BAB II

TINJAUAN PUSTAKA

2.1 Sejarah dan Perkembangan Rubik [3]

Rubik's Cube atau kubus Rubik adalah sebuah teka-teki (*puzzle*) mekanis dalam bentuk tiga dimensi, yang ditemukan pada tahun 1974 oleh seorang pemahat dan profesor arsitektur dari Hungaria bernama Erno Rubik. Pada pertengahan tahun 70-an, Erno Rubik ingin membuat sebuah model sebagai alat pembelajaran untuk membantu murid - muridnya memahami geometri tiga dimensi dan akhirnya tercipta menjadi sebuah mainan teka - teki yang paling laris didunia.

Rubik memberi nama hasil temuannya itu "*Magic Cube*" – Kubus Ajaib, yang kemudian mendapatkan hak paten Hungaria dan dijual pertama kali melalui perusahaan Ideal *Toy Corporation*. Pada tahun 1980, perusahaan Ideal *Toy* mengubah nama kubus ajaib tersebut menjadi "*Rubik's Cube*". Dan hingga saat ini, lebih dari 300 juta *Rubik's Cube* telah dijual di seluruh dunia.

Sebuah *Rubik's Cube* standar memiliki panjang sisi yang sama ukurannya yaitu 5.7 cm. Terbentuk dari 26 potongan kecil yang disebut juga "*cubelets*" atau "*cubies*". Ada 6 "*cubies*" yang disatukan dalam sebuah kerangka yang mempunyai 6 sumbu (*axis*). Dan pada kerangka inilah, sisa 20 "*cubies*" dipasang sedemikian rupa sehingga terbentuk sebuah kubus Rubik.

Rubik masuk ke Indonesia sekitar tahun 90-an dan mulai berkembang pesat ketika masuk keabad millennium yaitu tahun 2000 ke atas. Pertengahan tahun 2009, masuknya permainan kubik dengan nama *speedcubing*. *Speedcubing* adalah menyelesaikan Kubus Rubik dalam waktu secepat mungkin dengan dihitung waktunya. Dari situlah munculah komunitas - komunitas pencinta *speedrubik* muncul, salah satunya Nusantara *Speedcubing Association*. Visi mereka adalah memopulerkan *speedcubing* agar sejajar dengan olahraga yang lain seperti catur, billiard, renang, dan sebagainya. Melalui misi mengadakan kompetisi dan kegiatan - kegiatan *speedcubing* baik yang bertaraf lokal maupun internasional.

Nusantara *Speedcubing Association* merupakan wadah penggemar permainan *puzzle* pada umumnya dan Kubus Rubik (*Rubik's Cube*) pada khususnya. Berawal dari komunitas kecil dari beberapa orang yg memiliki minat dan ketertarikan yang sama terhadap beberapa jenis *puzzle* terutama *Rubik's Cube*. Komunitas rubik ini awalnya terbentuk ketika ada beberapa remaja penyuka rubik berkumpul untuk membentuk suatu wadah atau komunitas. Nusantara *Speedcubing Association* diresmikan ketika ada pertemuan Indonesia *Open Cube* 2009 sekitar 31 Agustus 2009 di Jakarta. Nusantara *Speedcubing Association* sendiri mempunyai anggota sendiri dibeberapa kota. Setiap kota sendiri menamakannya komunitas ini, sebagai contoh di Bandung yaitu Nusantara *Speedcubing Association* regional Bandung. Selain di Bandung, Nusantara *Speedcubing Association* juga terbentuk di kota - kota seperti Jakarta, Solo, Yogyakarta, Surabaya, dan Medan .

Nusantara *Speedcubing Association* regional Bandung memiliki anggota sekitar 12 orang. Biasanya mereka berkumpul atau diistilah mereka disebut *gathering* di *Foodcourt Bandung Elektronik Center*. Pelaksanaan *gathering* mereka setiap sebulan sekali, biasanya mereka berkumpul untuk membicarakan cara menyelesaikan rubik - rubik yang sangat sulit. Kadang-kadang mereka bertukar pengalaman bagaimana cara menyelesaikan permainan rubik. Selain itu mereka membicarakan kompetisi - kompetisi rubik yang akan diadakan di Indonesia. Biasanya kompetisi - kompetisi rubik di Indonesia dilaksanakan di kota Jakarta, Solo, dan Bandung. Akan tetapi dalam turnamen tersebut, mereka pernah menjuarainya.

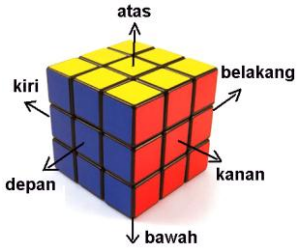
2.1.1. Mengenal Rubik [4]

Terminologi yang biasa dipakai untuk menjelaskan suatu algoritma *Rubik's Cube* sebagai berikut:

1. Rubik's Cube terdiri atas enam buah sisi: depan (*front*), belakang (*back*), kiri (*left*), kanan (*right*), atas (*up*), dan bawah (*down*) seperti pada gambar 2.1 yang menjelaskan sisi-sisi pada rubik.
2. Gerakan memutar sisi Rubik's Cube searah jarum jam disimbolkan dengan huruf pertama dari nama sisi tersebut dalam bahasa Inggris. Untuk yang berlawanan arah dengan jarum jam, ditambahkan tanda petik ('). Contohnya, F berarti memutar sisi depan (***f****ront*) searah jarum jam,

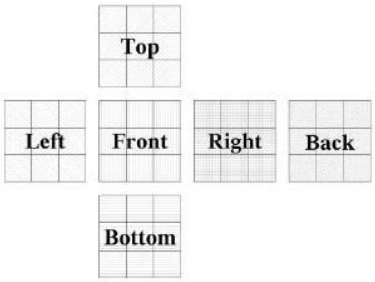
sedangkan L' berarti memutar sisi kiri (***l**eft*) berlawanan arah dengan jarum jam.

3. Gerakan memutar seluruh kubus tidak memiliki notasi yang tetap, sehingga laporan tugas akhir yang berjudul "**Implementasi Algoritma *Layer By Layer* Untuk Menyelesaikan Permainan *Rubiks Cube***" menggunakan "*RotateRight*", "*RotateLeft*", "*RotateUp*", dan "*RotateDown*". Nama gerakan tersebut sudah menjelaskan maknanya.



Gambar 2.1. Penamaan sisi-sisi pada Rubik's Cube

Untuk menampilkan *Rubik's Cube*, digunakan tampilan dalam bentuk dua dimensi, yaitu pada gambar 2.2 sebagai berikut:



Gambar 2.2. Tampilan dua dimensi Rubik's Cube

2.1.2. Algoritma Dalam Permainan Rubik [5]

Sebelum mempelajari pada pengenalan notasi atau algoritma rubik, salah satu hal yang perlu diperhatikan dalam memahami karakter rubik 3x3x3 adalah pembagian kategori *piece* atau kubus kecil. Pembagian peace ini terdiri dari :

1. ***Center Piece :*** *Piece* (kubus kecil) yang terdapat ditengah disetiap sisi rubik. *Center piece* berjumlah 6 buah sesuai dengan warna yang terdapat disetiap sisi.



Gambar 2.3. *Center Piece* Rubik

2. ***Corner Piece :*** *Piece* yang terdapat pada setiap sudut disemua sisi rubik, tentu jumlah seluruhnya 8 buah dimana sebuah *corner piece* terdiri dari 3 warna stiker.



Gambar 2.4. *Corner Piece* Rubik

3. ***Edge Piece :*** *Piece* yang mengelilingi *center piece* (bukan *corner piece*), sehingga apabila dilihat dari atas akan membentuk *cross*. *Piece* ini berjumlah 12 buah dimana sebuah *edge piece* terdiri dari 2 warna stiker.



Gambar 2.5. *Edge Piece* Rubik

Dalam menyelesaikan rubik, ada yang disebut dengan notasi. Notasi ini mengindikasikan bagian dari *cube* yang harus diputar. Beberapa notasi yang digabungkan menjadi satu set gerakan akan memberi efek tertentu pada rubik. Gabungan notasi inilah yang disebut "algoritma". Efek yang diberikan oleh algoritma adalah efek yang dibutuhkan, sehingga sangat dibutuhkan untuk menyelesaikan *cube*. Cara memutar cube ditulis dalam notasi khusus menggunakan huruf F, B, U, D, L, R, dan seterusnya. Notasi tersebut digunakan secara internasional oleh para cuber di seluruh dunia. Agar lebih jelas, perhatikan penjelasan dibawah ini [6]:

1. *Single Turn* (Putaran Tunggal)

Notasi *single turn* adalah notasi yang digunakan untuk memutar sebuah sisi (*layer*) dari rubik sebesar seperempat putaran (90 derajat). Notasi ini memiliki beberapa macam yang ditunjukkan dengan huruf, antara lain :

a. **F** (*front* atau sisi depan)

b. **B** (*back* atau sisi belakang)

c. **U** (*up* atau sisi atas)

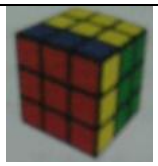
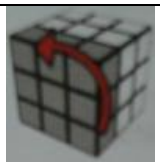
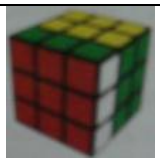
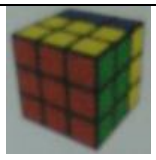
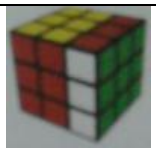
d. **D** (*down* atau sisi bawah)

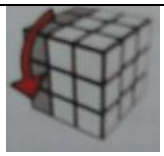
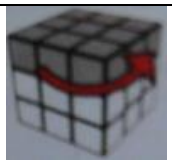
e. **R** (*right* atau sisi kanan)

f. **L** (*left* atau sisi kiri)

Implementasi gerakan notasi huruf pada rubik dapat dilihat pada tabel 2.1 sebagai berikut :

Tabel 2.1. *Single Turn* Rubik (Putaran Tunggal Rubik)

Notasi	**Gerakan**	**Hasil**
F		
F’		
B		
B’		
R		

R’		
L		
L’		
U		
U’		
D		
D’		

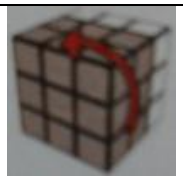
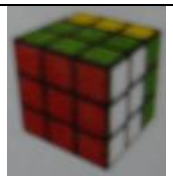
Secara umum notasi - notasi di atas dibagi menjadi dua kelompok. Kelompok pertama yaitu: F, B, R, L, U, D yang gerakannya adalah searah jarum jam. F, berarti

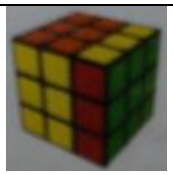
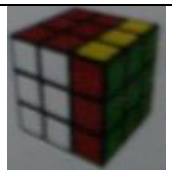
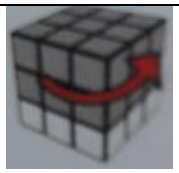
gerakan Front atau depan searah jarum jam. Kelompok kedua adalah: F', B', R', L', U', dan D' yang gerakannya adalah berlawanan dengan arah jarum jam. Untuk mendapatkan notasi-notasi di atas rubik harus diletakkan di depan pemain. Setiap notasi yang ditambahkan angka 2 dibelakangnya, berarti sisi/layer tersebut diputar 180 derajat atau setengah putaran.

2. *Double Turn* (Putaran Ganda)

Notasi *double turn* adalah notasi yang digunakan untuk memutar dua sisi (*layer*) dari rubik sekaligus sebesar seperempat putaran (90 derajat). Implementasi gerakan notasi huruf pada rubik dapat dilihat pada tabel 2.2 sebagai berikut :

Tabel 2.2. *Double Turn* Rubik (Putaran Ganda Rubik)

Notasi	**Gerakan**	**Hasil**
F		
F’		
B		

B’		
R		
R’		
L		
L’		
U		
U’		

D		
D’		

3. *Slice Turn* (Putaran Iris)

Notasi *slice turn* adalah notasi yang digunakan untuk memutar sisi tengah rubik. Adapun notasi – notasi tersebut menggunakan huruf :

a. **M** (*middle*)

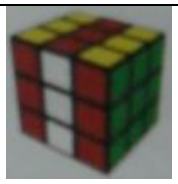
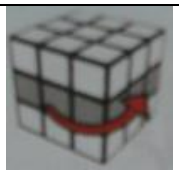
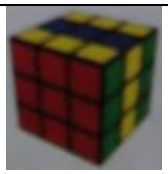
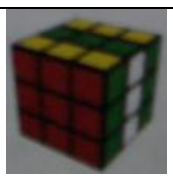
b. **E** (*Equator*)

c. **S** (*Stand*)

Implementasi gerakan notasi huruf pada rubik dapat dilihat pada tabel 2.3 sebagai berikut :

Tabel 2.3. *Slice Turn* Rubik (Putaran Iris Rubik)

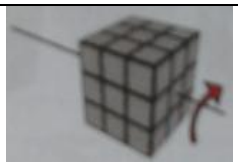
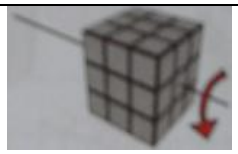
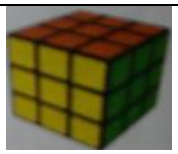
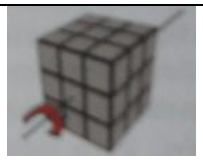
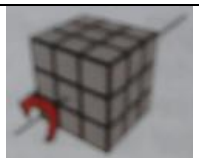
Notasi	**Gerakan**	**Hasil**
M		

M’		
E		
E’		
S		
S’		

4. *Rotation* (Rotasi)

Pada rotasi, rubik yang diputar bukanlah lapis atau *layer*, melainkan *cube* secara keseluruhan. Untuk lebih jelas perhatikan tabel 2.4 dibawah ini :

Tabel 2.4. *Rotation* Rubik (Putaran Rotasi Rubik)

Notasi	**Gerakan**	**Hasil**
x		
x’		
y		
y’		
z		
z’		

2.1.3. Pengaruh dan Manfaat Rubik [7]

Memainkan rubik mempunyai tantangan tersendiri, biasanya tantangannya adalah untuk dapat mengembalikan seluruh enam sisi kubus menjadi warna aslinya, tampaknya memiliki efek hipnotis pada beragam individu yang luar biasa tanpa mengenal batasan umur, pekerjaan, kekayaan dan kedudukan sosial. Biasanya anak muda, anak sekolah dan mahasiswa, yang berada dibarisan terdepan dalam membuat gerakan besar - besaran perkembangan rubik yang melanda dunia. Mereka adalah orang - orang yang terbukti paling ahli dalam memecahkan teka - teki, membentuk klub *rubik's cube*, mengatur tata cara kompetisi rubik, dan yang terpenting adalah bahwa mereka rela menderita sakit pergelangan tangan untuk bermain rubik terus - menerus selama berjam - jam dan bahkan berhari - hari dengan sebuah teka - teki tiga dimensi yang misterius itu.

Permainan rubik dapat dirasakan manfaatnya. Pertama, melatih saraf sensorik, ketika belajar mengenali warna dan pola dari bentuk 3D rubik. Kemudian, saraf motorik juga akan terlatih, karena koordinasi jari-jari tangan dalam bermain, terutama dalam melakukan *speedcubing* (bermain dengan kecepatan tinggi). Lebih jauh, rubik's cube akan melatih daya ingat ketika si pemain melakukan memorisasi pola-pola tertentu untuk menyelesaikan rubik yang teracak. Dan, terakhir, permainan ini akan melatih logika geometri atau susun bangun dalam kerangka otak pemain rubik.

2.1.4. Metode - Metode Penyelesaian Rubik [8]

Metode yang sangat dikenal dalam penyelesaian rubik yaitu Metode *Jessica Fridrich*, *Jessica Fridrich* adalah seorang dosen bergelar Professor yang berhasil menemukan metode tercepat dalam menyelesaikan rubik. Metode ini dikenal dengan sebutan CFOP (*Cross*-F2L-OLL-PLL). Ada 4 tahap yang perlu dilalui untuk menyamakan keenam sisi *Rubiks cube* dari posisi yang paling acak sekalipun. Berikut pembahasannya:

1. *Cross* (Metode *Layer By Layer*): Metode ini merupakan penyelesaian rubik secara *layer by layer*. Hal pertama yang harus dilakukan untuk menyelesaikan *Rubiks cube* adalah membuat *Cross.* Cross adalah tanda silang (atau tanda +) pada salah satu sisi warna *Rubiks cube*. *Cross* yang benar bisa dilihat digambar berikut:



Dari gambar disamping bisa kita lihat bahwa cross dibuat pada sisi putih. Dari gambar tersebut juga terlihat warna merah dan hijau (serta orange dan biru pada sisi sebaliknya) tergabung dengan *center*. Untuk membuat *cross* pada posisi *cube* teracak murni menggunakan logika dan latihan secara kontinyu. Pada awalnya seseorang pemula memerlukan waktu hingga 1 menit untuk membuat *cross*. Lambat laun proses ini bisa diturunkan hingga waktu idel 2-3 detik.

2. **F2L** : merupakan singkatan dari *First two Layer* atau tahap dimana kita menyelesaikan 2 layer pertama. Dalam metode *full fridrich* kita membentuk pasangan lalu memasukkannya ke 4 *slot* yang tersedia untuk menyelesaikan. *Layer*



1 dan 2 sekaligus. Tahap ini memerlukan latihan yang terus menerus. F2L tidak sulit untuk dipelajari, namun tingkat kesulitannya terletak pada bagaimana melakukannya secara cepat. F2L bisa ditekan hingga 8 detik rata - rata untuk menyelesaikannya. Bagi yang masih kesulitan belajar F2L bisa menggunakan metode *layer by layer* untuk menyelesaikan tahap ini.

Cross dan F2L digunakan untuk menyelesaikan *layer* 1 dan *layer* 2. Pada metode pemula untuk menyelesaikan *layer* 1 dan 2 dibagi dalam 3 tahap yakni membuat cross, menyelesaikan *layer* pertama (dengan menempatkan 4 *corner* ke tempat asalnya) lalu menyelesaikan *layer* kedua (memasukkan *edge* ke tempat yang sesuai). Perbedaan dengan F2L adalah, pada F2L *corner* dan *edge* digabungkan terlebih dulu, lalu dimasukkan ketempatnya sehingga lebih menghemat langkah dan waktu. Sementara itu untuk menyelesaikan *layer* terakhir dibagi dalam 2 tahap, yakni mengoreksi arah (OLL) dan mengoreksi tempat (PLL).

3. OLL: merupakan singkatan dari *Orientating Last Layer*. Segera setelah *layer* 1 dan



2 selesai, akan terbentuk pola tertentu dan tujuan kita adalah menyamakan warna sehingga '*top face*' akan memiliki warna yang sama (seringnya warna kuning, jika *cross* dilakukan pada warna putih dengan skema warna standar). Ada 57 kasus pada OLL yang menyebabkan kita perlu menghafal 57 langkah yang berbeda. Waktu eksekusi ideal untuk tahap ini adalah 2-3 detik. Bagi mereka yang belum mampu

menghafal 57 langkah sekaligus bisa melakukan OLL dalam 2 tahap (sebagai konsekuensinya waktu akan lebih lambat), yakni membetulkan seluruh *edge* dan disusul membetulkan seluruh *corner*. Hal ini dikenal sebagai 2 *look* OLL. Hanya perlu mempelajari 1-2 langkah untuk membetulkan orientasi *edge* dan 7 langkah untuk membetulkan orientasi *corner*.

4. PLL: Merupakan singkatan dari *Permutation Last layer*. Disini kita saling



menukar posisi *corner* dan *edge* ke tempat semula. Ada 21 langkah pada tahap ini yang secara ideal rata rata diseksekusi dalam 2-4 detik. Apabila sulit dalam menghafal 21 langkah bisa melakukannya dalam 2 tahap (dikenal sebagai 2 *look* PLL) yaitu mengoreksi letak *corner* disusul *edge*.

2.2 Algoritma *Layer by Layer* [7]

Ada beberapa metode penyelesaian rubik namun metode yang paling sederhana dan mudah untuk diaplikasikan adalah metode *layer by layer*. Algoritma *layer by layer* merupakan varian dari algoritma *greedy*, tetapi karena penggunaan algoritma ini dipakai untuk menyelesaikan permainan rubik maka untuk itu pada pengaplikasiannya algoritma *greedy* menjadi algoritma *layer by layer* untuk permainan rubik. Pada algoritma *greedy* yang diberikan yaitu membentuk solusi langkah per langkah (*step by step*) sama halnya dengan algoritma *layer by layer* pada permainan rubik. Beberapa tahapan dalam *layer by layer* terkadang menyediakan

alternatif solusi untuk setiap kondisi yang “mungkin” terjadi pada *cube* kita. Setiap kondisi tetap memiliki algoritma kunci yang harus kita hapalkan sisanya, kondisi untuk algoritma kunci tersebut yang kita ciptakan. Pada prinsipnya metode pemecahan *cube* secara *layer by layer* akan selalu melalui tahapan berikut:

1. *Form cross on first layer* – pembentukan silang pada *layer* pertama. Kuncinya adalah pastikan bagian tepi dari setiap *cross* memiliki warna yang sama dengan bagian tengah dari setiap sisi.
2. *Finishing first layer* – penyelesaian *layer* pertama. Tidak ada langkah khusus disini – namun justru ini yang membuat tahapan penyelesaian bisa menjadi panjang jika tidak melakukan perhitungan dari awal.
3. Penyelesaian *second layer* – penyelesaian *layer* kedua. Langkah yang perlu diperhatikan hanya langkah untuk menyelesaikan bagian tepi dari *layer* kedua. Rata-rata setiap tahapan untuk setiap tepi memakan 9 langkah.
4. *Form cross on last layer* – pembentukan silang pada *layer* terakhir. Ada dua langkah yang perlu diperhatikan dalam pembentukan *cross* di*layer* terakhir, yaitu jika sebagian cross sudah terbentuk pada dua sisi dengan kondisi (1) dua sisi berdekatan sudah terbentuk, atau (2) dua sisi berseberangan sudah terbentuk. Didalam tahapan ini tidak perlu langsung menyusun *cross* dengan warna yang sesuai, cukup bentuk *cross – regardless it’s color*.
5. Atur kesesuaian warna dari *cross* yang telah terbentuk. Satu langkah diperlukan untuk menyelesaikannya.

6. Atur masing - masing sudut pada *layer* terakhir. Pengaturan ini memerlukan paling tidak 2 langkah untuk mengatur posisi pada sudut yang benar dan mengatur warna pada setiap sudut.

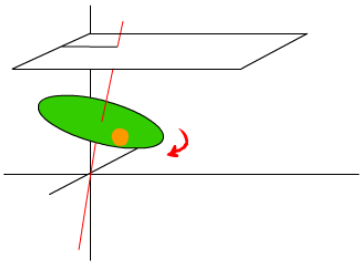
2.3 Dasar 3D Pada Flash [9]

Rotasi disekitar sumbu (x, y, z) 3d akan membahas tentang titik-titik 3D yang berputar disekitar sebuah sumbu. Untuk memutar data sekitar x atau y atau z sumbu, sebagai contoh, kita memutar (x, y, z) sekitar sumbu Z, maka kita dapat mengabaikan nilai z dan mengambil (x, y) dalam 2D. Untuk memutar (x, y) dalam 2D untuk delta busur sudut, kita hanya mengkonversi (x, y) data ke dalam jari-jari dan sudut. Radius adalah jarak (x, y) ke (0,0), yaitu Math.sqrt (x * x + y * y); busur sudut dalam format PI adalah Math.atan2 (y, x); sekarang memutar sudut delta. Radius adalah sama. Sementara sudut harus meningkat oleh delta. Jadi, yang baru (x, y) x = jari-jari * Math.cos (sudut + delta); y = radius * Math.sin (sudut + delta).

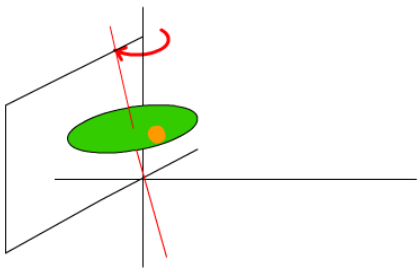
```
function rotateZ(x, y, center, dAngle)
{
        //note: here dAngle is in arc unit PI format
    var dy = y-center.y;
    var dx = x-center.x;
    var orgAngle = Math.atan2(dy, dx);
    var hypo = Math.sqrt(dy*dy+dx*dx);
    var newAngle = orgAngle+dAngle;
    var xx = hypo*Math.cos(newAngle)+center.x;
    var yy = hypo*Math.sin(newAngle)+center.y;
    var pt = {x:xx, y:yy};
    return pt;
}
```

Persamaan matematika yang sama dapat dimodifikasi dengan mudah untuk mengambil (z, x) untuk berputar disekitar sumbu Y, mengambil (y, z) untuk memutar

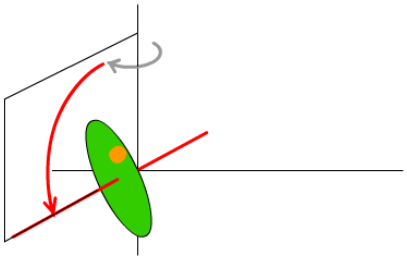
disekitar sumbu X. Tetapi bagaimana jika sumbu Z tidak lurus, tetapi memiliki sedikit penyimpangan? Maka kita tidak bisa hanya mengambil (x, y), kita perlu untuk menangani z. Jadi, setiap langkah adalah memutar disekitar sumbu koordinat. Pertama diputar disekitar sumbu Z, untuk membuat sumbu pada bidang YZ. Lalu memutar disekitar sumbu X untuk membuat sumbu yang sama sebagai sumbu Y, sekarang dapat dengan mudah melakukan rotasi di sekitar Y. Setelah ini, memutar dengan sudut yang sama seperti diatas untuk sumbu Y kembali ke YZ. Kemudian memutar balik sudut untuk membuat sumbu kembali ke vektor (x, y, z) asli. Lihat gambar dibawah ini untuk menunjukkan langkah-langkah.



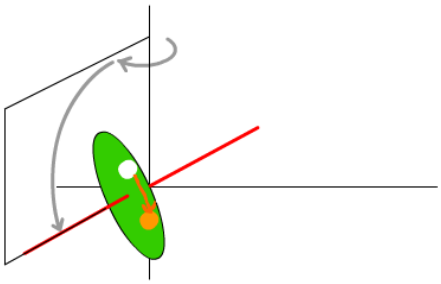
Gambar 2.6. Rotasi sekitar sumbu 3D



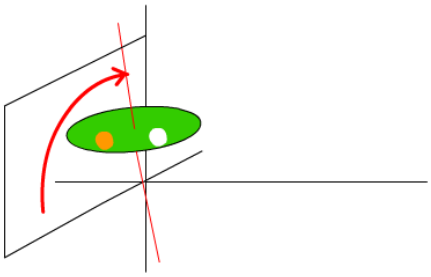
Gambar 2.7. Putaran sumbu Y



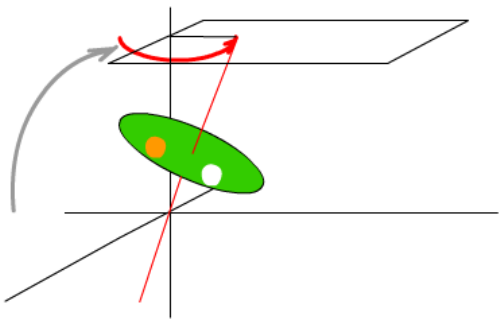
Gambar 2.8. Putaran sumbu X untuk membuat sumbu sesuai dengan sumbu Z



Gambar 2.9. Putaran sumbu Z



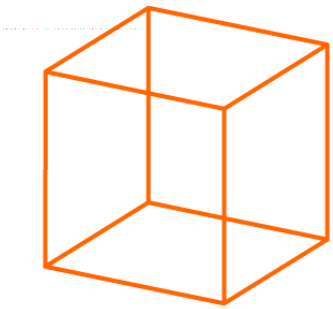
Gambar 2.10. Putaran sumbu X



Gambar 2.11. Memutar sumbu Y untuk membuat sudut

2.4 Memutar Kubus 3D [9]

Pada pembahasan ini, penulis menunjukkan cara membuat sebuah kubus 3D berputar, dengan permukaan bertekstur. Jadi langkah pertama adalah untuk setiap 8 titik data 3d untuk mewakili sebuah kubus dalam 3D.



Gambar 2.12. Kubus dalam 3D

```
// top face
p0 = {x:-50, y:-50, z:-50};
p1 = {x:50, y:-50, z:-50};
p2 = {x:50, y:-50, z:50};
p3 = {x:-50, y:-50, z:50};

// botton face
p4 = {x:-50, y:50, z:-50};
p5 = {x:50, y:50, z:-50};
p6 = {x:50, y:50, z:50};
p7 = {x:-50, y:50, z:50};

pointArray = [p0, p1, p2, p3, p4, p5, p6, p7];
// p0 to p8 , the 3d points of a cube;
```

Tidak ada efek perspektif, sehingga untuk membuat gambar 3D, kita misalkan z *off*. Ini 8 titik sudut untuk membangun menjadi kubus yang memiliki 6 sisi. Jadi, kita perlu 6 *movieklip skewing* untuk menutupi mereka. Bahkan, sebelum kita dapat menutupi sisinya, untuk praktek kita dapat mencoba untuk menggunakan 6 klip gelap

persegi untuk menutupi keenam sisi tersebut. Kemudian, kita tinggal memasang grafis pada kotak gelap untuk membuat mereka bertekstur.

```
clipArray = new Array();
for (k = 0; k < 6; k++)
{
    duplicateMovieClip (obj, "obj"+k, k+1);
    var clip = _root["obj"+k];
    clip.mc.holder.attachMovie("mv"+k, "mov", 1);
    clipArray[k] = clip;
}


center = {x:0, y:0, z:0};
function rotateZ (x1, y1, center, dAngle)
{
    var dy = y1-center.y;
    var dx = x1-center.x;
    var orgAngle = Math.atan2(dy, dx);
    var hypo = Math.sqrt(dy*dy+dx*dx);
    var newAngle = orgAngle+dAngle*(Math.PI/180);
    var xx = hypo*Math.cos(newAngle)+center.x;
    var yy = hypo*Math.sin(newAngle)+center.y;
    var pt = {x:xx, y:yy};
    return pt;
}
```

Persamaan di atas adalah sebuah rotasi pada gambar 2D. Hal ini berguna ketika kita melakukan rotasi dalam 2D. Hal ini juga dapat diterapkan ke gambar 3D. Kita perlu memperluas fungsi ini sedikit. Dalam hal ini, kita perlu memutar 8 titik sudut. Jadi, saya akan menyebutnya 8 kali. Juga, kita perlu rotasi sepanjang sumbu Y atau X. Persamaan ini hanya perbedaan dari x, y, z.

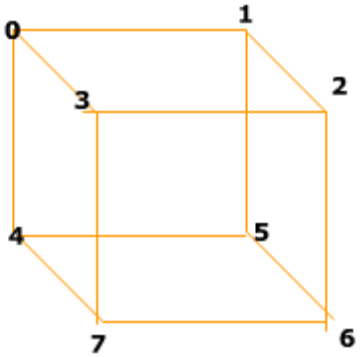
```
function groupRotateZ (pointArray, dAngle)
{
  for (k = 0; k < pointArray.length; k++)
        {
        var pt = rotateZ(pointArray[k].x, pointArray[k].y, _root.center, dAngle);
        pointArray[k].x = pt.x;
        pointArray[k].y = pt.y;
  }
}

function groupRotateY (pointArray, dAngle)
{
  for (k = 0; k < pointArray.length; k++)
        {
        var pt = rotateZ(pointArray[k].z, pointArray[k].x, _root.center, dAngle);
        pointArray[k].z = pt.x;
        pointArray[k].x = pt.y;
  }
}

function groupRotateX (pointArray, dAngle)
{
  for (k = 0; k < pointArray.length; k++)
        {
        var pt = rotateZ(pointArray[k].z, pointArray[k].y, _root.center, dAngle);
        pointArray[k].z = pt.x;
        pointArray[k].y = pt.y;
  }
}
```

Jadi, setiap 8 titik sudut condong 6 persegi klip untuk menutupi 6 sisi kubus. Hal ini diperbarui bila kita memutar ini 8 titik sepanjang x, y atau z sumbu. Kemudian, kita membuatnya mewah dengan menambahkan bayangan menurut data Z dan gambar yang melekat pada *MovieClips* persegi dari x, y, z.

```
p0 = {x:-50, y:-50, z:-50};
p1 = {x:50, y:-50, z:-50};
p2 = {x:50, y:-50, z:50};
p3 = {x:-50, y:-50, z:50};
// botton face
p4 = {x:-50, y:50, z:-50};
p5 = {x:50, y:50, z:-50};
p6 = {x:50, y:50, z:50};
p7 = {x:-50, y:50, z:50};
pointArray = [p0, p1, p2, p3,
p4, p5, p6, p7];
```



Langkah pertama untuk membangun kubus seperti gambar diatas adalah menyiapkan 8 titik dalam ruang 3d. Langkah kedua adalah menyiapkan 6 sisi.

```
clipArray=new Array();
clipArray[0].reg = [p0, p1, p2, p3];
// top
clipArray[1].reg = [p4, p5, p6, p7];
// bottom
clipArray[2].reg = [p0, p1, p5, p4];
// face 1
clipArray[3].reg = [p1, p2, p6, p5];
// face 2
clipArray[4].reg = [p2, p6, p7, p3];
// face 3
clipArray[5].reg = [p3, p7, p4, p0];
//face 4
```

Kemudian disetiap *movieclip* wajah, dan membuatnya menggambar segi empat diisi untuk menghubungkan sudut.

```
beginFill(fColor, 100);
pt1 = f3dPoint(clip.reg[0]);
pt2 = f3dPoint(clip.reg[1]);
pt3 = f3dPoint(clip.reg[2]);
pt4 = f3dPoint(clip.reg[3]);
moveTo(pt1.x, pt1.y);
lineTo(pt2.x, pt2.y);
lineTo(pt3.x, pt3.y);
lineTo(pt4.x, pt4.y);
endFill();
```

2.5 UML (*Unified Modeling Language*) [10]

UML *(Unified Modeling Language)* adalah sebuah bahasa untuk menentukan, visualisasi, kontruksi, dan mendokumentasikan *artifact* (bagian dari informasi yang digunakan atau dihasilkan dalam suatu proses pembuatan perangkat lunak. *Artifact* dapat berupa model, deskripsi atau perangkat lunak) dari sistem perangkat lunak, seperti pada pemodelan bisnis dan sistem non perangkat lunak lainnya.

UML merupakan suatu kumpulan teknik terbaik yang telah terbukti sukses dalam memodelkan sistem yang besar dan kompleks. UML tidak hanya digunakan dalam proses pemodelan perangkat lunak, namun hampir dalam semua bidang yang membutuhkan pemodelan.

Bagian-bagian utama dari UML adalah view, diagram, model element, dan *general mechanism*.

1. *View*

View digunakan untuk melihat sistem yang dimodelkan dari beberapa aspek yang berbeda. *View* bukan melihat grafik, tapi merupakan suatu abstraksi yang berisi sejumlah diagram. Beberapa jenis *view* dalam UML antara lain: *use case view, logical view, component view, concurrency view, dan deployment view.*

a) Use case view

Mendeskripsikan fungsionalitas sistem yang seharusnya dilakukan sesuai yang diinginkan *external actors*. *Actor* yang berinteraksi dengan sistem dapat berupa user atau sistem lainnya. *View* ini digambarkan dalam *use case diagrams* dan kadang-kadang dengan *activity diagrams*. *View* ini digunakan terutama untuk pelanggan, perancang (*designer*), pengembang (*developer*), dan penguji sistem (*tester*).

b) Logical view

Mendeskripsikan bagaimana fungsionalitas dari sistem, struktur statis (*class, object,dan relationship*) dan kolaborasi dinamis yang terjadi ketika *object* mengirim pesan ke *object* lain dalam suatu fungsi tertentu. *View* ini

digambarkan dalam class diagrams untuk struktur statis dan dalam *state, sequence, collaboration, dan activity diagram* untuk model dinamisnya. View ini digunakan untuk perancang (*designer*) dan pengembang (*developer*).

c) *Component view*

Mendeskripsikan implementasi dan ketergantungan modul. Komponen yang merupakan tipe lainnya dari code module diperlihatkan dengan struktur dan ketergantungannya juga alokasi sumber daya komponen dan informasi administrative lainnya. View ini digambarkan dalam *component view* dan digunakan untuk pengembang (*developer*).

d) *Concurrency view*

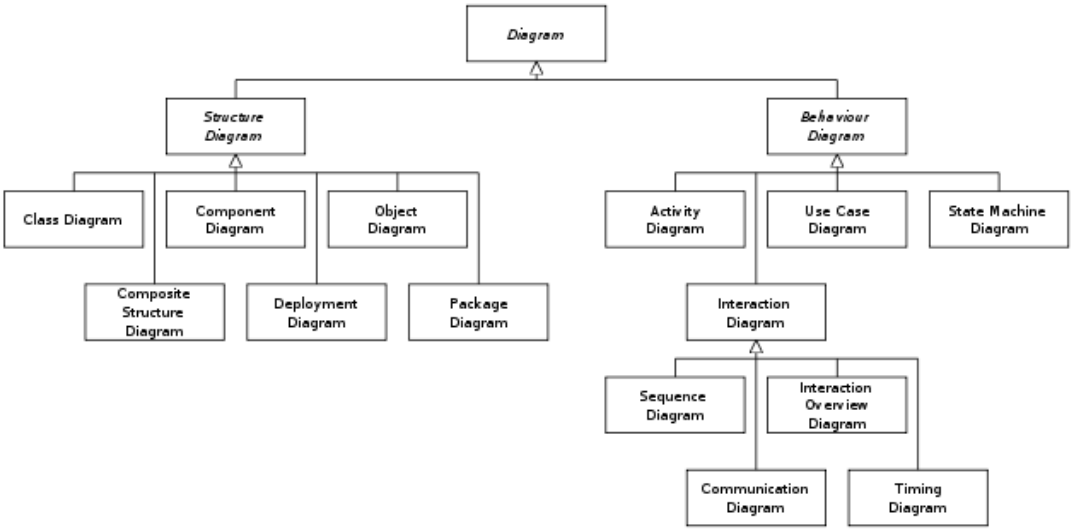
Membagi sistem ke dalam proses dan prosesor. *View* ini digambarkan dalam diagram dinamis *(state, sequence, collaboration, dan activity diagrams)* dan diagram implementasi *(component dan deployment diagrams)* serta digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

e) *Deployment view*

Mendeskripsikan fisik dari sistem seperti komputer dan perangkat (*nodes*) dan bagaimana hubungannya dengan lainnya. *View* ini digambarkan dalam *deployment diagrams* dan digunakan untuk pengembang (*developer*), pengintegrasi (*integrator*), dan penguji (*tester*).

2. Diagram

Diagram berbentuk grafik yang menunjukkan simbol elemen model yang disusun untuk mengilustrasikan bagian atau aspek tertentu dari sistem. Sebuah diagram merupakan bagian dari suatu *view* tertentu dan ketika digambarkan biasanya dialokasikan untuk *view* tertentu. Adapun jenis diagram antara lain dapat dilihat pada gambar 2.13 sebagai berikut:



Gambar 2.13. Diagram UML

Penjelasan diagram UML pada gambar 2.13 akan lebih jelas dapat dilihat dibawah ini:

a) *Use Case Diagram*

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

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

b) *Class Diagram*

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

c) *Component Diagram*

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

d) *Deployment Diagram*

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

e) *State Diagram*

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

f) *Sequence Diagram*

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

f) *Collaboration Diagram*

Menggambarkan kolaborasi dinamis seperti *sequence* diagrams. Dalam menunjukkan pertukaran pesan, *collaboration* diagrams menggambarkan *object* dan hubungannya (mengacu ke konteks). Jika

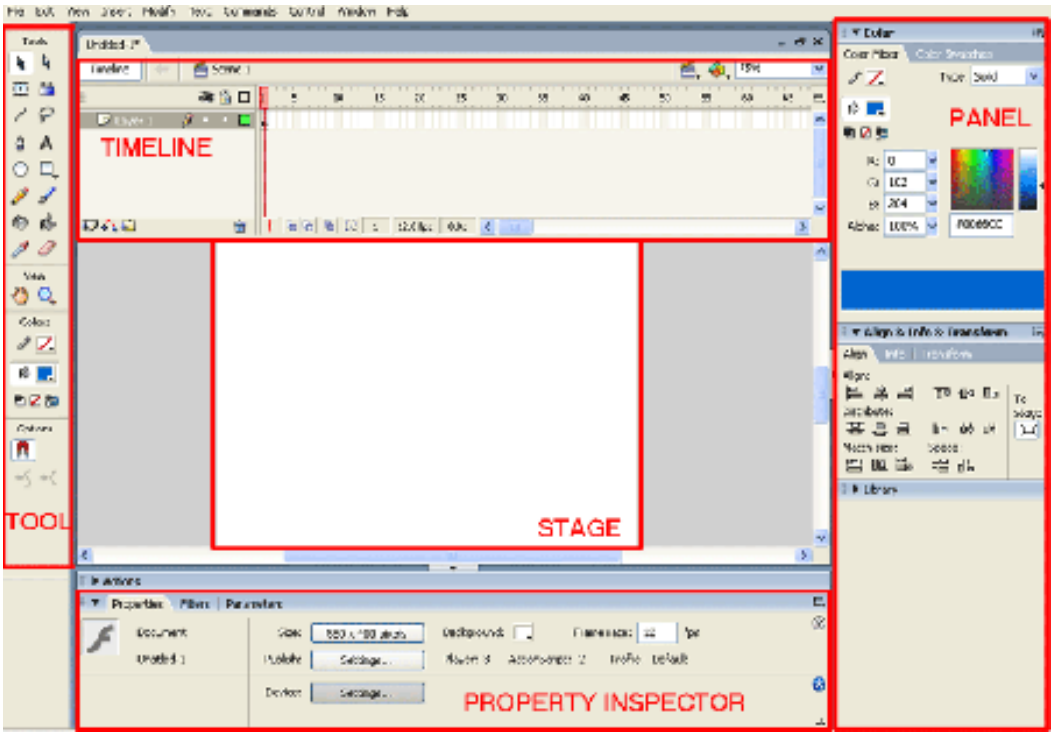
penekannya pada waktu atau urutan gunakan *sequence* diagrams, tapi jika penekanannya pada konteks gunakan *collaboration* diagram.

g) *Activity Diagram*

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

2.6 Macromedia Flash 8 [11]

Flash adalah perangkat lunak komputer (*soft-ware*) dalam kategori *authoring tool*. Program *flash* ini biasa digunakan untuk membangun sebuah aplikasi yang memerlukan interaksi dari pemakainya (*user*). Dengan kemudahan yang ditawarkan, beberapa kalangan diakademisi bahkan mulai menggunakan *flash* untuk membuat sebuah presentasi yang menarik. Sabagai *authoring tool*, *flash* mempunyai kemampuan untuk memadukan semua media yang ada (*multimedia*) dari mulai sebuah animasi yang sederhana, video dan gabungan antara gerak gambar, suara dan video sekaligus. *Flash* menggunakan konsep grafis *vector*. Penggunaan konsep dasar *vector* membuat aplikasi yang dihasilkan oleh *flash* mempunyai besaran file relative berukuran kecil dengan tetap mempertahankan kualitas. Hal ini tentu saja sangat menguntungkan untuk digunakan dimana saja termasuk didalam dunia internet. *User Interface Macromedia Flash* 8 dapat dilihat pada gambar 2.14 sebagai berikut:



Gambar 2.14. *User Interface Macromedia Flash* 8

Pada gambar 2.14 *User interface* di *Flash* terbagi atas 5 bagian, yaitu penjelasannya sebagai berikut:

1. Stage

Stage merupakan daerah yang bewarna putih, dimana *area* kerja utama jika akan membuat animasi maupun aplikasi *flash* lainnya. Seluruh objek/gambar/animasi yang ada *distage* nantinya akan tampil di*flash movie*, dan sebaliknya apabila objek/gambar tersebut berada di daerah abu-abu di pinggiran *Stage* tidak akan terlihat di*flash movie*.

2. Tools

Merupakan kumpulan alat gambar dan mewarnai sesuatu yang ada di *stage*.

3. Panel

Merupakan kumpulan kotak-kotak yang berfungsi untuk mengubah, mengatur dan mempercantik objek yang ada di *stage*. Misalkan memberi warna gradasi, meletakan objek persis di tengah *stage*, dan merotasi objek dengan sudut yang tepat.

4. property inspector

Merupakan bagian informasi objek yang ada di *stage*. sebagai contoh, klik saja salah satu objek di *stage*, maka informasinya akan terlihat disini. atau klik saja di daerah kosong yang ada di *stage*, maka informasi mengenai *stage* akan terlihat.

5. Timeline

Terdiri dari baris dan kolom. Kolom berhubungan dengan waktu, baris berhubungan dengan objek. Setiap *software* animasi pasti memliki *timeline* untuk mencatat aktivitas objek kapan harus tampil di *stage* dan kapan harus menghilang.

2.6.1 Pengenalan Tool Utama Dalam Flash [12]

Ada beberapa *too*l utama dalam *flash*, dapat dilihat pada gambar 2.15 dibawah ini serta penjelasannya sebagai berikut :

1. *Selection Tool (V)*

Merupakan tool utama yang berfungsi untuk menseleksi benda, baik berupa *shape, line,* titik maupun simbol.

2. *Subselection Tool (A)*

Merupakan tool utama yang berfungsi untuk menseleksi dan memodifikasi titik maupun garis (*stroke*).

3. *Free Transform Tool (Q)*

Merupakan *tool* yang berfungsi untuk memodifikasi *scale* (skala), *skew* (kemiringan), *rotation* (rotasi), *distorsi*, dan *envelop*.

4. *Gradien transform tool (F)*

Merupakan *tool* yang berfungsi untuk memodifikasi warna warna gradasi.

5. *Line tool (N)*

Tool yang berfungsi untuk membentuk garis.

6. *Laso Tool (L*)

Tool yang berfungsi untuk menyeleksi benda.

7. Pen Tool (P)

Tool yang berfungsi untuk membuat sebuah bentuk *path*, dapat digunakan juga untuk *tracing*.

8. Text Tool (T)

Tool yang berfungsi untuk membuat tulisan.

9. Oval Tool (O)

Tool yang berfungsi untuk membuat bentuk lingkaran.

10. Rectangle Tool (R)

Tool yang berfungsi untuk membuat bentuk kotak. *Rectangle Tool* juga memiliki *sub tool*, yakni *polystar Tool* yang berfungsi membuat bermacam bentuk segi.

11. Pencil Tool (Y)

Pencil Tool berfungsi untuk membuat goresan yang seperti pensil. setiap goresan akan menjadi *tipe stroke*.

12. Brush Tool (B)

Berfungsi untuk membuat goresan seperti *brush* (kuas) yang merupakan tipe *fill*.

13. Ink bottle Tool (S)

Berfungsi untuk membuat *stroke.*

14. Paint bucket (K)

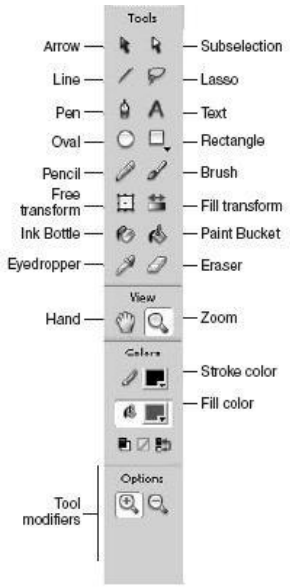
Berfungsi untuk membuat *fill*.

14. Eye dropper (I)

Berfungsi untuk mengambil sampel warna.

15. Eraser (E)

Berfungsi sebagai penghapus.



Gambar 2.15. *Tool* utama dalam *flash*

2.6.2 Hal-Hal Dasar Dalam *Flash* [13]

Hal – hal dasar dalam flash akan membahas mengenai bagaimana mengetahui dan mengubah properti, mengerti konsep *shape* dan sifatnya, *group* dan *ungrou*p, *zoom* dan *hand tool*. Penjelasannya sebagai berikut:

1. Mengetahui dan mengubah *properti*

Property inspector merupakan salah satu bagian yang sangat penting di *flash*. Disini dapat diketahui properti/informasi detail tiap objek, Klik daerah kosong di *stage*, kemudian dapat terlihat ada beberapa informasi: size : 550 x 400. yang berarti tinggi *stage* adalah 550 *pixel* dan lebar *stage* 400 *pixel*. *Background*: warna *stage Frame Rate*: 12 fps (12 *frame per second*) yang berarti dalam 1 detik akan di jalankan

12 *frame*. Sedangkan tombol *publish setting* berfungsi untuk mengatur *setting*-an detail pada saat akan mempublikasi *file flash* yang telah kita buat.

2. Mengerti konsep *shape* dan sifatnya

Shape terdiri dari garis (*stroke*) dan warna isian (*fill*). Kita akan belajar memahami sifat-sifat dari *shape* karena banyak orang yang kebingungan bila tidak memahami sifat *shape*.

a. *Fill*

Shape merupakan warna isian dari sebuah *shape.*

b. *Stroke*

Stroke tidak memiliki garis karena *stroke* itu sendiri adalah garis. Dengan adanya *stroke style,* bentuk *stroke* bisa diubah menjadi beberapa pilihan. Cara mengubah *stroke style*: seleksi terlebih dahulu *stroke obyek* yang ada, kemudian pilih *stroke style* dibagian *property inspector* (*setting default stroke style* adalah solid).

Ketika membuat sebuah *shape,* secara otomatis terbentuk *fill* dan *stroke*. namun *fill* dan *stroke* ini bisa dipisahkan. Klik 1x daerah *fill*, maka kita hanya menseleksi *fill*-nya saja. Untuk menseleksi keduanya, klik 2x daerah *fill*. untuk menseleksi *stroke*-nya saja, klik 1x pada *stroke*. Untuk memisahkan *fill* dan *stroke,* seleksi *fill* atau *stroke,* lalu drag/tarik. Untuk melepas seleksi, klik daerah luar/*workspace/stage* yang tidak ada objek. Pemberian warna *stroke* dan *fill* memiliki cara yang berbeda. Untuk memberi warna garis atau *stroke,* gunakanlah *ink bottle Tool (S)* untuk memberi warna *fill,* gunakanlah *Paint bucket Tool* (K). Apabila *shape*

sudah berisi warna dan ingin mengubahnya, pilih *selection Tool (V)* seleksi *fill* atau *stroke* yang akan diubah warna nya, selanjutnya pada *Tool Color*, kita bisa menggunakan *stroke color* untuk mewarnai garis dan *fill color* untuk mewarnai daerah isian warna.

3. *Group* dan *Ungroup*

Buat dua bentuk lingkaran atau kotak yang jaraknya berjauhan, kemudian seleksi 1 objek diantaranya lalu pindahkan tepat di atas objek yang lain lalu pisahkan. Yang terjadi adalah obejek yang tertimpa menjadi rusak. Untuk menghindari hal ini, gunakanlah grup dengan cara seleksi terlebih dahulu objek yang akan digrup, lalu tekan menu *Modify>Grup* atau gunakan *shortcut* CTRL+G. Untuk memecah grup, gunakan *ungroup* dengan cara pilih menu *Modify>Ungroup* atau dengan *shortcut* CTRL+SHIFT+G. Objek yang sudah digrup tidak dapat diubah warnanya secara langsung seperti biasa. Jika ingin mengubah warna objek, bisa di lakukan dengan cara masuk ke dalam grup dengan cara klik dua kali grupnya, lalu ubah warnanya. atau dengan cara mengungroup terlebih dahulu objek tersebut.

4. *Zoom* dan *Hand Tool*

Zoom tool (gambar kaca pembesar, *shortcut* Z) berfungsi Untuk memperbesar dan memperkecil tampilan objek-objek yang ada di *stage*. Klik *zoom tool*, lalu klik pada *stage* untuk *zoom-in*, untuk *zoom-out* tahan tombol Alt, lalu klik. Adapun cara alternatif selain *zoom tool* adalah CTRL + dan CTRL - Hand Tool (gambar tangan, shortcut H) berfungsi sebagai penggeser tampilan *stage*. Adapun cara alternatif dengan cara menahan tombol spasi.