini adalah gambar simulasi pencarian jalur simpul dengan DFS gambar berikut :



Gambar 2.2 *Simulasi jalur pencarian simpul dengan Depth First Search (DFS)*

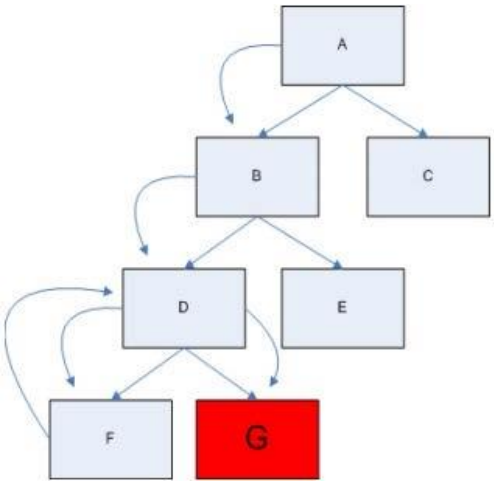
Pada gambar 2.2 ,dapat dilihat simulasi pencarian simpul 7 di dalam pohon biner menggunakan metode pencarian DFS. Pencarian dilakukan terhadap bagian kiri pohon terlebih dahulu. Ketika sampai di simpul 3, *backtracking* (tanda panah merah) kesimpul 2 untuk melanjutkan pencarian di bagian kanan (simpul 4). Ketika ternyata simpul 4 bukan simpul yang dicari, *backtracking* dilakukan kembali hingga simpul 2 dan 1,kemudian dilanjutkan pencarian di simpul bagian kanan, yaitu simpul 5. Ketika simpul yang dicari berhasil ditermukan di simpul 7, pencarian pun dihentikan.

2.2.5 Prinsip Backtracking

Algoritma *backtracking* pada pencarian kemungkinan solusi dibuat dalam bentuk pohon ruang status dan algoritma akan menelusuri pohon tersebut secara *Depth-First Seacrh* (DFS) sampai ditemukan solusi yang layak. Langkah-langkah pencarian solusi pada algoritma *backtracking* adalah sebagai berikut [8]:

1. Solusi dicari dengan membentuk lintasan dari akar ke daun. Aturan yang dipakai adalah mengikuti metode pencarian *Depth-First Search* (DFS). Simpul yang telah dilahirkan dinamakan simpul hidup dan simpul hidup yang diperluas dinamakan simpul-E (*expand node*).
2. Jika lintasan yang diperoleh dari perluasan simpul-E tidak mengarah ke solusi, maka simpul itu akan menjadi simpul mati yang tidak dapat diperluas lagi.
3. Jika posisi terakhir ada disimpul mati, maka pencarian dilakukan dengan membangkitkan simpul anak yang lainnya. Apabila tidak ada simpul anak, maka dilakukan *backtracking* ke simpul induk.
4. Pencarian dihentikan jika telah menemukan solusi atau tidak ada simpul hidup yang dapat ditemukan.

Pada tahapan-tahapan yang terdapat diatas dapat diberikan contoh melalui gambar 2.4 sebagai berikut:

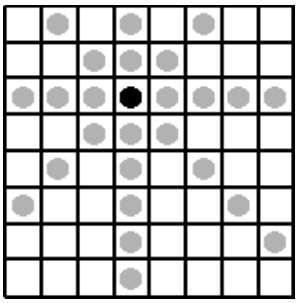


Gambar 2.4 *Contoh pohon ruang status algoritma backtracking*

2.2.6 Kegunaan Backtrack

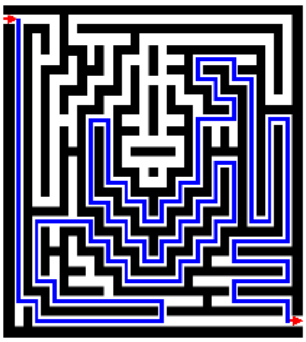
Penggunaan terbesar *backtrack* adalah untuk membuat kecerdasan buatan pada *board games.* Dengan algoritma ini, program dapat menghasilkan pohon sampai dengan kedalaman tertentu dari *current status* dan memilih solusi yang akan membuat langkah-langkah *user* dalam melakukan tindakan untuk mengahsilkan pohon solusi baru dengan jumlah pilihan langka terbanyak.

Beberapa kegunaan yang cukup terkenal dari algoritma *backtracking* pada suatu masalah statik adalah pemecahan masalah *N-Queen problem* dan *maze solver. N-Queen problem* adalah bagaimana cara meletakan bidak *queen* catur sebanyak N buah pada papan catur atau pada papan ukuran NxN yang dapat memangsa bidak lainnya dengan satu gerakan. Meskipun terdapat kemungkinan lebih dari satu solusi, tetapi pencarian semua solusi biasanya tidak terlalu diperlukan.



Gambar 2.8 Contoh *N-Queen problem*.

Pada *maze solver* cara kerjanya adalah bagaimana mencari jalan keluar dari suatu *maze* (labirin). Pada *maze* yang sederhana dimana *field* yang dibentuk dapat dipresentasikan dalam bentuk *biner* dan pada setiap petak maksimal terdapat 4 kemungkinan yaitu atas, kanan, bawah, dan kiri. Untuk masalah ini biasanya solusi pertama yang ditemukan bukanlah solusi yang paling optimal, sehingga untuk mendapatkan hasil yang optimal dibutuhkan pencarian terhadap seluruh kemungkinan solusi yang ada [4].



Gambar 2.9 Contoh *maze solver*

2.2.7 Kecerdasan Buatan

Kecerdasan buatan merupakan sub-bidang ilmu komputer (*computer science*) yang khusus ditujukan untuk membuat perangkat lunak dan perangkat keras yang sepenuhnya bisa menirukan beberapa fungsi otak manusia, atau cabang ilmu komputer yang mempelajari otomatisasi tingkah laku cerdas (*intelligent*). Kecerdasan buatan didasarkan pada teori suara (*sound theoretical*) dan prinsip-prinsip aplikasi dari bidangnya. Prinsip-prinsip ini meliputi struktur data yang digunakan dalam representasi pengetahuan algoritma yang diperlukan untuk mengaplikasikan pengetahuan serta bahasa dan teknik pemograman yang digunakan dalam mengimplementasikannya.

Beberapa para ahli, mendefinisikan kecerdasan buatan (*artificial intelligence*) secara berbeda-beda tergantung pada sudut pandangnya masing-masing. Para ahli ada yang mendefinisikan kecerdasan buatan (*artificial intelligence*) fokus kepada logika berpikir manusia saja, ada juga yang mendefinisikan kecerdasan buatan (*artificial intelligence*) secara lebih luas terhadap tingkah laku manusia. Stuart Russel dan Peter Norvig mengelompokan definisi kecerdasan buatan (*artificial intelligence*) yang diperoleh dari beberapa *textbook* berbeda kedalam empat kategori, yaitu [9]:

1. *Thinking humanly : The cognitive modeling approach*

   Pendekatan *thinking humanly* dilakukan dengan dua cara sebagai berikut :

   a. Melalui intropeksi dimana proses ini adalah menangkap pemikiran-pemikiran yang didapat pada saat berfikir.
   b. Melalui eksperimen-eksperimen psikologi.

2. *Acting humanly : The turing test approach*

   Pada tahun 1950, Alan Turing membuat sebuah eksperimen program penginterogasian untuk menguji kemampuan komputer pada manusia melalui *teletype* (komunikasi berbasis teks jarak jauh). Dalam hal ini, apabila *integrator* tidak dapat membedakan yang diinterogasi adalah manusia atau komputer maka program tersebut lolos dari *turing test. Turing test* sengaja menghindari interaksi fisik antara *integrator* dan komputer, karena simulasi fisik manusia tidak memerlukan intelejensia.

3. *Thinking rationally : The laws of thought approach*

   Terdapat dau masalah pada pendekatan ini, yaitu :

   a. Tidak mudahnya untuk membuat pengetahuan informal dan menyatakan pengetahuan tersebut kedalam *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 rationally 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. Apabila mencapai tujuan, maka *agent* dapat melakukan aksi berdasarkan kesimpulan tersebut.

Thinking humanly dan *acting humanly* adalah dua definisi dalam arti yang sangat luas. Hingga saat ini, pemikiran manusia yang diluar rasio, yakni refleks dan intuitif (perasaan), belum dapat ditirukan oleh komputer. Sedangkan definisi *thinking rationally* jauh lebih sempit dari *acting rationally* dengan pendekatan *rational agent.* Hal ini berdasarkan pemikiran bahwa komputer dapat melakukan penlaran secara logis dan juga dapat melakukan aksi secara rasional berdasarkan hasil penalaran. [6].

2.3 *Unified Modeling Language* (UML)

Unified Modeling Language merupakan salah satu alat bantu yang dapat digunakan dalam bahasa pemograman yang berorientasi objek, saat ini UML akan mulai menjadi standar masa depan bagi industri pengembangan sistem/perangkat lunak yang berorientasi objek sebab pada dasarnya UML digunakan oleh banyak perusahaan raksasa seperti IBM, Microsoft, dan sebagainya. Berikut ini beberapa definisi UML [10]:

Unified Modeling Language merupakan metode pengembangan perangkat lunak (sistem informasi) dengan menggunakan metode grafis serta merupakan bahasa untuk visualisasi, spesifikasi, konstruksi serta dokumentasi.

1. *Unified Modeling Language* (UML) adalah bahasa yang telah menjadi standar untuk visualisasi, menetapkan, membangun dan mendokumentasikan arti suatu sistem perangkat lunak.

2. Unified Modeling Language (UML) dapat didefinisikan sebagai sebuah bahasa yang telah menjadi standar dalam industri untuk visualisasi, merancang dan mendokumentasikan sistem perangkat lunak.

3. Unified Modeling Language (UML) merupakan standard modeling language yang terdiri dari kumpulan-kumpulan diagram, dikembangkan untuk membantu para pengembang system dan software agar bisa menyelesaikan tugas-tugas seperti spesifikasi, visualisasi, desain arsitektur, konstruksi, simulasi dan testing, serta dokumentasi.

Berdasarkan beberapa pendapat yang dikemukakan diatas dapat ditarik kesimpulan bahwa *Unified Modeling Language* (UML) adalah sebuah bahasa yang berdasarkan grafik atau gambar untuk memvisualisasikan, menspesifikasikan, membangun, dan pendokumentasian dari sebuah sistem pengembangan perangkat lunak berbasis Object Oriented Programming (OOP).



Gambar 2.9 Logo *Unified Modeling Language* (UML).

2.3.1 UML (Diagram *Unified Modeling Language*)

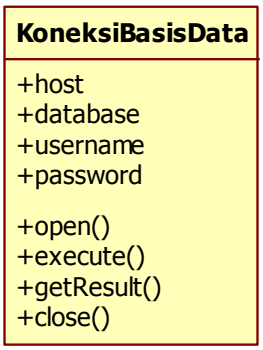
Beberapa literatur menyebutkan bahwa UML menyediakan sembilan jenis diagram, yang lain menyebutkan delapan karena ada beberapa diagram yang digabung, misalnya diagram komunikasi, diagram urutan dan diagram pewaktuan digabung menjadi diagram interaksi. Namun demikian model-model itu dapat dikelompokkan berdasarkan sifatnya yaitu statis atau dinamis. Jenis diagram itu antara lain [9].

1. Diagram Kelas (*Class Diagram*)

   Diagram kelas menggambarkan struktur sistem dari segi pendefinisian kelas-kelas yang akan dibuat untuk membangun sistem. Kelas memiliki apa yang disebut atribut dan metode atau operasi.

   a. Atribut merupakan variabel-variabel yang dimiliki oleh suatu kelas.
   b. Operasi atau metode adalah fungsi-fungsi yang dimiliki oleh suatu kelas.

Diagram kelas dibuat agar pembuat program atau programmer membuat kelas-kelas sesuai rancangan di dalam diagram kelas agar antara dokumentasi perancangan dan perangkat lunak menjadi sinkron. Berikut adalah contoh dari diagram kelas.

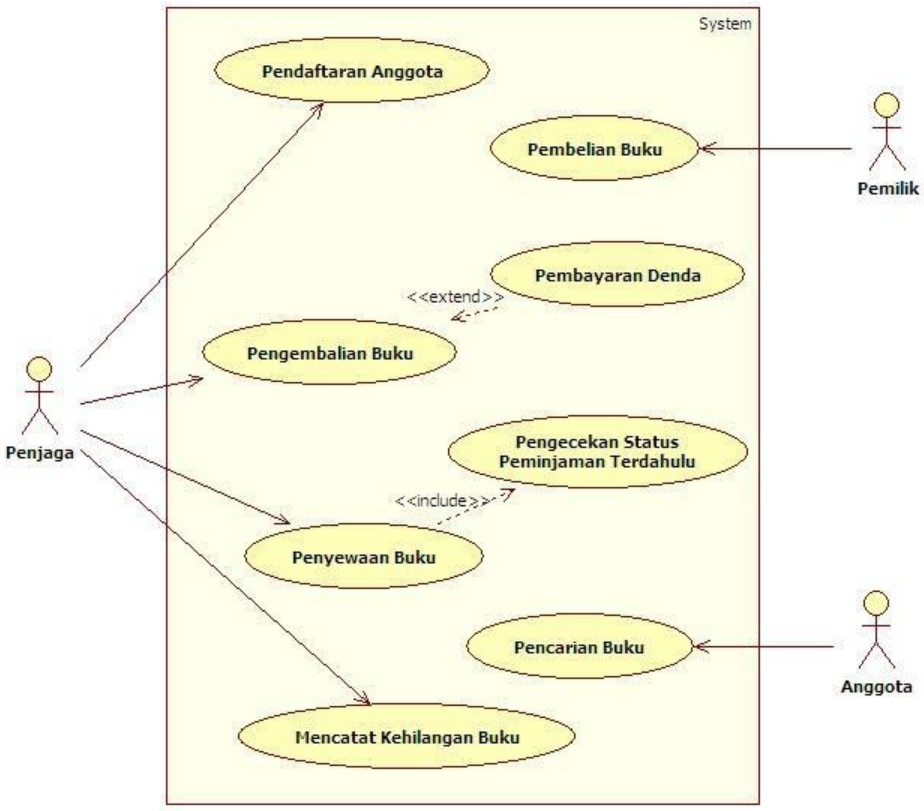


Gambar 2.7 *contoh class diagram*

2. Diagram *Use Case*

Use case mendeskripsikan sebuah interaksi antara satu atau lebih aktor dengan sistem informasi yang akan dibuat. Secara kasar, *use case* digunakan untuk mengetahui fungsi apa saja yang ada di dalam sebuah sistem informasi dan siapa saja yang berhak menggunakan fungsi-fungsi itu. 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.*

a. Aktor merupakan orang, proses, atau sistem lain yang berinteraksi dengan sistem informasi yang akan dibuat diluar sistem informasi yang akan dibuat itu sendiri, jadi walaupun simbol dari aktor adalah gambar orang, tapi aktor belum tentu merupakan orang.
b. *Use case* merupakan fungsionalitas yang disediakan sistem sebagai unit-unit yang saling bertukar pesan antar unit atau aktor.



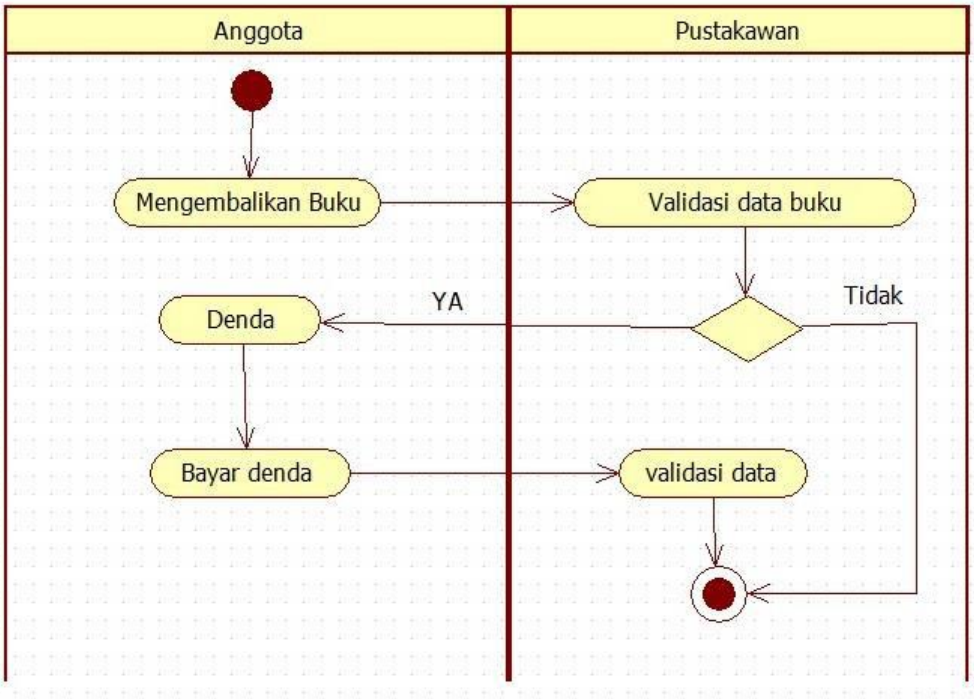
Gambar 2.8 *Contoh Use Case Diagram*

3. Diagram Aktivitas (*Activity Diagram*)

   Diagram aktivitas menggambarkan *workflow* (aliran kerja) atau aktivitas dari sebuah sistem atau proses bisnis atau menu yang ada pada perangkat lunak. Yang perlu diperhatikan disini 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 :

   a. Rancangan proses bisnis dimana setiap urutan aktivitas yang digambarkan merupakan proses bisnis sistem yang didefinisikan.
   b. Urutan atau pengelompokkan tampilan dari *system* / *user interface* dimana setiap aktivitas dianggap memiliki sebuah rancangan antarmuka tampilan.

c. Rancangan pengujian dimana setiap aktivitas dianggap memerlukan sebuah pengujian yang perlu didefinisikan kasus ujinya.
d. Rancangan menu yang ditampilkan pada perangkat lunak.

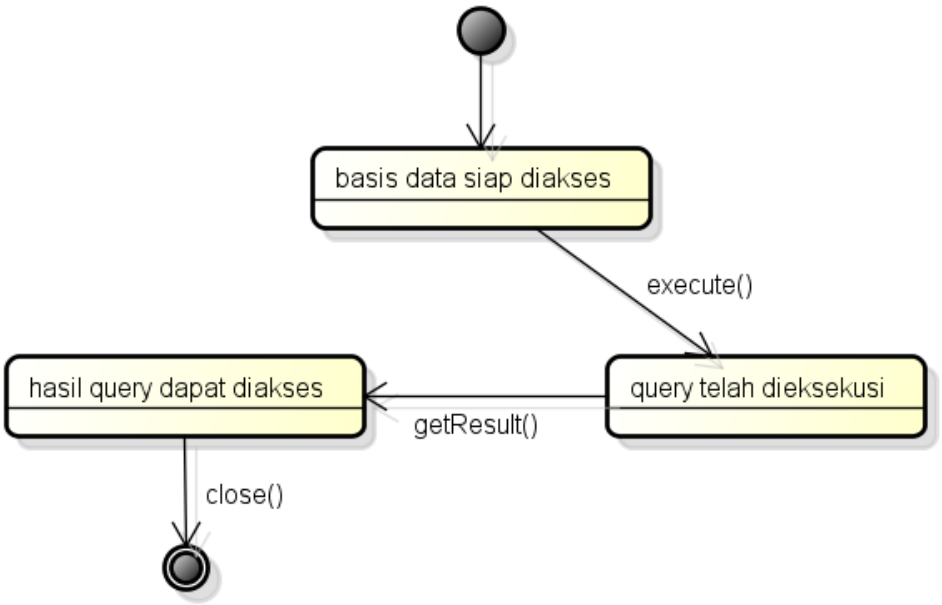


Gambar 2.9 *Contoh activity diagram*

4. Diagram State Machine (Statechart Diagram)

State machine diagram atau statechart diagram atau dalam bahasa Indonesia disebut diagram mesin status atau sering juga disebut diagram status digunakan untuk menggambarkan perubahan status atau transisi status dari sebuah mesin atau sistem atau objek. Jika diagram sekuen digunakan untuk interaksi antar objek maka diagram status digunakan untuk interaksi di dalam sebuah objek. Perubahan tersebut digambarkan dalam suatu graf berarah.

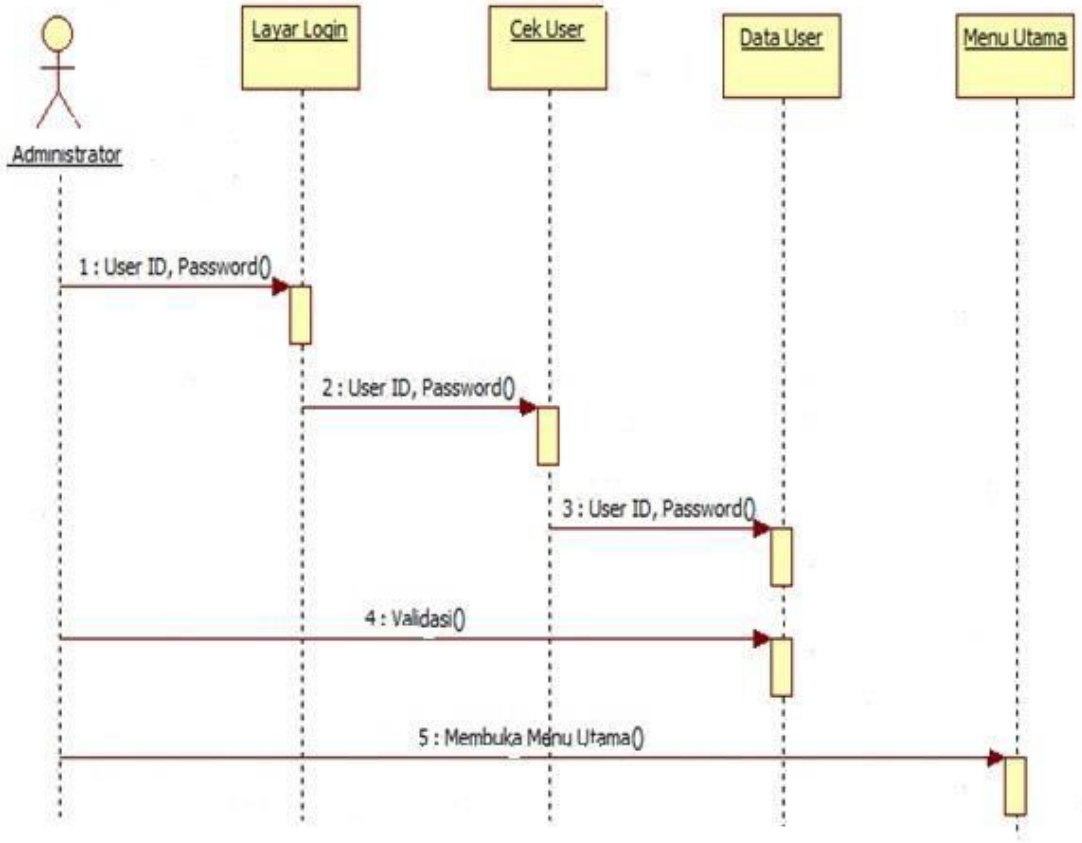
Berikut ini adalah contoh dari *statechart diagram* :



Gambar 2.10 *Contoh statechart diagram*

5. Diagram Sekuen (*Squence Diagram*)

   Diagram sekuen menggambarkan kelakuan pada objek *use case* dengan mendeskripsikan waktu hidup objek dan *message* yang dikirimkan dan diterima antar objek. Oleh karena itu untuk menggambar diagram sekuen maka harus diketahui objek-objek yang terlibat dalam sebuah *use case* beserta metode-metode yang dimiliki kelas yang diinstansiasi menjadi objek itu. Membuat diagram sekuen juga dibutuhkan untuk melihat skenario yang ada pada *use case*. Banyaknya diagram sekuen yang harus digambar adalah minimal sebanyak pendefinisian *use case* yang memiliki proses 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 2.11 *Contoh squence diagram*

2.4 Pembangunan Perangkat Lunak

Pada tahapan pembangunan perangkat lunak ini, tols atau aplikasi yang digunakan untuk mengimplementasikan algoritma *backtracking* pada game Unblockme diperkenalkan secara singkat.

2.4.1 Bahasa Pemrograman C#

C# (dibaca: C sharp) merupakan sebuah bahasa pemrograman yang berorientasi objek yang dikembangkan oleh Microsoft sebagai bagian dari inisiatif kerangka .NET Framework. Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti Java, Delphi, Visual Basic, dan lain-lain) dengan beberapa penyederhanaan. Menurut standar ECMA-334 C# *Language Specification*, nama C# terdiri atas sebuah huruf Latin C (U+0043) yang diikuti oleh tanda pagar yang menandakan angka # (U+0023). Tanda pagar # yang digunakan memang bukan tanda kres dalam seni musik (U+266F), dan tanda pagar # (U+0023) tersebut digunakan karena karakter kres dalam seni musik tidak terdapat di dalam keyboard standar.



Gambar 2.10 *Logo Extensi Bahasa C# pada Visual Studio.*

2.4.2 C# Sintaks

Bahasa C# sangat mirip dengan bahasa C++ dan Java. Kode pada C# terdiri dari kumpulan statement, dimana antar statementnya dipisahkan dengan titik koma. Anda dapat menambahkan lebih dari satu statement pada satu baris program C# merupakan block-structured language, yang berarti setiap statement merupakan bagian dari blok kode. Setiap blok dibatasi dengan kurung kurawal

buka dan kurung kurawal tutup ({ dan }) yang dapat berisi beberapa statement, contohnya:

```
{
<code line 1, statement 1>;
<code line 2, statement 2>
<code line 3, statement 2>;
                                    }
```

dapat melihat statement kedua dan statement ketiga adalah satu baris program karena statement kedua tidak diakhiri dengan titik koma. Pada C# dalam sebuah blok dapat terdapat blok dagi didalamnya atau dapat juga disebut sebagai *nested-blocks.*

```
{
<code line 1>;
      {
            <code line 2>;
            <code line 3>;
      }
<code line 4>;
                          }
```

Komentar adalah baris program yang tidak akan dieksekusi oleh compiler, gunanya untuk menambahkan keterangan pada program atau menonaktifkan sementara baris program tertentu untuk kepentingan debugging, baris komentar pada C# dapat ditulis sebagi berikut:

```
/* Ini adalah komentar */
/* Dan juga...
... ini komentar */
// ini juga komentar dalam satu baris
```

Contoh penulisan program dalam bahasa C# adalah sebagai berkut :

```
using System;
namespace HelloCSharp
      {
            class Program
            {
                  static void Main(string[] args)
                  {
                  //menampilkan output ke layar
                  Console.WriteLine("Ini adalah contoh program
C#");
            }
      }
}
```

2.4.3 Microsoft Visual Studio Express

Microsoft Visual Studio Express adalah kumpulan *freeware* lingkungan pengembangan terpadu (IDE) yang dikembangkan oleh Microsoft yang merupakan versi ringan dari Microsoft Visual Studio. Gagasan edisi ekspres menurut Microsoft adalah untuk menyediakan efisiensi, mudah digunakan dan mudah dipelajari bagi pengguna IDE selain pengembang perangkat lunak profesional, yaitu seperti penggemar dan mahasiswa.



Gambar 2.11 *logo Microsoft Visual Studio Express*