

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. Javasciprt 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 Game</i>	5
2.2 <i>HTML5 Canvas</i>	6
2.3 <i>Javascript</i>	7
2.3.1 Variabel	8
2.3.2 Constant	8
2.3.3 Function	9
2.3.4 Menggambar pada <i>Canvas</i>	9
2.3.5 <i>Object Oriented Programming Javascript</i>	12
2.3.6 Event	14
2.3.7 Membuat Animasi	17
2.3.8 Fungsi <i>Callback</i>	18
2.4 <i>jQuery</i>	18
2.4.1 Looping	20
2.4.2 Chaining	20
2.4.3 Mendapatkan dan Mengubah Konten Elemen	21
2.4.4 Mendapatkan dan Mengubah Properti CSS	21
2.4.5 Event	22
2.4.6 Mengubah dan Mendapatkan Dimensi dari Elemen	24
2.4.7 AJAX	24
2.5 <i>Git</i>	27
2.5.1 <i>Version Control</i>	27
2.5.2 <i>Git</i>	29
2.5.3 <i>Git Branching</i>	32
2.5.4 <i>GitHub</i>	38
3 ANALISIS	41
3.1 Analisis Permainan <i>Snake</i> yang Sudah Ada	41
3.1.1 Ular dan Makanan	41

3.1.2	Pergerakan Ular	43
3.1.3	Labirin	43
3.2	Analisis Sistem yang Dibangun	44
3.2.1	Menggambar Ular dan Apel	44
3.2.2	Pergerakan Ular	46
3.2.3	Mengacak posisi apel	47
3.2.4	Menentukan Besar <i>Canvas</i>	48
3.2.5	Menggambar Labirin	49
3.2.6	Pengecekan tabrakan(<i>Collision Detection</i>)	50
3.3	Analisis Berorientasi Objek	52
3.3.1	Skenario Permainan	52
3.3.2	Diagram Kelas	53
4	PERANCANGAN	57
4.1	Rancangan Diagram Sequence	57
4.1.1	Memuat Labirin	57
4.2	Rancangan Diagram Kelas Rinci	58
4.3	Rancangan Tampilan Antarmuka	63
4.3.1	Tampilan Menu Utama	63
4.3.2	Tampilan Bermain	64
4.3.3	Tampilan Permainan Berakhir	64
5	IMPLEMENTASI DAN PENGUJIAN	67
5.1	Implementasi	67
5.1.1	Lingkungan Perangkat Keras	67
5.1.2	Lingkungan Perangkat Lunak	68
5.1.3	Implementasi Antarmuka	68
5.2	Pengujian	74
5.2.1	Pengujian Fungsional	74
5.2.2	Pengujian Eksperimental	76
6	KESIMPULAN DAN SARAN	83
6.1	Kesimpulan	83
6.2	Saran	83
DAFTAR REFERENSI		85
A KODE PROGRAM		87
B FILE README		95

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	27
2.6	Centralized Version Control	28
2.7	Distributed Version Control	29
2.8	Working tree, staging area, dan Git directory	30
2.9	Siklus hidup pada status <i>file</i>	31
2.10	Commit dan tree dari file yang dicommit	33
2.11	Commit dan parent dari commit	33
2.12	Pointer <i>HEAD</i> menunjuk <i>branch master</i>	34
2.13	Pointer <i>HEAD</i> beserta <i>branch testing</i>	34
2.14	3 <i>snapshot</i> yang digunakan dalam <i>three way merge</i>	35
2.15	<i>Merge commit</i>	35
2.16	Perbedaan pada <i>branch</i> lokal dan <i>remote</i>	36
2.17	Update <i>remote-tracking branches</i> menggunakan perintah <i>git fetch</i>	36
2.18	<i>Rebasing commit</i> C4 ke C3	37
2.19	<i>Merge branch</i> setelah <i>rebasing</i>	38
2.20	Tombol 'Fork'	38
3.1	Ular pada <i>Slither.io</i>	42
3.2	Makanan pada <i>Slither.io</i>	42
3.3	Ular pada <i>Snake Nokia</i>	42
3.4	Makanan biasa(A) dan makanan bonus(B) pada <i>Snake Nokia</i>	42
3.5	Ular sedang melaju dengan cepat(<i>speed up</i>)	43
3.6	Peta labirin pada <i>Slither.io</i>	43
3.7	Koordinat bagian tubuh ular pada <i>array</i>	44
3.8	Tubuh ular setelah digambar menggunakan garis	44
3.9	Bagian pada apel(lingkaran merah) yang akan dibuat menggunakan kurva	45
3.10	Pembagian gambar apel dengan layout persegi beserta ukuran pada setiap bagian	45
3.11	<i>Start point, control point</i> dan <i>end point</i> untuk menggambar apel bagian kiri atas	46
3.12	<i>Start point, control point</i> dan <i>end point</i> untuk menggambar apel bagian kiri bawah	46
3.13	Ilustrasi ular sebelum bergerak maju(A) dan setelah bergerak maju(B)	47
3.14	Gambar apel yang terpotong sesudah mengacak posisi apel	48
3.15	Menggambar dinding menggunakan simbol pada file teks	49
3.16	Ular ingin melewati jalur yang diapit oleh 2 buah dinding	50
3.17	Daerah tabrakan pada apel	50
3.18	Daerah tabrakan berbentuk persegi pada apel	51
3.19	Posisi kepala ular pada sebuah daerah labirin	52
3.20	Diagram <i>use case</i> dari permainan <i>Snake 360</i>	52
3.21	Diagram class dari permainan <i>Snake 360</i>	53

4.1	Diagram <i>sequence</i> untuk memuat labirin	57
4.2	Diagram class rinci dari Open Source <i>Snake 360</i>	58
4.3	Rancangan tampilan menu utama	63
4.4	Rancangan tampilan menu utama jika pemain salah memasukkan data	64
4.5	Rancangan tampilan bermain	64
4.6	Rancangan tampilan permainan berakhir	65
5.1	Tampilan menu utama pada <i>desktop</i>	69
5.2	Tampilan menu utama pada <i>smartphone</i>	69
5.3	Tampilan menu utama jika pemain salah memasukkan data <i>level</i> labirin pada <i>desktop</i>	70
5.4	Tampilan menu utama jika pemain salah memasukkan data <i>level</i> labirin pada <i>smartphone</i>	70
5.5	Tampilan bermain pada <i>desktop</i>	71
5.6	Tampilan bermain pada <i>smartphone</i>	72
5.7	Tampilan permainan berakhir pada <i>desktop</i>	73
5.8	Tampilan permainan berakhir pada <i>smartphone</i>	73
5.9	Tampilan hasil pull request milik penguji 1	77
5.10	Tampilan pengujian labirin yang dibuat oleh penguji 1	77
5.11	Tampilan hasil pull request milik penguji 2	78
5.12	Tampilan pengujian labirin yang dibuat oleh penguji 2	78
5.13	Tampilan hasil pull request milik penguji 3	79
5.14	Tampilan pengujian labirin yang dibuat oleh penguji 3	79
5.15	Tampilan hasil pull request milik penguji 4	80
5.16	Tampilan pengujian labirin yang dibuat oleh penguji 4	80
5.17	Tampilan hasil pull request milik penguji 5	81
5.18	Tampilan pengujian labirin yang dibuat oleh penguji 5	81

DAFTAR TABEL

5.1 Pengujian Fungsional pada Tampilan Menu Utama	74
5.2 Pengujian Fungsional Tampilan Bermain pada Desktop	75
5.3 Pengujian Fungsional Tampilan Bermain pada <i>Smartphone</i>	75
5.4 Pengujian Fungsional Tampilan Permainan Berakhir	76

1

BAB 1

2

PENDAHULUAN

3 1.1 Latar Belakang

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

11

12 HTML(*Hyper Text Markup Language*) adalah sebuah bahasa markah yang digunakan untuk
13 membuat halaman web. HTML5 merupakan HTML versi 5 yang terbaru dan penerus dari HTML4,
14 XHTML1, dan DOM level 2 HTML. HTML5 memiliki beberapa elemen baru, salah satunya adalah
15 HTML5 Canvas. HTML5 Canvas adalah tempat untuk menggambar *pixel-pixel* yang dapat ditulis
16 menggunakan bahasa pemrograman *JavaScript*. *Javascript* adalah bahasa pemrograman tingkat
17 tinggi yang digunakan untuk membuat halaman web menjadi lebih interaktif. *jQuery* merupakan
18 *library Javascript* yang cepat, kecil dan kaya dengan fitur. *jQuery* membuat hal-hal seperti traversal
19 dan manipulasi dokumen HTML, penanganan *event*, animasi dan *Ajax* jauh lebih sederhana
20 dengan API(*Application Programming Interface*) yang mudah untuk digunakan pada banyak
21 *browsers*. *GitHub* adalah layanan *web hosting* bersama untuk proyek pengembangan perangkat lunak
22 yang menggunakan sistem *version control* yaitu *Git*. Dengan adanya *GitHub*, programmer dapat
23 mengetahui perubahan yang pada *repository* tersebut.

24

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

30 1.2 Rumusan Masalah

31 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))

- 1 • Bagaimana membangun permainan *Snake* menggunakan HTML5?
- 2 • Bagaimana cara menyimpan labirin pada *file* eksternal?
- 3 • Bagaimana cara menggunakan *pull request* pada *GitHub* agar orang lain dapat menambahkan labirin?
- 4

5 **1.3 Tujuan**

- 6 Tujuan-tujuan yang hendak dicapai melalui penulisan skripsi ini adalah sebagai berikut:
- 7 • Dapat membangun permainan *Snake* menggunakan HTML5.
 - 8 • Dapat menyimpan labirin pada *file* eksternal.
 - 9 • Dapat menggunakan *pull request* pada *GitHub* agar orang lain dapat menambahkan labirin.

10 **1.4 Batasan Masalah**

- 11 Beberapa batasan yang dibuat terkait dengan penggerjaan skripsi ini adalah sebagai berikut:
- 12 • Permainan ini hanya dapat dimainkan menggunakan *web browser*.
 - 13 • *Web browser* yang digunakan sudah mendukung HTML5 Canvas.

14 **1.5 Metodologi**

- 15 Metodologi pada penelitian ini adalah sebagai berikut:
- 16 1. Melakukan studi literatur tentang HTML5, *JavaScript*, *jQuery*, dan *Git*.
 - 17 2. Melakukan analisis dan menentukan objek-objek.
 - 18 3. Merancang algoritma untuk menggambar tubuh ular, pergerakan ular dan membuat labirin.
 - 19 4. Mengimplementasikan keseluruhan algoritma.
 - 20 5. Menambahkan labirin menggunakan *pull request* pada *GitHub*.
 - 21 6. Melakukan pengujian.
 - 22 7. Melakukan penarikan kesimpulan.

23 **1.6 Sistematika Pembahasan**

- 24 Sistematikan penulisan setiap bab pada penelitian ini adalah sebagai berikut:
- 25 1. Bab 1 berisikan latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan dari penelitian yang dilakukan.
 - 26

- 1 2. Bab 2 berisikan dasar-dasar teori yang menunjang penelitian ini. Teori yang digunakan adalah:
2 pengertian *Snake Game*, 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.

¹

BAB 2

²

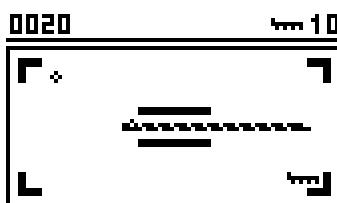
LANDASAN TEORI

³ 2.1 *Snake* Game

⁴ *Snake* Game merupakan permainan mengendalikan ular untuk mendapatkan makanan yang terdapat
⁵ pada labirin. Dalam permainan ini, pemain mengendalikan ular untuk mendapatkan makanan
⁶ sebanyak-banyaknya. Setiap ular memakan makanan, maka skor akan bertambah 1 poin dan tubuh
⁷ ular akan bertambah panjang. Pada umumnya, makanan hanya ada 1 saja pada sebuah labirin.
⁸ Ketika makanan itu sudah termakan oleh ular, makanan tersebut akan ditempatkan secara acak.
⁹ Ular dapat bergerak ke atas, bawah, kiri, dan kanan. Permainan akan berakhir jika ular menabrak
¹⁰ dinding yang terdapat pada labirin atau ular tersebut menabrak tubuhnya sendiri.

¹¹

¹² Permainan *Snake* ini dapat dimainkan secara *singleplayer* atau *multiplayer*. *Singleplayer game*
¹³ adalah permainan yang dapat dimainkan oleh 1 pemain. *Multiplayer game* adalah permainan
¹⁴ yang dapat dimainkan oleh beberapa pemain. Pada umumnya, permainan *Snake* dimainkan secara
¹⁵ *singleplayer*. Contoh *singleplayer game* *Snake* adalah *Snake* pada telepon genggam *Nokia* yang
¹⁶ dapat dilihat pada Gambar 2.1¹ dan contoh *multiplayer game* *Snake* adalah *Slither.io* yang dapat
¹⁷ 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*

1 2.2 HTML5 *Canvas*

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

11
 12 Untuk menambahkan *canvas* pada halaman *HTML*, diperlukan *tag* <canvas>. Pada *listing* 2.1
 13 terdapat potongan kode untuk menambahkan *canvas* pada halaman *HTML*.

14
 151 | <canvas id='canvas' width='500' height='300'>
 162 | Your browser does not support HTML5 Canvas.
 173 | </canvas>

Listing 2.1: Menambahkan *canvas*

18 Diantara *tag* <canvas> dan </canvas>, terdapat teks yang akan ditampilkan jika *browser* tidak
 19 mendukung HTML5 *Canvas*.

20 *Canvas* memiliki beberapa atribut diantaranya adalah:

- 21 • *id* : nama yang digunakan sebagai referensi objek *canvas* yang nantinya akan digunakan pada
 22 *Javascript*.
- 23 • *width* : lebar dari *canvas*.
- 24 • *height* : tinggi dari *canvas*.
- 25 • *title* : judul sebuah elemen.
- 26 • *draggable* : mengambil sebuah objek dan membawanya ke tempat lain

- 1 • *tabindex* : memfokuskan pada suatu elemen jika tombol tab ditekan.
- 2 • *class* : kelas pada elemen. Biasanya digunakan oleh CSS dan *Javascript* untuk mengakses
- 3 elemen tertentu.
- 4 • *dir* : arah penulisan (dari kiri ke kanan atau dari kanan ke kiri)
- 5 • *hidden* : membuat elemen menjadi tersembunyi/tidak terlihat
- 6 • *accesskey* : memberikan petunjuk untuk membuat pintasan *keyboard* pada sebuah elemen.

7 2.3 *Javascript*

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

16
 17 Untuk menambahkan *Javascript* pada sebuah halaman *web* yang dibuat, gunakan *tag* <script>.
 18 Ada 2 cara untuk menambahkan *Javascript* yaitu menambahkan langsung di halaman web terse-
 19 but(*Internal Javascript*) atau menambahkan *file Javascript* terpisah(*External Javascript*). Pada
 20 *Listing 2.2* dan *Listing 2.3* terdapat potongan kode untuk menambah *script* secara langsung di
 21 halaman *web* dan menambah *script* secara terpisah.

```
22
231  <!DOCTYPE html>
242      <html>
253          <body>
264
275              <h1>A Web Page</h1>
286                  <p id="demo">A Paragraph</p>
297
308                  <script>
319                      // tuliskan script di sini
320                  </script>
321
322          </body>
323      </html>
```

Listing 2.2: *Internal Javascript*

```
36
371  <!DOCTYPE html>
382      <html>
```

```

13    <body>
14
15      <h1>A Web Page</h1>
16      <p id="demo">A Paragraph</p>
17
18      <script src="myScript1.js"></script>
19
20    </body>
21 </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.

```

16
171 |   var myVariable;

```

Listing 2.4: Deklarasi variabel

Nilai variabel pada *listing 2.4* adalah *undefined* karena variabel tersebut tidak diberi nilai/*value*. Pada *listing 2.5* terdapat potongan kode untuk mengisi nilai pada variabel.

```

20
211 |   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*.

```

32

```

```
11 | const myConst = 1;
```

Listing 2.6: Deklarasi *constant*

2.3.3 Function

3 *Function* adalah sekumpulan perintah/*statements* untuk menjalankan suatu tugas atau menghitung
 4 nilai. Untuk membuat *function*, digunakan *keyword* ‘*function*’, kemudian diikuti dengan nama
 5 *function* tersebut, parameter yang dituliskan di dalam kurung, dan *statement*/perintah *Javascript*
 6 yang ditulis di dalam kurung kurawal. Parameter pada *function* bisa lebih dari 1 yang penulisanya
 7 dipisahkan oleh koma. *Function* bisa memiliki parameter atau tidak. Pada *Listing 2.7* terdapat
 8 potongan kode untuk membuat *function* penjumlahan 2 buah bilangan.

```
9
```

```
101 | function penjumlahan(angka1,angka2){  
112 |     var hasil = angka1+angka2;  
123 |     return hasil;  
134 | }
```

Listing 2.7: *Function* penjumlahan 2 buah bilangan

14 Setelah membuat *function*, *function* tersebut tidak langsung dieksekusi. Membuat *function*
 15 hanya memberi nama *function* tersebut dan mendeskripsikan apa yang akan dilakukan oleh *function*
 16 tersebut apabila dipanggil. Dengan memanggil *function*, maka *function* akan dieksekusi. Pada
 17 *listing 2.8* terdapat potongan kode untuk memanggil *function* dengan nama penjumlahan.

```
18
```

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

Listing 2.8: Memanggil *function* penjumlahan

2.3.4 Menggambar pada *Canvas*

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

```
24
```

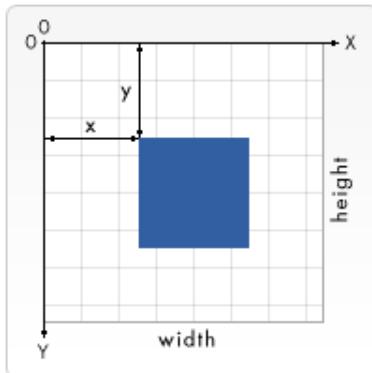
```
251 | var myCanvas = document.getElementById('canvas');  
262 | var context = myCanvas.getContext('2d');
```

Listing 2.9: Menambahkan *drawing context* *canvas*

27 Berdasarkan *listing 2.9*, variabel *myCanvas* menyimpan objek dengan *id* = ‘*canvas*’. *Id* ini
 28 mengacu ke objek *canvas* pada HTML yang memiliki *id* bernama *canvas*. Variabel *myCanvas*
 29 sekarang sudah menyimpan objek *canvas*. Kemudian variabel *context* menyimpan *drawing context*
 30 2D. Sesudah itu, *canvas* tersebut dapat digambar dengan bentuk 2D, garis, kurva, membuat tulisan,
 31 dan menambahkan gambar. Selain untuk menggambar, bentuk-bentuk yang sudah digambar dapat
 32 diberi warna sesuai dengan keinginan.

```
33
```

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, titik ujung kiri kotak biru tersebut berjarak x *pixel* dari sumbu y dan berjarak y *pixel* dari sumbu x.

Menggambar Persegi Panjang

Ada 3 *method* untuk menggambar persegi panjang:

- *fillRect(x,y,width,height)* : menggambar persegi panjang serta mengisi bagian tengah persegi panjang dengan warna.
- *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.

Method tersebut memiliki parameter yang sama. Parameter x dan y untuk menentukan posisi persegi panjang pada canvas. *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.

1 3. Sesudah *path* tersebut sudah dibuat, *path* tersebut dapat dirender menggunakan *stroke* atau
 2 *fill*.

3 Langkah pertama untuk membuat *path* baru adalah dengan menggunakan fungsi *beginPath()*.
 4 Setelah itu, perintah-perintah untuk menggambar dapat digunakan untuk membuat bentuk-bentuk
 5 yang diinginkan. Apabila sudah selesai menggambar, gunakan fungsi *stroke()* untuk menggambar
 6 *outline* dari *path* tersebut atau *fill()* untuk mengisi area *path* tersebut. Setelah itu, gunakan fungsi
 7 *closePath()* untuk menutup bentuk tersebut dengan cara menggambar garis lurus dari posisi titik
 8 terakhir ke titik awal. Fungsi lainnya yang menjadi bagian dari membuat *path* adalah fungsi
 9 *moveTo()*. Fungsi ini diibaratkan seperti mengangkat sebuah pensil dari sebuah titik pada kertas
 10 kemudian menempatkannya pada titik yang diinginkan. Listing 2.10 merupakan fungsi *moveTo()*.

11

121 | **moveTo(x,y);**Listing 2.10: Fungsi *moveTo()*

13 Fungsi *moveTo()* memiliki 2 parameter yaitu x dan y yang merupakan posisi titik pada *canvas*.
 14 Ketika *canvas* sudah diinisialisasi dan fungsi *beginPath()* sudah dipanggil, fungsi *moveTo()* berguna
 15 sebagai penempatan titik awal untuk menggambar. Fungsi *lineTo()* digunakan untuk menggambar
 16 sebuah garis. Listing 2.11 merupakan fungsi *lineTo()*.

17

181 | **lineTo(x,y);**Listing 2.11: Fungsi *lineTo()*

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

23

24 Fungsi *arc()* digunakan untuk menggambar lingkaran atau busur. Listing 2.12 merupakan fungsi
 25 *arc()*.

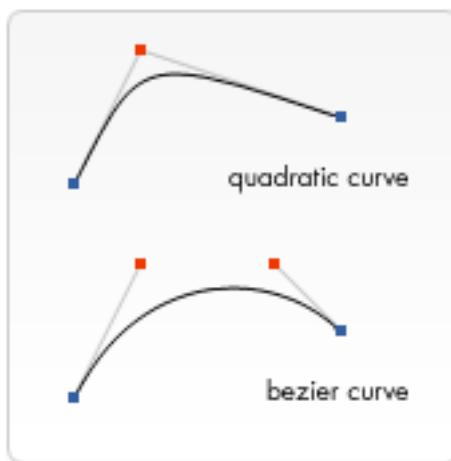
26

271 | **arc(x,y,radius,startAngle,endAngle,anticlockwise);**Listing 2.12: Fungsi *arc()*

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

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

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



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

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

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

2.3.5 Object Oriented Programming Javascript

OOP (*Object Oriented Programming*) adalah sebuah paradigma *programming* yang menggunakan abstraksi untuk membuat objek-objek yang ada pada dunia nyata. Bahasa pemrograman seperti Java, C++, Ruby, Phyton, PHP, dan Objective-C sudah mendukung OOP. Dalam OOP, setiap objek dapat menerima pesan, memproses data dan mengirim pesan ke objek lain. Program yang menggunakan konsep OOP ini mudah untuk dimengerti dan lebih mudah dikembangkan oleh *programmer*.

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

1 Kelas

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

6

```
71 |     function Mobil(){  
82 | }  
93 | }
```

Listing 2.13: Membuat kelas Mobil

10 Objek

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

13

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

Listing 2.14: Membuat instansi mobil

15 Konstruktor

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

20 Properti/Atribut

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

27

```
281 |     function Mobil(merkMobil,beratMobil,warnaMobil,tahunProduksi){  
292 |         this.merkMobil = merkMobil;  
303 |         this.beratMobil = beratMobil; //satuan dalam kg  
314 |         this.warnaMobil = warnaMobil;  
325 |         this.tahunProduksi = tahunProduksi;  
336 |     }  
347 | }
```

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

Listing 2.15: Mendefinisikan properti kelas Mobil pada konstruktor

2 Method

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

```
9
10|    function Mobil(merkMobil,beratMobil,warnaMobil,tahunProduksi){
11|        this.merkMobil = merkMobil;
12|        this.beratMobil = beratMobil; //satuan dalam kg
13|        this.warnaMobil = warnaMobil;
14|        this.tahunProduksi = tahunProduksi;
15|
16|        this.bergerakMaju = function(){
17|            //kode agar mobil bergerak maju
18|        }
19|
20|
21|
22|    var mobil1 = new Mobil('Toyota',1000,'Hitam',2010);
23|    mobil1.bergerakMaju(); //memanggil fungsi untuk bergerak maju
```

Listing 2.16: Membuat dan memanggil method bergerakMaju()

23 2.3.6 Event

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

- 31 • Pengguna mengklik sebuah elemen atau mengarahkan kursor ke sebuah elemen.
- 32 • Pengguna menekan sebuah tombol pada *keyboard*.
- 33 • Pengguna mengatur besar dan menutup *browser window*.
- 34 • Halaman *web* selesai dimuat.
- 35 • *Form* sedang *submit*.

1 • Video sedang dimainkan, dijeda, atau selesai.

2 • Ketika *error* terjadi.

3 Setiap *event* memiliki *event handler*, yang berisikan sekumpulan kode yang akan dijalankan

4 ketika *event* sudah terjadi. *Event handler* juga sering disebut sebagai *event listener*. *Listener*

5 menunggu *event* yang terjadi dan *handler* adalah kode yang dijalankan ketika *listener* mendapatkan

6 *event*/ketika *event* terjadi. Untuk memperjelas cara menggunakan *event*, pada *listing 2.17* terdapat

7 contoh kode untuk menambahkan event pada *button/tombol*.

8

```

91  <html>
102    <title>Event pada tombol</title>
113    <body>
124      <button id='tombol'>Change color</button>
135    </body>
146  </html>

157
168  <script>
179    var btn = document.getElementById('tombol');

180
191    function random(number) {
202      return Math.floor(Math.random()*(number+1));
213    }

224
235    btn.onclick = function() {
246      var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' +
25       random(255) + ')';
267      document.body.style.backgroundColor = rndCol;
278    }
289  </script>
```

Listing 2.17: Menambahkan event pada button

29 Berdasarkan *listing 2.17*, objek *button* dengan *id='tombol'* disimpan di dalam variabel bernama

30 '*btn*'. Ada fungsi bernama '*random*' untuk mengembalikan sebuah nilai acak. Setelah itu ada *event*

31 *handler*. *Event handler property* yang digunakan adalah *onclick*. *Event handler property onclick*

32 mengecek apakah objek(dalam kasus ini objeknya adalah *button*) sudah ditekan. Bila tombol sudah

33 ditekan, maka fungsi akan dieksekusi untuk mengubah warna *background*. Warna RGB tersebut

34 digenerate secara acak menggunakan fungsi *random* yang sudah dibuat sebelumnya. Tidak hanya

35 *event handler property onclick* saja yang dapat digunakan pada halaman *web*. Berikut ini adalah

36 beberapa *event handler property* lainnya:

- 37 • *onfocus* dan *onblur* : aksi akan dijalankan apabila sebuah objek difokuskan/tidak. Biasanya
- 38 digunakan untuk menampilkan informasi tentang cara mengisi *form* ketika difokuskan atau
- 39 menampilkan pesan *error* ketika *form* tersebut diisi dengan nilai yang salah/tidak valid.
- 40 • *ondblclick* : aksi akan dijalankan ketika objek diklik 2 kali/*double click*.

- *window.keypress, window.onkeydown, window.onkeyup* : aksi akan dijalankan 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 *register* pada objek *window* yang merepresentasikan *browser window*.
- *onmouseover* dan *onmouseout* : aksi akan dijalankan 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()*.

```

181
192  var btn = document.getElementById('tombol');
203
214  function bgChange() {
225      var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random
23          (255) + ')';
246      document.body.style.backgroundColor = rndCol;
257  }
268
279  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*.

```

331
342  btn.addEventListener('click', function() {
353      var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random
36          (255) + ')';
374      document.body.style.backgroundColor = rndCol;
385 });

```

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*

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

7

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

12

13 *Touch event* memiliki beberapa event handler properti diantaranya adalah :

- 14 • *touchstart* : *event* ini akan ditembakkan apabila sebuah jari/*stylus* menyentuh permukaan
15 layar.
- 16 • *touchend* : *event* ini akan ditembakkan apabila sebuah atau banyak jari tidak menyentuh
17 permukaan layar/*trackpads*.
- 18 • *touchcancel* : *event* ini akan ditembakkan apabila sebuah atau banyak *touch points* yang
19 terganggu dalam implementasi khusus. Contohnya adalah ketika terlalu banyak *touch points*
20 yang dibuat.
- 21 • *touchmove* : *event* ini akan ditembakkan apabila sebuah atau banyak *touch points* yang
22 berpindah di sekitar permukaan layar.

23 **2.3.7 Membuat Animasi**

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

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

34 Objek yang digambar pada *canvas* dapat menggunakan fungsi yang dimiliki oleh *canvas* atau
35 dengan membuat fungsi sendiri. Hasil yang ada pada *canvas* akan muncul setelah *script* selesai
36 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 berulang kali setiap *delay* milidetik.
- 4 • *setTimeout(function,delay)*: mengeksekusi fungsi 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 2.3.8 Fungsi *Callback*

11 Fungsi *callback* adalah sebuah fungsi yang digunakan pada fungsi lain sebagai argumen yang nantinya
 12 akan dipanggil di dalam fungsi luar untuk menyelesaikan sebuah aksi. Listing 2.20 merupakan
 13 contoh potongan kode pada fungsi callback.

14

```
151
162   function greeting(name) {
173     alert('Hello ' + name);
184   }
195
206   function processUserInput(callback) {
217     var name = prompt('Please enter your name.');
228     callback(name);
239
240   }
251 processUserInput(greeting);
```

Listing 2.20: Contoh fungsi *callback*

26 Pada listing 2.20, fungsi *processUserInput()* menggunakan fungsi *greeting()* sebagai fungsi *callback*.
 27 Setelah *processUserInput* selesai dieksekusi, maka fungsi *greeting()* akan dijalankan. Secara umum,
 28 fungsi *callback* digunakan untuk melanjutkan eksekusi kode program setelah operasi *asynchronous*
 29 selesai dieksekusi. Dengan menggunakan fungsi *callback*, jika sebuah fungsi belum selesai dieksekusi,
 30 maka fungsi *callback* tidak akan dijalankan.

31 2.4 *jQuery*

32 *jQuery* merupakan sebuah *file Javascript* yang sudah termasuk pada halaman *web* [3]. *jQuery*
 33 dapat memilih elemen pada halaman *web* menggunakan *CSS-style selector* dan elemen tersebut
 34 dapat melakukan sesuatu dengan menggunakan *method* *jQuery*. Sebuah fungsi yaitu *jQuery()*
 35 digunakan untuk menemukan satu atau lebih elemen yang berada pada halaman *web*. Fungsi ini
 36 membuat objek yang bernama *jQuery* untuk menyimpan referensi dari elemen yang akan dipilih. *\$()*

1 sering digunakan sebagai pengganti fungsi *jQuery()* dikarenakan penulisannya yang pendek. Fungsi
 2 *jQuery()* hanya memiliki sebuah parameter yaitu sebuah *selector*.

3

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

Listing 2.21: Mendapatkan elemen menggunakan *CSS-style selector*

5 Pada *listing 2.21*, *selector* akan mencari elemen `` yaitu *list* yang merupakan bagian dari
 6 kelas 'hot'. *jQuery* memiliki banyak *method* yang dapat digunakan oleh elemen yang sudah dipilih
 7 menggunakan *selector*. *Method* ini merepresentasikan tugas yang akan dilakukan oleh elemen
 8 tersebut. Setelah memilih elemen, tambahkan *method* yang diawali dengan titik kemudian diikuti
 9 dengan nama *method* beserta parameternya. Titik ini disebut sebagai *member operator*. Setiap
 10 *method* memiliki parameter untuk memberikan detail tentang bagaimana cara untuk mengubah
 11 elemen tersebut. Ada beberapa *method* yang memiliki parameter lebih dari 1. Member operator
 12 menunjukkan bahwa *method* yang terletak setelah *member operator* digunakan untuk mengubah
 13 elemen objek *jQuery* yang terletak pada sebelah kiri *member operator*. Pada *listing 2.22*, terdapat
 14 contoh untuk mengubah kelas dari elemen yang sudah dipilih. Method *addClass* digunakan untuk
 15 mengubah atribut kelas dari elemen *list* menjadi kelas yang bernama 'complete'.

16

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

Listing 2.22: Mengubah kelas dari elemen yang sudah dipilih

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

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

22 2. Membuat objek *jQuery*

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

24 Jika ingin menggunakan *selection* yang sama, maka lebih baik menggunakan objek *jQuery*
 25 yang sama daripada mengulang langkah-langkah yang sudah dijelaskan. Objek *jQuery* tersebut
 26 dapat disimpan pada sebuah variabel. Cara untuk menyimpan referensi objek *jQuery* tersebut
 27 pada variabel adalah membuat variabel yang diawali dengan simbol '\$'. Kemudian variabel tersebut
 28 diisi dengan objek *jQuery* yang diinginkan. Pada *listing 2.23*, variabel `$listItems` menyimpan objek
 29 *jQuery* yang berisi lokasi dari semua elemen list ada pada DOM *tree*.

30

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

Listing 2.23: Menyimpan objek *jQuery*

32 Untuk mengecek apakah sebuah halaman sudah siap untuk menjalankan kode program yang
 33 dibuat, maka gunakan *method ready()*. Maksud dari halaman sudah siap adalah DOM sudah ada
 34 pada halaman *web*. *Listing 2.24* adalah potongan kode untuk mengecek apakah halaman sudah siap.

- 1 Pada listing 2.24, `$(document)` merepresentasikan halaman *web*. Jika halaman sudah siap, maka
 2 kode program yang ada di dalam *method ready()* akan dijalankan. Listing 2.25 adalah pintasan dari
 3 *method ready()* pada objek *document*.

```
4
51 $(document).ready(function(){
62     //ketikan script di sini
73 });


```

Listing 2.24: Mengecek apakah halaman sudah siap

```
8
91 $(function(){
102     //ketikan script di sini
113 });


```

Listing 2.25: Pintasan dari method `$(document).ready()`

12 2.4.1 *Looping*

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

```
25
261 $('li').each(function(){
272     var ids = this.id;
283     $(this).append('<span class="order">' +ids+ '</span>');
294 });


```

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

30 2.4.2 *Chaining*

- 31 Jika ingin menggunakan *method* *jQuery* lebih dari 1 pada elemen yang sama, *method* tersebut dapat
 32 dituliskan secara bersamaan. Setiap *method* yang digunakan harus dipisahkan dengan titik. Pada
 33 listing 2.27 terdapat potongan kode untuk menggunakan beberapa *method* pada elemen yang sama.

11 | `$('li[id!="one"]').hide().delay(500).fadeIn(1400);`

Listing 2.27: Menggunakan beberapa *method* pada elemen yang sama

2 Pada listing 2.27, ada 3 *method* yaitu *hide()*, *delay()* dan *fadeIn()* digunakan pada sebuah elemen
3 yang sama. *Method hide()* digunakan untuk menyembunyikan elemen, *method delay()* digunakan
4 untuk menjeda waktu/pause, dan *method fadeIn()* digunakan untuk memudarkan elemen. Proses
5 menambahkan beberapa *method* pada *selector* yang sama disebut sebagai *chaining*. *Method-method*
6 yang digunakan untuk memperbarui elemen dapat menggunakan *chaining* sedangkan *method* yang
7 digunakan untuk mendapatkan informasi dari elemen tidak dapat menggunakan *chaining*.

8 **2.4.3 Mendapatkan dan Mengubah Konten Elemen**

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

- 14 1. *html()* : *method* ini memberikan konten HTML yang baru kepada semua elemen yang sesuai
15 dengan *jQuery selection*.
- 16 2. *text()* : *method* ini memberikan *text/tulisan* yang baru kepada setiap elemen yang sesuai
17 dengan *jQuery selection*.
- 18 3. *replaceWith()* : *method* ini menggantikan konten setiap elemen yang sesuai dengan *jQuery*
19 *selection* dengan konten yang baru. *Method* ini juga mengembalikan elemen-elemen yang
20 sudah diganti.

- 21 4. *remove()* : *method* ini akan menbuang semua elemen yang sesuai dengan *jQuery selection*.

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

- 24 1. *attr()* : *method* ini berfungsi untuk mendapatkan atau mengubah atribut secara spesifik.
- 25 2. *removeAttr()* : *method* ini berfungsi untuk menbuang atribut dari sebuah elemen.
- 26 3. *addClass()* : *method* ini berfungsi untuk menambah nilai atribut *class*. *Method* ini tidak
27 menggantikan nilai atribut yang sudah ada.
- 28 4. *removeClass()* : *method* ini berfungsi untuk menbuang nilai dari atribut *class*. *Method* ini
29 tidak menbuang nilai atribut dari kelas lainnya.

30 **2.4.4 Mendapatkan dan Mengubah Properti CSS**

31 *Method* untuk mengubah dan mendapatkan properti CSS adalah *method css()*. Untuk mendapatkan
32 nilai properti CSS, tentukan nama properti yang ingin didapat pada method CSS. Apabila pada
33 hasil selection memiliki elemen lebih dari 1, maka hasil yang dikembalikan adalah nilai properti

1 CSS dari elemen pertama. *Listing 2.28* adalah potongan kode untuk mendapatkan nilai background
 2 color dari elemen list pertama dan akan disimpan pada variabel bernama '*backgroundColor*'. Hasil
 3 dari warna tersebut akan dikembalikan dalam nilai RGB.

4

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

Listing 2.28: Mendapatkan nilai warna *background color* dari elemen *list* pertama

6 Untuk mengubah nilai properti CSS, tentukan nama properti sebagai argumen pertama dan
 7 tentukan nilai untuk properti yang sudah dipilih pada argumen pertama sebagai argumen kedua.
 8 Antara argumen pertama dan kedua dipisahkan dengan koma. *Method* ini akan mengubah semua
 9 elemen yang sesuai dengan *selection*. Dengan *object literal notation*, kita dapat mengubah sejumlah
 10 properti lainnya dalam *method* yang sama. Aturan penulisan pada *object literal notation* adalah
 11 properti dan nilai properti dituliskan di dalam kurung kurawal, antara properti dan nilainya
 12 dipisahkan dengan titik dua(:), dan koma memisahkan setiap pasangan properti. *Listing 2.29* adalah
 13 potongan kode untuk mengubah *background color* dari semua elemen *list* menjadi warna merah dan
 14 *listing 2.30* adalah potongan kode untuk mengubah sejumlah properti menggunakan *object literal*
 15 *notation*.

16

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

Listing 2.29: Mengubah warna background color semua elemen list

18

```
191| $('li').css({  
202|   'background-color': 'red',  
213|   'font-family': 'Courier'  
224| });
```

Listing 2.30: Mengubah warna background color dan jenis font untuk semua elemen list menggunakan
object literal notation

23 2.4.5 Event

24 Sama seperti *Javascript*, pada *jQuery* juga dapat ditambahkan *event*. *Method* yang digunakan untuk
 25 menambahkan *event* adalah *on()*. Untuk memperjelas cara menggunakan *method on()*, terdapat
 26 potongan kode pada *listing 2.28*. Untuk menambahkan *event*, maka pertama harus memilih elemen
 27 yang akan ditambahkan *event* dengan menggunakan *selector*. Pada *listing 2.31*, elemen yang dipilih
 28 adalah semua elemen *list*, kemudian tambahkan *method on()*. *Method .on()* memiliki 2 parameter
 29 yaitu *event* yang akan digunakan dan perintah yang akan dilakukan apabila *event* tersebut terjadi
 30 pada elemen yang dipilih. Perintah dapat berupa sebuah fungsi anonim yaitu fungsi yang dibuat
 31 langsung atau memanggil sebuah fungsi yang sudah ada. Pada *listing 2.31*, *event* yang digunakan
 32 adalah '*click*' dan akan diberikan sebuah fungsi anonim yang tugasnya adalah menambahkan atribut
 33 *class* yang bernilai '*complete*'.

34

```
11 |     $('li').on('click', function(){
12 |         $(_this).addClass('complete');
13 |     });
14 | 
```

Listing 2.31: Menambahkan atribut *class* pada setiap *list* menggunakan event '*click*'

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

- 5 • UI : *focus, blur, change*
- 6 • Keyboard : *input, keydown, keyup, keypress*
- 7 • Mouse : *click, dblclick, mouseup, mousedown, mouseover, mouseout, hover*
- 8 • Form : *submit, select, change*
- 9 • Document : *ready, load, unload*
- 10 • Browser : *error, resize, scroll*

11 Setiap fungsi *event handling* menerima sebuah *event object*. *Event object* memiliki properti dan
12 *method* yang berhubungan dengan *event* yang sudah terjadi. Contoh kode program dapat dilihat
13 pada *listing 2.32*. Pada *listing 2.32*, pada parameter *function* terdapat parameter yang bernama
14 *event*. Parameter yang bernama *event* ini adalah *event object*. Kemudian tipe dari *event object*
15 tersebut disimpan pada variabel yang bernama *eventType*.

```
16 |
17 |     $('li').on('click', function(event){
18 |         eventType = event.type;
19 |     });
19 | 
```

Listing 2.32: Mendapatkan tipe *event* dari *event object*

20 *Event object* memiliki 7 properti, diantaranya adalah:

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

1 2.4.6 Mengubah dan Mendapatkan Dimensi dari Elemen

2 Pada CSS, setiap elemen yang ada pada halaman *web* akan dianggap sebagai sebuah kotak. Sebuah
3 kotak memiliki *padding*, *border* dan *margin*. Jika dimensi dari kotak tersebut diatur pada CSS,
4 kotak tersebut belum termasuk *padding*, *border* dan *margin*. Berikut adalah *method-method* yang
5 digunakan untuk mengubah dan mendapatkan dimensi dari kotak :

- 6 • *height()* : tinggi dari kotak (tidak termasuk *margin*, *border* dan *padding*)
7 • *width()* : lebar dari kotak (tidak termasuk *margin*, *border* dan *padding*)

8 Berikut adalah method yang digunakan untuk mendapatkan dimensi kotak :

- 9 • *innerHeight()* : tinggi dari kotak ditambah *padding*
10 • *innerWidth()* : lebar dari kotak ditambah *padding*
11 • *outerHeight()* : tinggi dari kotak ditambah *padding* dan *border*
12 • *outerWidth()* : lebar dari kotak ditambah *padding* dan *border*
13 • *innerHeight(true)* : tinggi dari kotak ditambah *padding*, *border* dan *margin*
14 • *innerWidth(true)* : lebar dari kotak ditambah *padding*, *border* dan *padding*

15 2.4.7 AJAX

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

- 19 • *Live search/autocomplete* yang dapat ditemukan pada *Google search*.
20 • *Website* dengan konten *user-generated* yang dapat menampilkan informasi pada *website*,
21 contohnya adalah *Twitter* dan *Flickr*
22 • Menambah barang ke keranjang pada situs belanja *online*. Barang yang ditambahkan akan
23 diperbarui tanpa meninggalkan halaman tersebut.
24 • *Register username* pada *website* yang akan mengecek apakah *username* sudah digunakan oleh
25 orang lain atau tidak.

26 AJAX menggunakan *asynchronous processing model* yang artinya adalah pengguna dapat mela-
27 kukan hal lain ketika *browser* sedang menunggu data untuk dimuat. Ketika halaman *web* sudah
28 dimuat dan jika pengguna ingin mengubah sesuatu pada *browser*, maka pengguna biasanya akan
29 memuat ulang halaman *web*. Hal ini akan membuat pengguna harus menunggu halaman *web*
30 selesai dimuat dan dirender oleh *browser*. Dengan menggunakan AJAX, kita dapat mengubah
31 konten sebuah elemen jika ingin memperbarui sebagian halaman *web*. Caranya adalah dengan
32 menambahkan *event* dan request konten baru ke server menggunakan *asynchronous request*. Ketika
33 data sedang dimuat, maka halaman *web* akan tetap dimuat dan pengguna dapat tetap berinteraksi

1 dengan halaman *web*. Setelah server merespon *request*, *event special* AJAX akan *trigger* bagian lain
 2 dari *script* yang membaca data dari server dan memperbarui hanya sebuah bagian dari halaman *web*.
 3 Hal ini akan membuat data dimuat lebih cepat dan pengguna dapat berinteraksi dengan halaman *web*.

4
 5 Langkah-langkah AJAX mengirim *request* dan menerima respon dari server adalah : Pertama,
 6 *browser* meminta informasi dari server. Permintaan tersebut dapat mengandung informasi yang
 7 dibutuhkan oleh server untuk diproses. Browser mengimplementasi sebuah objek yang bernama
 8 *XMLHttpRequest* untuk menangani *Ajax request*. Browser tidak menunggu respon dari server.
 9 Setelah mengirim *request* dan server menerima *Ajax request*, server akan mengirimkan HTML atau
 10 data dalam format lainnya seperti JSON(*Javascript Object Notation*) atau XML. Setelah server
 11 selesai merespon *request* tersebut, *browser* akan menjalankan *event*. Event ini dapat digunakan
 12 untuk *trigger* fungsi-fungsi *Javascript* yang akan memproses data dan menambahkannya ke sebuah
 13 bagian/element dari halaman *web*.

14 **Ajax Request dan Response**

15 Untuk membuat *Ajax request*, *browser* menggunakan objek *XMLHttpRequest*. Ketika server me-
 16 respon *request* dari browser, objek *XMLHttpRequest* yang sama akan memproses hasilnya. Pada
 17 *listing 2.33* terdapat potongan kode untuk membuat *Ajax request*.

18
 191 | **var** xhr = **new** XMLHttpRequest();
 202 | xhr.open('GET', 'data/test.json', **true**);
 213 | xhr.send();

Listing 2.33: Membuat *Ajax request*

22 Berdasarkan *listing 2.33*, hal pertama yang dilakukan adalah membuat objek *XMLHttpRequest*
 23 yang disimpan pada variabel bernama *xhr*. Kemudian *method open()* berfungsi untuk menyiapkan
 24 *request*. *Method open* memiliki 3 parameter yaitu HTTP *method*, *url* dari halaman yang akan
 25 menangani *request*, dan tipe data *boolean* yang menentukan apakah *request* tersebut *asynchronous*
 26 atau tidak. Pada *listing 2.33* *request* tersebut menggunakan HTTP *method* yaitu *GET*, *url* halamanya
 27 adalah '*data/test.json*', dan *asynchronous*. *Method .send()* digunakan untuk mengirimkan *request*
 28 yang sudah disiapkan. Informasi tambahan dapat dikirimkan ke server yang dituliskan pada
 29 *parameter method .send()*. Sesudah mengirimkan *request* dan menerima respon dari server, data
 30 yang diterima akan diproses. Pada *listing 2.34* terdapat potongan kode untuk menerima respon
 31 dan memproses data dari server. Setelah *browser* menerima dan memuat respon dari server, event
 32 *onload* akan dijalankan dan akan *trigger* sebuah fungsi. Di dalam fungsi tersebut, akan dicek status
 33 dari objek tersebut untuk memastikan apakah respon dari server tidak ada masalah.

34
 351 | xhr.onload = **function**(){
 362 | **if**(xhr.status === 200){
 373 | **//proses data yang sudah diterima dari server**
 384 | }
 395 | }

Listing 2.34: Memproses respon yang didapat dari server

1 ***jQuery dan AJAX Request***

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

- 3 • *load()* : memuat HTML dalam sebuah elemen.
- 4 • *\$.get()* : memuat data menggunakan *method HTTP GET*. *Method* ini digunakan untuk *request*
5 data dari server.
- 6 • *\$.post()* : memuat data menggunakan *method HTTP POST*. *Method* ini digunakan untuk
7 mengirim data ke server.
- 8 • *\$.getJSON()* : memuat data JSON menggunakan *GET*.
- 9 • *\$.getScript()* : memuat dan mengeksekusi data pada *Javascript* menggunakan *GET*.
- 10 • *\$.ajax()*: *method* ini digunakan untuk menjalankan semua *request* yang sudah dijelaskan pada
11 poin sebelumnya.

12 ***jQuery dan AJAX Response***

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

- 16 • *responseText* : mengembalikan data text
- 17 • *responseXML* : mengembalikan data XML
- 18 • *status* : kode status
- 19 • *statusText* : deskripsi dari status
- 20 • *done()* : *method* yang digunakan untuk mengeksekusi kode apabila *request* berhasil
- 21 • *fail()* : *method* yang digunakan untuk mengeksekusi kode apabila *request* gagal
- 22 • *always()* : *method* yang digunakan untuk mengeksekusi kode apabila *request* berhasil atau
23 gagal
- 24 • *abort()* : menghentikan komunikasi

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

- 29 • *type* : mendapatkan nilai *GET* dan *POST* tergantung dari *request*. *Request* tersebut dapat
30 dibuat menggunakan *HTTP GET* atau *POST*.
- 31 • *url* : *request* yang akan dikirimkan

- 1 • *data* : data yang akan dikirimkan ke server bersama dengan *request*
- 2 • *success* : sebuah fungsi yang akan dijalankan apabila *Ajax request* berhasil.
- 3 • *error* : sebuah fungsi yang akan dijalankan apabila terdapat *error* pada *Ajax request*.
- 4 • *beforeSend* : sebuah fungsi yang akan dijalankan sebelum *Ajax request* dikirimkan.
- 5 • *complete* : pengaturan yang akan dijalankan setelah *event success* atau *error*
- 6 • *timeout* : angka dalam milisekon untuk menunggu sebelum *event* akan gagal

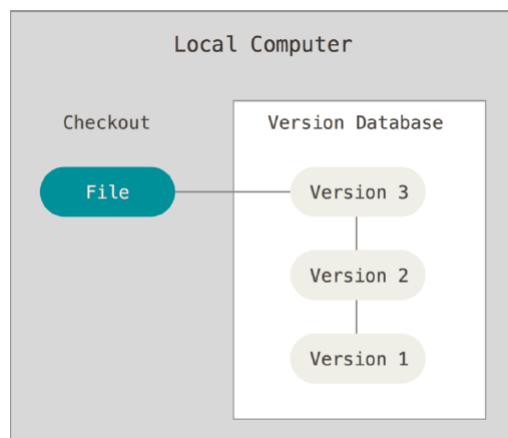
7 2.5 Git

8 2.5.1 Version Control

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

16 Local Version Control System

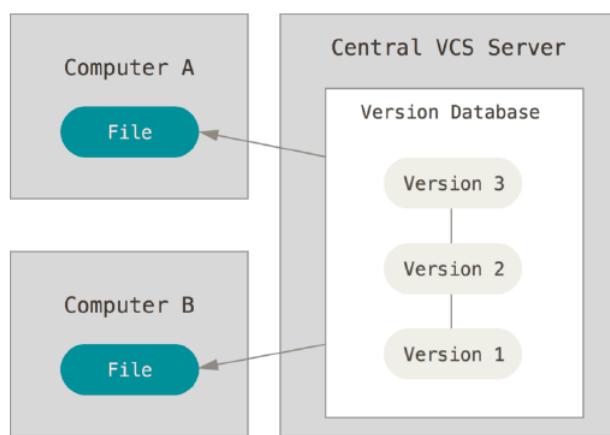
17 *Local Version Control System* memiliki sebuah basis data yang menyimpan semua perubahan pada
18 *file* dalam *revision control*. Salah satu VCS tools yang cukup terkenal adalah RCS yang masih
19 digunakan oleh banyak komputer hingga sekarang. Cara kerja RCS adalah menyimpan *patch sets*
20 yang merupakan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



Gambar 2.5: Local Version Control

1 Centralized Version Control System

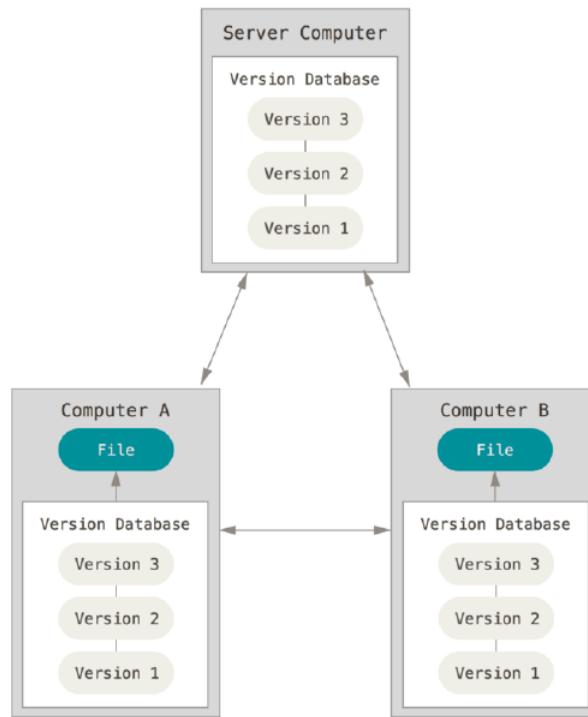
2 *Local Version Control System* menjadi kurang efektif, bila ada beberapa orang yang berkolaborasi
3 dengan pengembang. Karena pada *Local Version Control System*, *version control* dimiliki oleh
4 masing-masing komputer sehingga pengguna tidak tahu apakah *file* tersebut sudah diubah oleh kolab-
5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



Gambar 2.6: Centralized Version Control

12 Distributed Version Control System

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



Gambar 2.7: Distributed Version Control

¹ 2.5.2 Git

² Git merupakan sebuah *version control* namun berbeda dengan VCS lainnya dilihat dari cara me-
³ nyimpan datanya. Sistem seperti CVS, *Subversion*, *Perforce*, *Bazaar* menyimpan data sebagai
⁴ sekumpulan *file* dan perubahan setiap *file* disimpan setiap waktu. Pada Git, data tersebut dianggap
⁵ sebagai sekumpulan *snapshot* dari *miniature filesystem*. Setiap *commit* atau menyimpan proyek,
⁶ Git seolah-olah mengambil gambar untuk melihat seperti apa *file* yang terlihat pada saat itu dan
⁷ menyimpannya sebagai referensi pada *snapshot* tersebut. Singkatnya, apabila tidak ada *file* yang
⁸ diubah, Git tidak akan menyimpan *file* lagi.

⁹

¹⁰ Hampir semua operasi pada Git dapat dilakukan secara lokal. Ketika ingin menlihat histori
¹¹ suatu proyek, Git akan mengambil data histori tersebut dari basis data lokal, sehingga tidak perlu
¹² memintanya ke *server*. Selain itu, pengguna dapat bekerja secara *offline*. Pada sistem lain seperti
¹³ *Perforce*, pengguna tidak dapat melakukan banyak hal jika tidak terkoneksi ke *server* dan pada
¹⁴ CVS, pengguna dapat mengubah *file* tetapi tidak dapat *commit* ke basis data. Pada Git, pengguna
¹⁵ dapat *commit* dikarenakan Git memiliki basis data lokal.

¹⁶

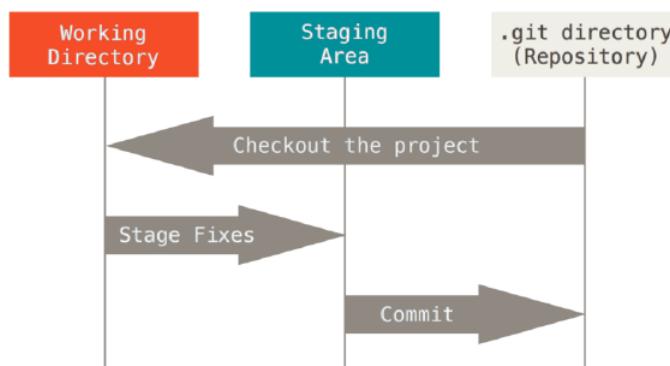
¹⁷ Git memiliki 3 *state* utama pada *file* yaitu:

- ¹⁸ • *committed* : data sudah tersimpan di basis data lokal.
- ¹⁹ • *modified* : *file* sudah diubah namun belum di*commit* ke basis data.
- ²⁰ • *staged* : menandai file yang sudah dimodifikasi dalam versi sekarang untuk di*commit*.

²¹ 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 dicompressed 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 menunjukan *working tree*, *staging area* dan *Git directory*.



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

Workflow pada *Git* adalah sebagai berikut :

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

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

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

24 Mendapatkan *Git Repository*

Untuk mendapatkan *Git repository* ada 2 cara yaitu : menjadikan sebuah proyek yang terdapat pada direktori lokal yang belum dalam *version control* lalu menjadikannya sebagai *Git repository*

1 dan dengan *clone Git repository* yang sudah ada.

2

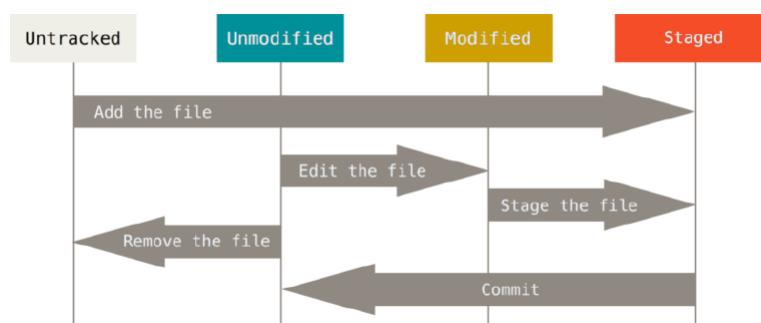
3 Jika memiliki direktori proyek yang belum dalam *version control* dan ingin mengontrolnya
 4 menggunakan *Git*, hal pertama yang harus dilakukan adalah dengan membuka direktori proyek.
 5 Perintah untuk membuat repository pada *Windows* adalah dengan mengetikan perintah `$ cd /c/u-`
 6 `ser/my_project`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

11

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

16 Record Perubahan pada Repository

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



Gambar 2.9: Siklus hidup pada status file

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

30

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

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

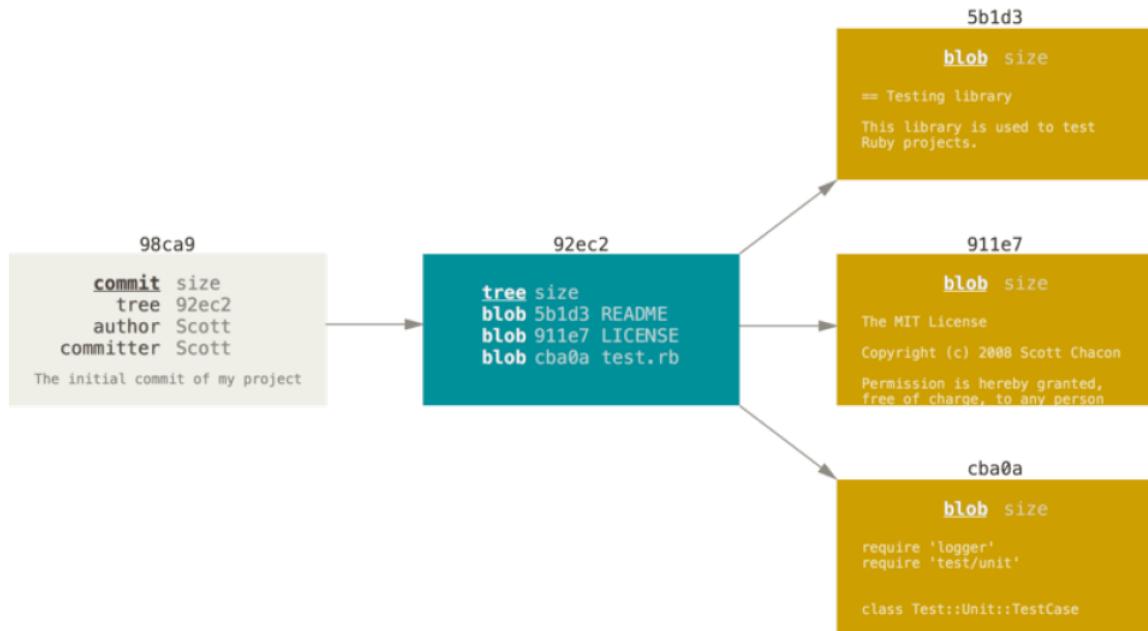
11 *Glob pattern* adalah *regular expression* yang digunakan oleh *shells*. Tanda bintang(*) untuk
12 nol atau beberapa karakter, [abc] untuk karakter apa saja yang berada di dalam kurung siku,
13 tanda tanya(?) untuk sebuah karakter apa saja dan tanda kurung siku dengan tanda strip(-) untuk
14 karakter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

15
16 Perintah *git commit* digunakan untuk *commit file* yang sudah diubah dan ditambahkan. *File*
17 tersebut harus sudah di-stage dengan menggunakan perintah *git add*. *File* yang belum di-stage akan
18 berada dalam state *modified* meskipun sudah melakukan *commit*. Untuk menambahkan keterangan
19 tentang *file* yang dicommit dapat dituliskan perintah *git commit -m* yang diikuti dengan keterangan
20 yang ingin disampaikan.

21 2.5.3 *Git Branching*

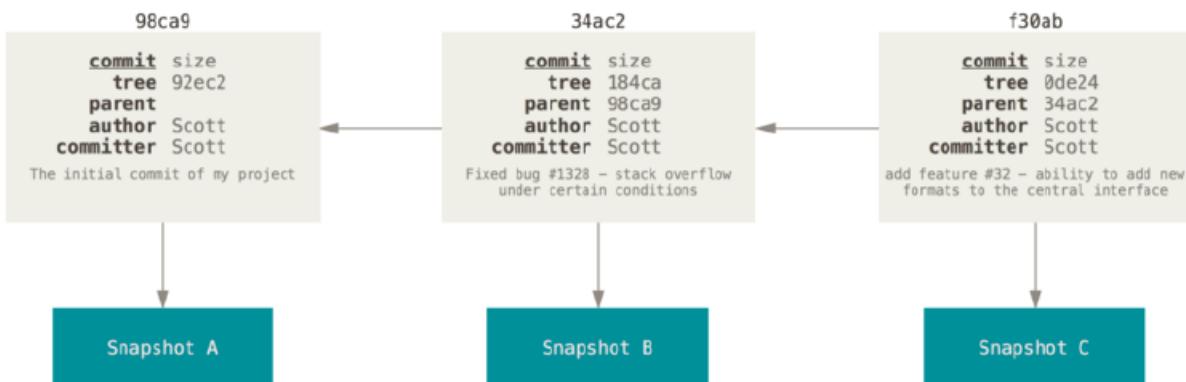
22 *Branching* artinya membuat dan mengerjakan sebuah proyek di tempat yang berbeda namun
23 masih dalam repository yang sama sehingga tidak mengubah proyek utama. Ketika *commit*, *Git*
24 menyimpan objek *commit* yang memiliki sebuah *pointer* pada *snapshot* sebuah konten yang sudah
25 dalam state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

27
28 Misalkan seorang pengguna memiliki 3 *file*, kemudian file tersebut semuanya di-stage dan
29 *commit*. *Staging file* akan mengkomputasi *checksum* untuk setiap *file*, menyimpan versi tersebut
30 pada *Git repository*(hal ini dapat disebut juga sebagai *blobs*), dan menambah *checksum* tersebut ke
31 *staging area*. Lalu *Git* melakukan *checksum* pada setiap *subdirectory* dan menyimpan ketiga objek
32 tersebut pada *Git repository*. Sesudah itu *Git* akan membuat objek *commit* yang mengandung
33 *metadata* dan *pointer* ke proyek *root* sehingga dapat melihat *snapshot* tersebut pada setiap versi.
34 Sekarang, *Git repository* memiliki 5 objek yaitu 3 *blob* yang merepresentasikan 3 *file*, sebuah *tree*
35 yang mengandung isi direktori dan memberi nama *blob* berdasarkan nama *file* yang dicommit, dan
36 sebuah *commit* dengan *pointer* ke *root tree* dan semua *commit metadata*. Gambar 2.10 merupakan
37 *tree* dari penjelasan tersebut.



Gambar 2.10: Commit dan tree dari file yang dicommit

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



Gambar 2.11: Commit dan parent dari commit

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

7

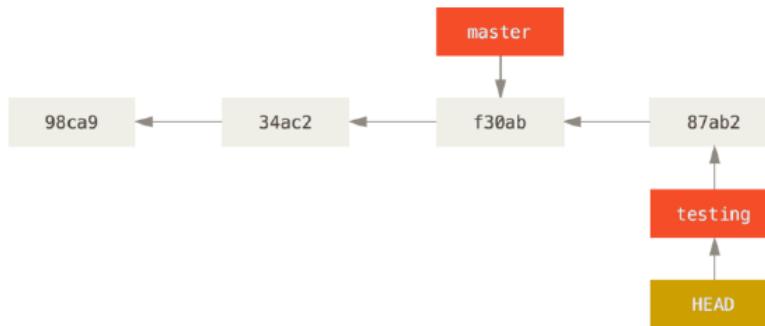
8 Untuk membuat *branch* baru, gunakan perintah *git branch* diikuti dengan nama *branch*. *Git*
9 menggunakan *pointer* yang disebut dengan *HEAD* untuk mengetahui bahwa pengguna sedang
10 berada dalam *branch* tertentu. Bila membuat *branch* baru, posisi *HEAD* tetap berada pada *branch*
11 yang sekarang.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



Gambar 2.12: *Pointer HEAD menunjuk branch master*

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

6



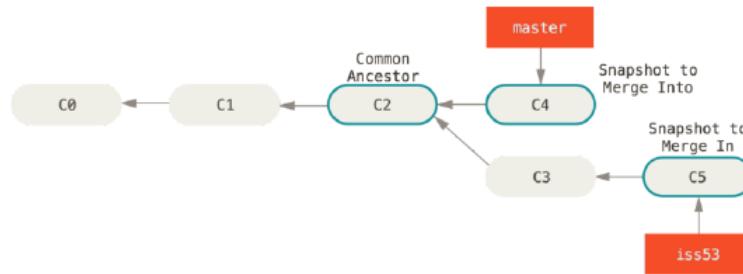
Gambar 2.13: *Pointer HEAD beserta branch testing*

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

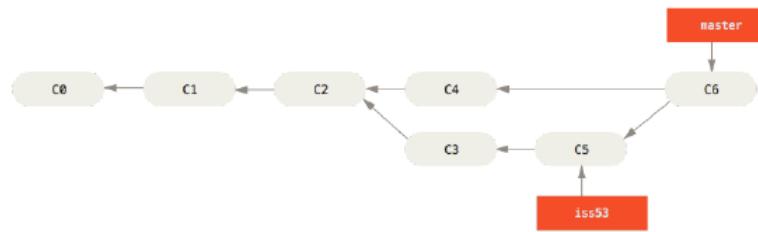
13 Basic Merging

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

- 1 *three way merge* dan secara otomatis akan membuat *commit* yang baru seperti yang terlihat pada
- 2 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*

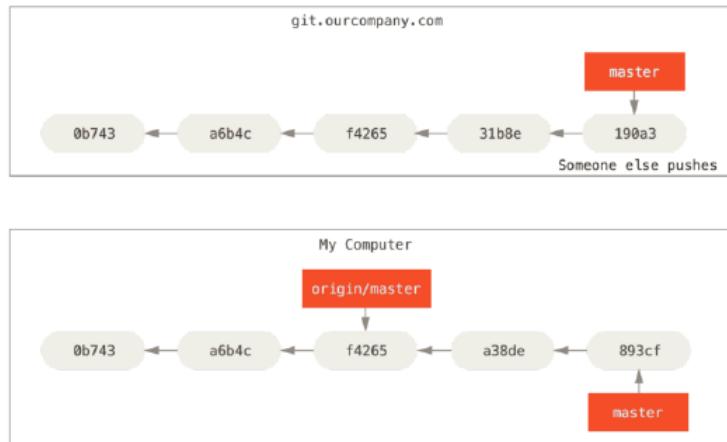
- 3 *Merge* pada *Git* mungkin akan menimbulkan konflik. Hal ini dapat terjadi apabila *file* yang
- 4 sama pada kedua *branch* tersebut diubah pada bagian yang sama. Ketika mengetikan perintah *git*
- 5 *merge*, maka *Git* tidak akan membuat *merge commit* secara otomatis. Proses *merge* akan dijeda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

10 *Remote Branches*

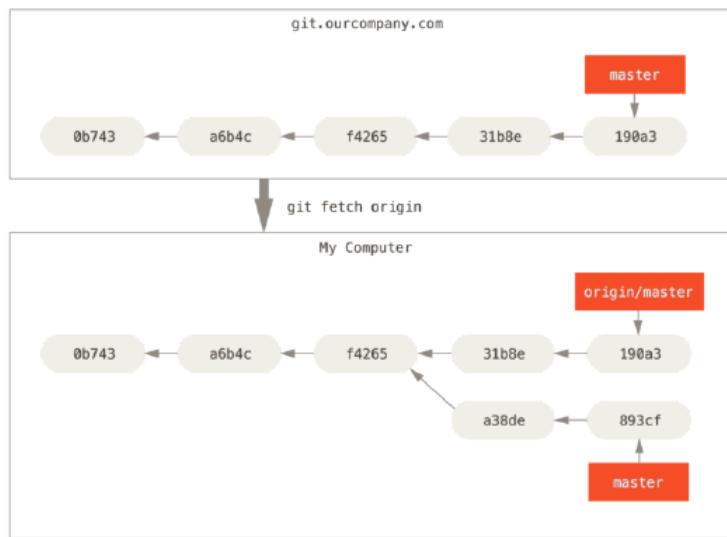
- 11 *Remote-tracking branches* adalah referensi dari *state remote branches*. Referensi tersebut merupakan
- 12 referensi lokal yang hanya dapat dipindahkan oleh *Git* untuk memastikan jika referensi tersebut
- 13 merepresentasikan *state* dari *remote repository*. <*remote*>/<*branches*> merupakan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

1 yang terlihat pada Gambar 2.17.

2



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



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

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

6
 7 *Check out* *branch* lokal dari *remote-tracking branch* secara otomatis akan membuat *tracking*
 8 *branch*. *Tracking branch* adalah *branch* lokal yang memiliki hubungan langsung dengan *branch*
 9 *remote*. Jika berada pada *tracking branch* dan mengetikan perintah *git pull*, secara otomatis
 10 *Git* mengetahui *server* mana yang akan di-*fetch* dan *branch* apa yang akan di-*merge*. Bila *clone*
 11 *repository*, maka secara otomatis akan membuat sebuah *branch* yang bernama *master* yang men-*track*
 12 *origin/master*. Untuk mengatur *tracking branch*, perintah yang digunakan adalah *git checkout -b*
 13 <*branch*> <*remote*>/<*branch*>. *Git* menyediakan perintah *git checkout -track <remote>/<branch>*
 14 sebagai shortcut dari perintah *checkout* sebelumnya. Perintah *git checkout* juga dapat digunakan

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

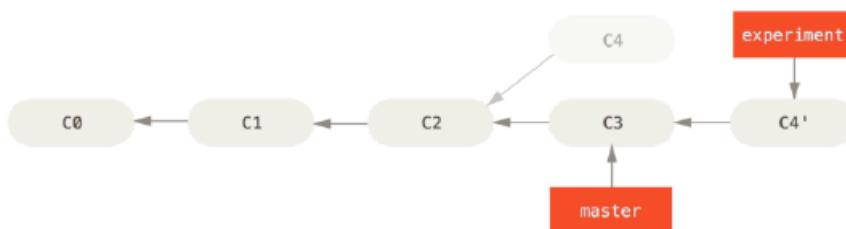
11 Perintah *git fetch* akan mengambil semua perubahan yang ada pada *server* yang tidak dimiliki
 12 oleh *branch* *lokal*, tetapi tidak mengubah *working directory* yang sesuai dengan *branch* *remote*.
 13 Perintah *git pull* digunakan untuk mengubah *working directory*. Perintah ini akan melihat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

16

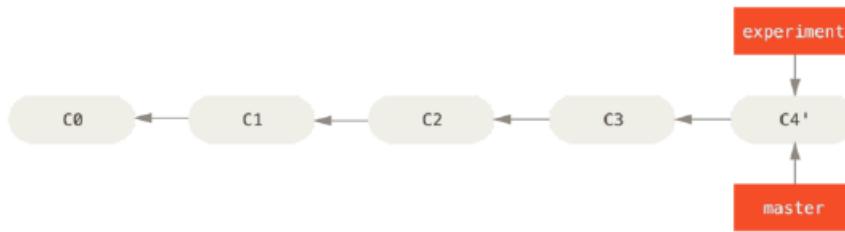
17 Branch pada *remote* dapat dihapus dengan menggunakan opsi *-delete* pada perintah *git push*.
 18 Branch pada *remote* tidak sepenuhnya dihapus, tetapi hanya *pointernya* saja yang dihilangkan.
 19 Jika *branch* tidak sengaja terhapus, maka data pada branch dapat dikembalikan/di *back-up*.

20 Rebasing

21 Selain *merge*, ada cara lain untuk menggabungkan kedua *branch* yaitu *rebasing*. Cara kerja dari
 22 *rebasing* adalah mencari *ancestor* yang sama dari kedua *branch*, mendapatkan perbedaan setiap
 23 *commit* pada *branch* saat ini, menyimpan perbedaan tersebut pada *file* sementara, mengatur
 24 ulang *branch* ke *commit* yang sama dengan *branch* yang akan direbase, dan menerapkan setiap
 25 perubahannya. Contoh *rebasing* dapat dilihat pada Gambar 2.18. Commit C4 pada *branch*
 26 *experiment* berpindah dari C4 ke C4' yang berada di atas C3. Setelah *rebasing*, *merge* kedua *branch*
 27 tersebut sehingga hasilnya terlihat seperti pada Gambar 2.19. Untuk *rebasing*, gunakan perintah *git*
 28 *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* menggunakan untuk *Git hosting*, *code review*, *issue tracking* dan lainnya.

Fork

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

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



Gambar 2.20: Tombol 'Fork'

Berikut adalah langkah-langkah untuk berkolaborasi dalam GitHub:

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

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

6 ***Pull Request***

7 *Pull Request* membuka tempat diskusi untuk *owner(pemilik repository)* dan kontributor sehingga 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.

14

15 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*.

23

1

BAB 3

2

ANALISIS

3 3.1 Analisis Permainan *Snake* yang Sudah Ada

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

10

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

15 3.1.1 Ular dan Makanan

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

21

22 Makanan pada *Slither.io* berbentuk lingkaran. Makanan ini ada yang berukuran besar dan ada
23 yang berukuran kecil. Makanan ini tersebar pada labirin, jumlahnya sangat banyak dan warnanya
24 bermacam-macam. Gambar 3.2 merupakan sekumpulan makanan yang terdapat pada labirin.
25 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
2 pada Gambar 3.3. Pada permainan ini apabila kepala ular sudah dekat dengan makanan, maka
3 kepala ular akan terlihat sedang membuka mulutnya. Makanan yang terdapat pada permainan ini
4 ada 2 macam yaitu makanan biasa dan makanan bonus seperti yang terlihat pada Gambar 3.4.
5 Makanan biasa memiliki skor 1 poin dan makanan bonus memiliki skor 10 poin. Makanan bonus
6 muncul secara acak dan memiliki batas waktu untuk berada pada labirin. Makanan bonus tidak
7 hanya menambah skor lebih banyak 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*

¹ 3.1.2 Pergerakan Ular

- ² Ular pada *Slither.io* digerakan dengan menggunakan *keyboard* dan *mouse*. Tombol ke kiri akan
³ membuat ular bergerak berlawanan arah jarum jam dan tombol ke kanan akan membuat ular
⁴ bergerak searah jarum jam. Semakin lama tombol ditekan, maka ular akan berbelok lebih cepat.
⁵ Kursor pada *mouse* membuat ular bergerak ke arah posisi kursor tersebut. Ular dapat melaju
⁶ dengan cepat(*speed up*) dengan menekan tombol *mouse* kiri seperti yang terdapat pada Gambar 3.5.
⁷ Ketika ular sedang melaju dengan cepat, total skor yang didapat akan berkurang.



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

- ⁸ Ular pada *Snake Nokia* hanya dapat bergerak ke atas, ke bawah, ke kiri dan ke kanan. Ular
⁹ dapat digerakan menggunakan tombol angka pada telepon genggam Nokia yaitu tombol 8 untuk
¹⁰ bergerak ke atas, tombol 4 untuk bergerak ke kiri, tombol 6 untuk bergerak ke kanan dan tombol 2
¹¹ untuk bergerak ke bawah. Kecepatan ular juga dapat dipilih. Semakin tinggi tingkat, maka ular
¹² akan bergerak semakin cepat.

¹³ 3.1.3 Labirin

- ¹⁴ Labirin pada *Slither.io* hanya ada 1 saja. Labirin ini berbentuk lingkaran yang sisinya merupakan
¹⁵ dinding. Apabila ular menabrak dinding labirin, maka permainan akan berakhir. Labirin ini
¹⁶ cukup besar sehingga sangat kecil kemungkinan ular untuk menabrak dinding labirin. Gambar 3.6
¹⁷ menunjukkan peta labirin pada *Slither.io*. Pada peta labirin tersebut terdapat sekumpulan titik
¹⁸ bewarna abu-abu yang merepresentasikan makanan.



Gambar 3.6: Peta labirin pada *Slither.io*

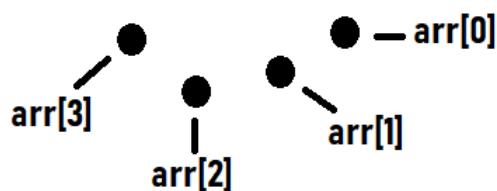
- ¹⁹ Labirin pada *Snake Nokia* lebih bervariasi dibandingkan dengan *Slither.io*. Pada permainan
²⁰ ini pemain dapat memilih labirin yang tersedia. Semakin tinggi level, maka labirin akan semakin
²¹ rumit.

3.2 Analisis Sistem yang Dibangun

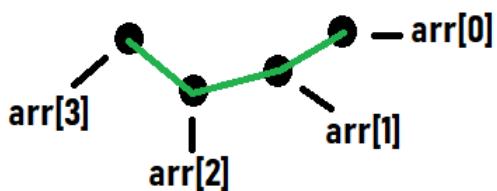
- 2 Open Source Snake 360 memiliki cara bermain yang mirip seperti permainan Snake pada umumnya.
- 3 Perbedaan antara Open Source Snake 360 dengan permainan *Snake* pada umumnya adalah Open
- 4 Source Snake 360 dapat menambahkan labirin sendiri.

3.2.1 Menggambar Ular dan Apel

- 6 Tubuh ular dibuat menggunakan sekumpulan *line/garis* pendek. Setiap bagian tubuh ular memiliki
- 7 panjang sebesar 2 *pixel* dan lebar tubuhnya sebesar 10 *pixel*. Panjang setiap bagian tubuh ular tidak
- 8 1 pixel karena akan membuat ular terlihat sangat pendek. Bagian tubuh ular dibuat pendek untuk
- 9 memudahkan pengecekan jika terjadi ular menabrak tubuhnya sendiri. Setiap bagian tubuh ular
- 10 memiliki koordinat masing-masing. Koordinat setiap bagian tubuh disimpan pada sebuah *array* agar
- 11 menggambar ular menjadi lebih mudah. Dalam tahap ini, tubuh ular masih berupa sekumpulan
- 12 titik-titik yang merupakan koordinat bagian tubuh ular seperti pada Gambar 3.7. Algoritma untuk
- 13 menggambar ular adalah dengan mengambil koordinat bagian tubuh ular mulai dari elemen *array*
- 14 paling pertama(*arr[0]*) dan elemen *array* selanjutnya(*arr[1]*) lalu buat garis yang *start pointnya*
- 15 adalah elemen pertama(*arr[0]*) dan *end pointnya* adalah elemen *array* kedua(*arr[1]*). Setelah itu
- 16 ambil koordinat elemen *array* yang merupakan *end point* pada garis sebelumnya(*arr[1]*) dengan
- 17 elemen *array* selanjutnya(*arr[2]*) dan gambar garisnya. Lakukan hal tersebut sampai *end point* garis
- 18 mencapai elemen *array* paling akhir. Setelah digambar maka ular akan terlihat seperti Gambar 3.8.



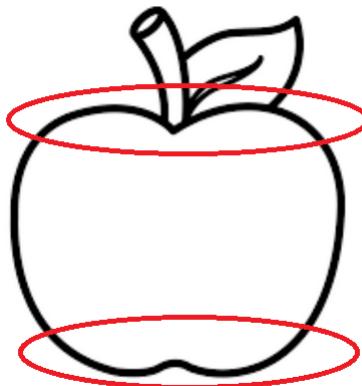
Gambar 3.7: Koordinat bagian tubuh ular pada *array*



Gambar 3.8: Tubuh ular setelah digambar menggunakan garis

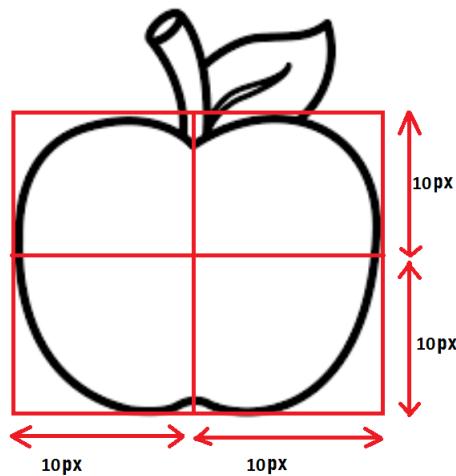
- 19 Untuk membuat apel digunakan *quadratic Bézier curve*. Kurva ini digunakan untuk membuat
- 20 bagian-bagian apel yang melengkung. Bagian tersebut ditandai dengan lingkaran bewarna merah
- 21 seperti yang ditunjukkan pada Gambar 3.9(gambar diambil dari pinterest. Link:<https://www.>

¹ pinterest.com/pin/690317449105509454/.



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

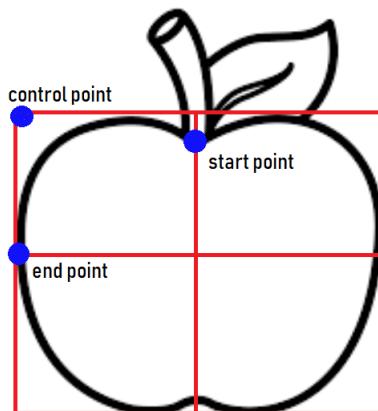
² Pertama, tentukan besar apel yang ingin dibuat. Dalam permainan ini besar apel yang dibuat
³ adalah 20 *pixel*. Besar apel dibuat lebih besar dari lebar ular karena jika besar apel sama dengan
⁴ lebar ular, besar apel terlihat kecil. Selain itu, apel ini digambar pada *layout* yang berbentuk
⁵ persegi. *Layout* persegi ini juga dapat mempermudah penggambaran apel. Karena menggunakan
⁶ *layout* persegi, maka *origin* terletak pada titik sudut di sebelah kiri atas. Setelah itu, gambar setiap
⁷ bagian apel. Bagian apel dibagi menjadi 4 seperti pada Gambar 3.10 sehingga besar setiap bagian
⁸ apel tersebut adalah 10 *pixel*.



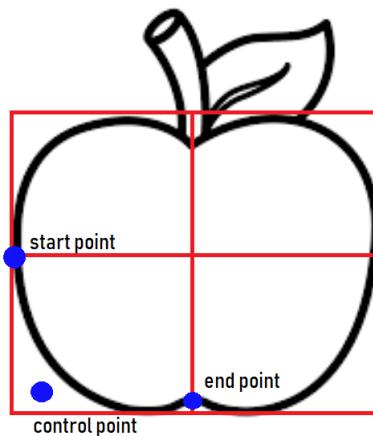
Gambar 3.10: Pembagian gambar apel dengan layout persegi beserta ukuran pada setiap bagian

⁹ Gambar bagian atas apel terlebih dahulu. Gunakan *method moveTo()* untuk menentukan titik
¹⁰ mulainya. Titik mulainya terletak pada bagian tengah atas apel yang melengkung ke dalam. Dari
¹¹ titik itu, buat kurva yang *control pointnya* adalah titik ujung *layout* persegi. Jika ingin menggambar
¹² bagian kiri apel terlebih dahulu maka *control pointnya* adalah titik ujung kiri *layout* tersebut.
¹³ Setelah itu, tentukan *end point* kurva tersebut. Pada Gambar 3.11 terdapat *start point*, *control point*
¹⁴ dan *end point* untuk membuat bagian sisi kiri atas apel. Sesudah itu, buatlah bagian bawah apel.
¹⁵ Caranya sama seperti sebelumnya namun *control pointnya* dan *end pointnya* berbeda. Posisi *control*

- 1 *pointnya sedikit menjorok ke dalam dan posisi end pointnya terdapat di tengah bawah seperti*
 2 *pada Gambar 3.12. Start point tidak perlu diatur lagi, karena start pointnya sudah tergantikan*
 3 *dengan posisi end point pada kurva sebelumnya. Sampai pada bagian ini, bagian kiri apel sudah*
 4 *selesai dibuat. Untuk membuat bagian kanan apel, caranya sama seperti membuat bagian kiri apel.*
 5 *Karena bagian kiri apel simetris dengan bagian kanan apel, maka hanya perlu mengubah control*
 6 *point dan end pointnya saja. Dengan memanfaatkan bentuk simetris dari apel, maka jarak antara*
 7 *control point dan end point pada bagian kiri apel dengan batasan tengah sama dengan jarak antara*
 8 *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

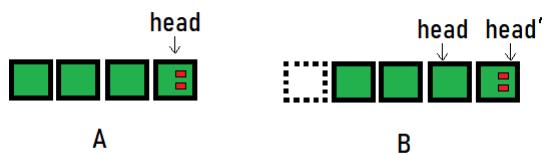


Gambar 3.12: *Start point, control point* dan *end point* untuk menggambar apel bagian kiri bawah

9 3.2.2 Pergerakan Ular

- 10 Untuk membuat ular bergerak maju, dilakukan penambahan kepala dan pembuangan ekor secara
 11 bersamaan ketika ular sedang bergerak maju. Ilustrasinya dapat dilihat pada Gambar 3.13. Untuk
 12 membuat ular bergerak sesuai ilustrasi pada Gambar 3.13, algoritmanya adalah sebagai berikut :
 13 Pertama, semua elemen *array* akan *dishift/digeser* dan elemen pertama akan digantikan dengan
 14 koordinat yang baru. Koordinat yang baru tersebut merupakan kepala ular. Setelah itu dilakukan
 15 pengecekan apakah panjang tubuh ular lebih besar dari jumlah elemen *array* tubuh ular. Jika benar,

- 1 maka tidak dilakukan pembuangan elemen terakhir dan jika salah, maka tidak akan dilakukan
- 2 apa-apa.



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

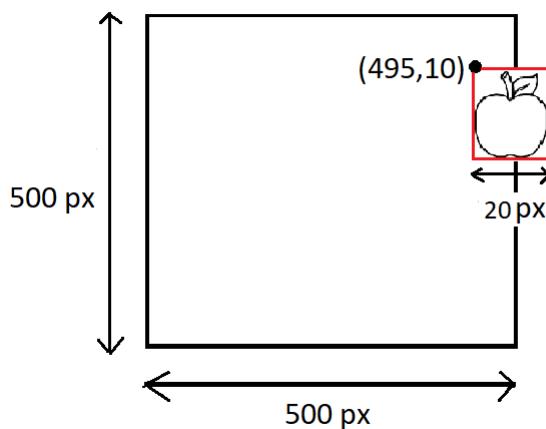
Kecepatan ular pada permainan ini adalah 2,4,6,8 dan 10 *pixel per frame*. Kecepatan ular minimal adalah 1 dan maksimum adalah 5. Kecepatan maksimal ular tidak boleh melebihi lebar tubuh ular. Jika kecepatanya melebihi lebar ular, maka ketika terjadi tabrakan dengan tubuhnya sendiri, kepala ular tidak 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 kecepatanya 3, maka ular akan maju sebesar 2 *pixel* 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 *cosinus* sedangkan untuk menghitung posisi koordinat y pada sudut tertentu menggunakan *sinus*. Jadi, koordinat x dari kepala ular akan ditambahkan dengan hasil perhitungan cosinus dari sudut dan koordinat y dari kepala ular akan ditambahkan dengan hasil perhitungan sinus dari sudut.

3.2.3 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()*

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



Gambar 3.14: Gambar apel yang terpotong sesudah mengacak posisi apel

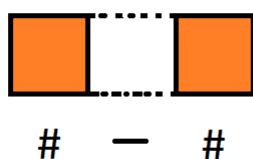
13 3.2.4 Menentukan Besar *Canvas*

14 Pada 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
 15 *canvas* yang cocok bila permainan tersebut dapat dimainkan pada *smartphone* dan *browser* pada
 16 *desktop*. Jika disesuaikan dengan layar *desktop*, maka besar *canvas* akan terlihat lebih lebar diban-
 17 dingkan dengan besar layar pada *smartphone* dan sebaliknya. Cara ini dapat ditangani dengan
 18 membuat *canvas* mengikuti besar layar *browser desktop* dan layar *browser smartphone*. Namun,
 19 cara ini juga dapat menimbulkan masalah yaitu dalam pembuatan labirin. Format labirin akan
 20 terus diubah sesuai dengan besar layar. Untuk menyesuaikan *canvas* dengan besar layar, maka akan
 21 dibuat *canvas* berbentuk persegi dengan dimensi 600×600 pixel. Dimensi ini sesuai jika permainan
 22 ini dimainkan pada *smartphone* dan *desktop*. Selain menentukan dimensi, besar objek yang ada
 23 pada *canvas* juga harus diperbesar agar objek-objek pada *canvas* tidak terlihat kecil bagi pemain
 24 bermain menggunakan *smartphone*.

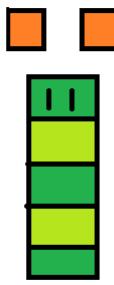
3.2.5 Menggambar Labirin

Pada permainan ini, format labirin dapat dibuat dengan menggunakan JSON dan *file* teks. Permainan ini menggunakan *file* teks sebagai format labirin, karena *file* text lebih mudah untuk dibuat dan dimengerti oleh pembuat labirin. Labirin dibuat menggunakan simbol '#' dan '-'. Setiap simbol merepresentasikan besar dinding. Daerah yang merupakan dinding labirin ditulis dengan menggunakan simbol '#' sedangkan untuk daerah yang bukan merupakan dinding labirin ditulis dengan menggunakan simbol '-'. Jika pada *file* tersebut terdapat teks '#-#', 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* teks akan disamakan dengan jumlah kolomnya. Sesuai dengan besar *canvas* dan besar dinding labirin yaitu 600×600 pixel, maka dapat ditentukan bahwa setiap *file* teks 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* labirin, *Javascript* tidak dapat digunakan karena *Javascript* tidak dapat membaca *file* dari server. Oleh karena itu, *AJAX* akan digunakan untuk membaca isi labirin dari *folder*. Dengan menggunakan *AJAX*, terdapat sebuah masalah yaitu *AJAX* bersifat *asynchronous* yang menyebabkan labirin belum selesai digambar sebelum permainan sudah siap untuk dimainkan. Cara untuk menangani masalah ini adalah dengan menggunakan *callback*. Dengan adanya *callback*, dapat dipastikan bahwa permainan sudah siap untuk dimainkan apabila labirin sudah selesai digambar.



Gambar 3.15: Menggambar dinding menggunakan simbol pada file teks

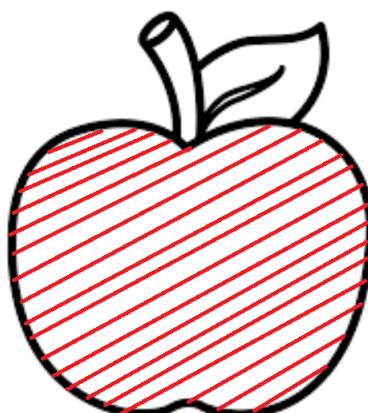


Gambar 3.16: Ular ingin melewati jalur yang diapit oleh 2 buah dinding

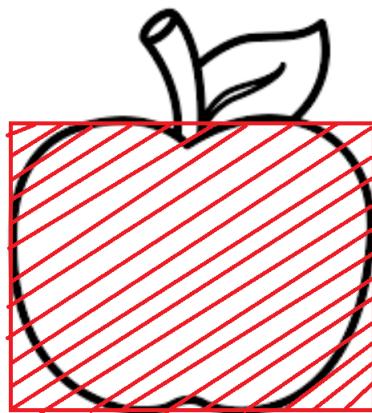
3.2.6 Pengecekan tabrakan(*Collision Detection*)

Pada permainan ini terdapat pengecekan tabrakan yang dapat mengecek apakah ular sudah memakan makanan, ular menabrak tubuhnya sendiri, dan ular menabrak dinding labirin. Seluruh pengecekan ini akan dilakukan pada setiap *frame*. Pada pengecekan tabrakan pada apel dan ular, hanya perlu mengecek tabrakan antara kepala ular dengan apel. Karena jalur yang dilalui oleh kepala ular, akan selalu dilalui oleh bagian tubuh ular. Dengan kata lain, bagian tubuh ular akan mengikuti ke mana kepala ular akan bergerak. Dengan ini, tidak perlu dilakukan *collision detection* antara bagian tubuh ular dengan apel. Untuk mengetahui terjadinya tabrakan antara ular dengan apel, maka akan dibuat daerah tabrakan pada apel. Daerah tabrakan ini digunakan untuk mengecek apakah 2 benda saling bertabrakan satu sama lain. Daerah tabrakan pada apel ditandai dengan arsiran bewarna merah yang terdapat pada Gambar 3.17. Namun, untuk membuat daerah tabrakan ini cukup sulit ketika mengecek adanya tabrakan antara ular dengan apel terutama pada bagian lengkungan pada apel. Karena itu, daerah tabrakan pada apel dibuat dengan menggunakan bentuk persegi seperti pada Gambar 3.18. Jika posisi kepala ular berada di dalam daerah tabrakan apel, maka dapat dipastikan bahwa ular tersebut sudah memakan apel.

16



Gambar 3.17: Daerah tabrakan pada apel

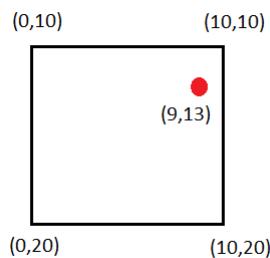


Gambar 3.18: Daerah tabrakan berbentuk persegi pada apel

Untuk mengecek tabrakan antara ular dengan tubuhnya sendiri adalah dengan mengecek tabrakan antara kepala ular dengan seluruh bagian tubuh ular. Ular dapat dipastikan menabrak tubuhnya sendiri apabila ketentuan berikut telah terpenuhi :

- koordinat x kepala ular lebih kecil dari koordinat x bagian tubuh ular dikurangi panjang bagian tubuh ular
- koordinat x kepala ular lebih besar dari koordinat x bagian tubuh ular ditambah dengan panjang bagian tubuh ular
- koordinat y kepala ular lebih kecil dari koordinat y bagian tubuh ular dikurangi panjang bagian tubuh ular
- koordinat y kepala ular lebih besar dari koordinat y bagian tubuh ular ditambah dengan panjang bagian tubuh ular

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

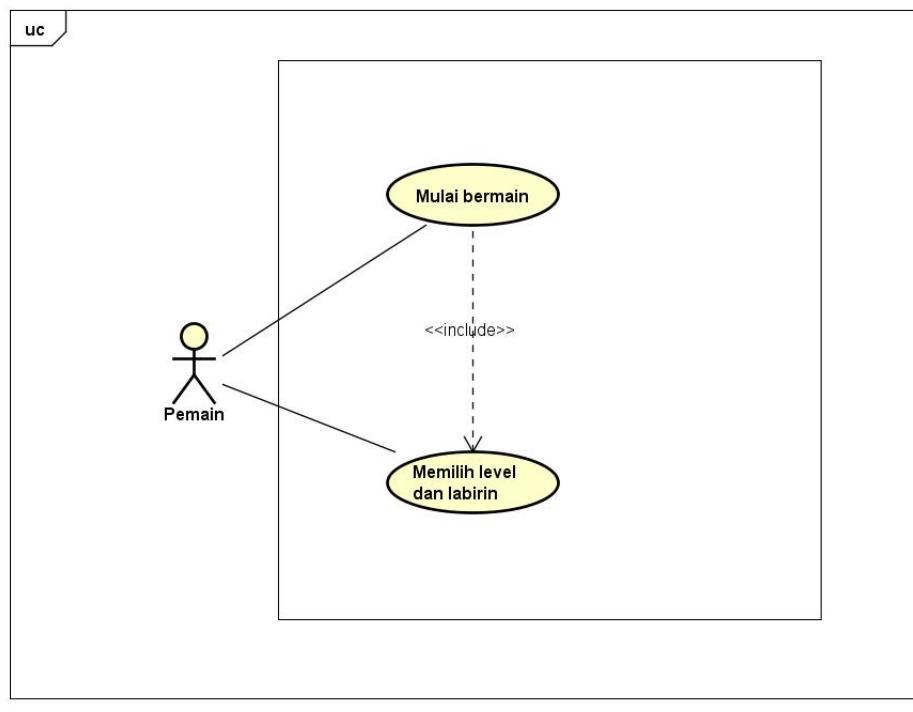


Gambar 3.19: Posisi kepala ular pada sebuah daerah labirin

3.3 Analisis Berorientasi Objek

3.3.1 Skenario Permainan

Pada bagian ini akan dijelaskan dan ditunjukkan diagram *use case* dari permainan *Snake 360*. Penjelasan meliputi skenario, aktor, prakondisi skenario normal dan eksepsi. Aktor yang melakukannya adalah pemain. Pada Gambar 3.20 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

1 dan labirin.

2 Eksepsi : -

4 2. Skenario : Memilih level labirin dan kecepatan ular

5 Aktor : Pemain

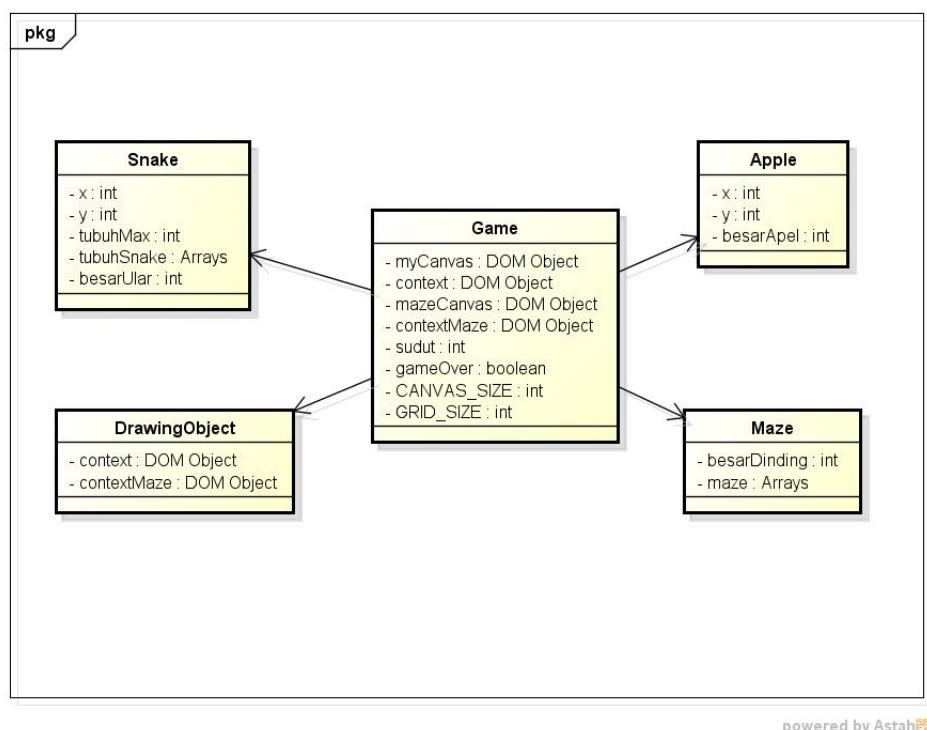
6 Prakondisi : Pemain sudah mulai bermain.

7 Skenario normal : Pemain memilih level dan labirin yang diinginkan.

8 Eksepsi : -

10 3.3.2 Diagram Kelas

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



Gambar 3.21: Diagram kelas dari permainan *Snake 360*

12 Diagram kelas terdiri dari beberapa kelas yaitu :

13 1. Kelas Snake merupakan kelas yang merepresentasikan objek ular.

14 2. Kelas Apple merupakan kelas yang merepresentasikan objek apel.

15 3. Kelas Game merupakan kelas yang mengatur jalanya permainan.

16 4. Kelas Maze merupakan kelas yang merepresentasikan objek labirin.

17 5. Kelas DrawingObject merupakan kelas untuk menggambar semua objek pada canvas.

18 Berikut adalah atribut yang dimiliki setiap kelas :

1 1. Kelas Snake

2 **int**

- 3 • x, merupakan posisi ular pada koordinat x.
4 • y, merupakan posisi ular pada koordinat y.
5 • tubuhMax, merupakan panjang tubuh ular.
6 • besarUlar, merupakan lebar tubuh ular.

7 **Array**

- 8 • tubuhSnake, merupakan posisi tubuh ular pada koordinat x dan y.

9 2. Kelas Apel

10 **int**

- 11 • x, merupakan posisi apel pada koordinat x.
12 • y, merupakan posisi apel pada koordinat y.
13 • besarApel, merupakan besar apel.

14 3. Kelas Game

15 **int**

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

20 **DOM Object**

- 21 • myCanvas, merupakan objek *canvas* untuk menggambar ular dan apel.
22 • context, merupakan *context* 2D pada *myCanvas*.
23 • mazeCanvas, merupakan objek canvas untuk menggambar labirin.
24 • contextMaze, merupakan *context* 2D pada *mazeCanvas*.

25 **boolean**

- 26 • gameOver, memberitahu apakah permainan sudah berakhir atau belum.

27 4. Kelas Maze

28 **int**

- 29 • besarDinding, merupakan besar lebar dinding.

30 **Array**

- 1 ● maze, merupakan layout labirin.

2 5. DrawingObject

3 **DOM Object**

- 4 ● context, merupakan objek canvas untuk menggambar ular dan apel.

- 5 ● contextMaze, merupakan objek canvas untuk menggambar labirin.

1

BAB 4

2

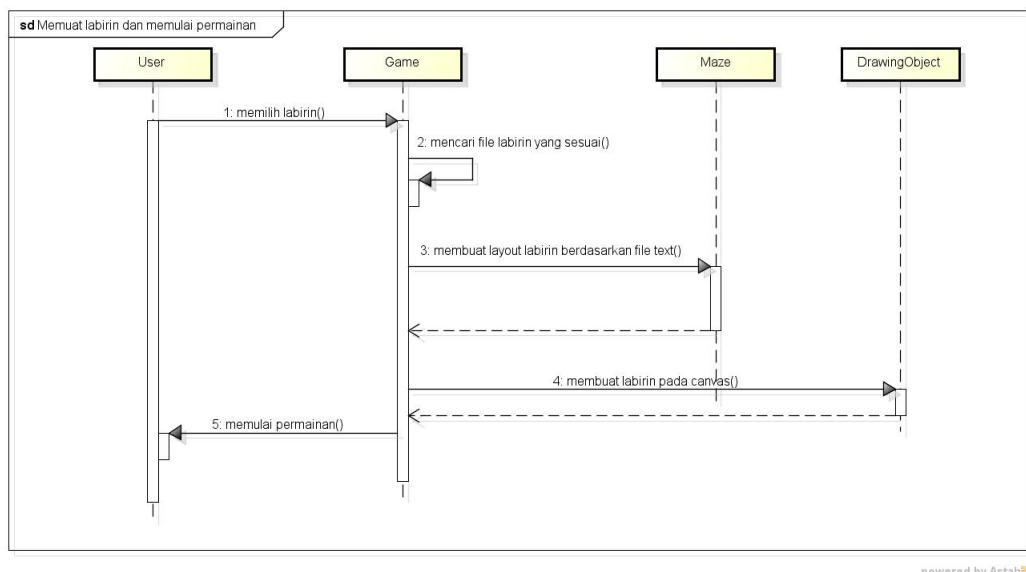
PERANCANGAN

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

6 4.1 Rancangan Diagram Sequence

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

9 4.1.1 Memuat Labirin



Gambar 4.1: Diagram *sequence* untuk memuat labirin

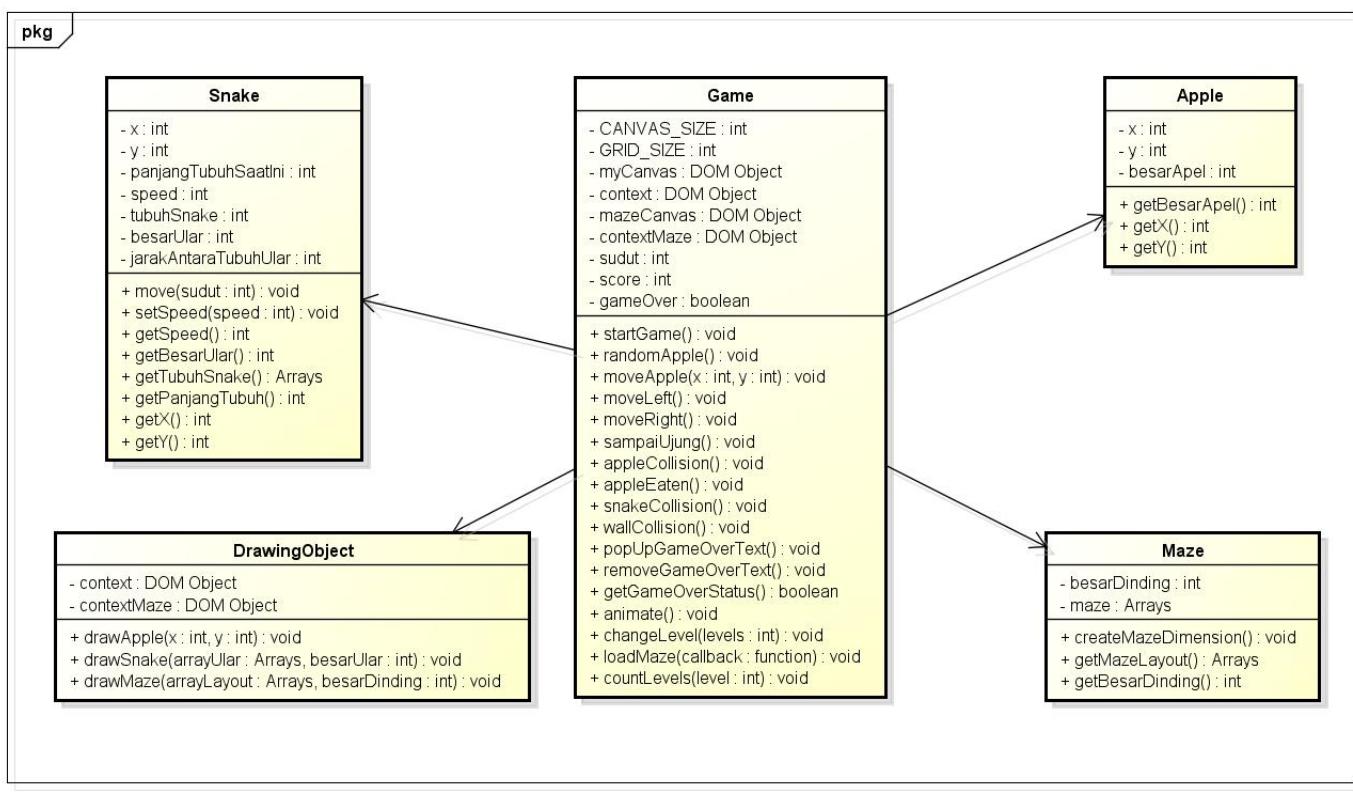
- 10 Pada Gambar 4.1, pemain memulai bermain dengan memilih labirin. Berikut adalah penjelasan
11 dari Gambar 4.1:

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

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

6 4.2 Rancangan Diagram Kelas Rinci

- 7 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*

9 Deskripsi Kelas dan Method

- 10 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.

14 1. Kelas *Game*

- 15 Kelas *Game* merupakan kelas utama dari permainan ini. Kelas ini mengatur jalannya permainan.
- 16

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

- 1 Output : tidak ada
2
3 • Nama *method* : *snakeCollision*
4 Deskripsi : mengecek tabrakan antara kepala ular dengan tubuhnya sendiri
5 Input : tidak ada
6 Output : tidak ada
7
8 • Nama *method* : *wallCollision*
9 Deskripsi : mengecek tabrakan antara kepala ular dengan dinding labirin
10 Input : tidak ada
11 Output : tidak ada
12
13 • Nama *method* : *popUpGameOverText*
14 Deskripsi : memunculkan tulisan 'Game Over'
15 Input : tidak ada
16 Output : tidak ada
17
18 • Nama *method* : *removeGameOverText*
19 Deskripsi : menghilangkan tulisan 'Game Over'
20 Input : tidak ada
21 Output : tidak ada
22
23 • Nama *method* : *getGameOverStatus*
24 Deskripsi : mendapatkan status gameOver
25 Input : tidak ada
26 Output : boolean *gameOver*
27 – *gameOver* : status apabila permainan sudah berakhir atau belum
28 • Nama *method* : *animate*
29 Deskripsi : membuat animasi dari setiap objek
30 Input : tidak ada
31 Output : tidak ada
32 • Nama *method* : *changeLevel*
33 Deskripsi : mengubah jumlah level pada menu utama
34 Input : int *levels*
35 – *levels* : jumlah *level* pada server
36 Output : tidak ada
37 • Nama *method* : *loadMaze*
38 Deskripsi : mengambil dan memenuhi labirin yang dipilih pemain
39 Input : function *callback*
40 – *callback* : fungsi yang akan dijalankan setelah *method loadMaze* selesai mengeksekusi

1 Output : tidak ada
2 ● Nama *method* : *countLevels*
3 Deskripsi : menghitung jumlah *file* labirin pada server
4 Input : *int level*
5 – *level* : *file* labirin yang ingin dicek keberadaanya pada server
6 Output : tidak ada
7

8 2. Kelas *Snake*
9 Kelas *Snake* merupakan kelas yang merepresentasikan objek ular.

10 ● Nama *method* : *move*
11 Deskripsi : memulai permainan
12 Input : *int x, int y*
13 – *x* : koordinat x milik ular
14 – *y* : koordinat y milik ular
15 Output : tidak ada
16
17 ● Nama *method* : *setSpeed*
18 Deskripsi : mengubah nilai atribut *speed*
19 Input : *int speed*
20 – *speed* : kecepatan laju ular
21 Output : tidak ada
22 ● Nama *method* : *getSpeed*
23 Deskripsi : mendapatkan nilai atribut *speed*
24 Input : tidak ada
25 Output : *int speed*
26 – *speed* : kecepatan laju ular
27 ● Nama *method* : *getBesarUlar*
28 Deskripsi : mendapatkan nilai atribut *besarUlar*
29 Input : tidak ada
30 Output : *int speed*
31 – *besarUlar* : lebar tubuh ular

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

35 ● Nama *method* : *drawApple*
36 Deskripsi : menggambar objek apel
37 Input : *int x, int y*

- 1 – x : koordinat x milik apel
- 2 – y : koordinat y milik apel
- 3 Output : tidak ada
- 4
- 5 • Nama *method* : *drawSnake*
- 6 Deskripsi : menggambar objek ular
- 7 Input : *int[] arrayUlar, int besarUlar*
 - 8 – arrayUlar : koordinat x dan y milik setiap bagian tubuh ular
 - 9 – besarUlar : lebar tubuh ular
- 10 Output : tidak ada
- 11
- 12 • Nama *method* : *drawMaze*
- 13 Deskripsi : menggambar labirin
- 14 Input : *String arrayLayout, int besarDinding*
 - 15 – arrayLayout : layout labirin yang akan digambar
 - 16 – besarDinding : besar dinding labirin
- 17 Output : tidak ada
- 18

19 4. Kelas *Apple*

20 Kelas *Apple* adalah kelas yang merepresentasikan objek apel.

- 21 • Nama *method* : *getBesarApel*
- 22 Deskripsi : mendapatkan nilai atribut besarApel
- 23 Input : tidak ada
- 24 Output : *int besarApel*
 - 25 – besarApel : besar apel

26 5. Kelas *Maze*

27 Kelas *Maze* adalah kelas yang merepresentasikan labirin.

- 28 • Nama *method* : *setMazeLayout*
- 29 Deskripsi : membuat labirin berdasarkan file teks yang dipilih
- 30 Input : *text layoutInText*
 - 31 – layoutInText : isi teks file labirin
- 32 Output : tidak ada
- 33
- 34 • Nama *method* : *getMazeLayout*
- 35 Deskripsi : mendapatkan labirin
- 36 Input : tidak ada
- 37 Output : *Arrays maze*

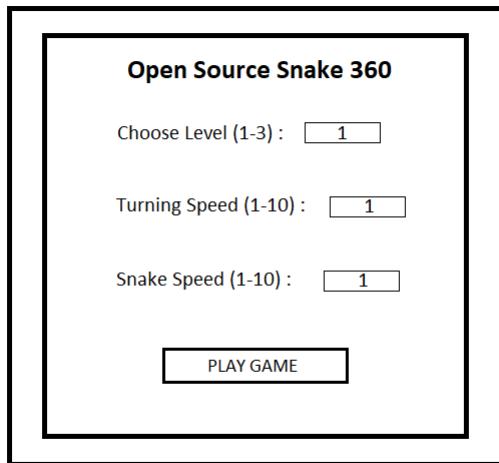
- 1 – maze : labirin
- 2 • Nama *method* : *getBesarDinding*
3 Deskripsi : mendapatkan besar dinding labirin
4 Input : tidak ada
5 Output : *int* besarDinding
6 – besarDinding : besar dinding labirin

7 4.3 Rancangan Tampilan Antarmuka

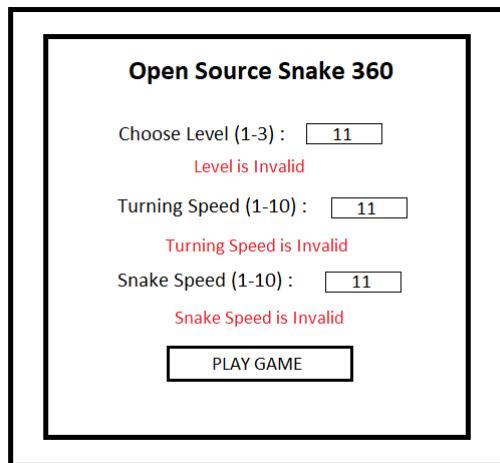
- 8 Pada bagian ini akan ditunjukan rancangan tampilan antarmuka dari permainan yang dibangun
- 9 yang terdiri dari menu pemilihan level labirin, mulai bermain, dan permainan berakhir.

10 4.3.1 Tampilan Menu Utama

- 11 Gambar 4.3 merupakan rancangan tampilan awal dari permainan yang dibangun. Pada tampilan ini
- 12 terdapat judul permainan, 3 buah input untuk memilih level labirin, memilih kecepatan gerak ular,
- 13 dan memilih kecepatan ular berbelok dan sebuah tombol '*Play Game*' untuk memulai permainan.
- 14 Tampilan menu utama akan menampilkan pesan kesalahan seperti terdapat pada Gambar 4.4.
- 15 Apabila pemain salah memasukkan salah satu data, maka permainan tidak akan dimulai jika tombol
- 16 '*Play Game*' ditekan.



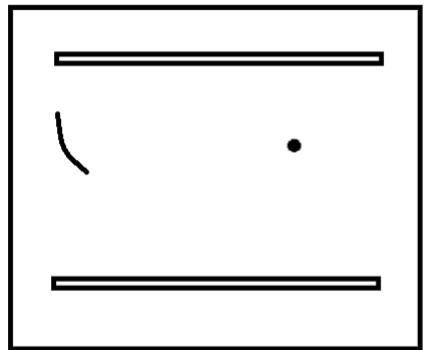
Gambar 4.3: Rancangan tampilan menu utama



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

4.3.2 Tampilan Bermain

- 2 Tampilan bermain akan muncul setelah pemain memilih level labirin, kecepatan ular dan kecepatan 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 labirin dan makanan berbentuk apel. Pada tampilan ini juga terdapat level labirin yang dipilih dan skor yang didapat.



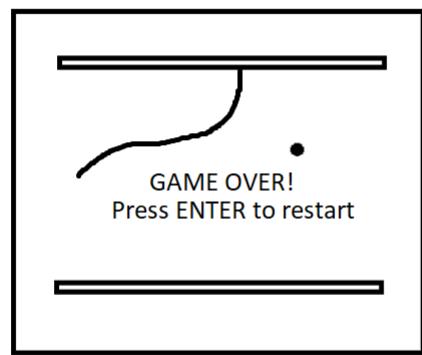
Level 1

Score : 2

Gambar 4.5: Rancangan tampilan bermain

4.3.3 Tampilan Permainan Berakhir

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



Level 1

Score : 15

Gambar 4.6: Rancangan tampilan permainan berakhir

1

BAB 5

2

IMPLEMENTASI DAN PENGUJIAN

- 3 Pada bab ini akan dibahas mengenai hasil implementasi dan pengujian dari Open Source Snake 360.
4 Permainan ini dapat dimainkan pada *link* ini : <https://generaldevilx.github.io/Snake360/>

5 **5.1 Implementasi**

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

8 **5.1.1 Lingkungan Perangkat Keras**

- 9 Berikut adalah lingkungan perangkat keras yang digunakan dalam pembangunan permainan ini:
- 10 1. Perangkat : Laptop
11 2. *Processor* : Intel Core i5-7200U 2.5GHz
12 3. RAM : 4.00 GB
13 4. *Video Card* : GeForce 930MX
14 5. Monitor : 14"
15 6. *Storage* : 1TB

16 Pada pengujian digunakan 1 buah perangkat *mobile* berbasis *Android* dan 1 buah perangkat
17 *desktop*. Berikut adalah lingkungan perangkat keras yang digunakan dalam pengujian permainan
18 ini:

19 **Perangkat 1**

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

1 6. *Storage* : 1TB

2 **Perangkat 2**

3 1. Perangkat : SM-J730G

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

5 3. RAM : 3.00 GB

6 4. *Video Card* : Mali-T830

7 5. Monitor : 5.5"

8 6. *Storage* : 32 GB

9 **5.1.2 Lingkungan Perangkat Lunak**

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

11 1. Sistem Operasi Laptop : Windows 10 64-bit

12 2. Bahasa Pemrograman : *Javascript*, HTML

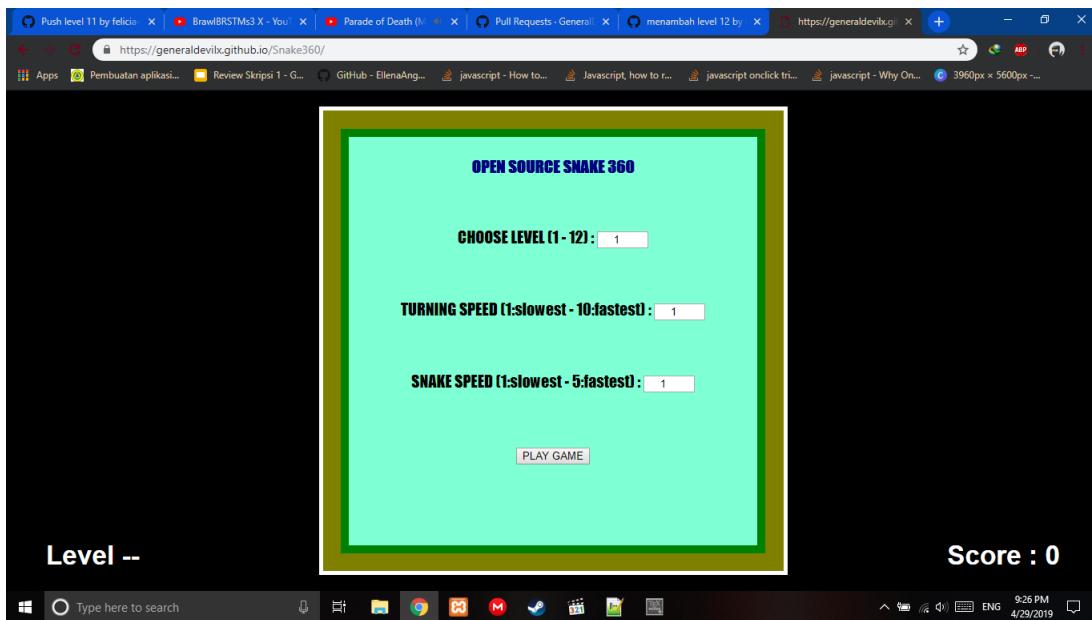
13 3. Sistem Operasi *Smartphone* : Android Nougat v7.0

14 **5.1.3 Implementasi Antarmuka**

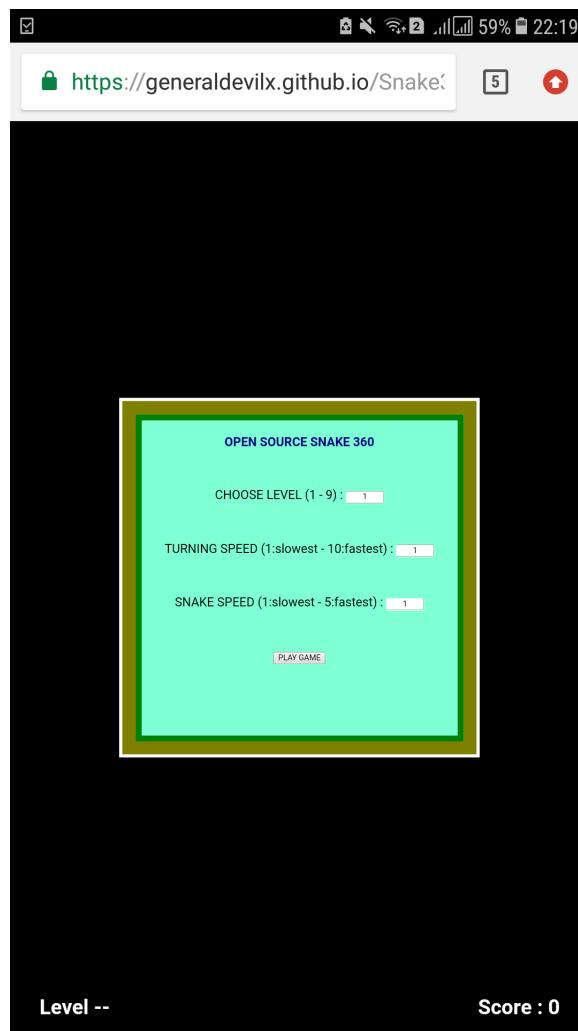
15 Pada subbab ini akan ditampilkan dan dijelaskan tampilan antarmuka dari Open Source Snake 360.

16 **Tampilan Menu Utama**

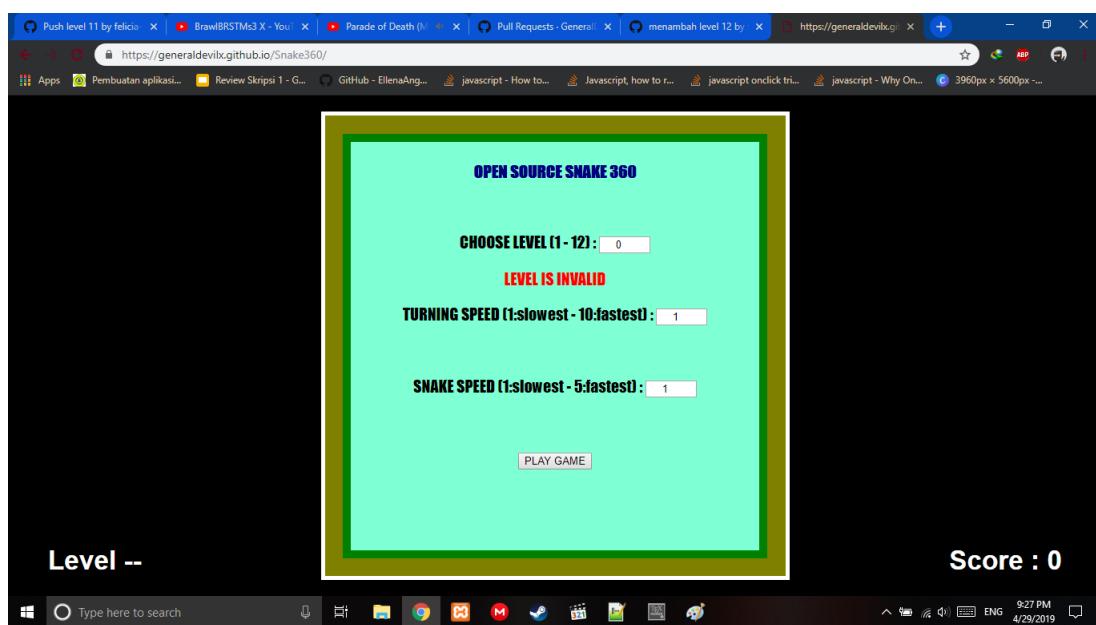
17 Gambar 5.1 dan Gambar 5.2 merupakan tampilan antarmuka menu utama pada *desktop* dan
18 *smartphone*. Pada tampilan ini terdapat judul dari permainan, *input* untuk mengisi *level* labirin,
19 kecepatan ular berbelok, kecepatan ular, dan tombol 'Play Game'. Jika pemain salah memasukkan
20 data, maka terdapat pesan kesalahan yang ditandai dengan tulisan bewarna merah. Misal, pada
21 Gambar 5.3 dan Gambar 5.4, pemain salah memasukkan data untuk level labirin sehingga muncul
22 pesan kesalahan bahwa *input* yang dimasukkan salah.



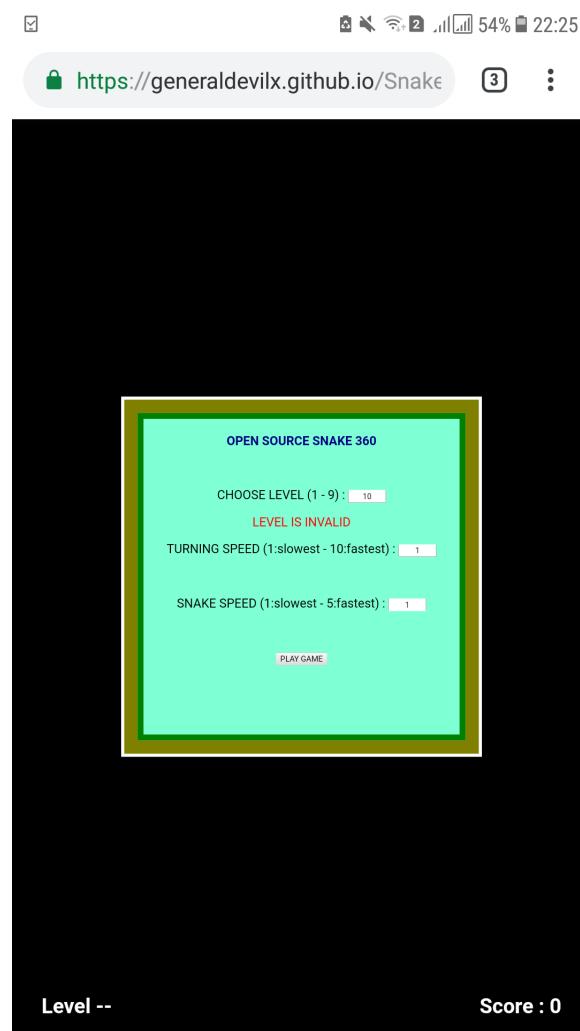
Gambar 5.1: Tampilan menu utama pada *desktop*



Gambar 5.2: Tampilan menu utama pada *smartphone*



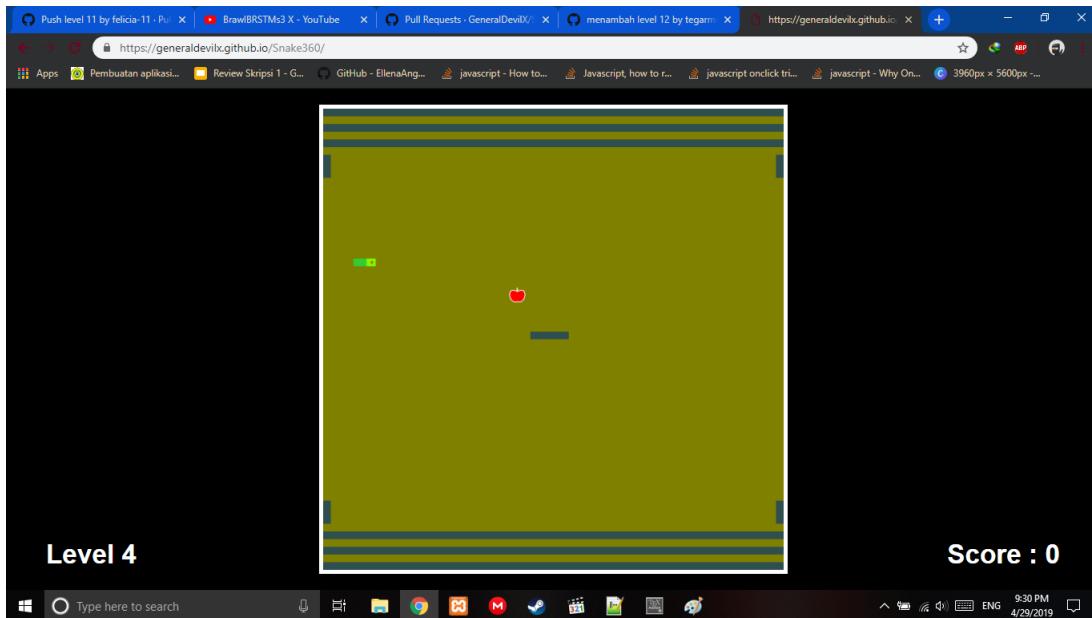
Gambar 5.3: Tampilan menu utama jika pemain salah memasukkan data *level* labirin pada *desktop*



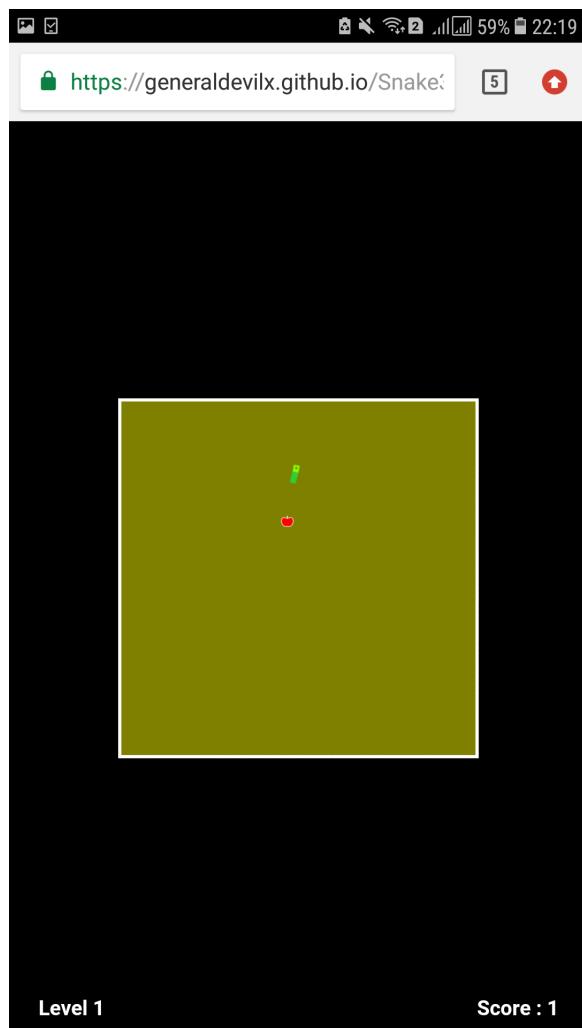
Gambar 5.4: Tampilan menu utama jika pemain salah memasukkan data *level* labirin pada *smartphone*

1 Tampilan Bermain

- 2 Gambar 5.5 dan Gambar 5.6 merupakan tampilan antarmuka mulai bermain pada *desktop* dan
3 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.



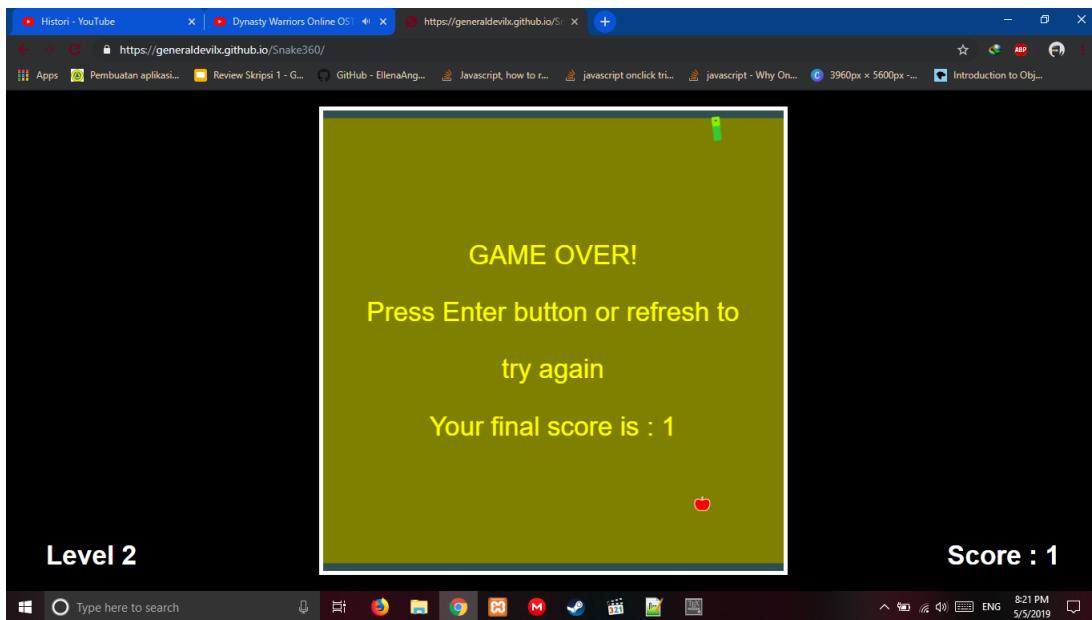
Gambar 5.5: Tampilan bermain pada *desktop*



Gambar 5.6: Tampilan bermain pada *smartphone*

¹ Tampilan Permainan Berakhir

- ² Gambar 5.7 dan Gambar 5.8 merupakan tampilan antarmuka jika permainan berakhir pada *desktop*
³ dan *smartphone*. Tampilan ini muncul apabila ular menabrak dinding labirin dan menabrak
⁴ tubuhnya sendiri. Pemain akan diarahkan ke tampilan utama jika pemain menekan tombol 'Enter'
⁵ pada tampilan ini. Karena pada smartphone tidak memiliki tombol 'Enter', maka pemain hanya
⁶ memuat ulang/refresh halaman tersebut.



Gambar 5.7: Tampilan permainan berakhir pada desktop



Gambar 5.8: Tampilan permainan berakhir smartphone

5.2 Pengujian

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

5.2.1 Pengujian Fungsional

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

- 8 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 "Play Game"	Pemain akan diarahkan ke tampilan bermain. Kondisi untuk dapat memulai permainan adalah data level labirin, kecepatan berbelok dan kecepatan laju ular sudah valid. Posisi ular dan dinding labirin sesuai dengan level labirin yang dipilih. Posisi apel tidak berada di atas dinding labirin atau tubuh ular.	Umumnya, posisi apel tidak berada tepat di atas dinding labirin. Pada beberapa kasus, posisi apel tepat berada di atas dinding labirin.

- 9 Berdasarkan tabel 5.1, dapat disimpulkan bahwa kasus uji pada tampilan menu utama belum
 10 membawakan hasil sesuai dengan yang diharapkan ketika pemain menekan tombol "Play
 11 Game". Hal ini disebabkan karena pengecekan tabrakan antara apel dengan dinding labirin
 12 kurang akurat.

- 13 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
Ular mencapai sisi ujung labirin	Ular akan muncul di sisi ujung labirin yang berlawanan	Hasil pengujian sesuai dengan yang diharapkan

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

3. Pengujian fungsionalitas tampilan bermain pada *smartphone*.

Tabel 5.3: Pengujian Fungsional Tampilan Bermain pada *Smartphone*

Kasus uji	Hasil yang diharapkan	Hasil uji
Bagian kiri layar ditekan	Ular akan bergerak melawan arah jarum jam	Ketika menekan layar lebih lama, akan muncul sebuah menu yang menginterupsi. Ular akan terus bergerak melawan arah jarum jam meskipun bagian kiri layar tidak ditekan sama sekali.
Bagian kanan layar ditekan	Ular akan bergerak searah jarum jam	Ketika menekan layar lebih lama, akan muncul sebuah menu yang menginterupsi. Ular akan terus bergerak searah arah jarum jam meskipun bagian kanan layar tidak ditekan sama sekali.
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
Ular mencapai sisi ujung labirin	Ular akan muncul di sisi ujung labirin yang berlawanan	Hasil pengujian sesuai dengan yang diharapkan

4 Berdasarkan tabel 5.3, dapat disimpulkan bahwa kasus uji tampilan bermain pada *smartphone* belum membawakan hasil sesuai dengan yang diharapkan. Hal ini dikarenakan *web browser*

1 pada smartphone akan otomatis memunculkan menu jika layar *smartphone* ditekan lebih
2 lama.

3 4. Pengujian fungsionalitas tampilan permainan berakhir

Tabel 5.4: Pengujian Fungsional Tampilan Permainan Berakhir

Kasus uji	Hasil yang diharapkan	Hasil uji
Tombol "Enter" ditekan (pada <i>desktop</i>) atau pemain memuat ulang halaman (pada <i>smartphone</i>)	Pemain akan diarahkan ke tampilan menu utama	Hasil pengujian sesuai dengan yang diharapkan

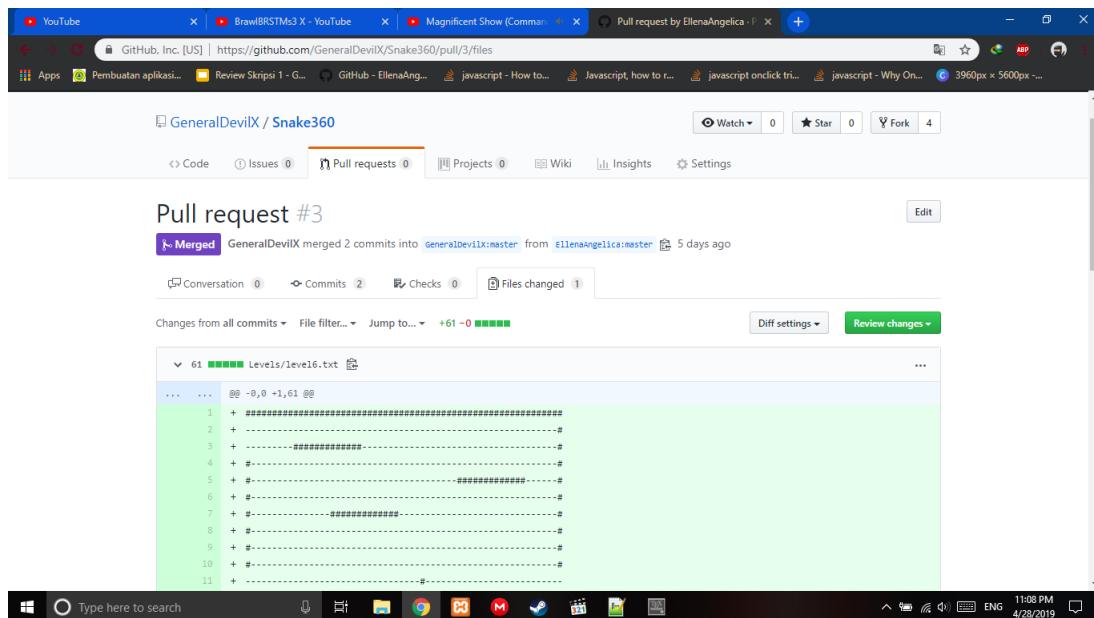
4 Berdasarkan tabel 5.4, dapat disimpulkan bahwa kasus uji pada tampilan "game over" mem-
5 bawakan hasil sesuai dengan yang diharapkan.

6 5.2.2 Pengujian Eksperimental

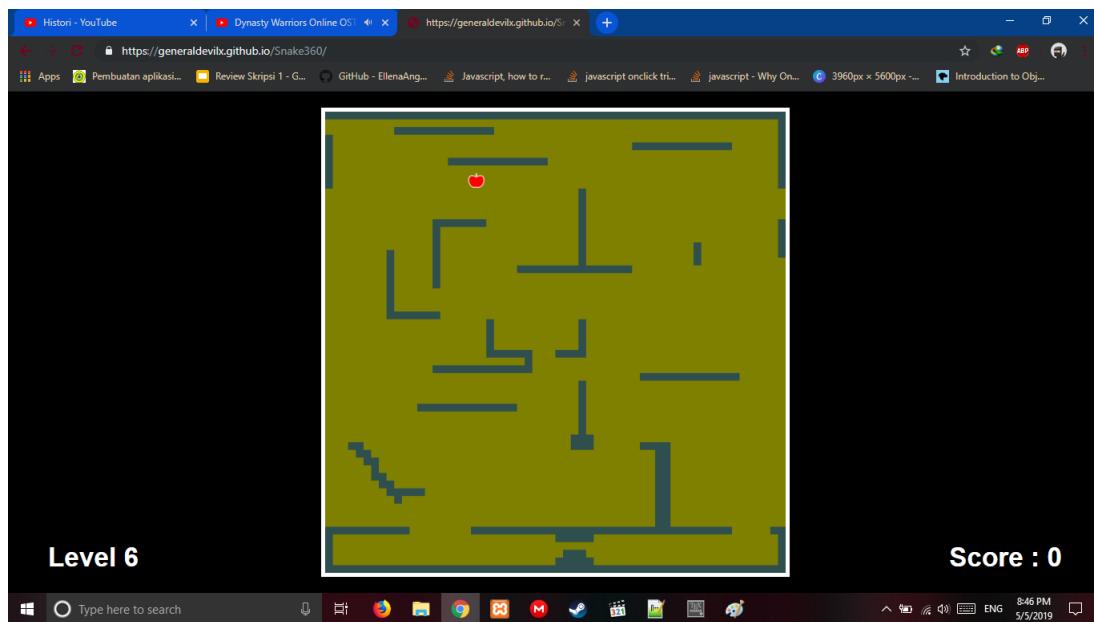
7 Pada pengujian eksperimental akan diuji penambahan labirin oleh orang lain. Pada pengujian
8 ini terdapat 5 orang yang akan menambahkan labirin menggunakan *pull request GitHub*. Setiap
9 penguji diminta untuk membaca file *readme* (lampiran B) yang berisikan tentang cara menambah
10 labirin. Berikut adalah hasil pengujian eksperimental :

11 1. Penguji 1 : EllenaAngelica

12 Penguji 1 berhasil menambahkan labirin *level 6* dengan menggunakan *pull request* seperti yang
13 terlihat pada Gambar 5.9. Link untuk file teks yang dibuat penguji 1 : <https://github.com/GeneralDevilX/Snake360/pull/3/files>. Labirin yang dibuat sudah sesuai dengan besar
14 *canvas*. Namun posisi ular yang dimasukkan tidak sesuai. Penguji 1 menambahkan sebuah
15 baris kosong di akhir file tersebut sehingga menyebabkan permainan menjadi error. Hal ini
16 disebabkan karena pembacaan posisi ular selalu mengambil teks baris paling terakhir. Karena
17 itu, penulis harus memperbaiki labirin yang dibuat oleh penguji 1. Gambar 5.10 merupakan
18 hasil pengujian menggunakan labirin yang dibuat oleh penguji 1.



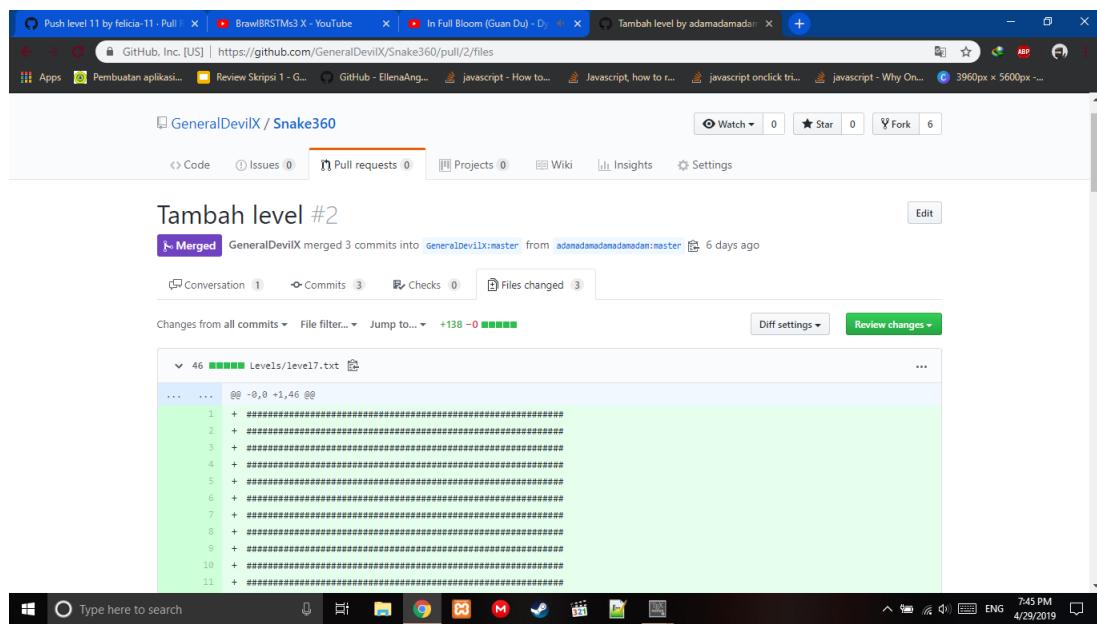
Gambar 5.9: Tampilan hasil pull request milik penguji 1



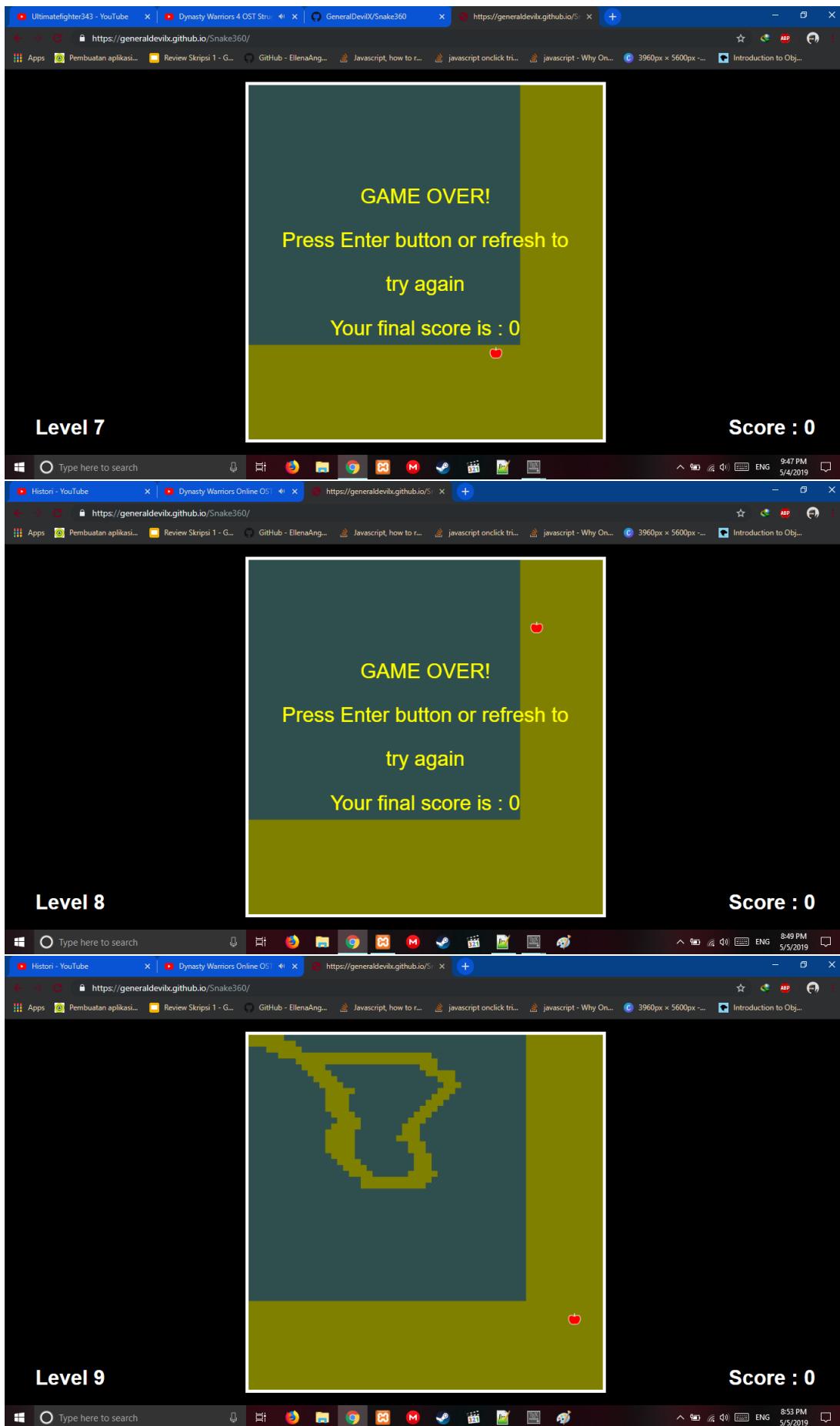
Gambar 5.10: Tampilan pengujian labirin yang dibuat oleh penguji 1

2. Penguin 2 : adamadamadamadamadam

Penguin 2 berhasil menambahkan labirin level 7, 8 dan 9 dengan menggunakan pull request seperti yang terlihat pada Gambar 5.11. Link untuk file teks yang dibuat penguin 2 : <https://github.com/GeneralDevilX/Snake360/pull/2/files>. Ketiga labirin yang dibuat oleh penguin 2 tidak sesuai dengan besar *canvas* sehingga tampilan menjadi seperti pada Gambar 5.12. Penguin 2 membuat labirin yang seluruhnya adalah dinding dan memposisikan ular di (0,0). Penulis ingin memberitahukan penguin 2 bahwa labirin yang dibuat belum benar, tetapi terjadi sebuah kesalahan yaitu penulis menekan tombol *merge request*. Hal ini membuat penulis harus memperbaiki labirin yang dibuat oleh penguin 2.



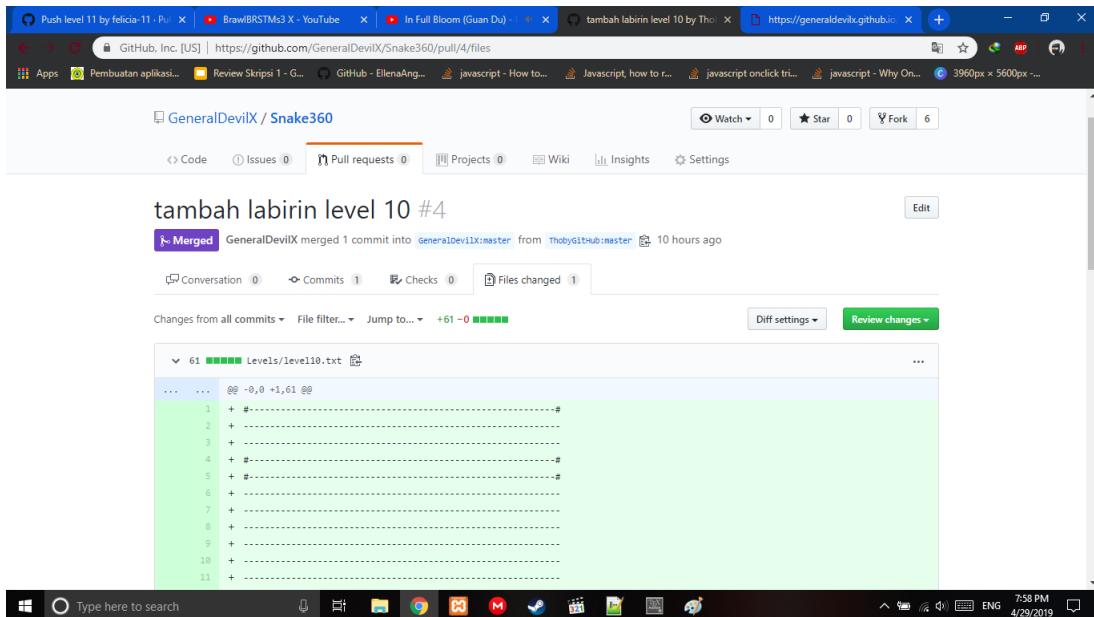
Gambar 5.11: Tampilan hasil pull request milik penguji 2



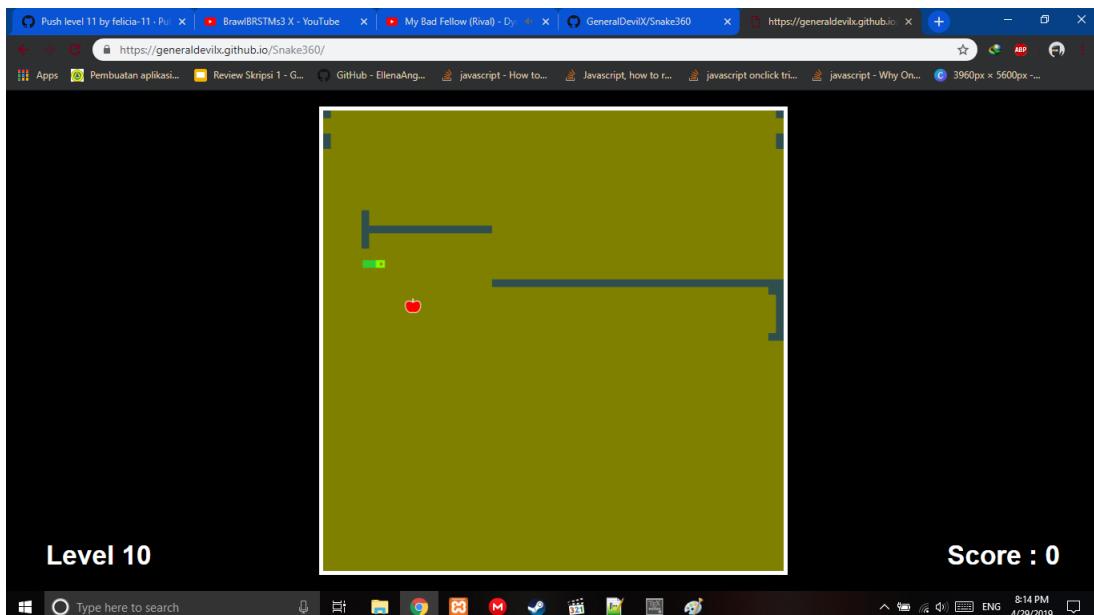
Gambar 5.12: Tampilan pengujian labirin yang dibuat oleh penguji 2

1 3. Penguji 3 : ThobyGitHub

2 Penguji 3 berhasil menambahkan labirin level 10 dengan menggunakan pull request seperti
 3 yang terlihat pada Gambar 5.13. Link untuk file teks yang dibuat penguji 3 : <https://github.com/GeneralDevilX/Snake360/pull/4/files>. Labirin yang dibuat oleh penguji
 4 3 sudah sesuai ketentuan membuat labirin pada file readme sehingga tampilan menjadi seperti
 5 pada Gambar 5.14.



Gambar 5.13: Tampilan hasil pull request milik penguji 3

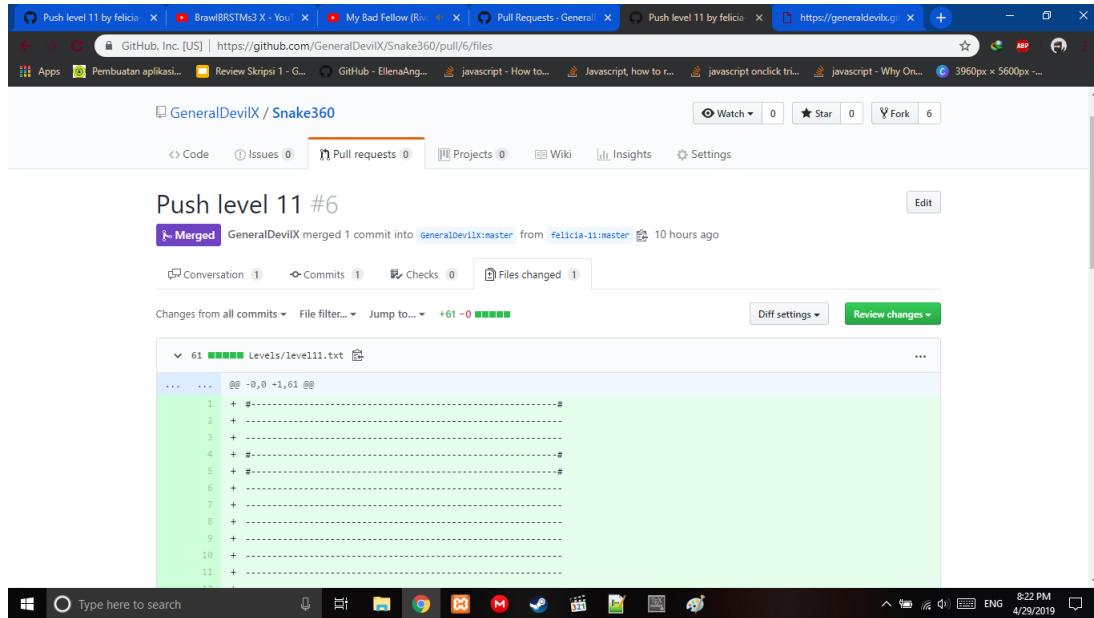


Gambar 5.14: Tampilan pengujian labirin yang dibuat oleh penguji 3

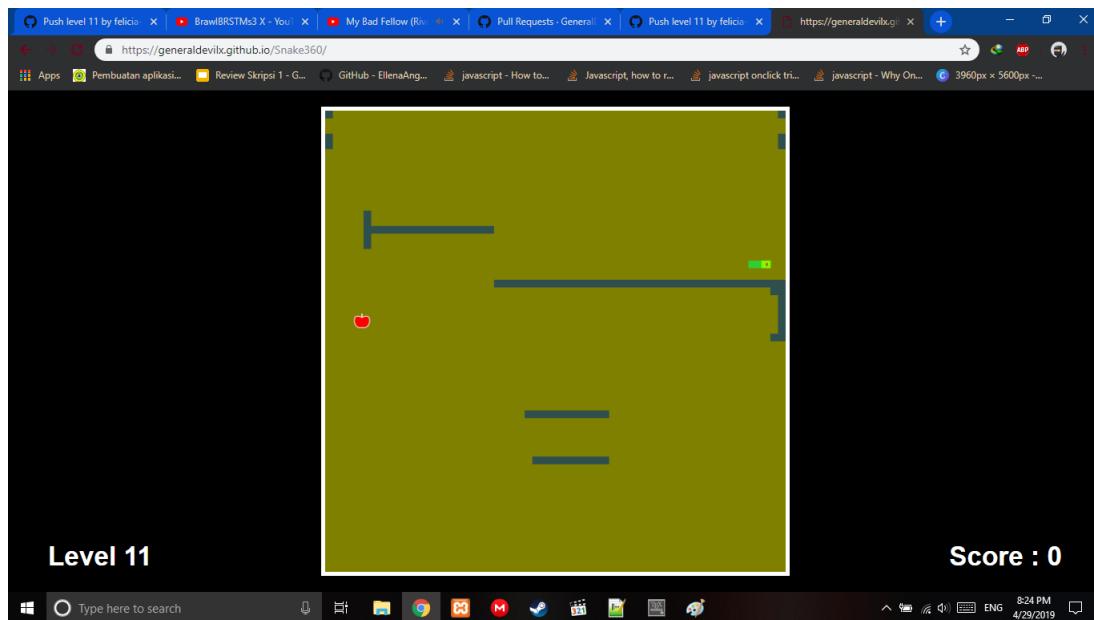
1 4. Penguji 4 : felicia-11

2 Penguji 4 berhasil menambahkan labirin level 11 dengan menggunakan pull request seperti

yang terlihat pada Gambar 5.15. Link untuk file teks yang dibuat penguji 4 : <https://github.com/GeneralDevilX/Snake360/pull/6/files>. Labirin yang dibuat oleh penguji 4 sudah sesuai ketentuan membuat labirin pada file readme sehingga tampilan menjadi seperti pada Gambar 5.16.



Gambar 5.15: Tampilan hasil pull request milik penguji 4

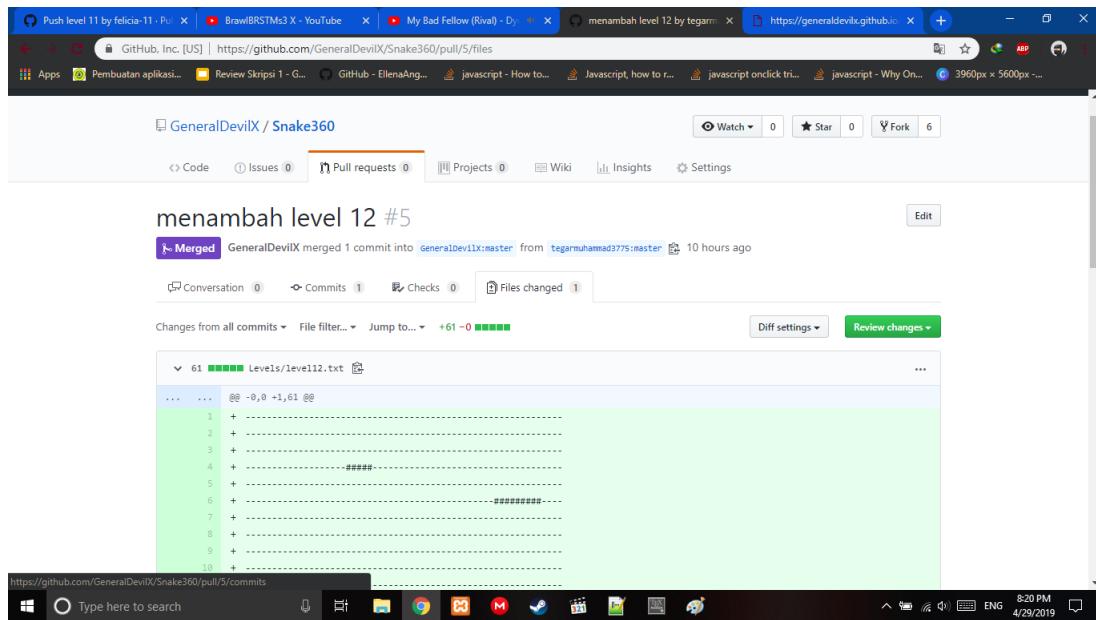


Gambar 5.16: Tampilan pengujian labirin yang dibuat oleh penguji 4

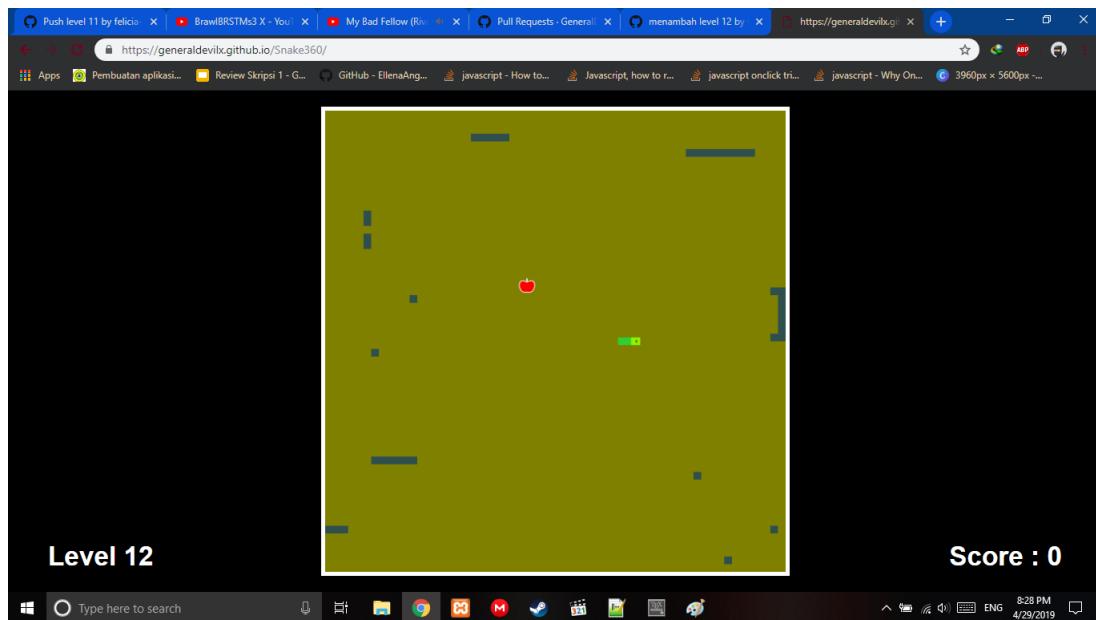
5. Penguji 5 : tegarmuhammad3775

Penguji 5 berhasil menambahkan labirin level 12 dengan menggunakan pull request seperti yang terlihat pada Gambar 5.17. Link untuk file teks yang dibuat penguji 5 : <https://github.com/GeneralDevilX/Snake360/pull/5/files>. Labirin yang dibuat oleh penguji

5 sudah sesuai ketentuan membuat labirin pada file readme sehingga tampilan menjadi seperti
6 pada Gambar 5.18.



Gambar 5.17: Tampilan hasil pull request milik penguji 5



Gambar 5.18: Tampilan pengujian labirin yang dibuat oleh penguji 5

1

BAB 6

2

KESIMPULAN DAN SARAN

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

5 **6.1 Kesimpulan**

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

- 8 1. Pembangunan permainan Open Source Snake 360 menggunakan HTML5 sudah berhasil
9 kecuali posisi apel pada labirin dan pergerakan berbelok ular pada smartphone belum sesuai
10 dengan yang diharapkan. Pada pengujian fungsional, pengujian menekan tombol '*Play Game*'
11 dan pengujian pergerakan berbelok ular pada smartphone belum berhasil.
- 12 2. Menambahkan labirin menggunakan *pull request GitHub* berhasil dilakukan. Hal ini dapat
13 dilihat berdasarkan pengujian eksperimental yang sudah dilakukan. Semua penguji berhasil
14 menambahkan labirin buatan sendiri menggunakan *pull request*.
- 15 3. Menyimpan labirin pada *file* eksternal berhasil dilakukan. Hal ini dapat dilihat pada pengujian
16 fungsional yang sudah dilakukan. Pada pengujian menekan tombol '*Play Game*', labirin sudah
17 dapat dimuat dan digambar.
- 18 4. Labirin cukup sulit untuk dibuat. Hal ini dapat dilihat berdasarkan pengujian eksperimental
19 yang sudah dilakukan. 2 dari 5 penguji tidak berhasil membuat labirin dengan benar.

20 **6.2 Saran**

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

- 23 1. Format labirin harus dibuat lebih mudah untuk mengurangi kesalahan pada pembuatan labirin
24 terutama pada memposisikan ular. Seharusnya, posisi ular tidak perlu ditulis dalam koordinat
25 melainkan dituliskan langsung menggunakan sebuah simbol pada labirin.
- 26 2. Tampilan pada Open Source Snake dapat dibuat menjadi lebih baik lagi. Pada tampilan
27 menu utama dan tampilan permainan berakhir dapat ditambahkan gambar atau animasi agar

¹ lebih menarik. Pada tampilan bermain, ular, dinding labirin dan apel tidak hanya dibuat
² menggunakan garis dan kurva saja.

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 : 5px;
24     margin:auto;
25     display:block;
26     position:absolute;
27     border : 10px solid green;
28     color: black;
29     background-color: aquamarine;
30     text-align:center;
31     top: 0;
32     bottom: 0;
33     left: 0;
34     right: 0;
35     font-family : Impact, Charcoal, sans-serif;
36     width : 400px;
37     height :400px;
38   }
39   #scoreText{
40     color : white;
41     position : fixed;
42     bottom : 3px;
43     right : 50px;
44     font-family: Arial, Helvetica, sans-serif;
45   }
46   #gameOver{
47     margin:auto;
48     color : yellow;
49     font-family: Arial, Helvetica, sans-serif;
50     visibility: hidden;
51     text-align :center;
52     position: absolute;
53     top: 50%;
54     transform: translateY(-50%);
55   }
56   #levelText{
57     color : white;
58     position : fixed;
59     bottom : 3px;
60     left : 50px;
61     font-family: Arial, Helvetica, sans-serif;
62   }
63   input[type=number]{
64     width : 60px;
65     text-align : center;
66   }
67   #snakeSpeedInvalid, #levelInvalid, #turningSpeedInvalid{
68     visibility : hidden;
69     color : red;
70   }
71   #maze{
72     background : olive;
73   }
74   #judul{
75     color: navy;
```

```

76         font-weight : bold;
77     }
78
79 </style>
80 <body>
81     <canvas id="maze">
82         <!-- Insert fallback content here -->
83     </canvas>
84     <canvas id="snake">
85         <!-- Insert fallback content here -->
86     </canvas>
87     <div id="menuDiv">
88         <p id="judul">OPEN SOURCE SNAKE 360</p><br>
89         <p>CHOOSE LEVEL (1 - <span id="totalLevel">2</span>) : <span><input type="number" value="1" id="level"></span></p>
90         <p id="levelInvalid">LEVEL IS INVALID</p>
91         <p>TURNING SPEED (1:slowest - 10:fastest) : <span><input type="number" value="1" id="turningSpeed"></span></p>
92         <p id="turningSpeedInvalid">TURNING SPEED IS INVALID</p>
93         <p>SNAKE SPEED (1:slowest - 5:fastest) : <span><input type="number" value="1" id="snakeSpeed"></span></p>
94         <p id="snakeSpeedInvalid">SNAKE SPEED IS INVALID</p>
95         <input type="button" value="PLAY GAME" id="ok">
96     </div>
97
98     <h1 id="scoreText">Score : <span id='score'>0</span></h1>
99     <h1 id="levelText">Level <span id='levels'>--</span></h1>
100    <div id="gameOver">
101        <p>GAME OVER!</p>
102        <p>Press Enter button or refresh to </p>
103        <p>try again</p>
104        <p>Your final score is : <span id="finalScore">0</span></p>
105    </div>
106    <script src="jquery-3.2.1.min.js"></script>
107    <script src="DrawingObject.js" type="text/javascript"></script>
108    <script src="Apple.js" type="text/javascript"></script>
109    <script src="Snake.js" type="text/javascript"></script>
110    <script src="Maze.js" type="text/javascript"></script>
111 </body>
112 </html>
113 <script type="text/javascript">
114     const CANVAS_SIZE = 600;
115     const GRID_SIZE = 10;
116
117     const BESAR_ULAR = GRID_SIZE;
118     const BESAR_APEL = GRID_SIZE*2;
119     const BESAR_DINDING = GRID_SIZE;
120
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         this.menu = $('#menuDiv');
127
128         this.mazeCanvas.width = CANVAS_SIZE;
129         this.mazeCanvas.height = CANVAS_SIZE;
130
131         this.myCanvas.width = CANVAS_SIZE;
132         this.myCanvas.height = CANVAS_SIZE;
133
134         this.mazeCanvas.style.border = "5px_solid_white";
135         this.myCanvas.style.border = "5px_solid_white";
136
137         this.sudut = 0;
138         var score = 0;
139         this.gameOver = false;
140         this.turningSpeed;
141
142         this.apel = new Apple(BESAR_APEL);
143         this.ular = new Snake(BESAR_ULAR);
144         this.maze = new Maze(BESAR_DINDING);
145         this.drawingObj = new DrawingObject(this.context,this.contextMaze);
146
147         //method untuk random posisi apel
148         this.randomApple = function(){
149             const besarApel = this.apel.getBesarApel();
150             const width = this.myCanvas.width-besarApel;
151             const height = this.myCanvas.height-besarApel;
152
153             let x = Math.floor(Math.random()*width);
154             let y = Math.floor(Math.random()*height);
155
156             this.moveApple(x,y);
157             if(!this.checkAppleCollisionSnake(x,y) && !this.checkAppleCollisionWall(x,y)){
158                 this.randomApple();
159             } else{
160                 this.drawingObj.drawApple(x,y,besarApel);
161             }
162         }
163
164         //cek random apel tabrakan dengan snake
165         this.checkAppleCollisionSnake = function(x,y){
166             for(var i =1; i< this.ular.tubuhSnake.length-1;i++){
167                 let posisiXUlar = this.ular.tubuhSnake[i].x;
168                 let posisiYUlar = this.ular.tubuhSnake[i].y;
169                 if(posisiXUlar > x && posisiYUlar > y &&
170                     posisiXUlar < x+this.apel.getBesarApel() && posisiYUlar < y+this.apel.getBesarApel()){
171                     return false;
172                 }
173             }
174         }

```

```

175         return true;
176     }
177 }
178 }
179
180 //cek random apel dengan dinding
181 this.checkAppleCollisionWall = function(x,y){
182     let arrayLayout = this.maze.getMazeLayout();
183     let besarApel = this.apel.getBesarApel();
184
185     let dindingX = Math.floor(x/besarApel);
186     let dindingY = Math.floor(y/besarApel);
187     let dindingX2 = Math.floor((x+besarApel)/besarApel);
188     let dindingY2 = Math.floor(y/besarApel);
189     let dindingX3 = Math.floor(x/besarApel);
190     let dindingY3 = Math.floor((y+besarApel)/besarApel);
191     let dindingX4 = Math.floor((x-besarApel)/besarApel);
192     let dindingY4 = Math.floor((y+besarApel)/besarApel);
193
194     let dindingX5 = Math.floor(x/besarApel);
195     let dindingY5 = Math.floor((y+besarApel)/besarApel);
196     let dindingX6 = Math.floor((x+besarApel)/besarApel);
197     let dindingY6 = Math.floor((y+besarApel)/besarApel);
198
199     let dindingX7 = Math.floor((x+besarApel)/besarApel);
200     let dindingY7 = Math.floor(y/besarApel);
201     let dindingX8 = Math.floor((x+besarApel)/besarApel);
202     let dindingY8 = Math.floor((y+besarApel)/besarApel);
203
204     if(arrayLayout[dindingY].charAt(dindingX) == '#' || arrayLayout[dindingY2].charAt(dindingX2) == '#' ||
205         arrayLayout[dindingY3].charAt(dindingX3) == '#' || arrayLayout[dindingY4].charAt(dindingX4) == '#' ||
206         arrayLayout[dindingY5].charAt(dindingX5) == '#' || arrayLayout[dindingY6].charAt(dindingX6) == '#' ||
207         arrayLayout[dindingY7].charAt(dindingX7) == '#' || arrayLayout[dindingY8].charAt(dindingX8) == '#'){
208         return false;
209     }
210     return true;
211 }
212
213 //method untuk memindahkan apel
214 this.moveApple = function(x,y){
215     this.apel.x = x;
216     this.apel.y = y;
217 }
218
219 //method untuk mengarahkan ular ke atas,bawah,kiri,kanan
220 this.moveLeft = function(){
221     this.sudut-=this.turningSpeed;
222     if(this.sudut < 0){
223         this.sudut = 360;
224     }
225 }
226 this.moveRight = function(){
227     this.sudut+=this.turningSpeed;
228     if(this.sudut > 360){
229         this.sudut = 0;
230     }
231 }
232
233
234 //method supaya ular dapat keluar dari sisi lain
235 this.sampaiUjung = function(){
236     if (this.ular.x < 0) {
237         this.ular.x = this.myCanvas.width;
238     }
239     else if (this.ular.x >= this.myCanvas.width) {
240         this.ular.x = 0;
241     }
242     if (this.ular.y < 0) {
243         this.ular.y = this.myCanvas.height-1;
244     }
245     else if (this.ular.y >= this.myCanvas.height) {
246         this.ular.y = 0;
247     }
248 }
249
250 //untuk menghilangkan text gameOver
251 this.removeGameOverText = function(){
252     let gameOverText = document.getElementById("gameOver");
253     gameOverText.style.visibility = "hidden";
254 }
255
256 //memulai game
257 this.startGame = function(kelas){
258     kelas.removeGameOverText();
259
260     let level = $('#level').val();
261     $('#levels').html(level);
262     let turningSpeedVar = $('#turningSpeed').val();
263     kelas.turningSpeed = parseInt(turningSpeedVar);
264     let snakeSpeed = $('#snakeSpeed').val();
265     kelas.ular.setSpeed(parseInt(snakeSpeed));
266
267     let layout = kelas.maze.getMazeLayout();
268     let posisi = layout[layout.length-1].split("„");
269
270     kelas.ular.x = parseInt(posisi[0]);
271     kelas.ular.y = parseInt(posisi[1]);
272     kelas.randomApple();
273     kelas.drawingObj.drawSnake(kelas.ular.tubuhSnake,kelas.ular.besarUlar);

```

```

274     kelas.drawingObj.drawMaze(kelas.maze.getMazeLayout(), kelas.maze.getBesarDinding());
275 }
276
277 //cek collision ular dengan apel
278 this.appleCollision = function(){
279     let posisiApelX = this.apel.x;
280     let posisiApelY = this.apel.y;
281     let posisiXUlar = this.ular.tubuhSnake[0].x;
282     let posisiYUlar = this.ular.tubuhSnake[0].y;
283
284     if(posisiXUlar > posisiApelX && posisiYUlar > posisiApelY &&
285        posisiXUlar < posisiApelX+this.apel.besarApel && posisiYUlar < posisiApelY+this.apel.besarApel){
286         this.appleEaten();
287     }
288 }
289
290 // ular memakan apel
291 this.appleEaten = function(){
292     this.randomApple();
293     score++;
294     this.ular.panjangTubuhSaatIni++;
295     $('#score').html(score);
296 }
297
298 //ular menabrak diri sendiri
299 this.snakeCollision = function(){
300     let posisiXUlar = this.ular.tubuhSnake[0].x;
301     let posisiYUlar = this.ular.tubuhSnake[0].y;
302     const besarBoundary = 1;
303
304     for(var i = 1; i< this.ular.tubuhSnake.length;i++){
305         let bagianTubuhSnakeX = this.ular.tubuhSnake[i].x;
306         let bagianTubuhSnakeY = this.ular.tubuhSnake[i].y;
307
308         if(posisiXUlar > bagianTubuhSnakeX-besarBoundary && posisiYUlar > bagianTubuhSnakeY-besarBoundary &&
309            posisiXUlar < bagianTubuhSnakeX+besarBoundary && posisiYUlar < bagianTubuhSnakeY+besarBoundary ){
310             this.gameOver = true;
311             this.popUpGameOverText();
312         }
313     }
314 }
315
316 //ular menabrak dinding
317 this.wallCollision = function(){
318     let arrayLayout = this.maze.getMazeLayout();
319     let posisiXUlar = this.ular.tubuhSnake[0].x;
320     let posisiYUlar = this.ular.tubuhSnake[0].y;
321     let boundaryWallX = Math.floor(posisiXUlar/10);
322     let boundaryWallY = Math.floor(posisiYUlar/10);
323
324     if(arrayLayout[boundaryWallY].charAt(boundaryWallX) == '#'){
325         this.gameOver = true;
326         this.popUpGameOverText();
327     }
328 }
329
330 //untuk menampilkan text gameOver
331 this.popUpGameOverText = function(){
332     let gameOverText = document.getElementById("gameOver");
333     $('#finalScore').html(score);
334     gameOverText.style.visibility = "visible";
335 }
336
337
338
339 //mendapatkan status gameOver
340 this.getGameOverStatus = function(){
341     return this.gameOver;
342 }
343
344 //posisi untuk animasi
345 this.animate = function(){
346     if(this.gameOver == true){
347
348     } else{
349         this.context.clearRect(0,0,this.myCanvas.width,this.myCanvas.height);
350
351         for(var i = 1;i<=this.ular.getSpeed();i++){
352             this.ular.move(this.sudut);
353             this.sampaiUjung();
354
355             this.temp = {};
356             this.temp['x'] = this.ular.x;
357             this.temp['y'] = this.ular.y;
358
359             this.ular.tubuhSnake.unshift(this.temp);
360             if(this.ular.tubuhSnake.length > this.ular.panjangTubuhSaatIni){
361                 this.ular.tubuhSnake.pop();
362             }
363         }
364
365         this.drawingObj.drawSnake(this.ular.tubuhSnake, this.ular.besarUlar);
366         this.drawingObj.drawApple(this.apel.x, this.apel.y, this.apel.getBesarApel());
367         this.wallCollision();
368         this.appleCollision();
369         this.snakeCollision();
370
371         var that = this;
372     }
}

```

```

373         setTimeout(function(){
374             that.animate();
375         },50);
376     }
377 }
378
379 this.changeLevel = function(levels){
380     $("#totalLevel").html(levels);
381     return levels;
382 }
383
384 this.countLevels = function(level){
385     var that = this;
386     url = "Levels/level"+level+".txt";
387     $.get( url, function() {
388         level+=1;
389         that.countLevels(level);
390     })
391     .fail(function() {
392         level = level-1;
393         that.changeLevel(level);
394     });
395 }
396
397 this.loadMaze = function(callback){
398     let level = $('#level').val();
399     let url = "Levels/level"+level+".txt";
400     let temp = this;
401
402     $.ajax({
403         url: "Levels/level"+level+".txt",
404         dataType: 'text',
405         context: temp,
406         success: function(data,textStatus,jqXHR) {
407             temp.maze.setMazeLayout(jqXHR.responseText);
408             callback(temp);
409         }
410     });
411 }
412
413 }
414
415 $(document).ready(function(){
416     var snakeSpeedValid = true;
417     var turningSpeedValid = true;
418     var levelValid = true;
419
420     resize();
421     var permainan = new Game();
422     permainan.countLevels(1);
423
424     function resize(){
425         var width = window.innerWidth-50;
426         var height = window.innerHeight-50;
427         var snake = $('#snake');
428         var maze = $('#maze');
429         var menu = $('#menuDiv');
430
431         if(width < 600 || height < 600){
432             if(width > height){
433                 maze.css('width', height);
434                 maze.css('height',height);
435                 snake.css('width', height);
436                 snake.css('height',height);
437             }
438             if(height > width){
439                 maze.css('height',width);
440                 maze.css('width',width);
441                 snake.css('height',width);
442                 snake.css('width',width);
443             }
444         }
445         else{
446             maze.css('width', '600px');
447             maze.css('height','600px');
448             snake.css('width', '600px');
449             snake.css('height','600px');
450         }
451         var widthMenu = snake.width()-75;
452         var heightMenu = snake.height()-75;
453
454         menu.css('width',widthMenu);
455         menu.css('height',heightMenu);
456
457         menu.css('font-size',menu.width()/25);
458         $('#scoreText').css('font-size',menu.width()/15);
459         $('#levelText').css('font-size',menu.width()/15);
460         $('#gameOver').css('font-size',menu.width()/15);
461     }
462 }
463
464 window.addEventListener('resize',resize,false);
465 window.addEventListener('orientationchange',resize,false);
466
467 document.getElementById('level').addEventListener('change',function(){
468     let chosenLevel = document.getElementById("level").value;
469     let temp = $("#totalLevel").html();
470     let totalLevel = parseInt(temp);
471

```

```

472     if(chosenLevel != ""){
473         if(chosenLevel > 0 && chosenLevel <= totalLevel){
474             document.getElementById("levelInvalid").style.visibility = "hidden";
475             levelValid = true;
476         }
477         else{
478             document.getElementById("levelInvalid").style.visibility = "visible";
479             levelValid = false;
480         }
481     }
482     else{
483         document.getElementById("levelInvalid").style.visibility = "visible";
484         levelValid = false;
485     }
486 });
487
488 $('#turningSpeed').change(function(){
489     let chosenSpeed = document.getElementById("turningSpeed").value;
490
491     if(chosenSpeed != ""){
492         if(chosenSpeed > 0 && chosenSpeed <= 10){
493             document.getElementById("turningSpeedInvalid").style.visibility = "hidden";
494             turningSpeedValid = true;
495         }
496         else{
497             document.getElementById("turningSpeedInvalid").style.visibility = "visible";
498             turningSpeedValid = false;
499         }
500     }
501     else{
502         document.getElementById("turningSpeedInvalid").style.visibility = "visible";
503         turningSpeedValid = false;
504     }
505 });
506
507 $('#snakeSpeed').change(function(){
508     let chosenSpeed = document.getElementById("snakeSpeed").value;
509
510     if(chosenSpeed != ""){
511         if(chosenSpeed > 0 && chosenSpeed <= 5){
512             document.getElementById("snakeSpeedInvalid").style.visibility = "hidden";
513             snakeSpeedValid = true;
514         }
515         else{
516             document.getElementById("snakeSpeedInvalid").style.visibility = "visible";
517             snakeSpeedValid = false;
518         }
519     }
520     else{
521         document.getElementById("snakeSpeedInvalid").style.visibility = "visible";
522         snakeSpeedValid = false;
523     }
524 });
525
526
527 document.getElementById('ok').addEventListener('click',function(){
528     if(levelValid && turningSpeedValid && snakeSpeedValid){
529         document.getElementById('menuDiv').style.visibility = 'hidden';
530         permainan.loadMaze(permainan.startGame);
531
532         function wait(){
533             if(permainan.maze.getMazeLayout() == null){
534                 setTimeout(function(){
535                     wait();
536                 },50);
537             }
538             else{
539                 permainan.animate();
540             }
541         }
542
543         //tombol pergerakan ular
544         document.addEventListener('keydown', function(e) {
545             if (e.keyCode == 37) {
546                 permainan.moveLeft();
547             }
548             else if (e.keyCode == 39) {
549                 permainan.moveRight();
550             }
551
552             else if(e.keyCode == 13 && permainan.getGameOverStatus() == true){
553                 document.location.href="";
554             }
555         });
556
557
558         var onlongtouch;
559         var timer;
560         var touchduration = 100;
561
562         window.addEventListener('touchstart',function(e){
563             timer = setInterval(function(){
564                 var width = window.innerWidth;
565                 var touchX = e.touches[0].clientX;
566                 if(touchX > width/2){
567                     permainan.moveLeft();
568                 }
569                 else{
570                     permainan.moveRight();
571                 }
572             });
573         });
574     }
575 });

```

```

571         }
572     },touchduration);
573 },false);
574
575     window.addEventListener('touchend',function(){
576         if (timer)
577             clearInterval(timer);
578     },false);
579
580     window.oncontextmenu = function(event) {
581         event.preventDefault();
582         event.stopPropagation();
583         return false;
584     };
585     wait();
586 });
587 });
588 });
589 </script>

```

Listing A.2: Snake.js

```

1 function Snake(besarUlar){
2     this.x;
3     this.y;
4     this.panjangTubuhSaatIni = 15;
5     this.speed;
6     this.tubuhSnake = [{x:this.x,y:this.y}];
7     this.besarUlar = besarUlar;
8     this.jarakAntaraTubuhUlar = 2;
9
10    this.move = function(sudut){
11        this.x += Math.cos(sudut*Math.PI/180)*this.jarakAntaraTubuhUlar;
12        this.y += Math.sin(sudut*Math.PI/180)*this.jarakAntaraTubuhUlar;
13    }
14
15    this.setSpeed = function(speed){
16        this.speed = speed;
17    }
18
19    this.getSpeed = function(){
20        return this.speed;
21    }
22
23    this.getBesarUlar = function(){
24        return this.besarUlar;
25    }
26
27    this.getTubuhSnake = function(){
28        return this.tubuhSnake;
29    }
30
31    this.getPanjangTubuh = function(){
32        return this.panjangTubuhSaatIni;
33    }
34
35    this.getX = function(){
36        return this.x;
37    }
38
39    this.getY = function(){
40        return this.y;
41    }
42 }

```

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     this.getX = function(){
10        return this.x;
11    }
12     this.getY = function(){
13        return this.y;
14    }
15 }

```

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                 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     //untuk gambar maze
63     this.drawMaze = function(arrayLayout,besarDinding){
64         this.contextMaze.lineWidth = besarDinding;
65         this.contextMaze.fillStyle = 'darkslategrey';
66
67         for(var i = 0;i< arrayLayout.length; i++){
68             let temp = arrayLayout[i];
69             for(var j = 0;j< arrayLayout.length;j++){
70                 if(temp.charAt(j) == '#'){
71                     this.contextMaze.fillRect(j*besarDinding,i*besarDinding,besarDinding,besarDinding);
72                 }
73             }
74         }
75     }
76 }

```

LAMPIRAN B

FILE README

Listing B.1: README.md

```
1 Open Source Snake 360
2 =====
3 Open Source Snake 360 adalah sebuah permainan di mana pemain mengendalikan ular untuk mendapatkan makanan sebanyak-banyaknya tanpa menabrak dinding atau tubuh ular itu sendiri. Pada Open Source Snake 360, ular dapat bergerak 360° dan Anda dapat menambahkan labirin buatan sendiri untuk menambah variasi labirin yang ada.
4
5 ## Kontrol bermain
6 * Desktop
7   - tombol <- : menggerakan ular berlawanan arah jarum jam
8   - tombol -> : menggerakan ular searah jarum jam
9 * Smartphone
10  - menekan bagian kiri layar : menggerakan ular berlawanan arah jarum jam
11  - menekan bagian kanan layar : menggerakan ular searah jarum jam
12
13 ## Cara menambah labirin
14 Berikut adalah cara untuk menambah labirin :
15 1. Fork dan clone repository ini
16 2. Sinkronkan repository fork yang Anda buat dengan repository aslinya(repository ini). Caranya adalah dengan menuliskan perintah
   : ``'$ git remote add upstream https://github.com/GeneralDevilX/Snake360/'` kemudian gunakan ``'$ git fetch upstream'``
   untuk mengambil repo asli
17 3. Checkout ke branch utama dan merge repo Anda dengan branch upstream yang sudah Anda tambahkan
18 4. Buat labirin (aturan pembuatan labirin sudah dijelaskan di bawah)
19 5. Commit dan push perubahan pada repository yang sudah Anda fork
20 6. Buat pull request ke repository ini
21
22 **Note : Untuk point 2 dan 3 harus dilakukan apabila Anda ingin update repo Anda dengan repo ini**
23
24 ## Membuat labirin
25 Berikut adalah hal yang perlu diperhatikan untuk membuat labirin :
26 * File labirin diawali dengan 'level' kemudian diikuti dengan angka. Angka yang digunakan adalah angka dari level terbaru. Misal :
   bila labirin terakhir adalah "level3.txt", maka labirin baru akan bernama "level4.txt"
27 * File level dibuat pada folder bernama 'Levels'
28 * Labirin dibuat pada file text (.txt)
29 * Labirin dibuat dengan dimensi 60 x 60. File text memiliki 60 baris dan setiap barisnya memiliki 60 karakter
30 * Karakter '#' merepresentasikan dinding dan karakter '-' merepresentasikan tidak ada dinding
31 * Baris terakhir diisi dengan posisi awal ular yaitu koordinat X ular dan koordinat Y ular yang keduanya dipisahkan oleh spasi (
   koordinat minimal untuk X dan Y adalah 0 dan koordinat maksimal untuk X dan Y adalah 600). Contoh: apabila posisi ular
   adalah (100,200) maka Anda harus mengganti karakter ke 10 dari baris ke 20.
32 * Tidak boleh ada space kosong di akhir file teks
```