

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 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. Permainan ini sudah umum dimainkan pada *web browser* dan beberapa perangkat. Umumnya pada permainan Snake, ular hanya dapat bergerak ke atas, ke bawah, ke kiri dan ke kanan saja. Selain itu labirin yang disediakan terbatas.

HTML(*Hyper Text Markup Language*) merupakan bahasa markah yang digunakan untuk membuat halaman web. HTML5 merupakan HTML versi terbaru. 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* merupakan bahasa tingkat tinggi yang digunakan untuk membuat halaman *web* menjadi lebih interaktif dan *jQuery* merupakan library milik *Javascript* yang kaya fitur. *Github* adalah layanan 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.

Open Source Snake 360 adalah sebuah permainan yang dibuat menggunakan HTML5 dan *Javascript*. *jQuery* digunakan untuk mengakses labirin dan memuat labirin dari server. Pada permainan ini, ular sudah dapat bergerak ke segala arah dan orang lain dapat menambahkan labirin buatan sendiri. Orang lain dapat menambahkan labirin buatan sendiri dengan menggunakan *pull request* pada *Github*. Permainan ini juga sudah dapat memuat labirin-labirin yang dibuat oleh orang lain.

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

ABSTRACT

This study discusses the construction of the Open Source Snake 360 game. This game is based on the references of the existing Snake game. Snake game is a game in which players control the movement of snakes to get food scattered in the maze. Every snake eats food, the player gets a score. In this game, players must control the snake to get as much food as possible without crashing into the wall of the labyrinth or himself. This game is commonly played on web browser and some devices. Generally speaking in Snake game, the snake can only move up, down, left and right. In addition, the labyrinth provided is limited.

HTML(Hyper Text Markup Language) is the marking language used to create web pages. HTML5 is the latest version of HTML. HTML5 has several new elements, one of which is HTML5 Canvas. HTML5 Canvas is a place to draw pixels that can be written using the Javascript programming language. Javascript is a high-level language used to make web pages more interactive and jQuery is a feature-rich Javascript library. Github is a shared hosting service for software development projects that uses the version control system, Git. With Github, the programmer can find out the changes in the repository.

Open Source Snake 360 is a game created using HTML5 and Javascript. jQuery is used to access and load the maze from the server. In this game, snake can move in any direction and others can add homemade mazes. Other people can add homemade mazes by using pull request on Github. This game can also load mazes made by other people.

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	8
2.3.2 <i>Constant</i>	8
2.3.3 <i>Function</i>	9
2.3.4 Menggambar pada <i>Canvas</i>	9
2.3.5 <i>Object Oriented Programming Javascript</i>	12
2.3.6 <i>Event</i>	14
2.3.7 Membuat Animasi	17
2.4 <i>jQuery</i>	18
2.4.1 Mendapatkan dan Mengubah Konten Elemen	19
2.4.2 Mendapatkan dan Mengubah Properti CSS	20
2.4.3 <i>Looping</i>	21
2.4.4 Event	21
2.4.5 AJAX	23
2.5 <i>Git</i>	25
2.5.1 <i>Version Control</i>	25
2.5.2 <i>Git</i>	27
2.5.3 <i>Git Branching</i>	31
2.5.4 <i>GitHub</i>	37
3 ANALISIS	39
3.1 Analisis Permainan <i>Snake</i> yang Sudah Ada	39
3.1.1 Ular dan Makanan	39
3.1.2 Pergerakan Ular	41
3.1.3 Labirin	41
3.2 Analisis Sistem yang Dibangun	42

3.2.1	Menentukan Besar <i>Canvas</i>	42
3.2.2	Menggambar Ular dan Apel	42
3.2.3	Pergerakan Ular	45
3.2.4	Mengacak posisi apel	46
3.2.5	Menggambar Labirin	47
3.2.6	Pengecekan tabrakan(<i>Collision Detection</i>)	48
3.3	Analisis Berorientasi Objek	50
3.3.1	Skenario Permainan	50
3.3.2	Diagram Kelas	51
4	PERANCANGAN	55
4.1	Rancangan Diagram Sequence	55
4.1.1	Memuat Labirin	55
4.2	Rancangan Diagram Kelas Rinci	56
4.3	Rancangan Tampilan Antarmuka	59
4.3.1	Tampilan Menu Utama	59
4.3.2	Tampilan Bermain	60
4.3.3	Tampilan Permainan Berakhir	61
5	IMPLEMENTASI DAN PENGUJIAN	63
5.1	Implementasi	63
5.1.1	Lingkungan Perangkat Keras	63
5.1.2	Lingkungan Perangkat Lunak	64
5.1.3	Implementasi Antarmuka	64
5.2	Pengujian	69
5.2.1	Pengujian Fungsional	69
5.2.2	Pengujian Eksperimental	70
6	KESIMPULAN DAN SARAN	73
6.1	Kesimpulan	73
6.2	Saran	73
	DAFTAR REFERENSI	75
	A KODE PROGRAM	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>	10
2.4	Perbedaan <i>quadratic Bézier curve</i> dan <i>cubic Bézier curve</i>	12
2.5	Local Version Control	26
2.6	Centralized Version Control	27
2.7	Distributed Version Control	27
2.8	Working tree, staging area, dan Git directory	28
2.9	Siklus hidup pada status <i>file</i>	30
2.10	Commit dan tree dari file yang dicommit	31
2.11	Commit dan parent dari commit	32
2.12	<i>Pointer HEAD</i> menunjuk <i>branch master</i>	32
2.13	<i>Pointer HEAD</i> beserta <i>branch testing</i>	33
2.14	3 <i>snapshot</i> yang digunakan dalam <i>three way merge</i>	33
2.15	<i>Merge commit</i>	34
2.16	Perbedaan pada <i>branch</i> lokal dan <i>remote</i>	35
2.17	<i>Update remote-tracking branches</i> menggunakan perintah <i>git fetch</i>	35
2.18	<i>Rebasing commit</i> C4 ke C3	36
2.19	<i>Merge branch</i> setelah <i>rebasing</i>	37
2.20	Tombol 'Fork'	37
3.1	Ular pada <i>Slither.io</i>	40
3.2	Makanan pada <i>Slither.io</i>	40
3.3	Ular pada <i>Snake Nokia</i>	40
3.4	Makanan biasa(A) dan makanan bonus(B) pada <i>Snake Nokia</i>	40
3.5	Ular sedang melaju dengan cepat(<i>speed up</i>)	41
3.6	Peta labirin pada <i>Slither.io</i>	41
3.7	Koordinat bagian tubuh ular pada <i>array</i>	43
3.8	Tubuh ular setelah digambar menggunakan garis	43
3.9	Bagian pada apel(lingkaran merah) yang akan dibuat menggunakan kurva	43
3.10	Pembagian gambar apel dengan layout persegi beserta ukuran pada setiap bagian	44
3.11	<i>Start point</i> , <i>control point</i> dan <i>end point</i> untuk menggambar apel bagian kiri atas	44
3.12	<i>Start point</i> , <i>control point</i> dan <i>end point</i> untuk menggambar apel bagian kiri bawah	45
3.13	Ilustrasi ular sebelum bergerak maju(A) dan setelah bergerak maju(B)	45
3.14	Gambar apel yang terpotong sesudah mengacak posisi apel	47
3.15	Menggambar dinding menggunakan simbol pada file text	48
3.16	Ular ingin melewati jalur yang diapit oleh 2 buah dinding	48
3.17	Daerah tabrakan pada apel	49
3.18	Daerah tabrakan berbentuk persegi pada apel	49
3.19	Posisi kepala ular pada sebuah daerah labirin	50
3.20	Diagram <i>use case</i> dari permainan <i>Snake 360</i>	51
3.21	Diagram class dari permainan <i>Snake 360</i>	52

4.1	Diagram <i>sequence</i> untuk memuat labirin	55
4.2	Diagram class rinci dari Open Source <i>Snake 360</i>	56
4.3	Rancangan tampilan menu utama	60
4.4	Rancangan tampilan menu utama jika pemain salah memasukkan data	60
4.5	Rancangan tampilan bermain	61
4.6	Rancangan tampilan permainan berakhir	61
5.1	Tampilan menu utama pada <i>desktop</i>	65
5.2	Tampilan menu utama pada <i>smartphone</i>	65
5.3	Tampilan menu utama jika pemain salah memasukkan data <i>level</i> labirin	66
5.4	Tampilan bermain pada <i>desktop</i>	66
5.5	Tampilan bermain pada <i>smartphone</i>	67
5.6	Tampilan permainan berakhir pada desktop	68
5.7	Tampilan permainan berakhir pada smartphone	68
5.8	Tampilan hasil pull request milik penguji 1	70
5.9	Tampilan pengujian labirin yang dibuat oleh penguji 1	71
5.10	Tampilan hasil pull request milik penguji 2	71
5.11	Tampilan pengujian labirin yang dibuat oleh penguji 2	72

DAFTAR TABEL

5.1	Pengujian Fungsional pada Tampilan Menu Utama	69
5.2	Pengujian Fungsional Tampilan Bermain pada Desktop	69
5.3	Pengujian Fungsional pada Tampilan " <i>Game Over</i> "	70

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 Javascript* yang cepat, kecil dan kaya dengan fitur. *jQuery* membuat hal-hal seperti traversal dan manipulasi dokumen HTML, penanganan *event*, animasi dan *Ajax* jauh lebih sederhana dengan API(*Application Programming Interface*) yang mudah untuk digunakan pada banyak *browser*. *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:

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

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

-
- 1 2. Bab 2 berisikan dasar-dasar teori yang menunjang penelitian ini. Teori yang digunakan adalah:
2 pengertian *Snake*, HTML5 Canvas, *Javascript*, *jQuery*, dan *Git*.
 - 3 3. Bab 3 berisikan analisis sistem yang sudah ada, analisis sistem yang dibangun dan analisis
4 berorientasi objek.
 - 5 4. Bab 4 berisikan perancangan perangkat lunak yang dibangun. Perancangan yang dilakukan
6 meliputi perancangan diagram *sequence*, perancangan diagram kelas dan perancangan tampilan
7 antarmuka.
 - 8 5. Bab 5 berisikan implementasi dan pengujian perangkat lunak.
 - 9 6. Bab 6 berisikan kesimpulan dan saran.

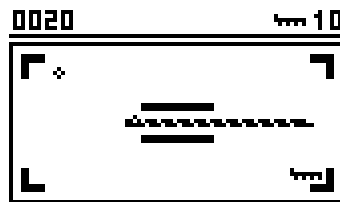
BAB 2

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.

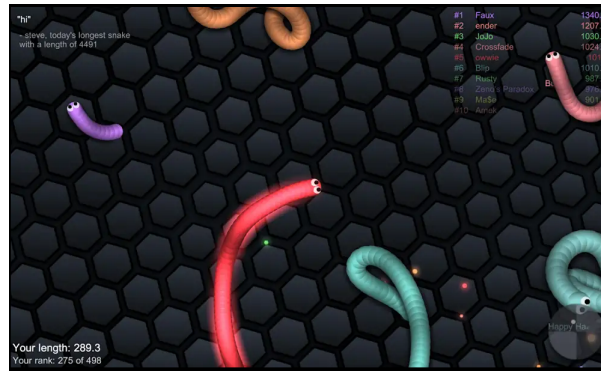
Permainan *Snake* ini dapat dimainkan secara *singleplayer* atau *multiplayer*. *Singleplayer game* adalah permainan yang dapat dimainkan oleh 1 pemain. *Multiplayer game* adalah permainan yang dapat dimainkan oleh beberapa pemain. Pada umumnya, permainan *Snake* dimainkan secara *singleplayer*. Contoh *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* sudah 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 juga 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*). Pada *Listing 2.2* dan *Listing 2.3* terdapat potongan kode untuk menambah *script* secara langsung di halaman *web* dan menambah *script* secara terpisah.

```

<!DOCTYPE html>
<html>
  <body>

    <h1>A Web Page</h1>
    <p id="demo">A Paragraph</p>

    <script>
      // tuliskan script di sini
    </script>

  </body>
</html>

```

Listing 2.2: *Internal Javascript*

```

<!DOCTYPE html>
<html>
  <body>

```

```

15 |         <h1>A Web Page</h1>
16 |         <p id="demo">A Paragraph</p>
37 |
48 |         <script src="myScript1.js"></script>
59 |
60 |     </body>
61 | </html>

```

Listing 2.3: *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 karena tipe data variabel akan otomatis mengikuti tipe data nilai yang diassign ke variabel tersebut. Pada *listing 2.4* terdapat potongan kode untuk mendeklarasikan variabel.

```

151 |     var myVariable;

```

Listing 2.4: Deklarasi variabel

Nilai variabel pada *listing ??* adalah *undefined* karena variabel tersebut tidak diberi nilai/*value*. Pada *listing 2.5* terdapat potongan kode untuk mengisi nilai pada variabel.

```

191 |     myVariable = 3;

```

Listing 2.5: 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 2.6* terdapat potongan kode untuk mendeklarasi *constant*.

```

311 |     const myConst = 1;

```

Listing 2.6: 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 koma. *Function* bisa memiliki parameter atau tidak. Pada *Listing 2.7* terdapat potongan kode untuk membuat *function* penjumlahan 2 buah bilangan.

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

Listing 2.7: *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 2.8* terdapat potongan kode untuk memanggil *function* dengan nama penjumlahan.

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

Listing 2.8: Memanggil *function* penjumlahan

2.3.4 Menggambar pada Canvas

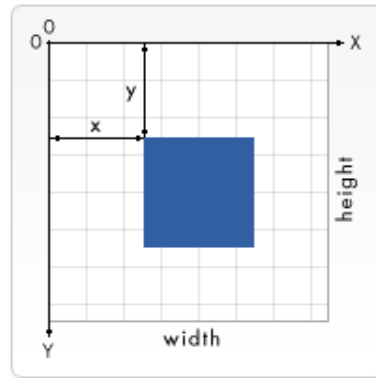
Sesudah menuliskan tag `<canvas>` pada HTML, *canvas* tidak bisa langsung digambar. Karena itu perlu ditambahkan *drawing context* pada *Javascript*. Pada *listing 2.9* terdapat potongan kode untuk menambahkan *drawing context*.

```
var myCanvas = document.getElementById('canvas');  
var context = myCanvas.getContext('2d');
```

Listing 2.9: Menambahkan *drawing context canvas*

Berdasarkan *listing 2.9*, 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 Gambar 2.3 di atas, titik ujung kiri kotak biru tersebut berjarak x *pixel* dari sumbu y dan berjarak y *pixel* dari sumbu x .

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 *render* 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 2.10](#) merupakan fungsi *moveTo()*.

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

Listing 2.10: 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 2.11](#) merupakan fungsi *lineTo()*.

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

Listing 2.11: 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 2.12](#) merupakan fungsi *arc()*.

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

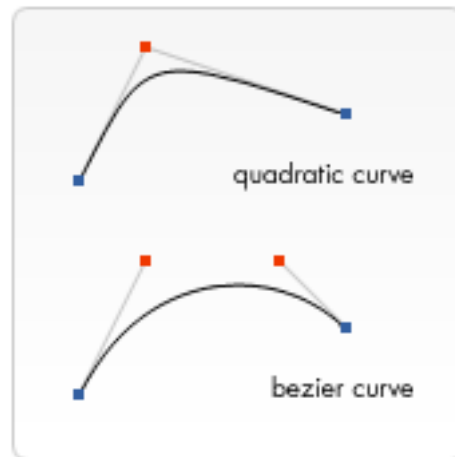
Listing 2.12: Fungsi *arc()*

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 :

$$radian = (Math.PI/180) * besarsudut$$

Bézier curve merupakan tipe *path* yang digunakan untuk membuat kurva. *Bézier curve* ada 2 jenis yaitu *cubic* dan *quadratic*. Perbedaananya adalah *quadratic Bézier curve* memiliki sebuah *control point*, sedangkan *cubic Bézier curve* memiliki 2 buah *control point*. Pada [Gambar 2.4](#)

- 1 menunjukkan perbedaan antara *quadratic Bézier curve* dan *cubic Bézier curve*. Titik merah pada
- 2 gambar merupakan *control point* dari *Bézier curve*.



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

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

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

8 2.3.5 Object Oriented Programming Javascript

9 OOP (*Object Oriented Programming*) adalah sebuah paradigma *programming* yang menggunakan

10 abstraksi untuk membuat objek-objek yang ada pada dunia nyata. Bahasa pemrograman seperti

11 *Java*, *C++*, *Ruby*, *Phyton*, *PHP*, dan *Objective-C* sudah mendukung OOP. Dalam OOP, setiap

12 objek dapat menerima pesan, memproses data dan mengirim pesan ke objek lain. Program yang

13 menggunakan konsep OOP ini mudah untuk dimengerti dan lebih mudah untuk dikembangkan oleh

14 *programmer*.

15

16 Ide umum pada OOP adalah menggunakan objek untuk memodelkan benda-benda yang ada

17 pada dunia nyata. Objek tersebut kemudian direpresentasi pada program yang dibuat. Objek-objek

18 dapat berisi data, fungsionalitas dan *behaviour* yang merepresentasikan informasi tentang objek

19 tersebut dan tugas objek. Contohnya, membuat objek sebuah mobil. Mobil memiliki beberapa

20 informasi diantaranya adalah merk mobil, berat mobil, warna mobil dan tahun produksi. Informasi

21 tersebut dapat disebut sebagai properti dari objek. Mobil dapat bergerak maju, berbelok ke kanan,

22 berbelok ke kiri, bergerak mundur dan berhenti. Hal-hal yang dapat dilakukan oleh objek disebut

23 sebagai *method* dari objek.

Kelas

Javascript tidak memiliki *statement* 'class' yang dapat digunakan pada bahasa pemrograman C++ atau Java. Untuk membuat kelas, Javascript menggunakan *function* sebagai konstruktor untuk kelas. Karena itu, membuat kelas sama dengan membuat *function* pada Javascript. Pada *listing* 2.13 terdapat potongan kode untuk membuat kelas bernama Mobil.

```
function Mobil(){  
}
```

Listing 2.13: Membuat kelas Mobil

Objek

Untuk membuat instansi baru dari objek, gunakan *statement* 'new' yang nantinya akan disimpan pada variabel. Pada *listing* 2.14 terdapat potongan kode untuk membuat instansi.

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

Listing 2.14: Membuat instansi mobil

Konstruktor

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

Properti/Atribut

Properti adalah variabel yang terdapat pada kelas. Properti ditulis pada konstruktor kelas sehingga setiap properti pada kelas akan dibuat ketika membuat instansi baru. Untuk membuat properti, gunakan *statement* 'this'. Cara ini mirip dengan bahasa pemrograman Java ketika membuat sebuah properti pada objek. Sintaks untuk mengakses properti di luar kelas adalah : namaInstansi.properti. Pada *listing* 2.15 terdapat potongan kode untuk mendefinisikan properti pada kelas Mobil pada saat instansiasi.

```
function Mobil(merkMobil,beratMobil,warnaMobil,tahunProduksi){  
    this.merkMobil = merkMobil;  
    this.beratMobil = beratMobil; //satuan dalam kg  
    this.warnaMobil = warnaMobil;  
    this.tahunProduksi = tahunProduksi;  
}
```

```
var mobil1 = new Mobil('Toyota',1000,'Hitam',2010);
```

Listing 2.15: Mendefinisikan properti pada kelas Mobil

2 Method

Method adalah hal yang dapat dilakukan oleh sebuah objek. Untuk membuat *method*, tuliskan nama *method* terlebih dahulu kemudian *assign* fungsi pada nama *method* tersebut. Untuk memanggil *method* sebuah objek, tuliskan nama objek/kelas terlebih dahulu, kemudian tuliskan nama *method* sesuai dengan yang sudah dibuat beserta tanda kurung. Tanda kurung berisi parameter. Pada *listing 2.16* terdapat potongan kode untuk membuat dan memanggil *method* *bergerakMaju()* pada kelas Mobil.

```
function Mobil(merkMobil,beratMobil,warnaMobil,tahunProduksi){
    this.merkMobil = merkMobil;
    this.beratMobil = beratMobil; //satuan dalam kg
    this.warnaMobil = warnaMobil;
    this.tahunProduksi = tahunProduksi;

    this.bergerakMaju = function(){
        //kode agar mobil bergerak maju
    }
}

var mobil1 = new Mobil('Toyota',1000,'Hitam',2010);
mobil1.bergerakMaju(); //memanggil fungsi untuk bergerak maju
```

Listing 2.16: 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 jika 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* ditembakkan 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 *submit*.

- 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 cara menggunakan *event*, pada *listing 2.17* 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;
  }
</script>

```

Listing 2.17: Menambahkan event pada button

Berdasarkan *listing 2.17*, 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 fungsi akan dieksekusi 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 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 2.18* terdapat potongan kode untuk menggunakan fungsi *addEventListener()*.

```

var btn = document.getElementById('tombol');

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

btn.addEventListener('click', bgChange);

```

Listing 2.18: Menggunakan fungsi *addEventListener()*

Pada fungsi *addEventListener()*, ada 2 buah parameter yaitu *event* yang ingin digunakan (dalam potongan kode di atas menggunakan *event click*) dan nama fungsi sebagai *handler* yang ingin dijalankan ketika *event* tersebut terjadi. Selain cara di atas, dapat juga menuliskan semua kode di dalam fungsi anonim *addEventListener()* seperti potongan kode pada *listing 2.19*.

```

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

```

Listing 2.19: Menuliskan kode di dalam fungsi anonim pada *method addEventListener()*

Event tidak hanya digunakan untuk *browser* pada *desktop* saja. Ada *event* yang dapat digunakan pada *smartphone*, yaitu *touch event*. *Touch event* dapat menginterpretasi aktivitas jari atau *stylus*

pada permukaan layar seperti layar sentuh atau *trackpads*. *Interface* pada touch event merupakan API tingkat rendah yang dapat digunakan untuk aplikasi yang mendukung interaksi *multi-touch* seperti *2-finger gesture*. Interaksi *multi touch* dimulai ketika sebuah jari menyentuh permukaan layar sentuh terlebih dahulu. Jari yang lain dapat menyentuh permukaan dan dapat menggerakkan jari di sekitar permukaan layar sentuh. Interaksi akan berakhir ketika jari tidak lagi menyentuh permukaan layar.

Touch event mirip seperti *mouse event*. Perbedaannya adalah *touch event* mendukung banyak sentuhan dalam lokasi yang berbeda pada permukaan layar. *Touch event* mengenkapsulasi semua *touch points* yang sedang aktif. *Interface touch* yang merepresentasikan sebuah *touch point* memiliki informasi seperti posisi *touch point* pada *viewport browser*.

Touch event memiliki beberapa event handler properti diantaranya adalah :

- *touchstart* : *event* ini akan ditembakkan apabila sebuah jari/*stylus* menyentuh permukaan layar.
- *touchend* : *event* ini akan ditembakkan apabila sebuah atau banyak jari tidak menyentuh permukaan layar/*trackpads*.
- *touchcancel* : *event* ini akan ditembakkan apabila sebuah atau banyak *touch points* yang terganggu dalam implementasi khusus. Contohnya adalah ketika terlalu banyak *touch points* yang dibuat.
- *touchmove* : *event* ini akan ditembakkan apabila sebuah atau banyak *touch points* yang berpindah di sekitar permukaan layar.

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 berdasarkan 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.

1 Ada 3 fungsi yang dapat digunakan untuk memanggil fungsi dalam kurun waktu tertentu diantaranya
2 adalah :

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

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

10 2.4 *jQuery*

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

```
18  
191 | $( 'li.hot' );
```

Listing 2.20: Mendapatkan elemen menggunakan *CSS-style selector*

20 Pada *listing 2.20*, *selector* akan mencari elemen ** yaitu *list* yang merupakan bagian dari
21 kelas 'hot'. *jQuery* memiliki banyak *method* yang dapat digunakan oleh elemen yang sudah dipilih
22 menggunakan *selector*. *Method* ini merepresentasikan tugas yang akan dilakukan oleh elemen
23 tersebut. Setelah memilih elemen, tambahkan *method* yang diawali dengan titik kemudian diikuti
24 dengan nama *method* beserta parameteranya. Titik ini disebut sebagai *member operator*. Setiap
25 *method* memiliki parameter untuk memberikan detail tentang bagaimana cara untuk mengubah
26 elemen tersebut. Ada beberapa *method* yang memiliki parameter lebih dari 1. Member operator
27 menunjukkan bahwa *method* yang terletak setelah *member operator* digunakan untuk mengubah
28 elemen objek *jQuery* yang terletak pada sebelah kiri *member operator*. Pada *listing 2.21*, terdapat
29 contoh untuk mengubah kelas dari elemen yang sudah dipilih. Method *addClass* digunakan untuk
30 mengubah atribut kelas dari elemen *list* menjadi kelas yang bernama 'complete'.

```
31  
321 | $( 'li.hot' ).addClass( 'complete' );
```

Listing 2.21: Mengubah kelas dari elemen yang sudah dipilih

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

1 1. Menemukan *node-node* yang sesuai di DOM *tree*

2 2. Membuat objek *jQuery*

3 3. Menyimpan referensi di *node* objek *jQuery*

4 Jika ingin menggunakan *selection* yang sama, maka lebih baik menggunakan objek *jQuery*
 5 yang sama daripada mengulang langkah-langkah yang sudah dijelaskan. Objek *jQuery* tersebut
 6 dapat disimpan pada sebuah variabel. Cara untuk menyimpan referensi objek *jQuery* tersebut
 7 pada variabel adalah membuat variabel yang diawali dengan simbol '\$'. Kemudian variabel tersebut
 8 diisi dengan objek *jQuery* yang diinginkan. Pada listing 2.22, variabel *\$listItems* menyimpan objek
 9 *jQuery* yang berisi lokasi dari semua elemen list ada pada DOM *tree*.

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

Listing 2.22: Menyimpan objek *jQuery*

12 Untuk mengecek apakah sebuah halaman sudah siap untuk menjalankan kode program yang
 13 dibuat, maka gunakan *method ready()*. Maksud dari halaman sudah siap adalah DOM sudah ada
 14 pada halaman *web*. Listing 2.23 adalah potongan kode untuk mengecek apakah halaman sudah siap.
 15 Pada listing 2.23, *\$(document)* merepresentasikan halaman *web*. Jika halaman sudah siap, maka
 16 kode program yang ada di dalam *method ready()* akan dijalankan. Listing 2.24 adalah pintasan dari
 17 *method ready()* pada objek *document*.

```
191 | $(document).ready(function(){  
202 |     //ketikan script di sini  
213 | });
```

Listing 2.23: Mengecek apakah halaman sudah siap

```
231 | $(function(){  
242 |     //ketikan script di sini  
253 | });
```

Listing 2.24: Pintasan dari *method \$(document).ready()*

26 2.4.1 Mendapatkan dan Mengubah Konten Elemen

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

- 32 1. *html()* : *method* ini memberikan konten HTML yang baru kepada semua elemen yang sesuai
 33 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 2.25* 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 2.25: 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 2.26* merupakan cara untuk mendapatkan *background color* dari semua elemen *list* dan *listing 2.27* merupakan cara mengubah sejumlah properti menggunakan *object literal notation*.

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

Listing 2.26: Mengubah warna background color semua elemen list

2

```
31 | $( 'li' ).css({
42 |     'background-color': 'red',
53 |     'font-family': 'Courier'
64 | });
```

Listing 2.27: Mengubah warna background color dan jenis font untuk semua elemen list

7 2.4.3 *Looping*

8 Pada *jQuery* dapat dilakukan *looping* untuk mendapatkan informasi dari setiap elemen atau untuk
 9 memberikan aksi pada setiap elemen. *Method* yang digunakan untuk looping adalah *each()*. *Method*
 10 ini akan memberikan sebuah atau lebih aksi pada setiap elemen. *Method* ini memiliki sebuah
 11 parameter yaitu sebuah fungsi yang isinya adalah perintah-perintah yang akan dijalankan oleh
 12 setiap elemen. Contohnya terdapat pada listing 2.28. Pada listing 2.28 akan dipilih elemen list.
 13 *Method .each()* akan menjalankan kode program yang sama untuk setiap elemen *list* tersebut. *'this.id'*
 14 mengacu kepada id milik sebuah elemen *list* yang sekarang berada dalam *loop*. Kemudian variabel
 15 yang bernama *ids* akan menyimpan id setiap elemen *list*. *\$(this)* digunakan untuk membuat sebuah
 16 objek *jQuery* baru yang isinya adalah sebuah elemen yang ada sekarang. *\$(this)* memungkinkan kita
 17 untuk menggunakan *method* pada elemen yang ada sekarang. Elemen *list* yang ada pada *loop* akan
 18 ditambahkan sebuah elemen *span* yang isinya adalah id dari elemen tersebut. Perintah ini akan
 19 dilakukan untuk setiap elemen *list*.

20

```
211 | $( 'li' ).each(function(){
222 |     var ids = this.id;
233 |     $(this).append( '<span class="order">'+ids+'</span>' );
244 | });
```

Listing 2.28: Menambah setiap elemen *list* dengan id *list* masing-masing

25 2.4.4 *Event*

26 Sama seperti *Javascript*, pada *jQuery* juga dapat ditambahkan *event*. *Method* yang digunakan untuk
 27 menambahkan *event* adalah *on()*. Untuk memperjelas cara menggunakan *method on()*, terdapat
 28 potongan kode pada listing 2.25. Untuk menambahkan *event*, maka pertama harus memilih elemen
 29 yang akan ditambahkan *event* dengan menggunakan *selector*. Pada listing 2.29, elemen yang dipilih
 30 adalah semua elemen *list*, kemudian tambahkan *method on()*. *Method .on()* memiliki 2 parameter
 31 yaitu *event* yang akan digunakan dan perintah yang akan dilakukan apabila *event* tersebut terjadi
 32 pada elemen yang dipilih. Perintah dapat berupa sebuah fungsi anonim yaitu fungsi yang dibuat
 33 langsung atau memanggil sebuah fungsi yang sudah ada. Pada listing 2.29, *event* yang digunakan
 34 adalah *'click'* dan akan diberikan sebuah fungsi anonim yang tugasnya adalah menambahkan atribut
 35 *class* yang bernilai *'complete'*.

36

```

11 | $( 'li' ).on( 'click', function(){
12 |     $(this).addClass( 'complete' );
13 | });

```

Listing 2.29: 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 2.30*. Pada *listing 2.30*, 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*.

```

171 | $( 'li' ).on( 'click', function(event){
182 |     eventType = event.type;
193 | });

```

Listing 2.30: 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 diperbarui 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 mengirmkan 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 2.31](#) terdapat potongan kode untuk membuat *Ajax request*.

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

```
13 | xhr.send();
```

Listing 2.31: Membuat *Ajax request*

2 Berdasarkan *listing 2.31*, hal pertama yang dilakukan adalah membuat objek *XMLHttpRequest*
 3 yang disimpan pada variabel bernama *xhr*. Kemudian *method open()* berfungsi untuk menyiapkan
 4 request. *Method open* memiliki 3 parameter yaitu *HTTP method*, *url* dari halaman yang akan
 5 menangani *request*, dan tipe data *boolean* yang menentukan apakah *request* tersebut *asynchronous*
 6 atau tidak. Pada *listing 2.31* *request* tersebut menggunakan *HTTP method* yaitu *GET*, *url* halamannya
 7 adalah *'data/test.json'*, dan *asynchronous*. *Method .send()* digunakan untuk mengirimkan *request*
 8 yang sudah disiapkan. Informasi tambahan dapat dikirimkan ke server yang dituliskan pada
 9 parameter *method .send()*. Sesudah mengirimkan *request* dan menerima respon dari server, data
 10 yang diterima akan diproses. Pada *listing 2.32* terdapat potongan kode untuk menerima respon
 11 dan memproses data dari server. Setelah *browser* menerima dan memuat respon dari server, event
 12 *onload* akan dijalankan dan akan *trigger* sebuah fungsi. Di dalam fungsi tersebut, akan dicek status
 13 dari objek tersebut untuk memastikan apakah respon dari server tidak ada masalah.

```
14  
151 | xhr.onload = function(){  
162 |     if(xhr.status === 200){  
173 |         //proses data yang sudah diterima dari server  
184 |     }  
195 | }
```

Listing 2.32: Memproses respon yang didapat dari server

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

- 21 • *load()* : memuat HTML dalam sebuah elemen.
- 22 • *\$.get()* : memuat data menggunakan *method HTTP GET*. *Method* ini digunakan untuk *request*
 23 data dari server.
- 24 • *\$.post()* : memuat data menggunakan *method HTTP POST*. *Method* ini digunakan untuk
 25 mengirim data ke server yang mengubah data pada server.
- 26 • *\$.getJSON()* : memuat data JSON menggunakan *GET*.
- 27 • *\$.getScript()* : memuat dan mengeksekusi data pada *Javascript* menggunakan *GET*.
- 28 • *\$.ajax()*: *method* ini digunakan untuk menjalankan semua *request* yang sudah dijelaskan pada
 29 poin sebelumnya.

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

- 33 • *responseText* : mengembalikan data text
- 34 • *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 pengaturan yang digunakan untuk mengontrol *Ajax request*. Semua pengaturan dituliskan menggunakan *object literal notation*. Berikut adalah pengaturan 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* : pengaturan yang akan dijalankan setelah *event success* atau *error*
- *timeout* : angka dalam milisekon untuk menunggu sebelum *event* akan gagal

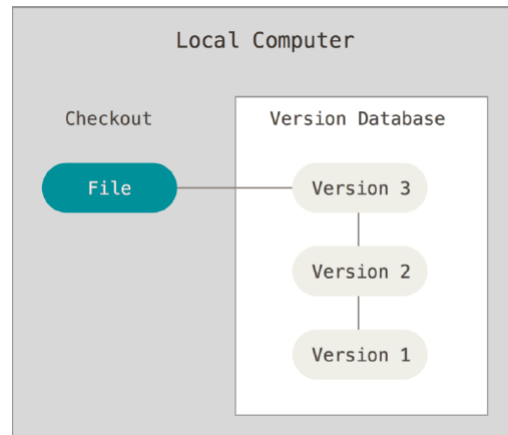
2.5 *Git*

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.

1 Local Version Control System

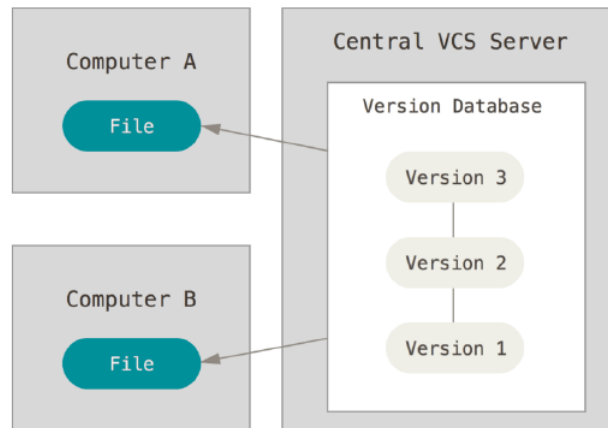
2 *Local Version Control System* memiliki sebuah basis data yang menyimpan semua perubahan pada
 3 *file* dalam *revision control*. Salah satu VCS tools yang cukup terkenal adalah RCS yang masih
 4 digunakan oleh banyak komputer hingga sekarang. Cara kerja RCS adalah menyimpan *patch sets*
 5 yang merupakan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



Gambar 2.5: Local Version Control

8 Centralized Version Control System

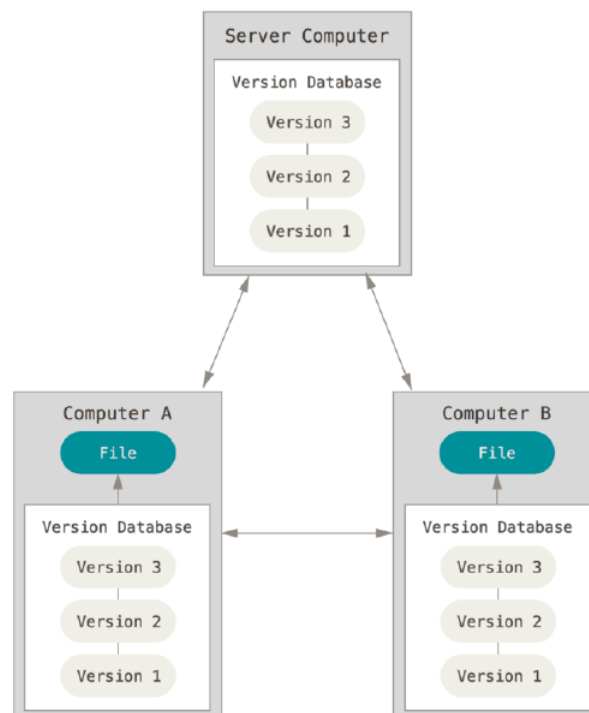
9 *Local Version Control System* menjadi kurang efektif, bila ada beberapa orang yang berkolaborasi
 10 dengan pengembang. Karena pada *Local Version Control System*, *version control* dimiliki oleh
 11 masing-masing komputer sehingga pengguna tidak tahu apakah *file* tersebut sudah diubah oleh kola-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



Gambar 2.6: Centralized Version Control

1 Distributed Version Control System

- 2 Dalam DVCS (*Distributed Version Control System*) seperti *Git*, *Mercurial*, *Bazaar* dan *Darcs*, *client*
- 3 tidak mengecek versi terbaru dari *file* tetapi *client* menggandakan *repository* termasuk historinya.
- 4 Jika *server* mati/kehilangan data, maka *client* memiliki *file back-up* untuk mengembalikannya.
- 5 Ilustrasi DVCS terdapat pada Gambar 2.7.



Gambar 2.7: Distributed Version Control

6 2.5.2 *Git*

- 7 *Git* merupakan sebuah *version control* namun berbeda dengan VCS lainnya dilihat dari cara me-
- 8 nyimpan datanya. Sistem seperti *CVS*, *Subversion*, *Perforce*, *Bazaar* menyimpan data sebagai
- 9 sekumpulan *file* dan perubahan setiap *file* disimpan setiap waktu. Pada *Git*, data tersebut dianggap

sebagai sekumpulan *snapshot* dari *miniature filesystem*. Setiap *commit* atau menyimpan proyek, *Git* seolah-olah mengambil gambar untuk melihat seperti apa *file* yang terlihat pada saat itu dan menyimpannya sebagai referensi pada *snapshot* tersebut. Singkatnya, apabila tidak ada *file* yang diubah, *Git* tidak akan menyimpan *file* lagi.

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

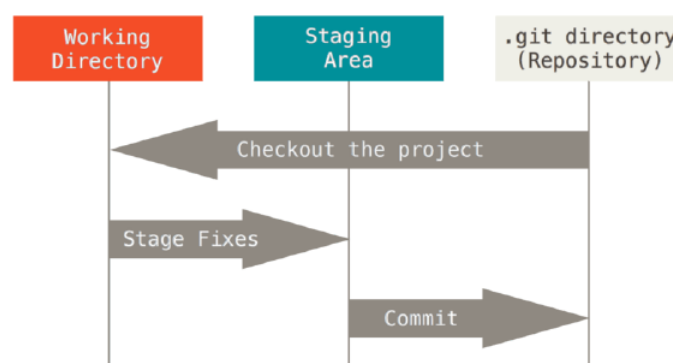
Git memiliki 3 *state* utama pada *file* yaitu:

- *committed* : data sudah tersimpan di basis data lokal.
- *modified* : *file* sudah diubah namun belum *commit* ke basis data.
- *staged* : menandai *file* yang sudah dimodifikasi dalam versi sekarang untuk *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 *decompressed* 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 1. Pengguna memodifikasi *file* di *working tree* milik pengguna.
- 2 2. Pengguna memilih *file* yang akan menjadi bagian dari *commit* selanjutnya. *File* yang terpilih
- 3 akan ditambahkan ke *staging area*.
- 4 3. Pengguna melakukan *commit file* tersebut yang berada pada *staging area* dan menyimpan
- 5 *snapshot* secara permanen ke *Git directory*.

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

10
11 Ada beberapa cara dalam menggunakan *Git* yaitu dengan menggunakan *command-line* dan
12 beberapa GUI(*Graphical User Interface*) yang memiliki kemampuan yang bermacam-macam. Pada
13 umumnya digunakan *command-line*, karena *command-line* dapat menjalankan semua perintah
14 *Git* sedangkan GUI hanya memiliki sebagian fungsionalitas pada *Git* supaya simpel dan mudah
15 digunakan.

16 **Mendapatkan *Git Repository***

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

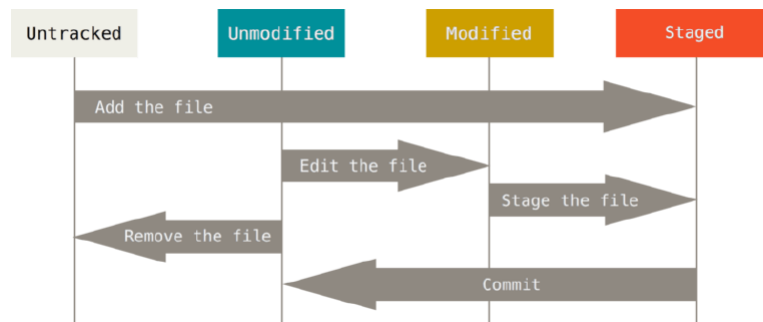
20
21 Jika memiliki direktori proyek yang belum dalam *version control* dan ingin mengontrolnya
22 menggunakan *Git*, hal pertama yang harus dilakukan adalah dengan membuka direktori proyek.
23 Perintah untuk membuat repository pada *Windows* adalah dengan mengetikkan perintah `$ cd /c/u-`
24 `ser/my_project` sesudah itu ketik perintah `$ git init`. Perintah tersebut akan membuat subdirektori
25 bernama `.git` yang mengandung semua repository yang dibutuhkan. Setelah mengetikkan perintah di
26 atas, proyek tersebut belum di-*track* sama sekali. Untuk men-*track file-file* pada sebuah proyek,
27 pertama gunakan perintah `git add` untuk men-*track file* yang diinginkan kemudian ketik `git commit`
28 untuk commit file tersebut.

29
30 *Clone repository* adalah mendapatkan *copy* dari *repository* yang sudah ada. Perintah yang
31 digunakan adalah `git clone`. Tidak hanya *file-file* pada *repository* saja yang di-*copy*, tetapi semua
32 histori pada *repository* tersebut akan ikut ter-*copy*. Perintah `git clone` diikuti dengan *url*. *Url* ini
33 berisi *link* di mana *repository* berada.

34 **Record Perubahan pada *Repository***

35 Setiap *file* dalam direktori memiliki 2 *state* yaitu *tracked* atau *untracked*. *Tracked file* adalah *file*
36 yang berada pada *snapshot* terakhir. *Tracked file* adalah *file* yang *Git* ketahui sekarang. *Untracked*
37 *file* adalah *file* yang tidak berada pada *snapshot* terakhir. Ketika *file* diubah, *Git* melihat bahwa
38 *file* tersebut sudah dimodifikasi, karena *file* tersebut diubah setelah *commit* terakhir. Kemudian

- 1 *file* yang sudah dimodifikasi tersebut di-*stage* dan *commit* semua *file* yang sudah distaged tersebut.
- 2 Gambar 2.9 menunjukkan siklus hidup dari status *file*.



Gambar 2.9: Siklus hidup pada status file

3 Perintah *git status* digunakan untuk mengecek status *file*. Jika mengetik perintah sesudah
 4 *clone*, maka tidak ada *untracked file* karena pada saat *clone*, tidak ada *file* yang dimodifikasi.
 5 Bila menambahkan sebuah *file* baru atau mengubah *file* lalu mengetik perintah *git status*, maka
 6 akan diberitahukan bahwa terdapat *untracked file*. Karena itu untuk men-*track file* baru, gunakan
 7 perintah *git add* yang diikuti dengan nama *filenya* seperti contoh ini : `$ git add README`. Perintah
 8 *git add* tidak hanya digunakan untuk men-*track file* baru. Selain digunakan untuk men-*track file*,
 9 perintah *git add* digunakan untuk stage *file* yang sudah dimodifikasi.

10
 11 Tidak semua *file* akan ditambahkan secara otomatis oleh *Git* atau ada *file* yang ditunjukan
 12 sebagai *file untracked*. Hal ini dapat diatasi dengan membuat sebuah *file* yang bernama *.gitignore*.
 13 *File .gitignore* ini berisi *file-file* yang tidak akan di-*track* oleh *Git*. *File* yang biasanya ada dalam
 14 *.gitignore* adalah *log*, *tmp* atau *file* dokumentasi yang digenerate secara otomatis. Adapun aturan
 15 untuk *pattern* yang dapat dimasukan pada *file .gitignore* diantaranya adalah :

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

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

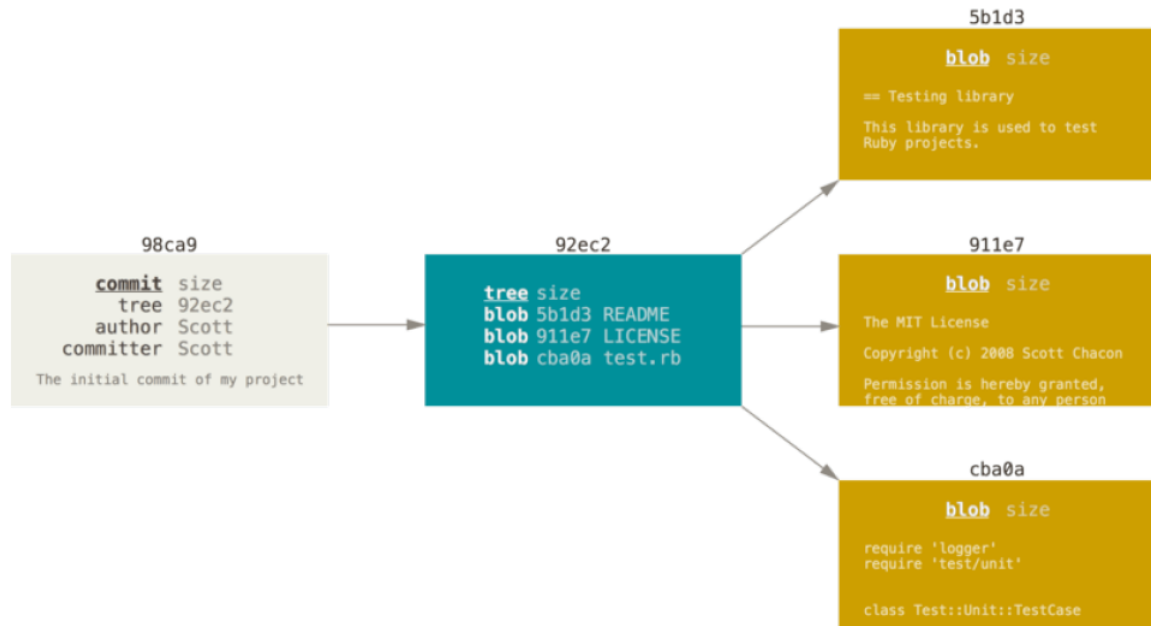
25
 26 Perintah *git commit* digunakan untuk *commit file* yang sudah diubah dan ditambahkan. *File*
 27 tersebut harus sudah di-*stage* dengan menggunakan perintah *git add*. *File* yang belum di-*stage* akan
 28 berada dalam state *modified* meskipun sudah melakukan *commit*. Untuk menambahkan keterangan

- 1 tentang *file* yang *docommit* dapat dituliskan perintah *git commit -m* yang diikuti dengan keterangan
- 2 yang ingin disampaikan.

3 2.5.3 *Git Branching*

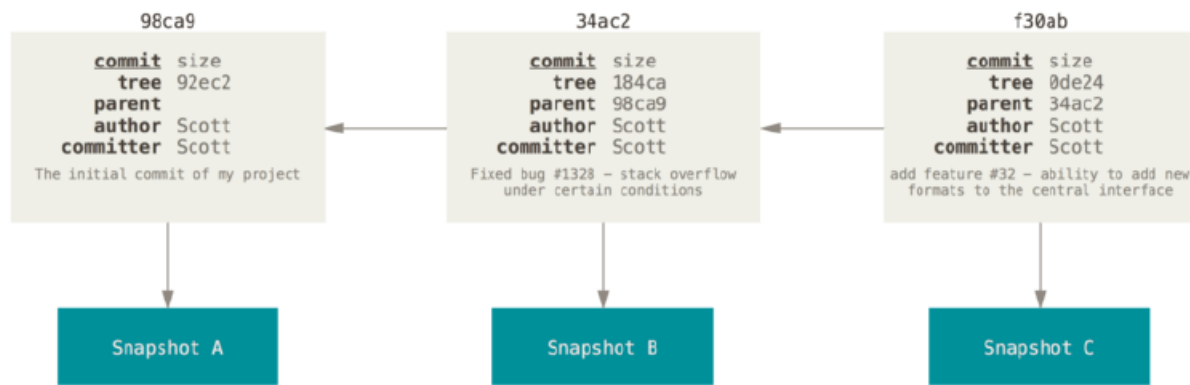
- 4 *Branching* artinya membuat dan mengerjakan sebuah proyek di tempat yang berbeda namun
- 5 masih dalam repository yang sama sehingga tidak mengubah proyek utama. Ketika *commit*, *Git*
- 6 menyimpan objek *commit* yang memiliki sebuah *pointer* pada *snapshot* sebuah konten yang sudah
- 7 dalam *state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

- 10 Misalkan seorang pengguna memiliki 3 *file*, kemudian file tersebut semuanya di-*stage* dan
- 11 *commit*. *Staging file* akan mengkomputasi *checksum* untuk setiap *file*, menyimpan versi tersebut
- 12 pada *Git repository* (hal ini dapat disebut juga sebagai *blobs*), dan menambah *checksum* tersebut ke
- 13 *staging area*. Lalu *Git* melakukan *checksum* pada setiap *subdirectory* dan menyimpan ketiga objek
- 14 tersebut pada *Git repository*. Sesudah itu *Git* akan membuat objek *commit* yang mengandung
- 15 *metadata* dan *pointer* ke proyek *root* sehingga dapat melihat *snapshot* tersebut pada setiap versi.
- 16 Sekarang, *Git repository* memiliki 5 objek yaitu 3 *blob* yang merepresentasikan 3 file, sebuah *tree*
- 17 yang mengandung isi direktori dan memberi nama *blob* berdasarkan nama file yang *docommit*, dan
- 18 sebuah *commit* dengan *pointer* ke *root tree* dan semua *commit metadata*. Gambar 2.10 merupakan
- 19 *tree* dari penjelasan tersebut.



Gambar 2.10: Commit dan tree dari file yang *docommit*

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



Gambar 2.11: Commit dan parent dari commit

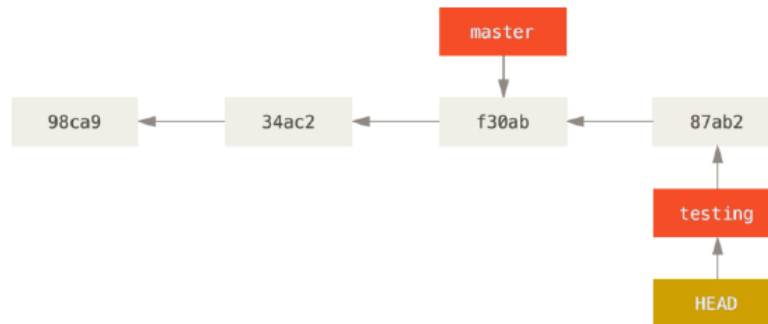
1 Nama *branch* pada *Git* awalnya disebut *master*. Ketika *commit*, pengguna diberikan *branch*
 2 *master* yang menunjuk pada *file* yang *dcommit* terakhir. Setiap *commit*, pointer pada *branch*
 3 *master* akan terus maju secara otomatis.

4
 5 Untuk membuat *branch* baru, gunakan perintah *git branch* diikuti dengan nama *branch*. *Git*
 6 menggunakan *pointer* yang disebut dengan *HEAD* untuk mengetahui bahwa pengguna sedang
 7 berada dalam *branch* tertentu. Bila membuat *branch* baru, posisi *HEAD* tetap berada pada *branch*
 8 yang sekarang. Perintah *git branch* hanya membuat *branch* baru dan tidak berpindah ke *branch*
 9 yang baru saja dibuat. Pada Gambar 2.12, jika mengetikkan perintah *git branch testing*, *branch*
 10 *testing* akan dibuat tetapi *pointer HEAD* akan tetap berada pada *branch master*.



Gambar 2.12: Pointer HEAD menunjuk branch master

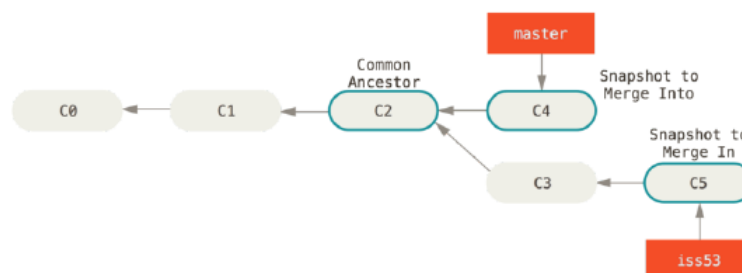
11 Untuk pindah *branch*, gunakan perintah *git checkout* diikuti dengan nama *branch*. *Pointer*
 12 *HEAD* akan berpindah ke *branch* tersebut. Bila pada *branch* tersebut pengguna melakukan *commit*,
 13 maka *branch* tersebut akan maju beserta dengan *pointer HEAD* seperti dicontohkan pada Gam-
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

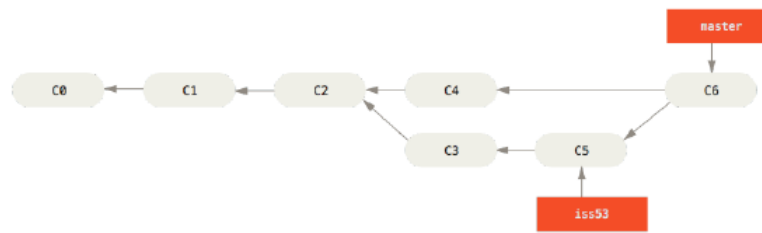
Gambar 2.13: *Pointer HEAD* beserta *branch testing*

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

7 *Basic Merging*

8 *Merging* adalah penggabungan sebuah *branch* dengan *branch* lain. Perintah untuk *merge* adalah *git*
 9 *merge* diikuti dengan nama *branch* yang ingin digabungkan. Bila sebuah *branch* ingin digabungkan
 10 dengan *branch* yang memiliki *direct ancestor* yang berbeda, *Git* akan melakukan *three way merge*.
 11 *Three way merge* ini menggunakan 2 *snapshot* yang menunjuk pada *branch* yang akan digabungkan
 12 dan 1 *snapshot* yang menunjuk pada *ancestor* yang sama dari kedua *branch* tersebut seperti yang
 13 terdapat pada Gambar 2.14. Kemudian *Git* membuat *snapshot* baru yang merupakan hasil dari
 14 *three way merge* dan secara otomatis akan membuat *commit* yang baru seperti yang terlihat pada
 15 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*

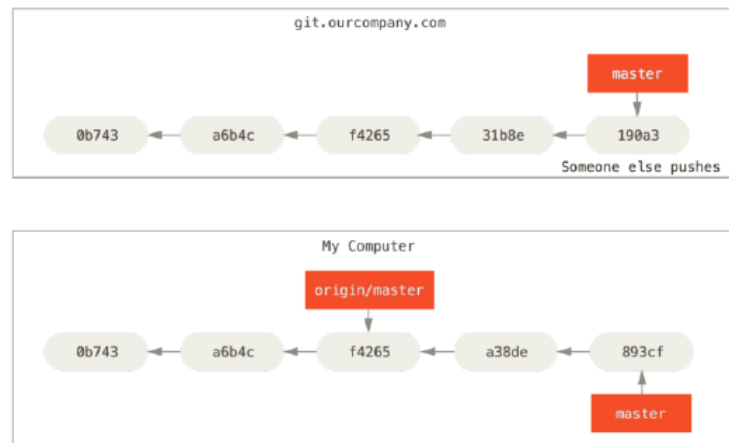
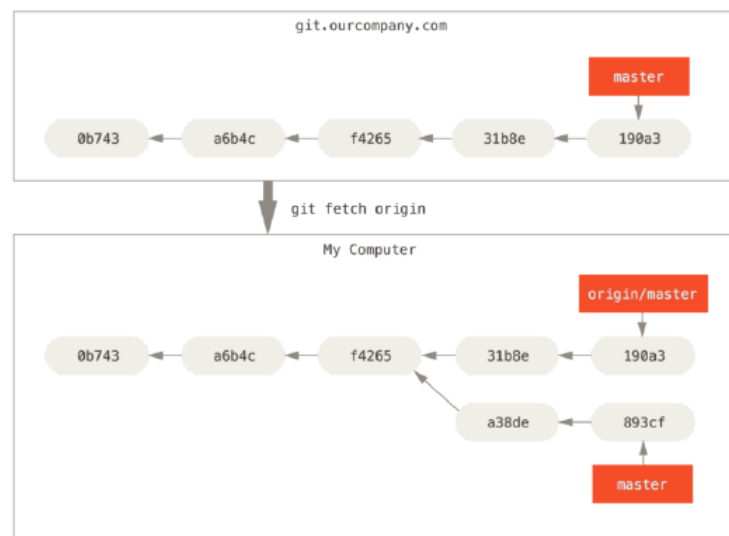


Gambar 2.15: Merge commit

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

8 Remote Branches

9 Remote-tracking branches adalah referensi dari state remote branches. Referensi tersebut merupakan
 10 referensi lokal yang hanya dapat dipindahkan oleh *Git* untuk memastikan jika referensi tersebut
 11 merepresentasikan state dari remote repository. `<remote>/<branches>` merupakan remote-tracking
 12 branches. Jika ingin mengecek *file* pada *branch master* yang berada dalam remote origin, maka
 13 pengguna harus mengecek *branch origin/master*. Sama seperti *branch master*, *origin* juga meru-
 14 pakan penamaan remote secara otomatis ketika *clone repository*. Jika pengguna mengubah *branch*
 15 lokal maka *branch* milik server tidak akan berubah dan hanya pointer pada lokal saja yang berubah.
 16 Maka dari itu *branch* di lokal dan *branch* di server bisa saja berbeda seperti yang terlihat pada
 17 Gambar 2.16 Untuk mensinkron *branch* di lokal dan *branch* di server, gunakan perintah *git fetch*
 18 diikuti dengan nama remote. Dengan cara ini, beberapa data yang belum dimiliki akan diambil
 19 dari server, meng-update basis data lokal dan memindahkan pointer ke posisi yang terbaru seperti
 20 yang terlihat pada Gambar 2.17.

Gambar 2.16: Perbedaan pada *branch* lokal dan *remote*Gambar 2.17: *Update remote-tracking branches* menggunakan perintah *git fetch*

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

Check out branch lokal dari *remote-tracking branch* secara otomatis akan membuat *tracking branch*. *Tracking branch* adalah *branch* lokal yang memiliki hubungan langsung dengan *branch remote*. Jika berada pada *tracking branch* dan mengetikkan perintah *git pull*, secara otomatis *Git* mengetahui *server* mana yang akan di-*fetch* dan *branch* apa yang akan di-*merge*. Bila *clone repository*, maka secara otomatis akan membuat sebuah *branch* yang bernama *master* yang men-*track origin/master*. Untuk mengatur *tracking branch*, perintah yang digunakan adalah *git checkout -b <branch> <remote>/<branch>*. *Git* menyediakan perintah *git checkout -track <remote>/<branch>* sebagai shortcut dari perintah checkout sebelumnya. Perintah *git checkout* juga dapat digunakan untuk mengatur *branch* lokal dengan nama yang berbeda dari *branch remote*. Jika sudah memiliki *branch* lokal dan ingin mengatur *branch* tersebut ke *branch remote* yang sudah di-*pull*, gunakan

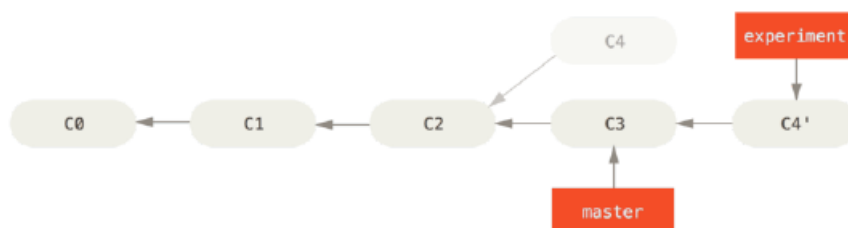
opsi `-u` atau `-set-upstream-to` pada perintah `git branch`. Untuk melihat *tracking branch* yang sudah diatur, gunakan opsi `-vv` pada perintah `git branch`. Perintah ini akan menampilkan *list* dari *branch* lokal dengan informasi tambahan mengenai *tracking* pada setiap *branch* dan apakah *branch* lokal tersebut memiliki *ahead*, *behind* atau keduanya. *Ahead* adalah ada *commit* lokal yang belum di-*push* ke *server*, sedangkan *behind* adalah *commit* yang belum digabungkan. Perintah ini tidak langsung mengambil datanya dari *server* tetapi data tersebut merupakan data saat terakhir *fetch* dari *server*. Untuk mendapatkan data yang terbaru, harus *fetch* dari semua *remote* kemudian mengetikkan perintah `git branch -vv`.

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

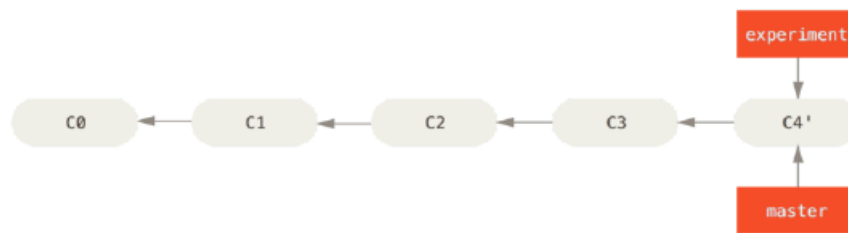
Branch pada *remote* dapat dihapus dengan menggunakan opsi `-delete` pada perintah `git push`. *Branch* pada *remote* tidak sepenuhnya dihapus, tetapi hanya *pointernya* saja yang dihilangkan. Jika *branch* tidak sengaja terhapus, maka data pada *branch* dapat dikembalikan/*diback-up*.

Rebasing

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

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

Pull Request

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

Setelah kontributor sudah membuat *Pull Request*, pemilik proyek dapat melihat saran perubahan proyek dari orang lain dan memberikan komentar/keterangan pada perubahan tersebut. Pemilik proyek dapat melihat perbedaan pada kode pemilik proyek dengan perubahan yang disarankan tersebut dan pemilik proyek dapat mengomentari baris pada kode tersebut. Orang lain dapat memberikan komentar pada *Pull Request*. Sesudah pemilik proyek memberikan keterangan tentang perubahan tersebut, kontributor menjadi tahu apa yang harus dilakukan agar perubahan tersebut dapat disetujui. Apabila perubahan tersebut membuat pemilik proyek puas, pemilik proyek akan *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.

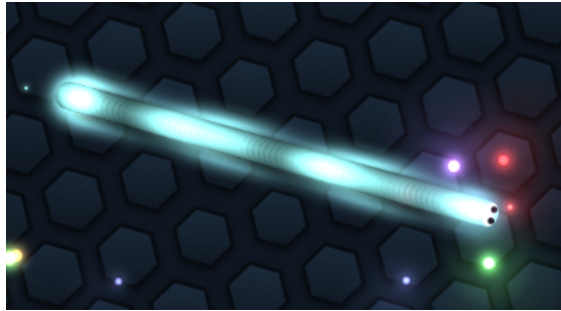
Gambar 3.1: Ular pada *Silther.io*Gambar 3.2: Makanan pada *Slither.io*

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

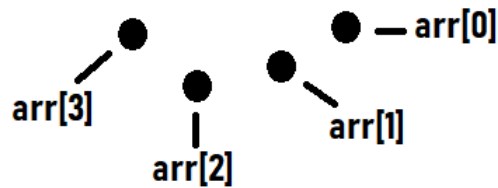
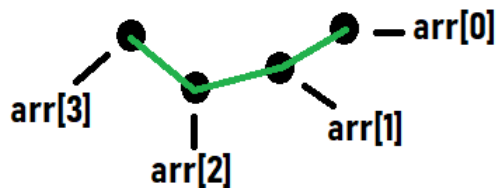
Permainan *Snake 360* yang akan dibangun memiliki cara bermain yang mirip seperti permainan *Snake* pada umumnya. Perbedaan antara *Snake 360* dengan permainan *Snake* pada umumnya adalah *Snake 360* dapat menambahkan labirin sendiri.

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 bila permainan tersebut dapat dimainkan pada *smartphone* dan *browser* pada *desktop*. Jika disesuaikan dengan layar desktop, maka besar canvas akan terlihat lebih lebar dibandingkan dengan besar layar pada *smartphone* dan sebaliknya. 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*.

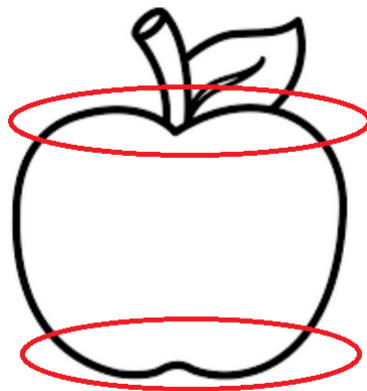
3.2.2 Menggambar Ular dan Apel

Tubuh ular dibuat menggunakan sekumpulan *line*/garis pendek. Setiap bagian tubuh ular memiliki panjang sebesar 2 *pixel* dan lebar tubuhnya sebesar 10 *pixel*. Panjang setiap bagian tubuh ular tidak 1 *pixel* karena akan membuat ular terlihat sangat pendek. 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*

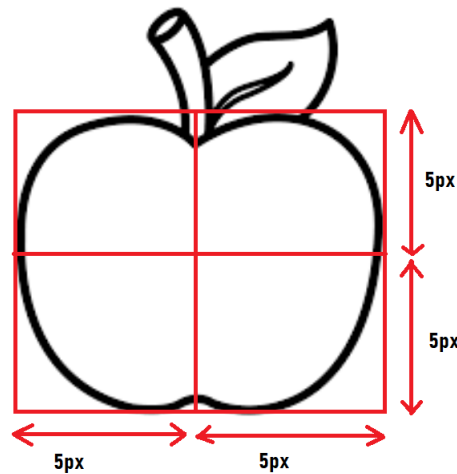
Gambar 3.8: Tubuh ular setelah digambar menggunakan garis

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



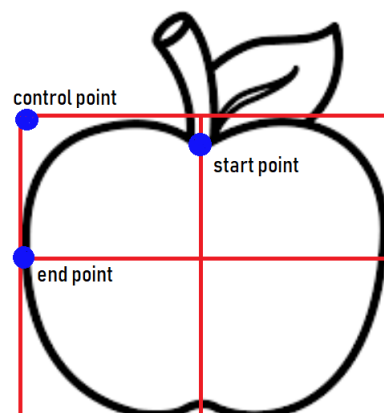
Gambar 3.9: Bagian pada apel (lingkaran merah) yang akan dibuat menggunakan kurva

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

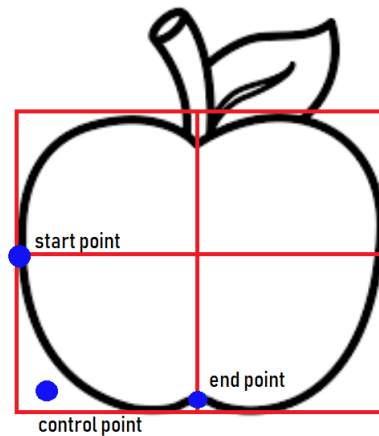


Gambar 3.10: Pembagian gambar apel dengan layout persegi beserta ukuran pada setiap bagian

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



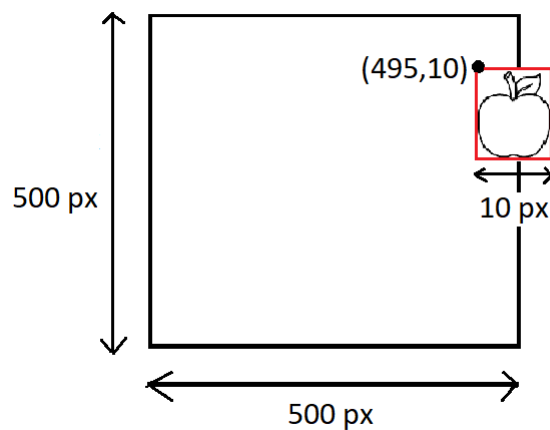
Gambar 3.13: Ilustrasi ular sebelum bergerak maju(A) dan setelah bergerak maju(B)

Kecepatan ular pada permainan ini adalah 2,4,6,8 dan 10 *pixel per frame*. Kecepatan ular minimal adalah 1 dan maksimum adalah 5. 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. Kecepatan ular tersebut bergantung pada besar bagian tubuh ular karena pada proses penggambaran tubuh ular, bagian koordinat kepala ular akan digeser posisinya di *array* apabila ular bergerak maju. Misal, jika besar bagian tubuh ular adalah 2 pixel dan kecepatan ular adalah 3 pixel per frame, maka akan ada bagian tubuh ular yang panjangnya 3 pixel. Bila diteruskan, maka besar bagian tubuh ular akan menjadi 3. Cara untuk menangani hal ini adalah dengan mengulangi pergerakan ular tersebut sebanyak kecepatan kali. Misal jika besar bagian tubuh ular adalah 2, dan kecepatannya 3 maka ular akan berpindah tempat sebanyak 3 kali. Kecepatan ular akan menjadi 6 pixel per frame. Setelah bergerak sebanyak 3 kali, ular akan digambar. Jadi, kecepatan ular merupakan kelipatan dari bagian tubuh ular.

Ular dapat berbelok dengan menggunakan tombol pada *keyboard* untuk desktop dan menyentuh layar untuk *smartphone*. Tombol ke kiri dan menekan layar bagian kiri akan membuat ular bergerak melawan arah jarum jam serta tombol ke kanan dan menekan layar bagian 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 ular bergerak berlawanan arah jarum jam, maka sudut akan berkurang dan jika ular bergerak searah jarum jam, 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, sudut tersebut akan diubah menjadi 360 dan apabila nilai sudut lebih besar dari 360, sudut tersebut akan diubah menjadi 0. Dibutuhkan rumus trigonometri untuk menentukan posisi kepala ular. Untuk menghitung posisi koordinat x pada sudut tertentu, digunakan *sinus* sedangkan untuk menghitung posisi koordinat y pada sudut tertentu menggunakan *cosinus*. Jadi, koordinat x dari kepala ular akan ditambahkan dengan hasil perhitungan sinus dari sudut dan koordinat y dari kepala ular akan ditambahkan dengan hasil perhitungan cosinus dari sudut.

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

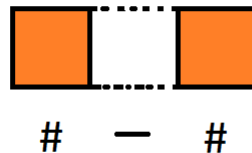
Pada permainan ini, format labirin dapat dibuat dengan menggunakan JSON dan *file text*. Permainan ini menggunakan *file text* sebagai format labirin, karena *file text* lebih mudah untuk dibuat dan dimengerti oleh pembuat labirin. Pada permainan ini, pemain dapat memilih labirin sesuai dengan yang pemain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

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.

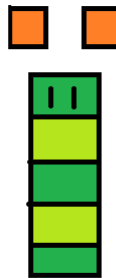
Untuk mendapatkan file tersebut, Javascript tidak dapat digunakan karena masalah keamanan

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

7



Gambar 3.15: Menggambar dinding menggunakan simbol pada file text



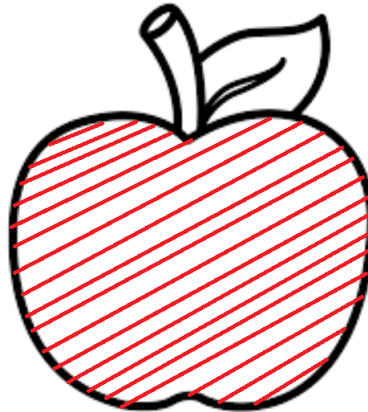
Gambar 3.16: Ular ingin melewati jalur yang diapit oleh 2 buah dinding

8 3.2.6 Pengecekan tabrakan(*Collision Detection*)

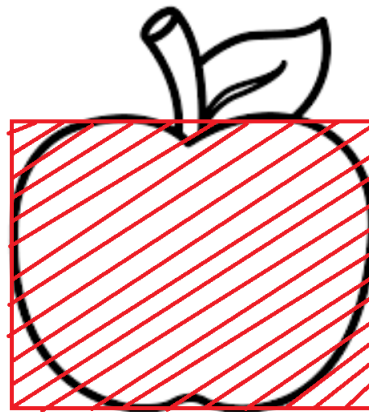
9 Pada permainan ini terdapat pengecekan tabrakan yang dapat mengecek apakah ular sudah me-
 10 makan makanan, ular menabrak tubuhnya sendiri, dan ular menabrak dinding labirin. Seluruh
 11 pengecekan ini akan dilakukan pada setiap *frame*. Pada pengecekan tabrakan pada apel dan ular,
 12 hanya perlu mengecek tabrakan antara kepala ular dengan apel. Karena jalur yang dilalui oleh
 13 kepala ular, akan selalu dilalui oleh bagian tubuh ular. Dengan kata lain, bagian tubuh ular akan
 14 mengikuti ke mana kepala ular akan bergerak. Dengan ini, tidak perlu dilakukan *collision detection*
 15 antara bagian tubuh ular dengan apel. Cukup hanya dengan mengecek tabrakan antara kepala
 16 ular dengan apel saja. Untuk mengetahui terjadinya tabrakan antara ular dengan apel, maka akan
 17 dibuat daerah tabrakan pada apel. Daerah tabrakan ini digunakan untuk mengecek apakah 2 benda
 18 saling bertabrakan satu sama lain. Daerah tabrakan pada apel ditandai dengan arsiran berwarna
 19 merah yang terdapat pada Gambar 3.17. Namun, untuk membuat daerah tabrakan ini cukup sulit
 20 ketika mengecek adanya tabrakan antara ular dengan apel terutama pada bagian lengkungan pada
 21 apel. Karena itu, daerah tabrakan pada apel dibuat dengan menggunakan bentuk persegi seperti

1 pada Gambar 3.18. Jika posisi kepala ular berada di dalam daerah tabrakan apel, maka dipastikan
2 bahwa ular tersebut sudah memakan apel. Algoritma untuk mengecek tabrakan adalah sebagai
3 berikut : cek apakah koordinat x dari kepala ular lebih besar dari posisi sisi kiri daerah tabrakan
4 dan lebih kecil dari posisi sisi kanan daerah tabrakan. Kemudian cek apakah koordinat y dari
5 kepala ular lebih besar dari posisi sisi atas daerah tabrakan dan lebih kecil dari posisi sisi bawah
6 daerah tabrakan. Jika posisi kepala ular berada memenuhi ketentuan tersebut, maka kepala ular
7 berada di dalam daerah tabrakan apel.

8



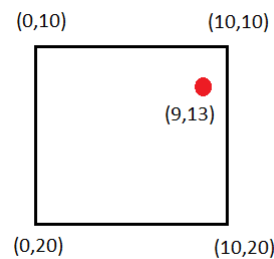
Gambar 3.17: Daerah tabrakan pada apel



Gambar 3.18: Daerah tabrakan berbentuk persegi pada apel

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

1
2 Untuk mengecek tabrakan dengan dinding labirin, dilakukan pengecekan antara kepala ular
3 dengan sebuah dinding. Bila dilakukan pengecekan antara kepala ular dengan seluruh dinding
4 labirin, maka animasi permainan akan berjalan lebih lambat. Semakin banyak dinding, animasi
5 akan berjalan lebih lambat. Cara untuk mengecek tabrakan antara kepala ular dengan dinding
6 adalah sebagai berikut : misal posisi kepala ular adalah (9,13). Jika besar dinding adalah 10 pixel,
7 maka kepala ular akan berada di daerah pada koordinat (0,10) sampai (10,20). Pada Gambar 3.19
8 terdapat gambaran untuk memperjelas contoh tersebut. Kemudian, posisi kepala ular tersebut akan
9 dibagi dengan besar dinding (10 pixel). Hasil yang didapat dari perhitungan tersebut adalah (0,1).
10 Hasil tersebut akan digunakan untuk mengecek dinding pada labirin yang diambil dari file text
11 yang sudah disimpan di array. Karena hasil dari perhitungan tersebut adalah (0,1) maka akan dicek
12 apakah array elemen kedua dan karakter pertama merupakan dinding. Misal pada array elemen
13 kedua dan karakter pertama merupakan dinding, maka kepala ular menabrak dinding.

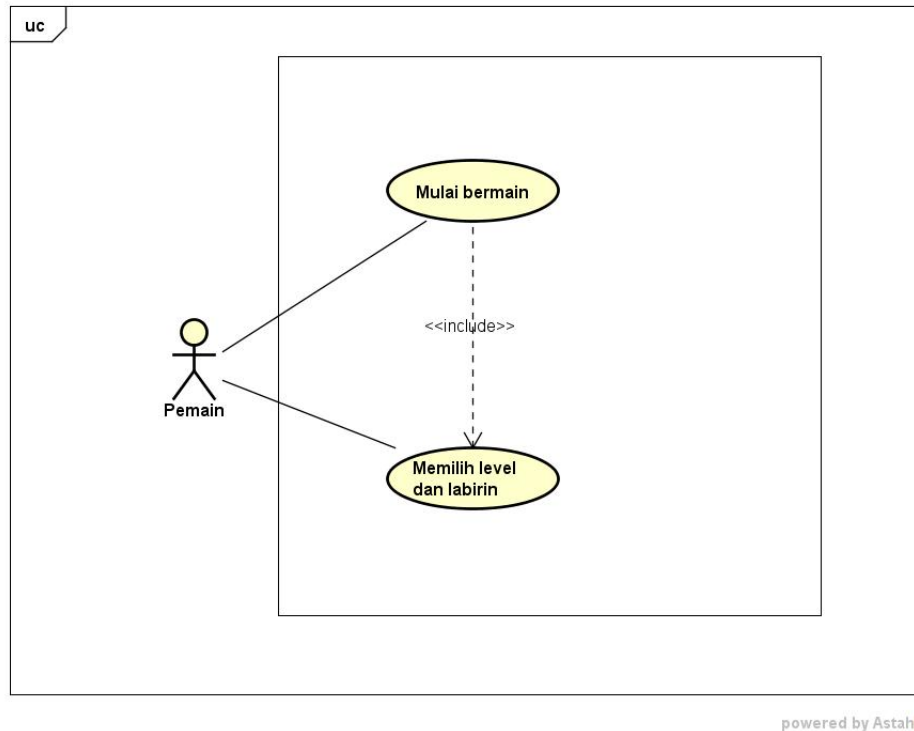


Gambar 3.19: Posisi kepala ular pada sebuah daerah labirin

14 3.3 Analisis Berorientasi Objek

15 3.3.1 Skenario Permainan

16 Pada bagian ini akan dijelaskan dan ditunjukkan diagram *use case* dari permainan *Snake 360*. Pen-
17 jelasan meliputi skenario, aktor, prakondisi skenario normal dan eksepsi. Aktor yang melakukannya
18 adalah pemain. Pada Gambar 3.20 terdapat diagram *use case* dari permainan *Snake 360*.

Gambar 3.20: 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

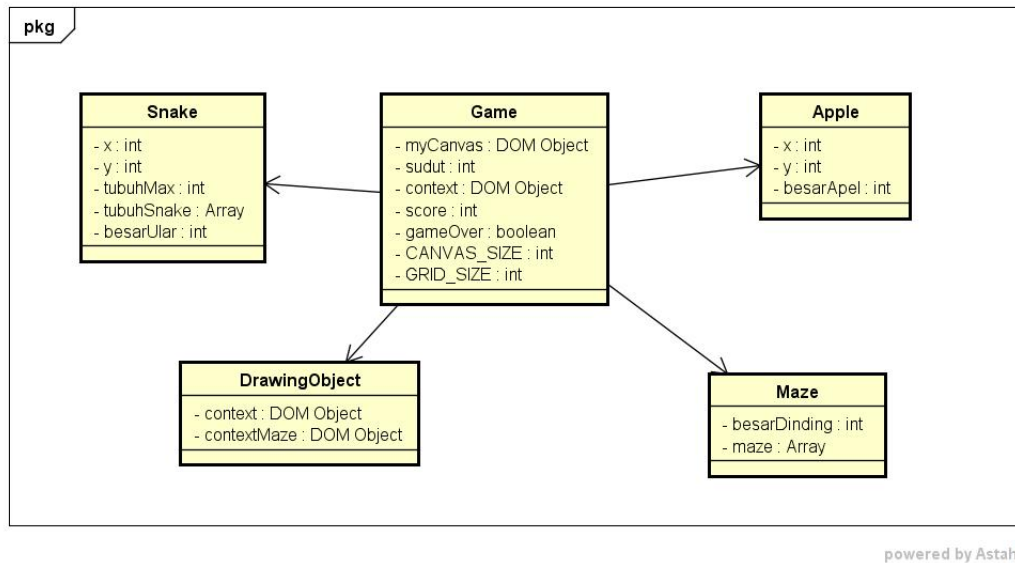
Prakondisi : Pemain sudah mulai bermain.

Skenario normal : Pemain memilih level dan labirin yang diinginkan.

Eksepsi : -

3.3.2 Diagram Kelas

Pada Gambar 3.21 terdapat diagram kelas dari *Snake 360*.



Gambar 3.21: 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 Apple

int

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

3. Kelas Game

int

-
- 1 • sudut, merupakan besar sudut yang digunakan untuk ular berbelok.
 - 2 • score, merupakan skor yang didapat pada permainan.
 - 3 • CANVAS_SIZE, merupakan lebar dan tinggi canvas.
 - 4 • GRID_SIZE, merupakan besar grid.
 - 5 **boolean**
 - 6 • gameOver, memberitahu apakah permainan sudah berakhir atau belum.
 - 7 4. Kelas Maze
 - 8 **int**
 - 9 • besarDinding, merupakan besar lebar dinding.

BAB 4

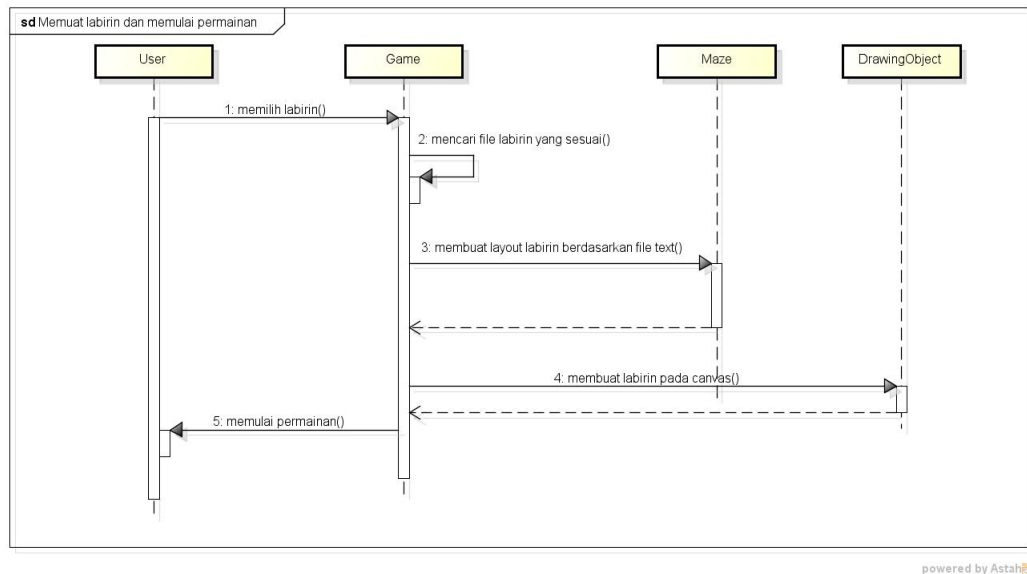
PERANCANGAN

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

4.1 Rancangan Diagram Sequence

Pada bagian ini akan ditunjukkan dan dijelaskan diagram *sequence* Open Source Snake 360. Diagram *sequence* yang dibuat adalah memuat labirin.

4.1.1 Memuat Labirin



Gambar 4.1: Diagram *sequence* untuk memuat labirin

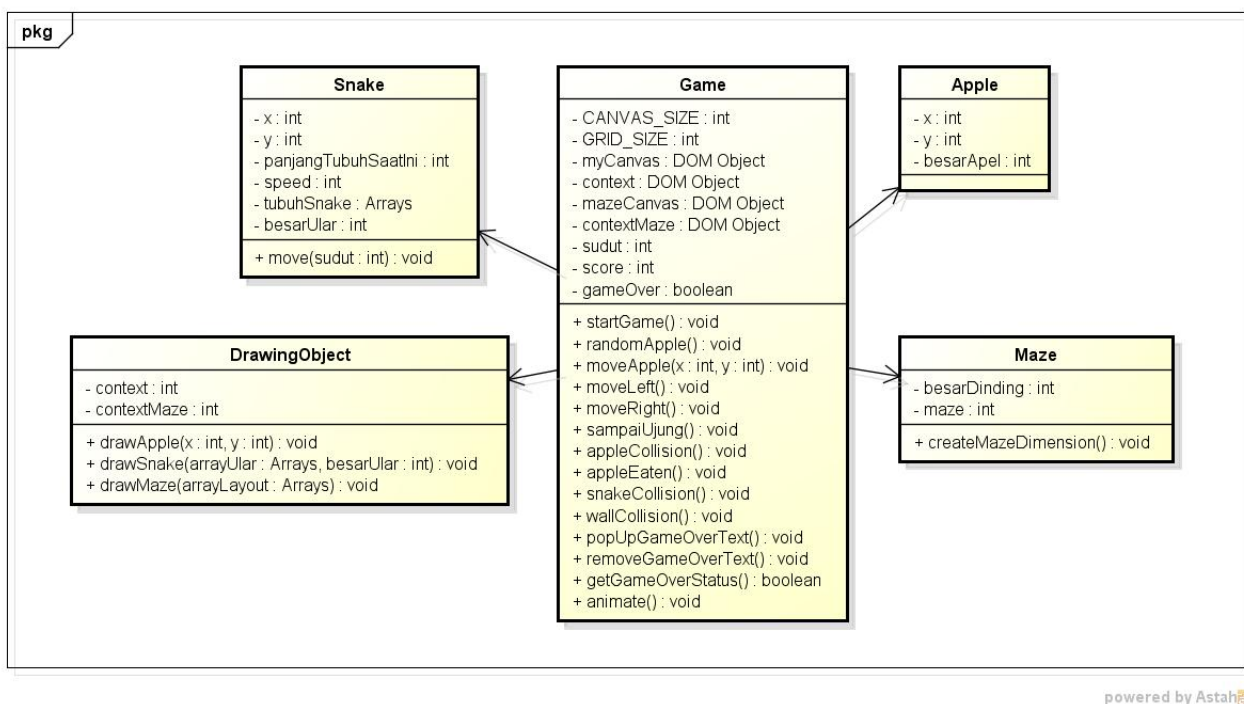
Pada Gambar 4.1, pemain memulai bermain dengan memilih labirin. Berikut adalah penjelasan dari Gambar 4.1:

1. Pemain memilih level labirin.
2. Kelas Game akan menerima input dari pemain dan mencari *file* labirin yang sesuai dengan yang pemain pilih di folder labirin. File labirin merupakan file teks.

3. Jika *file* ditemukan, maka kelas Game akan memanggil *method* kelas Maze untuk membuat labirin yang sesuai dengan isi *file* labirin.
4. Setelah labirin selesai dibuat, kelas Game akan memanggil *method* kelas DrawingObject untuk menggambar labirin.
5. Jika labirin sudah selesai digambar, kelas Game akan memulai permainan.

4.2 Rancangan 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

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

- 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

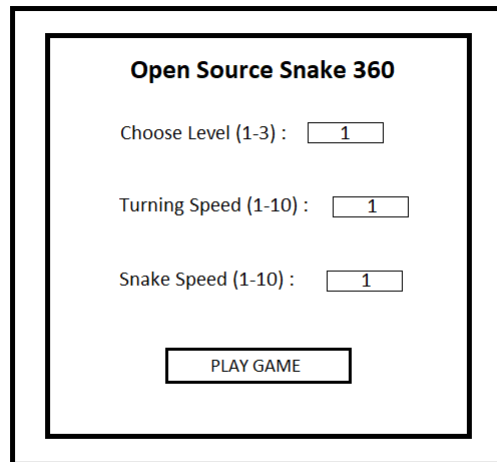
Output : tidak ada

4.3 Rancangan Tampilan Antarmuka

Pada bagian ini akan ditunjukkan rancangan tampilan antarmuka dari permainan yang dibangun yang terdiri dari menu pemilihan level labirin, 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, 3 buah input untuk memilih level labirin, memilih kecepatan gerak ular, dan memilih kecepatan ular berbelok dan sebuah tombol '*Play Game*' untuk memulai permainan. Tampilan menu utama akan menampilkan pesan kesalahan seperti terdapat pada Gambar 4.4. Apabila pemain salah memasukkan salah satu data, maka permainan tidak akan dimulai jika tombol '*Play Game*' ditekan.



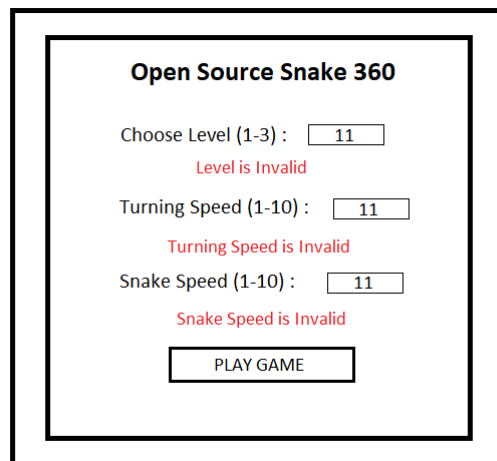
Open Source Snake 360

Choose Level (1-3) :

Turning Speed (1-10) :

Snake Speed (1-10) :

Gambar 4.3: Rancangan tampilan menu utama



Open Source Snake 360

Choose Level (1-3) :
Level is Invalid

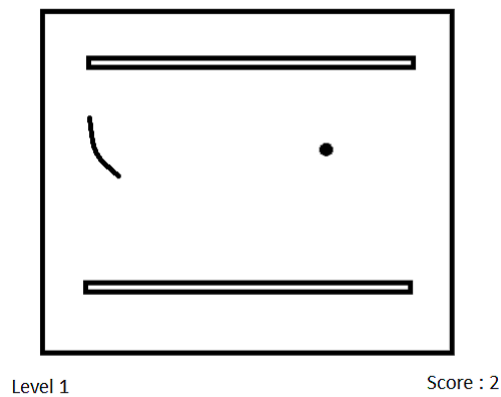
Turning Speed (1-10) :
Turning Speed is Invalid

Snake Speed (1-10) :
Snake Speed is Invalid

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

1 4.3.2 Tampilan Bermain

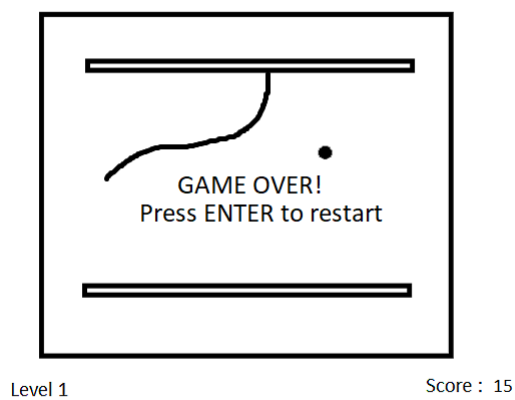
2 Tampilan bermain akan muncul setelah pemain memilih level labirin, kecepatan ular dan kecepatan
3 ular berbelok pada menu utama dengan benar dan sudah menekan tombol 'Play Game' (Gambar 4.3).
4 Gambar 4.5 merupakan tampilan permainan sudah dimulai. Pada tampilan ini terdapat ular, dinding
5 labirin dan makanan berbentuk apel. Pada tampilan ini juga terdapat level labirin yang dipilih dan
6 skor yang didapat.



Gambar 4.5: Rancangan tampilan bermain

1 4.3.3 Tampilan Permainan Berakhir

- 2 Tampilan ini akan muncul apabila permainan berakhir. Permainan akan berakhir jika ular menabrak
- 3 dinding labirin atau menabrak tubuhnya sendiri. Gambar 4.6 merupakan tampilan permainan
- 4 berakhir. Pada tampilan ini, pemain dapat mengulang permainan dengan menekan tombol '*Enter*'.
- 5 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. Permainan ini dapat dimainkan pada *link* ini : <https://generaldevilx.github.io/Snake360/>

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

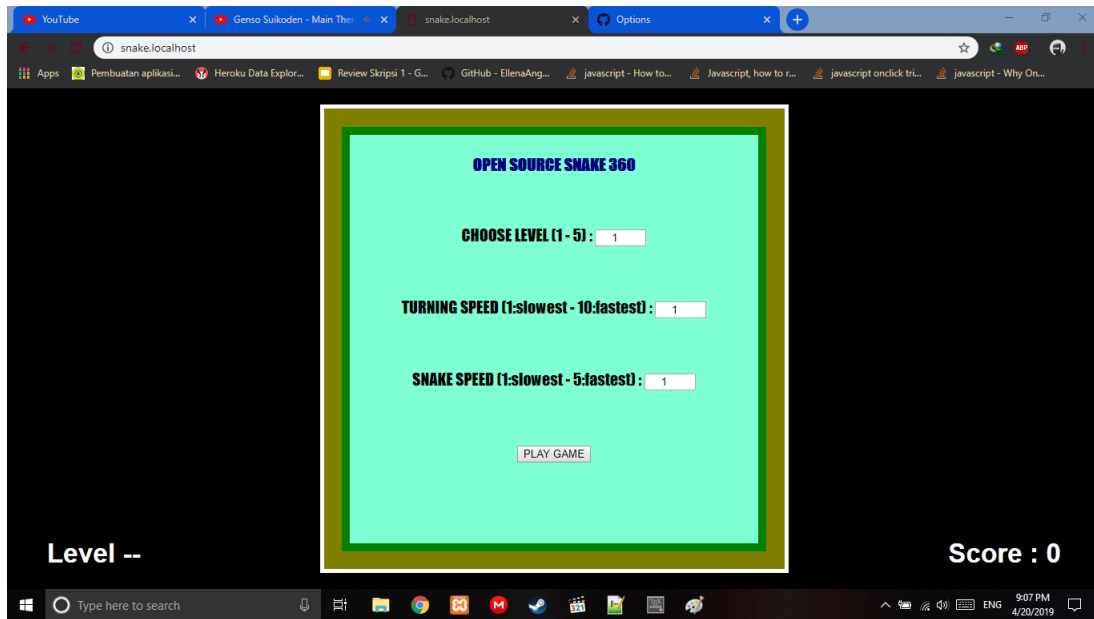
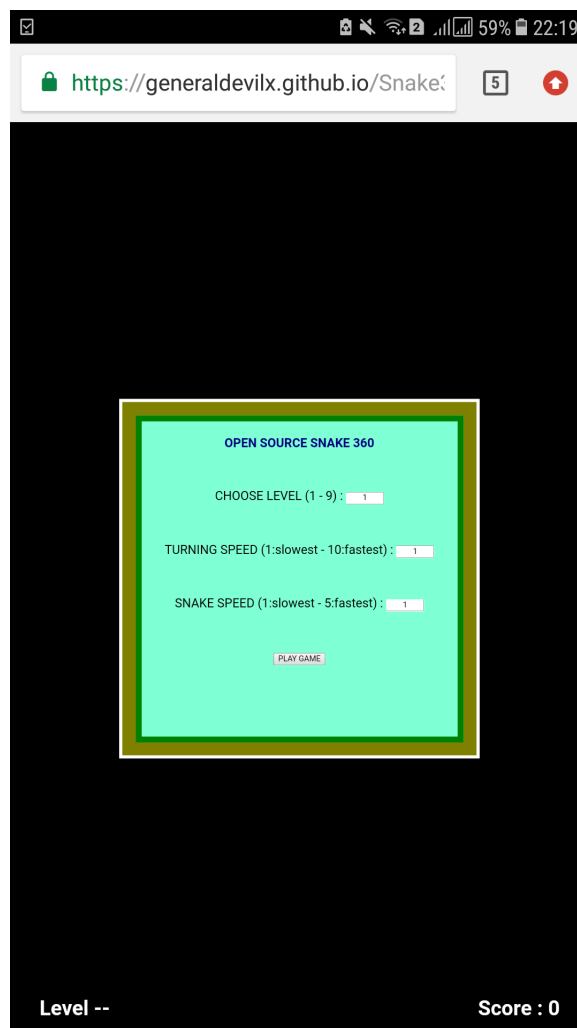
3. Sistem Operasi *Smartphone* : Android Nougat v7.0

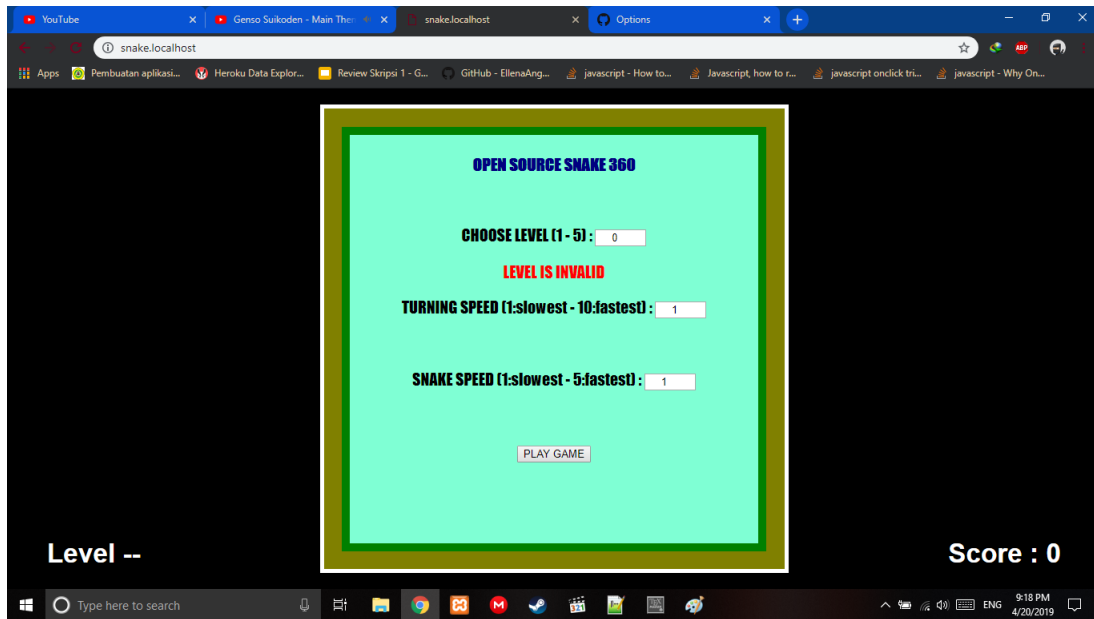
5.1.3 Implementasi Antarmuka

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

Tampilan Menu Utama

Gambar 5.1 dan Gambar 5.2 merupakan tampilan antarmuka menu utama pada *desktop* dan *smartphone*. Pada tampilan ini terdapat judul dari permainan, *input* untuk mengisi *level* labirin, kecepatan ular berbelok, kecepatan ular, dan tombol '*Play Game*'. Jika pemain salah memasukkan data, maka terdapat pesan kesalahan yang ditandai dengan tulisan berwarna merah. Misal, pada Gambar 5.3, pemain salah memasukkan data untuk level labirin sehingga muncul pesan kesalahan bahwa *input* yang dimasukkan salah.

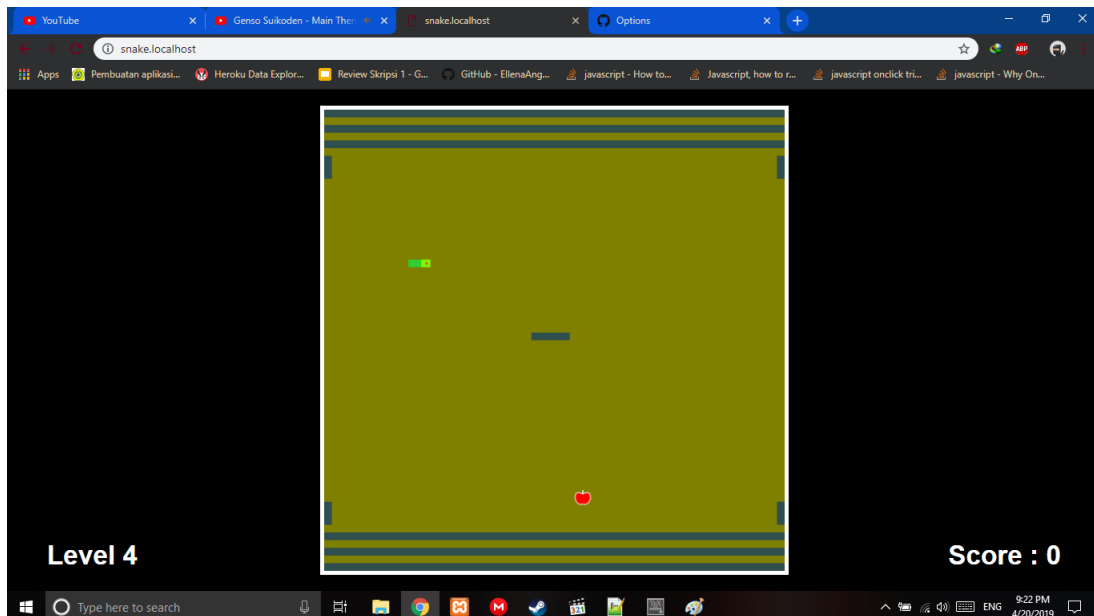
Gambar 5.1: Tampilan menu utama pada *desktop*Gambar 5.2: Tampilan menu utama pada *smartphone*



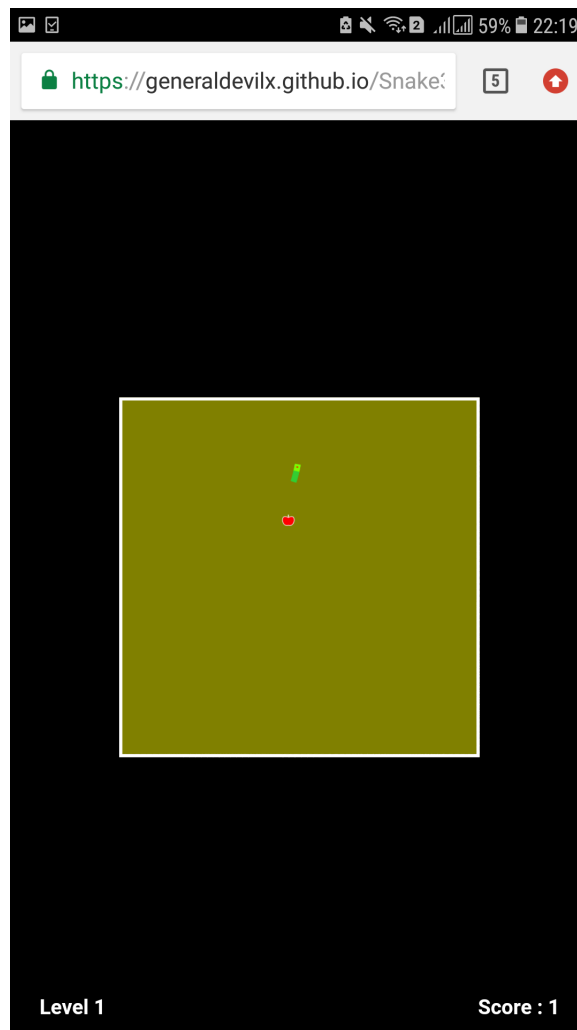
Gambar 5.3: Tampilan menu utama jika pemain salah memasukkan data *level* labirin

1 Tampilan Bermain

- 2 Gambar 5.4 dan Gambar 5.5 merupakan tampilan antarmuka mulai bermain pada *desktop* dan
- 3 *smartphone*. Tampilan ini muncul apabila pemain memasukkan data level labirin, kecepatan ular
- 4 berbelok dan kecepatan ular dengan benar dan menekan tombol "*Play Game*". Pada tampilan ini
- 5 terdapat ular yang dikontrol oleh pemain, dinding labirin, makanan ular, *level* labirin dan skor yang
- 6 didapat pemain.



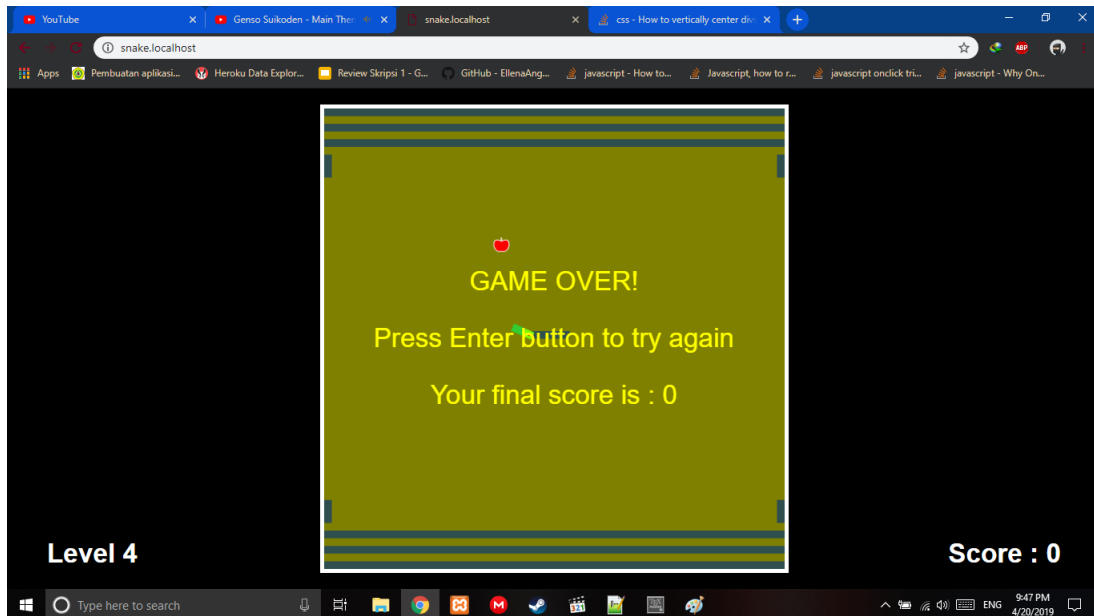
Gambar 5.4: Tampilan bermain pada *desktop*



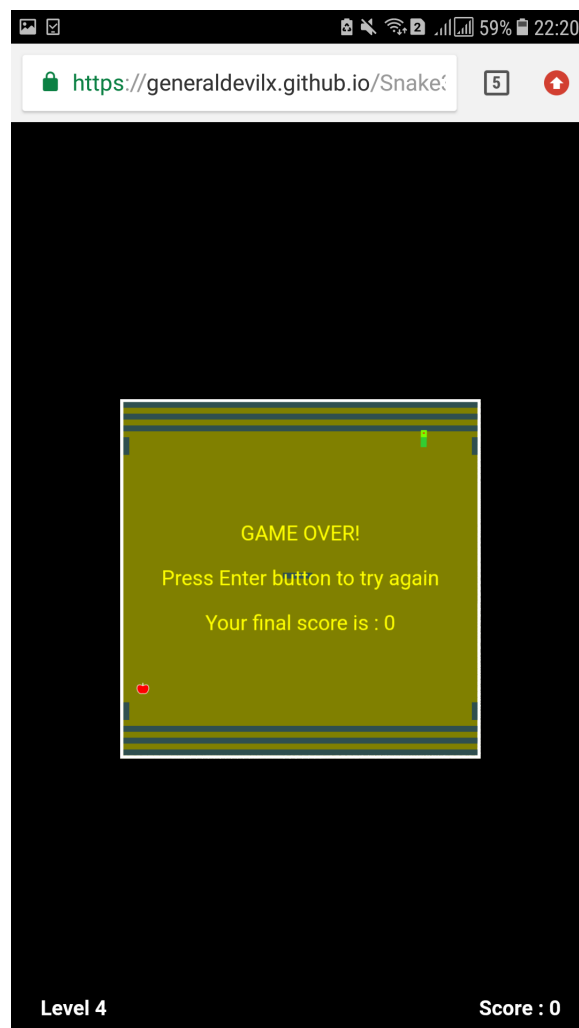
Gambar 5.5: Tampilan bermain pada *smartphone*

1 Tampilan Permainan Berakhir

- 2 Gambar 5.6 dan Gambar 5.7 merupakan tampilan antarmuka jika permainan berakhir. Tampilan
3 ini muncul apabila ular menabrak dinding labirin dan menabrak tubuhnya sendiri. Pemain akan
4 diarahkan ke tampilan utama jika pemain menekan tombol 'Enter' pada tampilan ini.



Gambar 5.6: Tampilan permainan berakhir pada desktop



Gambar 5.7: Tampilan permainan berakhir smartphone

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 dan pengujian eksperimental.

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 " <i>Enter</i> " ditekan	Pemain akan diarahkan ke tampilan menu utama	Hasil pengujian sesuai dengan yang diharapkan

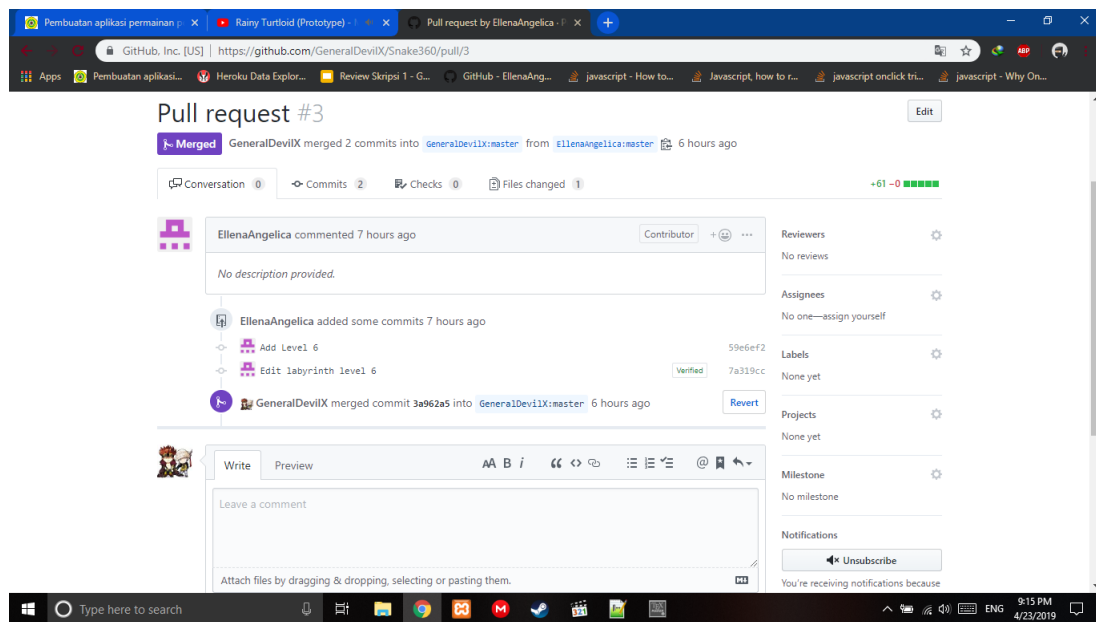
Berdasarkan tabel 5.3, dapat disimpulkan bahwa kasus uji pada tampilan "game over" membawakan hasil sesuai dengan yang diharapkan.

5.2.2 Pengujian Eksperimental

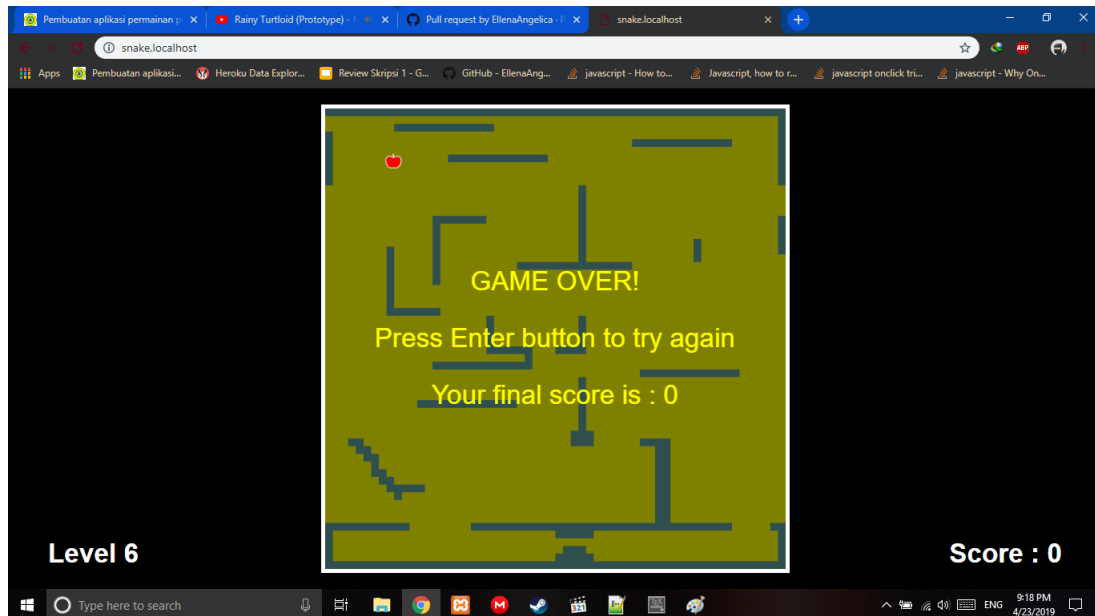
Pada pengujian eksperimental akan diuji penambahan labirin oleh orang lain. Pada pengujian ini terdapat 5 orang yang akan menambahkan labirin menggunakan *pull request Github*. Berikut adalah hasil pengujian eksperimental :

1. Penguji 1

Penguji 1 menambahkan labirin level 6 dengan menggunakan pull request berhasil dilakukan seperti yang terlihat pada Gambar 5.8. Dinding labirin yang dibuat sudah sesuai dengan besar canvas. Namun posisi ular yang dimasukkan tidak sesuai. Posisi ular tersebut berada pada daerah dinding labirin. Hasilnya adalah ketika permainan dimulai, permainan akan berakhir. Gambar 5.9 merupakan hasil pengujian menggunakan labirin yang dibuat oleh penguji 1.



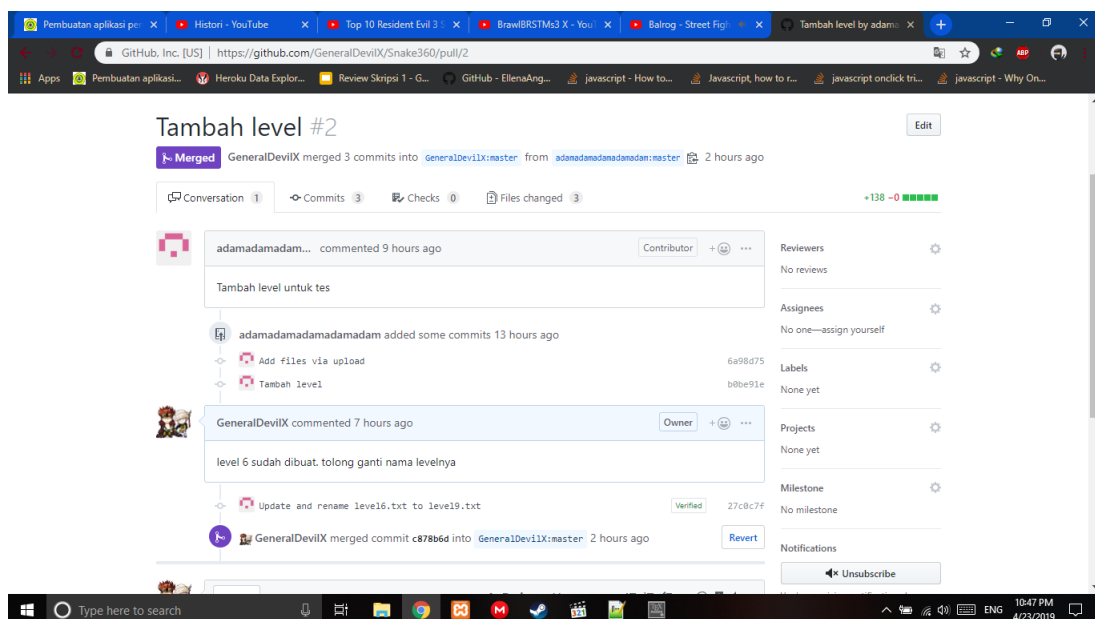
Gambar 5.8: Tampilan hasil pull request milik penguji 1



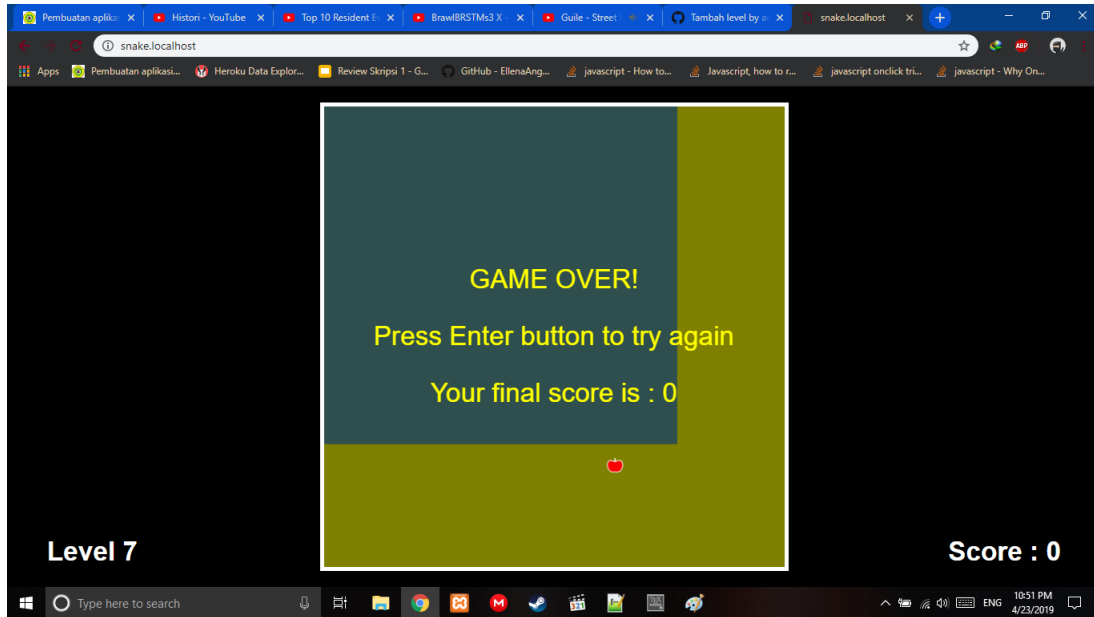
Gambar 5.9: Tampilan pengujian labirin yang dibuat oleh penguji 1

2. Penguji 2

Penguji 2 menambahkan labirin level 7 dengan menggunakan pull request berhasil dilakuka seperti yang terlihat pada Gambar 5.10. Penguji 2 menambah labirin yang sudah dibuat oleh penguji 1 dan penguji 2 berhasil mengganti nama labirin tersebut. Labirin yang dibuat oleh penguji 2 tidak sesuai dengan besar canvas sehingga tampilan menjadi seperti pada Gambar 5.11. Penguji 2 membuat labirin yang seluruhnya adalah dinding. Hasilnya sama seperti pengujian 1, pada saat permainan dimulai, permainan akan berakhir.



Gambar 5.10: Tampilan hasil pull request milik penguji 2



Gambar 5.11: Tampilan pengujian labirin yang dibuat oleh penguji 2

BAB 6

KESIMPULAN DAN SARAN

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

6.1 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.
2. Labirin cukup sulit untuk dibuat. Hal ini dapat dilihat berdasarkan pengujian eksperimental yang sudah dilakukan.

6.2 Saran

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

1. Format labirin harus dibuat lebih mudah untuk mengurangi kesalahan pada pembuatan labirin.

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 //posisi untuk animasi
305 this.animate = function(){
306     if(this.gameOver == true){
307     }
308     else{
309         this.context.clearRect(0,0,this.myCanvas.width,this.myCanvas.height);
310         this.ular.move(this.sudut);
311
312         this.sampaiUjung();
313
314         this.temp = {};
315         this.temp['x'] = this.ular.x;
316         this.temp['y'] = this.ular.y;
317
318         this.ular.tubuhSnake.unshift(this.temp);
319         if(this.ular.tubuhSnake.length > this.ular.panjangTubuhSaatIni){
320             this.ular.tubuhSnake.pop();
321         }
322
323         this.drawingObj.drawSnake(this.ular.tubuhSnake,this.ular.besarUlar);
324         this.drawingObj.drawApple(this.apel.x,this.apel.y,this.apel.getBesarApel());
325         this.appleCollision();
326         this.snakeCollision();
327         this.wallCollision();
328
329         var that = this;
330         setTimeout(function(){
331             that.animate();
332         },50);
333     }
334 }
335
336
337 this.checkLevel = function(){
338     let chosenLevel = document.getElementById("level").value;
339     let temp = $("#totalLevel").html();
340     let totalLevel = parseInt(temp);
341
342     if(chosenLevel != ""){
343         if(chosenLevel > 0 && chosenLevel <= totalLevel){
344             document.getElementById("levelInvalid").style.visibility = "hidden";
345             return true;
346         }
347         else{
348             document.getElementById("levelInvalid").style.visibility = "visible";
349             return false;
350         }
351     }
352     else{
353         document.getElementById("levelInvalid").style.visibility = "visible";
354         return false;
355     }
356 }
357
358
359 this.checkSpeed = function(){
360     let chosenSpeed = document.getElementById("speed").value;
361
362     if(chosenSpeed != ""){
363         if(chosenSpeed > 0 && chosenSpeed <= 10){
364             document.getElementById("speedInvalid").style.visibility = "hidden";
365             return true;
366         }
367         else{
368             document.getElementById("speedInvalid").style.visibility = "visible";
369             return false;
370         }
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| }

```