

SKRIPSI

OPEN SOURCE SNAKE 360



Evelyn Wijaya

NPM: 2015730030

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
2019**

UNDERGRADUATE THESIS

OPEN SOURCE SNAKE 360



Evelyn Wijaya

NPM: 2015730030

**DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY AND SCIENCES
PARAHYANGAN CATHOLIC UNIVERSITY
2019**

ABSTRAK

Penelitian ini membahas mengenai pembangunan permainan Open Source Snake 360. Permainan ini dibuat berdasarkan acuan dari permainan Snake yang sudah ada. Permainan Snake adalah permainan di mana pemain mengontrol gerakan ular untuk mendapatkan makanan yang tersebar di labirin. Setiap ular memakan makanan, pemain akan mendapatkan skor. Pada permainan ini, pemain harus mengontrol ular untuk mendapatkan makanan sebanyak-banyaknya tanpa menabrak dinding labirin atau dirinya sendiri.

Kata-kata kunci: Snake Game, HTML, Javascript, jQuery, Github

ABSTRACT

Keywords: Snake Game, HTML, Javascript, jQuery, Github

DAFTAR ISI

DAFTAR ISI	ix
DAFTAR GAMBAR	xi
DAFTAR TABEL	xiii
1 PENDAHULUAN	1
1.1 Latar Belakang	1
1.2 Rumusan Masalah	1
1.3 Tujuan	2
1.4 Batasan Masalah	2
1.5 Metodologi	2
1.6 Sistematika Pembahasan	2
2 LANDASAN TEORI	5
2.1 <i>Snake</i>	5
2.2 HTML5 <i>Canvas</i>	6
2.3 <i>Javascript</i>	7
2.3.1 Variabel	7
2.3.2 <i>Constant</i>	8
2.3.3 <i>Function</i>	8
2.3.4 Menggambar pada <i>Canvas</i>	8
2.3.5 <i>Object Oriented Programming Javascript</i>	11
2.3.6 <i>Event</i>	14
2.3.7 Membuat Animasi	16
2.4 <i>jQuery</i>	17
2.4.1 Mendapatkan dan Mengubah Konten Elemen	18
2.4.2 Mendapatkan dan Mengubah Properti CSS	19
2.4.3 Looping	20
2.4.4 Event	20
2.4.5 AJAX	21
2.5 <i>Git</i>	24
2.5.1 <i>Version Control</i>	24
2.5.2 <i>Git</i>	26
2.5.3 <i>Git Branching</i>	29
2.5.4 <i>GitHub</i>	35
3 ANALISIS	37
3.1 Analisis Permainan <i>Snake</i> yang Sudah Ada	37
3.1.1 Ular dan Makanan	37
3.1.2 Pergerakan Ular	39
3.1.3 Labirin	39
3.2 Analisis Sistem yang Dibangun	40

3.2.1	Menentukan Besar <i>Canvas</i>	40
3.2.2	Menggambar Ular dan Apel	40
3.2.3	Pergerakan Ular	43
3.2.4	Mengacak posisi apel	44
3.2.5	Menggambar Labirin	44
3.2.6	Pengecekan tabrakan(<i>Collision Detection</i>)	46
3.3	Analisis Berorientasi Objek	48
3.3.1	Skenario Permainan	48
3.3.2	Diagram Kelas	49
4	PERANCANGAN	51
4.1	Diagram Sequence	51
4.1.1	Memilih Level dan Kecepatan	51
4.2	Diagram Kelas Rinci	52
4.3	Mockup	55
4.3.1	Tampilan Menu Utama	55
4.3.2	Tampilan Bermain	56
4.3.3	Tampilan Permainan Berakhir	57
5	IMPLEMENTASI DAN PENGUJIAN	59
5.1	Implementasi	59
5.1.1	Lingkungan Perangkat Keras	59
5.1.2	Lingkungan Perangkat Lunak	60
5.1.3	Implementasi Antarmuka	60
5.2	Pengujian	62
5.2.1	Pengujian Fungsional	62
6	KESIMPULAN DAN SARAN	65
6.0.1	Kesimpulan	65
6.0.2	Saran	65
	DAFTAR REFERENSI	67
	A KODE PROGRAM	69
	B HASIL EKSPERIMEN	77

DAFTAR GAMBAR

2.1	Permainan Snake pada telepon genggam <i>Nokia</i>	5
2.2	Permainan <i>Slither.io</i> pada <i>Android</i>	6
2.3	Posisi kotak biru pada <i>canvas</i> terhadap <i>origin</i>	9
2.4	Perbedaan <i>quadratic Bézier curve</i> dan <i>cubic Bézier curve</i>	11
2.5	Local Version Control	25
2.6	Centralized Version Control	25
2.7	Distributed Version Control	26
2.8	Working tree, staging area, dan Git directory	27
2.9	Siklus hidup pada status <i>file</i>	28
2.10	Commit dan tree dari file yang dicommit	30
2.11	Commit dan parent dari commit	30
2.12	<i>Pointer HEAD</i> menunjuk <i>branch master</i>	31
2.13	<i>Pointer HEAD</i> beserta <i>branch testing</i>	31
2.14	3 <i>snapshot</i> yang digunakan dalam <i>three way merge</i>	32
2.15	<i>Merge commit</i>	32
2.16	Perbedaan pada <i>branch</i> lokal dan <i>remote</i>	33
2.17	<i>Update remote-tracking branches</i> menggunakan perintah <i>git fetch</i>	33
2.18	<i>Rebasing commit C4</i> ke <i>C3</i>	34
2.19	<i>Merge branch</i> setelah <i>rebasing</i>	35
2.20	Tombol 'Fork'	35
3.1	Ular pada <i>Slither.io</i>	38
3.2	Makanan pada <i>Slither.io</i>	38
3.3	Ular pada <i>Snake Nokia</i>	38
3.4	Makanan biasa(A) dan makanan bonus(B) pada <i>Snake Nokia</i>	38
3.5	Ular sedang melaju dengan cepat(<i>speed up</i>)	39
3.6	Peta labirin pada <i>Slither.io</i>	39
3.7	Koordinat bagian tubuh ular pada <i>array</i>	40
3.8	Tubuh ular setelah digambar menggunakan garis	41
3.9	Bagian pada apel(lingkaran merah) yang akan dibuat menggunakan kurva	41
3.10	Pembagian gambar apel dengan layout persegi beserta ukuran pada setiap bagian	42
3.11	<i>Start point</i> , <i>control point</i> dan <i>end point</i> untuk menggambar apel bagian kiri atas	42
3.12	<i>Start point</i> , <i>control point</i> dan <i>end point</i> untuk menggambar apel bagian kiri bawah	43
3.13	Ilustrasi ular sebelum bergerak maju(A) dan setelah bergerak maju(B)	43
3.14	Gambar apel yang terpotong sesudah mengacak posisi apel	44
3.15	Menggambar dinding menggunakan simbol pada file text	45
3.16	Ular ingin melewati jalur yang diapit oleh 2 buah dinding	45
3.17	Daerah tabrakan pada apel	46
3.18	Daerah tabrakan berbentuk persegi pada apel	47
3.19	Diagram <i>use case</i> dari permainan <i>Snake 360</i>	48
3.20	Diagram class dari permainan <i>Snake 360</i>	49
4.1	Diagram sequence untuk memilih level dan kecepatan	51

4.2	Diagram class rinci dari Open Source <i>Snake 360</i>	52
4.3	Rancangan tampilan menu utama	56
4.4	Rancangan tampilan menu utama jika pemain salah memasukkan data	56
4.5	Rancangan tampilan bermain	57
4.6	Rancangan tampilan permainan berakhir	57
5.1	Tampilan Menu Utama	61
5.2	Tampilan Bermain	61
B.1	Hasil 1	77
B.2	Hasil 2	77
B.3	Hasil 3	77
B.4	Hasil 4	77

DAFTAR TABEL

5.1	Pengujian Fungsional pada Tampilan Menu Utama	62
5.2	Pengujian Fungsional Tampilan Bermain pada Desktop	62
5.3	Pengujian Fungsional pada Tampilan "game over"	63

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Snake merupakan sebuah permainan yang pertama kali dibuat oleh Peter Trefonas pada tahun 1978. Konsep *Snake* berasal dari permainan arkade yaitu *Blockade*. Awalnya *Snake* hanya dapat dimainkan pada komputer pribadi. Namun pada tahun 1997, *Snake* dapat dimainkan pada telepon genggam *Nokia*¹. Cara bermain *Snake* adalah pemain menggerakkan ular pada sebuah labirin. Ular tersebut harus mendapatkan makanan sebanyak-banyaknya tanpa menabrak dinding atau ular itu sendiri. Setiap memakan makanan, tubuh ular akan memanjang dan pemain akan semakin sulit untuk menggerakkan ular tersebut dengan bebas karena tubuh ular semakin lama akan menutupi labirin tersebut.

HTML (*Hyper Text Markup Language*) adalah sebuah bahasa markah yang digunakan untuk membuat halaman web. HTML5 merupakan HTML versi 5 yang terbaru dan penerus dari HTML4, XHTML1, dan DOM level 2 HTML. HTML5 memiliki beberapa elemen baru, salah satunya adalah HTML5 Canvas. HTML5 Canvas adalah tempat untuk menggambar *pixel-pixel* yang dapat ditulis menggunakan bahasa pemrograman *JavaScript*. *Javascript* adalah bahasa pemrograman tingkat tinggi yang digunakan untuk membuat halaman web menjadi lebih interaktif. *jQuery* merupakan library milik Javascript. *GitHub* adalah layanan *web hosting* bersama untuk proyek pengembangan perangkat lunak yang menggunakan sistem *version control* yaitu *Git*. Dengan adanya *Github*, *programmer* dapat mengetahui perubahan yang pada *repository* tersebut.

Pada permainan *Snake*, umumnya pergerakan ular hanya atas, bawah, kiri, dan kanan saja. Pada skripsi ini, penulis akan membuat permainan *Snake* yang ularnya dapat bergerak ke segala arah dan orang lain dapat menambahkan labirin menggunakan mekanisme *pull request Github*. Dengan begitu, orang lain dapat menambahkan labirin sesuai dengan keinginannya dan pemain tidak akan cepat bosan karena labirin yang disediakan cukup banyak dan variatif.

1.2 Rumusan Masalah

Rumusan dari masalah yang akan dibahas pada skripsi ini adalah sebagai berikut:

- Bagaimana membangun permainan *Snake* menggunakan HTML5?
- Bagaimana cara menyimpan labirin pada file eksternal?

¹[https://en.wikipedia.org/wiki/Snake_\(video_game_genre\)](https://en.wikipedia.org/wiki/Snake_(video_game_genre))

- Bagaimana cara menggunakan *pull request* pada *Github* agar orang lain dapat menambahkan labirin?

1.3 Tujuan

Tujuan-tujuan yang hendak dicapai melalui penulisan skripsi ini adalah sebagai berikut:

- Dapat membangun permainan *Snake* menggunakan HTML5.
- Dapat menyimpan labirin pada file eksternal.
- Dapat menggunakan *pull request* pada *Github* agar orang lain dapat menambahkan labirin.

1.4 Batasan Masalah

Beberapa batasan yang dibuat terkait dengan pengerjaan skripsi ini adalah sebagai berikut:

- Permainan ini hanya dapat dimainkan menggunakan *web browser* khususnya pada *desktop* dan *smartphone*.
- *Web browser* yang digunakan sudah mendukung HTML5 Canvas.

1.5 Metodologi

Metodologi pada penelitian ini adalah sebagai berikut:

1. Melakukan studi literatur tentang HTML5, *JavaScript*, *jQuery*, dan *Git*.
2. Melakukan analisis dan menentukan objek-objek pada *Snake*.
3. Merancang algoritma untuk menggambar tubuh ular, pergerakan ular dan membuat labirin.
4. Mengimplementasikan keseluruhan algoritma.
5. Menambahkan labirin menggunakan *pull request* pada *Github*.
6. Melakukan pengujian.
7. Melakukan penarikan kesimpulan.

1.6 Sistematika Pembahasan

Sistematikan penulisan setiap bab pada penelitian ini adalah sebagai berikut:

1. Bab 1 berisikan latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan dari penelitian yang dilakukan.
2. Bab 2 berisikan dasar-dasar teori yang menunjang penelitian ini. Teori yang digunakan adalah: pengertian *Snake*, HTML5 Canvas, *Javascript*, *jQuery*, dan *Git*.

-
- 1 3. Bab 3 berisikan analisis sistem yang sudah ada, analisis sistem yang dibangun dan analisis
2 berorientasi objek.
 - 3 4. Bab 4 berisikan perancangan perangkat lunak yang dibangun. Perancangan yang dilakukan
4 meliputi diagram *sequence*, diagram kelas dan *mockup*.
 - 5 5. Bab 5 berisikan implementasi dan pengujian perangkat lunak.
 - 6 6. Bab 6 berisikan kesimpulan dan saran.

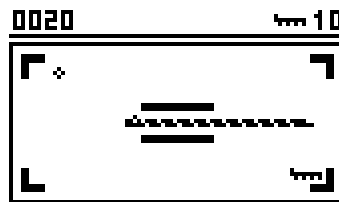
BAB 2

LANDASAN TEORI

2.1 *Snake*

Snake merupakan permainan mengendalikan ular untuk mendapatkan makanan yang terdapat pada labirin. Dalam permainan ini, pemain mengendalikan ular untuk mendapatkan makanan sebanyak-banyaknya. Setiap ular memakan makanan, maka skor akan bertambah 1 poin dan tubuh ular akan bertambah panjang. Biasanya makanan hanya ada 1 saja pada sebuah labirin. Ketika makanan itu sudah termakan oleh ular, makanan tersebut akan ditempatkan secara acak. Ular dapat bergerak ke atas, bawah, kiri, dan kanan. Permainan akan berakhir jika ular menabrak dinding yang terdapat pada labirin atau ular tersebut menabrak tubuhnya sendiri.

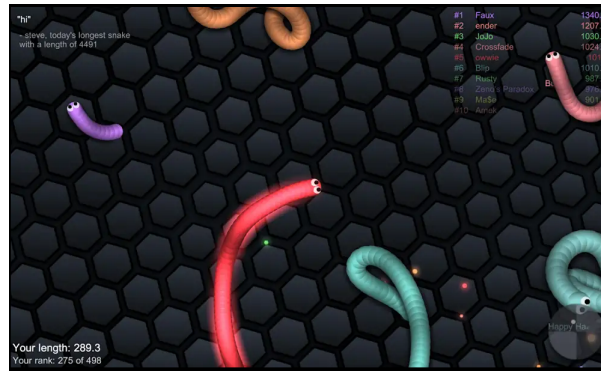
Permainan *Snake* ini dapat dimainkan secara *singleplayer* atau *multiplayer*. *Singleplayer game* adalah permainan yang dapat dimainkan oleh 1 pemain. *Multiplayer game* adalah permainan yang dapat dimainkan oleh beberapa pemain. Pada umumnya, permainan *Snake* dimainkan secara *singleplayer*. Contoh *singleplayer game Snake* adalah *Snake* pada telepon genggam *Nokia* yang dapat dilihat pada Gambar 2.1¹ dan contoh *multiplayer game Snake* adalah *Slither.io* yang dapat dilihat Gambar 2.2². *Snake* dapat dimainkan menggunakan *smartphone* dan *web browser*.



Gambar 2.1: Permainan Snake pada telepon genggam *Nokia*

¹[https://en.wikipedia.org/wiki/Snake_\(video_game_genre\)](https://en.wikipedia.org/wiki/Snake_(video_game_genre))

²<https://play.google.com/store/apps/details?id=air.com.hypah.io.slither>

Gambar 2.2: Permainan *Slither.io* pada *Android*

2.2 HTML5 Canvas

HTML5 Canvas adalah sebuah daerah *bitmap* yang dapat dimanipulasi oleh *Javascript* [1]. Pada daerah *bitmap* tersebut, *pixel-pixel* akan dirender oleh canvas. Setiap *frame*, HTML5 Canvas akan menggambar pada area *bitmap* tersebut menggunakan *Canvas API* (*Application Programming Interface*) yang dipanggil pada *Javascript*. API dari HTML5 Canvas yang umum adalah *2D Context*. Dengan adanya *2D Context*, *programmer* dapat membuat bentuk 2D, menampilkan gambar, *render* tulisan, memberi warna, membuat garis dan kurva, dan manipulasi *pixel*. HTML5 Canvas tidak hanya digunakan untuk menggambar dan menampilkan gambar serta tulisan. HTML5 Canvas dapat digunakan untuk membuat animasi, aplikasi pada *web* dan permainan.

Untuk menambahkan *canvas* pada halaman HTML, diperlukan tag `<canvas>`. Pada *listing 2.1* terdapat potongan kode untuk menambahkan *canvas* pada halaman HTML.

```
<canvas id='canvas' width='500' height='300'>
  Your browser does not support HTML5 Canvas.
</canvas>
```

Listing 2.1: Menambahkan *canvas*

Diantara tag `<canvas>` dan `</canvas>`, dapat dituliskan *text* yang akan ditampilkan jika browser tidak support HTML5 Canvas.

Canvas memiliki beberapa atribut diantaranya adalah:

- *id* : nama yang digunakan sebagai referensi objek canvas yang nantinya akan digunakan pada *Javascript*.
- *width* : lebar dari canvas.
- *height* : tinggi dari canvas.
- *title* : judul sebuah elemen.
- *draggable* : mengambil sebuah objek dan membawanya ke tempat lain
- *tabindex* : memfokuskan pada suatu elemen jika tombol tab ditekan.

- *class* : kelas pada elemen. Biasanya digunakan oleh CSS dan *Javascript* untuk mengakses elemen tertentu.
- *dir* : arah penulisan (dari kiri ke kanan atau dari kanan ke kiri)
- *hidden* : membuat elemen menjadi tersembunyi/tidak terlihat
- *accesskey* : memberikan petunjuk untuk membuat pintasan *keyboard* pada sebuah elemen.

2.3 Javascript

Javascript adalah bahasa pemrograman yang ringan, *interpreted* dan berorientasi objek yang digunakan pada halaman *web* [2]. *Javascript* dapat membuat objek dengan menambahkan *method* dan atributnya sama seperti bahasa pemrograman C++ dan *Java*. Setelah objek diinisialisasi, maka objek tersebut dapat dijadikan *blueprint* untuk membuat objek lain yang mirip. *Javascript* dapat digunakan untuk mengimplementasi hal yang kompleks pada halaman web. Contohnya adalah menampilkan peta yang interaktif dan membuat animasi 2D/3D. Selain *Javascript*, HTML(*HyperText Markup Language*) dan CSS(*Cascading Style Sheet*) merupakan bagian/komponen penting dalam pembuatan halaman *web*.

Untuk menambahkan *Javascript* pada sebuah halaman web yang dibuat, gunakan *tag* `<script>`. Ada 2 cara untuk menambahkan *Javascript* yaitu menambahkan langsung di halaman web tersebut(*Internal Javascript*) atau menambahkan *file Javascript* terpisah(*External Javascript*).

2.3.1 Variabel

Variabel adalah sebuah wadah untuk menyimpan nilai/*value*. Untuk mendeklarasi variabel pada *Javascript*, digunakan *keyword* `'var'`. Variabel pada *Javascript* tidak perlu menuliskan tipe datanya ketika mendeklarasikan variabel. Pada *listing 2.2* terdapat potongan kode untuk mendeklarasikan variabel.

```
var myVariable;
```

Listing 2.2: Deklarasi variabel

Nilai variabel pada potongan kode di atas adalah *undifined* karena variabel tersebut tidak diberi nilai/*value*. Pada *listing 2.3* terdapat potongan kode untuk mengisi nilai pada variabel.

```
myVariable = 3;
```

Listing 2.3: Mengisi nilai sebuah variabel

Variabel dapat menyimpan beberapa tipe data diantaranya adalah:

- *String* : nilai yang berupa teks atau sekumpulan huruf.
- *Number* : nilai yang berupa angka.

- *Boolean* : nilai *true/false*.
- *Array* : struktur untuk menyimpan lebih dari 1 nilai dalam sebuah *reference*
- *Object* : semua yang ada pada *Javascript* termasuk objek pada HTML.

2.3.2 *Constant*

Constant adalah sebuah variabel *read-only*, artinya nilai pada *constant* tidak dapat diubah. Untuk mendeklarasikan *constant*, digunakan keyword '*const*'. Pada *listing 2.4* terdapat potongan kode untuk mendeklarasi *constant*.

```
const myConst = 1;
```

Listing 2.4: Deklarasi *constant*

2.3.3 *Function*

Function adalah sekumpulan perintah/*statements* untuk menjalankan suatu tugas atau menghitung nilai. Untuk membuat *function*, digunakan keyword '*function*', kemudian diikuti dengan nama *function* tersebut, parameter yang dituliskan di dalam kurung, dan *statement*/perintah *Javascript* yang ditulis di dalam kurung kurawal. Parameter pada *function* bisa lebih dari 1 yang penulisanya dipisahkan oleh tanda koma (,). *Function* bisa memiliki parameter atau tidak. Pada *listing 2.5* terdapat potongan kode untuk membuat *function* penjumlahan 2 buah bilangan.

```
function penjumlahan(angka1,angka2){  
    var hasil = angka1+angka2;  
    return hasil;  
}
```

Listing 2.5: *Function* penjumlahan 2 buah bilangan

Setelah membuat *function*, *function* tersebut tidak langsung dieksekusi. Membuat *function* hanya memberi nama *function* tersebut dan mendeskripsikan apa yang akan dilakukan oleh *function* tersebut apabila dipanggil. Dengan memanggil *function*, maka *function* akan dieksekusi. Pada *listing 2.6* terdapat potongan kode untuk memanggil *function* dengan nama penjumlahan.

```
penjumlahan(10,5);
```

Listing 2.6: Memanggil *function* penjumlahan

2.3.4 Menggambar pada *Canvas*

Sesudah menuliskan tag `<canvas>` pada HTML, *canvas* tidak bisa langsung digambar. Karena itu perlu ditambahkan *drawing context* pada *Javascript*. Pada *listing 2.7* terdapat potongan kode untuk menambahkan *drawing context*.

```

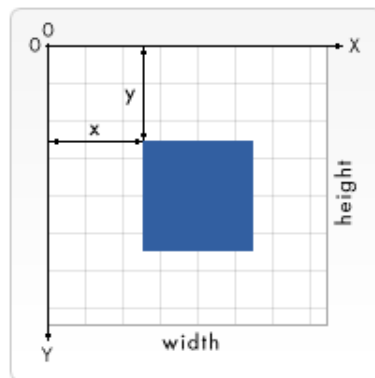
11 |   var myCanvas = document.getElementById('canvas');
22 |   var context = myCanvas.getContext('2d');

```

Listing 2.7: Menambahkan *drawing context canvas*

Berdasarkan *listing 2.7*, variabel `myCanvas` menyimpan objek dengan id = 'canvas'. Id ini mengacu ke objek *canvas* pada HTML yang memiliki id bernama *canvas*. Variabel `myCanvas` sekarang sudah menyimpan objek *canvas*. Kemudian variabel `context` menyimpan *drawing context* 2D. Sesudah itu, *canvas* tersebut dapat digambar dengan bentuk 2D, garis, kurva, membuat tulisan, dan menambahkan gambar. Selain untuk menggambar, bentuk-bentuk tersebut dapat diberi warna sesuai dengan keinginan.

Untuk menggambar bentuk 2D atau garis, diperlukan koordinat x dan y. Koordinat tersebut akan menempatkan gambar tersebut pada *canvas*. Posisi awal/*origin* pada *canvas* adalah (0,0) yang terletak di ujung kiri atas *canvas*. Gambar 2.3 adalah penempatan kotak biru pada *canvas* terhadap *origin*.

Gambar 2.3: Posisi kotak biru pada *canvas* terhadap *origin*[2]

Pada di atas, titik ujung kiri kotak biru tersebut berjarak *x pixel* dari kiri dan berjarak *y pixel* dari atas.

Menggambar Persegi Panjang

Ada 3 cara untuk menggambar persegi panjang:

- `fillRect(x,y,width,height)` : menggambar persegi panjang serta mengisi bagian tengah persegi panjang.
- `strokeRect(x,y,width,height)` : menggambar *outline* yang berbentuk persegi panjang.
- `clearRect(x,y,width,height)` : menghapus daerah yang ditentukan pada *canvas*. Daerah yang dihapus berbentuk persegi panjang.
- `rect(x,y,width,height)` : menambah *path* berbentuk persegi panjang.

Fungsi tersebut memiliki parameter yang sama. Parameter x dan y untuk menentukan posisi pada canvas dari titik ujung kiri atas persegi panjang. *Width* adalah lebar dari persegi panjang dan *height* adalah tinggi dari persegi panjang.

Menggambar *Path*

Path adalah sekumpulan titik yang dihubungkan oleh segmen garis. *Path* dapat membentuk kurva dan membuat bentuk 2D lainnya seperti segitiga, trapesium, belah ketupat dan lain-lain. Langkah-langkah untuk membuat bentuk menggunakan *path* adalah sebagai berikut :

1. Buat *path*.
2. Tuliskan perintah untuk menggambar pada *path* tersebut.
3. Sesudah *path* tersebut sudah dibuat, *path* tersebut dapat dirender menggunakan *stroke* atau *fill*.

Langkah pertama untuk membuat *path* baru adalah dengan menggunakan fungsi *beginPath()*. Setelah itu, perintah-perintah untuk menggambar dapat digunakan untuk membuat bentuk-bentuk yang diinginkan. Apabila sudah selesai menggambar, gunakan fungsi *stroke()* untuk menggambar outline dari *path* tersebut atau *fill()* untuk mengisi area *path* tersebut. Setelah itu, gunakan fungsi *closePath()* untuk menutup bentuk tersebut dengan cara menggambar garis lurus dari posisi titik terakhir ke titik awal. Fungsi lainnya yang menjadi bagian dari membuat *path* adalah fungsi *moveTo()*. Fungsi ini diibaratkan seperti mengangkat sebuah pensil dari sebuah titik pada kertas kemudian menempatkannya pada titik yang diinginkan. Listing 2.8 merupakan fungsi *moveTo()*.

```
moveTo(x,y);
```

Listing 2.8: Fungsi *moveTo()*

Fungsi *moveTo()* memiliki 2 parameter yaitu x dan y yang merupakan posisi titik pada *canvas*. Ketika *canvas* sudah diinisialisasi dan fungsi *beginPath()* sudah dipanggil, fungsi *moveTo()* berguna sebagai penempatan titik awal untuk menggambar. Fungsi *lineTo()* digunakan untuk menggambar sebuah garis. Listing 2.9 merupakan fungsi *lineTo()*.

```
lineTo(x,y);
```

Listing 2.9: Fungsi *lineTo()*

Fungsi *lineTo()* memiliki 2 parameter yaitu x dan y yang merupakan titik akhir dari garis. Garis akan digambar mulai dari posisi titik awal sampai ke posisi titik akhir garis. Titik awal ini bergantung pada titik akhir dari *path* sebelumnya. Titik awal dapat diubah dengan menggunakan fungsi *moveTo()*.

Fungsi *arc()* digunakan untuk menggambar lingkaran atau busur. Listing 2.10 merupakan fungsi *arc()*.

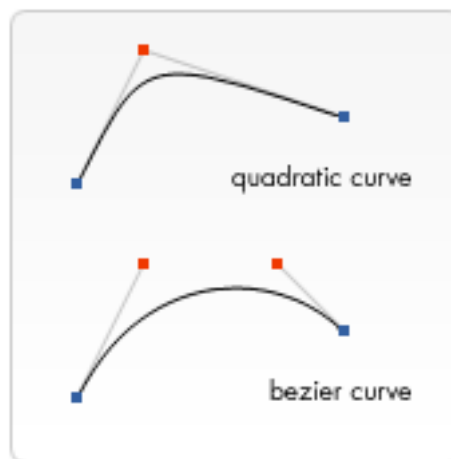

```
11 | arc(x,y,radius,startAngle,endAngle,anticlockwise);
```

Listing 2.10: Fungsi *arc()*

2 Parameter *x* dan *y* adalah posisi titik tengah busur pada *canvas*. Radius adalah besar jari-jari busur. *StartAngle* dan *endAngle* adalah titik awal dan titik akhir busur dalam satuan radian yang diukur dari sumbu *x*. *Anticlockwise* adalah parameter yang bernilai *boolean*, apabila bernilai *true*, maka busur akan digambar berlawanan arah jarum jam dan jika bernilai *false*, busur akan digambar searah jarum jam. Karena fungsi *arc()* menerima input sudut dalam radian, maka perlu dilakukan konversi dari satuan derajat menjadi radian terlebih dahulu. Rumusnya adalah sebagai berikut :

$$\text{radian} = (\text{Math.PI}/180) * \text{besarsudut}$$

8 *Bézier curve* merupakan tipe *path* yang digunakan untuk membuat kurva. *Bézier curve* ada 2 jenis yaitu *cubic* dan *quadratic*. Perbedaannya adalah *quadratic Bézier curve* memiliki sebuah *control point*, sedangkan *cubic Bézier curve* memiliki 2 buah *control point*. Pada Gambar 2.4 menunjukkan perbedaan antara *quadratic Bézier curve* dan *cubic Bézier curve*. Titik merah pada gambar merupakan *control point* dari *Bézier curve*.

Gambar 2.4: Perbedaan *quadratic Bézier curve* dan *cubic Bézier curve*^[2]

13 Berikut adalah fungsi *quadratic* dan *cubic Bézier curve* :

- 14 • *quadraticCurveTo(cp1,cp2,x,y)* : menggambar *quadratic Bézier curve* dari posisi pensil sekarang ke titik akhir yaitu *x* dan *y*, dengan titik control point yaitu *cp1* dan *cp2*.
- 16 • *bezierCurveTo(cp1x,cp1y,cp2x,cp2y,x,y)* : menggambar *cubic Bézier curve* dari posisi pensil sekarang ke titik akhir yaitu *x* dan *y*, dengan 2 titik control point yaitu (*cp1x,cp1y*) dan (*cp2x,cp2y*).

19 2.3.5 Object Oriented Programming Javascript

20 OOP (*Object Oriented Programming*) adalah sebuah paradigma *programming* yang menggunakan
21 abstraksi untuk membuat objek-objek yang ada pada dunia nyata. Bahasa pemrograman seperti

1 *Java*, C++, *Ruby*, *Phyton*, PHP, dan *Objective-C* sudah mendukung OOP. Dalam OOP, setiap
2 objek dapat menerima pesan, memproses data dan mengirim pesan ke objek lain. Program yang
3 menggunakan konsep OOP ini mudah untuk dimengerti dan lebih mudah untuk dikembangkan oleh
4 *programmer*.

5
6 Ide umum pada OOP adalah menggunakan objek untuk memodelkan benda-benda yang ada
7 pada dunia nyata. Objek tersebut kemudian direpresentasi pada program yang dibuat. Objek-objek
8 dapat berisi data, fungsionalitas dan *behaviour* yang merepresentasikan informasi tentang objek
9 tersebut dan tugas objek. Contohnya, bila ingin membuat objek sebuah mobil. Mobil memiliki
10 beberapa informasi diantaranya adalah merk mobil, berat mobil, warna mobil dan tahun produksi.
11 Informasi tersebut dapat disebut sebagai properti dari objek. Mobil dapat bergerak maju, berbelok
12 ke kanan, berbelok ke kiri, bergerak mundur dan berhenti. Hal-hal yang dapat dilakukan oleh objek
13 disebut sebagai method dari objek.

14 Kelas

15 *Javascript* tidak memiliki *statement* 'class' yang dapat digunakan pada bahasa pemrograman C++
16 atau *Java*. Untuk membuat kelas, *Javascript* menggunakan *function* sebagai konstruktor untuk
17 kelas. Karena itu, membuat kelas sama dengan membuat *function* pada *Javascript*. Pada *listing*
18 2.11 terdapat potongan kode untuk membuat kelas bernama Mobil.

```
201 | function Mobil(){  
212 |  
223 | }
```

Listing 2.11: Membuat kelas Mobil

23 Objek

24 Untuk membuat instansi baru dari objek, gunakan *statement* 'new' yang nantinya akan disimpan
25 pada variabel. Pada *listing* 2.12 terdapat potongan kode untuk membuat instansi.

```
271 | var mobil1 = new Mobil();
```

Listing 2.12: Membuat *instance* mobil

28 Konstruktor

29 Konstruktor adalah *method* yang ada pada kelas. Konstruktor akan dipanggil ketika pertama kali
30 inisialisasi atau saat instansi baru dari objek dibuat. *Function* pada *Javascript* berfungsi sebagai
31 konstruktor sehingga tidak perlu membuat method konstruktor lagi. Semua aksi yang terdapat
32 pada kelas akan dieksekusi pada saat instansiasi.

1 Properti/Atribut

2 Properti adalah variabel yang terdapat pada kelas. Properti ditulis pada konstruktor kelas sehingga
3 setiap properti pada kelas akan dibuat ketika membuat instansi baru. Untuk membuat properti,
4 gunakan *statement* *'this'*. Cara ini mirip dengan bahasa pemrograman *Java* ketika membuat sebuah
5 properti pada objek. Sintaks untuk mengakses properti di luar kelas adalah : *namaInstansi.properti*.
6 Pada *listing 2.13* terdapat potongan kode untuk mendefinisikan properti pada kelas Mobil pada
7 saat instansiasi.

8

```
91 function Mobil(merkMobil,beratMobil,warnaMobil,tahunProduksi){  
102     this.merkMobil = merkMobil;  
113     this.beratMobil = beratMobil; //satuan dalam kg  
124     this.warnaMobil = warnaMobil;  
135     this.tahunProduksi = tahunProduksi;  
146 }  
157  
168 var mobil1 = new Mobil('Toyota',1000,'Hitam',2010);
```

Listing 2.13: Mendefinisikan properti pada kelas Mobil

17 Method

18 *Method* adalah hal yang dapat dilakukan oleh sebuah objek. Untuk membuat *method*, tuliskan nama
19 *method* terlebih dahulu kemudian *assign* fungsi pada nama *method* tersebut. Untuk memanggil
20 *method* sebuah objek, tuliskan nama objek/kelas terlebih dahulu, kemudian tuliskan nama *method*
21 sesuai dengan yang sudah dibuat beserta tanda kurung. Tanda kurung berisi parameter. Pada
22 *listing 2.14* terdapat potongan kode untuk membuat dan memanggil *method* *bergerakMaju()* pada
23 kelas Mobil.

24

```
251 function Mobil(merkMobil,beratMobil,warnaMobil,tahunProduksi){  
262     this.merkMobil = merkMobil;  
273     this.beratMobil = beratMobil; //satuan dalam kg  
284     this.warnaMobil = warnaMobil;  
295     this.tahunProduksi = tahunProduksi;  
306  
317     this.bergerakMaju = function(){  
328         //kode agar mobil bergerak maju  
339     }  
340 }  
341  
342 var mobil1 = new Mobil('Toyota',1000,'Hitam',2010);  
343 mobil1.bergerakMaju(); //memanggil fungsi untuk bergerak maju
```

Listing 2.14: Membuat dan memanggil *method* *bergerakMaju()*

2.3.6 *Event*

Event adalah kejadian/peristiwa yang terjadi pada sistem yang diprogram. Sistem akan memberitahu apabila kejadian tersebut sudah terjadi dan akan melakukan suatu aksi ketika kejadian sudah terjadi. Misalnya, di bandara ketika landasan pacu sudah bersih untuk pesawat lepas landas, sinyal akan dikomunikasikan kepada pilot bahwa pesawat sudah boleh untuk lepas landas. Dalam *web*, *event* ditenbak di dalam *browser window* dan dikaitkan pada objek yang spesifik seperti sekumpulan elemen, dokumen HTML yang dimuat atau keseluruhan *browser window*. Ada beberapa *event* yang dapat terjadi diantaranya adalah :

- Pengguna mengklik sebuah element atau mengarahkan kursor ke sebuah elemen.
- Pengguna menekan sebuah tombol pada *keyboard*.
- Pengguna mengatur besar dan menutup *browser window*.
- Halaman *web* selesai dimuat.
- *Form* sedang *disubmit*.
- Video sedang dimainkan, dijeda, atau selesai.
- Ketika *error* terjadi.

Setiap *event* memiliki *event handler*, yang berisikan sekumpulan kode yang akan dijalankan ketika *event* sudah terjadi. *Event handler* juga sering disebut sebagai *event listener*. *Listener* menunggu *event* yang terjadi dan *handler* adalah kode yang dijalankan ketika *listener* mendapatkan *event*/ketika *event* terjadi. Untuk memperjelas bagaimana cara menggunakan *event*, pada [listing 2.15](#) terdapat contoh kode untuk menambahkan event pada button/tombol.

```
<html>
  <title>Event pada tombol</title>
  <body>
    <button id='tombol'>Change color</button>
  </body>
</html>

<script>
  var btn = document.getElementById('tombol');

  function random(number) {
    return Math.floor(Math.random()*(number+1));
  }

  btn.onclick = function() {
    var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' +
      random(255) + ')';
    document.body.style.backgroundColor = rndCol;
  }
}
```

```

18 |     }
19 | </script>

```

Listing 2.15: Menambahkan event pada button

Berdasarkan kode di atas, objek *button* dengan id='tombol' disimpan di dalam variabel bernama 'btn'. Ada fungsi bernama 'random' untuk mengembalikan sebuah nilai acak. Setelah itu ada *event handler*. *Event handler property* yang digunakan adalah *onclick*. *Event handler property onclick* mengecek apakah objek (dalam kasus ini objeknya adalah button) sudah ditekan/diklik. Bila tombol sudah diklik, maka akan mengeksekusi fungsi untuk mengubah warna *background*. Warna RGB tersebut digenerate secara acak menggunakan fungsi *random* yang sudah dibuat sebelumnya. Tidak hanya *event handler property onclick* saja yang dapat digunakan pada halaman web. Berikut ini adalah beberapa *event handler property* lainnya:

- *onfocus* dan *onblur* : event akan terjadi apabila sebuah objek difokuskan/tidak. Biasanya digunakan untuk menampilkan informasi tentang bagaimana cara mengisi *form* ketika difokuskan atau menampilkan pesan *error* ketika *form* tersebut diisi dengan nilai yang salah/tidak valid.
- *ondblclick* : event akan terjadi ketika objek diklik 2 kali/*double click*.
- *window.keypress*, *window.onkeydown*, *window.onkeyup* : event akan terjadi apabila sebuah tombol pada *keyboard* ditekan. *Keypress* adalah event ketika tombol ditekan kemudian dilepas. *Keydown* adalah event ketika tombol ditekan dan *keyup* adalah event ketika tombol dalam keadaan tidak ditekan. Untuk ketiga event ini, event tersebut harus diregister pada objek *window* yang merepresentasikan *browser window*.
- *onmouseover* dan *onmouseout* : event akan terjadi ketika posisi kursor *mouse* berada luar objek lalu ditempatkan di atas objek dan ketika posisi kursor *mouse* berada di atas objek lalu keluar dari objek.

Beberapa *event handler property* tersebut sangat umum dan tersedia di manapun, sedangkan beberapa *event handler property* lainnya sangat spesifik dan hanya digunakan untuk elemen tertentu, contohnya adalah menggunakan *onplay* untuk elemen tertentu yaitu <video>.

Mekanisme event terbaru dalam spesifikasi DOM (*Document Object Model*) *level 2 Events* yang memberikan *browser* sebuah fungsi baru yaitu *addEventListener()*. Fungsi ini mirip seperti *event handler property* namun memiliki sintaks yang berbeda. Pada *listing 2.16* terdapat potongan kode untuk menggunakan fungsi *addEventListener()*.

```

331 |
342 | var btn = document.getElementById('tombol');
353 |
364 | function bgChange() {
375 |     var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random
38 |         (255) + ')';
396 |     document.body.style.backgroundColor = rndCol;

```

```

17 |     }
28 |
39 |     btn.addEventListener('click', bgChange);

```

Listing 2.16: Menggunakan fungsi `addEventListener()`

Pada fungsi `addEventListener()`, ada 2 buah parameter yaitu *event* yang ingin digunakan (dalam potongan kode di atas menggunakan *event click*) dan kode sebagai *handler* yang ingin dijalankan ketika *event* tersebut terjadi. Selain cara di atas, dapat juga menuliskan semua kode di dalam fungsi `addEventListener()` seperti potongan kode pada *listing 2.17*.

```

8 |
91 |
102 | btn.addEventListener('click', function() {
113 |     var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random
12 |     (255) + ')';
134 |     document.body.style.backgroundColor = rndCol;
145 | });

```

Listing 2.17: Menuliskan kode di dalam fungsi `addEventListener()`

2.3.7 Membuat Animasi

Ketika menggambar sebuah bentuk pada *canvas*, bentuk tersebut tidak berpindah tempat. Agar bentuk dapat bergerak, bentuk tersebut harus digambar ulang dan semua yang sudah digambar sebelumnya. Langkah-langkah untuk membuat animasi adalah sebagai berikut :

1. Membersihkan *canvas* : hilangkan semua bentuk-bentuk yang sudah tergambar di *canvas*. Untuk menghapus keseluruhan *canvas*, gunakan fungsi `clearRect()`.
2. Menyimpan *state canvas* : ketika mengubah atribut (seperti *style*) yang mempengaruhi *state canvas* dan ingin *original state* tersebut digunakan kembali, *state* tersebut harus disimpan.
3. Gambar bentuk : gambar bentuk yang ingin dianimasikan.
4. Mengembalikan *state canvas* : jika *state* sudah disimpan, kembalikan *state* tersebut sebelum menggambar di *frame* yang baru.

Bentuk yang digambar pada *canvas* dapat menggunakan fungsi yang dimiliki oleh *canvas* atau dengan membuat fungsi sendiri. Hasil yang ada pada *canvas* akan muncul setelah *script* selesai dieksekusi. Jadi dibutuhkan cara mengeksekusi fungsi untuk menggambar dalam waktu tertentu. Ada 3 fungsi yang dapat digunakan untuk memanggil fungsi dalam kurun waktu tertentu diantaranya adalah :

- `setInterval(function, delay)`: mengeksekusi fungsi *function* berulang kali setiap *delay* milidetik.
- `setTimeout(function, delay)`: mengeksekusi fungsi *function* setiap *delay* milidetik.
- `requestAnimationFrame(callback)`: memberitahu *browser* untuk menjalankan animasi dan meminta *browser* memanggil fungsi yang spesifik untuk memperbarui animasi.

Jika tidak ingin ada interaksi user, gunakan fungsi *setInterval()* untuk mengeksekusi fungsi berulang kali. Bila ingin ada interaksi user, terutama dalam pembuatan *game* yang membutuhkan input *keyboard* atau *mouse* untuk mengontrol animasi, gunakan fungsi *setTimeout()*.

2.4 *jQuery*

jQuery merupakan sebuah *file Javascript* yang dimasukkan pada halaman *web* [3]. *jQuery* dapat memilih elemen pada halaman *web* menggunakan *CSS-style selector* dan elemen tersebut dapat melakukan sesuatu dengan menggunakan *method jQuery*. Sebuah fungsi yaitu *jQuery()* digunakan untuk menemukan satu atau lebih elemen yang berada pada halaman *web*. Fungsi ini membuat objek yang bernama *jQuery* untuk menyimpan referensi dari elemen yang akan dipilih. *\$()* sering digunakan sebagai pengganti fungsi *jQuery()* dikarenakan penulisanya yang pendek. Fungsi *jQuery()* hanya memiliki sebuah parameter yaitu sebuah *selector*.

```
$('li.hot');
```

Listing 2.18: Mendapatkan elemen menggunakan *CSS-style selector*

Pada *listing 2.18*, *selector* akan mencari elemen `` yaitu *list* yang merupakan bagian dari kelas 'hot'. *jQuery* memiliki banyak *method* yang dapat digunakan oleh elemen yang sudah dipilih menggunakan *selector*. *Method* ini merepresentasikan tugas yang akan dilakukan oleh elemen tersebut. Setelah memilih elemen, tambahkan *method* yang diawali dengan titik kemudian diikuti dengan nama *method* beserta parameternya. Titik ini disebut sebagai *member operator*. Setiap *method* memiliki parameter untuk memberikan detail tentang bagaimana cara untuk mengubah elemen tersebut. Ada beberapa *method* yang memiliki parameter lebih dari 1. Member operator menunjukkan bahwa *method* yang terletak setelah *member operator* digunakan untuk mengubah elemen objek *jQuery* yang terletak pada sebelah kiri *member operator*. Pada *listing 2.19*, terdapat contoh untuk mengubah kelas dari elemen yang sudah dipilih. *Method addClass* digunakan untuk mengubah atribut kelas dari elemen *list* menjadi kelas yang bernama 'complete'.

```
$('li.hot').addClass('complete');
```

Listing 2.19: Mengubah kelas dari elemen yang sudah dipilih

Ketika membuat sebuah *jQuery selection*, objek *jQuery* akan menyimpan referensi elemen pada DOM yang dipilih. Maksud dari menyimpan referensi adalah objek *jQuery* menyimpan lokasi elemen tersebut di memori *browser*. Membuat objek *jQuery* membutuhkan beberapa langkah yaitu:

1. Menemukan *node-node* yang sesuai di DOM *tree*
2. Membuat objek *jQuery*
3. Menyimpan referensi di *node* objek *jQuery*

Jika ingin menggunakan *selection* yang sama, maka lebih baik menggunakan objek *jQuery* yang sama daripada mengulang langkah-langkah yang sudah dijelaskan. Objek *jQuery* tersebut

dapat disimpan pada sebuah variabel. Cara untuk menyimpan referensi objek *jQuery* tersebut pada variabel adalah membuat variabel yang diawali dengan simbol '\$'. Kemudian variabel tersebut diisi dengan objek *jQuery* yang diinginkan. Pada listing 2.20, variabel `$listItems` menyimpan objek *jQuery* yang berisi lokasi dari semua elemen list ada pada DOM tree.

```
61 | $listItems = $('li');
```

Listing 2.20: Menyimpan objek jQuery

Untuk mengecek apakah sebuah halaman sudah siap untuk menjalankan kode program yang dibuat, maka gunakan *method ready()*. Maksud dari halaman sudah siap adalah DOM sudah ada pada halaman web. Listing 2.21 adalah potongan kode untuk mengecek apakah halaman sudah siap. Pada listing 2.21, `$(document)` merepresentasikan halaman web. Jika halaman sudah siap, maka kode program yang ada di dalam *method ready()* akan dijalankan. Listing 2.22 adalah pintasan dari *method ready()* pada objek *document*.

```
141 | $(document).ready(function(){
152 |     //ketikan script di sini
163 | });
```

Listing 2.21: Mengecek apakah halaman sudah siap

```
181 | $(function(){
192 |     //ketikan script di sini
203 | });
```

Listing 2.22: Pintasan dari *method \$(document).ready()*

2.4.1 Mendapatkan dan Mengubah Konten Elemen

Untuk mendapatkan konten dari elemen, dapat digunakan 2 *method* yaitu *method html()* dan *method text()*. *Method html()* berfungsi untuk mendapatkan HTML yang sesuai dengan elemen pertama yang sesuai dengan *jQuery selection*. *Method .text()* berfungsi untuk mendapatkan *text*/tulisan dari semua elemen yang sesuai dengan *jQuery selection*. Untuk mengganti konten dari semua elemen terdapat 4 *method* yaitu:

1. *html()* : *method* ini memberikan konten HTML yang baru kepada semua elemen yang sesuai dengan *jQuery selection*.
2. *text()* : *method* ini memberikan *text*/tulisan yang baru kepada setiap elemen yang sesuai dengan *jQuery selection*.
3. *replaceWith()* : *method* ini menggantikan konten setiap elemen yang sesuai dengan *jQuery selection* dengan konten yang baru. *Method* ini juga mengembalikan elemen-elemen yang sudah diganti.
4. *remove()* : *method* ini akan membuang semua elemen yang sesuai dengan *jQuery selection*.

Selain mengubah konten dari elemen, atribut dari elemen yang dipilih dapat diakses dan diubah dengan menggunakan 4 *method*, diantaranya adalah :

1. `attr()` : *method* ini berfungsi untuk mendapatkan atau mengubah atribut secara spesifik dan isi dari atribut.
2. `removeAttr()` : *method* ini berfungsi untuk membuang atribut dari sebuah elemen.
3. `addClass()` : *method* ini berfungsi untuk menambah nilai dari atribut *class*. *Method* ini tidak menggantikan nilai atribut yang sudah ada.
4. `removeClass()` : *method* ini berfungsi untuk membuang nilai dari atribut *class*. *Method* ini tidak membuang nilai atribut kelas lainnya.

2.4.2 Mendapatkan dan Mengubah Properti CSS

Method untuk mengubah dan mendapatkan properti CSS adalah *method* `css()`. Untuk mendapatkan nilai properti CSS, tentukan nama properti yang ingin didapat pada *method* CSS. Apabila pada hasil *selection* memiliki elemen lebih dari 1, maka hasil yang dikembalikan adalah nilai properti CSS dari elemen pertama. *Listing 2.23* merupakan cara untuk mendapatkan nilai background color dari elemen *list* pertama dan akan disimpan pada variabel bernama `'backgroundColor'`. Hasil dari warna tersebut akan dikembalikan dalam nilai RGB.

```
var backgroundColor = $('li').css('background-color');
```

Listing 2.23: Mendapatkan nilai warna *background color* dari elemen *list* pertama

Untuk mengubah nilai properti CSS, tentukan nama properti sebagai argumen pertama dan tentukan nilai untuk properti yang sudah dipilih pada argumen pertama sebagai argumen kedua. Antara argumen pertama dan kedua dipisahkan dengan koma (,). *Method* ini akan mengubah semua elemen yang sesuai dengan *selection*. Dengan *object literal notation*, kita dapat mengubah sejumlah properti lainnya dalam *method* yang sama. Ada 3 cara penulisan pada *object literal notation*, properti dan nilai properti dituliskan di dalam kurung kurawal, antara properti dan nilainya dipisahkan dengan titik dua (:), dan koma(,) memisahkan setiap pasangan properti. *Listing 2.24* merupakan cara untuk background color semua elemen *list* dan *listing 2.25* merupakan cara mengubah sejumlah properti menggunakan *object literal notation*.

```
$('li').css('background-color', 'red');
```

Listing 2.24: Mengubah warna background color semua elemen *list*

```
$('li').css({
    'background-color': 'red',
    'font-family': 'Courier'
});
```

Listing 2.25: Mengubah warna background color dan jenis font untuk semua elemen *list*

2.4.3 Looping

Pada *jQuery* dapat dilakukan *looping* untuk mendapatkan informasi dari setiap elemen atau untuk memberikan aksi pada setiap elemen. *Method* yang digunakan untuk looping adalah *each()*. *Method* ini akan memberikan sebuah atau lebih aksi pada setiap elemen. *Method* ini memiliki sebuah parameter yaitu sebuah fungsi yang isinya adalah perintah-perintah yang akan dijalankan oleh setiap elemen. Contohnya terdapat pada listing 2.26. Pada listing 2.26 akan dipilih elemen *list*. *Method* *.each()* akan menjalankan kode program yang sama untuk setiap elemen *list* tersebut. *'this.id'* mengacu kepada id milik sebuah elemen *list* yang sekarang berada dalam *loop*. Kemudian variabel yang bernama *ids* akan menyimpan id setiap elemen *list*. *\$(this)* digunakan untuk membuat sebuah objek *jQuery* yang baru yang isinya adalah sebuah elemen yang ada sekarang. *\$(this)* memungkinkan kita untuk menggunakan *method* pada elemen yang ada sekarang. Elemen *list* yang ada pada *loop* akan ditambahkan sebuah elemen *span* yang isinya adalah id dari elemen tersebut. Perintah ini akan dilakukan untuk setiap elemen *list*.

```

151    $('li').each(function(){
162        var ids = this.id;
173        $(this).append('<span class="order">'+ids+'</span>');
184    });

```

Listing 2.26: Menambah setiap elemen *list* dengan id *list* masing-masing

2.4.4 Event

Sama seperti *Javascript*, pada *jQuery* juga dapat ditambahkan *event*. *Method* yang digunakan untuk menambahkan *event* adalah *on()*. Untuk memperjelas cara menggunakan *method* *on()*, terdapat potongan kode pada listing 2.27. Untuk menambahkan *event*, maka pertama harus memilih elemen yang akan ditambahkan *event* dengan menggunakan *selector*. Pada listing 2.27, elemen yang dipilih adalah semua elemen *list*, kemudian tambahkan *method* *on()*. *Method* *.on()* memiliki 2 parameter yaitu *event* yang akan digunakan dan perintah yang akan dilakukan apabila *event* tersebut terjadi pada elemen yang dipilih. Perintah dapat berupa sebuah fungsi anonim yaitu fungsi yang dibuat langsung atau memanggil sebuah fungsi yang sudah ada. Pada listing 2.27, *event* yang digunakan adalah *'click'* dan akan diberikan sebuah fungsi anonim yang tugasnya adalah menambahkan atribut *class* yang bernilai *'complete'*.

```

311    $('li').on('click',function(){
322        $(this).addClass('complete');
333    });

```

Listing 2.27: Menambahkan atribut *class* pada setiap *list* menggunakan event *'click'*

Event yang dimiliki *jQuery* cukup banyak. Berikut adalah *event* yang sering digunakan :

- UI : *focus*, *blur*, *change*
- Keyboard : *input*, *keydown*, *keyup*, *keypress*

- *Mouse* : *click, dblclick, mouseup, mousedown, mouseover, mouseout, hover*
- *Form* : *submit, select, change*
- *Document* : *ready, load, unload*
- *Browser* : *error, resize, scroll*

Setiap fungsi *event handling* menerima sebuah *event object*. *Event object* memiliki properti dan *method* yang berhubungan dengan *event* yang sudah terjadi. Contoh kode program dapat dilihat pada *listing 2.28*. Pada *listing 2.28*, pada parameter *function* terdapat parameter yang bernama *event*. Parameter yang bernama *event* ini adalah *event object*. Kemudian tipe dari *event object* tersebut disimpan pada variabel yang bernama *eventType*.

```

$( 'li' ).on( 'click', function(event){
    eventType = event.type;
});

```

Listing 2.28: Mendapatkan tipe *event* dari *event object*

Event object memiliki 7 properti, diantaranya adalah:

- *type* : tipe dari *event* contohnya adalah *click, mouseover*
- *which* : tombol atau key yang sudah ditekan
- *data* : sebuah *object literal* yang mengandung informasi tambahan yang diberikan ke fungsi lain ketika *event* terjadi
- *target* : elemen pada DOM yang memulai *event*
- *pageX* : posisi *mouse* dihitung dari ujung kiri *viewport*
- *pageY* : posisi *mouse* dihitung dari *viewport* paling atas
- *timeStamp* : jumlah milisekon dihitung dari 1 Januari 1970 sampai *event* terjadi

Event object hanya memiliki 2 *method* yaitu *preventDefault()* dan *stopPropagation()*. *Method preventDefault()* akan mencegah pengguna untuk *submit form* dan *method stopPropagation()* akan memberhentikan *event* dari *bubbling* sampai *ancestor*.

2.4.5 AJAX

AJAX(Asynchronous Javascript and XML) adalah sebuah teknik pengembangan *web* yang digunakan untuk memuat data pada bagian halaman *web* tanpa memuat ulang/*refresh* halaman *web*. Penggunaan AJAX yang umum adalah sebagai berikut:

- *Live search/autocomplete* yang dapat ditemukan pada *Google search*.
- Website dengan konten user-generated yang dapat menampilkan informasi pada website, contohnya adalah Twitter dan Flickr

- Menambah barang ke keranjang pada situs belanja online. Barang yang ditambahkan akan diupdate tanpa meninggalkan halaman tersebut.
- Register username pada website yang akan mengecek apakah username sudah digunakan oleh orang lain atau tidak.

AJAX menggunakan *asynchronous processing model* yang artinya adalah pengguna dapat melakukan hal lain ketika *browser* sedang menunggu data untuk dimuat. Ketika halaman *web* sudah dimuat dan jika pengguna ingin mengubah sesuatu pada *browser*, maka pengguna biasanya akan memuat ulang halaman *web*. Hal ini akan membuat pengguna harus menunggu halaman *web* selesai dimuat dan *render* oleh *browser*. Dengan menggunakan AJAX, kita dapat mengubah konten sebuah elemen jika ingin memperbarui sebagian halaman *web*. Caranya adalah dengan menambahkan *event* dan request konten baru ke server menggunakan *asynchronous request*. Ketika data sedang dimuat, maka halaman *web* akan tetap dimuat dan pengguna dapat tetap berinteraksi dengan halaman *web*. Setelah server merespon *request*, *event special* AJAX akan *trigger* bagian lain dari *script* yang membaca data dari server dan *memperbarui* hanya sebuah bagian dari halaman *web*. Hal ini akan membuat data dimuat lebih cepat dan pengguna dapat berinteraksi dengan halaman *web* ketika menunggu dapat untuk dimuat.

Langkah-langkah AJAX mengirim *request* dan menerima respon dari server adalah : Pertama, *browser* meminta informasi dari server. Permintaan tersebut dapat mengandung informasi yang dibutuhkan oleh server untuk diproses. Browser mengimplemen sebuah objek yang bernama *XMLHttpRequest* untuk menangani *Ajax request*. Browser tidak menunggu respon dari server. Setelah mengirim *request* dan server menerima *Ajax request*, server akan mengirimkan HTML atau data dalam format lainnya seperti JSON atau XML. Setelah server selesai merespon *request* tersebut, *browser* akan menjalankan *event*. Event ini dapat digunakan untuk *trigger* fungsi-fungsi *Javascript* yang akan memproses data dan memnambahkannya ke sebuah bagian/elemen dari halaman *web*.

Ajax Request dan Response

Untuk membuat *Ajax request*, *browser* menggunakan objek *XMLHttpRequest*. Ketika server merespon *request* dari browser, objek *XMLHttpRequest* yang sama akan memproses hasilnya. Pada *listing 2.29* terdapat potongan kode untuk membuat *Ajax request*.

```
var xhr = new XMLHttpRequest();
xhr.open('GET', 'data/test.json', true);
xhr.send();
```

Listing 2.29: Membuat *Ajax request*

Berdasarkan *listing 2.29*, hal pertama yang dilakukan adalah membuat objek *XMLHttpRequest* yang disimpan pada variabel bernama *xhr*. Kemudian *method open()* berfungsi untuk menyiapkan request. *Method open* memiliki 3 parameter yaitu HTTP *method*, *url* dari halaman yang akan menangani *request*, dan tipe data *boolean* yang menentukan apakah *request* tersebut *asynchronous* atau tidak. Pada *listing 2.29* *request* tersebut menggunakan HTTP *method* yaitu GET, *url* halamannya adalah *'data/test.json'*, dan *asynchronous*. *Method .send()* digunakan untuk mengirimkan *request*

yang sudah disiapkan. Informasi tambahan dapat dikirimkan ke server yang dituliskan pada parameter method `.send()`. Sesudah mengirimkan request dan menerima respon dari server, data yang diterima akan diproses. Pada *listing 2.30* terdapat potongan kode untuk menerima respon dan memproses data dari server. Setelah *browser* menerima dan memuat respon dari server, event *onload* akan dijalankan dan akan *trigger* sebuah fungsi. Di dalam fungsi tersebut, akan dicek status dari objek tersebut untuk memastikan apakah respon dari server tidak ada masalah.

```

7
81 |     xhr.onload = function(){
82 |         if(xhr.status === 200){
83 |             //proses data yang sudah diterima dari server
84 |         }
85 |     }

```

Listing 2.30: Memproses respon yang didapat dari server

jQuery menyediakan beberapa *method* untuk menangani *Ajax request* diantaranya adalah :

- `load()` : memuat HTML dalam sebuah elemen.
- `$.get()` : memuat data menggunakan method HTTP GET. Method ini digunakan untuk request data dari server.
- `$.post()` : memuat data menggunakan method HTTP POST. Method ini digunakan untuk mengirim data ke server yang mengubah data pada server.
- `$.getJSON()` : memuat data JSON menggunakan GET.
- `$.getScript()` : memuat dan mengeksekusi data pada Javascript menggunakan GET.
- `$.ajax()`: method ini digunakan untuk menjalankan semua request.

Ketika menggunakan method `load()`, HTML yang dikirim dari server dan dimasukkan ke *jQuery selection*. *jQuery* memiliki objek yaitu `jqXHR` yang mempermudah untuk menangani data yang dikirim dari server. Berikut adalah properti dan method dari `jqXHR` :

- `responseText` : mengembalikan data text
- `responseXML` : mengembalikan data XML
- `status` : kode status
- `statusText` : deskripsi dari status
- `done()` : method yang digunakan untuk mengeksekusi kode apabila request berhasil
- `fail()` : method yang digunakan untuk mengeksekusi kode apabila request gagal
- `always()` : method yang digunakan untuk mengeksekusi kode apabila request berhasil atau gagal
- `abort()` : menghentikan komunikasi

Method \$.ajax() memberikan kontrol lebih terhadap Ajax request. Maksud dari memberi kontrol lebih adalah method ini memiliki lebih dari 30 setting yang digunakan untuk mengontrol Ajax request. Semua setting dituliskan menggunakan object literal notation. Berikut adalah setting yang sering dipakai dalam method \$.ajax() :

- type : mendapatkan nilai GET dan POST tergantung dari request. Request tersebut dapat dibuat menggunakan HTTP GET atau POST.
- url : request yang akan dikirimkan
- data : data yang akan dikirimkan ke server bersama dengan request
- success : sebuah fungsi yang akan dijalankan apabila Ajax request berhasil.
- error : sebuah fungsi yang akan dijalankan apabila terdapat error pada Ajax request.
- beforeSend : sebuah fungsi yang akan dijalankan sebelum Ajax request dikirimkan.
- complete : setting yang akan dijalankan setelah event success atau error
- timeout : angka dalam milisekon untuk menunggu sebelum event akan gagal

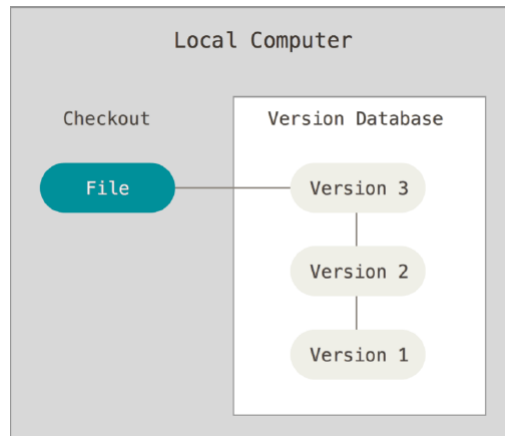
2.5 Git

2.5.1 Version Control

Version control adalah sistem yang menyimpan perubahan pada sebuah *file* atau sekumpulan *file* secara berkala sehingga dapat mendapatkan versi yang spesifik nantinya [4]. VCS (*Version Control System*) memungkinkan pengguna untuk mengembalikan *file* yang diinginkan ke *state* sebelumnya, mengembalikan keseluruhan proyek ke *state* sebelumnya, membandingkan perubahan secara berkala, dapat melihat pengguna terakhir yang memodifikasi sesuatu yang menyebabkan masalah, dan masih banyak lagi. Ketika beberapa *file* ada yang hilang karena sebuah kesalahan, *file-file* tersebut dapat dikembalikan dengan mudah.

Local Version Control System

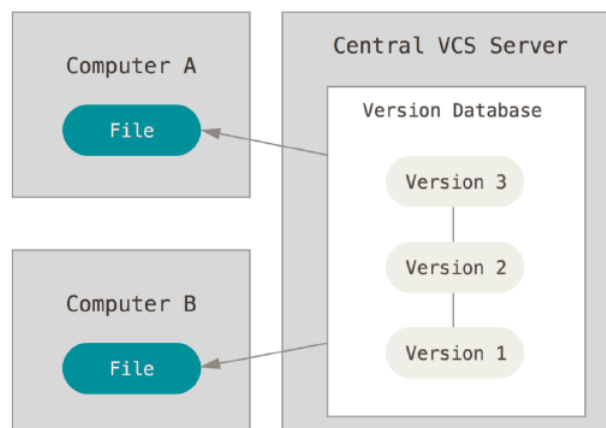
Local Version Control System memiliki sebuah basis data yang menyimpan semua perubahan pada *file* dalam *revision control*. Salah satu VCS tools yang cukup terkenal adalah RCS yang masih digunakan oleh banyak komputer hingga sekarang. Cara kerja RCS adalah menyimpan *patch sets* yang merupakan perbedaan antara beberapa *file* seperti pada Gambar 2.5. *Patch sets* tersebut disimpan di *disk*. RCS dapat menampilkan *file* apa saja pada suatu waktu dengan menggabungkan *patch-patch* tersebut.



Gambar 2.5: Local Version Control

1 Centralized Version Control System

2 *Local Version Control System* menjadi kurang efektif, bila ada beberapa orang yang berkolaborasi
3 dengan pengembang. Karena pada *Local Version Control System*, *version control* dimiliki oleh
4 masing-masing komputer sehingga pengguna tidak tahu apakah *file* tersebut sudah diubah oleh kola-
5 borator lain. CVCS(*Centralized Version Control System*) memiliki sebuah *server* yang menyimpan
6 semua *file* beserta historinya dan jumlah *client* yang mengecek *file* tersebut. Dengan adanya CVCS,
7 semua orang mengetahui apa yang dilakukan oleh kolaborator yang mengerjakan proyek. Tetapi
8 kelemahannya adalah ketika *server* tersebut *down*, tidak akan ada yang bisa berkolaborasi dan tidak
9 dapat menyimpan perubahan yang sudah dikerjakan. Selain itu apabila data di *server* tersebut
10 hilang maka dan tidak melakukan *back-up*, proyek yang sedang dikerjakan akan hilang beserta
11 semua historinya. Struktur CVCS dapat dilihat pada Gambar 2.6.

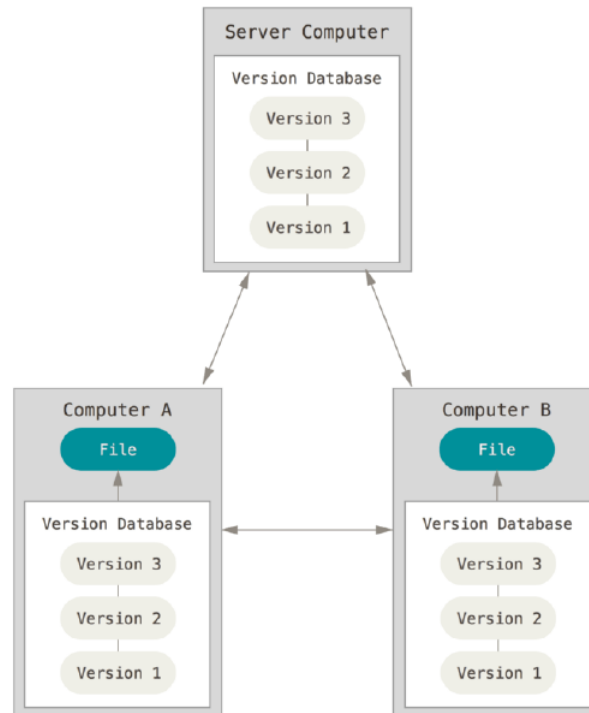


Gambar 2.6: Centralized Version Control

12 Distributed Version Control System

13 Dalam DVCS(*Distributed Version Control System*) seperti *Git*, *Mercurial*, *Bazaar* dan *Darcs*, *client*
14 tidak mengecek versi terbaru dari *file* tetapi *client* menggandakan *repository* termasuk historinya.
15 Jika *server* mati/kehilangan data, maka *client* memiliki *file back-up* untuk mengembalikannya.

1 Ilustrasi DVCS terdapat pada Gambar 2.7.



Gambar 2.7: Distributed Version Control

2 2.5.2 *Git*

3 *Git* merupakan sebuah *version control* namun berbeda dengan VCS lainnya dilihat dari cara me-
 4 nyimpan datanya. Sistem seperti CVS, *Subversion*, *Perforce*, *Bazaar* menyimpan data sebagai
 5 sekumpulan *file* dan perubahan setiap *file* disimpan setiap waktu. Pada *Git*, data tersebut dianggap
 6 sebagai sekumpulan *snapshot* dari *miniature filesystem*. Setiap *commit* atau menyimpan proyek,
 7 *Git* seolah-olah mengambil gambar untuk melihat seperti apa *file* yang terlihat pada saat itu dan
 8 menyimpannya sebagai referensi pada *snapshot* tersebut. Singkatnya, apabila tidak ada *file* yang
 9 diubah, *Git* tidak akan menyimpan *file* lagi.

11 Hampir semua operasi pada *Git* dapat dilakukan secara lokal. Ketika ingin melihat histori
 12 suatu proyek, *Git* akan mengambil data histori tersebut dari basis data lokal, sehingga tidak perlu
 13 memintanya ke *server*. Selain itu, pengguna dapat bekerja secara *offline*. Pada sistem lain seperti
 14 *Perforce*, pengguna tidak dapat melakukan banyak hal jika tidak terkoneksi ke *server* dan pada
 15 CVS, pengguna dapat mengubah *file* tetapi tidak dapat *commit* ke basis data. Pada *Git*, pengguna
 16 dapat *commit* dikarenakan *Git* memiliki basis data lokal.

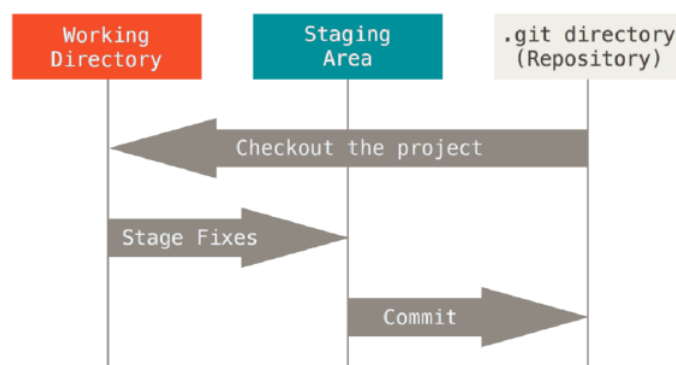
18 *Git* memiliki 3 *state* utama pada *file* yaitu:

- 19 • *committed* : data sudah tersimpan di basis data lokal.
- 20 • *modified* : *file* sudah diubah namun belum di*commit* ke basis data.
- 21 • *staged* : menandai *file* yang sudah dimodifikasi dalam versi sekarang untuk di*commit*.

Terdapat 3 bagian utama dalam proyek *Git* yaitu :

- *Git directory* : tempat untuk menyimpan *metadata* dan objek basis data untuk proyek yang dibuat. Ini adalah bagian terpenting dari *Git* dan inilah yang di-copy ketika *clone repository* dari komputer lain.
- *Working tree* : *single checkout* sebuah versi dari proyek. *File* diambil dari basis data yang sudah di*compressed* di *Git directory* dan disimpan pada *disk* untuk digunakan dan dimodifikasi.
- *Staging area* : sebuah *file* yang ada di *Git directory* yang menyimpan informasi tentang apa yang akan disimpan untuk *commit* selanjutnya.

Gambar 2.8 di bawah ini menunjukkan *working tree*, *staging area* dan *Git directory*.



Gambar 2.8: Working tree, staging area, dan Git directory

Workflow pada *Git* adalah sebagai berikut :

1. Pengguna memodifikasi *file* di *working tree* milik pengguna.
2. Pengguna memilih *file* yang akan menjadi bagian dari *commit* selanjutnya. *File* yang terpilih akan ditambahkan ke *staging area*.
3. Pengguna melakukan *commit file* tersebut yang berada pada *staging area* dan menyimpan *snapshot* secara permanen ke *Git directory*.

Apabila versi tertentu dari sebuah *file* sudah ada pada *Git directory*, maka *file* tersebut dalam berada dalam *state committed*. Jika *file* sudah dimodifikasi dan sudah ditambahkan ke *staging area*, maka *file* tersebut dalam *state staged*. Jika *file* sudah diubah dan sudah di*checkout* tetapi belum dalam *state staged*, maka *file* tersebut dalam *state modified*.

Ada beberapa cara dalam menggunakan *Git* yaitu dengan menggunakan *command-line* dan beberapa GUI(*Graphical User Interface*) yang memiliki kemampuan yang bermacam-macam. Pada umumnya digunakan *command-line*, karena *command-line* dapat menjalankan semua perintah *Git* sedangkan GUI hanya memiliki sebagian fungsionalitas pada *Git* supaya simpel dan mudah digunakan.

1 Mendapatkan *Git Repository*

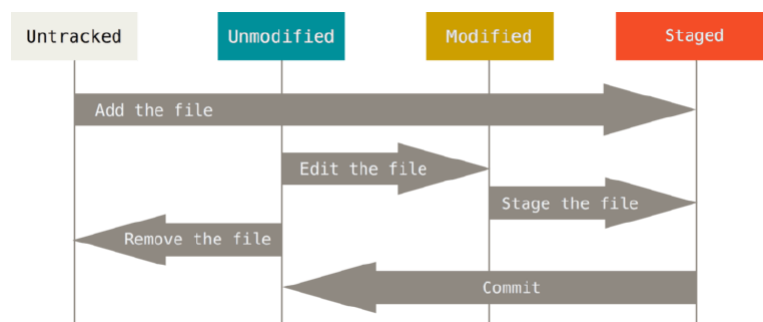
2 Untuk mendapatkan *Git repository* ada 2 cara yaitu : menjadikan sebuah proyek yang terdapat
3 pada direktori lokal yang belum dalam *version control* lalu menjadikannya sebagai *Git repository*
4 dan dengan *clone Git repository* yang sudah ada.

5
6 Jika memiliki direktori proyek yang belum dalam *version control* dan ingin mengontrolnya
7 menggunakan *Git*, hal pertama yang harus dilakukan adalah dengan membuka direktori proyek.
8 Perintah untuk membuat repository pada *Windows* adalah dengan mengetikkan perintah `$ cd /c/u-`
9 `ser/my_project` sesudah itu ketik perintah `$ git init`. Perintah tersebut akan membuat subdirektori
10 bernama `.git` yang mengandung semua repository yang dibutuhkan. Setelah mengetikkan perintah di
11 atas, proyek tersebut belum di-*track* sama sekali. Untuk men-*track file-file* pada sebuah proyek,
12 pertama gunakan perintah `git add` untuk men-*track file* yang diinginkan kemudian ketik `git commit`
13 untuk commit file tersebut.

14
15 *Clone repository* adalah mendapatkan *copy* dari *repository* yang sudah ada. Perintah yang
16 digunakan adalah `git clone`. Tidak hanya *file-file* pada *repository* saja yang di-*copy*, tetapi semua
17 histori pada *repository* tersebut akan ikut ter-*copy*. Perintah `git clone` diikuti dengan *url*. *Url* ini
18 berisi *link* di mana *repository* berada.

19 Record Perubahan pada *Repository*

20 Setiap *file* dalam direktori memiliki 2 *state* yaitu *tracked* atau *untracked*. *Tracked file* adalah *file*
21 yang berada pada *snapshot* terakhir. *Tracked file* adalah *file* yang *Git* ketahui sekarang. *Untracked*
22 *file* adalah *file* yang tidak berada pada *snapshot* terakhir. Ketika *file* diubah, *Git* melihat bahwa
23 *file* tersebut sudah dimodifikasi, karena *file* tersebut diubah setelah *commit* terakhir. Kemudian
24 *file* yang sudah dimodifikasi tersebut di-*stage* dan *commit* semua *file* yang sudah di-*staged* tersebut.
25 Gambar 2.9 menunjukkan siklus hidup dari status *file*.



Gambar 2.9: Siklus hidup pada status file

26 Perintah `git status` digunakan untuk mengecek status *file*. Jika mengetik perintah sesudah
27 *clone*, maka tidak ada *untracked file* karena pada saat *clone*, tidak ada *file* yang dimodifikasi.
28 Bila menambahkan sebuah *file* baru atau mengubah *file* lalu mengetik perintah `git status`, maka
29 akan diberitahukan bahwa terdapat *untracked file*. Karena itu untuk men-*track file* baru, gunakan
30 perintah `git add` yang diikuti dengan nama *filenya* seperti contoh ini : `$ git add README`. Perintah

git add tidak hanya digunakan untuk men-*track file* baru. Selain digunakan untuk men-*track file*, perintah *git add* digunakan untuk stage *file* yang sudah dimodifikasi.

Tidak semua *file* akan ditambahkan secara otomatis oleh *Git* atau ada *file* yang ditunjukan sebagai *file untracked*. Hal ini dapat diatasi dengan membuat sebuah *file* yang bernama *.gitignore*. *File .gitignore* ini berisi *file-file* yang tidak akan di-*track* oleh *Git*. *File* yang biasanya ada dalam *.gitignore* adalah *log*, *tmp* atau *file* dokumentasi yang digenerate secara otomatis. Adapun aturan untuk *pattern* yang dapat dimasukan pada *file .gitignore* diantaranya adalah :

- Baris kosong atau baris yang diawali dengan tanda pagar(#) akan dibiarkan.
- *Standard glob patterns*.
- *Pattern* diawali dengan garis miring(/) untuk mencegah rekursif.
- *Pattern* diakhiri dengan garis miring untuk menspesifikasikan direktori.
- Menegasikan *pattern* diawali dengan tanda seru(!).

Glob pattern adalah *regular expression* yang digunakan oleh *shells*. Tanda bintang(*) untuk nol atau beberapa karakter, [abc] untuk karakter apa saja yang berada di dalam kurung siku, tanda tanya(?) untuk sebuah karakter apa saja dan tanda kurung siku dengan tanda strip(-) untuk karakter antara sebuah karakter dengan karakter lainnya.

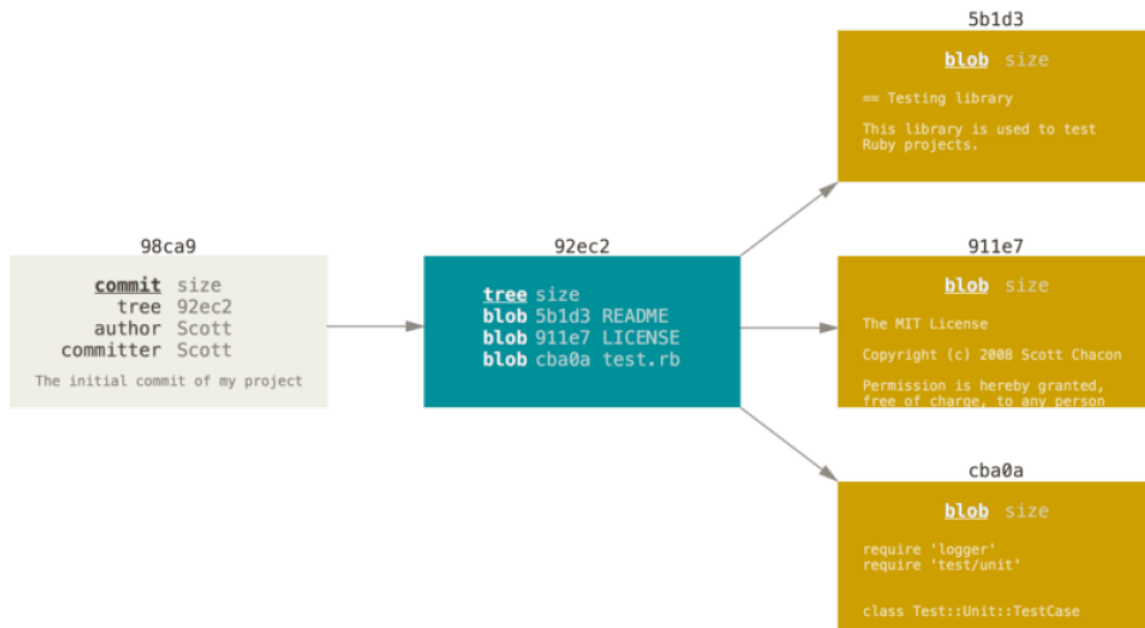
Perintah *git commit* digunakan untuk *commit file* yang sudah diubah dan ditambahkan. *File* tersebut harus sudah di-*stage* dengan menggunakan perintah *git add*. *File* yang belum di-*stage* akan berada dalam state *modified* meskipun sudah melakukan *commit*. Untuk menambahkan keterangan tentang *file* yang di-*commit* dapat dituliskan perintah *git commit -m* yang diikuti dengan keterangan yang ingin disampaikan.

2.5.3 *Git Branching*

Branching artinya membuat dan mengerjakan sebuah proyek di tempat yang berbeda namun masih dalam repository yang sama sehingga tidak mengubah proyek utama. Ketika *commit*, *Git* menyimpan objek *commit* yang memiliki sebuah *pointer* pada *snapshot* sebuah konten yang sudah dalam *state staged*. Objek ini mengandung nama pembuat dan alamat email, pesan yang diketik, dan *pointer* ke *commit*.

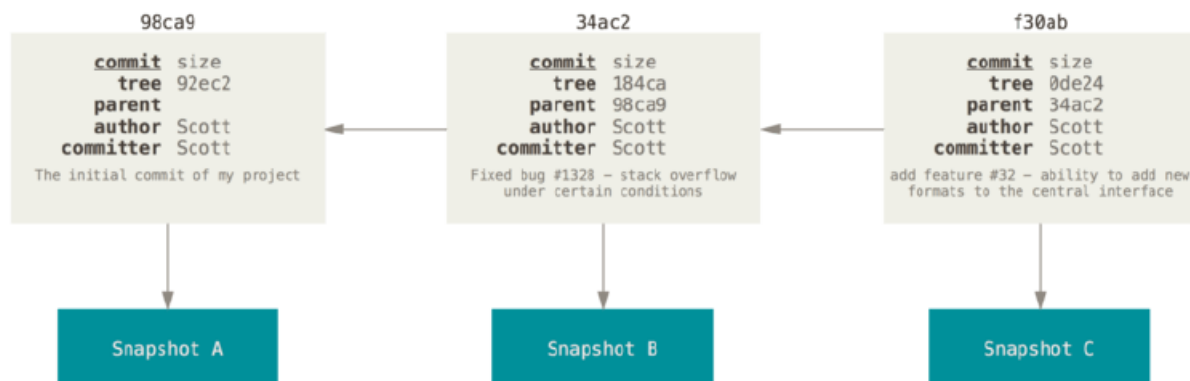
Misalkan seorang pengguna memiliki 3 *file*, kemudian *file* tersebut semuanya di-*stage* dan *commit*. *Staging file* akan mengkomputasi *checksum* untuk setiap *file*, menyimpan versi tersebut pada *Git repository* (hal ini dapat disebut juga sebagai *blobs*), dan menambah *checksum* tersebut ke *staging area*. Lalu *Git* melakukan *checksum* pada setiap *subdirectory* dan menyimpan ketiga objek tersebut pada *Git repository*. Sesudah itu *Git* akan membuat objek *commit* yang mengandung *metadata* dan *pointer* ke proyek *root* sehingga dapat melihat *snapshot* tersebut pada setiap versi. Sekarang, *Git repository* memiliki 5 objek yaitu 3 *blob* yang merepresentasikan 3 *file*, sebuah *tree* yang mengandung isi direktori dan memberi nama *blob* berdasarkan nama *file* yang di-*commit*, dan

- 1 sebuah *commit* dengan *pointer* ke *root tree* dan semua *commit metadata*. Gambar 2.10 merupakan
- 2 *tree* dari penjelasan tersebut.



Gambar 2.10: Commit dan tree dari file yang dicommit

- 3 Jika ada perubahan pada proyek dan *commit* proyek tersebut, maka *commit* sesudahnya
- 4 menyimpan *pointer* pada *commit* sebelum *commit* terbaru seperti yang terdapat pada Gambar 2.11.
- 5 Jadi *parent* dari sebuah *commit* adalah *commit* sebelumnya dan kemudian seterusnya.



Gambar 2.11: Commit dan parent dari commit

- 6 Nama *branch* pada *Git* awalnya disebut *master*. Ketika *commit*, pengguna diberikan *branch*
- 7 *master* yang menunjuk pada *file* yang dicommit terakhir. Setiap *commit*, pointer pada *branch*
- 8 *master* akan terus maju secara otomatis.

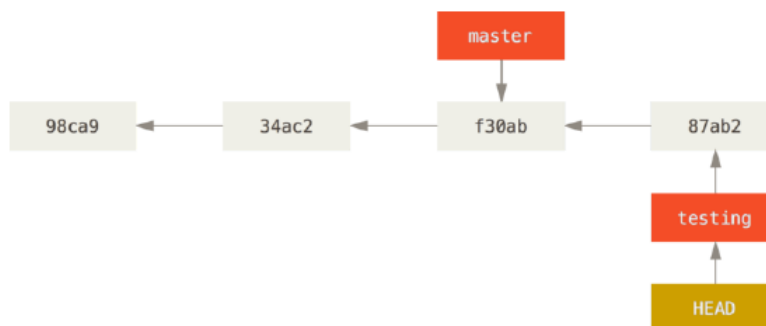
- 9
- 10 Untuk membuat *branch* baru, gunakan perintah *git branch* diikuti dengan nama *branch*. *Git*
- 11 menggunakan *pointer* yang disebut dengan *HEAD* untuk mengetahui bahwa pengguna sedang
- 12 berada dalam *branch* tertentu. Bila membuat *branch* baru, posisi *HEAD* tetap berada pada *branch*

- 1 yang sekarang. Perintah *git branch* hanya membuat *branch* baru dan tidak berpindah ke *branch*
- 2 yang baru saja dibuat. Pada Gambar 2.12, jika mengetikkan perintah *git branch testing*, *branch*
- 3 *testing* akan dibuat tetapi *pointer HEAD* akan tetap berada pada *branch master*.



Gambar 2.12: *Pointer HEAD* menunjuk *branch master*

- 4 Untuk pindah *branch*, gunakan perintah *git checkout* diikuti dengan nama *branch*. *Pointer*
- 5 *HEAD* akan berpindah ke *branch* tersebut. Bila pada *branch* tersebut pengguna melakukan *commit*,
- 6 maka *branch* tersebut akan maju beserta dengan *pointer HEAD* seperti dicontohkan pada Gam-
- 7 bar 2.13. Misalkan pengguna *commit* pada *branch testing*, maka hanya *branch testing* saja yang
- 8 maju sedangkan *branch master* tidak. Ini dikarenakan *file* pada *branch master* tidak diubah.



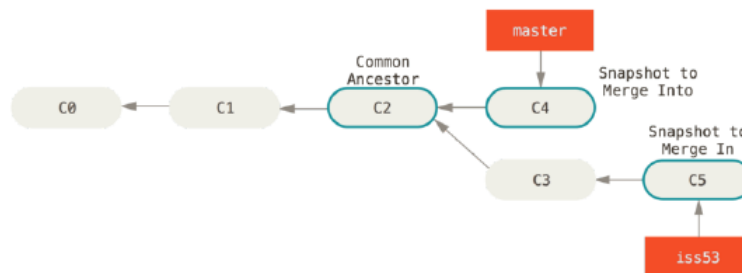
Gambar 2.13: *Pointer HEAD* beserta *branch testing*

- 10 Perintah *git checkout* tidak hanya sebatas untuk pindah ke *branch* yang diinginkan. *File* yang
- 11 ada pada *working directory* akan diubah dengan *file* yang ada pada *branch* tersebut. Bila berpindah
- 12 ke *branch* sebelumnya, maka *file* dalam *working directory* akan dikembalikan sesuai dengan *commit*
- 13 terakhir dari *branch* tersebut. Untuk membuat *branch* baru sekaligus pindah *branch*, gunakan
- 14 perintah *git checkout -b* diikuti dengan nama *branch* yang ingin dibuat. Dengan ini *pointer HEAD*
- 15 akan berada pada *branch* yang baru dibuat.

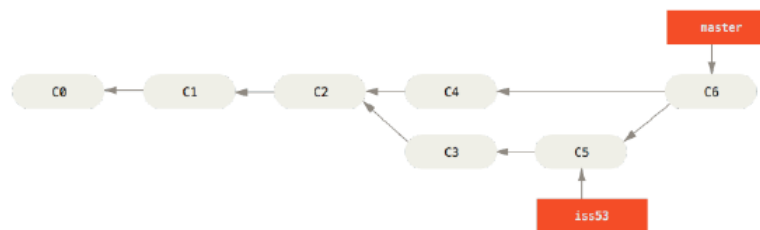
16 Basic Merging

- 17 *Merging* adalah penggabungan sebuah *branch* dengan *branch* lain. Perintah untuk *merge* adalah *git*
- 18 *merge* diikuti dengan nama *branch* yang ingin digabungkan. Bila sebuah *branch* ingin digabungkan

- 1 dengan *branch* yang memiliki *direct ancestor* yang berbeda, *Git* akan melakukan *three way merge*.
 2 *Three way merge* ini menggunakan 2 *snapshot* yang menunjuk pada *branch* yang akan digabungkan
 3 dan 1 *snapshot* yang menunjuk pada *ancestor* yang sama dari kedua *branch* tersebut seperti yang
 4 terdapat pada Gambar 2.14. Kemudian *Git* membuat *snapshot* baru yang merupakan hasil dari
 5 *three way merge* dan secara otomatis akan membuat *commit* yang baru seperti yang terlihat pada
 6 Gambar 2.15. Hal ini disebut sebagai *merge commit* karena memiliki lebih dari 2 *parent*.



Gambar 2.14: 3 *snapshot* yang digunakan dalam *three way merge*



Gambar 2.15: *Merge commit*

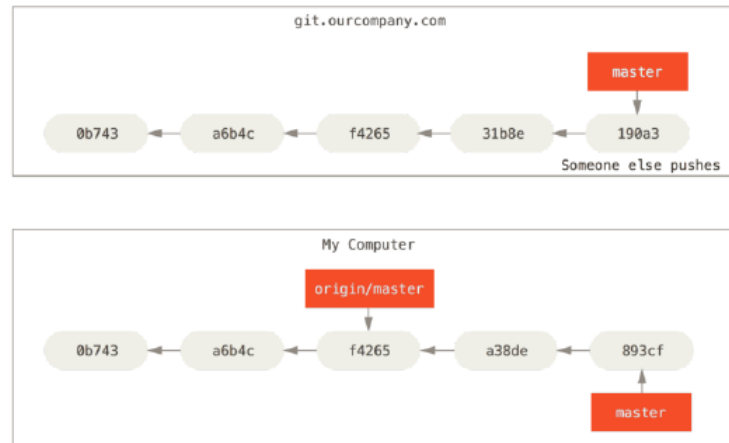
- 7 *Merge* pada *Git* mungkin akan menimbulkan konflik. Hal ini dapat terjadi apabila *file* yang
 8 sama pada kedua *branch* tersebut diubah pada bagian yang sama. Ketika mengetikkan perintah *git*
 9 *merge*, maka *Git* tidak akan membuat *merge commit* secara otomatis. Proses *merge* akan dijeda
 10 sesudah konflik tersebut sudah diselesaikan. Untuk menangani konflik tersebut, pilihlah salah satu
 11 *branch*. Maksud dari memilih salah satu *branch* adalah dengan mengubah *file* yang berada pada
 12 salah satu *branch*. Sesudah mengubah *file* pada *branch* yang dipilih, maka *Git* akan *merge branch*
 13 jika tidak ada konflik lagi.

14 **Remote Branches**

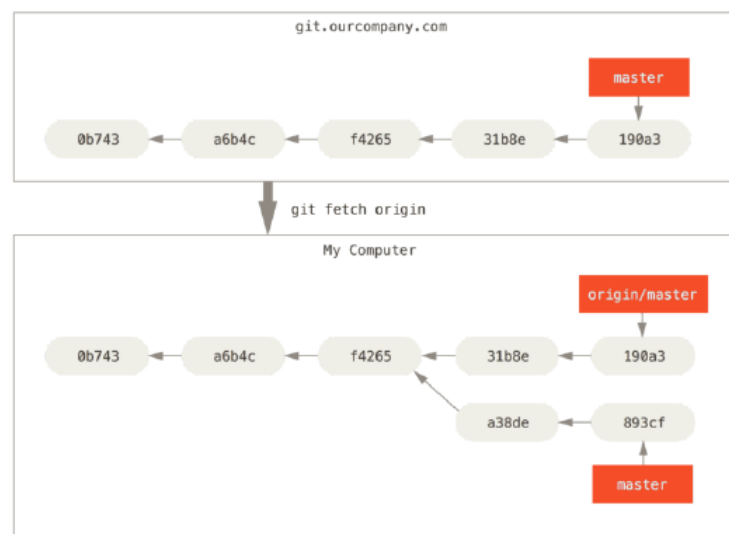
- 15 *Remote-tracking branches* adalah referensi dari *state remote branches*. Referensi tersebut merupakan
 16 referensi lokal yang hanya dapat dipindahkan oleh *Git* untuk memastikan jika referensi tersebut
 17 merepresentasikan *state* dari *remote repository*. `<remote>/<branches>` merupakan *remote-tracking*
 18 *branches*. Jika ingin mengecek *file* pada *branch master* yang berada dalam *remote origin*, maka
 19 pengguna harus mengecek *branch origin/master*. Sama seperti *branch master*, *origin* juga meru-
 20 pakan penamaan *remote* secara otomatis ketika *clone repository*. Jika pengguna mengubah *branch*
 21 lokal maka *branch* milik server tidak akan berubah dan hanya *pointer* pada lokal saja yang berubah.
 22 Maka dari itu *branch* di lokal dan *branch* di *server* bisa saja berbeda seperti yang terlihat pada

- 1 Gambar 2.16 Untuk mensinkron *branch* di lokal dan *branch* di *server*, gunakan perintah *git fetch*
 2 diikuti dengan nama *remote*. Dengan cara ini, beberapa data yang belum dimiliki akan diambil
 3 dari *server*, meng-*update* basis data lokal dan memindahkan *pointer* ke posisi yang terbaru seperti
 4 yang terlihat pada Gambar 2.17.

5



Gambar 2.16: Perbedaan pada *branch* lokal dan *remote*



Gambar 2.17: *Update remote-tracking branches* menggunakan perintah *git fetch*

- 6 Jika ingin membagikan *branch* ke pengguna lain, pengguna harus *push branch* tersebut ke remote
 7 karena *branch* lokal tidak sinkron secara otomatis dengan *remote*. Perintah yang digunakan untuk
 8 *push* adalah *git push* diikuti dengan nama *remote* dan nama *branch*.

9

- 10 *Check out branch* lokal dari *remote-tracking branch* secara otomatis akan membuat *tracking*
 11 *branch*. *Tracking branch* adalah *branch* lokal yang memiliki hubungan langsung dengan *branch*
 12 *remote*. Jika berada pada *tracking branch* dan mengetikkan perintah *git pull*, secara otomatis
 13 *Git* mengetahui *server* mana yang akan di-*fetch* dan *branch* apa yang akan di-*merge*. Bila *clone*
 14 *repository*, maka secara otomatis akan membuat sebuah *branch* yang bernama *master* yang men-*track*

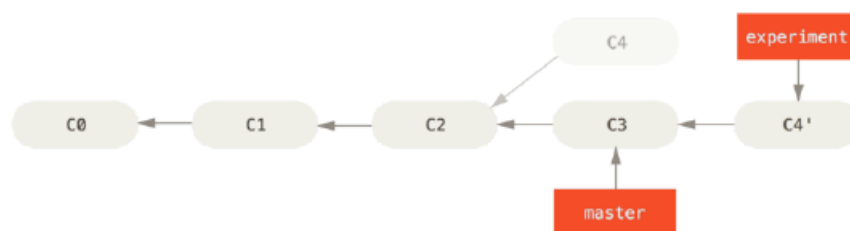
1 *origin/master*. Untuk mengatur *tracking branch*, perintah yang digunakan adalah *git checkout -b*
 2 *<branch> <remote>/<branch>*. *Git* menyediakan perintah *git checkout -track <remote>/<branch>*
 3 sebagai shortcut dari perintah *checkout* sebelumnya. Perintah *git checkout* juga dapat digunakan
 4 untuk mengatur *branch* lokal dengan nama yang berbeda dari *branch remote*. Jika sudah memiliki
 5 *branch* lokal dan ingin mengatur *branch* tersebut ke *branch remote* yang sudah di-*pull*, gunakan
 6 opsi *-u* atau *-set-upstream-to* pada perintah *git branch*. Untuk melihat *tracking branch* yang sudah
 7 diatur, gunakan opsi *-vv* pada perintah *git branch*. Perintah ini akan menampilkan *list* dari *branch*
 8 lokal dengan informasi tambahan mengenai *tracking* pada setiap *branch* dan apakah *branch* lokal
 9 tersebut memiliki *ahead*, *behind* atau keduanya. *Ahead* adalah ada *commit* lokal yang belum di-*push*
 10 ke *server*, sedangkan *behind* adalah *commit* yang belum digabungkan. Perintah ini tidak langsung
 11 mengambil datanya dari *server* tetapi data tersebut merupakan data saat terakhir *fetch* dari *server*.
 12 Untuk mendapatkan data yang terbaru, harus *fetch* dari semua *remote* kemudian mengetikkan
 13 perintah *git branch -vv*.

14 Perintah *git fetch* akan mengambil semua perubahan yang ada pada *server* yang tidak dimiliki
 15 oleh *branch lokal*, tetapi tidak mengubah *working directory* yang sesuai dengan *branch remote*.
 16 Perintah *git pull* digunakan untuk mengubah *working directory*. Perintah ini akan melihat *server*
 17 dan *branch* yang sedang di-*track*, mengambil data dari *server* tersebut dan menggabungkannya.
 18 Singkatnya, perintah *git pull* merupakan gabungan dari perintah *git fetch* dengan *git merge*.

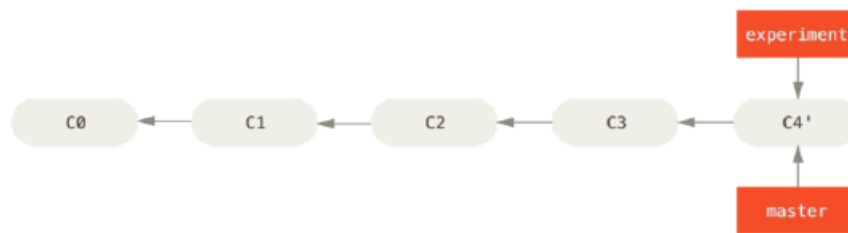
19
 20 *Branch* pada *remote* dapat dihapus dengan menggunakan opsi *-delete* pada perintah *git push*.
 21 *Branch* pada *remote* tidak sepenuhnya dihapus, tetapi hanya *pointernya* saja yang dihilangkan.
 22 Jika *branch* tidak sengaja terhapus, maka data pada *branch* dapat dikembalikan/*di-back-up*.

23 **Rebasing**

24 Selain *merge*, ada cara lain untuk menggabungkan kedua *branch* yaitu *rebasing*. Cara kerja dari
 25 *rebasing* adalah mencari *ancestor* yang sama dari kedua *branch*, mendapatkan perbedaan setiap
 26 *commit* pada *branch* saat ini, menyimpan perbedaan tersebut pada *file* sementara, mengatur
 27 ulang *branch* ke *commit* yang sama dengan *branch* yang akan direbase, dan menerapkan setiap
 28 perubahannya. Contoh *rebasing* dapat dilihat pada Gambar 2.18. *Commit C4* pada *branch*
 29 *experiment* berpindah dari C4 ke C4' yang berada di atas C3. Setelah *rebasing*, *merge* kedua *branch*
 30 tersebut sehingga hasilnya terlihat seperti pada Gambar 2.19. Untuk *rebasing*, gunakan perintah *git*
 31 *rebase* kemudian diikuti nama *branch* yang ingin direbase.



Gambar 2.18: *Rebasing commit C4 ke C3*

Gambar 2.19: *Merge branch setelah rebasing*

Hasil terakhirnya tidak berbeda dengan menggunakan perintah *merge*, namun *rebasing* membuat histori menjadi lebih sedikit dibandingkan dengan *merge*. *Rebasing* juga berguna dalam berkontribusi pada proyek yang bukan milik sendiri. Hal ini akan mempermudah kerja pemilik proyek, karena pemilik proyek hanya tinggal *clean apply* saja.

2.5.4 *GitHub*

GitHub merupakan *single host* terbesar untuk *Git repository* dan sebagai titik tengah dari kolaborasi untuk jutaan pengembang dan proyek. Persentase terbesar dari semua *Git repository* dihosting di *GitHub* dan banyak proyek *open-source* menggunakannya untuk *Git hosting*, *code review*, *issue tracking* dan lainnya.

Fork

Jika pengguna ingin berkontribusi pada proyek yang sudah ada dan pengguna tidak memiliki akses untuk *push*, maka pengguna dapat *fork* proyek tersebut. Ketika proyek tersebut telah di-*fork*, *GitHub* akan membuatkan sebuah *copy/clone* dari proyek tersebut yang sekarang sudah menjadi milik penggunanya dan dapat di-*push*. Orang lain dapat *fork* proyek, *push* proyek, dan berkontribusi dalam perubahan tersebut dan menyarankan untuk menggabungkan perubahan tersebut dengan *repository* aslinya dengan membuat *Pull Request*.

Untuk *fork* proyek, kunjungi halaman proyek dan klik tombol '*Fork*' seperti pada Gambar 2.20 yang berada di atas kanan halaman.

Gambar 2.20: Tombol '*Fork*'

Berikut adalah langkah-langkah untuk berkolaborasi dalam *GitHub*:

1. *Fork* proyek yang diinginkan.
2. Buat topik *branch* dari *master*.
3. Lakukan *commit* untuk memperbaiki proyek.

- 1 4. *Push branch* ke proyek *GitHub*.
- 2 5. Buka *Pull Request* di *GitHub*.
- 3 6. Diskusikan dan *commit* proyek tersebut apabila proyek tersebut masih membutuhkan perba-
4 ikan.
- 5 7. Pemilik proyek *merges*/menggabungkan atau menutup *Pull Request*.

6 ***Pull Request***

7 *Pull Request* membuka tempat diskusi untuk *owner*(*pemilik repository*) dan kontributor sehingga
8 dapat berkomunikasi tentang perubahan tersebut sampai *owner* merasa puas dan senang. Setelah
9 itu *owner* akan *merge*/menggabungkan perubahan tersebut. Untuk membuat *Pull Request*, bukalah
10 halaman '*Branches*' dan buat *Pull Request* baru. Sesudah itu, akan muncul sebuah laman yang
11 meminta mengisi judul dan deskripsi *Pull Request* tersebut. Ketika tombol '*Create pull request*'
12 diklik, maka pemilik proyek akan mendapatkan notifikasi bahwa seseorang menyarankan sebu-
13 ah perubahan dan akan menghubungkan ke sebuah halaman yang memiliki semua informasi tersebut.

14
15 Setelah kontributor sudah membuat *Pull Request*, pemilik proyek dapat melihat saran perubahan
16 proyek dari orang lain dan memberikan komentar/keterangan pada perubahan tersebut. Pemilik
17 proyek dapat melihat perbedaan pada kode pemilik proyek dengan perubahan yang disarankan
18 tersebut dan pemilik proyek dapat mengomentari baris pada kode tersebut. Orang lain dapat
19 memberikan komentar pada *Pull Request*. Sesudah pemilik proyek memberikan keterangan tentang
20 perubahan tersebut, kontributor menjadi tahu apa yang harus dilakukan agar perubahan tersebut
21 dapat disetujui. Apabila perubahan tersebut membuat pemilik proyek puas, pemilik proyek akan
22 *merge* perubahan tersebut dengan proyek aslinya dan otomatis akan menutup *Pull Request*.

BAB 3

ANALISIS

3.1 Analisis Permainan *Snake* yang Sudah Ada

Permainan *Snake* yang akan dianalisis adalah *Slither.io* dan *Snake* pada telepon genggam *Nokia*. *Slither.io* adalah permainan *web* yang dapat dimainkan oleh lebih dari 1 pemain (*multiplayer*). Cara bermainnya mirip seperti permainan *Snake* pada umumnya yaitu ular harus memakan makanan untuk mendapatkan skor. Dalam permainan ini, setiap pemain berkompetisi untuk menjadi pemain terbaik dengan cara mendapatkan skor sebanyak-banyaknya. Pemain akan kalah apabila ular milik pemain menabrak ular milik pemain lain.

Snake pada telepon genggam *Nokia* hanya dapat dimainkan oleh 1 pemain. Dalam permainan ini, ular harus mendapatkan skor sebanyak-banyaknya dengan memakan makanan. Setiap memakan makanan, skor akan bertambah sebanyak 1 poin. Pemain akan kalah apabila ular menabrak dinding labirin dan menabrak tubuh sendiri.

3.1.1 Ular dan Makanan

Ular pada *Slither.io* dibentuk dengan menggunakan sekumpulan lingkaran yang saling berdempetan satu sama lain seperti pada Gambar 3.1. Bagian kepala pada ular ditandai menggunakan sepasang mata. Ketika memakan makanan, tubuh ular akan memanjang dengan menambahkan sebuah lingkaran pada bagian ekor ular. Setiap memulai permainan, tubuh ular akan memiliki warna yang ditentukan secara acak.

Makanan pada *Slither.io* berbentuk lingkaran. Makanan ini ada yang berukuran besar dan ada yang berukuran kecil. Makanan ini tersebar pada labirin, jumlahnya sangat banyak dan warnanya bermacam-macam. Gambar 3.2 merupakan sekumpulan makanan yang terdapat pada labirin. Setiap makanan akan menambah skor sebanyak 1 poin.

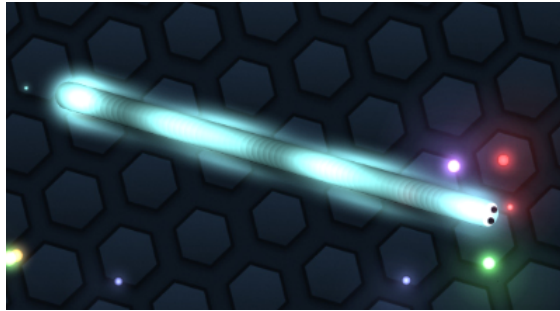
Gambar 3.1: Ular pada *Silther.io*Gambar 3.2: Makanan pada *Slither.io*

1 Ular pada *Snake Nokia* dibuat seperti permainan 8 bit yang terdiri dari *pixel-pixel* seperti pada
 2 Gambar 3.3. Pada permainan ini apabila kepala ular sudah dekat dengan makanan, maka kepala
 3 ular akan terlihat sedang membuka mulutnya. Makanan yang terdapat pada permainan ini ada 2
 4 macam yaitu makanan biasa dan makanan bonus seperti yang terlihat pada Gambar 3.4. Makanan
 5 biasa memiliki skor 1 poin dan makanan bonus memiliki skor 10 poin. Makanan bonus muncul
 6 secara acak dan memiliki batas waktu untuk berada pada labirin. Makanan bonus tidak hanya
 7 menambah skor lebih banyak saja tetapi makanan ini dapat membuat tubuh ular lebih panjang
 8 dibandingkan dengan memakan makanan biasa.

Gambar 3.3: Ular pada *Snake Nokia*Gambar 3.4: Makanan biasa(A) dan makanan bonus(B) pada *Snake Nokia*

3.1.2 Pergerakan Ular

Ular pada *Slither.io* digerakan dengan menggunakan *keyboard* dan *mouse*. Tombol ke kiri akan membuat ular bergerak berlawanan arah jarum jam dan tombol ke kanan akan membuat ular bergerak searah jarum jam. Semakin lama tombol ditekan, maka ular akan berbelok lebih cepat. Kursor pada *mouse* membuat ular bergerak ke arah posisi kursor tersebut. Ular dapat melaju dengan cepat(*speed up*) dengan menekan tombol *mouse* kiri seperti yang terdapat pada Gambar 3.5. Ketika ular sedang melaju dengan cepat, total skor yang didapat akan berkurang.



Gambar 3.5: Ular sedang melaju dengan cepat(*speed up*)

Ular pada *Snake Nokia* hanya dapat bergerak ke atas, ke bawah, ke kiri dan ke kanan. Ular dapat digerakan menggunakan tombol angka pada telepon genggam Nokia yaitu tombol 8 untuk bergerak ke atas, tombol 4 untuk bergerak ke kiri, tombol 6 untuk bergerak ke kanan dan tombol 2 untuk bergerak ke bawah. Kecepatan ular juga dapat dipilih. Semakin tinggi tingkat, maka ular akan bergerak semakin cepat.

3.1.3 Labirin

Labirin pada *Slither.io* hanya ada 1 saja. Labirin ini berbentuk lingkaran yang sisinya merupakan dinding. Apabila ular menabrak dinding labirin, maka permainan akan berakhir. Labirin ini cukup besar sehingga sangat kecil kemungkinan ular untuk menabrak dinding labirin. Gambar 3.6 menunjukkan peta labirin pada *Slither.io*. Pada peta labirin tersebut terdapat sekumpulan titik berwarna abu-abu yang merepresentasikan makanan.



Gambar 3.6: Peta labirin pada *Slither.io*

Labirin pada *Snake Nokia* lebih bervariasi dibandingkan dengan *Slither.io*. Pada permainan ini pemain dapat memilih labirin yang tersedia. Labirin dengan level yang lebih tinggi akan memiliki lebih banyak dinding.

3.2 Analisis Sistem yang Dibangun

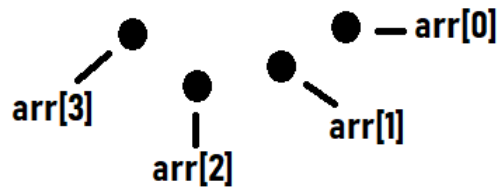
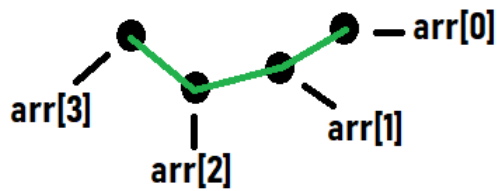
Permainan *Snake* 360 yang akan dibangun memiliki cara bermain yang mirip seperti permainan *Snake* pada umumnya. Perbedaan antara *Snake* 360 dengan permainan *Snake* pada umumnya adalah *Snake* 360 dapat menambahkan level dan labirin sendiri.

3.2.1 Menentukan Besar *Canvas*

Pada permainan *Open Source Snake* 360 ini, pemain dapat memainkan permainan tersebut di browser *smartphone* dan *browser desktop*. Hal ini akan memunculkan sebuah kesulitan yaitu menentukan dimensi *canvas* yang cocok apabila permainan tersebut dapat dimainkan pada *smartphone* dan *browser* pada *desktop*. Apabila besar *canvas* disesuaikan dengan besar layar pada *desktop*, maka objek yang ditampilkan (ular dan apel) akan terlihat sangat kecil dan apabila besar *canvas* disesuaikan dengan besar layar *smartphone*, maka besar *canvas* akan terlihat terlalu tinggi bila dilihat pada *desktop*. Cara ini dapat ditangani dengan membuat *canvas* mengikuti besar layar browser desktop dan layar browser *smartphone*. Namun, cara ini juga dapat menimbulkan masalah yaitu dalam pembuatan labirin. Format labirin akan terus diubah sesuai dengan besar layar. Untuk menyesuaikan *canvas* dengan besar layar, maka akan dibuat *canvas* berbentuk persegi dengan dimensi 600×600 *pixel*. Dimensi ini sesuai jika permainan ini dimainkan pada *smartphone* dan *desktop*. Selain menentukan dimensi, besar objek yang ada pada *canvas* juga harus diperbesar agar objek-objek pada *canvas* tidak terlihat kecil bagi pemain bermain menggunakan *smartphone*.

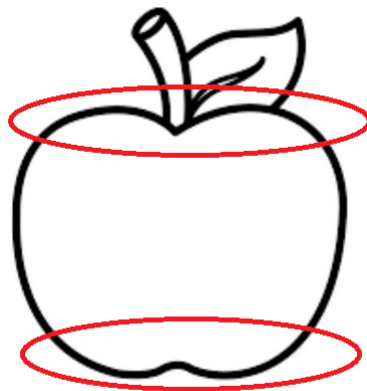
3.2.2 Menggambar Ular dan Apel

Tubuh ular dibuat menggunakan sekumpulan *line*/garis pendek. Setiap bagian tubuh ular memiliki panjang sebesar 1 *pixel* dan lebar tubuhnya sebesar 10 *pixel*. Bagian tubuh ular dibuat pendek untuk memudahkan pengecekan jika terjadi ular menabrak tubuhnya sendiri. Setiap bagian tubuh ular memiliki koordinat masing-masing. Koordinat setiap bagian tubuh disimpan pada sebuah *array* agar menggambar ular menjadi lebih mudah. Dalam tahap ini, tubuh ular masih berupa sekumpulan titik-titik yang merupakan koordinat bagian tubuh ular seperti pada Gambar 3.7. Algoritma untuk menggambar ular adalah dengan mengambil koordinat bagian tubuh ular mulai dari elemen *array* paling pertama (*arr*[0]) dan elemen *array* selanjutnya (*arr*[1]) lalu buat garis yang *start point*nya adalah elemen pertama (*arr*[0]) dan *end point*nya adalah elemen *array* kedua (*arr*[1]). Setelah itu ambil koordinat elemen *array* yang merupakan *end point* pada garis sebelumnya (*arr*[1]) dengan elemen *array* selanjutnya (*arr*[2]) dan gambar garisnya. Lakukan hal tersebut sampai *end point* garis mencapai elemen *array* paling akhir. Setelah digambar maka ular akan terlihat seperti Gambar 3.8.

Gambar 3.7: Koordinat bagian tubuh ular pada *array*

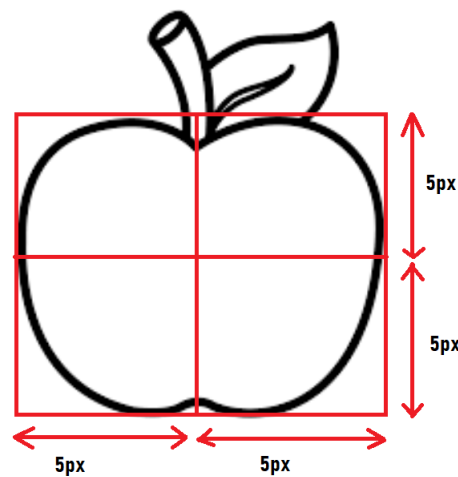
Gambar 3.8: Tubuh ular setelah digambar menggunakan garis

4 Untuk membuat apel digunakan *quadratic Bézier curve*. Kurva ini digunakan untuk membuat
 5 bagian-bagian apel yang melengkung. Bagian tersebut ditandai dengan lingkaran berwarna merah se-
 6 perti yang ditunjukkan pada Gambar 3.9 (gambar diambil dari pinterest. Link: <https://www.pinterest.com/pin/6903>)



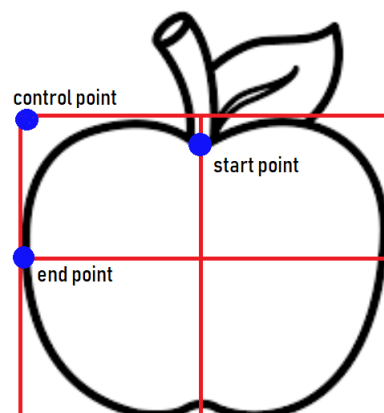
Gambar 3.9: Bagian pada apel (lingkaran merah) yang akan dibuat menggunakan kurva

7 Pertama, tentukan besar apel yang ingin dibuat. Dalam permainan ini besar apel yang dibuat
 8 adalah 20 *pixel*. Besar apel dibuat lebih besar dari lebar ular karena jika besar apel sama dengan
 9 lebar ular, besar apel terlihat sangat kecil. Selain itu, apel ini digambar pada *layout* yang berbentuk
 10 persegi. *Layout* persegi ini juga dapat mempermudah penggambaran apel. Karena menggunakan
 11 *layout* persegi, maka *origin* terletak pada titik sudut di sebelah kiri atas. Setelah itu, gambar setiap
 12 bagian apel. Bagian apel dibagi menjadi 4 seperti pada Gambar 3.10 sehingga besar setiap bagian
 13 apel tersebut adalah 10 *pixel*.

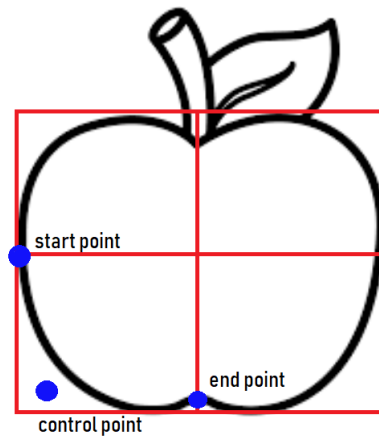


Gambar 3.10: Pembagian gambar apel dengan layout persegi beserta ukuran pada setiap bagian

14 Gambar bagian atas apel terlebih dahulu. Gunakan *method moveTo()* untuk menentukan titik
 15 mulainya. Titik mulainya terletak pada bagian tengah atas apel yang melengkung ke dalam. Dari
 1 titik itu, buat kurva yang *control pointnya* adalah titik ujung *layout* persegi. Jika ingin menggambar
 2 bagian kiri apel terlebih dahulu maka *control pointnya* adalah titik ujung kiri *layout* tersebut.
 3 Setelah itu, tentukan *end point* kurva tersebut. Pada Gambar 3.11 terdapat *start point*, *control point*
 4 dan *end point* untuk membuat bagian sisi kiri atas apel. Sesudah itu, buatlah bagian bawah apel.
 5 Caranya sama seperti sebelumnya namun *control pointnya* dan *end pointnya* berbeda. Posisi *control*
 6 *pointnya* sedikit menjorok ke dalam dan posisi *end pointnya* terdapat di tengah bawah seperti
 7 pada Gambar 3.12. *Start point* tidak perlu diatur lagi, karena *start pointnya* sudah tergantikan
 8 dengan posisi *end point* pada kurva sebelumnya. Sampai pada bagian ini, bagian kiri apel sudah
 9 selesai dibuat. Untuk membuat bagian kanan apel, caranya sama seperti membuat bagian kiri apel.
 10 Karena bagian kiri apel simetris dengan bagian kanan apel, maka hanya perlu mengubah *control*
 11 *point* dan *end pointnya* saja. Dengan memanfaatkan bentuk simetris dari apel, maka jarak antara
 12 *control point* dan *end point* pada bagian kiri apel dengan batasan tengah sama dengan jarak antara
 13 *control point* dan *end point* dengan batas tengah pada bagian kanan apel.



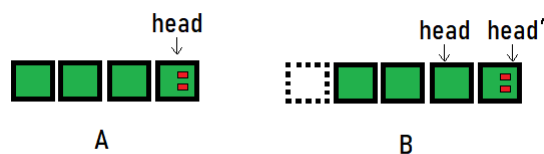
Gambar 3.11: *Start point*, *control point* dan *end point* untuk menggambar apel bagian kiri atas



Gambar 3.12: *Start point*, *control point* dan *end point* untuk menggambar apel bagian kiri bawah

3.2.3 Pergerakan Ular

Untuk membuat ular bergerak maju, dilakukan penambahan kepala dan pembuangan ekor secara bersamaan ketika ular sedang bergerak maju. Ilustrasinya dapat dilihat pada Gambar 3.13. Untuk membuat ular bergerak dengan menggunakan cara pada Gambar 3.13, algoritmanya adalah sebagai berikut : Pertama, semua elemen *array* akan *dishift*/digeser dan elemen pertama akan digantikan dengan koordinat yang baru. Setelah itu dilakukan pengecekan apakah panjang tubuh ular lebih besar dari jumlah elemen *array* tubuh ular. Jika benar, maka tidak dilakukan pembuangan elemen terakhir dan jika salah, maka tidak akan dilakukan apa-apa.



Gambar 3.13: Ilustrasi ular sebelum bergerak maju(A) dan setelah bergerak maju(B)

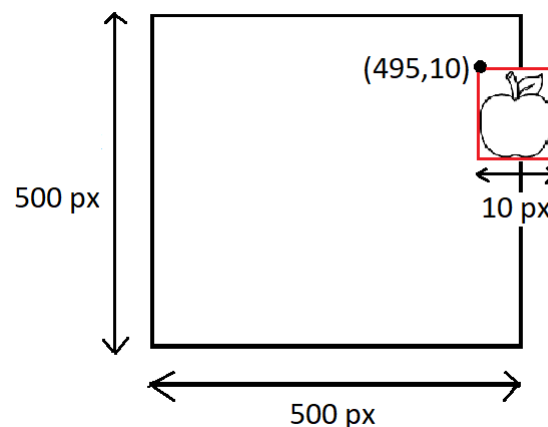
Kecepatan ular pada permainan ini adalah 1 sampai 5 *pixel per frame*. Kecepatan maksimal ular tidak boleh melebihi lebar tubuh ular. Jika kecepatannya melebihi lebar ular, maka ketika terjadi tabrakan dengan tubuhnya sendiri, kepala ular tidak akan bertabrakan dengan tubuhnya. Kepala ular akan terlihat seolah-olah melompati tubuhnya sendiri. Dalam permainan ini, kecepatan ular adalah 2 *pixel per frame*, karena dengan kecepatan 1 *pixel per frame*, ular terlihat bergerak lebih lambat.

Ular dapat berbelok dengan menggunakan tombol pada *keyboard*. Tombol ke kiri akan membuat ular bergerak melawan arah jarum jam dan tombol ke kanan akan membuat ular akan bergerak searah jarum jam. Pada permainan yang akan dibuat ini, digunakan sudut sebagai nilai untuk membuat ular dapat bergerak 360°. Jika menekan tombol ke kiri maka sudut akan berkurang dan jika menekan tombol ke kanan maka sudut akan bertambah. Ketika menambahkan dan mengurangi sudut, perlu dilakukan pengecekan apabila nilai sudut valid atau tidak. Karena nilai sudut yang valid

adalah antara nilai 0 sampai 360, maka apabila nilai sudut kurang dari 0, ubahlah sudut tersebut menjadi 360 dan apabila nilai sudut lebih besar dari 360, ubahlah nilai sudut tersebut menjadi 0. Dibutuhkan rumus trigonometri untuk menentukan posisi kepala ular. Untuk menghitung posisi koordinat x, digunakan *sinus* sedangkan untuk menghitung posisi koordinat y menggunakan *cosinus*. Jadi koordinat x dan y pada kepala ular akan ditambahkan dengan hasil perhitungan *sinus* dan *cosinus*.

3.2.4 Mengacak posisi apel

Posisi apel akan diacak di daerah *canvas*. Untuk mengacak posisi apel, digunakan fungsi *Math.random()*. Nilai yang akan diacak adalah posisi x dan y dari apel. Hasil dari fungsi *Math.random()* akan dikalikan dengan lebar *canvas* untuk mendapatkan nilai x dan dikalikan dengan tinggi *canvas* untuk mendapatkan nilai y. Karena apel ini dibuat dengan menggunakan *layout* persegi, maka posisi x dan y pada apel terletak di titik sudut kiri atas. Hal ini akan memungkinkan gambar apel akan terpotong seperti yang terlihat pada Gambar 3.14. Misal, besar *canvas* adalah 500 x 500 dan besar apel adalah 10 dan mendapatkan posisi apel adalah (495,10). Posisi x apel ditambah dengan besar apel hasilnya akan melebihi besar *canvas* sehingga membuat sebagian gambar apel terlihat terpotong. Maka dari itu, lebar dan tinggi *canvas* yang dikalikan dengan bilangan acak, akan dikurangi sebesar ukuran apel tersebut. Nilai yang dihasilkan adalah nilai yang bertipe *float* sedangkan posisi x dan y pada apel membutuhkan input bilangan bulat. Untuk mendapatkan bilangan bulat tersebut, nilai yang sudah dihitung tadi dibulatkan ke bawah. Mengacak posisi apel tidak hanya mengacak posisi pada *canvas* saja, tetapi harus mengecek apakah posisi apel tersebut tidak bertabrakan dengan tubuh ular atau dinding labirin.



Gambar 3.14: Gambar apel yang terpotong sesudah mengacak posisi apel

3.2.5 Menggambar Labirin

Pada permainan ini, format labirin dapat dibuat dengan menggunakan JSON dan *file text*. Permainan ini menggunakan *file text* sebagai format labirin, karena *file text* lebih mudah untuk dibuat dan dimengerti oleh pembuat labirin. Pada permainan ini, pemain dapat memilih labirin sesuai dengan yang pemain inginkan. Ketika pemain sudah memilih labirin, maka akan dicari nama file text

yang sesuai dengan yang dipilih. Setiap nama file labirin diawali dengan kata 'level' dan diakhiri dengan angka yang merupakan labirin yang dipilih pemain. Misal, jika pemain memilih labirin yang pertama, maka file yang akan dibaca adalah file yang bernama 'level1.txt'. Setelah file text sudah ditemukan, maka labirin sudah siap untuk digambar.

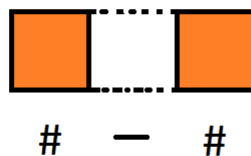
5

Pada *file* text, daerah yang merupakan dinding labirin ditulis dengan menggunakan simbol '#' sedangkan untuk daerah yang bukan merupakan dinding labirin ditulis dengan menggunakan simbol '-'. Sebuah simbol merepresentasikan besar dinding. Jika pada *file* tersebut terdapat text '#-#', itu artinya menggambar dinding, tidak menggambar dinding dan menggambar dinding lagi. Hasilnya dapat dilihat pada Gambar 3.15. Besar *canvas* untuk permainan ini adalah 600×600 *pixel* dan besar dinding labirin sebesar 10 *pixel*. Besar dinding tidak boleh lebih kecil dari lebar tubuh ular. Apabila besar dinding lebih kecil dari ular, ular tidak akan dapat melewati jalur yang diapit oleh 2 buah dinding seperti yang terlihat pada Gambar 3.16. Jumlah baris pada *file text* akan disamakan dengan jumlah kolomnya. Sesuai dengan besar *canvas* dan besar dinding labirin yaitu 600×600 *pixel*, maka dapat ditentukan bahwa setiap *file text* memiliki 60 baris dan setiap barisnya terdiri dari 60 karakter. Untuk menggambar dinding secara horizontal, maka hanya menggambar garis dengan panjang 10 *pixel* dari titik awal ke titik akhir. Sebagai contoh, apabila karakter pada baris pertama dan kolom pertama adalah '#', maka dinding akan digambar pada *canvas* dari titik(0,0) sampai titik(10,0) dengan lebar dinding 10 *pixel*. Level pada labirin dapat ditentukan berdasarkan kerumitan labirin. Labirin yang memiliki dinding yang banyak dan kompleks akan mendapatkan level yang lebih tinggi dibandingkan dengan labirin yang memiliki sedikit dinding dan lebih simpel.

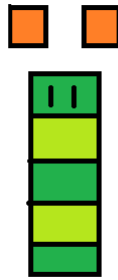
22

Untuk mendapatkan file tersebut, Javascript tidak dapat digunakan karena masalah keamanan pada browser yang akan mem-block akses untuk membaca isi file dari harddisk. Oleh karena itu, AJAX akan digunakan untuk membaca isi labirin dari folder. Dengan menggunakan AJAX, terdapat sebuah masalah yaitu AJAX bersifat *asynchronous* yang menyebabkan labirin belum selesai digambar tetapi permainan sudah siap untuk dimainkan. Cara untuk menangani masalah ini adalah dengan menggunakan *callback*. Dengan adanya *callback*, kita dapat memastikan bahwa permainan sudah siap untuk dimainkan apabila labirin sudah digambar.

8



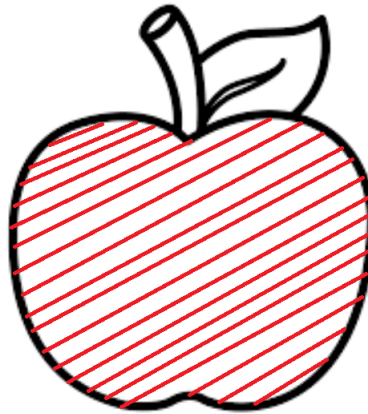
Gambar 3.15: Menggambar dinding menggunakan simbol pada file text



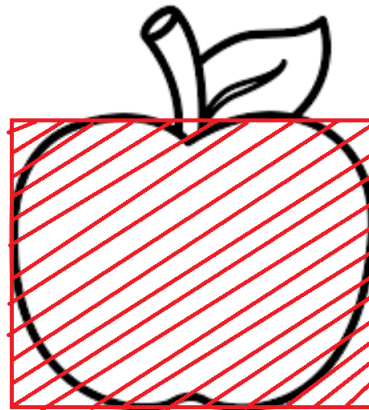
Gambar 3.16: Ular ingin melewati jalur yang diapit oleh 2 buah dinding

3.2.6 Pengecekan tabrakan(*Collision Detection*)

Pada permainan ini terdapat pengecekan tabrakan yang dapat mengecek apakah ular sudah memakan makanan, ular menabrak tubuhnya sendiri, dan ular menabrak dinding labirin. Seluruh pengecekan ini akan dilakukan pada setiap *frame*. Pada pengecekan tabrakan pada apel dan ular, hanya perlu mengecek tabrakan antara kepala ular dengan apel. Karena jalur yang dilalui oleh kepala ular, akan selalu dilalui oleh bagian tubuh ular. Dengan kata lain, bagian tubuh ular akan mengikuti ke mana kepala ular akan bergerak. Dengan ini, tidak perlu dilakukan *collision detection* antara bagian tubuh ular dengan apel. Cukup hanya dengan mengecek tabrakan antara kepala ular dengan apel saja. Untuk mengetahui terjadinya tabrakan antara ular dengan apel, maka akan dibuat daerah tabrakan pada apel. Daerah tabrakan ini digunakan untuk mengecek apakah 2 benda saling bertabrakan satu sama lain. Daerah tabrakan pada apel ditandai dengan arsiran berwarna merah yang terdapat pada Gambar 3.17. Namun, untuk membuat daerah tabrakan ini cukup sulit ketika mengecek adanya tabrakan antara ular dengan apel terutama pada bagian lengkungan pada apel. Karena itu, daerah tabrakan pada apel dibuat dengan menggunakan bentuk persegi seperti pada Gambar 3.18. Jika posisi kepala ular berada di dalam daerah tabrakan apel, maka dipastikan bahwa ular tersebut sudah memakan apel. Algoritma untuk mengecek tabrakan adalah sebagai berikut : cek apakah koordinat x dari kepala ular lebih besar dari posisi sisi kiri daerah tabrakan dan lebih kecil dari posisi sisi kanan daerah tabrakan. Kemudian cek apakah koordinat y dari kepala ular lebih besar dari posisi sisi atas daerah tabrakan dan lebih kecil dari posisi sisi bawah daerah tabrakan. Jika posisi kepala ular berada memenuhi ketentuan tersebut, maka kepala ular berada di dalam daerah tabrakan apel.



Gambar 3.17: Daerah tabrakan pada apel



Gambar 3.18: Daerah tabrakan berbentuk persegi pada apel

6 Untuk mengecek tabrakan antara ular dengan tubuhnya sendiri adalah dengan mengecek tabrak-
7 an antara kepala ular dengan seluruh bagian tubuh ular. Algoritma pengecekanya adalah sebagai
8 berikut : jika koordinat x kepala ular lebih kecil dari koordinat x bagian tubuh ular dikurangi
9 panjang dari bagian tubuh ular dan lebih besar dari koordinat x bagian tubuh ular ditambah
10 dengan panjang dari bagian tubuh ular. Kemudian dicek apabila koordinat y kepala ular lebih kecil
11 dari koordinat y bagian tubuh ular dikurangi panjang dari bagian tubuh ular dan lebih besar dari
12 koordinat y bagian tubuh ular ditambah dengan panjang dari bagian tubuh ular. Apabila posisi
13 kepala ular memenuhi ketentuan tersebut, maka posisi kepala ular berada di dalam daerah tabrakan
14 pada sebuah bagian tubuh ular.

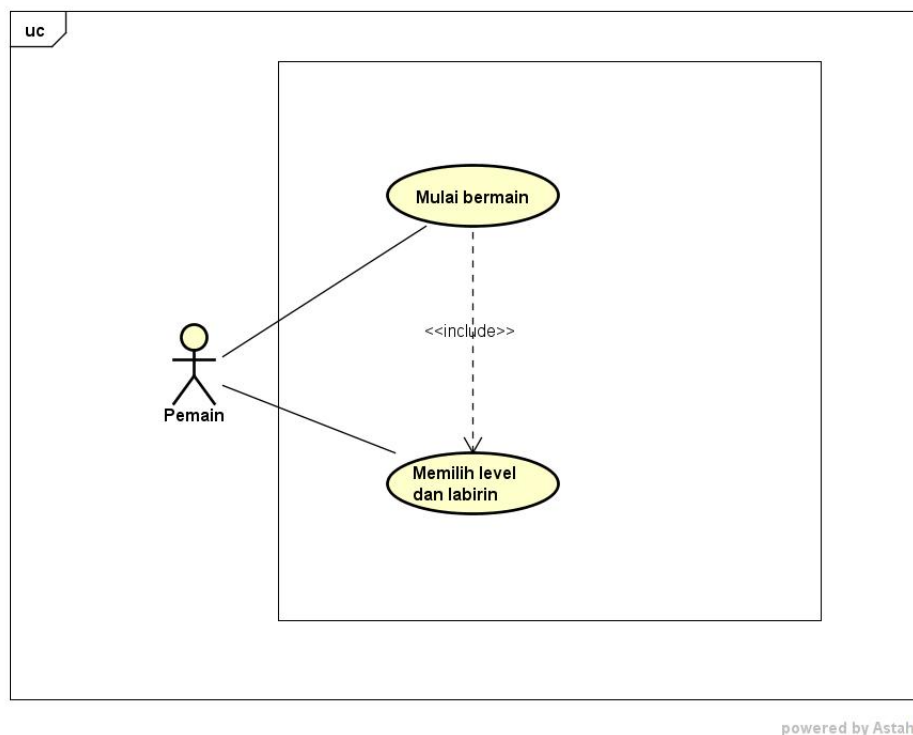
15
16 Untuk mengecek tabrakan dengan dinding labirin, dilakukan pengecekan antara kepala ular
17 dengan sebuah dinding. Bila dilakukan pengecekan antara kepala ular dengan seluruh dinding
18 labirin, maka animasi permainan akan berjalan lebih lambat. Semakin banyak dinding, animasi
19 akan berjalan lebih lambat. Cara untuk mengecek tabrakan antara kepala ular dengan dinding
1 adalah sebagai berikut : misal posisi kepala ular adalah (9,9). Jika besar dinding adalah 10 pixel,
2 maka kepala ular akan berada di daerah pada koordinat (0,0) sampai (10,10). Pada Gambar...
3 terdapat gambaran untuk memperjelas contoh tersebut. Kemudian, posisi kepala ular tersebut
4 akan dibagi dengan besar dinding (10 pixel). Hasil yang didapat dari perhitungan tersebut adalah

(0,0). Hasil tersebut akan digunakan untuk mengecek dinding pada labirin yang diambil dari file text yang sudah disimpan di array. Karena hasil dari perhitungan tersebut adalah (0,0) maka akan dicek apakah array elemen pertama dan karakter pertama merupakan dinding. Misal pada array elemen pertama dan karakter pertama merupakan dinding, maka kepala ular menabrak dinding. Pengecekan tabrakan dengan dinding labirin ini akan membuat ular dapat bergerak diantara 2 dinding seperti pada Gambar... Jika pengecekan terlalu akurat maka ular tidak dapat bergerak diantara kedua dinding.

3.3 Analisis Berorientasi Objek

3.3.1 Skenario Permainan

Pada bagian ini akan dijelaskan dan ditunjukkan diagram *use case* dari permainan *Snake 360*. Penjelasan meliputi skenario, aktor, prakondisi skenario normal dan eksepsi. Aktor yang melakukannya adalah pemain. Pada Gambar 3.19 terdapat diagram *use case* dari permainan *Snake 360*.



Gambar 3.19: Diagram *use case* dari permainan *Snake 360*

Berikut adalah skenario dari diagram *use case* :

1. Skenario : Mulai bermain

Aktor : Pemain

Prakondisi : Pemain memulai permainan.

Skenario normal : Pemain memulai bermain. Setelah memilih, pemain akan memilih level dan labirin.

Eksepsi : -

2. Skenario : Memilih level dan labirin

Aktor : Pemain

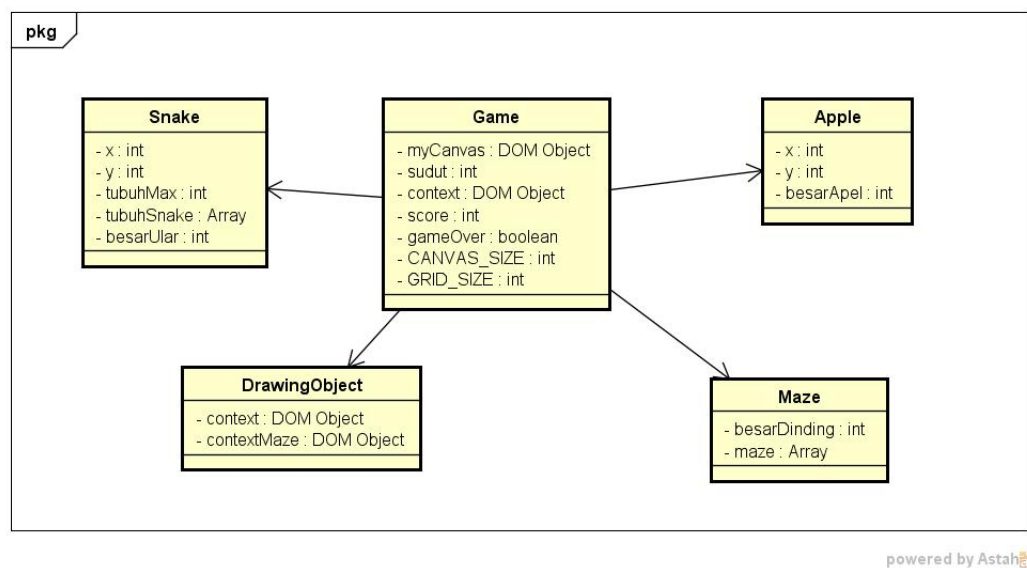
Prakondisi : Pemain sudah mulai bermain.

Skenario normal : Pemain memilih level dan labirin yang diinginkan.

Eksepsi : -

3.3.2 Diagram Kelas

Pada Gambar 3.20 terdapat diagram kelas dari *Snake 360*.



Gambar 3.20: Diagram kelas dari permainan *Snake 360*

Diagram kelas terdiri dari beberapa kelas yaitu :

1. Kelas Snake merupakan kelas yang merepresentasikan objek ular.
2. Kelas Apple merupakan kelas yang merepresentasikan objek apel.
3. Kelas Game merupakan kelas yang mengatur jalanya permainan.
4. Kelas Maze merupakan kelas yang merepresentasikan objek labirin.
5. Kelas DrawingObject merupakan kelas untuk menggambar semua objek pada canvas.

Berikut adalah atribut yang dimiliki setiap kelas :

1. Kelas Snake

int

- x, merupakan posisi ular pada koordinat x.
- y, merupakan posisi ular pada koordinat y.
- tubuhMax, merupakan panjang tubuh ular.

- besarUlar, merupakan lebar tubuh ular.

2. Kelas Apel

int

- x, merupakan posisi apel pada koordinat x.
- y, merupakan posisi apel pada koordinat y.
- besarApel, merupakan besar apel.

3. Kelas Game

int

- sudut, merupakan besar sudut yang digunakan untuk ular berbelok.
- score, merupakan skor yang didapat pada permainan.
- CANVAS_SIZE, merupakan lebar dan tinggi canvas.
- GRID_SIZE, merupakan besar grid.

boolean

- gameOver, memberitahu apakah permainan sudah berakhir atau belum.

4. Kelas Maze

int

- besarDinding, merupakan besar lebar dinding.

BAB 4

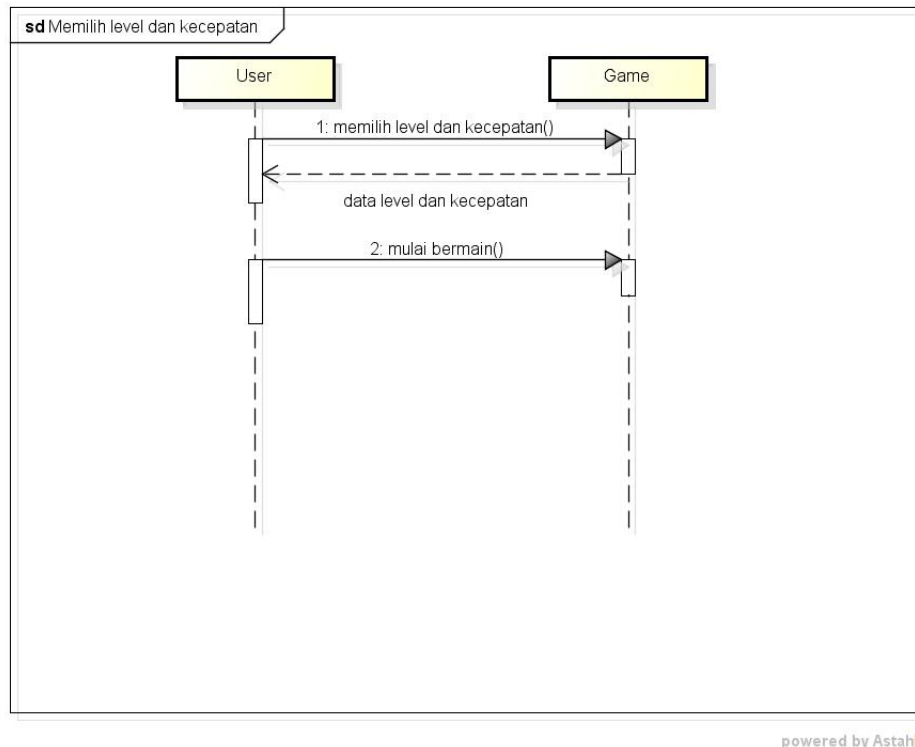
PERANCANGAN

Pada bab ini akan dibahas mengenai perancangan permainan yang dibangun. Perancangan akan dilakukan meliputi perancangan diagram *sequence*, perancangan diagram kelas, dan perancangan *mockup*.

4.1 Diagram Sequence

Pada bagian ini akan ditunjukkan dan dijelaskan diagram sequence Open Source Snake 360. Diagram sequence yang dibuat meliputi memilih level dan kecepatan.

4.1.1 Memilih Level dan Kecepatan



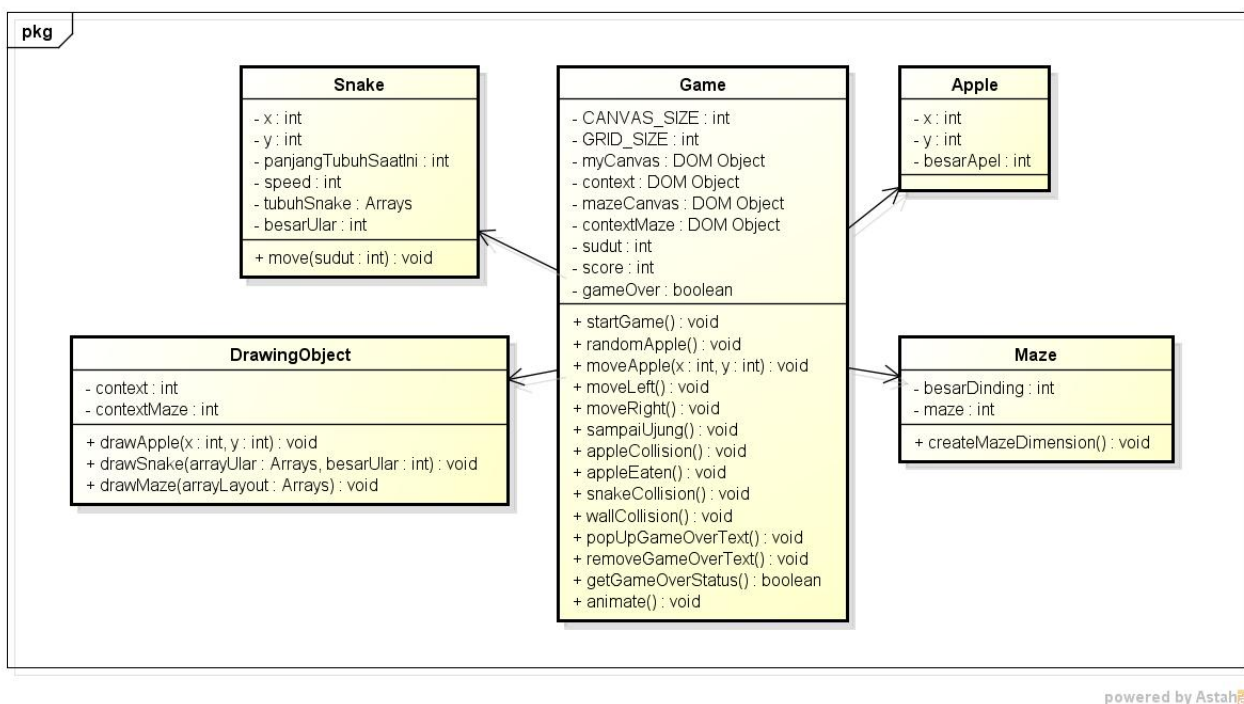
Gambar 4.1: Diagram sequence untuk memilih level dan kecepatan

Pada Gambar 4.1, pemain memulai bermain dengan memilih level dan kecepatan terlebih dahulu. Berikut adalah penjelasan dari Gambar 4.1:

1. Pemain memilih level labirin dan kecepatan ular.
2. Kelas Game akan mengecek apakah input yang dimasukkan oleh user sudah benar atau belum. Jika input belum benar, maka kelas Game akan memberitahu bahwa input yang dimasukkan tidak tepat.
3. Jika input sudah benar, maka pemain dapat mulai bermain.

4.2 Diagram Kelas Rinci

Pada bagian ini akan ditunjukkan dan dijelaskan diagram kelas dari *Open Source Snake 360* secara lengkap. Diagram kelas dapat dilihat pada Gambar 4.2.



Gambar 4.2: Diagram kelas rinci dari Open Source *Snake 360*

Deskripsi Kelas dan Method

Pada bagian ini akan dijelaskan deskripsi kelas dan *method-method* pada setiap kelas. Penjelasan kelas dan *method* meliputi nama kelas, deskripsi *method*, input yang dibutuhkan, dan output yang dihasilkan.

1. Kelas *Game*

Kelas *Game* merupakan kelas utama dari permainan ini. Kelas ini mengatur jalannya permainan.

- Nama *method* : *startGame*
 Deskripsi : memulai permainan
 Input : tidak ada

31	Output : tidak ada
32	
33	• Nama <i>method</i> : <i>randomApple</i>
34	Deskripsi : mengacak posisi apel
35	Input : tidak ada
36	Output : tidak ada
37	
38	• Nama <i>method</i> : <i>moveApple</i>
39	Deskripsi : memindahkan posisi apel
40	Input : <i>int</i> x, <i>int</i> y
1	– x : koordinat x milik apel
2	– y : koordinat y milik apel
3	Output : tidak ada
4	
5	• Nama <i>method</i> : <i>moveLeft</i>
6	Deskripsi : membuat ular bergerak berlawanan arah jarum jam
7	Input : tidak ada
8	Output : tidak ada
9	
10	• Nama <i>method</i> : <i>moveRight</i>
11	Deskripsi : membuat ular bergerak searah jarum jam
12	Input : tidak ada
13	Output : tidak ada
14	
15	• Nama <i>method</i> : <i>sampaiUjung</i>
16	Deskripsi : membuat ular akan muncul di sisi yang berlawanan ketika ular sudah menca-
17	pai ujung labirin
18	Input : tidak ada
19	Output : tidak ada
20	
21	• Nama <i>method</i> : <i>appleColiision</i>
22	Deskripsi : mengecek tabrakan antara apel dengan kepala ular
23	Input : tidak ada
24	Output : tidak ada
25	
26	• Nama <i>method</i> : <i>appleEaten</i>
27	Deskripsi : aksi yang dilakukan apabila ular sudah memakan apel
28	Input : tidak ada
29	Output : tidak ada
30	

- Nama *method* : *snakeCollision*

Deskripsi : mengecek tabrakan antara kepala ular dengan tubuhnya sendiri

Input : tidak ada

Output : tidak ada

- Nama *method* : *wallCollision*

Deskripsi : mengecek tabrakan antara kepala ular dengan dinding labirin

Input : tidak ada

Output : tidak ada

- Nama *method* : *popUpGameOverText*

Deskripsi : memunculkan tulisan 'Game Over'

Input : tidak ada

Output : tidak ada

- Nama *method* : *removeGameOverText*

Deskripsi : menghilangkan tulisan 'Game Over'

Input : tidak ada

Output : tidak ada

- Nama *method* : *getGameOverStatus*

Deskripsi : mendapatkan status gameOver

Input : tidak ada

Output : *boolean gameOver*

– *gameOver* : status apabila permainan sudah berakhir atau belum

- Nama *method* : *animate*

Deskripsi : membuat animasi dari setiap objek

Input : tidak ada

Output : tidak ada

2. Kelas *Snake*

Kelas *Snake* merupakan kelas yang merepresentasikan objek ular.

- Nama *method* : *move*

Deskripsi : memulai permainan

Input : *int x, int y*

– *x* : koordinat x milik ular

– *y* : koordinat y milik ular

Output : tidak ada

3. Kelas *DrawingObject*

Kelas *DrawingObject* merupakan kelas yang bertugas untuk menggambar objek-objek yang terdapat pada canvas.

- Nama *method* : *drawApple*

Deskripsi : menggambar objek apel

Input : int x, int y

– x : koordinat x milik apel

– y : koordinat y milik apel

Output : tidak ada

- Nama *method* : *drawSnake*

Deskripsi : menggambar objek ular

Input : *int*[] arrayUlar, *int* besarUlar

– arrayUlar : koordinat x dan y milik setiap bagian tubuh ular

– besarUlar : lebar tubuh ular

Output : tidak ada

- Nama *method* : *drawMaze*

Deskripsi : menggambar labirin

Input : *String* arrayLayout, *int* besarDinding

– arrayLayout : layout labirin yang akan digambar

– besarDinding : besar dinding labirin

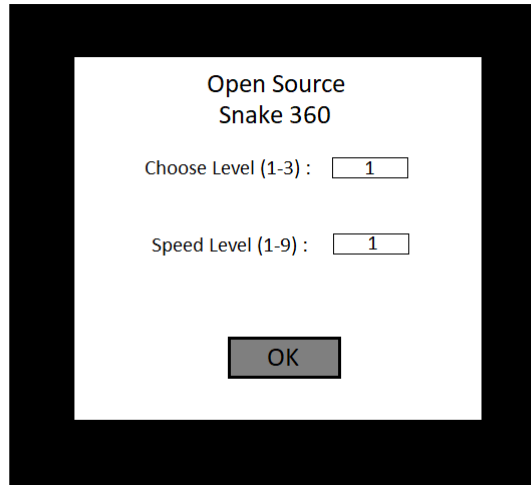
Output : tidak ada

4.3 Mockup

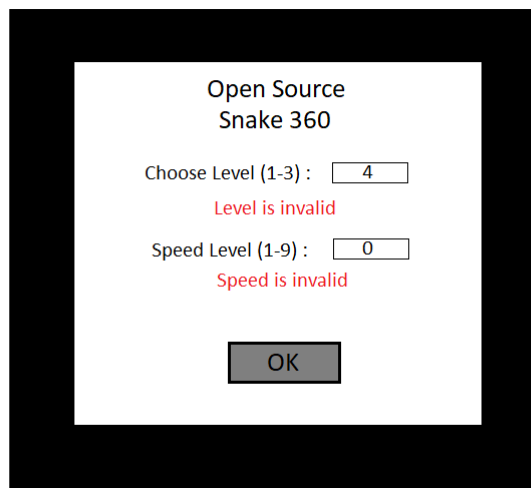
Pada bagian ini akan ditunjukkan rancangan antarmuka dari permainan yang dibangun yang terdiri dari menu pemilihan level, mulai bermain, dan permainan berakhir.

4.3.1 Tampilan Menu Utama

Gambar 4.3 merupakan rancangan tampilan awal dari permainan yang dibangun. Pada tampilan ini terdapat judul permainan, 2 buah input untuk memilih level dan kecepatan gerak ular, dan tombol OK. Tampilan menu utama akan menampilkan pesan kesalahan seperti terdapat pada Gambar 4.4. Apabila pemain salah memasukkan data, maka permainan tidak akan dimulai jika tombol OK ditekan.



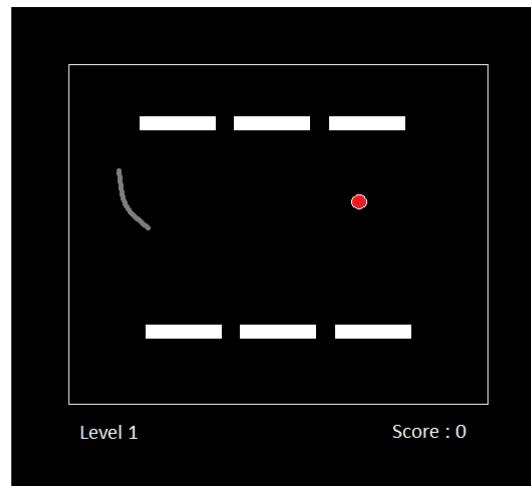
Gambar 4.3: Rancangan tampilan menu utama



Gambar 4.4: Rancangan tampilan menu utama jika pemain salah memasukkan data

21 4.3.2 Tampilan Bermain

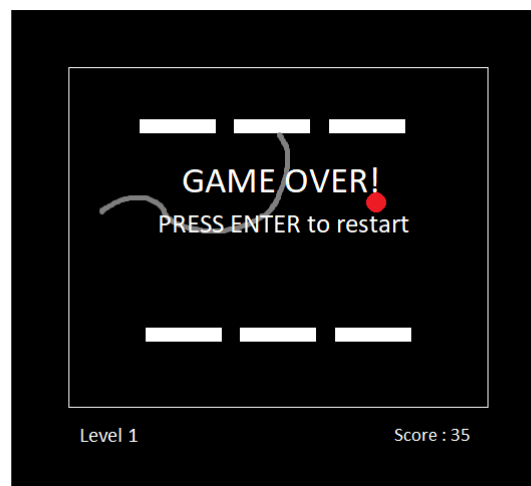
22 Tampilan bermain akan muncul setelah pemain memilih level dan kecepatan ular pada menu utama
23 dengan benar dan menekan tombol OK (Gambar 4.3). Gambar 4.5 merupakan tampilan permainan
24 sudah dimulai. Pada tampilan ini terdapat ular, dinding labirin dan makanan berbentuk apel. Pada
1 tampilan ini juga terdapat level labirin yang dipilih dan skor yang didapat.



Gambar 4.5: Rancangan tampilan bermain

2 4.3.3 Tampilan Permainan Berakhir

- 3 Tampilan ini akan muncul apabila permainan berakhir. Permainan akan berakhir jika ular menabrak
- 4 dinding labirin atau menabrak tubuhnya sendiri. Gambar 4.6 merupakan tampilan permainan
- 5 berakhir. Pada tampilan ini, pemain dapat mengulang permainan dengan menekan tombol 'ENTER'.
- 6 Pemain akan dialihkan ke tampilan menu utama(Gambar 4.3) apabila tombol 'ENTER' ditekan.



Gambar 4.6: Rancangan tampilan permainan berakhir

BAB 5

IMPLEMENTASI DAN PENGUJIAN

Pada bab ini akan dibahas mengenai hasil implementasi dan pengujian dari Open Source Snake 360.

5.1 Implementasi

Pada bagian ini akan dijelaskan mengenai lingkungan yang digunakan untuk membangun dan implementasi antarmuka dari Open Source Snake 360.

5.1.1 Lingkungan Perangkat Keras

Berikut adalah lingkungan perangkat keras yang digunakan dalam pembangunan permainan ini:

1. Perangkat : Laptop
2. Processor : Intel Core i5-7200U 2.5GHz
3. RAM : 4.00 GB
4. Video Card : GeForce 930MX
5. Monitor : 14"
6. Storage : 1TB

Pada pengujian digunakan 1 buah perangkat mobile berbasis android dan 1 buah perangkat desktop. Berikut adalah lingkungan perangkat keras yang digunakan dalam pengujian permainan ini:

Perangkat 1

1. Perangkat : Laptop
2. Processor : Intel Core i5-7200U 2.5GHz
3. RAM : 4.00 GB
4. Video Card : GeForce 930MX
5. Monitor : 14"

6. Storage : 1TB

Perangkat 2

1. Perangkat : SM-J730G

2. Processor : Exynos 7870 Octa 1600MHz Cortex-A53

3. RAM : 3.00 GB

4. Video Card : Mali-T830

5. Monitor : 5.5"

6. Storage : 32 GB

5.1.2 Lingkungan Perangkat Lunak

Berikut adalah lingkungan perangkat lunak yang digunakan dalam pembangunan permainan ini:

1. Sistem Operasi Laptop : Windows 10 64-bit

2. Bahasa Pemrograman : Javascript, HTML

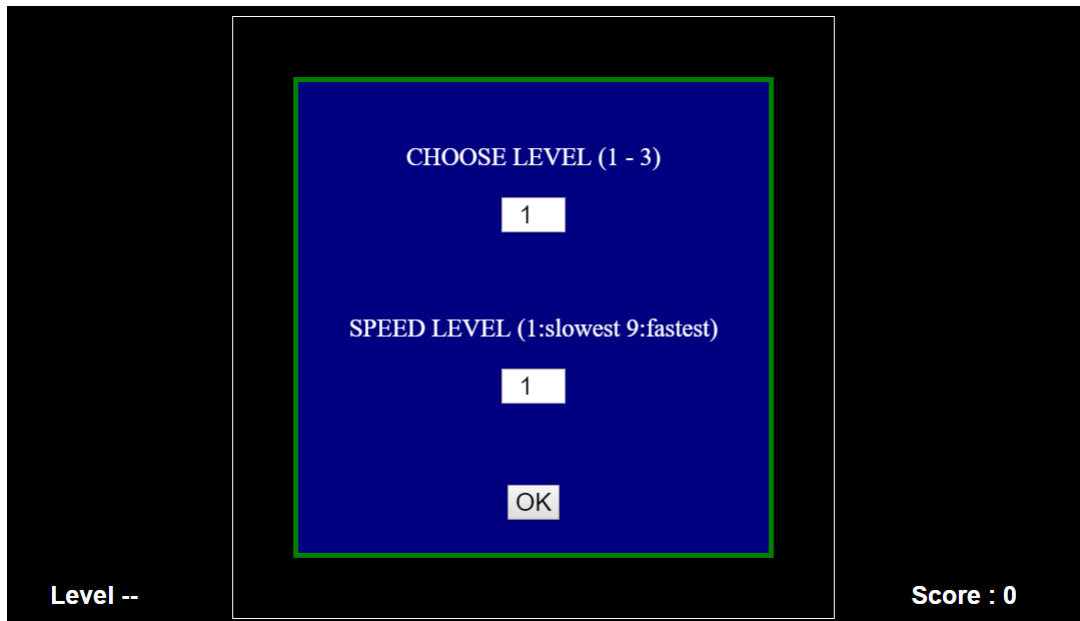
3. Sistem Operasi Smartphone : Android Nougat v7.0

5.1.3 Implementasi Antarmuka

Pada subbab ini akan ditampilkan dan dijelaskan tampilan antarmuka dari Open Source Snake 360.

Tampilan Menu Utama

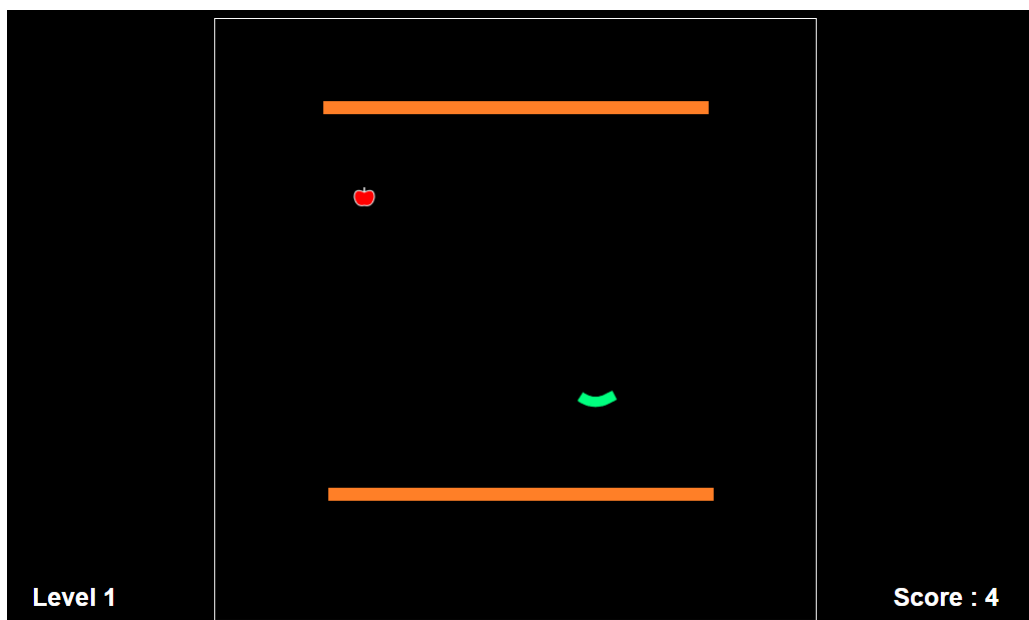
Gambar 5.1 merupakan tampilan antarmuka menu utama. Pada tampilan ini terdapat judul dari permainan, input untuk mengisi level dan kecepatan, dan tombol "OK".



Gambar 5.1: Tampilan Menu Utama

Tampilan Bermain

Gambar 5.2 merupakan tampilan antarmuka mulai bermain. Tampilan ini muncul apabila pemain memasukkan data level dan kecepatan ular dengan benar dan menekan tombol "OK". Pada tampilan ini terdapat ular yang dikontrol oleh pemain, dinding labirin, makanan ular, level labirin dan skor yang didapat pemain.



Gambar 5.2: Tampilan Bermain

5.2 Pengujian

Pengujian terhadap permainan Open Source Snake 360 ini bertujuan untuk mengetahui apakah permainan yang dibangun sudah berjalan sesuai dengan rancangan. Pengujian yang dilakukan meliputi pengujian fungsional.

5.2.1 Pengujian Fungsional

Pengujian fungsional dilakukan untuk mengetahui tingkat keberhasilan perangkat lunak menjalankan fungsi-fungsi yang ada. Berikut akan ditunjukkan pengujian pada tampilan:

1. Pengujian fungsionalitas pada tampilan menu utama.

Tabel 5.1: Pengujian Fungsional pada Tampilan Menu Utama

Kasus uji	Hasil yang diharapkan	Hasil uji
Pemain memilih labirin dan kecepatan berbelok	Jika pemain salah memasukkan data level labirin, maka akan ditampilkan sebuah text bahwa data yang diisi tidak valid	Hasil pengujian sesuai dengan yang diharapkan
Pemain menekan tombol "mulai bermain"	Pemain dapat memulai permainan. Kondisi untuk dapat memulai permainan adalah data level labirin dan kecepatan berbelok sudah valid.	Hasil pengujian sesuai dengan yang diharapkan

Berdasarkan tabel 5.1, dapat disimpulkan bahwa kasus uji pada tampilan menu utama membawakan hasil sesuai dengan yang diharapkan.

2. Pengujian fungsionalitas tampilan bermain pada desktop.

Tabel 5.2: Pengujian Fungsional Tampilan Bermain pada Desktop

Kasus uji	Hasil yang diharapkan	Hasil uji
Tombol arah kiri ditekan	Ular akan bergerak melawan arah jarum jam	Hasil pengujian sesuai dengan yang diharapkan
Tombol arah kanan ditekan	Ular akan bergerak searah jarum jam	Hasil pengujian sesuai dengan yang diharapkan
Ular memakan apel	Pemain akan mendapatkan skor	Hasil pengujian sesuai dengan yang diharapkan
Ular menabrak dinding	Tampilan "game over" akan muncul	Hasil pengujian sesuai dengan yang diharapkan
Ular menabrak tubuh sendiri	Tampilan "game over" akan muncul	Hasil pengujian sesuai dengan yang diharapkan

Berdasarkan tabel 5.2, dapat disimpulkan bahwa kasus uji tampilan bermain pada desktop membawakan hasil sesuai dengan yang diharapkan.

3. Pengujian fungsionalitas pada tampilan "game over"

Tabel 5.3: Pengujian Fungsional pada Tampilan "game over"

Kasus uji	Hasil yang diharapkan	Hasil uji
Tombol "enter" ditekan	Pemain akan diarahkan ke tampilan menu utama	Hasil pengujian sesuai dengan yang diharapkan

1468 Berdasarkan tabel 5.3, dapat disimpulkan bahwa kasus uji pada tampilan "game over" mem-
1469 bawakan hasil sesuai dengan yang diharapkan.

BAB 6

KESIMPULAN DAN SARAN

Pada bab ini berisi kesimpulan dari pembangunan permainan dan saran untuk pengembangan permainan ini.

6.0.1 Kesimpulan

Dari hasil pembangunan permainan Open Source Snake 360, dapat diambil beberapa kesimpulan, diantaranya adalah:

1. Pembangunan permainan Open Source Snake 360 menggunakan HTML5 berhasil. Hal ini dapat dilihat berdasarkan pengujian fungsional yang sudah dilakukan.

6.0.2 Saran

Berdasarkan kesimpulan yang telah dipaparkan, terdapat beberapa saran yang dapat digunakan untuk pengembangan permainan ini. Berikut adalah saran-saran yang ada:

1. Tidak ada

DAFTAR REFERENSI

- [1] Fulton, S. dan Fulton, J. (2013) *HTML5 canvas: native interactivity and animation for the web*. " O'Reilly Media, Inc."
- [2] MDN (2005) Web technology for developers. <https://developer.mozilla.org/en-US/docs/Web>. 17 Oktober 2018.
- [3] Duckett, J. (2014) *JavaScript and JQuery: interactive front-end web development*. Wiley Publishing.
- [4] Chacon, S. dan Straub, B. (2014) *Pro git*. Apress.

LAMPIRAN A

KODE PROGRAM

Listing A.1: index.html

```
1 <!DOCTYPE html>
2 <html>
3   <title></title>
4   <head></head>
5   <style>
6     body {
7       background: black;
8       display: flex;
9       align-items: center;
10      justify-content: center;
11    }
12    canvas{
13      padding: 0;
14      margin: auto;
15      display: block;
16      position: absolute;
17      top: 0;
18      bottom: 0;
19      left: 0;
20      right: 0;
21    }
22    #menuDiv{
23      padding : 35px;
24      margin:auto;
25      display:block;
26      width:400px;
27      height:400px;
28      position:absolute;
29      border : 10px solid green;
30      color: black;
31      background-color: aquamarine;
32      text-align:center;
33      line-height:0.8cm;
34      top: 0;
35      bottom: 0;
36      left: 0;
37      right: 0;
38      font-family : Impact, Charcoal, sans-serif;
39    }
40    #scoreText{
41      color : white;
42      position : fixed;
43      bottom : 3px;
44      right : 200px;
45      font-family: Arial, Helvetica, sans-serif;
46      font-size : 35px;
47    }
48    #gameOver{
49      color : yellow;
50      position : relative;
51      display : block;
52      font-family: Arial, Helvetica, sans-serif;
53      visibility: hidden;
54      top : 200px;
55      text-align :center;
56    }
57    #levelText{
58      color : white;
59      position : fixed;
60      bottom : 3px;
61      left : 200px;
62      font-family: Arial, Helvetica, sans-serif;
63      font-size : 35px;
64    }
65    input[type=number]{
66      width : 60px;
67      font-size :25px;
68      text-align : center;
69    }
70    #speedInvalid, #levelInvalid{
71      visibility : hidden;
72      color : red;
73      font-size : 25px;
74    }
75    p,span,input[type=button]{
```

```

76         font-size : 25px;
77     }
78     #maze{
79         background : olive;
80     }
81
82 </style>
83 <body>
84     <canvas id="maze">
85         <!-- Insert fallback content here -->
86     </canvas>
87     <canvas id="snake">
88         <!-- Insert fallback content here -->
89     </canvas>
90     <div id="menuDiv">
91         <p>CHOOSE LEVEL (1 - <span id="totalLevel">2</span>)</p> <input type="number" value="1" id="level">
92         <p id="levelInvalid">LEVEL IS INVALID</p>
93         <p>TURNING SPEED (1:slowest - 10:fastest)</p> <input type="number" value="1" id="speed">
94         <p id="speedInvalid">TURNING SPEED IS INVALID</p>
95         <input type="button" value="OK" id="ok">
96     </div>
97
98     <h1 id="scoreText">Score : <span id='score' style="font-size:35px">0</span></h1>
99     <h1 id="levelText">Level <span id='levels' style="font-size:35px">--</span></h1>
100     <div id="gameOver">
101         <p>GAME OVER!</p>
102         <p>Press Enter button to try again</p>
103         <p>Your final score is : <span id="finalScore">0</span></p>
104     </div>
105     <script src="jquery-3.2.1.min.js"></script>
106     <script src="DrawingObject.js" type="text/javascript"></script>
107     <script src="Apple.js" type="text/javascript"></script>
108     <script src="Snake.js" type="text/javascript"></script>
109     <script src="Maze.js" type="text/javascript"></script>
110 </body>
111 </html>
112 <script type="text/javascript">
113     const CANVAS_SIZE = 600;
114     const GRID_SIZE = 10;
115
116     const BESAR_ULAR = GRID_SIZE;
117     const BESAR_APEL = GRID_SIZE*2;
118     const BESAR_DINDING = GRID_SIZE;
119     //kelas Game
120     // 900 375
121     function Game(){
122         this.mazeCanvas = document.getElementById('maze');
123         this.contextMaze = this.mazeCanvas.getContext('2d');
124         this.myCanvas = document.getElementById('snake');
125         this.context = this.myCanvas.getContext('2d');
126
127         this.mazeCanvas.width = CANVAS_SIZE;
128         this.mazeCanvas.height = CANVAS_SIZE;
129         this.myCanvas.width = CANVAS_SIZE;
130         this.myCanvas.height = CANVAS_SIZE;
131         this.mazeCanvas.style.border = "5px_solid_white";
132         this.myCanvas.style.border = "5px_solid_white";
133
134         this.sudut = 0;
135         var score = 0;
136         this.gameOver = false;
137         this.turningSpeed;
138
139         this.apel = new Apple(BESAR_APEL);
140         this.ular = new Snake(BESAR_ULAR);
141         this.maze = new Maze(BESAR_DINDING);
142         this.drawingObj = new DrawingObject(this.context,this.contextMaze);
143
144         //method untuk random posisi apel
145         this.randomApple = function(){
146             const width = this.myCanvas.width-this.apel.getBesarApel();
147             const height = this.myCanvas.height-this.apel.getBesarApel();
148
149             let randomX = Math.floor(Math.random()*width);
150             let randomY = Math.floor(Math.random()*height);
151
152             this.moveApple(randomX,randomY);
153
154             let arrayLayout = this.maze.getMazeLayout();
155             let dindingX = Math.floor(randomX/10);
156             let dindingY = Math.floor(randomY/10);
157
158             for(var i = 0; i< this.ular.tubuhSnake.length-1;i++){
159                 let posisiXUlar = this.ular.tubuhSnake[i].x;
160                 let posisiYUlar = this.ular.tubuhSnake[i].y;
161
162                 if(posisiXUlar > randomX && posisiYUlar > randomY &&
163                    posisiXUlar < randomX+this.apel.besarApel && posisiYUlar < randomY+this.apel.besarApel){
164                     if(arrayLayout[dindingY].charAt(dindingX) == '#'){
165                         this.moveApple(randomX,randomY);
166                     }
167                 }
168                 else{
169                     break;
170                 }
171             }
172
173             this.drawingObj.drawApple(this.apel.x,this.apel.y,this.apel.getBesarApel());
174         }

```

```

175 |
176 | //method untuk memindahkan apel
177 | this.moveApple = function(x,y){
178 |     this.apel.x = x;
179 |     this.apel.y = y;
180 | }
181 |
182 | //method untuk mengarahkan ular ke atas,bawah,kiri,kanan
183 | this.moveLeft = function(){
184 |     this.sudut-=this.turningSpeed;
185 |     console.log(this.sudut);
186 |     if(this.sudut < 0){
187 |         this.sudut = 360;
188 |     }
189 | }
190 | this.moveRight = function(){
191 |     this.sudut+=this.turningSpeed;
192 |     console.log(this.sudut);
193 |     console.log(typeof(this.turningSpeed));
194 |     if(this.sudut > 360){
195 |         this.sudut = 0;
196 |     }
197 | }
198 |
199 |
200 | //method supaya ular dapat keluar dari sisi lain
201 | this.sampaiUjung = function(){
202 |     if (this.ular.x < 0) {
203 |         this.ular.x = this.myCanvas.width;
204 |     }
205 |     else if (this.ular.x >= this.myCanvas.width) {
206 |         this.ular.x = 0;
207 |     }
208 |     if (this.ular.y < 0) {
209 |         this.ular.y = this.myCanvas.height;
210 |     }
211 |     else if (this.ular.y >= this.myCanvas.height) {
212 |         this.ular.y = 0;
213 |     }
214 | }
215 |
216 | //untuk menghilangkan text gameOver
217 | this.removeGameOverText = function(){
218 |     let gameOverText = document.getElementById("gameOver");
219 |     gameOverText.style.visibility = "hidden";
220 | }
221 |
222 | //memulai game
223 | this.startGame = function(kelas){
224 |     kelas.removeGameOverText();
225 |
226 |     let level = $('#level').val();
227 |     $('#levels').html(level);
228 |     let speed = $('#speed').val();
229 |     kelas.turningSpeed = parseInt(speed);
230 |
231 |     kelas.ular.x = 0;
232 |     kelas.ular.y = 200;
233 |     kelas.randomApple();
234 |     kelas.drawingObj.drawSnake(kelas.ular.tubuhSnake, kelas.ular.besarUlar);
235 |     kelas.drawingObj.drawMaze(kelas.maze.getMazeLayout(), kelas.maze.getBesarDinding());
236 | }
237 |
238 | //cek collision ular dengan apel
239 | this.appleCollision = function(){
240 |     let posisiApelX = this.apel.x;
241 |     let posisiApelY = this.apel.y;
242 |     let posisiXUlar = this.ular.tubuhSnake[0].x;
243 |     let posisiYUlar = this.ular.tubuhSnake[0].y;
244 |
245 |     if(posisiXUlar > posisiApelX && posisiYUlar > posisiApelY &&
246 |        posisiXUlar < posisiApelX+this.apel.besarApel && posisiYUlar < posisiApelY+this.apel.besarApel){
247 |         this.appleEaten();
248 |     }
249 | }
250 |
251 | // ular memakan apel
252 | this.appleEaten = function(){
253 |     this.randomApple();
254 |     score++;
255 |     this.ular.panjangTubuhSaatIni++;
256 |     $('#score').html(score);
257 | }
258 |
259 | //ular menabrak diri sendiri
260 | this.snakeCollision = function(){
261 |     let posisiXUlar = this.ular.tubuhSnake[0].x;
262 |     let posisiYUlar = this.ular.tubuhSnake[0].y;
263 |     const besarBoundary = 1;
264 |
265 |     for(var i = 1; i< this.ular.tubuhSnake.length;i++){
266 |         let bagianTubuhSnakeX = this.ular.tubuhSnake[i].x;
267 |         let bagianTubuhSnakeY = this.ular.tubuhSnake[i].y;
268 |
269 |         if(posisiXUlar > bagianTubuhSnakeX-besarBoundary && posisiYUlar > bagianTubuhSnakeY-besarBoundary &&
270 |            posisiXUlar < bagianTubuhSnakeX+besarBoundary && posisiYUlar < bagianTubuhSnakeY+besarBoundary ){
271 |             this.gameOver = true;
272 |             this.popUpGameOverText();
273 |         }

```

```

274     }
275 }
276
277 //ular menabrak dinding
278 this.wallCollision = function(){
279     let arrayLayout = this.maze.getMazeLayout();
280     let posisiXUlar = this.ular.tubuhSnake[0].x;
281     let posisiYUlar = this.ular.tubuhSnake[0].y;
282     let boundaryWallX = Math.floor(posisiXUlar/10);
283     let boundaryWallY = Math.floor(posisiYUlar/10);
284
285     if(arrayLayout[boundaryWallY].charAt(boundaryWallX) == '#'){
286         this.gameOver = true;
287         this.popUpGameOverText();
288     }
289 }
290
291 //untuk menampilkan text gameOver
292 this.popUpGameOverText = function(){
293     let gameOverText = document.getElementById("gameOver");
294     $("#finalScore").html(score);
295     gameOverText.style.visibility = "visible";
296 }
297
298
299 //mendapatkan status gameOver
300 this.getGameOverStatus = function(){
301     return this.gameOver;
302 }
303
304
305 //posisi untuk animasi
306 this.animate = function(){
307     if(this.gameOver == true){
308     }
309     else{
310         this.context.clearRect(0,0,this.myCanvas.width,this.myCanvas.height);
311         this.ular.move(this.sudut);
312
313         this.sampaiUjung();
314
315         this.temp = {};
316         this.temp['x'] = this.ular.x;
317         this.temp['y'] = this.ular.y;
318
319         this.ular.tubuhSnake.unshift(this.temp);
320         if(this.ular.tubuhSnake.length > this.ular.panjangTubuhSaatIni){
321             this.ular.tubuhSnake.pop();
322         }
323
324         this.drawingObj.drawSnake(this.ular.tubuhSnake,this.ular.besarUlar);
325         this.drawingObj.drawApple(this.apel.x,this.apel.y,this.apel.getBesarApel());
326         this.appleCollision();
327         this.snakeCollision();
328         this.wallCollision();
329
330         var that = this;
331         setTimeout(function(){
332             that.animate();
333         },50);
334     }
335 }
336
337
338 this.checkLevel = function(){
339     let chosenLevel = document.getElementById("level").value;
340     let temp = $("#totalLevel").html();
341     let totalLevel = parseInt(temp);
342
343     if(chosenLevel != ""){
344         if(chosenLevel > 0 && chosenLevel <= totalLevel){
345             document.getElementById("levelInvalid").style.visibility = "hidden";
346             return true;
347         }
348         else{
349             document.getElementById("levelInvalid").style.visibility = "visible";
350             return false;
351         }
352     }
353     else{
354         document.getElementById("levelInvalid").style.visibility = "visible";
355         return false;
356     }
357 }
358
359
360 this.checkSpeed = function(){
361     let chosenSpeed = document.getElementById("speed").value;
362
363     if(chosenSpeed != ""){
364         if(chosenSpeed > 0 && chosenSpeed <= 10){
365             document.getElementById("speedInvalid").style.visibility = "hidden";
366             return true;
367         }
368         else{
369             document.getElementById("speedInvalid").style.visibility = "visible";
370             return false;
371         }
372     }

```

```

373     else{
374         document.getElementById("speedInvalid").style.visibility = "visible";
375         return true;
376     }
377 }
378
379 this.changeLevel = function(levels){
380     $("#totalLevel").html(levels);
381     return levels;
382 }
383
384 this.countLevels = function(callback){
385     let level = 0;
386
387     $.ajax({
388         url: "Levels/",
389         success: function(data){
390             $(data).find("tbody tr a").each(function(i){
391                 if(i>4){
392                     level = level+1;
393                 }
394             });
395             callback(level);
396         }
397     });
398 }
399
400 this.loadMaze = function(callback){
401     let level = $('#level').val();
402     let url = "Levels/level"+level+".txt";
403     let temp = this;
404
405     $.ajax({
406         url: "Levels/level"+level+".txt",
407         dataType: 'text',
408         context: temp,
409         success: function(data, textStatus, jqXHR) {
410             temp.maze.setMazeLayout(jqXHR.responseText);
411             callback(temp);
412         }
413     });
414 }
415
416 }
417
418 $(document).ready(function(){
419     var permanan = new Game();
420     permanan.countLevels(permanan.changeLevel);
421
422     document.getElementById('ok').addEventListener('click',function(){
423         if(permanan.checkSpeed() && permanan.checkLevel()){
424             document.getElementById('menuDiv').style.visibility = 'hidden';
425             permanan.loadMaze(permanan.startGame);
426         }
427     })
428
429     //tombol pergerakan ular
430     document.addEventListener('keydown', function(e) {
431         if (e.keyCode == 37) {
432             permanan.moveLeft();
433         }
434         else if (e.keyCode == 39) {
435             permanan.moveRight();
436         }
437
438         else if(e.keyCode == 13 && permanan.getGameOverStatus() == true){
439             document.location.href="";
440         }
441     });
442
443     document.addEventListener('touchstart',function(e){
444         var width = $(document).width();
445         var clickX = e.clientX;
446         if(clickX > width/2){
447             permanan.moveLeft();
448         }
449         else{
450             permanan.moveRight();
451         }
452     });
453
454     function wait(){
455         if(permanan.maze.getMazeLayout() == null){
456             setTimeout(function(){
457                 wait();
458             },50);
459         }
460         else{
461             permanan.animate();
462         }
463     }
464
465     wait();
466 });
467
468 </script>
469

```

Listing A.2: Snake.js

```

1 function Snake(besarUlar){
2     this.x;
3     this.y;
4     this.panjangTubuhSaatIni = 15;
5     this.speed = 2;
6     this.tubuhSnake = [{x:this.x,y:this.y}];
7     this.besarUlar = besarUlar;
8
9     this.move = function(sudut){
10         this.x += Math.cos(sudut*Math.PI/180)*this.speed;
11         this.y += Math.sin(sudut*Math.PI/180)*this.speed;
12     }
13 }

```

Listing A.3: Apple.js

```

1 function Apple(besarApel){
2     this.x;
3     this.y;
4     this.besarApel = besarApel;
5
6     this.getBesarApel = function(){
7         return this.besarApel;
8     }
9 }

```

Listing A.4: Maze.js

```

1 function Maze(besarDinding){
2     this.besarDinding = besarDinding;
3     this.mazeLayout = null;
4
5     this.setMazeLayout = function(layoutInText){
6         var lines = layoutInText.split('\n');
7         this.mazeLayout = [];
8         for ( var i = 0 ; i < lines.length ; i++ ) {
9             this.mazeLayout[i] = lines[i];
10        }
11    }
12
13    this.getMazeLayout = function(){
14        return this.mazeLayout;
15    }
16
17    this.getBesarDinding = function(){
18        return this.besarDinding;
19    }
20
21 }

```

Listing A.5: DrawingObject.js

```

1 function DrawingObject(context,contextMaze){
2     this.context = context;
3     this.contextMaze = contextMaze;
4
5     //untuk gambar apel
6     this.drawApple = function(x,y,besarApel){
7         this.context.lineWidth = 1;
8         this.context.strokeStyle = 'white';
9         this.context.beginPath();
10        this.context.moveTo(x+(besarApel/2),y+5);
11        this.context.quadraticCurveTo(x+besarApel,y,x+besarApel,y+(besarApel/2));
12        this.context.quadraticCurveTo(x+(besarApel-2),y+besarApel,x+(besarApel/2),y+(besarApel-2));
13        this.context.quadraticCurveTo(x+2,y+besarApel,x,y+(besarApel/2));
14        this.context.quadraticCurveTo(x,y,x+(besarApel/2),y+5);
15        this.context.closePath();
16
17        this.context.fillStyle = 'red';
18        this.context.fill();
19        this.context.moveTo(x+10,y+5);
20        this.context.lineTo(x+10,y);
21        this.context.closePath();
22        this.context.stroke();
23    }
24
25    //untuk gambar ular
26    this.drawSnake = function(arrayUlar,besarUlar){
27        this.context.lineWidth = besarUlar;
28        this.context.strokeStyle = 'lawngreen';
29
30        for(var i = 0;i< arrayUlar.length-1;i++){
31            if(Math.abs(arrayUlar[i].x - arrayUlar[i+1].x) > 2||
32               Math.abs(arrayUlar[i].y - arrayUlar[i+1].y) > 2){
33                i++;
34            }
35            else{
36                this.context.lineWidth = besarUlar;
37                this.context.strokeStyle = 'lawngreen';
38
39                if(i > 5){
40                    this.context.strokeStyle = 'limegreen';
41                }
42            }
43        }
44    }
45 }

```



```

42|
43|         this.context.beginPath();
44|         this.context.moveTo(arrayUlar[i].x,arrayUlar[i].y);
45|         this.context.lineTo(arrayUlar[i+1].x,arrayUlar[i+1].y);
46|         this.context.closePath();
47|         this.context.stroke();
48|
49|         if(i == 2){
50|             this.context.strokeStyle = 'red';
51|             this.context.lineWidth = 3;
52|             this.context.beginPath();
53|             this.context.moveTo(arrayUlar[i].x,arrayUlar[i].y);
54|             this.context.lineTo(arrayUlar[i+1].x,arrayUlar[i+1].y);
55|             this.context.closePath();
56|             this.context.stroke();
57|         }
58|     }
59| }
60|
61| }
62|
63| //untuk gambar maze
64| this.drawMaze = function(arrayLayout,besarDinding){
65|     this.contextMaze.lineWidth = besarDinding;
66|     this.contextMaze.fillStyle = 'darkslategrey';
67|
68|     for(var i = 0;i< arrayLayout.length; i++){
69|         let temp = arrayLayout[i];
70|         for(var j = 0;j< arrayLayout.length;j++){
71|             if(temp.charAt(j) == '#'){
72|                 this.contextMaze.fillRect(j*besarDinding,i*besarDinding,besarDinding,besarDinding);
73|             }
74|         }
75|     }
76| }
77| }

```


LAMPIRAN B

HASIL EKSPERIMEN

Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4