

SKRIPSI

PEMBANGUNAN GIM SNAKE 360 BERBASIS WEB DENGAN KODE TERBUKA



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**

LEMBAR PENGESAHAN

PEMBANGUNAN GIM SNAKE 360 BERBASIS WEB DENGAN KODE TERBUKA

Evelyn Wijaya

NPM: 2015730030

Bandung, 24 Mei 2019

Menyetujui,

Pembimbing

Dr. Veronica Sri Moertini

Ketua Tim Penguji

Anggota Tim Penguji

Chandra Wijaya, M.T.

Raymond Chandra Putra S.T., M.T.

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

Dengan ini saya yang bertandatangan di bawah ini menyatakan bahwa skripsi dengan judul:

PEMBANGUNAN GIM SNAKE 360 BERBASIS WEB DENGAN KODE TERBUKA

adalah benar-benar karya saya sendiri, dan saya tidak melakukan penjiplakan atau pengutipan dengan cara-cara yang tidak sesuai dengan etika keilmuan yang berlaku dalam masyarakat keilmuan.

Atas pernyataan ini, saya siap menanggung segala risiko dan sanksi yang dijatuhan kepada saya, apabila di kemudian hari ditemukan adanya pelanggaran terhadap etika keilmuan dalam karya saya, atau jika ada tuntutan formal atau non-formal dari pihak lain berkaitan dengan keaslian karya saya ini.

Dinyatakan di Bandung,
Tanggal 24 Mei 2019

Meterai
Rp. 6000

Evelyn Wijaya
NPM: 2015730030

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. Umumnya pada permainan *Snake*, ular hanya dapat bergerak ke atas, ke bawah, ke kiri dan ke kanan saja. Selain itu labirin yang disediakan terbatas. Tujuan dari penelitian ini adalah membangun permainan *Snake* yang ularnya dapat bergerak ke segala arah dan orang lain dapat menambahkan labirin buatan sendiri.

Open Source Snake 360 adalah sebuah permainan yang dibuat menggunakan HTML5 Canvas dan *Javascript*. *jQuery* digunakan untuk mengakses labirin dan memuat labirin dari server. Pull request Github digunakan untuk menambahkan labirin buatan sendiri.

Berdasarkan penelitian yang sudah dilakukan dapat diperoleh kesimpulan bahwa 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: Permainan *Snake*, HTML5 Canvas, *Javascript*, *jQuery*, *GitHub*

ABSTRACT

This study discusses the construction of the Open Source Snake 360. This game is based on a reference to the existing game, Snake. Snake is a game that controls the movement of snake to get food that scattered in the maze. In Snake, the snake can only move up, down, left and right. In addition, the labyrinth provided is limited. The purpose of this research is to build a game of Snake whose snake can move in any direction and others can add mazes of their own.

Open Source Snake 360 is a game created using HTML5 Canvas and Javascript. jQuery is used to access the maze and load the maze from the server. Pull request GitHub is used to add mazes made by others.

Based on the research that has been done, it can be concluded that the snake can move in any direction and other people can add their own mazes. Others can add their own mazes by using pull request on GitHub. This game can also load mazes made by other people.

Keywords: Snake Game, HTML5 Canvas, Javascript, jQuery, GitHub

Dipersembahkan kepada orang tua

KATA PENGANTAR

Puji Syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas karunia-Nya, sehingga penulis dapat menyelesaikan skripsi yang berjudul Pembangunan Gim Snake 360 Berbasis Web dengan Kode Terbuka. Skripsi ini ditulis untuk memenuhi syarat kelulusan pada Teknik Informatika Unpar.

Penulisan skripsi ini tidak dapat selesai pada waktunya tanpa bantuan dari berbagai pihak. Oleh karena itu penulis ingin mengucapkan terima kasih yang sebesar-besarnya kepada :

1. Tuhan Yang Maha Esa yang memberikan kekuatan dan selalu menyertai dalam pembuatan skripsi ini.
2. Bapak Pascal Alfadian Nugroho selaku dosen pembimbing skripsi yang telah membimbing dan membantu penulis dalam pembuatan skripsi ini.
3. Ayah dan Ibu yang selalu mendoakan dan telah memberikan semangat dan motivasi kepada penulis agar dapat menyelesaikan skripsi ini.
4. Teman-teman Teknik Informatika yang telah menemani selama 7 semester.
5. Semua pihak yang membantu penulis dan tidak sempat terucapkan.

Penulis berharap bahwa skripsi ini dapat memberikan informasi tentang pembangunan permainan berbasis web kepada masyarakat. Penulis menyadari bahwa skripsi ini masih terdapat banyak kekurangan. Oleh karena itu, penulis bersedia menerima kritik dan saran yang membangun.

Bandung, Mei 2019

Penulis

DAFTAR ISI

KATA PENGANTAR	xv
DAFTAR ISI	xvii
DAFTAR GAMBAR	xix
DAFTAR TABEL	xxi
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>Open Source</i>	6
2.3 <i>HTML5 Canvas</i>	6
2.4 <i>Javascript</i>	7
2.4.1 Menggambar pada <i>Canvas</i>	8
2.4.2 <i>Object Oriented Programming Javascript</i>	10
2.4.3 <i>Event</i>	12
2.4.4 Membuat Animasi	14
2.4.5 Fungsi <i>Callback</i>	15
2.5 <i>jQuery</i>	16
2.5.1 <i>Looping</i>	17
2.5.2 <i>Chaining</i>	17
2.5.3 Mendapatkan dan Mengubah Konten Elemen	18
2.5.4 Mendapatkan dan Mengubah Properti CSS	18
2.5.5 Event	19
2.5.6 Mengubah dan Mendapatkan Dimensi dari Elemen	20
2.6 <i>AJAX</i>	20
2.6.1 <i>Ajax Request</i> dan <i>Response</i>	21
2.6.2 <i>jQuery</i> dan <i>AJAX Request</i>	22
2.6.3 <i>jQuery</i> dan <i>AJAX Response</i>	22
2.7 <i>Version Control</i>	23
2.7.1 <i>Git</i>	24
2.7.2 <i>Git Branching</i>	27
2.7.3 <i>GitHub</i>	33
2.8 Teknik Tabrakan	34

3 ANALISIS	35
3.1 Analisis Permainan <i>Snake</i> yang Sudah Ada	35
3.1.1 Ular dan Makanan	35
3.1.2 Pergerakan Ular	36
3.1.3 Labirin	37
3.2 Analisis Sistem yang Dibangun	37
3.2.1 Menggambar Ular dan Apel	37
3.2.2 Pergerakan Ular	40
3.2.3 Mengacak posisi apel	41
3.2.4 Menentukan Besar <i>Canvas</i>	42
3.2.5 Membuat Labirin	42
3.2.6 Pengecekan tabrakan(<i>Collision Detection</i>)	43
3.2.7 Memilih Labirin, Kecepatan Laju Ular dan Kecepatan Berbelok Ular	45
3.3 Analisis Berorientasi Objek	46
3.3.1 Skenario Permainan	46
3.3.2 Diagram Kelas	46
4 PERANCANGAN	49
4.1 Rancangan Diagram <i>Sequence</i>	49
4.2 Rancangan Diagram <i>State</i>	50
4.3 Rancangan Diagram Kelas Rinci	50
4.4 Rancangan Tampilan Antarmuka	55
4.4.1 Tampilan Menu Utama	55
4.4.2 Tampilan Bermain	56
4.4.3 Tampilan Permainan Berakhir	56
5 IMPLEMENTASI DAN PENGUJIAN	59
5.1 Implementasi	59
5.1.1 Lingkungan Perangkat Keras	59
5.1.2 Lingkungan Perangkat Lunak	60
5.1.3 Lingkungan Pengujian	60
5.1.4 Implementasi Antarmuka	61
5.2 Pengujian	66
5.2.1 Pengujian Fungsional	66
5.2.2 Pengujian Eksperimental	69
6 KESIMPULAN DAN SARAN	77
6.1 Kesimpulan	77
6.2 Saran	77
DAFTAR REFERENSI	79
A KODE PROGRAM	81
B FILE README	89

DAFTAR GAMBAR

2.1	Permainan Snake pada telepon genggam <i>Nokia</i>	5
2.2	Permainan <i>Slither.io</i> pada <i>Android</i>	5
2.3	Posisi kotak biru pada <i>canvas</i> terhadap <i>origin</i>	8
2.4	Perbedaan <i>quadratic Bézier curve</i> dan <i>cubic Bézier curve</i>	10
2.5	Local Version Control	23
2.6	Centralized Version Control	24
2.7	Distributed Version Control	24
2.8	Working tree, staging area, dan Git directory	25
2.9	Siklus hidup pada status <i>file</i>	26
2.10	Commit dan tree dari file yang dicommit	28
2.11	Commit dan parent dari commit	28
2.12	Pointer <i>HEAD</i> menunjuk <i>branch master</i>	29
2.13	Pointer <i>HEAD</i> beserta <i>branch testing</i>	29
2.14	3 <i>snapshot</i> yang digunakan dalam <i>three way merge</i>	30
2.15	<i>Merge commit</i>	30
2.16	Perbedaan pada <i>branch</i> lokal dan <i>remote</i>	31
2.17	<i>Update remote-tracking branches</i> menggunakan perintah <i>git fetch</i>	31
2.18	<i>Rebasing commit C4 ke C3</i>	32
2.19	<i>Merge branch</i> setelah <i>rebasing</i>	32
2.20	Tombol 'Fork'	33
3.1	Ular pada <i>Slither.io</i>	35
3.2	Makanan pada <i>Slither.io</i>	36
3.3	Ular pada <i>Snake Nokia</i>	36
3.4	Makanan biasa(A) dan makanan bonus(B) pada <i>Snake Nokia</i>	36
3.5	Ular sedang melaju dengan cepat(<i>speed up</i>)	37
3.6	Peta labirin pada <i>Slither.io</i>	37
3.7	Koordinat bagian tubuh ular pada <i>array</i>	38
3.8	Tubuh ular setelah digambar menggunakan garis	38
3.9	Bagian pada apel(lingkaran merah) yang akan dibuat menggunakan kurva	38
3.10	Pembagian gambar apel dengan layout persegi beserta ukuran pada setiap bagian	39
3.11	<i>Start point, control point</i> dan <i>end point</i> untuk menggambar apel bagian kiri atas	40
3.12	<i>Start point, control point</i> dan <i>end point</i> untuk menggambar apel bagian kiri bawah	40
3.13	Ilustrasi ular sebelum bergerak maju(A) dan setelah bergerak maju(B)	40
3.14	Gambar apel yang terpotong sesudah mengacak posisi apel	42
3.15	Menggambar dinding menggunakan simbol pada file teks	43
3.16	Ular ingin melewati jalur yang diapit oleh 2 buah dinding	43
3.17	Daerah tabrakan pada apel	44
3.18	Daerah tabrakan berbentuk persegi pada apel	44
3.19	Posisi kepala ular pada sebuah daerah labirin	45
3.20	Diagram <i>use case</i> dari permainan <i>Snake 360</i>	46
3.21	Diagram class dari permainan <i>Snake 360</i>	47

4.1	Diagram <i>sequence</i> untuk memuat labirin	49
4.2	Diagram state proses bermain dari Open Source <i>Snake 360</i>	50
4.3	Diagram class rinci dari Open Source <i>Snake 360</i>	51
4.4	Rancangan tampilan menu utama	55
4.5	Rancangan tampilan menu utama jika pemain salah memasukkan data	56
4.6	Rancangan tampilan bermain	56
4.7	Rancangan tampilan permainan berakhir	57
5.1	Tampilan memilih branch repository untuk dihosting	60
5.2	Tampilan setelah melakukan hosting di GitHub	61
5.3	Tampilan menu utama pada <i>desktop</i>	61
5.4	Tampilan menu utama pada <i>smartphone</i>	62
5.5	Tampilan menu utama jika pemain salah memasukkan data labirin pada <i>desktop</i>	62
5.6	Tampilan menu utama jika pemain salah memasukkan data labirin pada <i>smartphone</i>	63
5.7	Tampilan bermain pada <i>desktop</i>	64
5.8	Tampilan bermain pada <i>smartphone</i>	64
5.9	Tampilan permainan berakhir pada <i>desktop</i>	65
5.10	Tampilan permainan berakhir pada <i>smartphone</i>	66
5.11	Tampilan hasil <i>pull request</i> milik penguji 1	69
5.12	Tampilan pengujian labirin ke 6 yang dibuat oleh penguji 1	70
5.13	Tampilan hasil <i>pull request</i> milik penguji 2	71
5.14	Tampilan pengujian labirin ke 7 yang dibuat oleh penguji 2	71
5.15	Tampilan pengujian labirin ke 8 yang dibuat oleh penguji 2	72
5.16	Tampilan pengujian labirin ke 9 yang dibuat oleh penguji 2	72
5.17	Tampilan hasil pull request milik penguji 3	73
5.18	Tampilan pengujian labirin yang dibuat oleh penguji 3	73
5.19	Tampilan hasil <i>pull request</i> milik penguji 4	74
5.20	Tampilan pengujian labirin yang dibuat oleh penguji 4	74
5.21	Tampilan hasil <i>pull request</i> milik penguji 5	75
5.22	Tampilan pengujian labirin yang dibuat oleh penguji 5	75

DAFTAR TABEL

5.1 Pengujian Fungsional pada Tampilan Menu Utama	67
5.2 Pengujian Fungsional Tampilan Bermain pada Desktop	67
5.3 Pengujian Fungsional Tampilan Bermain pada <i>Smartphone</i>	68
5.4 Pengujian Fungsional Tampilan Permainan Berakhir	68

BAB 1

PENDAHULUAN

1.1 Latar Belakang

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

HTML(*Hyper Text Markup Language*) adalah sebuah bahasa markah yang digunakan untuk membuat halaman web. HTML5 merupakan HTML versi 5 dan penerus dari HTML4, XHTML1, dan DOM *level 2* HTML. HTML5 memiliki beberapa elemen baru, salah satunya adalah HTML5 Canvas. HTML5 Canvas adalah tempat untuk menggambar *pixel-pixel* yang dapat ditulis menggunakan bahasa pemrograman *JavaScript*. *JavaScript* adalah bahasa pemrograman tingkat tinggi yang digunakan untuk membuat halaman web menjadi lebih interaktif. *jQuery* merupakan *library Javascript* yang cepat, kecil dan kaya dengan fitur. *jQuery* membuat hal-hal seperti traversal dan manipulasi dokumen HTML, penanganan *event*, animasi dan *Ajax* jauh lebih sederhana dengan API(*Application Programming Interface*) yang mudah untuk digunakan pada banyak *browser*. *GitHub* adalah layanan *web hosting* bersama untuk proyek pengembangan perangkat lunak yang menggunakan sistem *version control* yaitu *Git*. Dengan adanya *GitHub*, *programmer* dapat mengetahui perubahan yang pada *repository* tersebut. *Pull request* adalah sebuah fitur *GitHub* yang digunakan sebagai tempat diskusi antara kontributor dengan pemilik proyek/*owner*. Proyek yang diajukan oleh kontributor akan digabungkan dengan proyek milik *owner* jika *owner* merasa senang dengan perubahan yang sudah dibuat oleh kontributor.

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

1.2 Rumusan Masalah

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

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

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

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

1.3 Tujuan

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

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

1.4 Batasan Masalah

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

- *Web browser* yang digunakan sudah mendukung HTML5 Canvas.
- Permainan ini hanya dapat dimainkan oleh 1 orang.

1.5 Metodologi

Metodologi pada penelitian ini adalah sebagai berikut:

1. Melakukan studi literatur tentang HTML5 Canvas, *Open Source*, *JavaScript*, *jQuery*, *AJAX*, dan *Version Control*.
2. Melakukan analisis dan menentukan objek-objek.
3. Merancang algoritma untuk menggambar tubuh ular, pergerakan ular dan membuat labirin.
4. Mengimplementasikan keseluruhan algoritma.
5. Menambahkan labirin menggunakan *pull request* pada *GitHub*.
6. Melakukan pengujian.
7. Melakukan penarikan kesimpulan.
8. Menuliskan dokumen skripsi.

1.6 Sistematika Pembahasan

Sistematikan penulisan setiap bab pada penelitian ini adalah sebagai berikut:

1. Bab 1 berisikan latar belakang, rumusan masalah, tujuan, batasan masalah, metodologi, dan sistematika pembahasan dari penelitian yang dilakukan.
2. Bab 2 berisikan dasar-dasar teori yang menunjang penelitian ini. Teori yang digunakan adalah: pengertian *Snake Game*, *Open Source*, HTML5 Canvas, *Javascript*, *jQuery*, *AJAX*, dan *Version Control*.
3. Bab 3 berisikan analisis sistem yang sudah ada, analisis sistem yang dibangun dan analisis berorientasi objek. Analisis sistem yang sudah ada berisikan analisis permainan *Snake Nokia* dan *Slither.io*. Analisis berorientasi objek berisikan skenario permainan dan diagram kelas.

4. Bab 4 berisikan perancangan perangkat lunak yang dibangun. Perancangan yang dilakukan meliputi perancangan diagram *sequence*, perancangan diagram kelas dan perancangan tampilan antarmuka.
5. Bab 5 berisikan implementasi dan pengujian perangkat lunak.
6. Bab 6 berisikan kesimpulan dan saran.

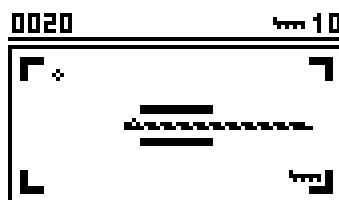
BAB 2

LANDASAN TEORI

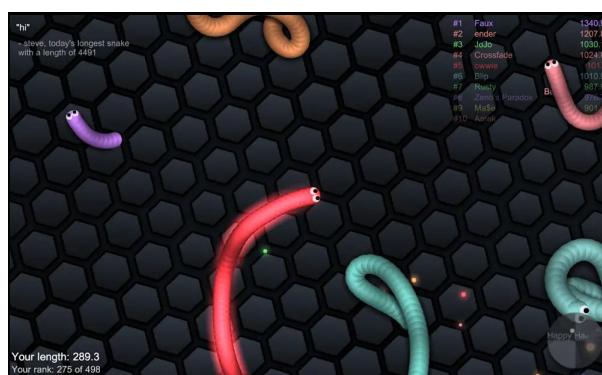
2.1 *Snake* Game

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

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



Gambar 2.1: Permainan *Snake* pada telepon genggam *Nokia*



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

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>

2.2 Open Source

Open source adalah istilah yang digunakan pada perangkat lunak yang sumber kodennya dapat dilihat dan diubah/dimodifikasi oleh orang lain untuk memperbaiki dan membantu mengembangkan perangkat lunak menjadi lebih baik.^[1] Proyek *open source* dikerjakan oleh pengembang lain yang tidak kita kenal. Proyek *open source* belum tentu menarik perhatian pengembang lainnya untuk membantu mengembangkan proyek tersebut. Selain itu, pemilik proyek harus membuat dokumentasi tentang proyek, membuat forum diskusi dan memastikan bahwa sumber kode proyek dapat dilihat dan dimodifikasi oleh orang lain. Jika ada pengembang yang tertarik dengan proyek tersebut, maka pemilik proyek harus sudah siap untuk menjawab pertanyaan meskipun hasil yang dilakukan oleh pengembang belum terlihat. Proyek *open source* memiliki kerja ekstra dibandingkan dengan proyek *closed source*, namun hasilnya dapat terlihat pada saat mulai pengembangan proyek. Keuntungan dari proyek ini adalah pengembang akan merasa puas dengan hasil proyek dan kerjanya akan dihargai oleh orang lain.

2.3 HTML5 Canvas

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

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

```

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

Listing 2.1: Menambahkan *canvas*

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

Canvas memiliki beberapa atribut diantaranya adalah:

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

- *dir* : arah penulisan (dari kiri ke kanan atau dari kanan ke kiri)
- *hidden* : membuat elemen menjadi tersembunyi/tidak terlihat
- *accesskey* : memberikan petunjuk untuk membuat pintasan *keyboard* pada sebuah elemen.

2.4 Javascript

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

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

```

1  <!DOCTYPE html>
2      <html>
3          <body>
4
5              <h1>A Web Page</h1>
6              <p id="demo">A Paragraph</p>
7
8              <script>
9                  // tuliskan script di sini
10             </script>
11
12         </body>
13     </html>
```

Listing 2.2: Internal Javascript

```

1  <!DOCTYPE html>
2      <html>
3          <body>
4
5              <h1>A Web Page</h1>
6              <p id="demo">A Paragraph</p>
7
8              <script src="myScript1.js"></script>
9
10             </body>
11         </html>
```

Listing 2.3: External Javascript

2.4.1 Menggambar pada *Canvas*

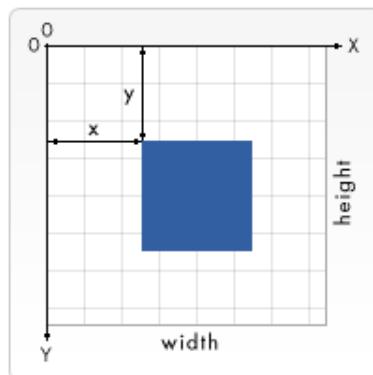
Sesudah menuliskan tag <canvas> pada HTML, *canvas* tidak bisa langsung digambar. Karena itu perlu ditambahkan *drawing context* pada Javascript. Pada *listing 2.4* terdapat potongan kode untuk menambahkan *drawing context*.

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

Listing 2.4: Menambahkan *drawing context canvas*

Berdasarkan *listing 2.4*, variabel *myCanvas* menyimpan objek dengan *id* = 'canvas'. *Id* ini mengacu ke objek *canvas* pada HTML yang memiliki *id* bernama *canvas*. Variabel *myCanvas* sekarang sudah menyimpan objek *canvas*. Kemudian variabel *context* menyimpan *drawing context* 2D. Sesudah itu, *canvas* tersebut dapat digambar dengan bentuk 2D, garis, kurva, membuat tulisan, dan menambahkan gambar. Selain untuk menggambar, bentuk-bentuk yang sudah digambar dapat diberi warna sesuai dengan keinginan.

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



Gambar 2.3: Posisi kotak biru pada *canvas* terhadap *origin*[3]

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

Langkah pertama untuk membuat *path* baru adalah dengan menggunakan fungsi *beginPath()*. Setelah itu, perintah-perintah untuk menggambar dapat digunakan untuk membuat bentuk-bentuk yang diinginkan. Apabila sudah selesai menggambar, gunakan fungsi *stroke()* untuk menggambar *outline* dari *path* tersebut atau *fill()* untuk mengisi area *path* tersebut. Setelah itu, gunakan fungsi *closePath()* untuk menutup bentuk tersebut dengan cara menggambar garis lurus dari posisi titik terakhir ke titik awal. Fungsi lainnya yang menjadi bagian dari membuat *path* adalah fungsi *moveTo()*. Fungsi ini diibaratkan seperti mengangkat sebuah pensil dari sebuah titik pada kertas kemudian menempatkannya pada titik yang diinginkan. Listing 2.5 merupakan fungsi *moveTo()*.

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

Listing 2.5: Fungsi *moveTo()*

Fungsi *moveTo()* memiliki 2 parameter yaitu *x* dan *y* yang merupakan posisi titik pada *canvas*. Ketika *canvas* sudah diinisialisasi dan fungsi *beginPath()* sudah dipanggil, fungsi *moveTo()* berguna sebagai penempatan titik awal untuk menggambar. Fungsi *lineTo()* digunakan untuk menggambar sebuah garis. Listing 2.6 merupakan fungsi *lineTo()*.

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

Listing 2.6: Fungsi *lineTo()*

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

Fungsi *arc()* digunakan untuk menggambar lingkaran atau busur. Listing 2.7 merupakan fungsi *arc()*.

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

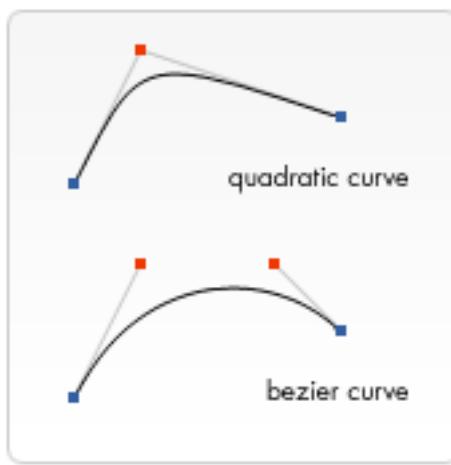
Listing 2.7: Fungsi *arc()*

Parameter *x* dan *y* adalah posisi titik tengah busur pada *canvas*. Radius adalah besar jari-jari busur. *StartAngle* dan *endAngle* adalah titik awal dan titik akhir busur dalam satuan radian yang diukur dari sumbu x. *Anticlockwise* adalah parameter yang bernilai *boolean*, apabila bernilai *true*, maka busur akan digambar berlawanan arah jarum jam dan jika bernilai *false*, busur akan digambar

searah jarum jam. Karena fungsi `arc()` menerima input sudut dalam radian, maka perlu dilakukan konversi dari satuan derajat menjadi radian terlebih dahulu. Rumusnya adalah sebagai berikut :

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

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*[3]

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).

2.4.2 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.

Kelas

Javascript tidak memiliki *statement* 'class' yang digunakan pada bahasa pemrograman C++ atau *Java*. Untuk membuat kelas, *Javascript* menggunakan *function* sebagai konstruktor untuk kelas. Karena itu, membuat kelas sama dengan membuat *function* pada *Javascript*. Pada *listing 2.8* terdapat potongan kode untuk membuat kelas bernama *Mobil*.

```
1| function Mobil(){  
2| }  
3| }
```

Listing 2.8: Membuat kelas *Mobil*

Objek

Untuk membuat instansi baru dari objek, gunakan *statement* 'new' yang nantinya akan disimpan pada variabel. Pada *listing 2.9* terdapat potongan kode untuk membuat instansi.

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

Listing 2.9: Membuat instansi mobil

Konstruktor

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

Properti/Atribut

Properti adalah variabel yang terdapat pada kelas. Properti ditulis pada konstruktor kelas sehingga setiap properti pada kelas akan dibuat ketika membuat instansi baru. Untuk membuat properti, gunakan *statement* 'this'. Cara ini mirip dengan bahasa pemrograman *Java* ketika membuat sebuah properti pada objek. Sintaks untuk mengakses properti di luar kelas adalah : *namaInstansi.properti*. Pada *listing 2.10* terdapat potongan kode untuk mendefinisikan properti pada kelas *Mobil* pada saat instansiasi.

```
1| function Mobil(merkMobil,beratMobil,warnaMobil,tahunProduksi){  
2|     this.merkMobil = merkMobil;  
3|     this.beratMobil = beratMobil; //satuan dalam kg  
4|     this.warnaMobil = warnaMobil;  
5|     this.tahunProduksi = tahunProduksi;  
6| }  
7|  
8| var mobil1 = new Mobil('Toyota',1000,'Hitam',2010);
```

Listing 2.10: Mendefinisikan properti kelas *Mobil* pada konstruktor

Method

Method adalah hal yang dapat dilakukan oleh sebuah objek. Untuk membuat *method*, tuliskan nama *method* terlebih dahulu kemudian *assign* fungsi pada nama *method* tersebut. Untuk memanggil *method* sebuah objek, tuliskan nama objek/kelas terlebih dahulu, kemudian tuliskan nama *method*

sesuai dengan yang sudah dibuat beserta tanda kurung. Tanda kurung berisi parameter. Pada *listing 2.11* terdapat potongan kode untuk membuat dan memanggil *method* bergerakMaju() pada kelas Mobil.

```

1  function Mobil(merkMobil,beratMobil,warnaMobil,tahunProduksi){
2      this.merkMobil = merkMobil;
3      this.beratMobil = beratMobil; //satuan dalam kg
4      this.warnaMobil = warnaMobil;
5      this.tahunProduksi = tahunProduksi;
6
7      this.bergerakMaju = function(){
8          //kode agar mobil bergerak maju
9      }
10 }
11
12 var mobil1 = new Mobil('Toyota',1000,'Hitam',2010);
13 mobil1.bergerakMaju(); //memanggil fungsi untuk bergerak maju
```

Listing 2.11: Membuat dan memanggil method bergerakMaju()

2.4.3 Event

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

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

Setiap *event* memiliki *event handler*, yang berisikan sekumpulan kode yang akan dijalankan ketika *event* sudah terjadi. *Event handler* juga sering disebut sebagai *event listener*. *Listener* menunggu *event* yang terjadi dan *handler* adalah kode yang dijalankan ketika *listener* mendapatkan *event*/ketika *event* terjadi. Untuk memperjelas cara menggunakan *event*, pada *listing 2.12* terdapat contoh kode untuk menambahkan event pada *button/tombol*.

```

1 <html>
2     <title>Event pada tombol</title>
3     <body>
4         <button id='tombol'>Change color</button>
5     </body>
6 </html>
```

```

7
8 <script>
9     var btn = document.getElementById('tombol');
10
11     function random(number) {
12         return Math.floor(Math.random()*(number+1));
13     }
14
15     btn.onclick = function() {
16         var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' +
17             random(255) + ')';
18         document.body.style.backgroundColor = rndCol;
19     }
</script>

```

Listing 2.12: Menambahkan event pada button

Berdasarkan listing 2.12, objek *button* dengan id='tombol' disimpan di dalam variabel bernama 'btn'. Ada fungsi bernama 'random' untuk mengembalikan sebuah nilai acak. Setelah itu ada *event handler*. *Event handler property* yang digunakan adalah *onclick*. *Event handler property onclick* mengecek apakah objek(dalam kasus ini objeknya adalah *button*) sudah ditekan. Bila tombol sudah ditekan, maka fungsi akan dieksekusi untuk mengubah warna *background*. Warna RGB tersebut digenerate secara acak menggunakan fungsi *random* yang sudah dibuat sebelumnya. Tidak hanya *event handler property onclick* saja yang dapat digunakan pada halaman *web*. Berikut ini adalah beberapa *event handler property* lainnya:

- *onfocus* dan *onblur* : aksi akan dijalankan apabila sebuah objek difokuskan/tidak. Biasanya digunakan untuk menampilkan informasi tentang cara mengisi *form* ketika difokuskan atau menampilkan pesan *error* ketika *form* tersebut diisi dengan nilai yang salah/tidak valid.
- *ondblclick* : 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 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 2.13 terdapat potongan kode untuk menggunakan fungsi *addEventListener()*.

```

1
2 var btn = document.getElementById('tombol');
3
4 function bgChange() {

```

```

5      var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random
6          (255) + ')';
7      document.body.style.backgroundColor = rndCol;
8
9  btn.addEventListener('click', bgChange);

```

Listing 2.13: 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 2.14*.

```

1
2  btn.addEventListener('click', function() {
3      var rndCol = 'rgb(' + random(255) + ',' + random(255) + ',' + random
4          (255) + ')';
5      document.body.style.backgroundColor = rndCol;
});
```

Listing 2.14: 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 mengintepretasi aktivitas jari atau *stylus* pada permukaan layar seperti layar sentuh atau *trackpads*. *Interface* pada *touch event* merupakan API tingkat rendah yang dapat digunakan untuk aplikasi yang mendukung interaksi *multi-touch* seperti *2-finger gesture*. Interaksi *multi touch* dimulai ketika sebuah jari menyentuh permukaan layar sentuh terlebih dahulu. Jari yang lain dapat menyentuh permukaan dan dapat menggerakkan jari di sekitar permukaan layar sentuh. Interaksi akan berakhir ketika jari tidak lagi menyentuh permukaan layar.

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

Touch event memiliki beberapa event handler properti diantaranya adalah :

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

2.4.4 Membuat Animasi

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

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

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

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

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

2.4.5 Fungsi *Callback*

Fungsi *callback* adalah sebuah fungsi yang digunakan pada fungsi lain sebagai argumen yang nantinya akan dipanggil di dalam fungsi luar untuk menyelesaikan sebuah aksi. Listing 2.15 merupakan contoh potongan kode pada fungsi *callback*.

```

1  function greeting(name) {
2      alert('Hello ' + name);
3  }
4
5
6  function processUserInput(callback) {
7      var name = prompt('Please enter your name.');
8      callback(name);
9
10 }
11 processUserInput(greeting);

```

Listing 2.15: Contoh fungsi *callback*

Pada listing 2.15, fungsi *processUserInput()* menggunakan fungsi *greeting()* sebagai fungsi *callback*. Setelah *processUserInput* selesai dieksekusi, maka fungsi *greeting()* akan dijalankan. Secara umum, fungsi *callback* digunakan untuk melanjutkan eksekusi kode program setelah operasi *asynchronous* selesai dieksekusi. Dengan menggunakan fungsi *callback*, jika sebuah fungsi belum selesai dieksekusi, maka fungsi *callback* tidak akan dijalankan.

2.5 *jQuery*

jQuery merupakan sebuah *file Javascript* yang sudah termasuk pada halaman *web* [4]. *jQuery* dapat memilih elemen pada halaman *web* menggunakan *CSS-style selector* dan elemen tersebut dapat melakukan sesuatu dengan menggunakan *method jQuery*. Sebuah fungsi yaitu *jQuery()* digunakan untuk menemukan satu atau lebih elemen yang berada pada halaman *web*. Fungsi ini membuat objek yang bernama *jQuery* untuk menyimpan referensi dari elemen yang akan dipilih. *\$()* sering digunakan sebagai pengganti fungsi *jQuery()* dikarenakan penulisannya yang pendek. Fungsi *jQuery()* hanya memiliki sebuah parameter yaitu sebuah *selector*.

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

Listing 2.16: Mendapatkan elemen menggunakan *CSS-style selector*

Pada *listing 2.16*, *selector* akan mencari elemen ** yaitu *list* yang merupakan bagian dari kelas 'hot'. *jQuery* memiliki banyak *method* yang dapat digunakan oleh elemen yang sudah dipilih menggunakan *selector*. *Method* ini merepresentasikan tugas yang akan dilakukan oleh elemen tersebut. Setelah memilih elemen, tambahkan *method* yang diawali dengan titik kemudian diikuti dengan nama *method* beserta parameternya. Titik ini disebut sebagai *member operator*. Setiap *method* memiliki parameter untuk memberikan detail tentang bagaimana cara untuk mengubah elemen tersebut. Ada beberapa *method* yang memiliki parameter lebih dari 1. Member operator menunjukkan bahwa *method* yang terletak setelah *member operator* digunakan untuk mengubah elemen objek *jQuery* yang terletak pada sebelah kiri *member operator*. Pada *listing 2.17*, terdapat contoh untuk mengubah kelas dari elemen yang sudah dipilih. Method *addClass* digunakan untuk mengubah atribut kelas dari elemen *list* menjadi kelas yang bernama 'complete'.

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

Listing 2.17: Mengubah kelas dari elemen yang sudah dipilih

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

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

Jika ingin menggunakan *selection* yang sama, maka lebih baik menggunakan objek *jQuery* yang sama daripada mengulang langkah-langkah yang sudah dijelaskan. Objek *jQuery* tersebut dapat disimpan pada sebuah variabel. Cara untuk menyimpan referensi objek *jQuery* tersebut pada variabel adalah membuat variabel yang diawali dengan simbol '\$'. Kemudian variabel tersebut diisi dengan objek *jQuery* yang diinginkan. Pada *listing 2.18*, variabel *\$listItems* menyimpan objek *jQuery* yang berisi lokasi dari semua elemen list ada pada DOM *tree*.

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

Listing 2.18: Menyimpan objek *jQuery*

Untuk mengecek apakah sebuah halaman sudah siap untuk menjalankan kode program yang dibuat, maka gunakan *method ready()*. Maksud dari halaman sudah siap adalah DOM sudah ada pada halaman *web*. Listing 2.19 adalah potongan kode untuk mengecek apakah halaman sudah siap. Pada listing 2.19, *\$(document)* merepresentasikan halaman *web*. Jika halaman sudah siap, maka

kode program yang ada di dalam *method ready()* akan dijalankan. Listing 2.20 adalah pintasan dari *method ready()* pada objek *document*.

```
1 $(document).ready(function(){
2     //ketikan script di sini
3});
```

Listing 2.19: Mengecek apakah halaman sudah siap

```
1 $(function(){
2     //ketikan script di sini
3});
```

Listing 2.20: Pintasan dari method \$(document).ready()

2.5.1 *Looping*

Pada *jQuery* dapat dilakukan *looping* untuk mendapatkan informasi dari setiap elemen atau untuk memberikan aksi pada setiap elemen. *Method* yang digunakan untuk looping adalah *each()*. *Method* ini akan memberikan sebuah atau lebih aksi pada setiap elemen. *Method* ini memiliki sebuah parameter yaitu sebuah fungsi yang isinya adalah perintah-perintah yang akan dijalankan oleh setiap elemen. Contohnya terdapat pada listing 2.21. Pada listing 2.21 akan dipilih elemen *list*. *Method .each()* akan menjalankan kode program yang sama untuk setiap elemen *list* tersebut. '*this.id*' mengacu kepada id milik sebuah elemen *list* yang sekarang berada dalam *loop*. Kemudian variabel yang bernama *ids* akan menyimpan id setiap elemen *list*. *\$(this)* digunakan untuk membuat sebuah objek *jQuery* baru yang isinya adalah sebuah elemen yang ada sekarang. *\$(this)* memungkinkan kita untuk menggunakan *method* pada elemen yang ada sekarang. Elemen *list* yang ada pada *loop* akan ditambahkan sebuah elemen *span* yang isinya adalah id dari elemen tersebut. Perintah ini akan dilakukan untuk setiap elemen *list*.

```
1 $('li').each(function(){
2     var ids = this.id;
3     $(this).append('<span class="order">' +ids+ '</span>');
4});
```

Listing 2.21: Menambah setiap elemen *list* dengan id *list* masing-masing

2.5.2 *Chaining*

Jika ingin menggunakan *method* *jQuery* lebih dari 1 pada elemen yang sama, *method* tersebut dapat dituliskan secara bersamaan. Setiap *method* yang digunakan harus dipisahkan dengan titik. Pada listing 2.22 terdapat potongan kode untuk menggunakan beberapa *method* pada elemen yang sama.

```
1 $('li[id!="one"]').hide().delay(500).fadeIn(1400);
```

Listing 2.22: Menggunakan beberapa *method* pada elemen yang sama

Pada listing 2.22, ada 3 *method* yaitu *hide()*, *delay()* dan *fadeIn()* digunakan pada sebuah elemen yang sama. *Method hide()* digunakan untuk menyembunyikan elemen, *method delay()* digunakan untuk menjeda waktu/*pause*, dan *method fadeIn()* digunakan untuk memudarkan elemen. Proses menambahkan beberapa *method* pada *selector* yang sama disebut sebagai *chaining*. *Method-method* yang digunakan untuk memperbarui elemen dapat menggunakan *chaining* sedangkan *method* yang digunakan untuk mendapatkan informasi dari elemen tidak dapat menggunakan *chaining*.

2.5.3 Mendapatkan dan Mengubah Konten Elemen

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

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

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

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

2.5.4 Mendapatkan dan Mengubah Properti CSS

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

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

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

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

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

Listing 2.24: Mengubah warna background color semua elemen list

```
1| $('li').css({
2|   'background-color': 'red',
3|   'font-family': 'Courier'
4| });
```

Listing 2.25: Mengubah warna background color dan jenis font untuk semua elemen list menggunakan *object literal notation*

2.5.5 Event

Sama seperti *Javascript*, pada *jQuery* juga dapat ditambahkan *event*. *Method* yang digunakan untuk menambahkan *event* adalah *on()*. Untuk memperjelas cara menggunakan *method on()*, terdapat potongan kode pada *listing 2.23*. Untuk menambahkan *event*, maka pertama harus memilih elemen yang akan ditambahkan *event* dengan menggunakan *selector*. Pada *listing 2.26*, elemen yang dipilih adalah semua elemen *list*, kemudian tambahkan *method on()*. *Method .on()* memiliki 2 parameter yaitu *event* yang akan digunakan dan perintah yang akan dilakukan apabila *event* tersebut terjadi pada elemen yang dipilih. Perintah dapat berupa sebuah fungsi anonim yaitu fungsi yang dibuat langsung atau memanggil sebuah fungsi yang sudah ada. Pada *listing 2.26*, *event* yang digunakan adalah '*click*' dan akan diberikan sebuah fungsi anonim yang tugasnya adalah menambahkan atribut *class* yang bernilai '*complete*'.

```
1| $('li').on('click', function(){
2|   $(this).addClass('complete');
3| });
```

Listing 2.26: Menambahkan atribut *class* pada setiap *list* menggunakan event '*click*'

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

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

Setiap fungsi *event handling* menerima sebuah *event object*. *Event object* memiliki properti dan *method* yang berhubungan dengan *event* yang sudah terjadi. Contoh kode program dapat dilihat pada *listing 2.27*. Pada *listing 2.27*, pada parameter *function* terdapat parameter yang bernama *event*. Parameter yang bernama *event* ini adalah *event object*. Kemudian tipe dari *event object* tersebut disimpan pada variabel yang bernama *eventType*.

```
1| $('li').on('click', function(event){
2|   eventType = event.type;
3| });
```

Listing 2.27: Mendapatkan tipe *event* dari *event object*

Event object memiliki 7 properti, diantaranya adalah:

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

2.5.6 Mengubah dan Mendapatkan Dimensi dari Elemen

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

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

Berikut adalah method yang digunakan untuk mendapatkan dimensi kotak :

- *innerHeight()* : tinggi dari kotak ditambah *padding*
- *innerWidth()* : lebar dari kotak ditambah *padding*
- *outerHeight()* : tinggi dari kotak ditambah *padding* dan *border*
- *outerWidth()* : lebar dari kotak ditambah *padding* dan *border*
- *innerHeight(true)* : tinggi dari kotak ditambah *padding, border* dan *margin*
- *innerWidth(true)* : lebar dari kotak ditambah *padding, border* dan *padding*

2.6 AJAX

AJAX(*Asynchronous Javascript and XML*) adalah sebuah teknik pengembangan *web* yang digunakan untuk memuat data pada bagian halaman *web* tanpa memuat ulang/*refresh* halaman *web* [4]. Penggunaan AJAX yang umum adalah sebagai berikut:

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

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

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

2.6.1 Ajax Request dan Response

Untuk membuat *Ajax request*, *browser* menggunakan objek *XMLHttpRequest*. Ketika server merespon *request* dari browser, objek *XMLHttpRequest* yang sama akan memproses hasilnya. Pada *listing 2.28* terdapat potongan kode untuk membuat *Ajax request*.

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

Listing 2.28: Membuat *Ajax request*

Berdasarkan *listing 2.28*, hal pertama yang dilakukan adalah membuat objek *XMLHttpRequest* yang disimpan pada variabel bernama *xhr*. Kemudian *method open()* berfungsi untuk menyiapkan *request*. *Method open* memiliki 3 parameter yaitu HTTP *method*, *url* dari halaman yang akan menangani *request*, dan tipe data *boolean* yang menentukan apakah *request* tersebut *asynchronous* atau tidak. Pada *listing 2.28* *request* tersebut menggunakan HTTP *method* yaitu *GET*, *url* halamannya adalah '*data/test.json*', dan *asynchronous*. *Method .send()* digunakan untuk mengirimkan *request* yang sudah disiapkan. Informasi tambahan dapat dikirimkan ke server yang dituliskan pada parameter *method .send()*. Sesudah mengirimkan *request* dan menerima respon dari server, data yang diterima akan diproses. Pada *listing 2.29* terdapat potongan kode untuk menerima respon dan memproses data dari server. Setelah *browser* menerima dan memuat respon dari server, event *onload* akan dijalankan dan akan *trigger* sebuah fungsi. Di dalam fungsi tersebut, akan dicek status dari objek tersebut untuk memastikan apakah respon dari server tidak ada masalah.

```
1 xhr.onload = function(){
2     if(xhr.status === 200){
3         //proses data yang sudah diterima dari server
4     }
5 }
```

Listing 2.29: Memproses respon yang didapat dari server

2.6.2 *jQuery* dan *AJAX Request*

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

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

2.6.3 *jQuery* dan *AJAX Response*

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

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

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

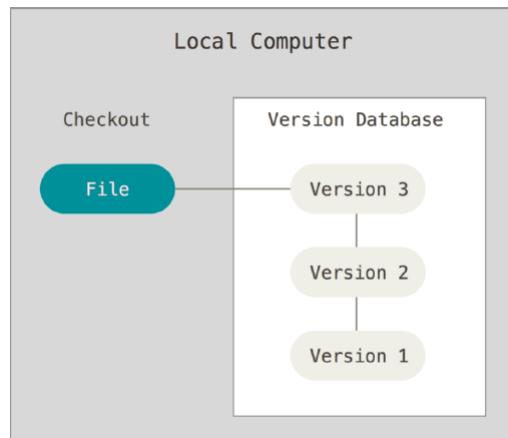
- *type* : mendapatkan nilai *GET* dan *POST* tergantung dari *request*. *Request* tersebut dapat dibuat menggunakan *HTTP GET* atau *POST*.
- *url* : *request* yang akan dikirimkan
- *data* : data yang akan dikirimkan ke server bersama dengan *request*
- *success* : sebuah fungsi yang akan dijalankan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

2.7 Version Control

Version control adalah sistem yang menyimpan perubahan pada sebuah *file* atau sekumpulan *file* secara berkala sehingga dapat mendapatkan versi yang spesifik nantinya [5]. VCS(*Version Control System*) memungkinkan pengguna untuk mengembalikan *file* yang diinginkan ke *state* sebelumnya, mengembalikan keseluruhan proyek ke *state* sebelumnya, membandingkan perubahan secara berkala, dapat melihat pengguna terakhir yang memodifikasi sesuatu yang menyebabkan masalah, dan masih banyak lagi. Ketika beberapa *file* ada yang hilang karena sebuah kesalahan, *file-file* tersebut dapat dikembalikan dengan mudah.

Local Version Control System

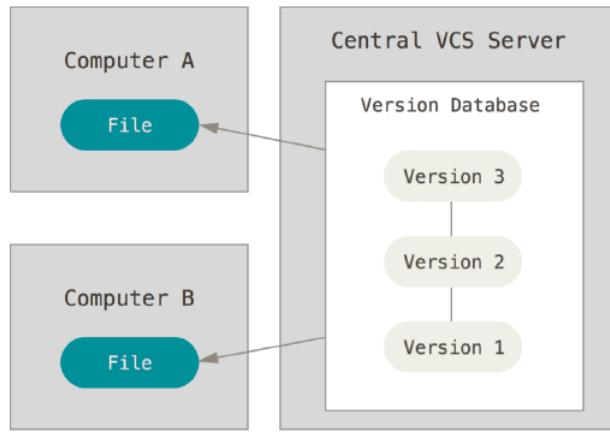
Local Version Control System memiliki sebuah basis data yang menyimpan semua perubahan pada *file* dalam *revision control*. Salah satu VCS tools yang cukup terkenal adalah RCS yang masih digunakan oleh banyak komputer hingga sekarang. Cara kerja RCS adalah menyimpan *patch sets* yang merupakan perbedaan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



Gambar 2.5: Local Version Control

Centralized Version Control System

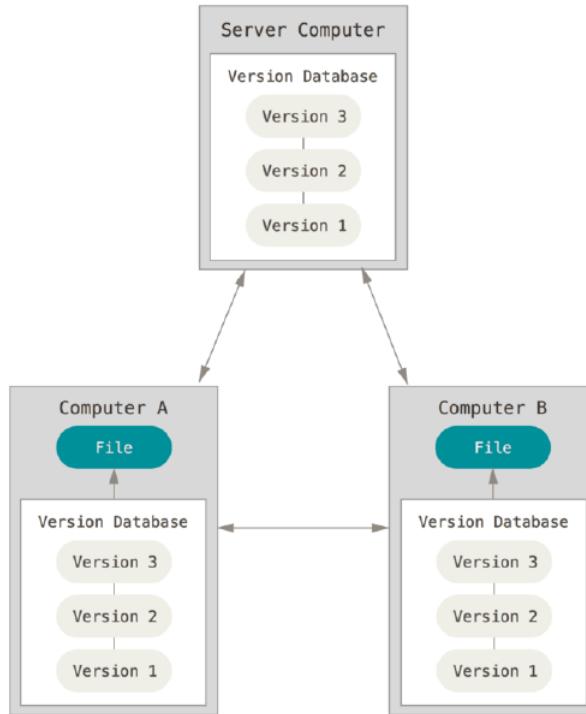
Local Version Control System menjadi kurang efektif, bila ada beberapa orang yang berkolaborasi dengan pengembang. Karena pada *Local Version Control System*, *version control* dimiliki oleh masing-masing komputer sehingga pengguna tidak tahu apakah *file* tersebut sudah diubah oleh kolaborator lain.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



Gambar 2.6: Centralized Version Control

Distributed Version Control System

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



Gambar 2.7: Distributed Version Control

2.7.1 *Git*

Git merupakan sebuah *version control* namun berbeda dengan VCS lainnya dilihat dari cara menyimpan datanya. Sistem seperti CVS, *Subversion*, *Perforce*, *Bazaar* menyimpan data sebagai sekumpulan *file* dan perubahan setiap *file* disimpan setiap waktu. Pada *Git*, data tersebut dianggap sebagai sekumpulan *snapshot* dari *miniature filesystem*. Setiap *commit* atau menyimpan proyek, *Git* seolah-olah mengambil gambar untuk melihat seperti apa *file* yang terlihat pada saat itu dan

menyimpannya sebagai referensi pada *snapshot* tersebut. Singkatnya, apabila tidak ada *file* yang diubah, Git tidak akan menyimpan *file* lagi.

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

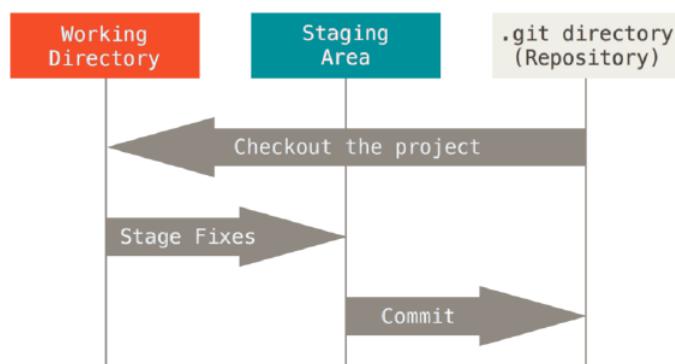
Git memiliki 3 *state* utama pada *file* yaitu:

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

Terdapat 3 bagian utama dalam proyek *Git* yaitu :

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

Gambar 2.8 di bawah ini 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 *checkout* tetapi belum dalam *state staged*, maka *file* tersebut dalam *state modified*.

Ada beberapa cara dalam menggunakan *Git* yaitu dengan menggunakan *command-line* dan beberapa *GUI(Graphical User Interface)* yang memiliki kemampuan yang 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.

Mendapatkan *Git Repository*

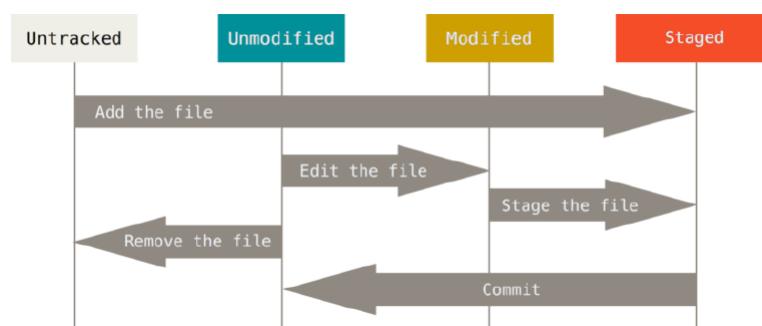
Untuk mendapatkan *Git repository* ada 2 cara yaitu : menjadikan sebuah proyek yang terdapat pada direktori lokal yang belum dalam *version control* lalu menjadikannya sebagai *Git repository* dan dengan *clone Git repository* yang sudah ada.

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

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

Record Perubahan pada *Repository*

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



Gambar 2.9: Siklus hidup pada status file

Perintah `git status` digunakan untuk mengecek status *file*. Jika mengetik perintah sesudah

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

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

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

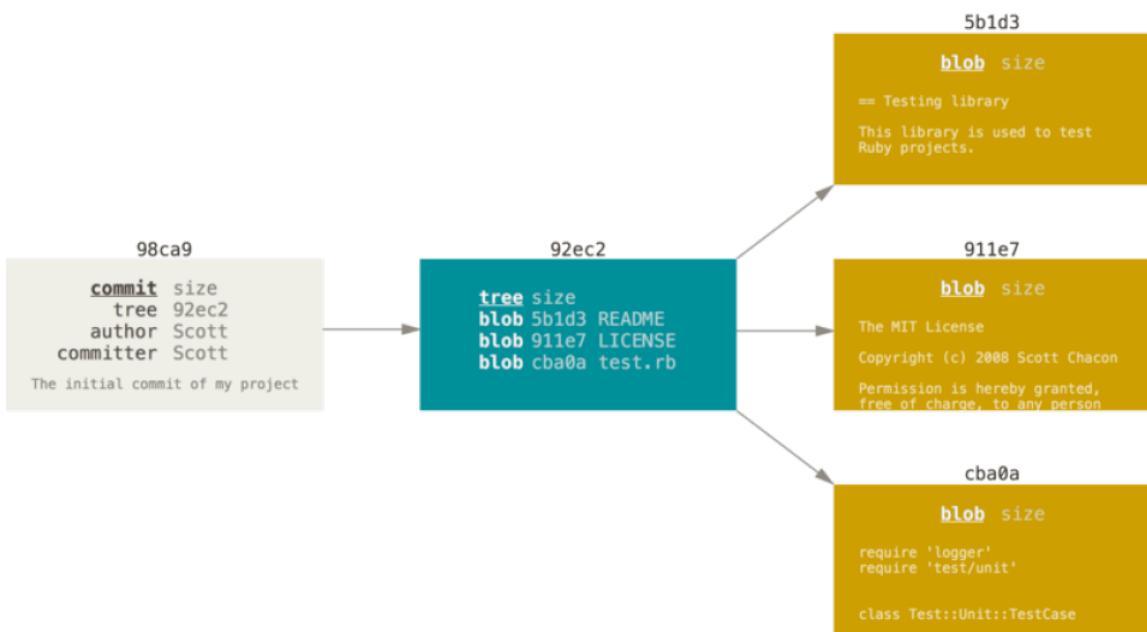
Glob pattern adalah *regular expression* yang digunakan oleh *shells*. Tanda bintang(*) untuk nol atau beberapa karakter, [abc] untuk karakter apa saja yang berada di dalam kurung siku, tanda tanya(?) untuk sebuah karakter apa saja dan tanda kurung siku dengan tanda strip(-) untuk karakter antara sebuah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

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

2.7.2 Git Branching

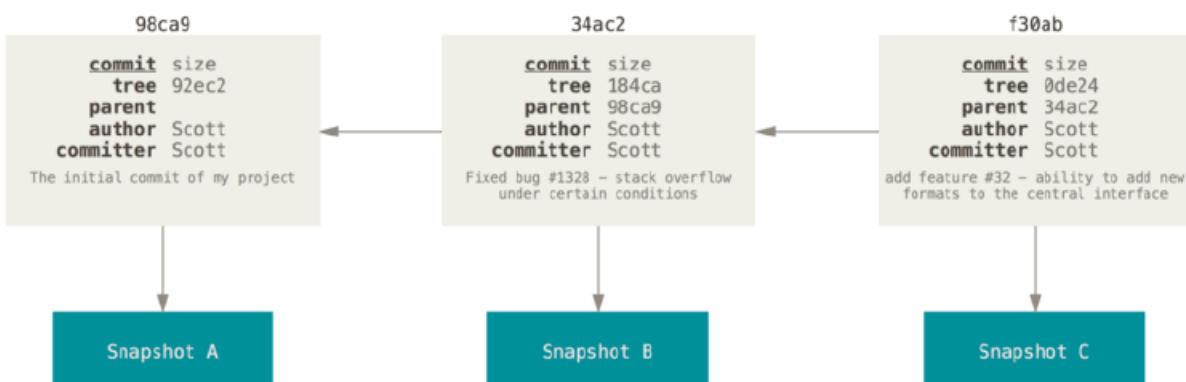
Branching artinya membuat dan mengerjakan sebuah proyek di tempat yang berbeda namun masih dalam repository yang sama sehingga tidak mengubah proyek utama. Ketika *commit*, *Git* menyimpan objek *commit* yang memiliki sebuah *pointer* pada *snapshot* sebuah konten yang sudah dalam state *staged*. Objek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

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



Gambar 2.10: Commit dan tree dari file yang dicommit

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



Gambar 2.11: Commit dan parent dari commit

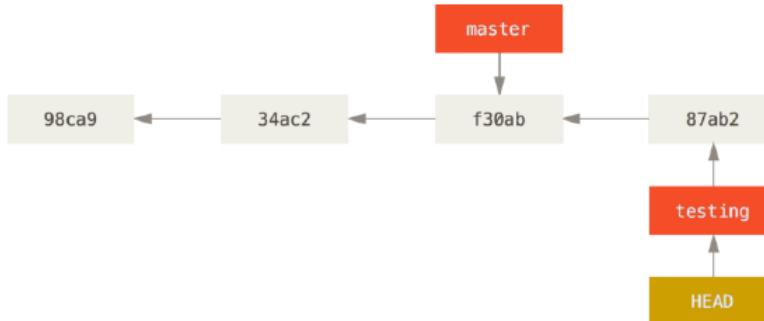
Nama *branch* pada *Git* awalnya disebut *master*. Ketika *commit*, pengguna diberikan *branch* *master* yang menunjuk pada *file* yang dicommit terakhir. Setiap *commit*, pointer pada *branch* *master* akan terus maju secara otomatis.

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

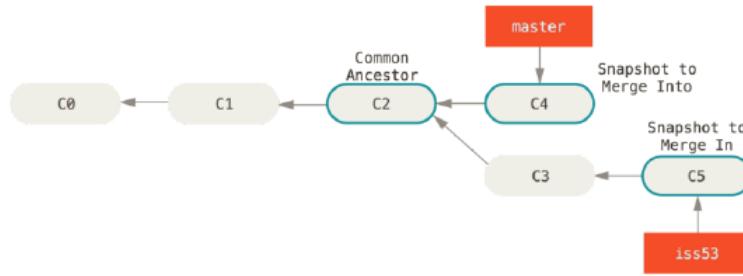


Gambar 2.13: Pointer HEAD beserta branch testing

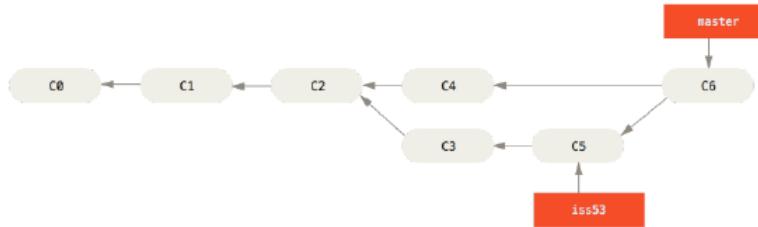
Perintah *git checkout* tidak hanya sebatas untuk pindah ke *branch* yang diinginkan. File yang ada pada *working directory* akan diubah dengan file yang ada pada *branch* tersebut. Bila berpindah ke *branch* sebelumnya, maka file dalam *working directory* akan dikembalikan sesuai dengan *commit* terakhir 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.

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

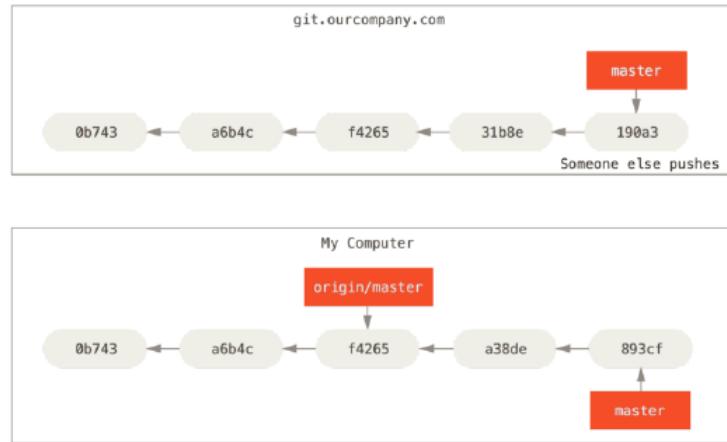
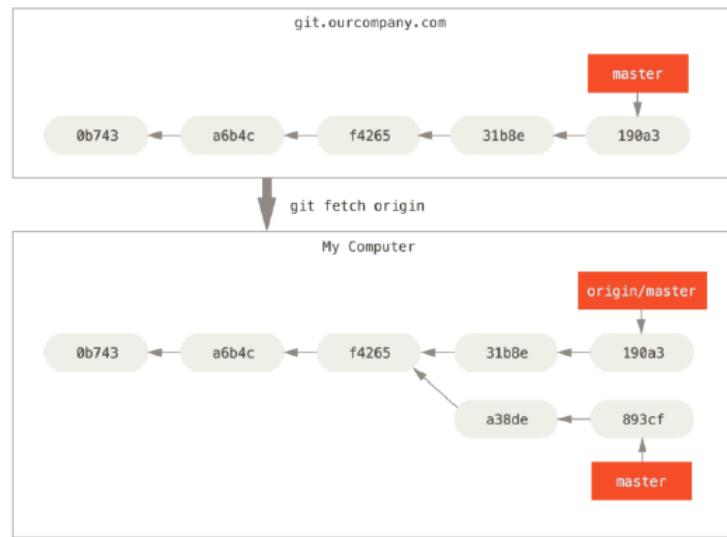


Gambar 2.15: *Merge commit*

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

Remote Branches

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

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

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

Check out *branch* lokal dari *remote-tracking branch* secara otomatis akan membuat *tracking branch*. *Tracking branch* adalah *branch* lokal yang memiliki hubungan langsung dengan *branch* *remote*. Jika berada pada *tracking branch* dan mengetikan perintah *git pull*, secara otomatis *Git* mengetahui *server* mana yang akan di-*fetch* dan *branch* apa yang akan di-*merge*. Bila *clone repository*, maka secara otomatis akan membuat sebuah *branch* yang bernama *master* yang men-*track* *origin/master*. Untuk mengatur *tracking branch*, perintah yang digunakan adalah *git checkout -b <branch> <remote>/<branch>*. *Git* menyediakan perintah *git checkout -track <remote>/<branch>* sebagai shortcut dari perintah *checkout* sebelumnya. Perintah *git checkout* juga dapat digunakan untuk mengatur *branch* lokal dengan nama yang berbeda dari *branch* *remote*. Jika sudah memiliki *branch* lokal dan ingin mengatur *branch* tersebut ke *branch* *remote* yang sudah di-*pull*, gunakan opsi *-u* atau *-set-upstream-to* pada perintah *git branch*. Untuk melihat *tracking branch* yang sudah diatur, gunakan opsi *-vv* pada perintah *git branch*. Perintah ini akan menampilkan *list* dari *branch* lokal dengan informasi tambahan mengenai *tracking* pada setiap *branch* dan apakah *branch* lokal

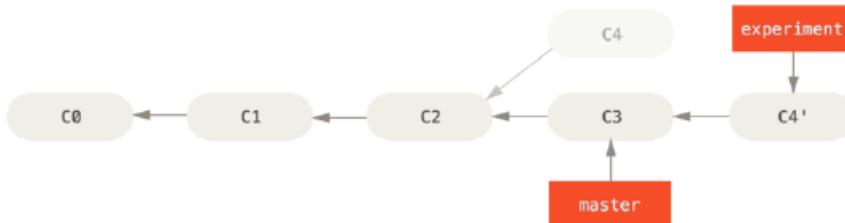
tersebut memiliki *ahead*, *behind* atau keduanya. *Ahead* adalah ada *commit* lokal yang belum di-*push* ke *server*, sedangkan *behind* adalah *commit* yang belum digabungkan. Perintah ini tidak langsung mengambil datanya dari *server* tetapi data tersebut merupakan data saat terakhir *fetch* dari *server*. Untuk mendapatkan data yang terbaru, harus *fetch* dari semua *remote* kemudian mengetikan perintah *git branch -vv*.

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

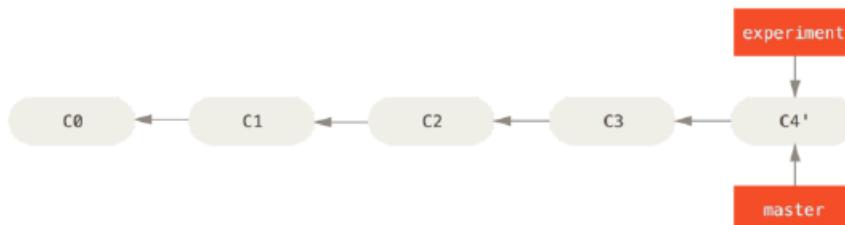
Branch pada *remote* dapat dihapus dengan menggunakan opsi *-delete* pada perintah *git push*. *Branch* pada *remote* tidak sepenuhnya dihapus, tetapi hanya *pointer*nya saja yang dihilangkan. Jika *branch* tidak sengaja terhapus, maka data pada *branch* dapat dikembalikan/di*back-up*.

Rebasing

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



Gambar 2.18: *Rebasing commit C4 ke C3*



Gambar 2.19: *Merge branch setelah rebasing*

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

2.7.3 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.
4. *Push branch* ke proyek *GitHub*.
5. Buka *Pull Request* di *GitHub*.
6. Diskusikan dan *commit* proyek tersebut apabila proyek tersebut masih membutuhkan perbaikan.
7. Pemilik proyek *merges/menggabungkan* atau menutup *Pull Request*.

Pull Request

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

Setelah kontributor sudah membuat *Pull Request*, pemilik proyek dapat melihat saran perubahan proyek dari orang lain dan memberikan komentar/keterangan pada perubahan tersebut. Pemilik proyek dapat melihat perbedaan pada kode pemilik proyek dengan perubahan yang disarankan tersebut dan pemilik proyek dapat mengomentari baris pada kode tersebut. Orang lain dapat

memberikan komentar pada *Pull Request*. Sesudah pemilik proyek memberikan keterangan tentang perubahan tersebut, kontributor menjadi tahu apa yang harus dilakukan agar perubahan tersebut dapat disetujui. Apabila perubahan tersebut membuat pemilik proyek puas, pemilik proyek akan *merge* perubahan tersebut dengan proyek aslinya dan otomatis akan menutup *Pull Request*.

2.8 Teknik Tabrakan

Teknik tabrakan digunakan untuk mengecek apakah 2 buah objek saling bertabrakan atau tidak. Terdapat 2 teknik tabrakan yaitu teknik tabrakan dengan pendektsian pixel dan teknik tabrakan dengan pendektsian vektor. Teknik tabrakan dengan pendektsian pixel merupakan cara pengecekan tabrakan antara 2 buah objek pada setiap frame. Contoh yang cukup simpel adalah pengecekan tabrakan antara 2 buah persegi panjang. Teknik ini kurang cocok untuk apabila digunakan untuk mengecek tabrakan antara 2 buah objek yang bergerak dengan kecepatan tinggi dan ukuran objek yang kecil. Teknik tabrakan dengan pendektsian vektor merupakan cara mengecek tabrakan antara 2 buah objek dengan menelusuri lompatan objek diantara 2 buah frame secara per-pixel.

BAB 3

ANALISIS

3.1 Analisis Permainan *Snake* yang Sudah Ada

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

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

3.1.1 Ular dan Makanan

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

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



Gambar 3.1: Ular pada *Slither.io*



Gambar 3.2: Makanan pada *Slither.io*

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



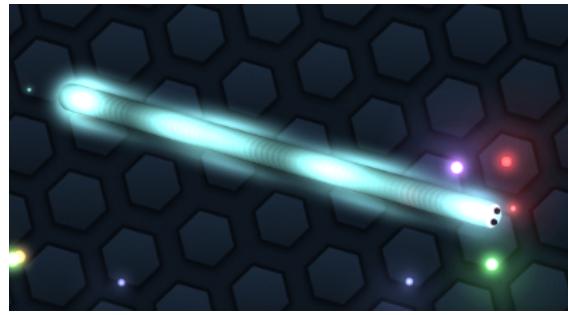
Gambar 3.3: Ular pada *Snake Nokia*



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

3.1.2 Pergerakan Ular

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

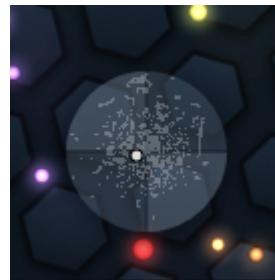


Gambar 3.5: Ular sedang melaju dengan cepat (*speed up*)

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

3.1.3 Labirin

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



Gambar 3.6: Peta labirin pada *Slither.io*

Labirin pada *Snake Nokia* lebih bervariasi dibandingkan dengan *Slither.io*. Pada permainan ini pemain dapat memilih labirin yang tersedia.

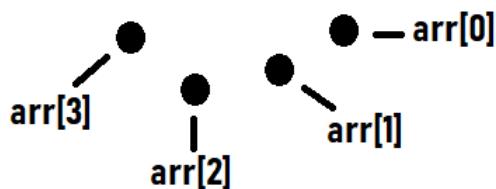
3.2 Analisis Sistem yang Dibangun

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

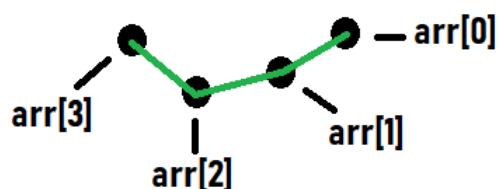
3.2.1 Menggambar Ular dan Apel

Tubuh ular dibuat menggunakan sekumpulan *line/garis pendek*. Setiap bagian tubuh ular memiliki panjang sebesar 2 *pixel* dan lebar tubuhnya sebesar 10 *pixel*. Panjang setiap bagian tubuh ular tidak 1 pixel karena akan membuat ular terlihat sangat pendek. Bagian tubuh ular dibuat pendek untuk memudahkan pengecekan jika terjadi ular menabrak tubuhnya sendiri. Setiap bagian tubuh ular memiliki koordinat masing-masing. Koordinat setiap bagian tubuh disimpan pada sebuah *array* agar

menggambar ular menjadi lebih mudah. Dalam tahap ini, tubuh ular masih berupa sekumpulan titik-titik yang merupakan koordinat bagian tubuh ular seperti pada Gambar 3.7. Algoritma untuk menggambar ular adalah dengan mengambil koordinat bagian tubuh ular mulai dari elemen *array* paling pertama(*arr[0]*) dan elemen *array* selanjutnya(*arr[1]*) lalu buat garis yang *start pointnya* adalah elemen pertama(*arr[0]*) dan *end pointnya* adalah elemen *array* kedua(*arr[1]*). Setelah itu ambil koordinat elemen *array* yang merupakan *end point* pada garis sebelumnya(*arr[1]*) dengan elemen *array* selanjutnya(*arr[2]*) dan gambar garisnya. Lakukan hal tersebut sampai *end point* garis mencapai elemen *array* paling akhir. Setelah digambar maka ular akan terlihat seperti Gambar 3.8.

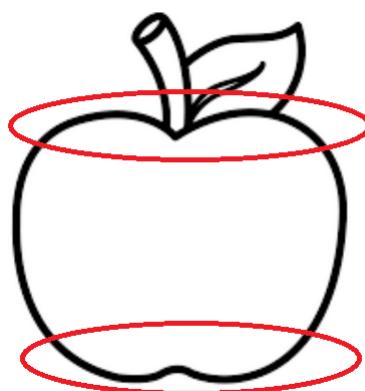


Gambar 3.7: Koordinat bagian tubuh ular pada *array*



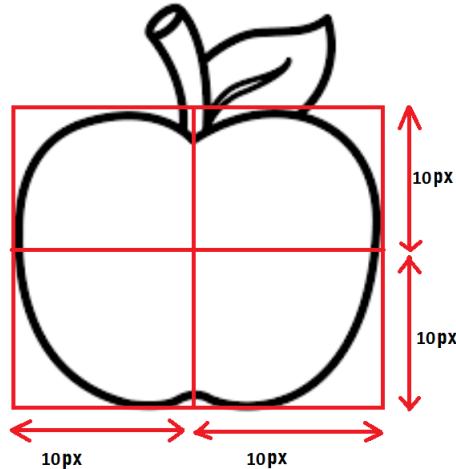
Gambar 3.8: Tubuh ular setelah digambar menggunakan garis

Untuk membuat apel digunakan *quadratic Bézier curve*. Kurva ini digunakan untuk membuat bagian-bagian apel yang melengkung. Bagian tersebut ditandai dengan lingkaran bewarna merah seperti yang ditunjukkan pada Gambar 3.9(gambar diambil dari pinterest. Link:<https://www.pinterest.com/pin/690317449105509454/>.



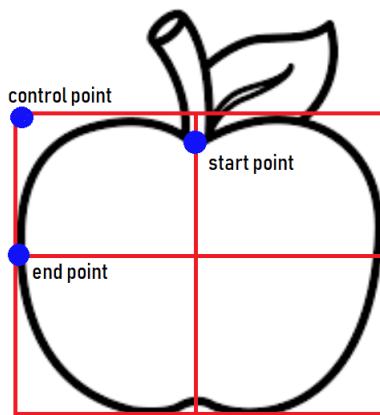
Gambar 3.9: Bagian pada apel(lingkaran merah) yang akan dibuat menggunakan kurva

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

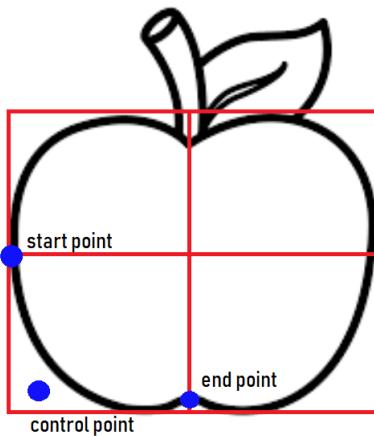


Gambar 3.10: Pembagian gambar apel dengan layout persegi beserta ukuran pada setiap bagian

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



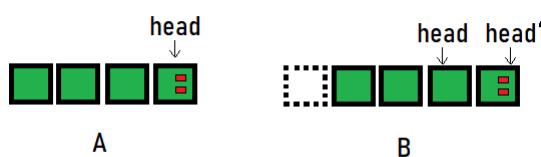
Gambar 3.11: *Start point*, *control point* dan *end point* untuk menggambar apel bagian kiri atas



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

3.2.2 Pergerakan Ular

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



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

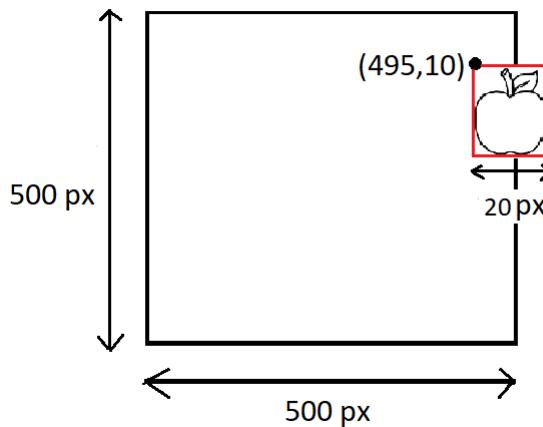
Kecepatan ular pada permainan ini adalah 2,4,6,8 dan 10 *pixel per frame*. Kecepatan ular

minimal adalah 1 dan maksimum adalah 5. Kecepatan maksimal ular tidak boleh melebihi lebar tubuh ular. Jika 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. Sudut tersebut akan ditambah atau dikurangi sebanyak kecepatan berbelok yang dipilih oleh pemain. Kecepatan berbelok minimal adalah 1° dan maksimal adalah 10°. Misal, jika pemain memilih kecepatan berbelok sebesar 1°, maka ketika tombol kiri ditekan 1 kali maka sudut akan dikurangi sebanyak 1° dan jika tombol kiri ditekan 10 kali, maka sudut akan dikurangi sebesar 10°. 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. Cara untuk menghitung posisi koordinat x adalah posisi koordinat x ular sekarang ditambah dengan *cosinus* dari besar sudut. Cara untuk menghitung posisi koordinat y adalah posisi koordinat y ular sekarang ditambah dengan *sinus* dari besar 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()* akan dikalikan dengan lebar *canvas* untuk mendapatkan nilai x dan dikalikan dengan tinggi *canvas* untuk mendapatkan nilai y. Karena apel ini dibuat dengan menggunakan *layout* persegi, maka posisi x dan y pada apel terletak di titik sudut kiri atas. Hal ini akan memungkinkan gambar apel akan terpotong seperti yang terlihat pada Gambar 3.14. Misal, besar *canvas* adalah 500 x 500 dan besar apel adalah 10 dan mendapatkan posisi apel adalah (495,10). Posisi x apel ditambah dengan besar apel hasilnya akan melebihi besar *canvas* sehingga membuat sebagian gambar apel terlihat terpotong. Maka dari itu, lebar dan tinggi *canvas* yang dikalikan dengan bilangan acak, akan dikurangi sebesar ukuran apel tersebut. Nilai yang dihasilkan adalah nilai yang bertipe *float* sedangkan posisi x dan y pada apel membutuhkan *input* bilangan bulat. Untuk mendapatkan bilangan bulat tersebut, nilai yang sudah dihitung tadi dibulatkan ke bawah. Mengacak posisi apel tidak hanya mengacak posisi pada *canvas* saja, tetapi harus mengecek apakah posisi apel tersebut tidak bertabrakan dengan tubuh ular atau dinding labirin.



Gambar 3.14: Gambar apel yang terpotong sesudah mengacak posisi apel

3.2.4 Menentukan Besar *Canvas*

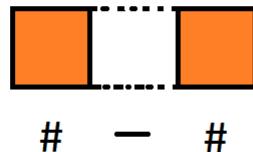
Pada Open Source Snake 360 ini, pemain dapat memainkan permainan tersebut di browser *smartphone* dan *browser desktop*. Hal ini akan memunculkan sebuah kesulitan yaitu menentukan dimensi *canvas* yang cocok bila permainan tersebut dapat dimainkan pada *smartphone* dan *browser* pada *desktop*. Jika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

3.2.5 Membuat Labirin

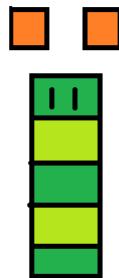
Pada permainan ini, format labirin dapat dibuat dengan menggunakan JSON dan *file* teks. Permainan ini menggunakan *file* teks sebagai format labirin, karena *file* text lebih mudah untuk dibuat dan dimengerti oleh pembuat labirin. Labirin dibuat menggunakan simbol '#' dan '-'. Setiap simbol merepresentasikan 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.

Terdapat beberapa hal yang perlu diperhatikan dalam pembuatan labirin. Nama *file* labirin harus diawali dengan ‘maze’ kemudian diikuti oleh sebuah angka. Jika menambahkan labirin terbaru, maka angka pada nama *file* labirin akan digunakan sebagai patokan. Contoh, jika labirin terakhir bernama ‘maze3.txt’, maka labirin terbaru yang akan ditambahkan harus bernama ‘maze4.txt’. *File* labirin tersebut harus disimpan pada folder bernama ‘Levels’. Baris terakhir dari *file* labirin diisi dengan posisi awal ular saat permainan dimulai. Posisi ular terdiri dari koordinat X dan koordinat Y dalam satuan *pixel* mulai dari 0 sampai 600 *pixel*. Kedua koordinat tersebut dipisahkan oleh spasi. Contoh, jika posisi ular ingin berada di posisi (100,200), maka pastikan pada labirin bahwa karakter ke 10 dari baris ke 20 bukan merupakan dinding labirin. Jika posisi ular tersebut berada pada daerah yang merupakan dinding labirin, maka saat permainan dimulai, permainan akan berakhiri. Pastikan juga tidak ada baris kosong di baris terakhir, karena baris terakhir merupakan posisi ular pada labirin. Jika baris tersebut kosong, maka posisi ular tidak dapat ditentukan.

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

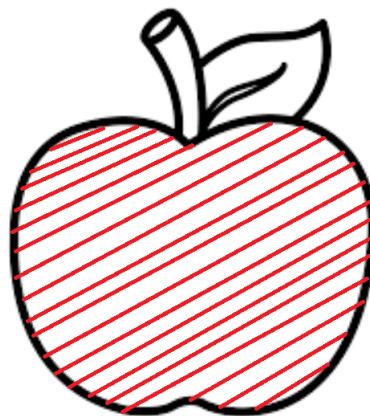
3.2.6 Pengecekan tabrakan (*Collision Detection*)

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

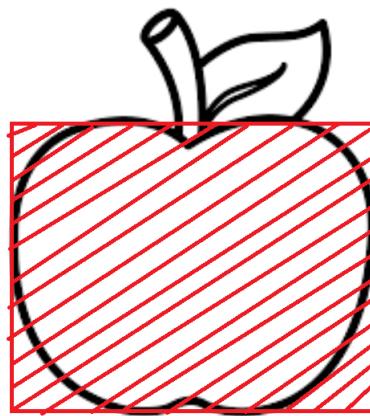
mengikuti ke mana kepala ular akan bergerak. Dengan ini, tidak perlu dilakukan *collision detection* antara bagian tubuh ular dengan apel.

Teknik tabrakan yang digunakan untuk pengecekan tabrakan pada Open Source Snake 360 adalah teknik tabrakan dengan pendeksi pixel. Teknik ini dipilih karena pergerakan ular tidak terlalu cepat. Selain itu, kecepatan ular tidak melebihi besar apel dan lebar dinding, sehingga pergerakan setiap framenya dapat dipastikan bahwa ular tidak akan melewati apel atau dinding labirin. Teknik tabrakan dengan pendeksi vektor lebih akurat namun teknik ini membutuhkan perhitungan tambahan untuk menelusuri pergerakan ular antara 2 buah frame.

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.



Gambar 3.17: Daerah tabrakan pada apel



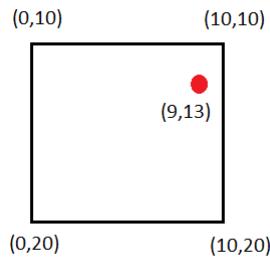
Gambar 3.18: Daerah tabrakan berbentuk persegi pada apel

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

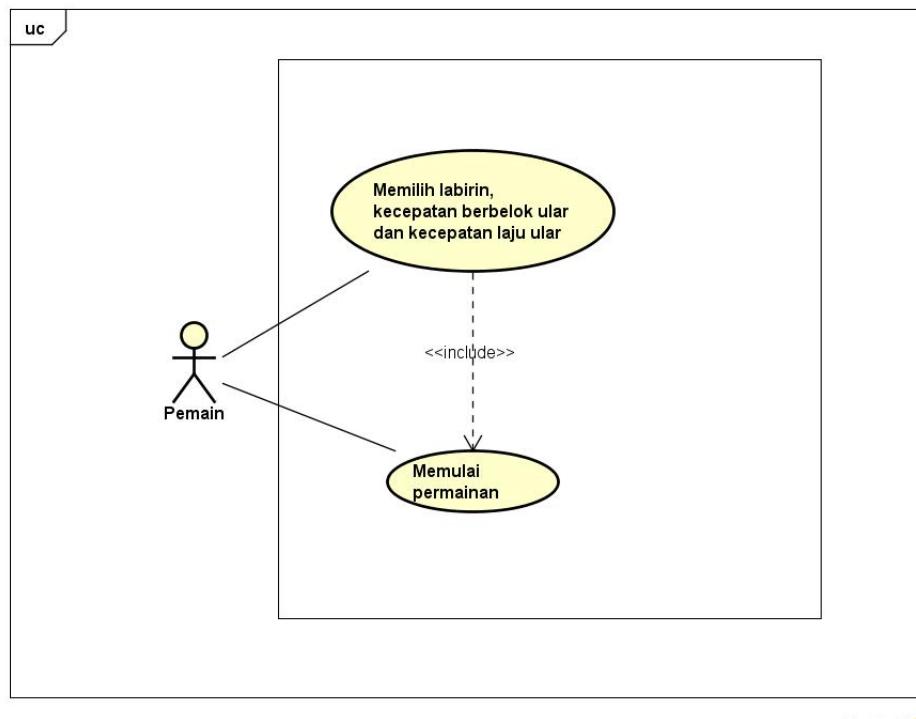
3.2.7 Memilih Labirin, Kecepatan Laju Ular dan Kecepatan Berbelok Ular

Sebelum memulai permainan, pemain harus memilih labirin, kecepatan laju ular dan kecepatan berbelok ular. Pemain memilih labirin mulai dari labirin pertama sampai jumlah labirin yang ada pada folder. Jika jumlah labirin pada folder ada 10, maka pemain dapat memilih labirin mulai dari labirin pertama sampai labirin ke 10. Pemain memilih kecepatan ular mulai dari 1 sampai dengan 5. Jika pemain memilih 1, maka ular akan melaju sebesar 2 *pixel/frame*, jika memilih 2, maka ular akan melaju sebesar 4 *pixel/frame* dan seterusnya. Pemain memilih kecepatan berbelok ular mulai dari 1 sampai 10. Jika pemain memilih 1 maka ular akan berbelok sebesar 1° , 2 untuk 2° dan seterusnya. Setelah pemain sudah memilih ketiga pilihan tersebut, pemain dapat memulai permainan.

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 melakukanya adalah pemain. Pada Gambar 3.20 terdapat diagram *use case* dari permainan *Snake 360*.



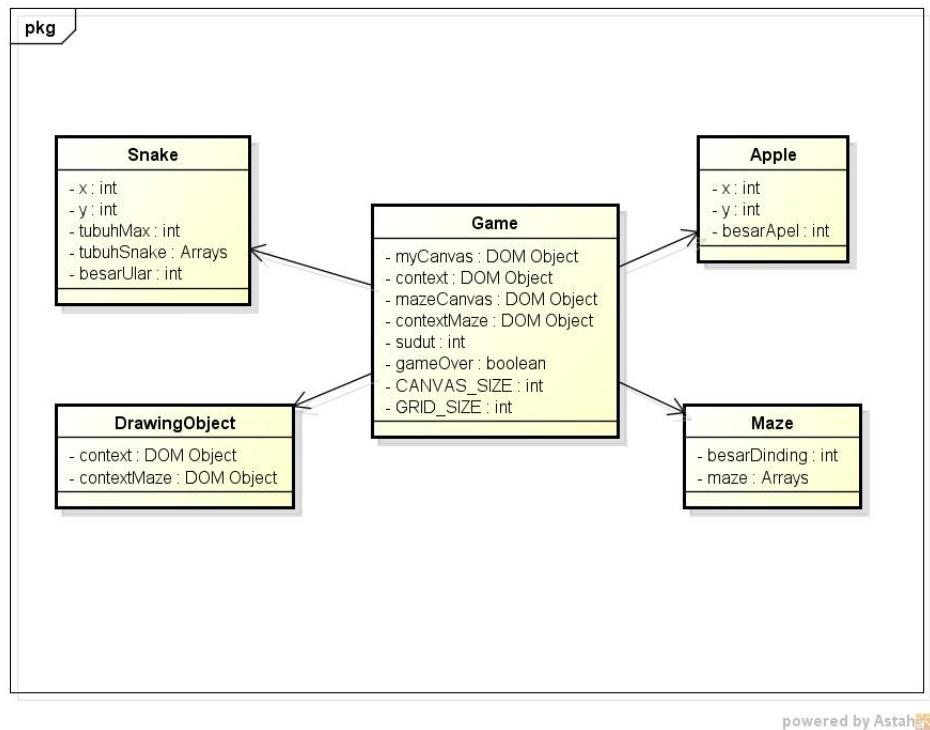
Gambar 3.20: Diagram *use case* dari permainan *Snake 360*

Berikut adalah skenario dari diagram *use case* :

1. Skenario : Memilih labirin, kecepatan berbelok ular dan kecepatan laju ular.
 Aktor : Pemain
 Prakondisi : Pemain memulai permainan.
 Skenario normal : Pemain memilih labirin, kecepatan berbelok ular, dan kecepatan laju ular sesuai keinginan.
 Eksepsi : -
2. Skenario : Mulai bermain
 Aktor : Pemain
 Prakondisi : Pemain sudah memilih labirin, kecepatan berbelok ular, dan kecepatan laju ular.
 Skenario normal : Pemain sudah mulai bermain.
 Eksepsi : -

3.3.2 Diagram Kelas

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



Gambar 3.21: Diagram kelas dari permainan *Snake 360*

Diagram kelas terdiri dari beberapa kelas yaitu :

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

Berikut adalah atribut yang dimiliki setiap kelas :

1. Kelas Snake

int

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

Array

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

2. Kelas Apel

int

- x, merupakan posisi apel pada koordinat x.

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

3. Kelas Game

int

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

DOM Object

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

boolean

- gameOver, memberitahu apakah permainan sudah berakhir atau belum.

4. Kelas Maze

int

- besarDinding, merupakan besar lebar dinding.

Array

- maze, merupakan layout labirin.

5. DrawingObject

DOM Object

- context, merupakan objek canvas untuk menggambar ular dan apel.
- contextMaze, merupakan objek canvas untuk menggambar labirin.

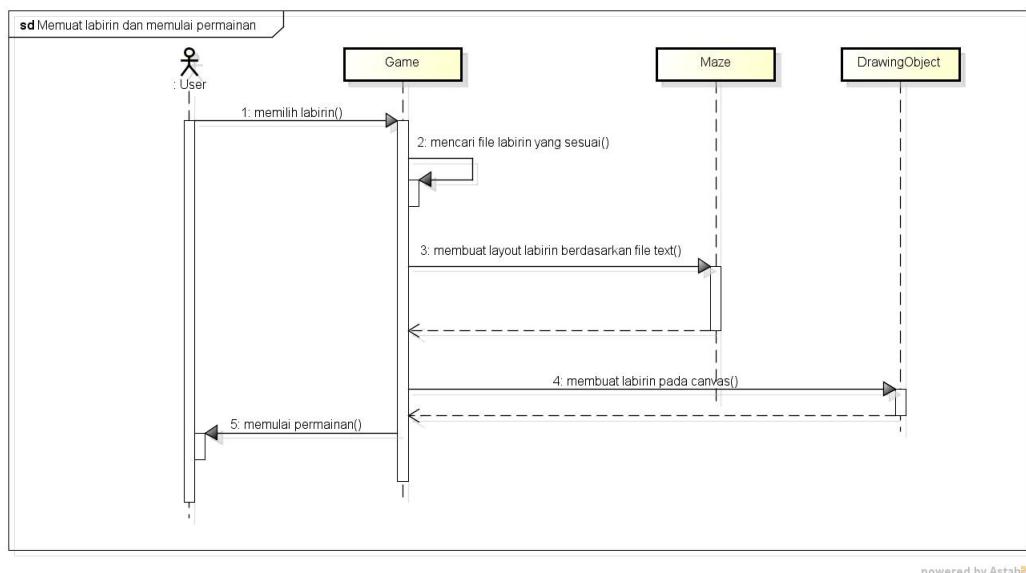
BAB 4

PERANCANGAN

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

4.1 Rancangan Diagram *Sequence*

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



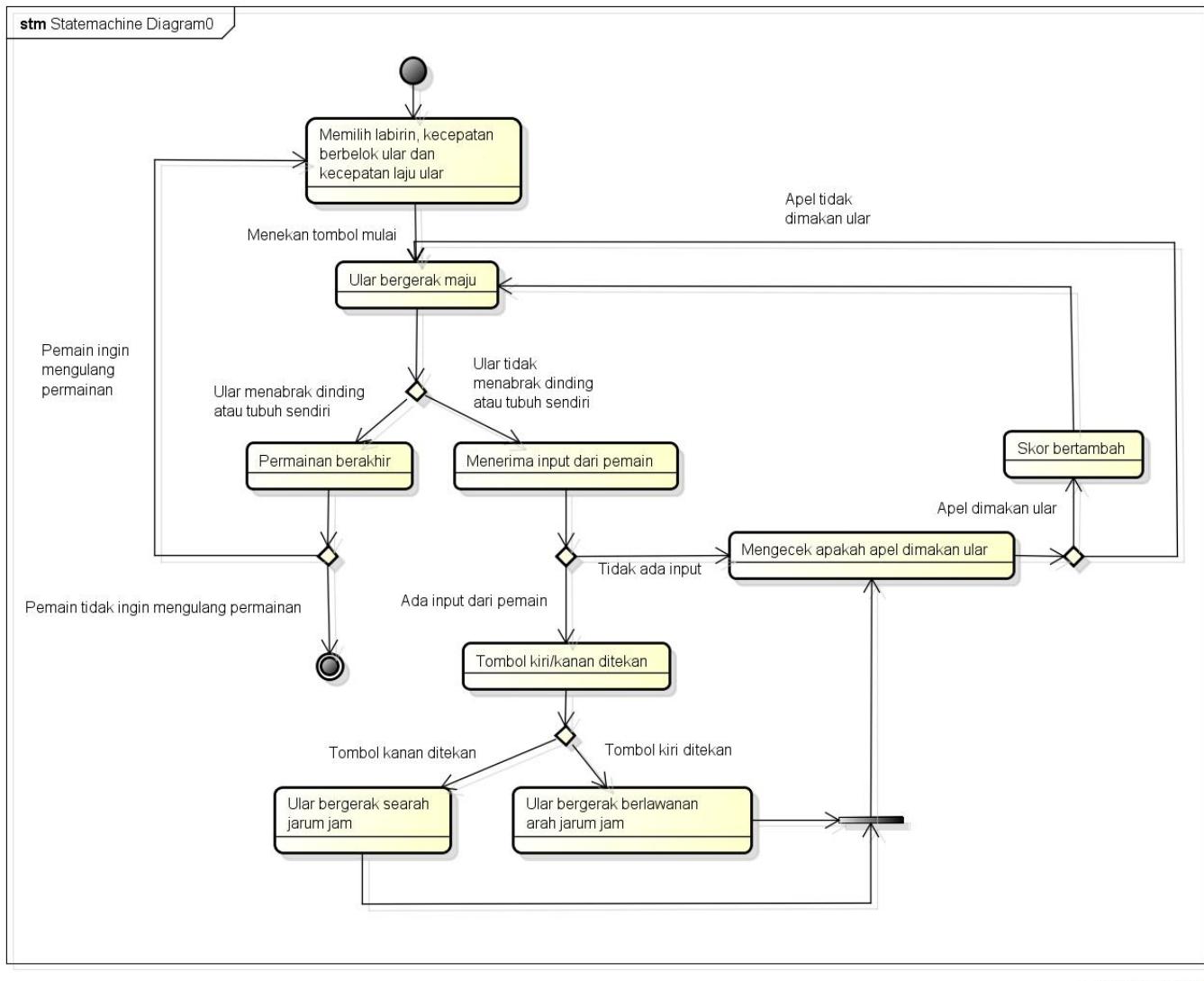
Gambar 4.1: Diagram *sequence* untuk memuat labirin

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

1. Pemain memilih labirin.
2. Kelas *Game* akan menerima input dari pemain dan mencari *file* labirin yang sesuai dengan yang pemain pilih di folder labirin. *File* labirin merupakan file teks.
3. Jika *file* ditemukan, maka kelas *Game* akan memanggil *method* kelas *Maze* untuk membuat labirin yang sesuai dengan isi *file* labirin.
4. Setelah labirin selesai dibuat, kelas *Game* akan memanggil *method* kelas *DrawingObject* untuk menggambar labirin.
5. Jika labirin sudah selesai digambar, kelas *Game* akan memulai permainan.

4.2 Rancangan Diagram State

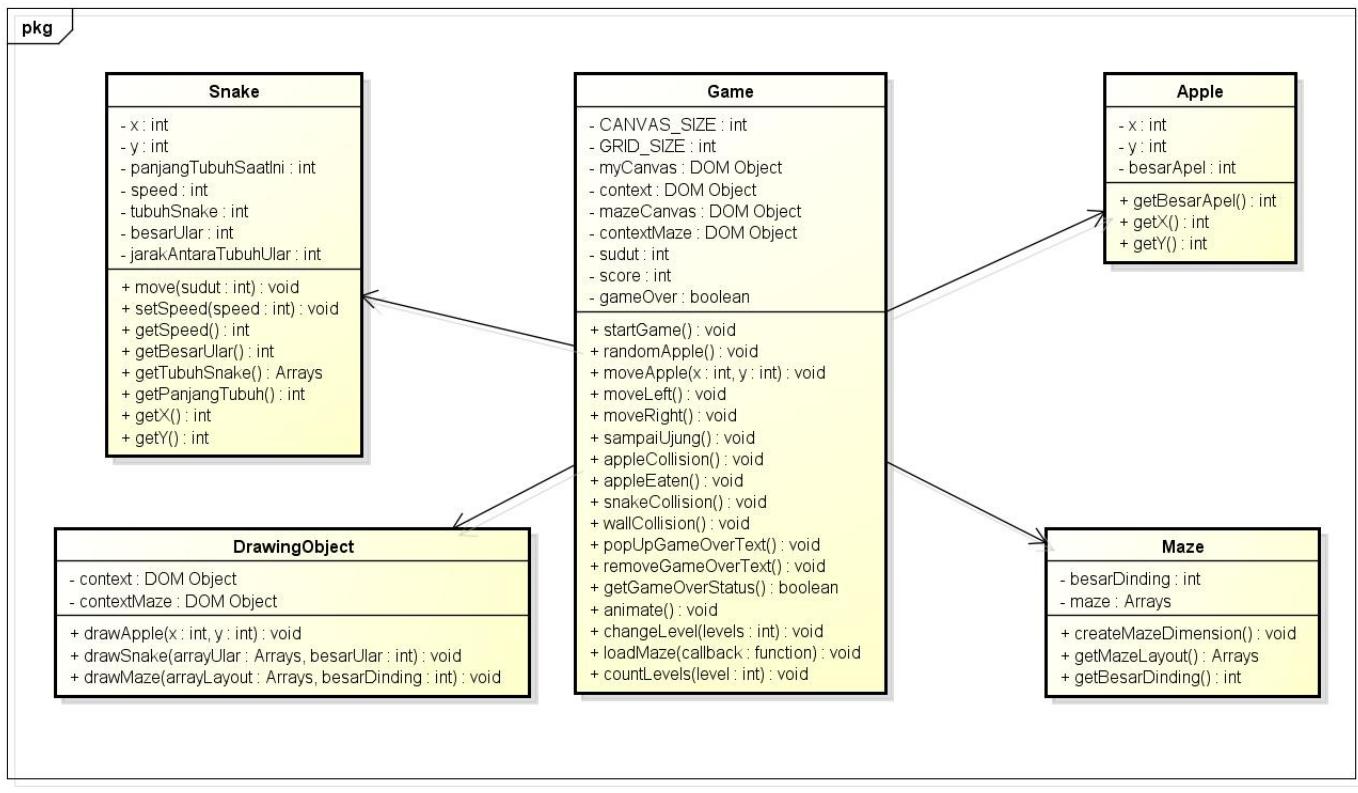
Pada bagian ini akan ditunjukkan diagram *state* pada Open Source Snake 360. Diagram *state* yang dibuat adalah proses bermain mulai dari awal sampai akhir permainan. Diagram *state* dapat dilihat pada Gambar 4.2.



Gambar 4.2: Diagram state proses bermain dari Open Source *Snake 360*

4.3 Rancangan Diagram Kelas Rinci

Pada bagian ini akan ditunjukkan dan dijelaskan diagram kelas dari *Open Source Snake 360* secara lengkap. Diagram kelas dapat dilihat pada Gambar 4.3.



powered by Astah

Gambar 4.3: Diagram kelas rinci dari Open Source *Snake* 360

Deskripsi Kelas dan Method

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

1. Kelas *Game*

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

- Nama *method* : *startGame*
Deskripsi : memulai permainan
Input : tidak ada
Output : tidak ada
- Nama *method* : *randomApple*
Deskripsi : mengacak posisi apel
Input : tidak ada
Output : tidak ada
- Nama *method* : *moveApple*
Deskripsi : memindahkan posisi apel
Input : *int x, int y*
 - x : koordinat x milik apel
 - y : koordinat y milik apel

Output : tidak ada

- Nama *method* : *moveLeft*

Deskripsi : membuat ular bergerak berlawanan arah jarum jam

Input : tidak ada

Output : tidak ada

- Nama *method* : *moveRight*

Deskripsi : membuat ular bergerak searah jarum jam

Input : tidak ada

Output : tidak ada

- Nama *method* : *sampaiUjung*

Deskripsi : membuat ular akan muncul di sisi yang berlawanan ketika ular sudah mencapai ujung labirin

Input : tidak ada

Output : tidak ada

- Nama *method* : *appleCollision*

Deskripsi : mengecek tabrakan antara apel dengan kepala ular

Input : tidak ada

Output : tidak ada

- Nama *method* : *appleEaten*

Deskripsi : aksi yang dilakukan apabila ular sudah memakan apel

Input : tidak ada

Output : tidak ada

- Nama *method* : *snakeCollision*

Deskripsi : mengecek tabrakan antara kepala ular dengan tubuhnya sendiri

Input : tidak ada

Output : tidak ada

- Nama *method* : *wallCollision*

Deskripsi : mengecek tabrakan antara kepala ular dengan dinding labirin

Input : tidak ada

Output : tidak ada

- Nama *method* : *popUpGameOverText*

Deskripsi : memunculkan tulisan 'Game Over'

Input : tidak ada

Output : tidak ada

- Nama *method* : *removeGameOverText*

Deskripsi : menghilangkan tulisan 'Game Over'

Input : tidak ada

Output : tidak ada

- Nama *method* : *getGameOverStatus*
Deskripsi : mendapatkan status gameOver
Input : tidak ada
Output : *boolean gameOver*
 - *gameOver* : status apabila permainan sudah berakhir atau belum
- Nama *method* : *animate*
Deskripsi : membuat animasi dari setiap objek
Input : tidak ada
Output : tidak ada
- Nama *method* : *changeLevel*
Deskripsi : mengubah jumlah labirin pada menu utama
Input : *int levels*
 - *levels* : jumlah *labirin* pada server
- Nama *method* : *loadMaze*
Deskripsi : mengambil dan memenuhi labirin yang dipilih pemain
Input : *function callback*
 - *callback* : fungsi yang akan dijalankan setelah *method loadMaze* selesai mengeksekusi
- Nama *method* : *countLevels*
Deskripsi : menghitung jumlah *file* labirin pada server
Input : *int level*
 - *level* : *file* labirin yang ingin dicek keberadaanya pada server
- Nama *method* : *getLevel*
Deskripsi : mendapatkan jumlah *file* labirin pada server
Input : tidak ada

2. Kelas *Snake*

Kelas *Snake* merupakan kelas yang merepresentasikan objek ular.

- Nama *method* : *move*
Deskripsi : memulai permainan
Input : *int x, int y*
 - *x* : koordinat x milik ular
 - *y* : koordinat y milik ular
- Nama *method* : *setSpeed*
Deskripsi : mengubah nilai atribut *speed*
Input : *int speed*
 - *speed* : kecepatan laju ular
- Nama *method* : *getSpeed*
Deskripsi : mendapatkan nilai atribut *speed*
Input : tidak ada
Output : *int speed*
 - *speed* : kecepatan laju ular

- Nama *method* : *getBesarUlar*
Deskripsi : mendapatkan nilai atribut besarUlar
Input : tidak ada
Output : *int speed*
 - besarUlar : lebar tubuh ular

3. Kelas *DrawingObject*

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

- Nama *method* : *drawApple*
Deskripsi : menggambar objek apel
Input : *int x, int y*
 - x : koordinat x milik apel
 - y : koordinat y milik apel
Output : tidak ada
- Nama *method* : *drawSnake*
Deskripsi : menggambar objek ular
Input : *int[] arrayUlar, int besarUlar*
 - arrayUlar : koordinat x dan y milik setiap bagian tubuh ular
 - besarUlar : lebar tubuh ular
Output : tidak ada
- Nama *method* : *drawMaze*
Deskripsi : menggambar labirin
Input : *String arrayLayout, int besarDinding*
 - arrayLayout : layout labirin yang akan digambar
 - besarDinding : besar dinding labirin
Output : tidak ada

4. Kelas *Apple*

Kelas *Apple* adalah kelas yang merepresentasikan objek apel.

- Nama *method* : *getBesarApel*
Deskripsi : mendapatkan nilai atribut besarApel
Input : tidak ada
Output : *int besarApel*
 - besarApel : besar apel

5. Kelas *Maze*

Kelas *Maze* adalah kelas yang merepresentasikan labirin.

- Nama *method* : *setMazeLayout*
Deskripsi : membuat labirin berdasarkan file teks yang dipilih
Input : *text layoutInText*
 - layoutInText : isi teks file labirin

Output : tidak ada

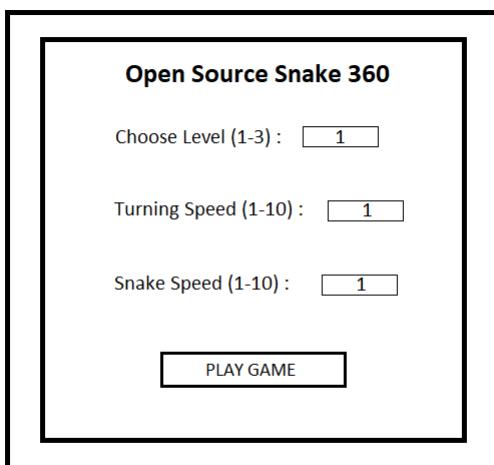
- Nama *method* : *getMazeLayout*
Deskripsi : mendapatkan labirin
Input : tidak ada
Output : *Arrays maze*
 - maze : labirin
- Nama *method* : *getBesarDinding*
Deskripsi : mendapatkan besar dinding labirin
Input : tidak ada
Output : *int besarDinding*
 - besarDinding : besar dinding labirin

4.4 Rancangan Tampilan Antarmuka

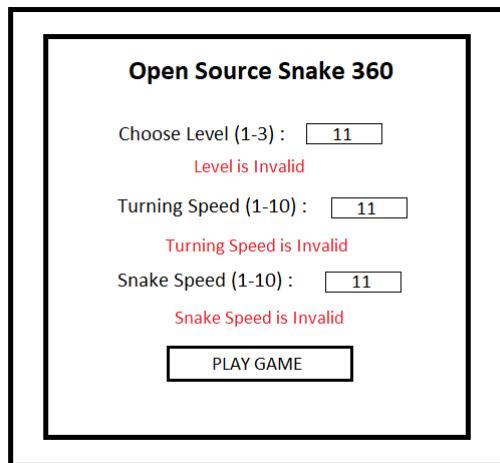
Pada bagian ini akan ditunjukkan rancangan tampilan antarmuka dari permainan yang dibangun yang terdiri dari menu pemilihan labirin, mulai bermain, dan permainan berakhir.

4.4.1 Tampilan Menu Utama

Gambar 4.4 merupakan rancangan tampilan awal dari permainan yang dibangun. Pada tampilan ini terdapat judul permainan dan 3 buah *input*, yaitu memilih labirin, memilih kecepatan gerak ular, dan memilih kecepatan ular berbelok dan sebuah tombol '*Play Game*' untuk memulai permainan. Tampilan menu utama akan menampilkan pesan kesalahan seperti terdapat pada Gambar 4.5. Apabila pemain salah memasukkan salah satu data, maka permainan tidak akan dimulai jika tombol '*Play Game*' ditekan.



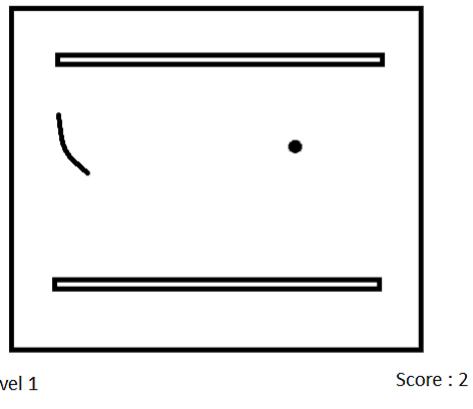
Gambar 4.4: Rancangan tampilan menu utama



Gambar 4.5: Rancangan tampilan menu utama jika pemain salah memasukkan data

4.4.2 Tampilan Bermain

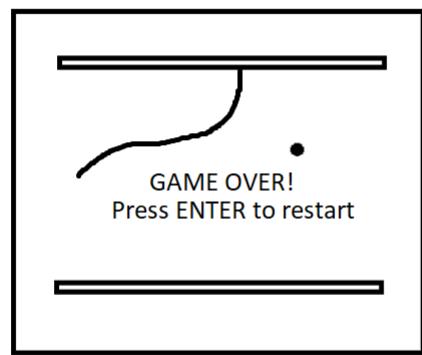
Tampilan bermain akan muncul setelah pemain memilih labirin, kecepatan ular dan kecepatan ular berbelok pada menu utama dengan benar dan sudah menekan tombol '*Play Game*' (Gambar 4.4). Gambar 4.6 merupakan tampilan permainan sudah dimulai. Pada tampilan ini terdapat ular, dinding labirin dan makanan berbentuk apel. Pada tampilan ini juga terdapat labirin yang dipilih dan skor yang didapat.



Gambar 4.6: Rancangan tampilan bermain

4.4.3 Tampilan Permainan Berakhir

Tampilan ini akan muncul apabila permainan berakhir. Permainan akan berakhir jika ular menabrak dinding labirin atau menabrak tubuhnya sendiri. Gambar 4.7 merupakan tampilan permainan berakhir. Pada tampilan ini, pemain dapat mengulang permainan dengan menekan tombol '*Enter*'. Pemain akan dialihkan ke tampilan menu utama(Gambar 4.4) apabila tombol '*Enter*' ditekan.



Level 1

Score : 15

Gambar 4.7: Rancangan tampilan permainan berakhir

BAB 5

IMPLEMENTASI DAN PENGUJIAN

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

5.1 Implementasi

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

5.1.1 Lingkungan Perangkat Keras

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

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

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

Perangkat 1

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

Perangkat 2

1. Perangkat : SM-J730G
2. *Processor* : Exynos 7870 Octa 1600MHz Cortex-A53

3. RAM : 3.00 GB
4. *Video Card* : Mali-T830
5. Monitor : 5.5"
6. *Storage* : 32 GB

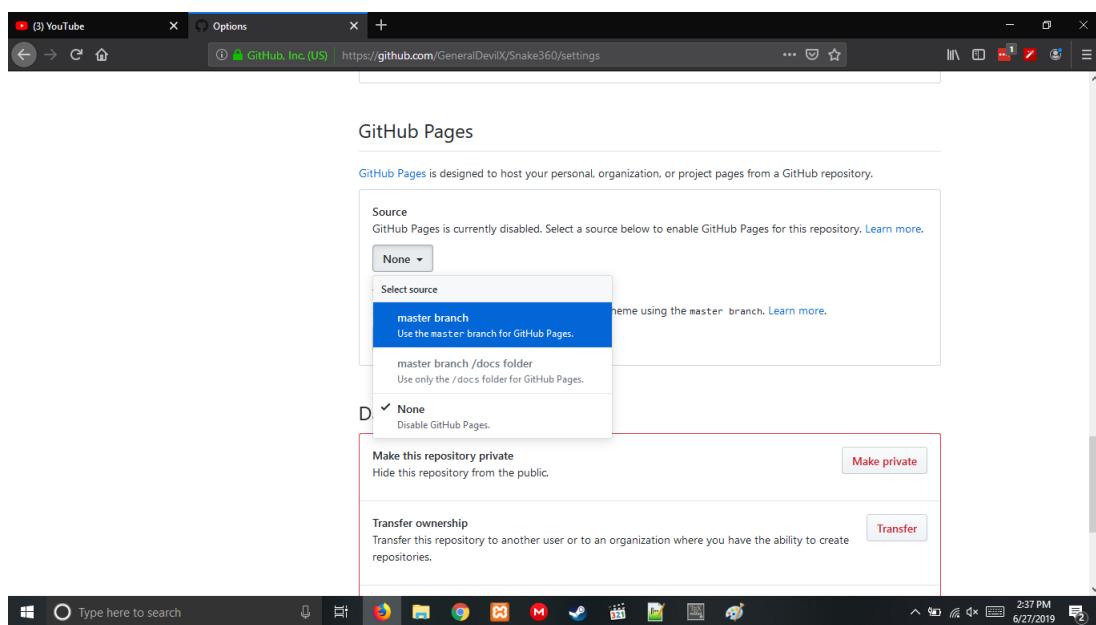
5.1.2 Lingkungan Perangkat Lunak

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

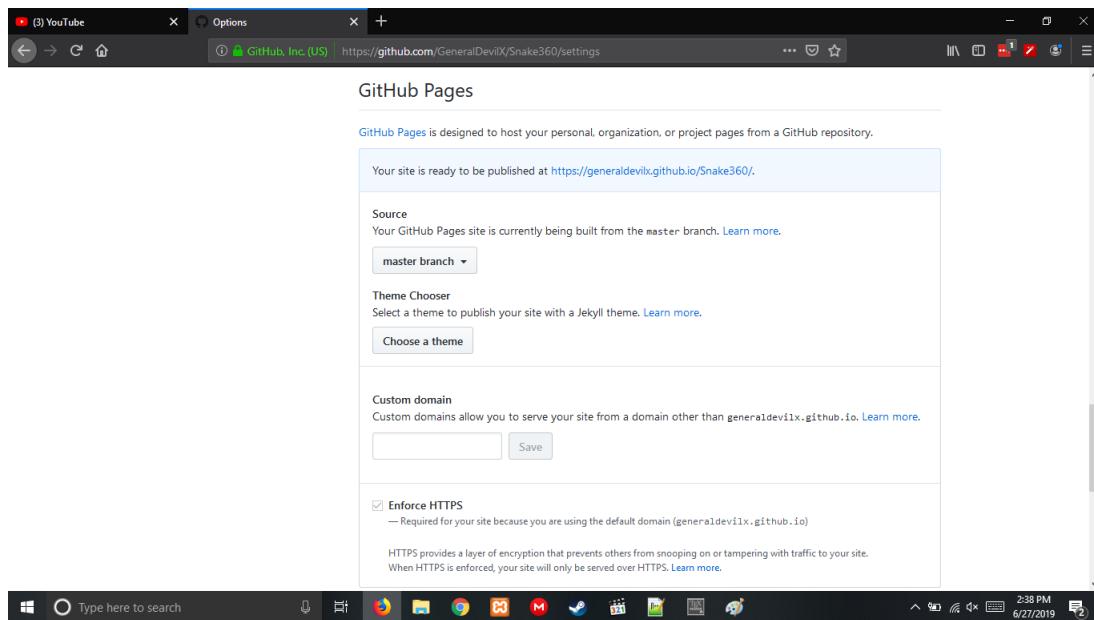
1. Sistem Operasi Laptop : Windows 10 64-bit
2. Bahasa Pemrograman : *Javascript*, HTML
3. Sistem Operasi *Smartphone* : Android Nougat v7.0

5.1.3 Lingkungan Pengujian

Open Source Snake 360 dapat dimainkan pada link: <https://generaldevilx.github.io/Snake360/> karena permainan ini sudah dihosting di GitHub. Dengan cara ini, orang lain dapat memainkan Open Source Snake 360 secara langsung dengan mengakses link tersebut. Sebelum hosting, pada repository harus terdapat halaman utama yaitu 'index.html'. Caranya adalah : buka repository kemudian pilih tab 'settings'. Sesudah itu, carilah bagian 'GitHub Pages' kemudian pada bagian source, klik dropdown list dan akan muncul list seperti pada Gambar 5.1. List tersebut adalah branch pada repository yang akan dihosting. Karena peneliti menggunakan branch master untuk membuat Open Source Snake 360, maka list pertama akan dipilih. Setelah itu, GitHub akan menyediakan link yang dapat diakses oleh orang lain yang hasilnya dapat dilihat pada Gambar 5.2.



Gambar 5.1: Tampilan memilih branch repository untuk dihosting



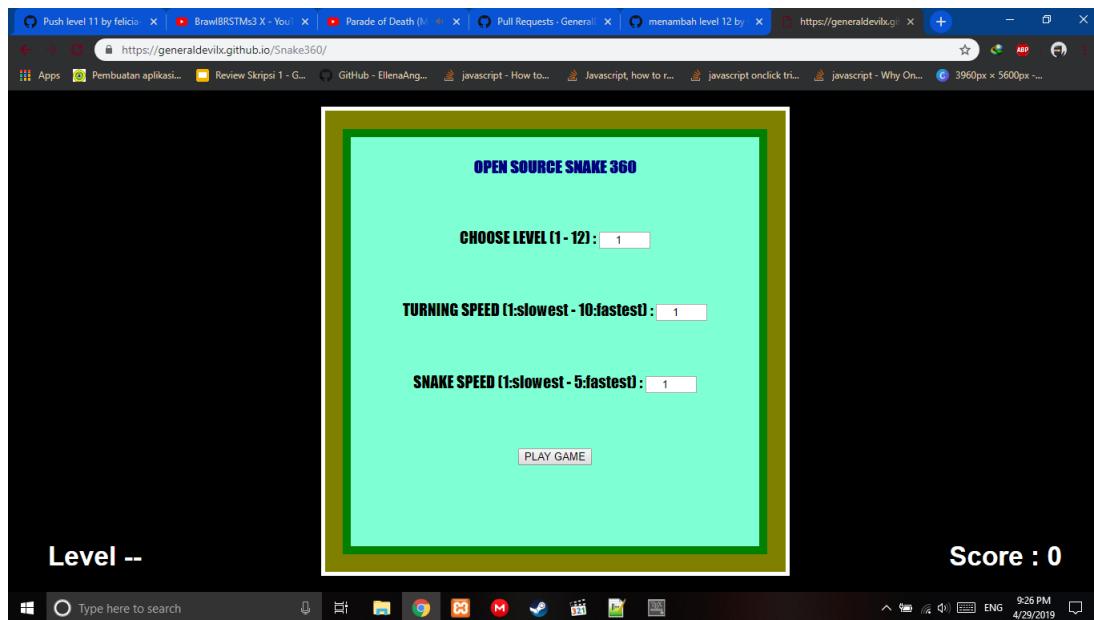
Gambar 5.2: Tampilan setelah melakukan hosting di GitHub

5.1.4 Implementasi Antarmuka

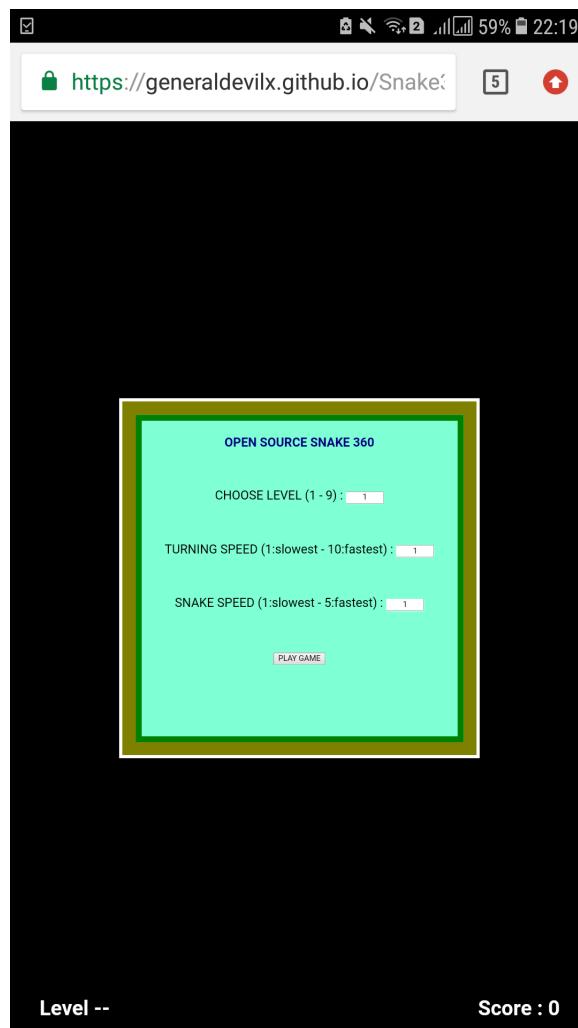
Pada subbab ini akan ditampilkan dan dijelaskan tampilan antarmuka dari Open Source Snake 360.

Tampilan Menu Utama

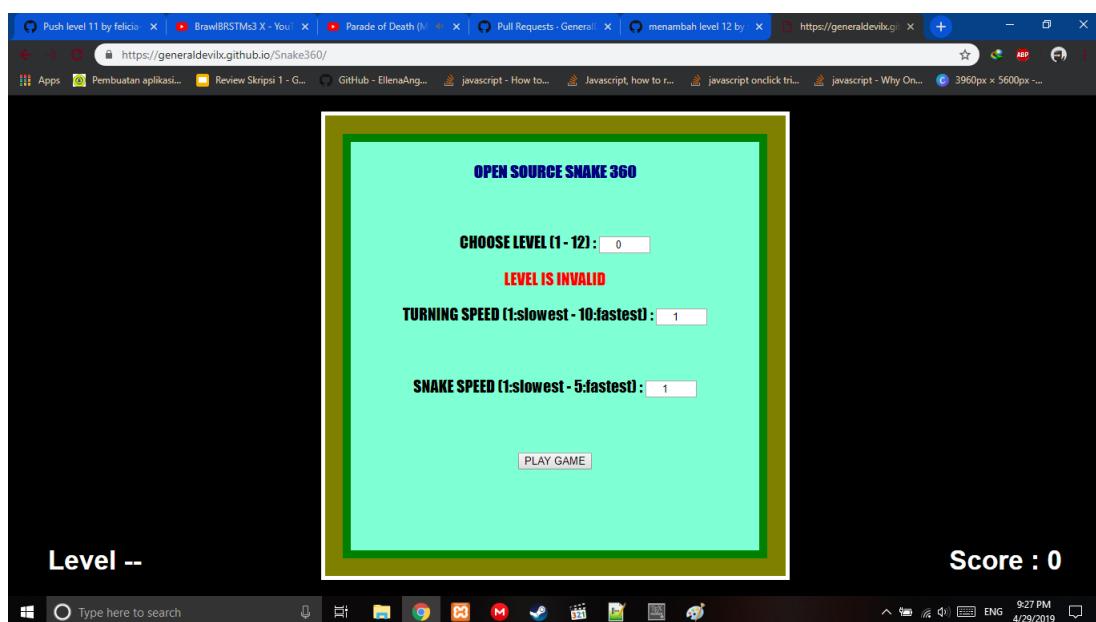
Gambar 5.3 dan Gambar 5.4 merupakan tampilan antarmuka menu utama pada *desktop* dan *smartphone*. Pada tampilan ini terdapat judul dari permainan, *input* untuk mengisi labirin, kecepatan ular berbelok, kecepatan ular, dan tombol '*Play Game*'. Jika pemain salah memasukkan data, maka terdapat pesan kesalahan yang ditandai dengan tulisan bewarna merah. Misal, pada Gambar 5.5 dan Gambar 5.6, pemain salah memasukkan data untuk labirin sehingga muncul pesan kesalahan bahwa *input* yang dimasukkan salah.



Gambar 5.3: Tampilan menu utama pada *desktop*



Gambar 5.4: Tampilan menu utama pada *smartphone*



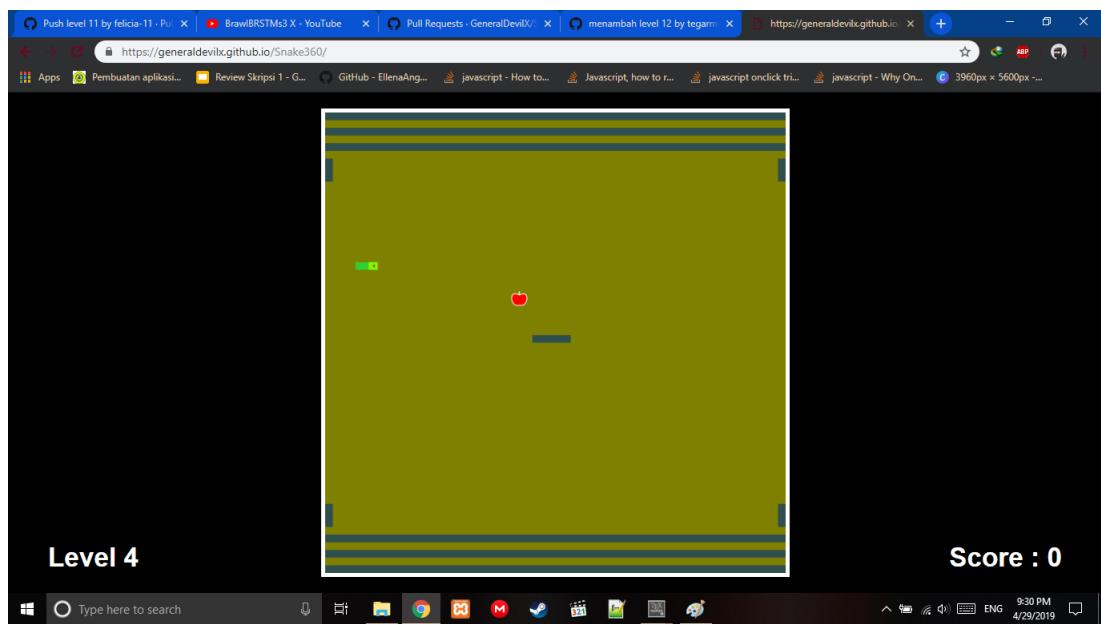
Gambar 5.5: Tampilan menu utama jika pemain salah memasukkan data labirin pada *desktop*



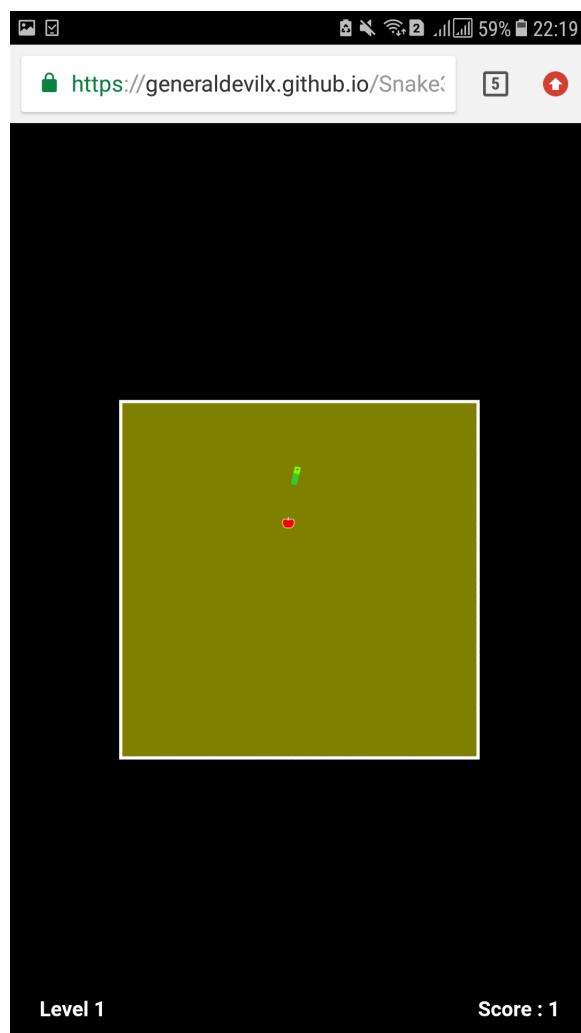
Gambar 5.6: Tampilan menu utama jika pemain salah memasukkan data labirin pada *smartphone*

Tampilan Bermain

Gambar 5.7 dan Gambar 5.8 merupakan tampilan antarmuka mulai bermain pada *desktop* dan *smartphone*. Tampilan ini muncul apabila pemain memasukkan data labirin, kecepatan ular berbelok dan kecepatan ular dengan benar dan menekan tombol "*Play Game*". Pada tampilan ini terdapat ular yang dikontrol oleh pemain, dinding labirin, makanan ular, labirin dan skor yang didapat pemain.



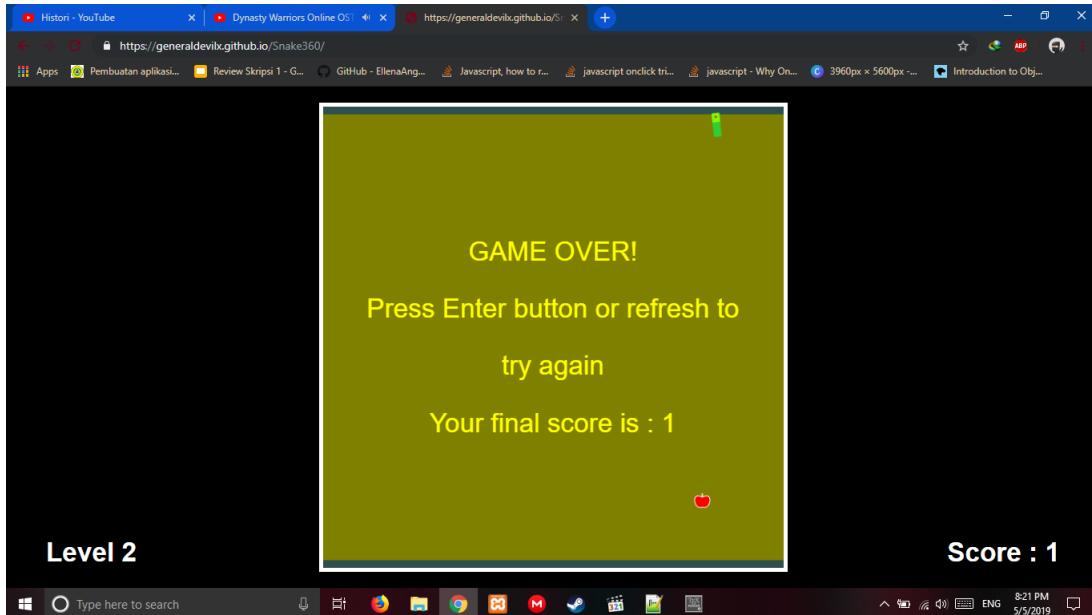
Gambar 5.7: Tampilan bermain pada *desktop*



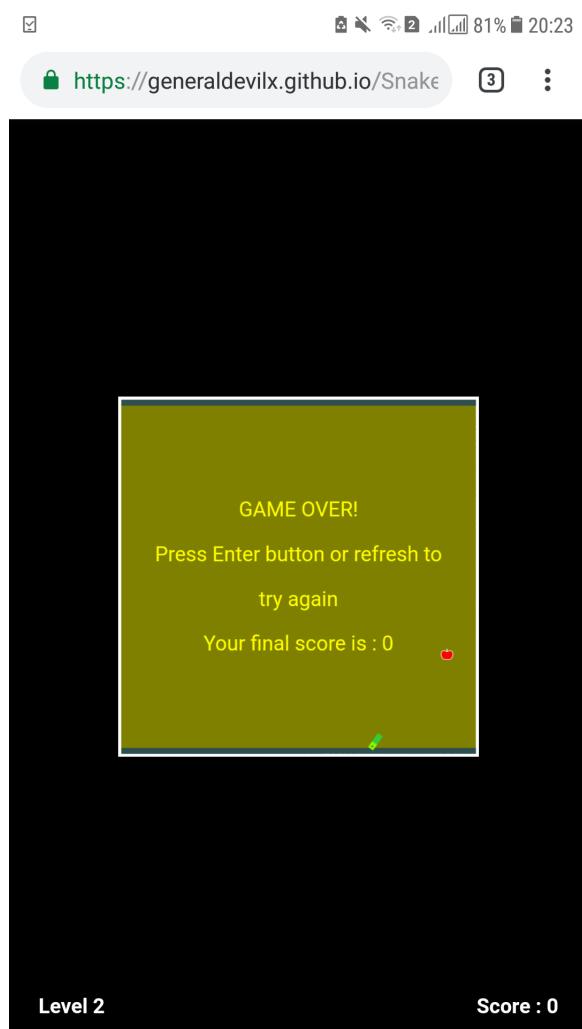
Gambar 5.8: Tampilan bermain pada *smartphone*

Tampilan Permainan Berakhir

Gambar 5.9 dan Gambar 5.10 merupakan tampilan antarmuka jika permainan berakhir pada *desktop* dan *smartphone*. Tampilan ini muncul apabila ular menabrak dinding labirin dan menabrak tubuhnya sendiri. Pemain akan diarahkan ke tampilan utama jika pemain menekan tombol 'Enter' pada tampilan ini. Karena pada smartphone tidak memiliki tombol 'Enter', maka pemain hanya memuat ulang/refresh halaman tersebut.



Gambar 5.9: Tampilan permainan berakhir pada desktop



Gambar 5.10: Tampilan permainan berakhir smartphone

5.2 Pengujian

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

5.2.1 Pengujian Fungsional

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

1. Pengujian fungsionalitas pada tampilan menu utama.

Tabel 5.1: Pengujian Fungsional pada Tampilan Menu Utama

Kasus uji	Hasil yang diharapkan	Hasil uji
Pemain memilih labirin dan kecepatan berbelok	Jika pemain salah memasukkan data labirin, maka akan ditampilkan sebuah text bahwa data yang diisi tidak valid	Hasil pengujian sesuai dengan yang diharapkan
Pada bagian pemilihan labirin, terdapat jumlah labirin yang dapat dipilih oleh pemain	Jumlah labirin yang ditampilkan harus sesuai dengan jumlah labirin yang terdapat pada server	Hasil pengujian sesuai dengan yang diharapkan
Pemain menekan tombol "Play Game"	Pemain akan diarahkan ke tampilan bermain. Kondisi untuk dapat memulai permainan adalah data labirin, kecepatan berbelok dan kecepatan laju ular sudah valid. Posisi ular dan dinding labirin sesuai dengan 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.

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

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 " <i>game over</i> " akan muncul	Hasil pengujian sesuai dengan yang diharapkan
Ular menabrak tubuh sendiri	Tampilan " <i>game over</i> " 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
Tombol arah kanan ditekan 9 kali dengan kecepatan sudut sebesar 10	Ular akan bergerak ke bawah	Hasil pengujian sesuai dengan yang diharapkan
Tombol arah kiri ditekan 9 kali dengan kecepatan sudut sebesar 10	Ular akan bergerak ke atas	Hasil pengujian sesuai dengan yang diharapkan

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

3. Pengujian fungsionalitas 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
Bagian kanan layar ditekan 9 kali dengan kecepatan sudut sebesar 10	Ular akan bergerak ke bawah	Hasil pengujian sesuai dengan yang diharapkan
Bagian kiri layar ditekan 9 kali dengan kecepatan sudut sebesar 10	Ular akan bergerak ke atas	Hasil pengujian sesuai dengan yang diharapkan

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

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

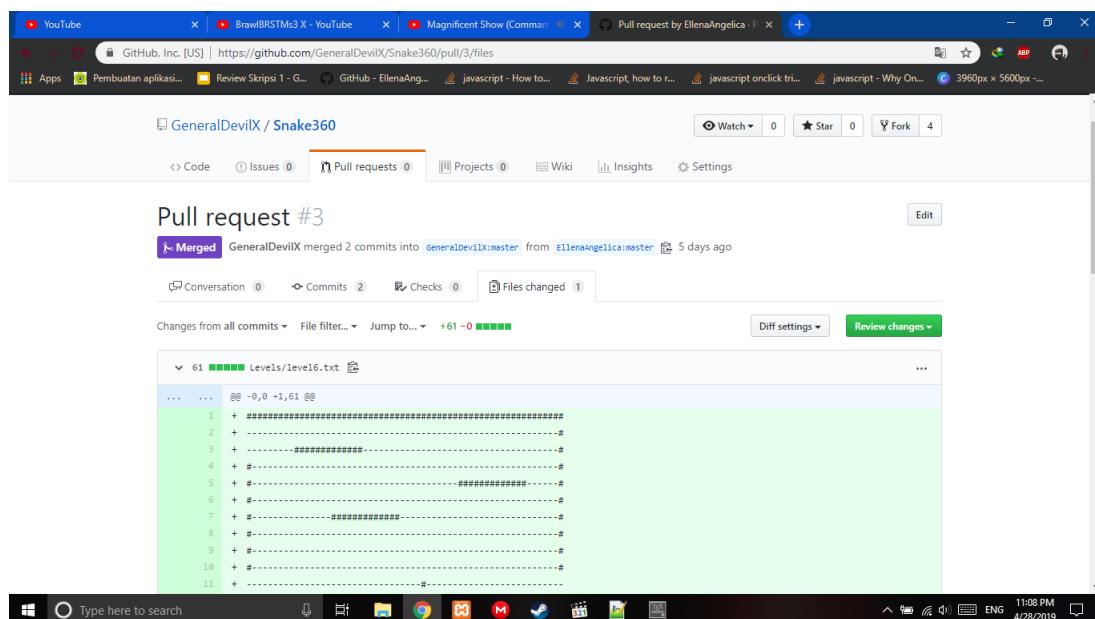
Berdasarkan tabel 5.4, dapat disimpulkan bahwa kasus uji pada tampilan "game over" membandingkan hasil sesuai dengan yang diharapkan.

5.2.2 Pengujian Eksperimental

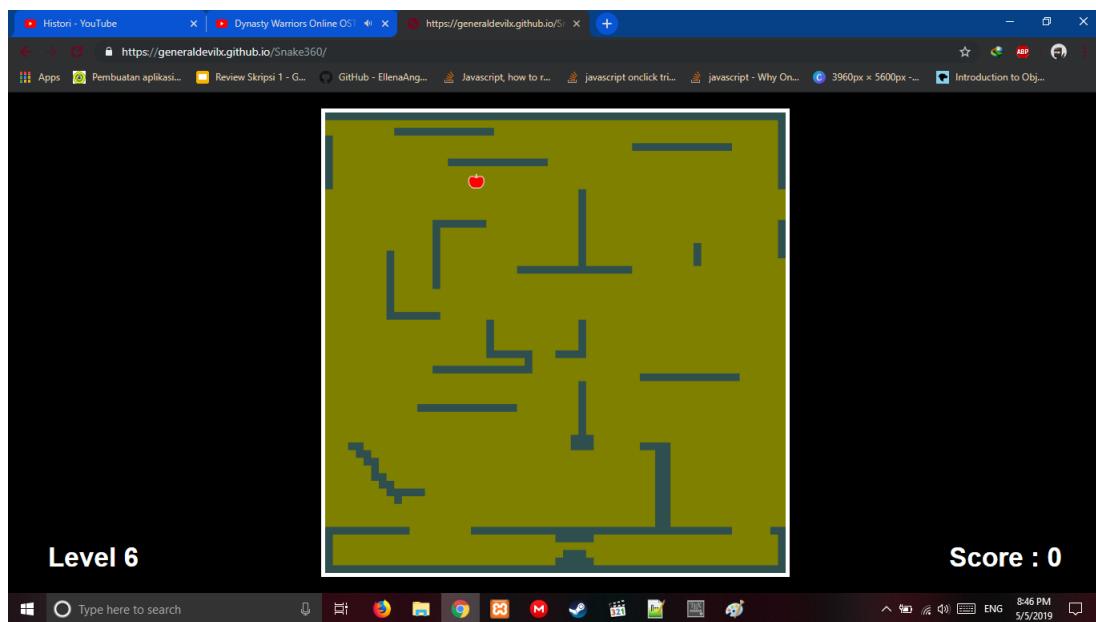
Pada pengujian eksperimental akan diuji penambahan labirin oleh orang lain. Pada pengujian ini terdapat 5 orang yang akan menambahkan labirin menggunakan *pull request GitHub*. Setiap penguji diminta untuk membaca *file readme* (lampiran B) yang berisikan tentang cara menambah labirin. Berikut adalah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

1. Penguji 1 : EllenaAngelica

Penguji 1 berhasil menambahkan labirin ke 6 dengan menggunakan *pull request* seperti yang terlihat pada Gambar 5.11. *Link* untuk *file* teks yang dibuat penguji 1 : <https://github.com/GeneralDevilX/Snake360/pull/3/files>. Gambar 5.12 merupakan hasil pengujian menggunakan labirin yang dibuat oleh penguji 1. Pada Gambar 5.12, Penguji 1 membuat labirin yang sudah sesuai dengan besar *canvas*. Namun posisi ular yang dimasukkan tidak sesuai. Penguji 1 menambahkan sebuah baris kosong di akhir *file* tersebut sehingga menyebabkan permainan menjadi *error*. Hal ini disebabkan karena pembacaan posisi ular selalu mengambil teks baris paling terakhir. Karena itu, peneliti harus memperbaiki labirin yang dibuat oleh penguji 1.



Gambar 5.11: Tampilan hasil *pull request* milik penguji 1

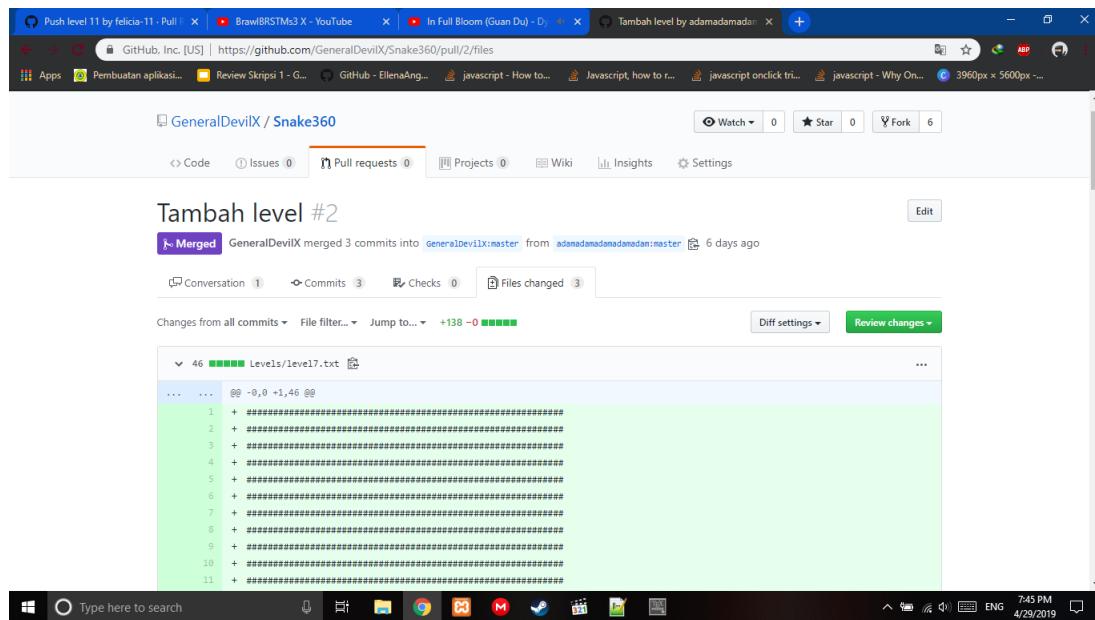


Gambar 5.12: Tampilan pengujian labirin yang dibuat oleh penguji 1

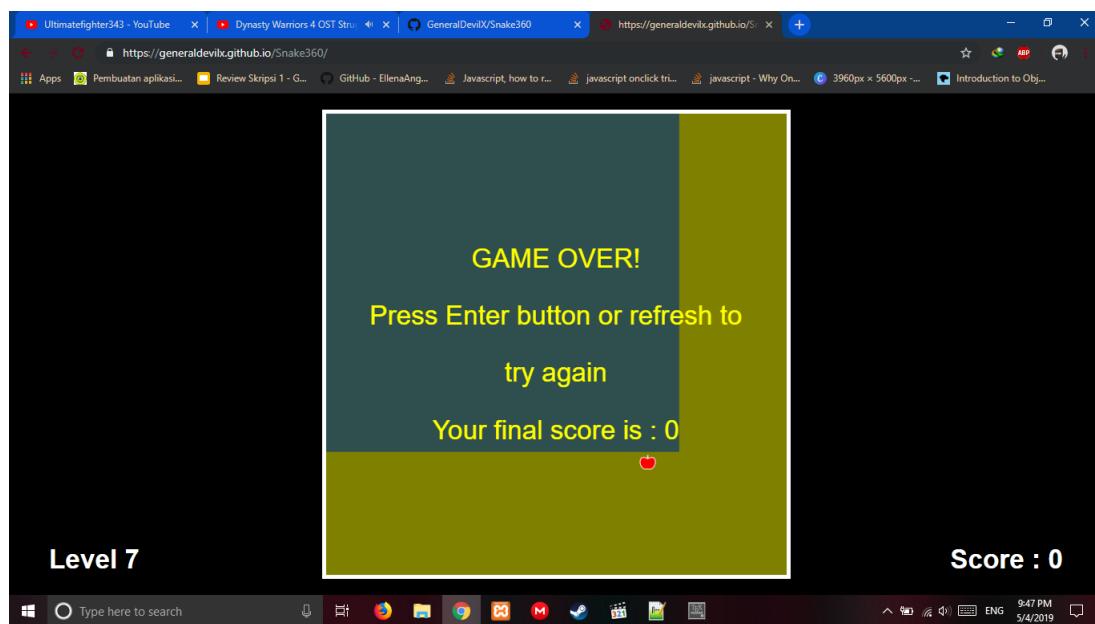
2. Penguji 2 : adamadamadamadam

Penguji 2 berhasil menambahkan labirin ke 7, 8 dan 9 dengan menggunakan *pull request* seperti yang terlihat pada Gambar 5.13. *Link* untuk file teks yang dibuat penguji 2 : <https://github.com/GeneralDevilX/Snake360/pull/2/files>. Penguji 2 membuat labirin ke 7 yang seluruhnya terdapat dinding dan memposisikan ular di koordinat (0,0). Selain itu, pada file teks labirin ke 7, labirin dibuat kurang dari 60 baris. Penguji 2 membuat labirin ke 7 dan labirin ke 8. Gambar 5.14 merupakan hasil pengujian labirin ke 7.

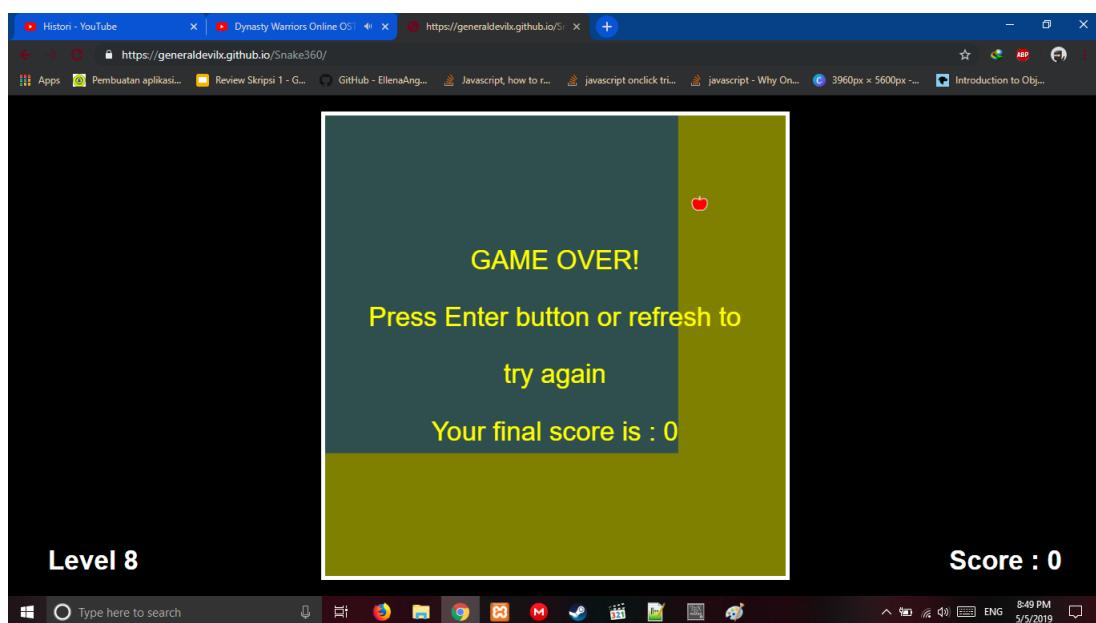
Ketiga labirin yang dibuat oleh penguji 2 tidak sesuai dengan besar *canvas* sehingga tampilan menjadi seperti pada Gambar 5.14. Penguji 2 membuat labirin ke 7 dan 8 yang seluruhnya adalah dinding dan memposisikan ular di (0,0). Pada labirin ke 9, penguji 2 menambahkan sebuah baris kosong di akhir file sehingga menyebabkan permainan menjadi *error*. Peneliti ingin memberitahukan penguji 2 bahwa labirin yang dibuat belum benar, tetapi terjadi sebuah kesalahan yaitu peneliti menekan tombol *merge request*. Hal ini membuat peneliti harus memperbaiki labirin yang dibuat oleh penguji 2.



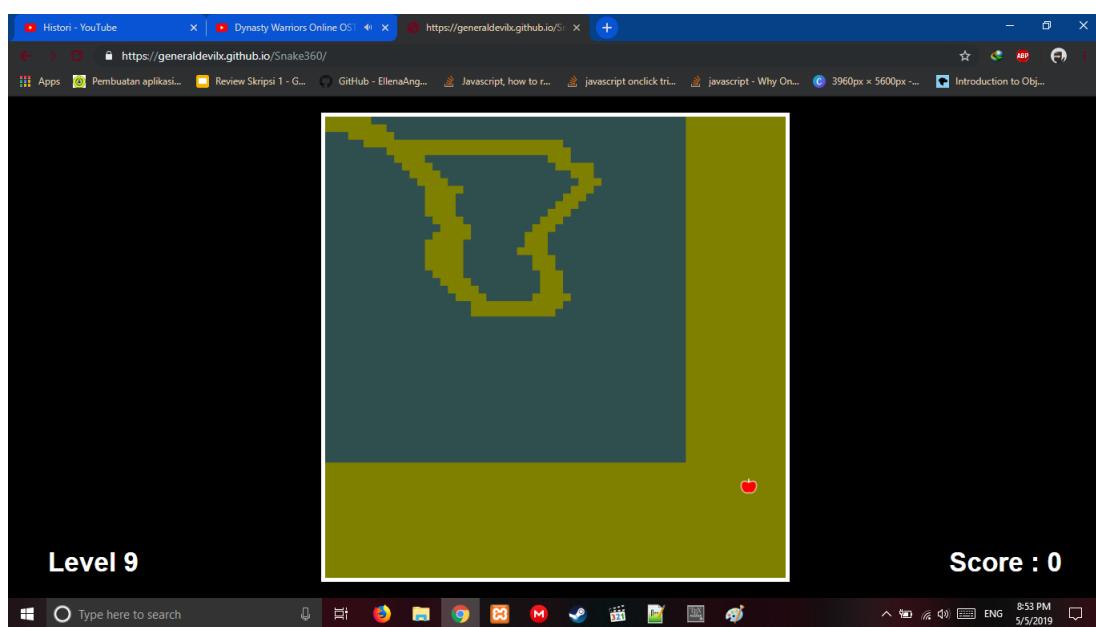
Gambar 5.13: Tampilan hasil *pull request* milik penguji 2



Gambar 5.14: Tampilan pengujian labirin ke 7 yang dibuat oleh penguji 2



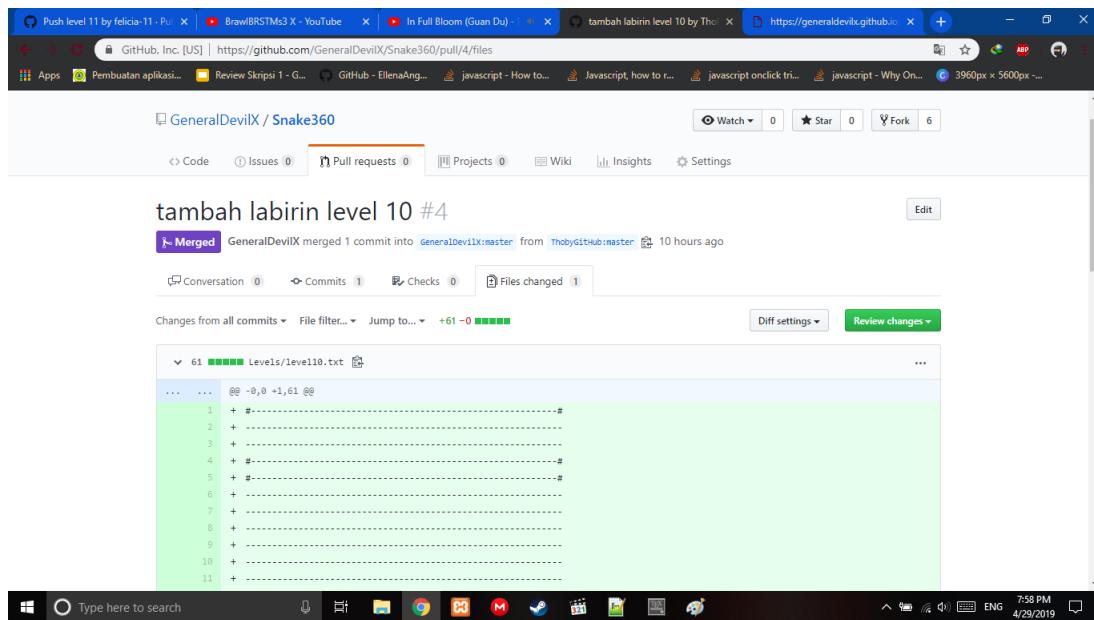
Gambar 5.15: Tampilan pengujian labirin ke 8 yang dibuat oleh penguji 2



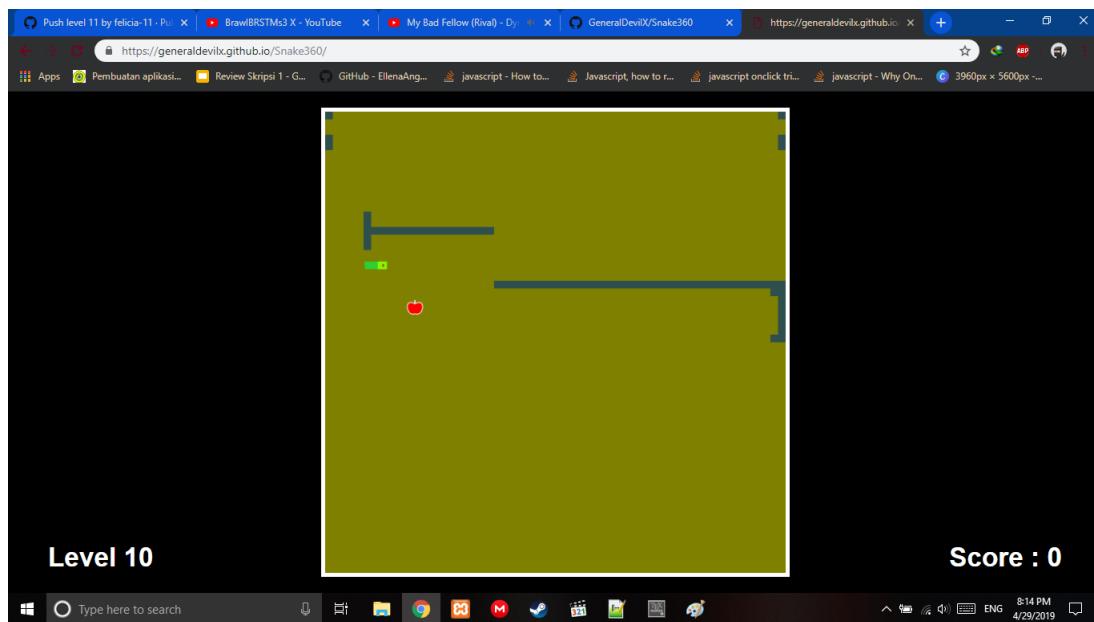
Gambar 5.16: Tampilan pengujian labirin ke 9 yang dibuat oleh penguji 2

3. Penguji 3 : ThobyGitHub

Penguji 3 berhasil menambahkan labirin ke 10 dengan menggunakan *pull request* seperti yang terlihat pada Gambar 5.17. Link untuk *file* teks yang dibuat penguji 3 : <https://github.com/GeneralDevilX/Snake360/pull/4/files>. Labirin yang dibuat oleh penguji 3 sudah sesuai ketentuan membuat labirin pada *file readme*. Gambar 5.18 merupakan tampilan labirin yang dibuat oleh penguji 3.



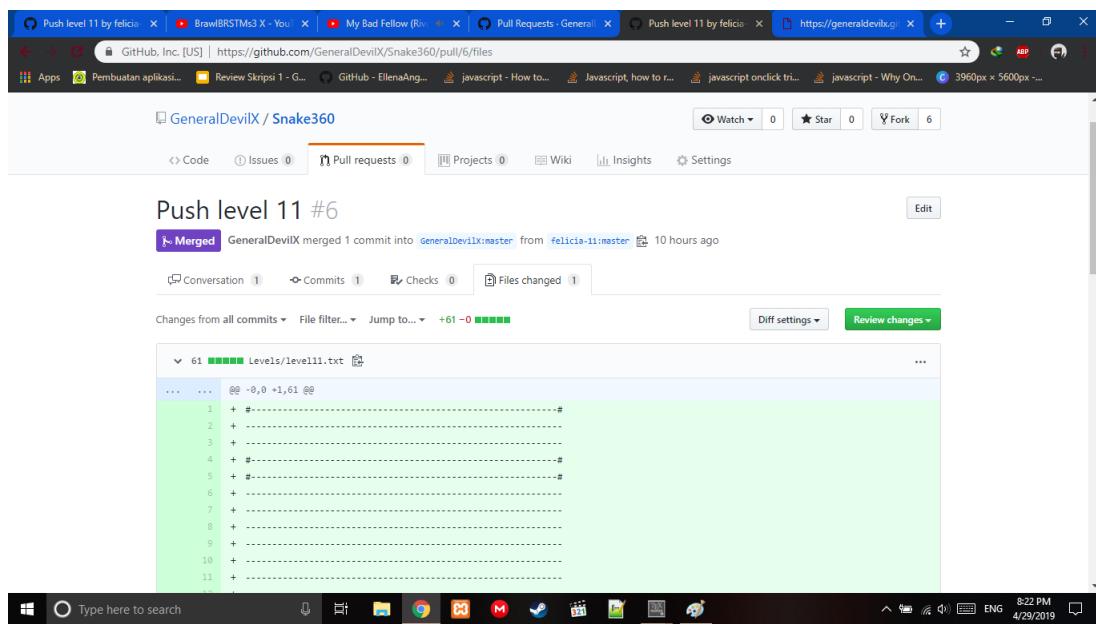
Gambar 5.17: Tampilan hasil pull request milik penguji 3



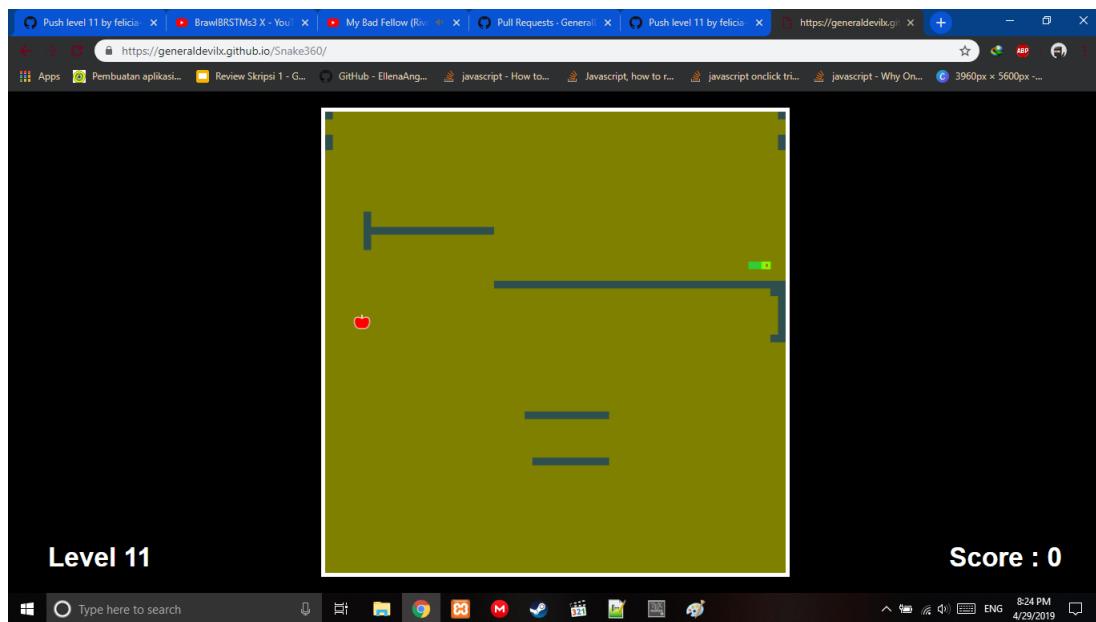
Gambar 5.18: Tampilan pengujian labirin yang dibuat oleh penguji 3

- #### 4. Pengaji 4 : felicia-11

Penguji 4 berhasil menambahkan labirin ke 11 dengan menggunakan *pull request* seperti yang terlihat pada Gambar 5.19. *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 5.20.



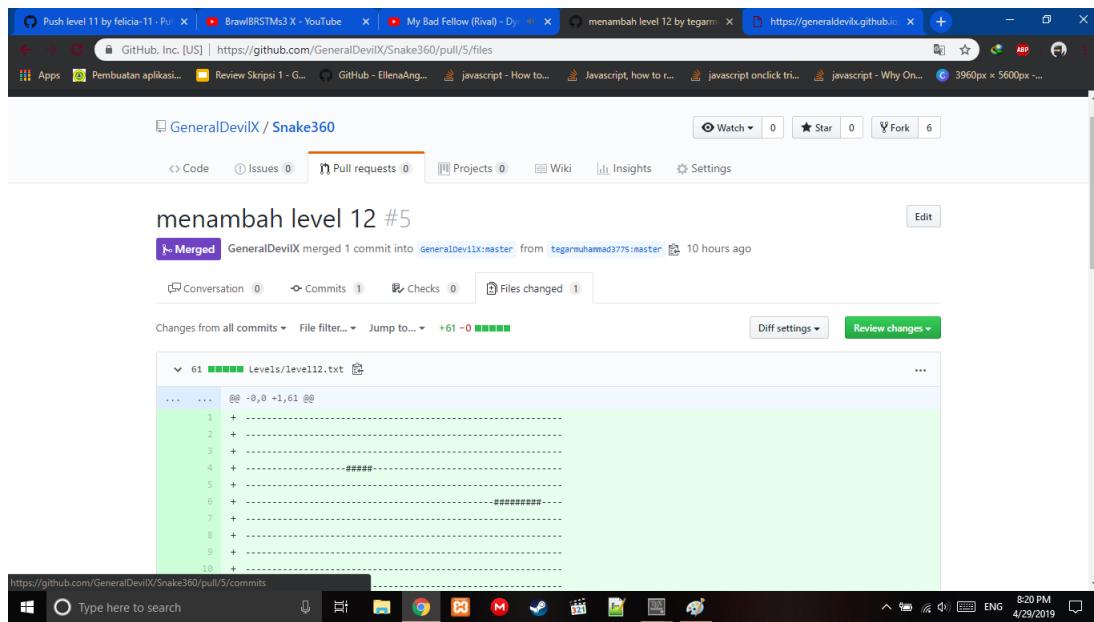
Gambar 5.19: Tampilan hasil *pull request* milik penguji 4



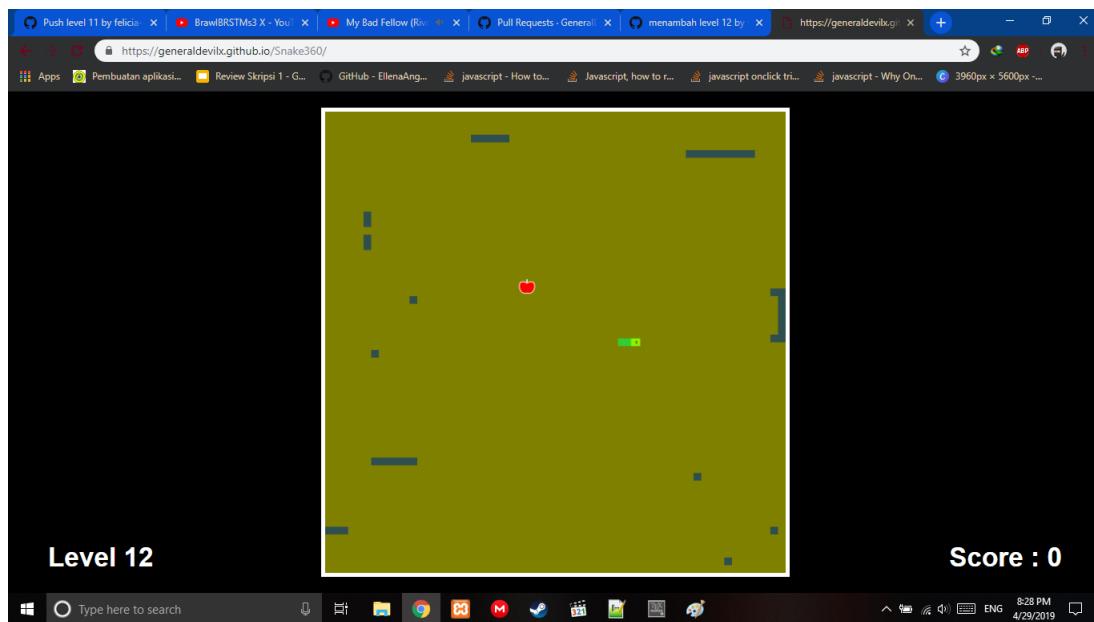
Gambar 5.20: Tampilan pengujian labirin yang dibuat oleh penguji 4

5. Penguji 5 : tegarmuhammad3775

Penguji 5 berhasil menambahkan labirin ke 12 dengan menggunakan *pull request* seperti yang terlihat pada Gambar 5.21. 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 pada Gambar 5.22.



Gambar 5.21: Tampilan hasil *pull request* milik penguji 5



Gambar 5.22: Tampilan pengujian labirin yang dibuat oleh penguji 5

BAB 6

KESIMPULAN DAN SARAN

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

6.1 Kesimpulan

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

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

6.2 Saran

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

1. *Format* labirin harus dibuat lebih mudah untuk mengurangi kesalahan pada pembuatan labirin terutama pada memposisikan ular. Seharusnya, posisi ular tidak perlu ditulis dalam koordinat melainkan dituliskan langsung menggunakan sebuah simbol pada labirin.
2. Sebelum memulai permainan, sebaiknya terdapat preview untuk setiap labirin, sehingga pemain tidak perlu memulai permainan terlebih dahulu untuk melihat labirin yang dipilih.

DAFTAR REFERENSI

- [1] Fogel, K. (2005) *Producing open source software: How to run a successful free software project.* " O'Reilly Media, Inc. ".
- [2] Fulton, S. dan Fulton, J. (2013) *HTML5 canvas: native interactivity and animation for the web.* " O'Reilly Media, Inc.".
- [3] MDN (2005) Web technology for developers. <https://developer.mozilla.org/en-US/docs/Web>. 17 Oktober 2018.
- [4] Duckett, J. (2014) *JavaScript and JQuery: interactive front-end web development.* Wiley Publishing.
- [5] 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             this.moveApple(x,y);
156
157             let arrayLayout = this.maze.getMazeLayout();
158             let dindingX = Math.floor(x/10);
159             let dindingY = Math.floor(y/10);
160             let dindingX2 = Math.floor((x+besarApel)/besarApel);
161             let dindingY2 = Math.floor((y+besarApel)/besarApel);
162             let dindingX3 = Math.floor((x+besarApel)/besarApel);
163             let dindingY3 = Math.floor((y+besarApel)/besarApel);
164             let dindingX4 = Math.floor((x+besarApel)/besarApel);
165             let dindingY4 = Math.floor((y+besarApel)/besarApel);
166             let dindingX5 = Math.floor((x+besarApel)/besarApel);
167             let dindingY5 = Math.floor((y+besarApel)/besarApel);
168             let dindingX6 = Math.floor((x+besarApel)/besarApel);
169             let dindingY6 = Math.floor((y+besarApel)/besarApel);
170             let dindingX7 = Math.floor((x+besarApel)/besarApel);
171             let dindingY7 = Math.floor((y+besarApel)/besarApel);
172             let dindingX8 = Math.floor((x+besarApel)/besarApel);
173             let dindingY8 = Math.floor((y+besarApel)/besarApel);
174         }

```

```

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

```

```

274 }
275
276 // ular memakan apel
277 this.appleEaten = function(){
278     this.randomApple();
279     score++;
280     this.ular.panjangTubuhSaatIni++;
281     $('#score').html(score);
282 }
283
284 //ular menabrak diri sendiri
285 this.snakeCollision = function(){
286     let posisiXUlar = this.ular.tubuhSnake[0].x;
287     let posisiYUlar = this.ular.tubuhSnake[0].y;
288     const besarBoundary = 1;
289
290     for(var i = 1; i< this.ular.tubuhSnake.length;i++){
291         let bagianTubuhSnakeX = this.ular.tubuhSnake[i].x;
292         let bagianTubuhSnakeY = this.ular.tubuhSnake[i].y;
293
294         if(posisiXUlar > bagianTubuhSnakeX-besarBoundary && posisiYUlar > bagianTubuhSnakeY-besarBoundary &&
295             posisiXUlar < bagianTubuhSnakeX+besarBoundary && posisiYUlar < bagianTubuhSnakeY+besarBoundary ){
296             this.gameOver = true;
297             this.popUpGameOverText();
298         }
299     }
300 }
301
302 //ular menabrak dinding
303 this.wallCollision = function(){
304     let arrayLayout = this.maze.getMazeLayout();
305     let posisiXUlar = this.ular.tubuhSnake[0].x;
306     let posisiYUlar = this.ular.tubuhSnake[0].y;
307     let boundaryWallX = Math.floor(posisiXUlar/10);
308     let boundaryWallY = Math.floor(posisiYUlar/10);
309
310     if(arrayLayout[boundaryWallY].charAt(boundaryWallX) == '#'){
311         this.gameOver = true;
312         this.popUpGameOverText();
313     }
314 }
315
316 //untuk menampilkan text gameOver
317 this.popUpGameOverText = function(){
318     let gameOverText = document.getElementById("gameOver");
319     $("#finalScore").html(score);
320     gameOverText.style.visibility = "visible";
321 }
322
323
324
325 //mendapatkan status gameOver
326 this.getGameOverStatus = function(){
327     return this.gameOver;
328 }
329
330 //posisi untuk animasi
331 this.animate = function(){
332     if(this.gameOver == true){
333
334     } else{
335         this.context.clearRect(0,0,this.myCanvas.width,this.myCanvas.height);
336
337         for(var i = 1;i<=this.ular.getSpeed();i++){
338             this.ular.move(this.sudut);
339             this.sampaiUjung();
340
341             this.temp = {};
342             this.temp['x'] = this.ular.x;
343             this.temp['y'] = this.ular.y;
344
345             this.ular.tubuhSnake.unshift(this.temp);
346             if(this.ular.tubuhSnake.length > this.ular.panjangTubuhSaatIni){
347                 this.ular.tubuhSnake.pop();
348             }
349         }
350
351
352         this.drawingObj.drawSnake(this.ular.tubuhSnake,this.ular.besarUlar);
353         this.drawingObj.drawApple(this.apel.x,this.apel.y,this.apel.getBesarApel());
354         this.wallCollision();
355         this.appleCollision();
356         this.snakeCollision();
357
358         var that = this;
359         setTimeout(function(){
360             that.animate();
361         },50);
362     }
363 }
364
365 this.changeLevel = function(levels){
366     $("#totalLevel").html(levels);
367     return levels;
368 }
369
370 this.countLevels = function(level){
371     var that = this;
372     url = "Levels/level"+level+".txt";

```

```

373         $.get( url, function() {
374             level+=1;
375             that.countLevels(level);
376         })
377         .fail(function() {
378             level = level-1;
379             that.changeLevel(level);
380         });
381     }
382 
383     this.loadMaze = function(callback){
384         let level = $('#level').val();
385         let url = "Levels/level"+level+".txt";
386         let temp = this;
387 
388         $.ajax({
389             url: "Levels/level"+level+".txt",
390             dataType: 'text',
391             context: temp,
392             success: function(data,textStatus,jqXHR) {
393                 temp.maze.setMazeLayout(jqXHR.responseText);
394                 callback(temp);
395             }
396         });
397     }
398 
399 }
400 
401 $(document).ready(function(){
402     var snakeSpeedValid = true;
403     var turningSpeedValid = true;
404     var levelValid = true;
405 
406     resize();
407     var permainan = new Game();
408     permainan.countLevels(1);
409 
410     function resize(){
411         var width = window.innerWidth-50;
412         var height = window.innerHeight-50;
413         var snake = $('#snake');
414         var maze = $('#maze');
415         var menu = $('#menubdiv');
416 
417         if(width < 600 || height < 600){
418             if(width > height){
419                 maze.css('width', height);
420                 maze.css('height',height);
421                 snake.css('width', height);
422                 snake.css('height',height);
423             }
424             if(height > width){
425                 maze.css('height',width);
426                 maze.css('width',width);
427                 snake.css('height',width);
428                 snake.css('width',width);
429             }
430         }
431         else{
432             maze.css('width', '600px');
433             maze.css('height','600px');
434             snake.css('width', '600px');
435             snake.css('height','600px');
436         }
437         var widthMenu = snake.width()-75;
438         var heightMenu = snake.height()-75;
439 
440         menu.css('width',widthMenu);
441         menu.css('height',heightMenu);
442 
443         menu.css('font-size',menu.width()/25);
444         $('#scoreText').css('font-size',menu.width()/15);
445         $('#levelText').css('font-size',menu.width()/15);
446         $('#gameOver').css('font-size',menu.width()/15);
447     }
448 }
449 
450     window.addEventListener('resize',resize,false);
451     window.addEventListener('orientationchange',resize,false);
452 
453     document.getElementById('level').addEventListener('change',function(){
454         let chosenLevel = document.getElementById("level").value;
455         let temp = $("#totalLevel").html();
456         let totalLevel = parseInt(temp);
457 
458         if(chosenLevel != ""){
459             if(chosenLevel > 0 && chosenLevel <= totalLevel){
460                 document.getElementById("levelInvalid").style.visibility = "hidden";
461                 levelValid = true;
462             }
463             else{
464                 document.getElementById("levelInvalid").style.visibility = "visible";
465                 levelValid = false;
466             }
467         }
468         else{
469             document.getElementById("levelInvalid").style.visibility = "visible";
470             levelValid = false;
471         }
472     })
473 }

```

```

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

```

```

571         wait();
572     });
573   });
574 });
575 </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 }

```

Listing A.3: Apple.js

```

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

```

Listing A.4: Maze.js

```

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

```

Listing A.5: DrawingObject.js

```

1 function DrawingObject(context,contextMaze){
2     this.context = context;
3     this.contextMaze = contextMaze;
4
5     //untuk gambar apel
6     this.drawApple = function(x,y,besarApel){
7         this.context.lineWidth = 1;
8         this.context.strokeStyle = 'white';
9         this.context.beginPath();
10        this.context.moveTo(x+(besarApel/2),y+5);
11        this.context.quadraticCurveTo(x+besarApel,y,x+besarApel,y+(besarApel/2));
12        this.context.quadraticCurveTo(x+(besarApel-2),y+besarApel,x+(besarApel/2),y+(besarApel-2));
13        this.context.quadraticCurveTo(x+2,y+besarApel,x,y+(besarApel/2));
14        this.context.quadraticCurveTo(x,y,x+(besarApel/2),y+5);
15        this.context.closePath();

```

```
16    this.context.fillStyle = 'red';
17    this.context.fill();
18    this.context.moveTo(x+10,y+5);
19    this.context.lineTo(x+10,y);
20    this.context.closePath();
21    this.context.stroke();
22}
23
24//untuk gambar ular
25this.drawSnake = function(arrayUlar,besarUlar){
26    this.context.lineWidth = besarUlar;
27    this.context.strokeStyle = 'lawngreen';
28
29    for(var i = 0;i< arrayUlar.length-1;i++){
30        if(Math.abs(arrayUlar[i].x - arrayUlar[i+1].x) > 2 ||
31            Math.abs(arrayUlar[i].y - arrayUlar[i+1].y) > 2){
32            i++;
33        }
34        else{
35            this.context.lineWidth = besarUlar;
36            this.context.strokeStyle = 'lawngreen';
37
38            if(i > 5){
39                this.context.strokeStyle = 'limegreen';
40            }
41
42            this.context.beginPath();
43            this.context.moveTo(arrayUlar[i].x,arrayUlar[i].y);
44            this.context.lineTo(arrayUlar[i+1].x,arrayUlar[i+1].y);
45            this.context.closePath();
46            this.context.stroke();
47
48            if(i == 2){
49                this.context.strokeStyle = 'red';
50                this.context.lineWidth = 3;
51                this.context.beginPath();
52                this.context.moveTo(arrayUlar[i].x,arrayUlar[i].y);
53                this.context.lineTo(arrayUlar[i+1].x,arrayUlar[i+1].y);
54                this.context.closePath();
55                this.context.stroke();
56            }
57        }
58    }
59}
60
61}
62
63//untuk gambar maze
64this.drawMaze = function(arrayLayout,besarDinding){
65    this.contextMaze.lineWidth = besarDinding;
66    this.contextMaze.fillStyle = 'darkslategrey';
67
68    for(var i = 0;i< arrayLayout.length; i++){
69        let temp = arrayLayout[i];
70        for(var j = 0;j< arrayLayout.length;j++){
71            if(temp.charAt(j) == '#'){
72                this.contextMaze.fillRect(j*besarDinding,i*besarDinding,besarDinding,besarDinding);
73            }
74        }
75    }
76}
77}
```

LAMPIRAN B

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
```