

SKRIPSI

PEMANFAATAN SMARTPHONE SEBAGAI PENGENDALI PERMAINAN BERBASIS
WEB



Priambodo Pangestu

NPM: 2013730055

PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI DAN SAINS
UNIVERSITAS KATOLIK PARAHYANGAN
«tahun»

UNDERGRADUATE THESIS

«JUDUL BAHASA INGGRIS»



Priambodo Pangestu

NPM: 2013730055

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

«tahun»

LEMBAR PENGESAHAN

**PEMANFAATAN SMARTPHONE SEBAGAI PENGENDALI PERMAINAN BERBASIS
WEB**

Priambodo Pangestu

NPM: 2013730055

Bandung, «tanggal» «bulan» «tahun»

Menyetujui,

Pembimbing Utama

Pembimbing Pendamping

Pascal Alfadian, M.Comp.

Ketua Tim Penguji

Anggota Tim Penguji

«penguji 1»

«penguji 2»

Mengetahui,

Ketua Program Studi

Mariskha Tri Adithia, P.D.Eng

PERNYATAAN

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

PEMANFAATAN SMARTPHONE SEBAGAI PENGENDALI PERMAINAN BERBASIS WEB

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 dijatuahkan 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 «**tanggal**» «**bulan**» «**tahun**»

Meterai
Rp. 6000

Priambodo Pangestu
NPM: 2013730055

ABSTRAK

«Tuliskan abstrak anda di sini, dalam bahasa Indonesia»

 Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Kata-kata kunci: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Indonesia»

ABSTRACT

«Tuliskan abstrak anda di sini, dalam bahasa Inggris»

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Keywords: «Tuliskan di sini kata-kata kunci yang anda gunakan, dalam bahasa Inggris»

«kepada siapa anda mempersembahkan skripsi ini...?»

KATA PENGANTAR

«Tuliskan kata pengantar dari anda di sini ... »

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Bandung, «bulan» «tahun»

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	2
1.3 Tujuan	2
1.4 Batasan Masalah	2
1.5 Metodologi	2
1.6 Sistematika Pembahasan	3
2 LANDASAN TEORI	5
2.1 Node.js	5
2.1.1 HTTP	5
2.1.2 Path	5
2.1.3 Module	6
2.2 Express.js	6
2.2.1 express()	6
2.2.2 Application	7
2.2.3 Response	8
2.3 WebSockets	8
2.4 Socket.io	9
2.4.1 Server API	9
2.4.2 Client API	11
2.5 Canvas API	12
2.5.1 Animation	13
2.5.2 canvasRenderingContext2D	14
3 ANALISIS	17
3.1 Analisis Aplikasi Sejenis	17
3.2 Analisis Sequence Diagram	26
3.3 Analisis Pemanfaatan Socket.io	27
4 PERANCANGAN	29
4.1 Perancangan Sequence Diagram	29
4.1.1 Sequence Diagram Join Room	30
4.1.2 Sequence Diagram Choose Character	31
4.1.3 Sequence Diagram Game Begin	32

4.1.4	Sequence Diagram Winning Page	33
4.2	Perancangan Antarmuka	34
4.3	Perancangan Struktur Direktori	40
DAFTAR REFERENSI		41
A KODE PROGRAM		43
B HASIL EKSPERIMEN		45

DAFTAR GAMBAR

3.1 Halaman awal web pada <i>PC browser</i>	17
3.2 Kode yang harus dimasukan oleh pemain pada <i>mobile browser</i>	18
3.3 Halaman awal pada <i>mobile browser</i>	19
3.4 Pemain diminta untuk memasukan kode yang sudah didapatkan pada <i>PC browser</i>	20
3.5 Pemain diminta untuk memasukan kode yang sudah didapatkan pada <i>PC browser</i>	21
3.6 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	22
3.7 Halaman pada <i>smartphone</i> yang berfungsi sebagai pengendali.	22
3.8 Halaman awal permainan The Neighborhood pada <i>PC</i>	23
3.9 Halaman awal permainan The Neighborhood pada <i>smartphone</i>	23
3.10 Halaman pada <i>PC</i> dimana permainan sedang berlangsung.	24
3.11 Halaman pada <i>smartphone</i> dimana permainan sedang berlangsung.	24
3.12 Halaman pada <i>PC</i> apabila permainan sudah dimenangkan.	25
3.13 Halaman pada <i>smartphone</i> apabila permainan sudah dimenangkan.	25
3.14 Halaman pada <i>PC</i> yang menunjukan pemutusan koneksi.	26
4.1 Proses melakukan koneksi ke <i>socket.io</i> dan bergabung kedalam <i>room</i>	30
4.2 Proses memilih karakter.	31
4.3 Proses memulai permainan.	32
4.4 Menampilkan para pemain yang telah selesai bermain.	33
4.5 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	34
4.6 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	35
4.7 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	36
4.8 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	36
4.9 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	37
4.10 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	37
4.11 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	38
4.12 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	38
4.13 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	39
4.14 Halaman pada <i>PC</i> yang menunjukan berbagai permainan yang dapat dipilih.	40
B.1 Hasil 1	45
B.2 Hasil 2	45
B.3 Hasil 3	45
B.4 Hasil 4	45

DAFTAR TABEL

BAB 1

PENDAHULUAN

1.1 Latar Belakang

WebSockets adalah teknologi yang memungkinkan *web browser* pengguna dan *web server* membuka sesi komunikasi interaktif satu sama lain. Teknologi *WebSockets* didesain untuk diimplementasikan pada *web browser* dan *web server*, tetapi dapat juga digunakan oleh setiap aplikasi *client* maupun *server*. *WebSockets* memiliki standar yang menyediakan cara agar *web server* dapat mengirim konten ke *web browser* tanpa diminta oleh *client*, dan memungkinkan agar pesan dikirimkan berulang-ulang dengan tetap menjaga koneksi yang terbuka. Oleh karena itu, protokol *WebSockets* memungkinkan interaksi antara *web browser* dan *web server* dengan *overhead* yang rendah, dan juga memfasilitasi transfer data *realtime* dari *server* maupun menuju *server*.

Salah satu teknologi yang memanfaatkan protokol *WebSockets* adalah *Socket.io*. Teknologi ini memungkinkan untuk melakukan komunikasi secara *realtime*, dan dua arah antara *client* dan *server*. *Socket.io* memiliki dua bagian: *client-side library* yang berjalan didalam *web browser*, dan *server-side library* yang berjalan pada bagian *server*. *Socket.io* memiliki fitur-fitur yang beragam, seperti melakukan broadcast ke beberapa *sockets*, dan menyimpan data yang berhubungan dengan masing-masing *client*. Teknologi ini sangat berguna untuk membantu membangun sebuah aplikasi yang membutuhkan koneksi *realtime* seperti dalam aplikasi *chatting* maupun *game*.

Untuk memanfaatkan teknologi *Socket.io* dalam membangun aplikasi permainan, akan dibutuhkan beberapa teknologi yang dapat membantu pembangunan aplikasinya. Salah satu teknologi tersebut yaitu *Canvas API*. Teknologi ini merupakan bagian dari *HTML5 element* yang dapat digunakan untuk menggambar suatu grafis melalui *JavaScript* secara *on the fly*. *Canvas API* dapat juga digunakan untuk membuat komposisi foto dan membuat animasi. Oleh karena itu, fungsi-fungsi yang ada pada *Canvas API* akan membantu pembangunan aplikasi permainan terutama pada bagian pengembangan grafis pada aplikasinya.

Teknologi lain yang dapat membantu membangun aplikasi permainan dalam memanfaatkan teknologi *Socket.io* adalah *Node.js*. Teknologi ini merupakan sebuah *platform* yang didesain untuk mengembangkan aplikasi berbasis web pada bagian *web server*. *Node.js* ditulis dalam sintaks bahasa pemrograman *JavaScript* dan menggunakan *V8* yang merupakan *engine JavaScript* milik perusahaan *Google* untuk mengeksekusi *JavaScript* pada *web server*. *Node.js* memiliki sifat *non-blocking*, yang berarti *Node.js* tidak akan menunggu untuk mengerjakan *request* selanjutnya. Fitur-fitur yang dimiliki oleh *Node.js* akan sangat membantu untuk membangun aplikasi permainan yang membutuhkan koneksi *real-time*.

Salah satu teknologi yang akan membantu dalam memanfaatkan *Node.js* adalah *Express.js*. Teknologi ini menyediakan kumpulan fitur untuk mengatur penyimpanan data lokal dalam membangun aplikasi web maupun *mobile*. *Express.js* hanya dapat digunakan untuk membangun aplikasi apabila aplikasi tersebut berjalan berdasarkan *Node.js*. Oleh karena itu, fitur-fitur yang dimiliki oleh *Express.js* akan membantu dalam pembangunan aplikasi berbasis *Node.js*.

Pada skripsi ini, akan dibuat sebuah aplikasi permainan yang memanfaatkan protokol *WebSockets*, dimana dalam penggunaan protokol tersebut akan dibantu dengan teknologi *Socket.io*. Selain itu, aplikasi yang dibuat akan memanfaatkan *personal computer (PC)* dan *smartphone* untuk

pengembangan aplikasinya. Para pemain akan mengkoneksikan *smartphone* pada suatu *PC* yang akan berfungsi sebagai *console*, dan *smartphone* tersebut akan berfungsi sebagai *controller* untuk memainkan permainannya. Oleh karena itu, protokol *WebSockets* akan digunakan sebagai koneksi antara *smartphone* dan *PC* dalam aplikasi permainan yang akan dibangun. Aplikasi permainan akan menggunakan teknologi berbasis web, sehingga untuk memainkannya, *client* harus mengakses melalui *web browser*.

1.2 Rumusan Masalah

1. Bagaimana membangun aplikasi permainan berbasis web dengan memanfaatkan protokol *WebSockets* untuk penggunaan *smartphone* sebagai pengendali permainan berbasis web ?
2. Berapa *latency* yang dihasilkan berdasarkan penggunaan protokol *WebSockets* ?

1.3 Tujuan

1. Mengetahui cara membangun aplikasi permainan berbasis web dengan memanfaatkan protokol *WebSockets* untuk penggunaan *smartphone* sebagai pengendali permainan berbasis web.
2. Mengetahui jumlah *latency* yang dihasilkan berdasarkan pemanfaatan protokol *WebSockets*.

1.4 Batasan Masalah

Batasan masalah yang dibuat terkait dengan penggerjaan skripsi ini adalah sebagai berikut:

1. Aplikasi permainan yang dibuat merupakan permainan *multiplayer* yang hanya bisa dimainkan oleh dua orang saja.

1.5 Metodologi

Metodologi yang dilakukan dalam penggerjaan skripsi ini adalah sebagai berikut:

1. Studi literatur mengenai :
 - *WebSockets* yang akan digunakan untuk koneksi antara *smartphone* dan *PC*.
 - *Socket.io* sebagai teknologi yang akan menggunakan *WebSockets* dalam pembangunan aplikasi.
 - *Canvas API* yang akan digunakan untuk antarmuka permainan.
 - *Node.js* sebagai *web server* dalam pembangunan aplikasi.
 - *Express.js* sebagai *Node.js framework* yang akan digunakan untuk mengatur penyimpanan data.
2. Menganalisis aplikasi sejenis.
3. Merancang antarmuka permainan pada *PC* dan *smartphone*. Antarmuka pada *PC* akan berbeda dengan yang ada di *smartphone*, karena *smartphone* akan bekerja sebagai *controller* dan *PC* akan bekerja sebagai *console*.
4. Menyusun cara bermain aplikasi permainan yang dibangun.
5. Mengimplementasi program aplikasi permainan berbasis web.
6. Menganalisis *latency* yang dihasilkan pada aplikasi.
7. Melakukan eksperimen dan pengujian yang melibatkan responden.

1.6 Sistematika Pembahasan

Setiap bab dalam skripsi ini memiliki sistematika penulisan yang dijelaskan kedalam poin-poin sebagai berikut:

1. Bab 1 : Pendahuluan

Membahas mengenai gambaran umum penelitian ini. Berisi tentang latar belakang, rumusan masalah, tujuan, batasan masalah, metode penelitian, dan sistematika penulisan.

2. Bab 2 : Dasar Teori

Membahas mengenai teori-teori yang mendukung berjalannya penelitian ini. Berisi tentang *WebSockets*, *Socket.io*, *Node.js*, *Express.js*, dan *Canvas API*.

3. Bab 3 : Analisis

Membahas mengenai analisa masalah.

4. Bab 4 : Perancangan

Membahas mengenai perancangan yang dilakukan sebelum melakukan tahapan implementasi.

5. Bab 5 : Implementasi dan Pengujian

Membahas mengenai implementasi dan pengujian yang telah dilakukan.

6. Bab 6 : Kesimpulan dan Saran

Membahas hasil kesimpulan dari keseluruhan penelitian ini dan saran-saran yang dapat diberikan untuk penelitian berikutnya.

BAB 2

LANDASAN TEORI

Pada bab ini akan dijelaskan landasan teori mengenai *Node.js*, *Express.js*, *WebSockets*, *Socket.io*, dan *Canvas API*.

2.1 Node.js

Node.js adalah *JavaScript runtime* yang dibangun berdasarkan *V8* yang merupakan *JavaScript engine* milik perusahaan *Google* [1]. *Node.js* memiliki model *event-driven*, dan *non-blocking I/O* yang membuat teknologi tersebut efisien dalam implementasinya. Teknologi ini menyediakan beberapa kelas yang berfungsi untuk mengimplementasi fitur-fitur yang dimiliki.

Beberapa kelas yang terdapat pada *Node.js* adalah sebagai berikut:

2.1.1 HTTP

Interfaces HTTP pada *Node.js* digunakan untuk menangani *request* dari protokol *HTTP* yang secara *native* sulit untuk digunakan. *Interface* ini akan menangani protokol *HTTP* dengan tidak melakukan *buffer* pada seluruh *request* atau *responses*.

Salah satu *Method* yang dimiliki oleh *HTTP* yaitu sebagai berikut:

- **http.createServer([options][,requestListener])**

Parameter:

– **options** objek-objek sebagai berikut:

- * **IncomingMessage** menentukan objek dari kelas *IncomingMessage* yang akan digunakan.
- * **ServerResponse** menentukan objek dari kelas *ServerResponse* yang akan digunakan.

– **requestListener** fungsi yang akan secara otomatis ditambahkan pada *event 'request'* milik kelas *http.Server*.

Kembalian: objek *http.Server*

Method ini akan membuat objek *http.Server* untuk menangani *request* dari *client* dan memberikan *response* kepada *client*. Fungsi yang diberikan pada *method* ini akan dipanggil satu kali setiap *request* dibuat kepada *server*.

2.1.2 Path

Modul *Path* menyediakan fungsi untuk mengatur akses suatu *file* dan direktori. Modul tersebut dapat diakses dengan cara sebagai berikut:

```
const path = require('path');
```

Salah satu method yang dimiliki oleh modul *Path* adalah :

- **path.join([...paths])**

parameter:

- **...paths**

tipe: **String**

Urutan suatu lokasi *file* yang akan digunakan.

kembalian: String

Method ini akan menggabungkan seluruh bagian-bagian *path* dengan menormalisasinya dan mengembalikan bentuk *path* yang menyeluruh.

2.1.3 Module

Pada aplikasi berbasis *Node.js*, setiap *file* yang terdapat dalam pembangunan aplikasi dianggap sebagai modul-modul yang terpisah satu sama lain. Variabel dan fungsi yang terdapat pada satu *file*, atau modul, hanya dapat digunakan pada satu lingkup modul tersebut. Suatu modul tidak dapat menggunakan variabel atau fungsi yang terdapat pada modul lainnya. Oleh karena itu, apabila variabel dan fungsi yang terdapat pada satu modul dapat digunakan oleh modul yang lain, diperlukan cara tertentu. Cara tersebut dapat dilakukan seperti berikut:

```
module.exports = functions
```

\OR

```
module.exports = object
```

module.exports merupakan objek yang dibentuk oleh sistem *Module*. Dengan menggunakan cara ini, suatu modul dapat berubah menjadi global dan dapat diakses oleh modul-modul lain.

2.2 Express.js

Express.js merupakan *framework* aplikasi web untuk *Node.js* [2]. *Express.js* menyediakan fitur-fitur untuk web dan aplikasi *mobile* agar dapat bertahan lama. Untuk dapat menggunakan *Express.js*, dapat dilakukan langkah sebagai berikut:

```
var express = require('express');
var app = express();
```

Dengan begitu, fitur-fitur yang terdapat pada *Express.js* dapat digunakan untuk pengembangan aplikasi tertentu.

Subbab-subbab berikut akan menjelaskan kelas-kelas yang terdapat pada *Express.js*.

2.2.1 express()

Untuk membuat aplikasi *Express*, langkah yang dilakukan adalah sebagai berikut:

```
const express = require('express');
const app = express();
```

Salah satu *method* yang dimiliki oleh fungsi *express()* yaitu sebagai berikut:

- **express.static(root, [,options])**

Parameter:

- **root**

tipe: **String**

Menentukan direktori *root* yang akan digunakan untuk menyediakan *static file*.

- **options** Objek-objek seperti berikut:

- * **dotfiles** menentukan bagaimana mengatasi suatu *dotfiles* (suatu *file* atau direktori yang dimulai dengan tanda ".").

Method ini akan menyediakan cara agar dapat menggunakan *static file* yang ada.

- **express.Router([options])**

Parameter:

- **options** merupakan parameter pilihan yang akan menentukan perilaku dari *server*. Parameter dapat berupa objek sebagai berikut:

- * **caseSensitive** membedakan huruf besar dan huruf kecil.

Method ini akan membuat objek *router* yang dapat digunakan dengan menambahkan *middleware* dan *method HTTP* seperti *get*, *post*, dan *put*.

2.2.2 Application

Kelas ini akan menangani berbagai proses yang terjadi dalam aplikasi *Express* seperti melakukan *routing* terhadap *HTTP requests*, mengatur *middleware*, *rendering* sebuah *HTML views*, dan mendaftarkan *template engine* tertentu. Untuk dapat melakukan fungsi-fungsi tersebut dapat dilakukan langkah berikut:

```
const express = require('express');
const app = express();
```

Baris pertama dari potongan kode tersebut berarti variabel *express* memanggil modul '*express*' agar dapat mengakses fungsi-fungsi yang ada pada modul tersebut. Sedangkan baris kedua, Objek *app* memanggil fungsi *express()* yang telah didapatkan dari variabel *express*.

Kelas ini memiliki beberapa *method* sebagai berikut:

- **app.set(name, value)**

Parameter:

- **name** nama tertentu yang dapat digunakan untuk menentukan perilaku dari suatu *server*.
- **value** nilai yang akan ditetapkan pada parameter *name*.

Kembalian: -

Method ini akan menetapkan suatu *value* tertentu pada parameter *name*.

- **app.use([path,] callback[, callback...])**

Parameter:

- **path** suatu *path* yang akan ditangani oleh *middleware*. Dapat berupa *string*, *path pattern*, atau *array* dari kombinasi *string* dan *path pattern*.
- **callback** merupakan fungsi *callback*, dimana fungsi tersebut dapat berupa fungsi *middleware*, kumpulan dari fungsi *middleware* (yang dipisahkan dengan menggunakan koma), fungsi *array of middleware*, atau kombinasi dari seluruh *item* tersebut.

Method ini akan menghubungkan *middleware* atau suatu fungsi tertentu dengan *path* yang sudah ditentukan. Dalam implementasi *method* ini, urutan penempatan pada baris kode sangat berpengaruh. Setelah *app.use()* dieksekusi, maka suatu *request* tidak akan mengeksekusi *middleware* yang ada dibawah baris kode *app.use()*.

- **app.listen(port, [hostname], [backlog], [callback])**

Parameter:

- **port** nomor yang akan dituju oleh server.
- **hostname** *string* yang diberikan pada gawai tertentu agar dapat dikenali. Parameter ini bersifat opsional.
- **backlog** nomor yang menentukan ukuran maksimal dalam antrian koneksi yang tertunda. Parameter ini bersifat opsional.
- **callback** merupakan fungsi *callback*, dimana fungsi tersebut dapat berupa fungsi *middleware*, kumpulan dari fungsi *middleware* (yang dipisahkan dengan menggunakan koma), fungsi *array of middleware*, atau kombinasi dari seluruh *item* tersebut. Parameter ini bersifat opsional.

Method ini akan menghubungkan suatu koneksi pada *host* dan *port* yang sudah ditentukan.

2.2.3 Response

Sebuah objek dari kelas *Response* akan merepresentasikan respon *HTTP* yang dikirim oleh *Express* pada saat menerima *HTTP request*.

Salah satu properti yang dimiliki oleh kelas ini adalah:

- **res.locals**

Objek yang berisi variabel lokal milik *response* yang berada dalam lingkup suatu *request* tertentu. Objek ini hanya tersedia selama waktu *request* atau *response* tertentu.

Beberapa *method* yang terdapat pada kelas *Response* adalah:

- **res.render(view[, locals][, callback])**

Parameter:

- **view** suatu *string* yang menunjukkan *path* dari suatu *view file*.
- **locals** suatu objek yang memiliki properti yang menunjukkan variabel lokal dari *view*.
- **callback** suatu fungsi *callback*.

Method ini berfungsi untuk merubah *view file* dan mengirim *file* tersebut kepada *client*.

- **res.status(code)**

Parameter:

- **code** kode status *HTTP*.

Method ini akan menetapkan kode status *HTTP* untuk respon.

2.3 WebSockets

WebSockets merupakan *Application Programming Interface (API)* yang memiliki kemampuan untuk membuka sesi komunikasi interaktif antara *browser* pengguna dan *server* [3]. Dengan *API* ini, pengguna dapat mengirim pesan ke *server* dan menerima respon tanpa harus melakukan *polling* pada *server* terlebih dahulu. Protokol *WebSockets* akan digunakan pada teknologi *Socket.io*.

2.4 Socket.io

Socket.io merupakan salah satu teknologi yang memanfaatkan protokol *WebSockets* [4]. Teknologi ini memungkinkan sebuah aplikasi untuk melakukan komunikasi dua arah secara *real-time*. *Socket.io* dapat dijalankan di setiap *platform*, *browser*, dan gawai.

Sebelum dapat menggunakan *socket.io*, *Node.js* harus sudah terinstall pada sistem komputer. Apabila hal tersebut sudah dilakukan, maka *socket.io* dapat diinstall dengan menggunakan *command line tools* atau sejenisnya dengan melakukan langkah seperti berikut:

```
npm install socket.io
```

Dengan begitu, aplikasi yang dibuat sudah dapat mengakses fitur-fitur yang dimiliki oleh *socket.io*.

Socket.io dibagi menjadi dua *API*, yaitu *Server API* dan *Client API*. Subbab-subbab berikut menjelaskan kelas-kelas yang dimiliki *Socket.io*.

2.4.1 Server API

Kelas-kelas yang ada pada *Server API* digunakan untuk menangani proses yang terjadi dalam *server*[5]. Kelas-kelas tersebut adalah sebagai berikut:

1. Server

Kelas ini merupakan inti untuk dapat menangani proses yang terjadi dalam *socket.io server*. Kelas ini memiliki konstruktor seperti berikut:

- **new Server(httpServer[, options])**

Parameter:

– **httpServer**

tipe: **http.Server**

Server yang akan dituju.

– **options**

tipe: **Object**

Parameter ini dapat berupa berbagai jenis objek. Objek-objek tersebut yaitu sebagai berikut:

* **path**

tipe: **String**

Nama dari path yang akan ditangkap oleh *server* (contoh: */socket.io*).

* **serveClient**

tipe: **Boolean**

Menunjukan apakah *server* akan melayani *file* dari *client* atau tidak.

Beberapa *method* yang dimiliki oleh kelas ini yaitu sebagai berikut:

- **server.listen(port[, options])**

Parameter:

– **port**

tipe: **Number**

Nomor yang akan digunakan untuk melakukan koneksi kepada *server*

– **options**

tipe: **Object**

Parameter ini dapat berupa berbagai jenis objek.

Method ini akan melakukan koneksi kepada *server* dengan menggunakan *port* yang terdapat pada parameter.

2. Namespace

Kelas ini merepresentasikan kumpulan *sockets* yang terhubung dalam lingkup yang diidentifikasi oleh nama *path*. *Client* akan selalu terhubung ke / (*namespace* utama), kemudian dapat terhubung ke *namespace* lain saat berada dalam koneksi yang sama.

Beberapa *method* yang dimiliki oleh kelas ini yaitu:

- **namespace.emit(eventName[, ...args])**

Berfungsi untuk memancarkan suatu *event* pada seluruh *clients* yang terhubung.

Parameter:

- **eventName**
tipe: **String**
Nama dari *event*.
- **args**
Argumen tambahan.

Contoh implementasi:

```
const io = require('socket.io')();
```

```
// akan memancarkan event pada namespace utama (/)
io.emit('an event sent to all connected clients');
```

- **namespace.to(room)**

Berfungsi untuk memancarkan *event* kepada *client* yang sudah bergabung dalam *room* tertentu.

Parameter:

- **room**
tipe: **String**
Nama dari *room*.

Kembalian: *namespace*.

Contoh implementasi:

```
const io = require('socket.io')();
const adminNamespace = io.of('/admin');
```

```
adminNamespace.to('level1').emit('an event', { some: 'data' });
```

3. Socket

Kelas ini merupakan kelas yang mendasar untuk berinteraksi dengan *browser* milik *clients*. *Socket* merupakan milik *namespace* tertentu dan menggunakan kelas *Client* untuk berkomunikasi. Dalam setiap *namespace*, dapat ditentukan suatu *room* yang dimana sebuah *socket* dapat bergabung atau keluar. Kelas ini pun merupakan turunan dari *EventEmitter* milik *Node.js*. Kelas ini melakukan *override* pada *method* *emit* milik *EventEmitter*, dan tidak memodifikasi *method* lain.

Beberapa properti yang dimiliki oleh kelas ini yaitu sebagai berikut:

- **socket.id**

Tanda pengenal unik untuk sesi saat ini, yang didapatkan dari kelas *Client*.

- **socket.rooms**

Objek yang menandakan *room* dari *client* saat ini.

Beberapa *method* yang dimiliki oleh kelas ini yaitu sebagai berikut:

- **socket.join(room[, callback])**

Berfungsi untuk menambah *client* ke *room*.

Parameter:

- **room**

tipe: **String**

Nama *room*.

- **callback**

tipe: **Function**

Fungsi *callback*.

Kembalian: *Socket*.

2.4.2 Client API

Client API digunakan untuk menangani proses pengaturan koneksi yang terjadi pada bagian *client*. Agar dapat menggunakan *API* yang tersedia, *client* harus menambahkan *url* pada *syntax script* didalam *html* yang bersangkutan. Hal tersebut dapat dilakukan seperti berikut:

```
<script src="/socket.io/socket.io.js"></script>
```

Kelas-kelas yang ada pada *Client API* yaitu sebagai berikut:

1. IO

Untuk dapat menggunakan fungsi yang ada pada *IO*, dapat dilakukan langkah seperti berikut:

```
// berfungsi untuk melayani file client
<script src="/socket.io/socket.io.js"></script>
```

Method yang dimiliki oleh kelas ini yaitu sebagai berikut:

- **io([url][, options])**

Berfungsi untuk membuat objek baru dari kelas *Manager* dengan *url*, dan akan menggunakan objek kelas *Manager* yang sudah ada untuk pemanggilan selanjutnya, apabila *multiplex* pada parameter *option* bernilai *true*. Objek *Socket* akan dikembalikan untuk *namespace* yang sudah ditentukan oleh nama *path* pada *URL*, dengan nilai *default* /.

Parameter:

- **url**

tipe: **String**

Nama *URL*.

- **options**

tipe: **Object**

Parameter ini dapat berupa beberapa jenis objek, seperti milik kelas *Manager*

Kembalian: *Socket*

2. Socket

Salah satu properti yang dimiliki oleh kelas ini yaitu:

- **socket.id**

Identifier unik yang dimiliki oleh satu *socket* untuk satu sesi. Properti ini akan mempunyai nilai segera setelah koneksi terhubung, dan akan diperbarui setelah melakukan koneksi ulang.

Salah satu *method* yang dimiliki oleh kelas ini yaitu:

- **socket.emit(eventName[, ...args][, ack])**

Berfungsi untuk memancarkan *event* kepada *socket* yang ditandai dengan nama dari *event* tersebut.

Parameter:

- **eventName**
tipe: **String**
Nama *event*.
- **args**
Argumen tambahan (opsional).
- **ack**
tipe: **Function**
Fungsi tambahan (opsional).

Contoh implementasi:

```
socket . emit ( ' ferret ' , ' tobi ' , ( data ) => {
    console . log ( data );
});
```

Beberapa *events* yang ada pada kelas ini yaitu sebagai berikut:

- **connect**

Akan dipancarkan apabila berhasil melakukan koneksi dan setelah melakukan koneksi ulang.

- **disconnect**

Akan dipancarkan apabila *server* memutus koneksi atau *client* memutus koneksi.

2.5 Canvas API

Canvas API merupakan salah satu elemen *HTML5* yang digunakan untuk membuat gambar grafis dalam aplikasi web [6]. Teknologi ini memiliki fitur untuk membuat komposisi foto dan membuat animasi. Untuk dapat menggunakan fitur-fitur yang ada pada *Canvas API*, langkah yang harus dilakukan adalah sebagai berikut:

1. Menambahkan *tag* `<canvas>` pada *file HTML*, dan menambahkan *id* yang akan digunakan pada *file JavaScript*.

```
<canvas id="canvas"></canvas>
```

2. Membuat variabel untuk mendapatkan konteks *rendering* dan fungsi-fungsi menggambar agar dapat menampilkan sesuatu pada `<canvas>`.

```
// Variabel yang akan menampilkan sesuatu
// pada <canvas> dengan id='canvas'
var canvas = document.getElementById('canvas');
```

```
// Variabel yang akan mendapatkan fungsi-fungsi menggambar
var ctx = canvas.getContext('2d');
```

Subbab-subbab berikut menjelaskan tentang beberapa *interface* dari *Canvas*.

2.5.1 Animation

Dengan menggunakan *JavaScript* untuk mengontrol elemen <canvas>, hal tersebut akan sangat membantu dalam membuat animasi interaktif. Subbab ini akan membahas tentang pembuatan animasi didalam *Canvas API*.

Langkah Dasar Animasi

Ada beberapa langkah yang dibutuhkan untuk menggambar suatu *frame*.

1. *Clear the canvas*

Gambar atau bentuk yang berada dalam *canvas* pada *state* sebelumnya harus dihapus terlebih dahulu. Cara yang paling mudah untuk melakukan hal tersebut adalah menggunakan *method* *clearRect()*.

2. *Save the canvas state*

Apabila akan dilakukan perubahan pada gaya, transformasi, atau hal lainnya yang mengakibatkan kondisi *canvas* saat ini, maka kondisi *canvas* yang asli harus disimpan terlebih dahulu. Hal tersebut dilakukan agar kondisi *canvas* yang asli digunakan setiap suatu *frame* digambar.

3. *Draw animated shapes*

Langkah ini melakukan proses *rendering* pada *frame* yang telah digambar.

4. *Restore the canvas state*

Apabila kondisi *canvas* telah disimpan sebelumnya, maka kondisi tersebut harus dikembalikan lagi sebelum menggambar *frame* baru.

Untuk dapat melihat suatu gambar atau bentuk yang bergerak didalam *canvas*, diperlukan cara menjalankan fungsi menggambar selama periode waktu tertentu. Berikut merupakan beberapa cara yang dapat dilakukan:

- **setInterval(func, delay[, param1, param2, ...])**

Parameter:

- **func**

Suatu fungsi yang akan dieksekusi setiap *delay milliseconds*.

- **delay**

Waktu dalam *milliseconds*. *Timer* akan melakukan *delay* diantara waktu selama menjalankan fungsi tertentu.

- **param1,...,paramN**

Parameter tambahan untuk fungsi apabila waktu *timer* telah habis.

Kembalian:

intervalID identifikasi unik milik suatu interval.

Method ini akan memanggil suatu fungsi atau mengeksekusi kode tertentu selama rentang waktu yang telah ditentukan. Pemanggilan fungsi atau eksekusi kode dilakukan dengan penundaan selama waktu *delay* tertentu.

- **clearInterval(intervalID)**

Parameter:

- **intervalID** identifikasi milik suatu interval yang akan diberhentikan.

Method ini akan memberhentikan suatu fungsi yang berjalan berulang-ulang selama rentang waktu tertentu, yang dipanggil oleh *method* **setInterval()**.

- **requestAnimationFrame(callback)**

Parameter:

- **callback**

Parameter fungsi yang akan dipanggil saat harus memperbarui animasi dan menggambar *frame* selanjutnya.

Method ini akan memberitahu *browser* bahwa akan dilakukan suatu animasi, dan melakukan *request* kepada *browser* untuk memanggil fungsi tertentu untuk memperbarui animasi sebelum melakukan gambar ulang selanjutnya.

2.5.2 canvasRenderingContext2D

Interface ini digunakan untuk menggambar persegi panjang, teks, gambar, dan objek-objek lain kedalam elemen *canvas*. *CanvasRenderingContext2D* menyediakan konteks *2D rendering* untuk suatu elemen *<canvas>*. Untuk mendapatkan objek dari *interface* ini, harus memanggil *getContext()* didalam elemen *<canvas>*, dengan memberi "2d" sebagai argumen. Berikut contoh penggunaannya :

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

Salah satu properti yang dimiliki oleh *interface* ini adalah:

- **CanvasRenderingContext2D.globalCompositeOperation**

Properti ini akan menetapkan tipe operasi komposisi yang akan digunakan saat menggambar suatu bentuk baru kedalam *canvas*. Tipe tersebut berupa *String* yang akan menentukan komposisi atau *blending mode* yang akan digunakan.

Berikut contoh tipe yang dapat digunakan:

- **destination-over** Bentuk yang baru akan digambar pada posisi dibelakang konten *canvas* yang telah ada sebelumnya.

Beberapa *method* yang dimiliki oleh *interface* ini adalah sebagai berikut:

- **CanvasRenderingContext2D.clearRect(x, y, width, height)**

Method ini akan menghapus gambar sebelumnya dengan membentuk suatu persegi. *Method* ini akan menggambar koordinat titik awal (*x*, *y*) dengan lebar dan tinggi yang sudah ditentukan oleh *width* dan *height*.

Parameter:

- **x**

Koordinat *x* yang menandakan titik awal persegi.

- **y**

Koordinat *y* yang menandakan titik awal persegi.

- **width**

Lebar persegi.

- **height**

tinggi persegi.

- **CanvasRenderingContext2D.drawImage(image, dx, dy)**

Parameter:

- **image** elemen yang akan digambar kedalam *context* tertentu.

- **dx** koordinat *x* pada *canvas* untuk menempatkan *image* di pojok kiri atas.
- **dy** koordinat *y* pada *canvas* untuk menempatkan *image* di pojok kiri atas.

Method ini akan menyediakan cara untuk menggambar suatu elemen *image* pada *canvas*.

- **CanvasRenderingContext2D.save()**

Method ini akan menyimpan seluruh *state* dari *canvas* dengan menaruh *state* tersebut kedalam suatu *stack* yang sudah diatur didalam elemen <*canvas*>.

- **CanvasRenderingContext2D.restore()**

Method ini akan mengembalikan *state* yang baru saja disimpan dengan mengeluarkannya dari tumpukan paling atas suatu *stack*. Apabila *stack* tersebut kosong, maka *method* ini tidak melakukan apapun.

BAB 3

ANALISIS

Pada bab ini akan dijelaskan mengenai analisis aplikasi sejenis yang menggunakan *smartphone* sebagai pengendali permainan berbasis web, analisis *sequence* diagram.

3.1 Analisis Aplikasi Sejenis

Salah satu aplikasi sejenis permainan berbasis web dengan memanfaatkan *smartphone* sebagai pengendali yaitu AirConsole. Aplikasi tersebut memanfaatkan teknologi *browser*, *smartphone*, *PC*, dan juga jaringan internet untuk dapat menggunakananya. Aplikasi ini dikembangkan oleh N-Dream AG.

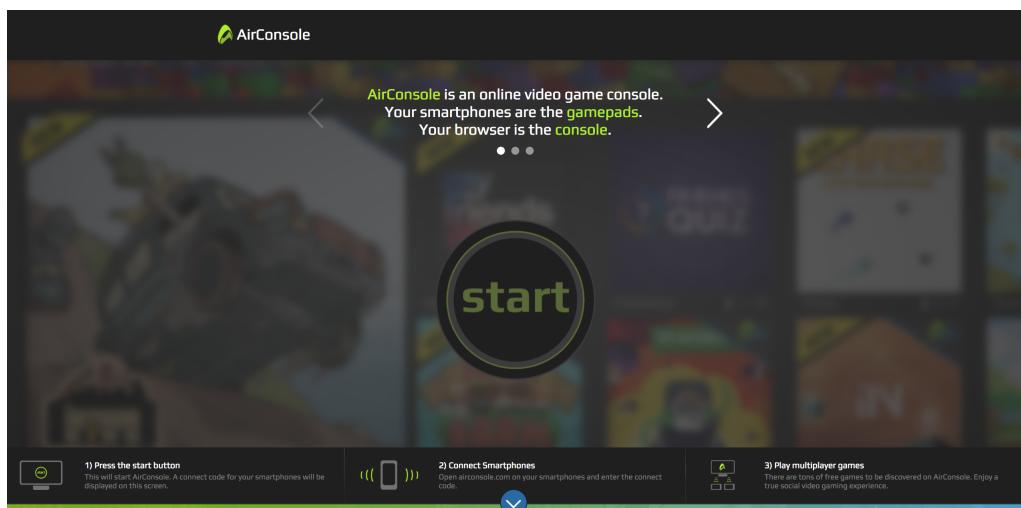
Analisis AirConsole

AirConsole merupakan permainan berbasis web dimana *browser* pada *mobile device* dapat melakukan koneksi ke *browser* pada *PC*. Pada aplikasi ini, terdapat berbagai macam permainan yang dapat dipilih oleh pemain. Untuk dapat memainkan aplikasi tersebut, pemain harus membuka alamat web *airconsole.com* pada *PC browser* dan juga pada *smartphone browser*.

Analisis dilakukan dengan cara berikut:

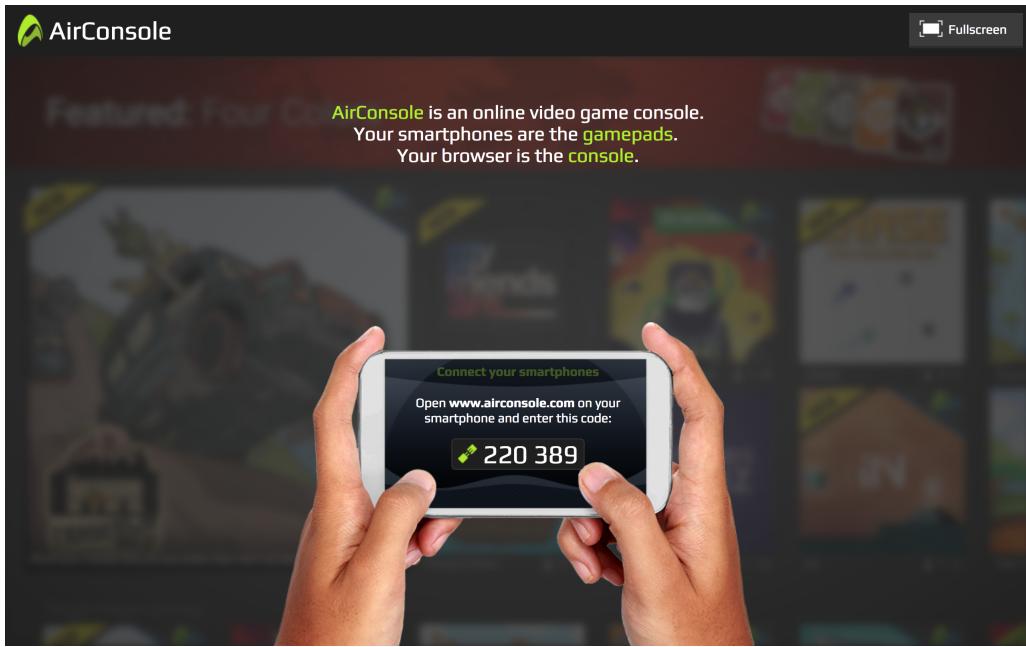
1. Memainkan permainan dari awal hingga akhir.
2. Keluar dari *browser* pada *PC* pada saat permainan berlangsung.
3. Keluar dari *browser* pada *smartphone* pada saat permainan berlangsung.

Pada halaman awal web di *PC*, pemain diminta untuk menekan tombol *start* yang ada pada gambar berikut:



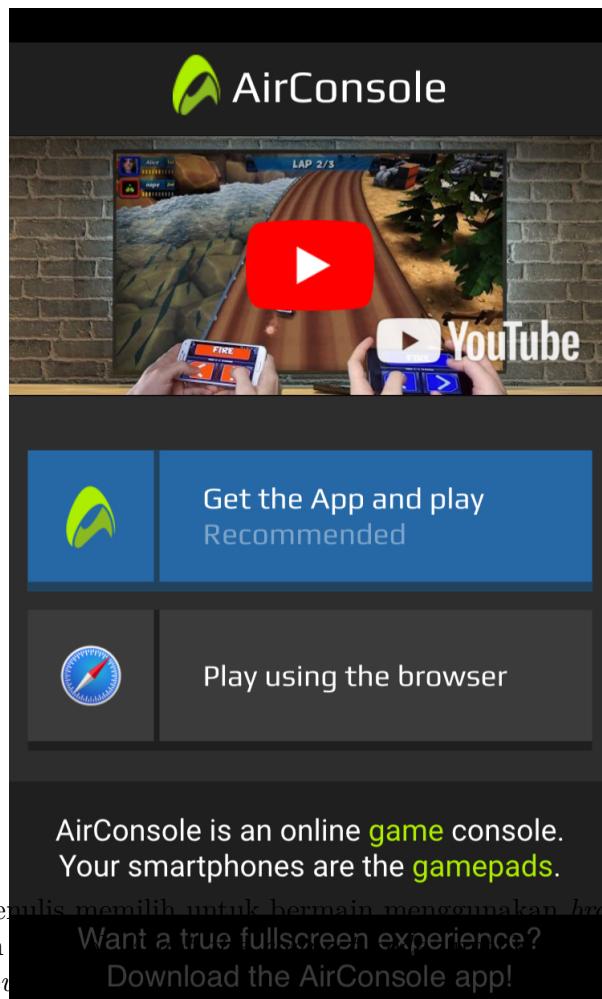
Gambar 3.1: Halaman awal web pada *PC browser*.

Setelah tombol *start* ditekan, maka akan muncul halaman berikutnya yang menunjukan kode yang harus dimasukan oleh pemain pada *mobile browser*.

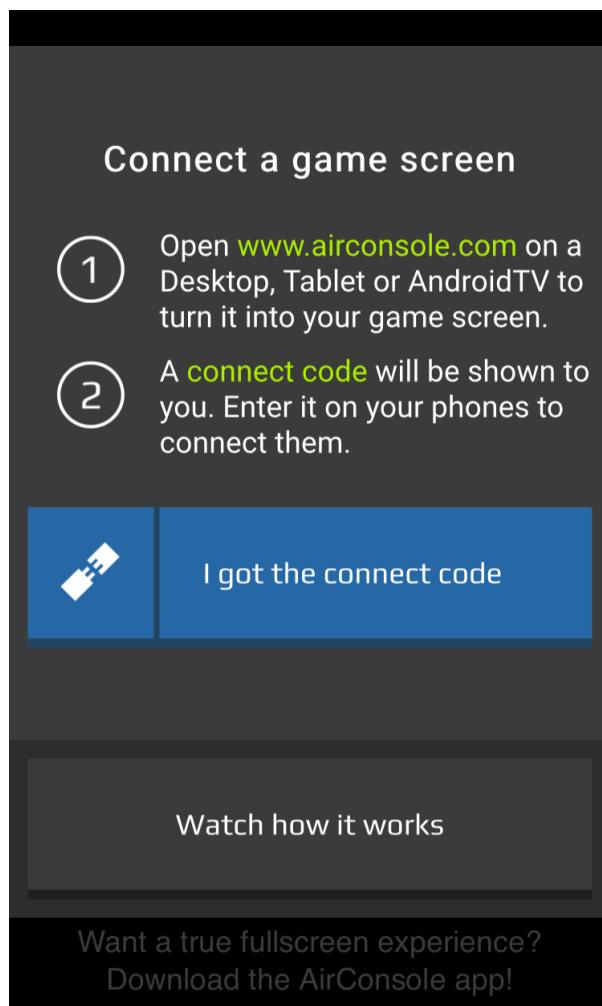


Gambar 3.2: Kode yang harus dimasukan oleh pemain pada *mobile browser*.

Pemain harus mengakses alamat web yang sama pada *mobile browser*. Pada halaman awal, pemain akan diminta untuk memilih apakah akan bermain dengan menggunakan aplikasi, atau bermain dengan menggunakan *browser*.

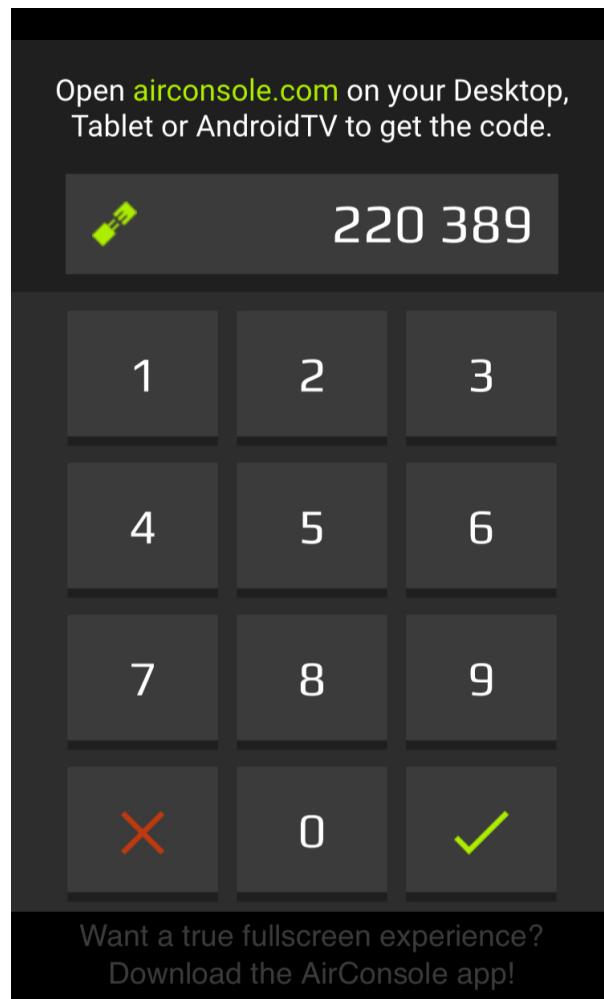


Gambar 3.3: Halaman awal pada *mobile browser*.



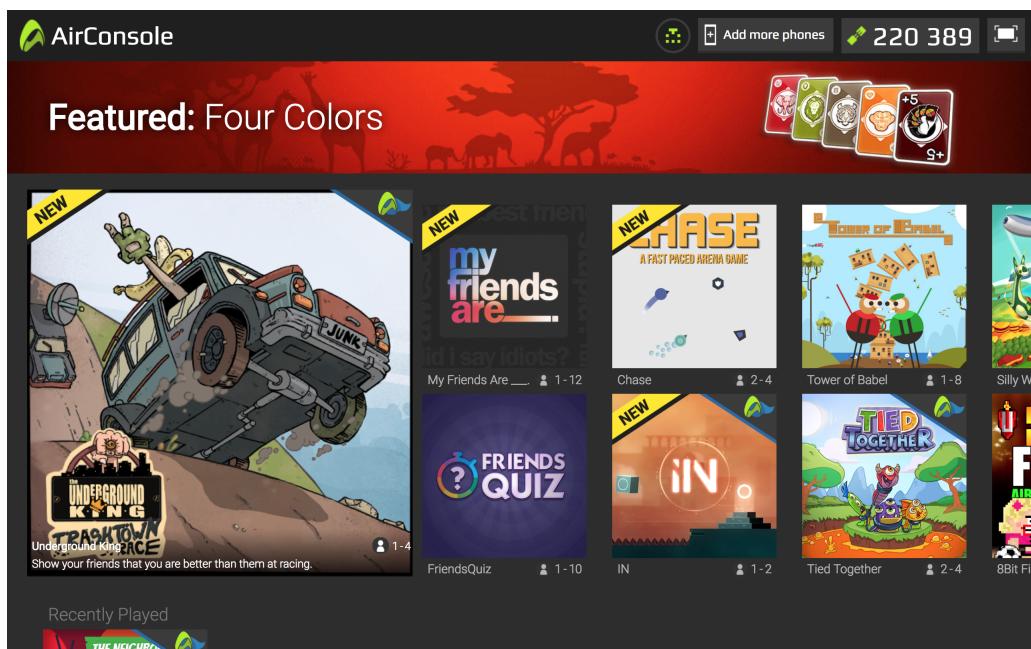
Gambar 3.4: Pemain diminta untuk memasukan kode yang sudah didapatkan pada *PC browser*.

Setelah menekan tombol tersebut, pemain dapat mulai memasukan kode yang sudah didapatkan. Kode ini bertujuan untuk proses otentikasi, sehingga para pemain yang dapat bermain dalam satu sesi yang sama, hanya para pemain yang mengetahui kode tersebut.

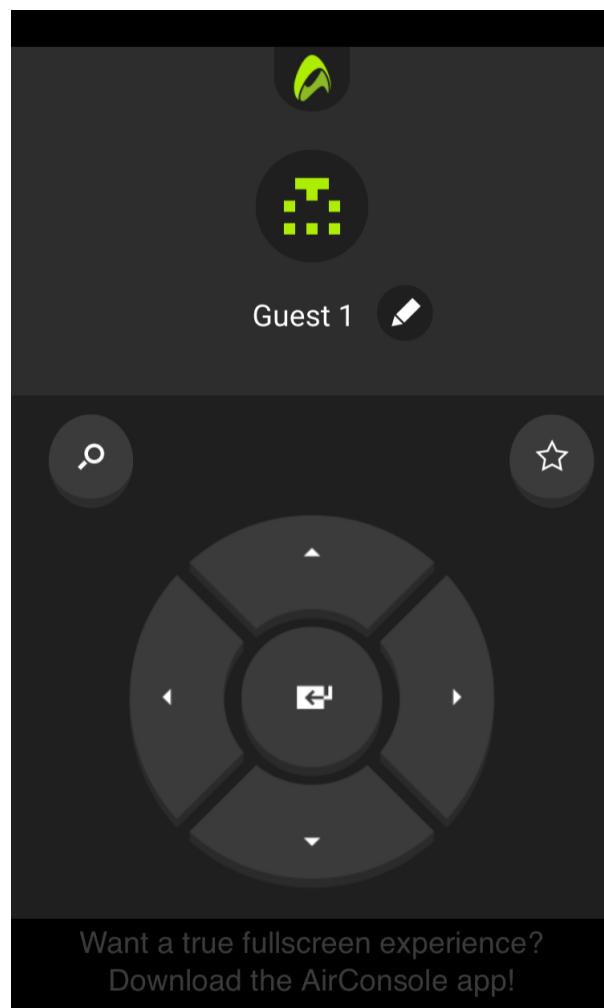


Gambar 3.5: Pemain diminta untuk memasukan kode yang sudah didapatkan pada *PC browser*.

Setelah pemain memasukan kode, maka halaman web di *PC* dan *smartphone* akan berubah. Pada *PC*, halaman akan menunjukan berbagai jenis permainan yang dapat dipilih. Pada *smartphone*, halaman akan berubah menjadi pengendali permainan, dimana pemain dapat menggerakan halaman yang ada di *PC* dengan menggunakan *smartphone*.

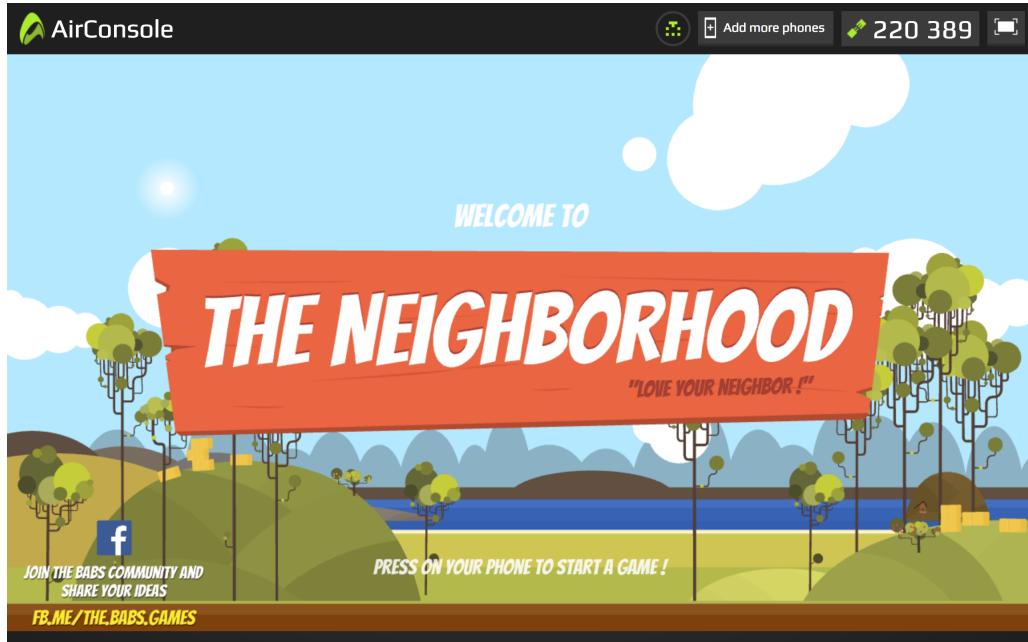


Gambar 3.6: Halaman pada *PC* yang menunjukan berbagai permainan yang dapat dipilih.

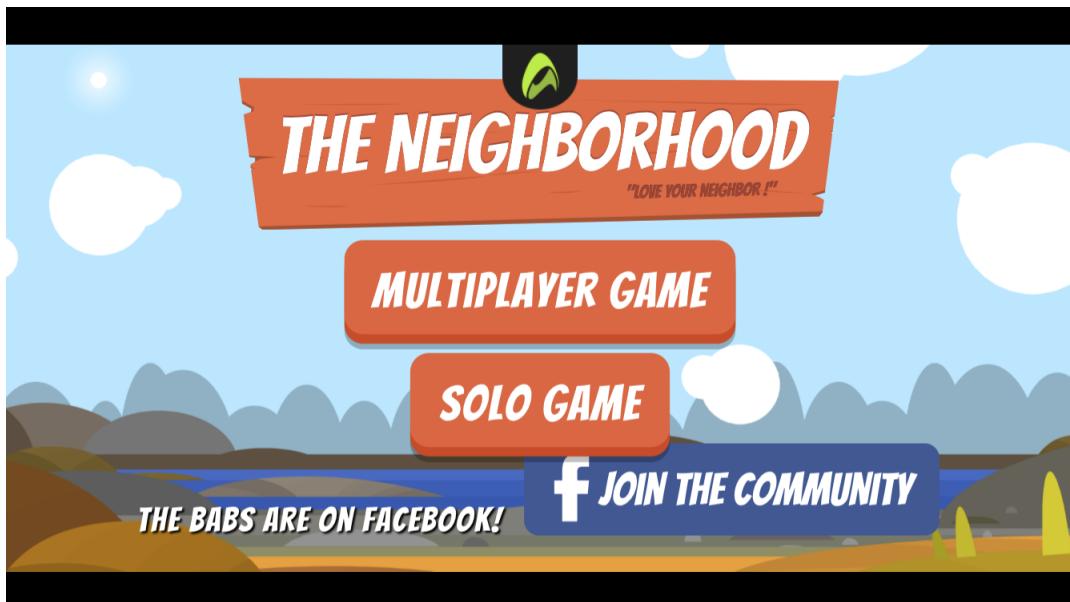


Gambar 3.7: Halaman pada *smartphone* yang berfungsi sebagai pengendali.

Dalam analisis ini, penulis memilih untuk memainkan permainan yang bernama *The Neighborhood*. Permainan ini sejenis permainan Angry Birds. Permainan ini bercerita tentang dua kelompok yang bertetangga, dimana kelompok tersebut bermusuhan dan berusaha untuk saling menghancurkan satu sama lain. Tujuan dari permainan ini yaitu lebih dulu menghancurkan anggota kelompok tetangga. Setelah memilih permainan tersebut, halaman pada *PC* dan *smartphone* akan berubah. Pada *smartphone*, pemain akan diminta untuk merubah mode tampilan *smartphone* menjadi *landscape*.



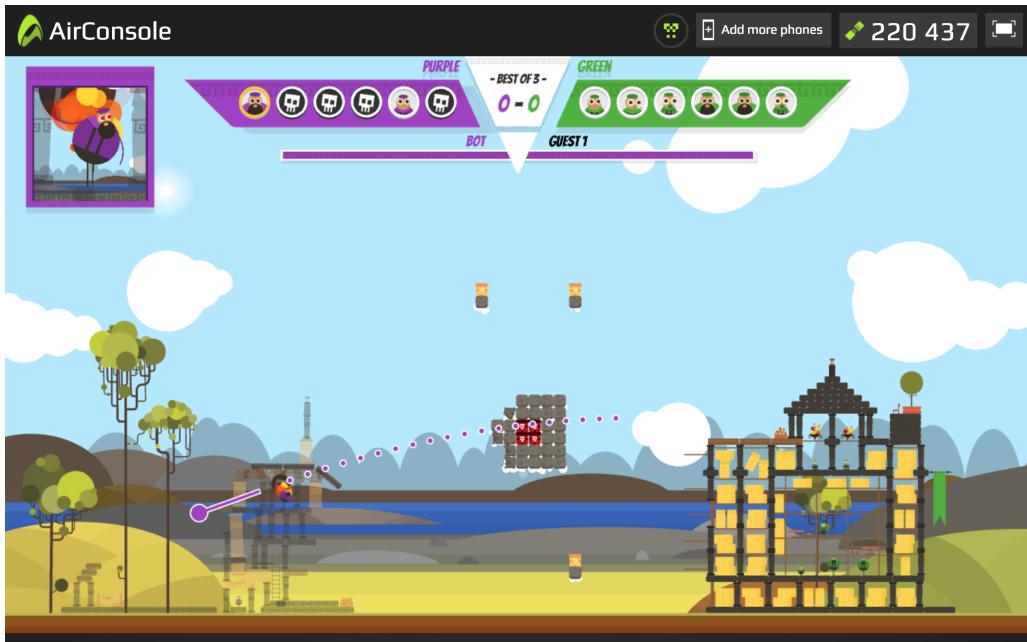
Gambar 3.8: Halaman awal permainan *The Neighborhood* pada *PC*.



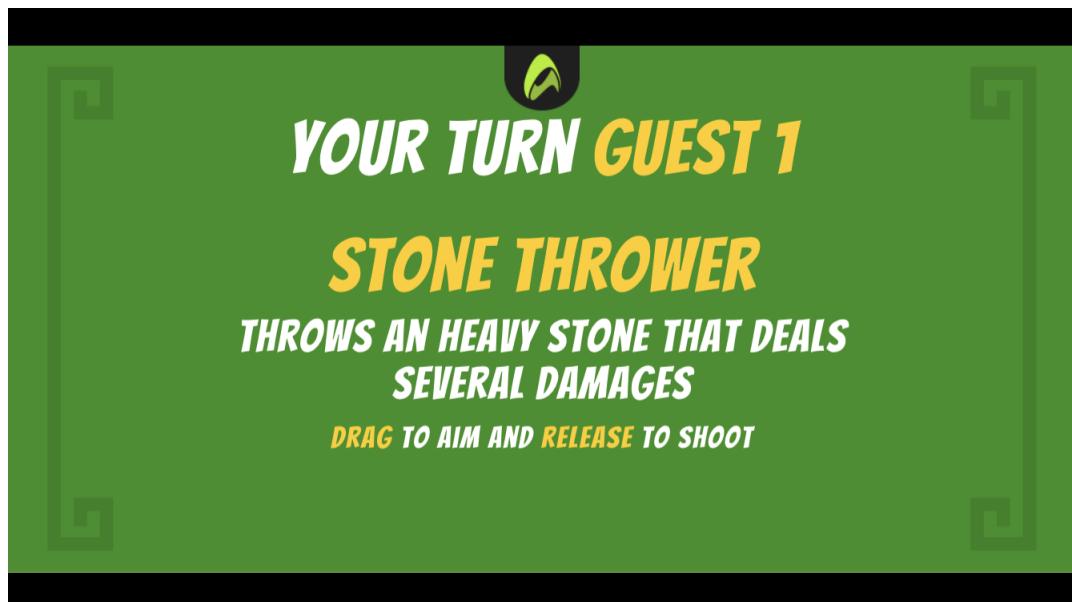
Gambar 3.9: Halaman awal permainan *The Neighborhood* pada *smartphone*.

Cara bermain dari permainan tersebut yaitu dengan menggunakan *smartphone*, dimana pemain harus menekan layar *smartphone*, kemudian menariknya sesuai dengan arah yang berlawanan dengan lawan, lalu melepas jari dari layar *smartphone* dengan tujuan untuk melempar suatu benda dari

ketapel. Semakin jauh pemain menarik, maka lontaran benda tersebut akan semakin kencang mengenai lawan.

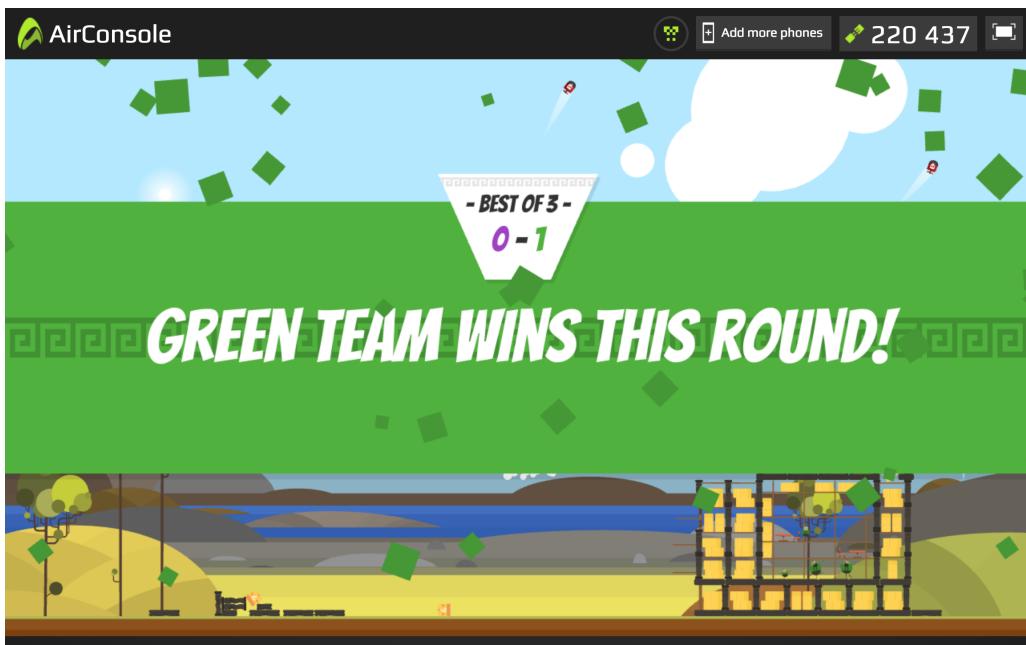


Gambar 3.10: Halaman pada *PC* dimana permainan sedang berlangsung.

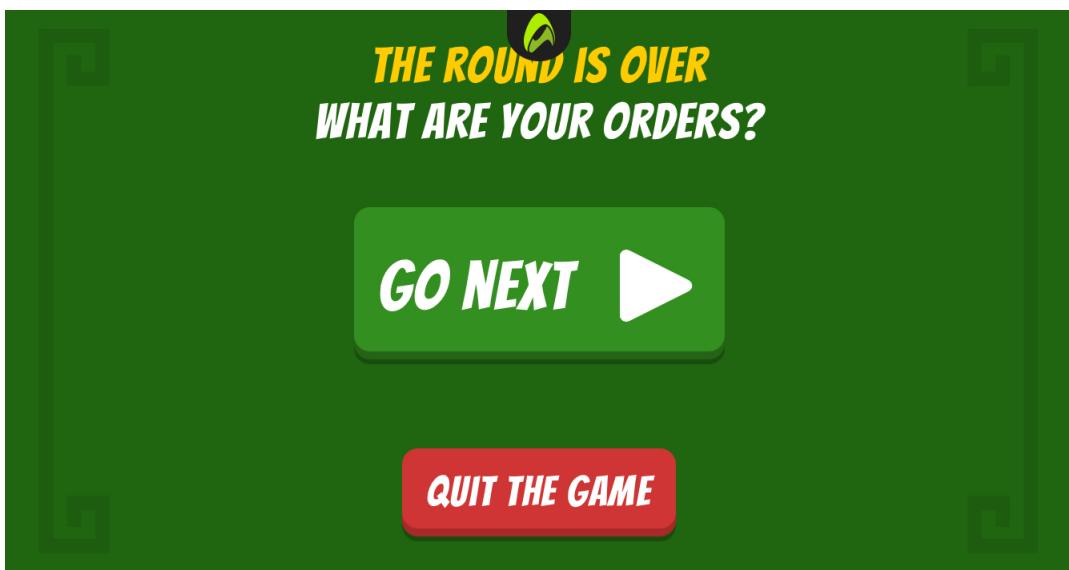


Gambar 3.11: Halaman pada *smartphone* dimana permainan sedang berlangsung.

Apabila memenangkan permainan tersebut, maka pemain dapat memilih untuk keluar dari permainan atau melanjutkan permainannya kembali.

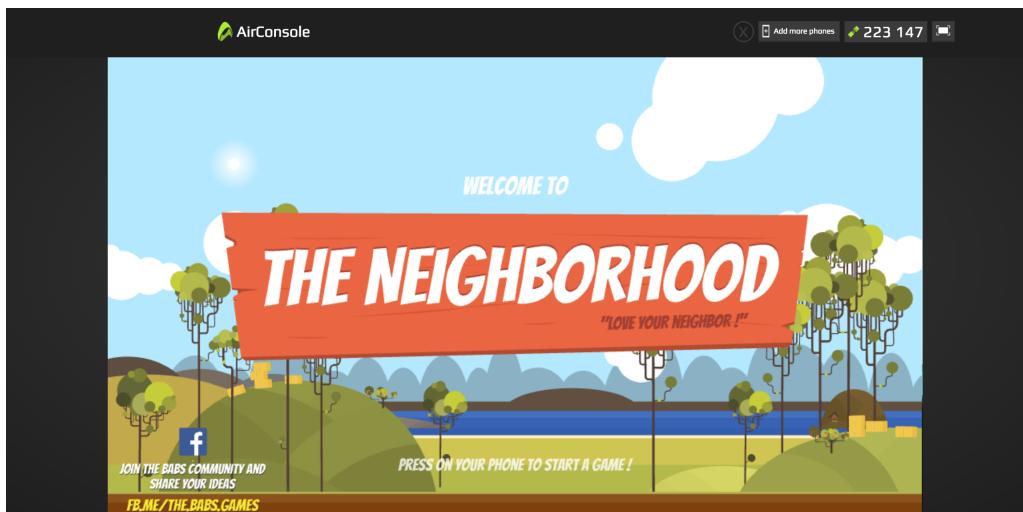


Gambar 3.12: Halaman pada *PC* apabila permainan sudah dimenangkan.



Gambar 3.13: Halaman pada *smartphone* apabila permainan sudah dimenangkan.

Dari ketiga percobaan yang sudah dilakukan, ada beberapa hal yang dapat diperbaiki dari permainan berbasis web tersebut. Percobaan pertama menunjukkan hasil yang bagus, dimana koneksi antara *smartphone* dan *PC* tidak putus saat permainan berlangsung, dan juga tidak ada keterlambatan antara gerakan pada *smartphone* dan *PC*. Pada percobaan kedua, apabila *browser* pada *PC* ditutup pada saat permainan berlangsung, maka koneksi akan terputus. Tetapi, tampilan pada *smartphone* tidak menunjukkan bahwa adanya koneksi yang terputus, sehingga pemain tidak mengetahui apakah permainan masih dapat berlangsung atau tidak. Tampilan hanya akan langsung kembali pada halaman awal permainan. Begitupun dengan percobaan ketiga, apabila *browser* pada *smartphone* ditutup pada saat permainan sedang berlangsung, maka koneksi akan terputus. Tampilan pada *PC* hanya menunjukkan tanda kecil bahwa telah terjadi pemutusan koneksi pada *smartphone*, yaitu tanda x pada bagian atas tampilan yang ditunjukan seperti gambar berikut:



Gambar 3.14: Halaman pada *PC* yang menunjukan pemutusan koneksi.

Perbaikan yang dapat dilakukan adalah dengan memberi *feedback* yang lebih jelas pada pemain, apabila ada kesalahan pada aplikasi yang terjadi seperti pemutusan koneksi. Dengan begitu, pemain akan lebih mengetahui bahwa koneksi dapat saja terputus dan tidak dapat melanjutkan permainannya.

3.2 Analisis Sequence Diagram

Pada analisis ini dilakukan pengamatan pada jalannya koneksi *socket.io* antara *server* dan *client*. Hal-hal yang dianalisis adalah bagaimana koneksi tersambung di awal permainan, bagaimana data-data yang dibutuhkan dikirim melalui koneksi tersebut, hingga koneksi antara *client* dan *server* terputus. Analisis dilakukan dengan menggunakan *developer tools* yang tersedia pada *web browser* untuk mengamati koneksi *client*, dan *terminal* untuk mengamati koneksi *server*.

Pada awal permainan, *client* pertama yang melakukan koneksi pada *server* adalah *desktop computer*, yang berperan sebagai *host* dalam permainan. *Host* akan menyediakan suatu kode yang berguna sebagai *room* untuk kedua pemain yang akan bergabung dengan melakukan koneksi ke *server*. *Room* yang disediakan hanya akan menerima tiga *client* saja, yaitu *host*, *player1*, dan *player2*.

Setelah berhasil melakukan koneksi, maka *client* akan memiliki *socket.id* yang berfungsi sebagai tanda pengenal unik, dan kode *room* dimana proses pengiriman dan penerimaan data hanya dapat dilakukan didalam *room* tersebut.

Setelah *client* pertama berhasil tersambung, selanjutnya para pemain yang akan bergabung akan melakukan koneksi pada *server*. Pemain akan menggunakan *smartphone* yang akan berfungsi sebagai *controller* permainan. Untuk melakukan koneksi, pemain harus memasukan kode *room* yang telah disediakan *host* agar dapat bergabung. Pada proses ini, akan dilakukan pengecekan oleh *server*, apakah kode *room* yang telah dikirim oleh *client* tersedia atau tidak. Apabila *room* tersedia, maka pemain akan berhasil bergabung, dan apabila tidak tersedia, maka pemain tidak dapat bergabung.

Pengecekan tambahan yang dilakukan pada proses ini adalah jumlah *client* yang sudah ada didalam *room*. Apabila jumlahnya masih kurang dari tiga, maka *room* masih membuka koneksi bagi yang akan bergabung. Apabila *room* sudah diisi oleh tiga *client*, maka *room* tidak akan menerima *client* yang akan bergabung lagi.

Pada tahap ini, halaman pada *client* dan *server* akan menuju ke halaman pemilihan karakter. Pada jalannya permainan, *socket.id* yang dimiliki oleh masing-masing *client* akan berperan sangat penting untuk proses pengiriman data. Pada koneksi *socket.io*, apabila suatu *client* berpindah

halaman dari satu *file html* menuju *file html* lainnya, koneksi tersebut akan terputus dan akan melakukan koneksi ulang kembali. Dengan begitu, *socket.id* yang dimiliki oleh *client* pun akan berganti karena proses perpindahan *file html* dan koneksi ulang.

Agar koneksi yang dimiliki *client* tidak terputus dan *socket.id* yang dimiliki tidak berganti, maka untuk proses perpindahan halaman harus dibutuhkan satu *file html* saja. Hal tersebut dapat dilakukan dengan cara menggunakan *syntax* yang disediakan oleh *html*, yaitu *template*.

Dengan menggunakan *template* pada satu *file html*, maka proses perpindahan halaman hanya akan terjadi didalam satu *file* saja. Halaman yang akan ditunjukkan kepada *client* akan dilakukan dengan cara berpindah dari satu *template* ke *template* yang lainnya. Dengan begitu, koneksi *socket.io* tidak akan terputus, dan *socket.id* milik *client* tidak akan berganti.

Setelah masuk ke halaman pemilihan karakter, pemain dapat memilih karakter yang akan dimainkan. Pada *smartphone*, akan ditampilkan daftar karakter yang dapat dipilih, dan pada *desktop*, akan ditampilkan karakter mana yang telah dipilih oleh para pemain untuk dimainkan.

Agar karakter yang dipilih pada *smartphone* sesuai dengan yang ditunjukkan oleh *desktop*, maka dibutuhkan proses pengiriman data didalam koneksi *socket.io*. Saat pemain memilih karakter, pemain akan mengirimkan suatu *event* kepada *server* dengan parameter yang berisi *socket.id* dan *value* yang menandakan karakter mana yang dipilih. Setelah *event* tersebut sampai pada *server*, akan dilanjutkan kembali kepada *client* yang berperan sebagai *host*. Pada tahap ini, *host* akan menerima *event* tersebut dan memprosesnya. Parameter *value* yang dikirimkan oleh pemain dibutuhkan oleh *host* untuk menampilkan karakter yang dipilih, sedangkan *socket.id* berfungsi untuk mengetahui pemain mana yang memilih karakter tertentu sehingga dapat menampilkannya dengan tepat.

Setelah karakter ditampilkan pada *desktop*, maka pemain dapat menekan tombol *choose* yang berarti pemain sudah memutuskan untuk memakai karakter tersebut di permainan. Pada proses ini, akan dilakukan pengecekan pada *server*, apakah kedua pemain sudah menetapkan karakter yang akan dimainkan. Apabila belum ada atau hanya satu pemain yang sudah menetapkan karakter, maka permainan belum bisa dimulai. Permainan akan dimulai apabila kedua pemain telah menetapkan karakter, yang kemudian halaman akan berpindah pada halaman permainan.

Kedua pemain yang telah menetapkan karakter akan ditampilkan halaman permainan, begitu juga dengan *host*. Pada halaman *desktop*, pertama-tama akan ditampilkan *countdown* selama tiga detik sebelum para pemain dapat menggunakan *smartphononya* sebagai *controller*. Apabila *countdown* sudah habis, maka permainan dapat dimulai.

Pemain akan menekan tombol yang ditampilkan di *smartphone* secara terus menerus. Pada tahap ini, apabila tombol ditekan, pemain akan mengirimkan suatu *event* pada *server* dengan parameter berisi *socket.id*. Setelah *event* tersebut sampai ke *server*, maka akan dilanjutkan kembali menuju *client* yang berperan sebagai *host*. *Host* akan memproses *event* tersebut, dan menyesuaikan pemain mana yang memiliki *id* sesuai dengan yang diterima. Karakter akan mulai bergerak sesuai dengan *id* para pemain. Agar karakter yang dimainkan dapat mencapai garis akhir, maka pemain harus menekan tombol pada *smartphone* terus menerus.

Apabila sudah ada pemain yang menyentuh garis akhir, maka permainan pun akan berakhir dan halaman akan berubah menuju ke halaman pemenang. Pada saat permainan berakhir, akan dikirimkan suatu *event* ke *server* yang berisi parameter *socket.id* dan *value* untuk menampilkan pemain mana yang menang dan karakter mana yang dimilikinya. *Event* tersebut kemudian dilanjutkan kembali ke *host* lalu pemenang pun dapat ditampilkan. Pada tahap ini, pemain dapat memilih untuk keluar dari permainan dan kembali ke halaman utama.

3.3 Analisis Pemanfaatan Socket.io

BAB 4

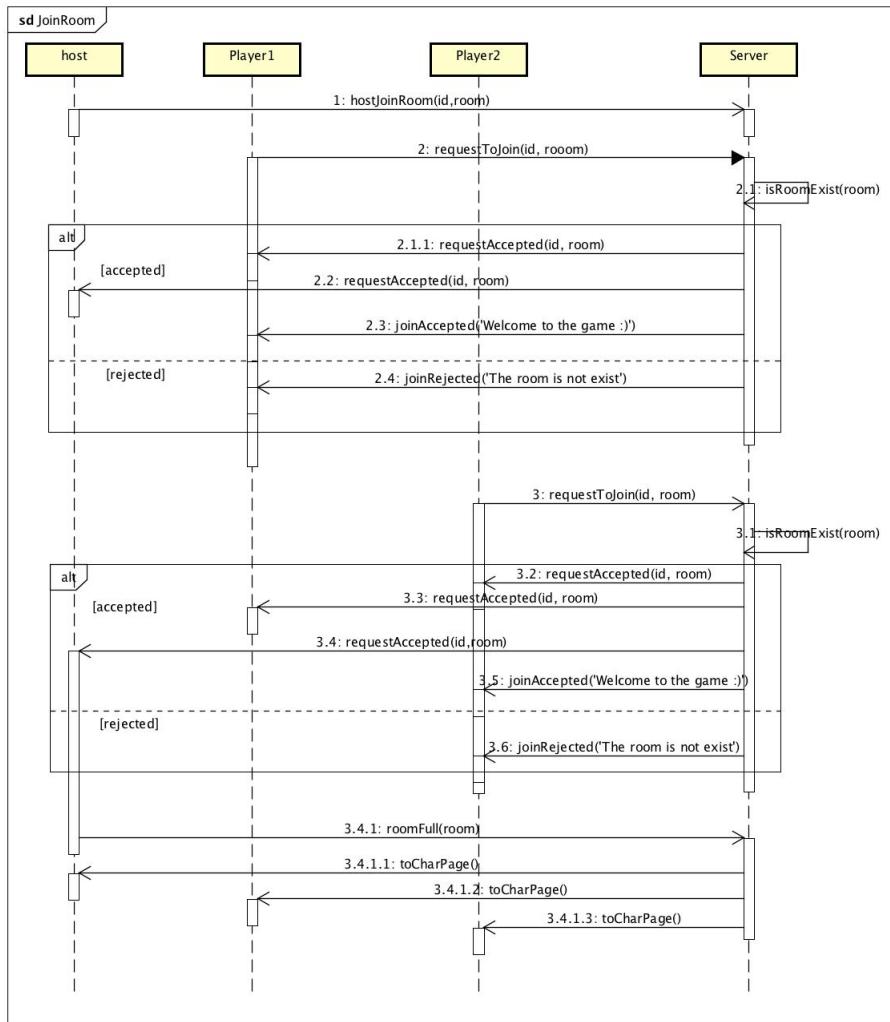
PERANCANGAN

Pada bab ini akan dijelaskan mengenai perancangan aplikasi yang akan dibangun meliputi diagram kelas rinci beserta deskripsi dan fungsinya, dan perancangan antarmuka.

4.1 Perancangan Sequence Diagram

Pada *sequence* diagram akan dibahas mengenai jalannya koneksi *socket.io* dari awal koneksi mulai tersambung hingga koneksi terputus.

4.1.1 Sequence Diagram Join Room



Gambar 4.1: Proses melakukan koneksi ke *socket.io* dan bergabung kedalam *room*.

Pada awal permianan, *client* pertama yang melakukan koneksi pada *server* adalah *desktop computer*, yang berperan sebagai *host* dalam permianan. *Host* akan menyediakan suatu kode yang berguna sebagai *room* untuk kedua pemain yang akan bergabung dengan melakukan koneksi ke *server*. *Room* yang disediakan hanya akan menerima tiga *client* saja, yaitu *host*, *player1*, dan *player2*.

Host akan mengirimkan *event* yang menandakan akan bergabung kedalam *room*, *event* tersebut adalah **hostJoinRoom(id, room)**. *Event* ini memiliki data yang akan dikirimkan kepada *server*, data-data tersebut dijelaskan sebagai berikut:

- **id** identifikasi unik yang dimiliki masing-masing *client* yang terkoneksi dengan *socket.io*.
- **room** suatu *string* yang menandakan ruangan dimana *client* hanya akan melakukan komunikasi didalam ruangan tersebut.

Data-data tersebut akan dimasukan kedalam suatu *array*, dimana *array* tersebut akan menyimpan seluruh *client* yang terkoneksi dengan *socket.io*.

Setelah *host* terkoneksi dengan *socket.io*, maka *room* milik *host* sudah tersedia dan para pemain dapat bergabung kedalam *room* tersebut. Untuk dapat mulai bermain, para pemain harus memasukan kode *room* yang disediakan di halaman *host*. Pemain akan mengirimkan *event*

requestToJoin(id, room) pada *server*. Data yang dikirimkan oleh pemain sama dengan data yang dikirimkan oleh *host*.

Setelah *event* tersebut diterima oleh *server*, maka akan dilakukan pengecekan apakah pemain dapat bergabung atau tidak kedalam *room*. Pengecekan tersebut dilakukan dengan menggunakan *method* **isRoomExist(room)**. Beberapa pengecekan yang dilakukan *method* ini akan dijelaskan sebagai berikut:

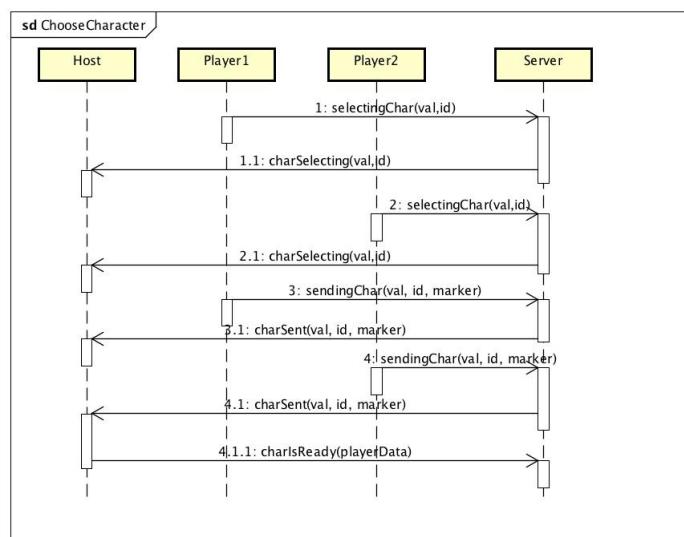
1. Memeriksa apakah data *room* didalam *array* ada yang sesuai dengan parameter *room*.
2. Memeriksa apakah jumlah yang ada didalam *room* yang dimaksud sudah lebih dari tiga atau belum.

Apabila kedua hal diatas terpenuhi, maka pemain akan terkoneksi ke *socket.io* dan bergabung ke dalam *room*. *Server* akan mengirimkan *event* **requestAccepted(id, room)** kepada seluruh *client* yang berada di *room* tersebut. *Event* tersebut berfungsi untuk mencatat *client* telah berhasil bergabung kedalam *room*. *Server* pun akan mengirimkan *event* **joinAccepted('Welcome to the game :')** kepada pemain yang berhasil bergabung kedalam *room*. *Event* tersebut berfungsi untuk memberikan *feedback* kepada pemain bahwa pemain telah berhasil bergabung kedalam permainan.

Apabila kedua hal diatas tidak terpenuhi, maka pemain tidak dapat terkoneksi ke *socket.io*. *Server* akan mengirimkan *event* **joinRejected('The room is not exist')** kepada pemain. *Event* tersebut berfungsi sebagai *feedback* kepada pemain yang tidak dapat bergabung. Pemain pertama yang berhasil bergabung akan berperan sebagai *Player1*. Setelah satu pemain bergabung, maka *server* akan menunggu untuk pemain kedua agar permainan dapat dimulai. Pemain kedua akan melakukan hal yang sama dengan pemain pertama untuk dapat bergabung kedalam room.

Setelah *room* berisi tiga *client*, maka *host* akan mengirimkan *event* **roomFull(room)** kepada *server*. *Event* tersebut menandakan bahwa *room* sudah tidak dapat menerima *client* yang akan bergabung. Setelah *event* tersebut diterima oleh *server*, maka *event* **toCharPage()** akan dipancarkan oleh *server* kepada seluruh koneksi yang ada didalam *room* tersebut. *Event* ini berfungsi untuk mengganti halaman saat ini ke halaman selanjutnya.

4.1.2 Sequence Diagram Choose Character



Gambar 4.2: Proses memilih karakter.

Halaman ini merupakan halaman yang dituju oleh para pemain yang telah berhasil melakukan koneksi dan bergabung kedalam *room*. Para pemain akan memilih karakter yang ada pada halaman

smartphone browser, dimana karakter yang dipilih akan muncul pada halaman *desktop browser*. Pemain yang memilih karakter tertentu akan mengirimkan *event selectingChar(val, id)* kepada *server*. Data-data yang dikirimkan akan dijelaskan sebagai berikut:

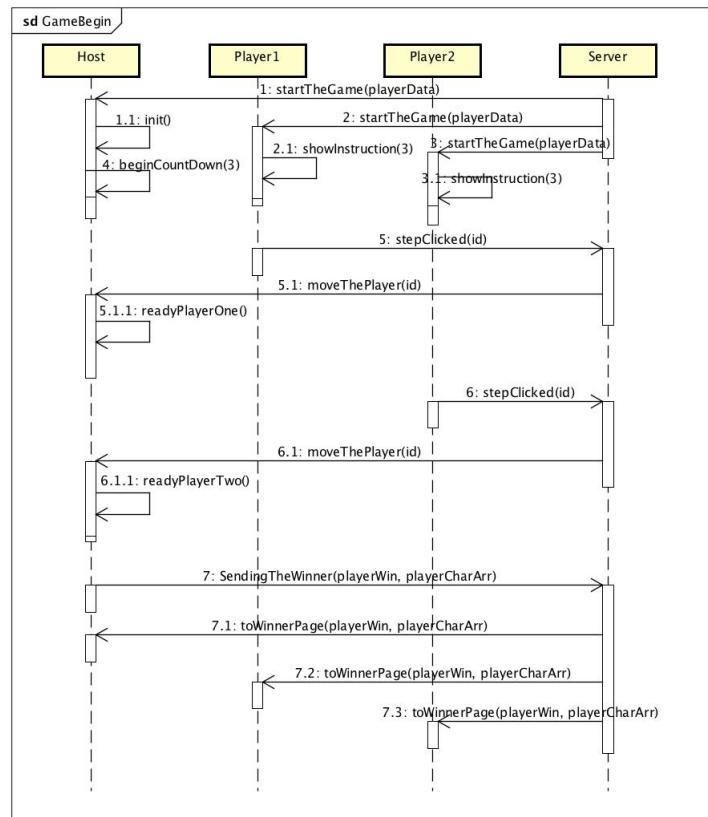
- **val** identifikasi unik yang dimiliki oleh masing-masing karakter.
- **id** identifikasi unik yang dimiliki masing-masing *client* yang terkoneksi dengan *socket.io*.

Setelah *event* tersebut diterima oleh *server*, maka *server* akan mengirimkan *event charSelecting(val, id)* kepada *host*. *Event* tersebut berfungsi untuk menampilkan karakter yang dipilih oleh pemain dihalaman *desktop*.

Apabila pemain akan menetapkan karakter yang telah ditampilkan dihalaman *desktop*, maka *pemain* akan mengirimkan *event sendingChar(val,id,marker)* kepada *server*. Data-data tersebut sama seperti yang dikirimkan oleh *event* sebelumnya, hanya saja ada tambahan data **marker** pada parameter. Data tersebut berfungsi sebagai penanda bahwa sudah ada satu pemain yang telah menetapkan karakter yang akan dimainkan. Setelah *server* menerima *event* tersebut, maka *server* akan meneruskan data-data tersebut pada *host* dengan mengirimkan *event charSent(val,id,marker)*.

Pemain kedua pun akan melakukan hal yang sama untuk memilih dan menetapkan karakter yang akan dimainkan. Apabila kedua pemain telah menetapkan karakter yang akan dimainkan, *host* akan memancarkan *event charIsReady(playerData)*. Data yang dikirimkan merupakan suatu *object array* yang berisi data para pemain dan data karakter yang telah dipilih. *Event* tersebut berfungsi untuk memberi informasi kepada *server* untuk pindah kehalaman selanjutnya.

4.1.3 Sequence Diagram Game Begin



Gambar 4.3: Proses memulai permainan.

Pada tahap ini, *server* akan memancarkan *event startTheGame(playerData)*. Data yang dikirimkan merupakan data yang didapatkan dari *event* sebelumnya. *Event* tersebut akan diterima

oleh seluruh *client* yang berada didalam *room* tertentu, yang kemudian akan memulai permainannya. *Host* akan mengeksekusi *method init()* dan **beginCountDown(3)** untuk memulai permainan. *Method init()* berfungsi untuk mulai menggambar lintasan lari di halaman saat ini yang akan menjadi tempat para karakter dimainkan. *Method beginCountDown(3)* berfungsi untuk melakukan hitungan mundur selama tiga detik.

Para pemain yang menerima *event startTheGame(playerData)* akan mengeksekusi *method showInstruction(3)*. *Method* tersebut berfungsi untuk menampilkan instruksi untuk memainkan permainan selama tiga detik. Apabila hitungan mundur telah selesai, maka permainan akan dimulai.

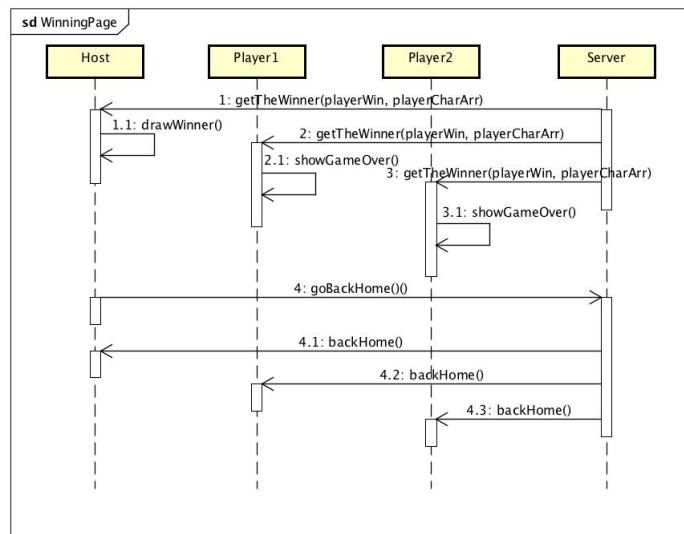
Permainan dilakukan dengan cara menekan tombol-tombol yang ada pada halaman *smartphone*. Apabila pemain menekan tombol, maka *event stepClicked(id)* akan dipancarkan. *Server* akan menerima *event* tersebut, yang kemudian akan meneruskannya dengan memancarkan *event moveThePlayer(id)* kepada *host*. Setelah *event* tersebut diterima oleh *host*, maka *host* akan menjalankan *method* tertentu. Apabila *Player1* yang menekan tombol, maka *method readyPlayerOne()* akan dieksekusi oleh *host*. Apabila *Player2*, maka *readyPlayerTwo()* akan dieksekusi. Kedua *Method* tersebut berfungsi untuk memindahkan posisi karakter milik pemain dari posisi semula hingga posisi tertentu. Permainan akan berakhir apabila ada karakter yang menyentuh garis akhir pertama kali.

Apabila ada pemain yang berhasil menyentuh garis akhir, maka *host* akan memancarkan *event sendingTheWinner(playerWin, playerCharArr)*. Data-data yang dikirimkan akan dijelaskan sebagai berikut:

- **playerWin** angka *integer* yang menandakan pemain nomor berapa yang memenangkan permainan.
- **playerCharArr** *array* yang menyimpan karakter dari masing-masing pemain.

Server akan menangkap *event* tersebut dan meneruskannya dengan memancarkan *event toWinnerPage(playerWin, playerCharArr)* ke setiap *client* pada *room* tertentu. *Event* tersebut berfungsi untuk berpindah ke halaman selanjutnya.

4.1.4 Sequence Diagram Winning Page



Gambar 4.4: Menampilkan para pemain yang telah selesai bermain.

Pada tahap ini, *server* akan memancarkan *event getTheWinner(playerWin, playerCharArr)* dengan data-data yang diterima dari *event* sebelumnya. *Host* akan menerima *event* tersebut

dan akan mengeksekusi *method drawWinner()*. *Method* tersebut berfungsi untuk menampilkan karakter milik para pemenang yang telah selesai bermain. Para pemain akan menerima *event* yang dipancarkan oleh *server* dan mengeksekusi *method showGameOver()*. *Method* ini akan menampilkan teks yang menunjukkan bahwa permainan telah selesai.

Apabila pemain menekan tombol *back* yang ada pada halaman *browser*, maka *host* akan memancarkan *event goBackHome()*. *Event* tersebut akan diterima oleh *server* dan akan diteruskan dengan memancarkan *event backHome()*. Seluruh *client* didalam *room* yang menerima *event* tersebut akan kembali kehalaman semula dan koneksi pun akan terputus.

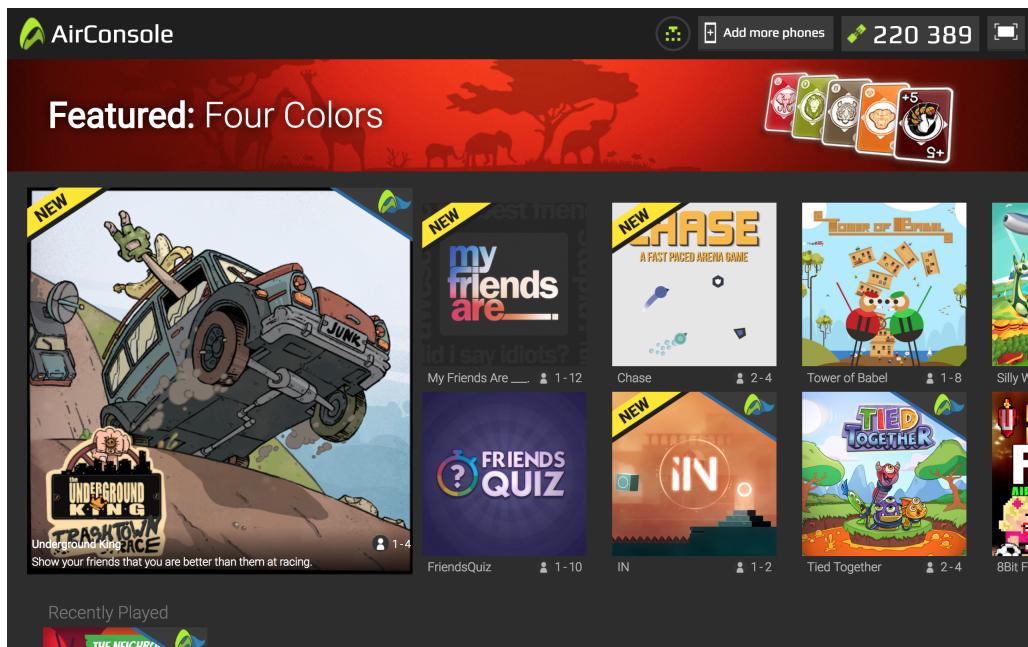
4.2 Perancangan Antarmuka

Antarmuka yang dirancang terbagi menjadi dua bagian, yaitu antarmuka pada *browser* yang ada di *PC* dan *smartphone*.

1. Antarmuka halaman *home*

PC

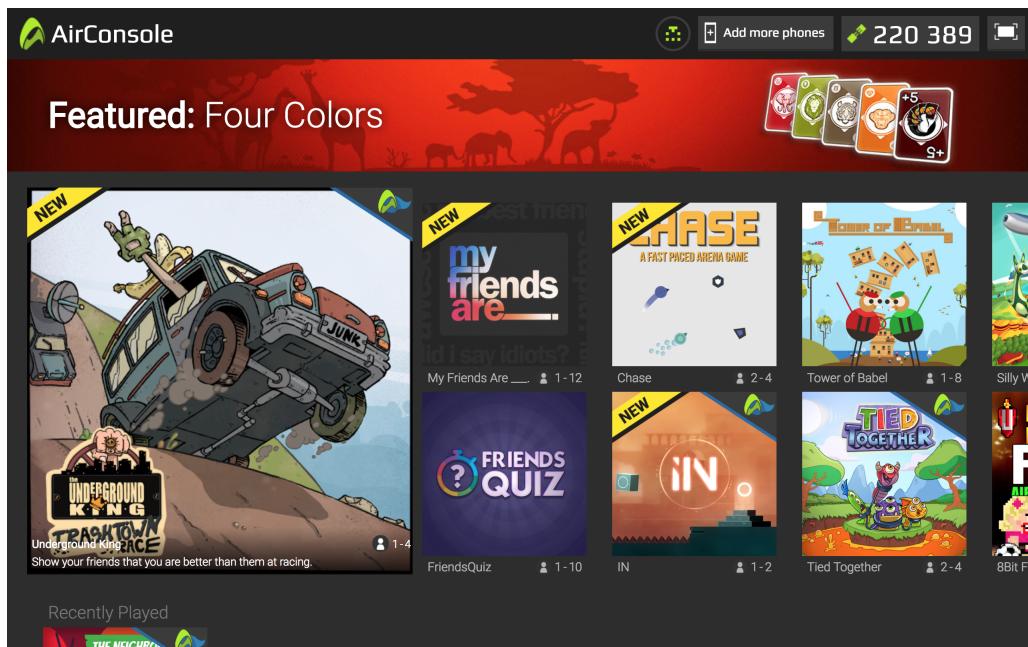
Halaman ini merupakan halaman utama yang pertama kali dituju oleh pengguna yang menggunakan *PC*. Komponen halaman ini terdiri dari dua buah gambar jari yang melambangkan permainan, teks yang menunjukan nama permainan yaitu *finger for life*, dan tombol *start* yang digunakan untuk memulai permainan. Rancangan antarmuka halaman *home* dapat dilihat pada Gambar 4.5



Gambar 4.5: Halaman pada *PC* yang menunjukan berbagai permainan yang dapat dipilih.

Smartphone

Halaman ini merupakan halaman utama yang pertama kali dituju oleh pengguna yang menggunakan *smartphone*. Komponen halaman ini terdiri dari teks yang menunjukan nama permainan yaitu *finger for life*, dan tombol *join* yang digunakan untuk memulai permainan. Rancangan antarmuka halaman *home* dapat dilihat pada Gambar 4.6.

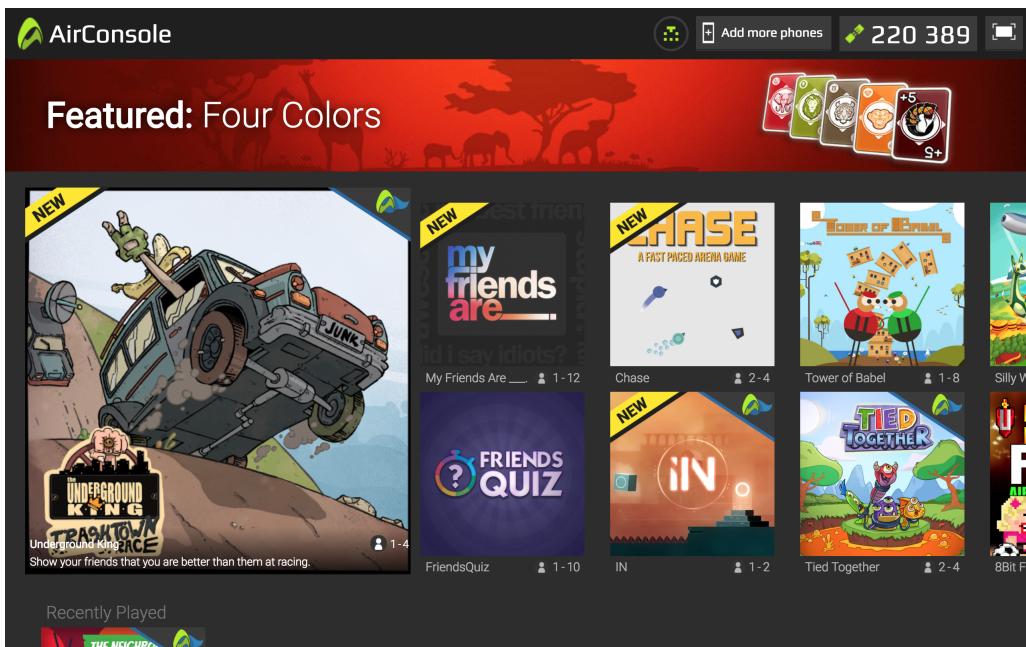


Gambar 4.6: Halaman pada *PC* yang menunjukkan berbagai permainan yang dapat dipilih.

2. Antarmuka halaman *sync*

PC

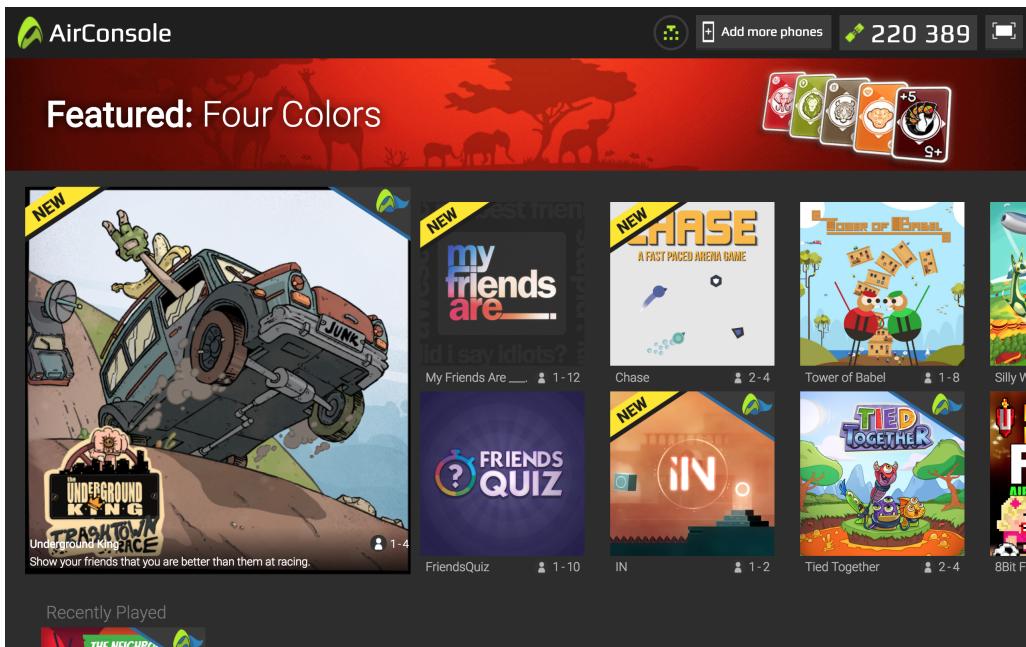
Halaman ini menampilkan suatu perintah yang dapat dilakukan oleh pengguna apabila akan bergabung dalam permainan. Halaman ini pun menyediakan suatu kode untuk para pemain dimana kode tersebut akan berfungsi sebagai suatu *room* bagi para pemain. Pada halaman ini akan dilakukan proses sinkronisasi untuk para pengguna, apakah berhasil bergabung dalam permainan atau tidak. Apabila berhasil, maka akan muncul suatu teks yang menunjukkan suatu pemain telah berhasil bergabung. Apabila tidak, maka tidak akan menampilkan apapun dan halaman tidak akan menuju ke halaman selanjutnya sebelum ada dua pemain yang berhasil bergabung. Rancangan antarmuka halaman *sync* pada *PC* dapat dilihat pada Gambar 4.7.



Gambar 4.7: Halaman pada *PC* yang menunjukkan berbagai permainan yang dapat dipilih.

Smartphone

Halaman ini berfungsi untuk melakukan *request* untuk bergabung dalam permainan. Komponen halaman ini terdiri dari kolom kode, dan tombol *send*. Rancangan antarmuka halaman *sync* pada *smartphone* dapat dilihat pada Gambar 4.8.



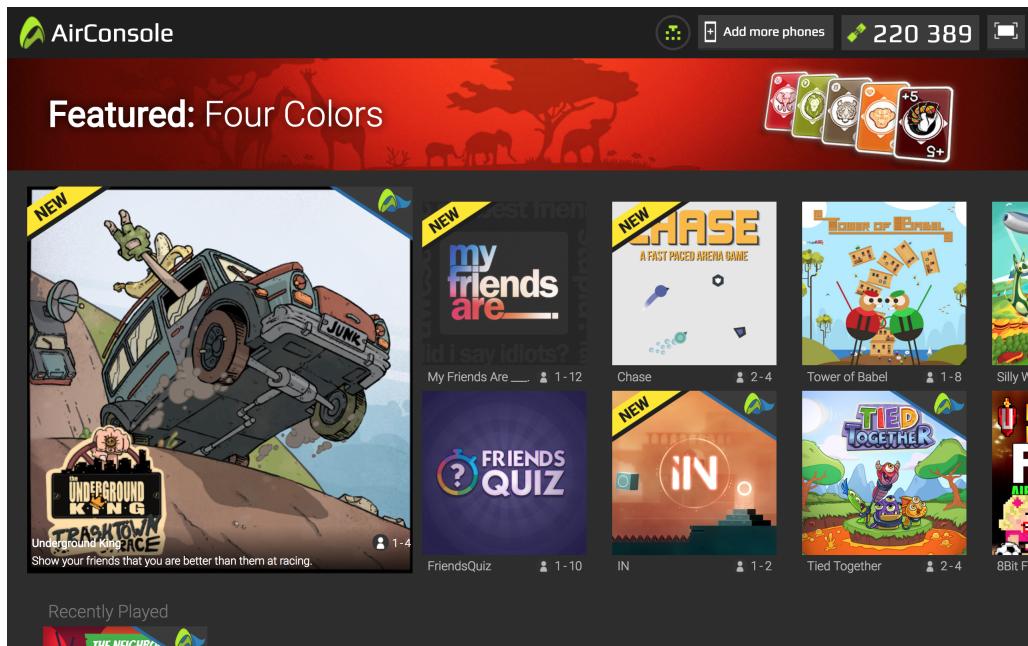
Gambar 4.8: Halaman pada *PC* yang menunjukkan berbagai permainan yang dapat dipilih.

3. Antarmuka halaman *character*

PC

Halaman ini akan menampilkan karakter yang telah dipilih oleh pemain. Apabila karakter belum memilih karakter yang akan dimainkan, maka halaman ini belum menampilkan apapun.

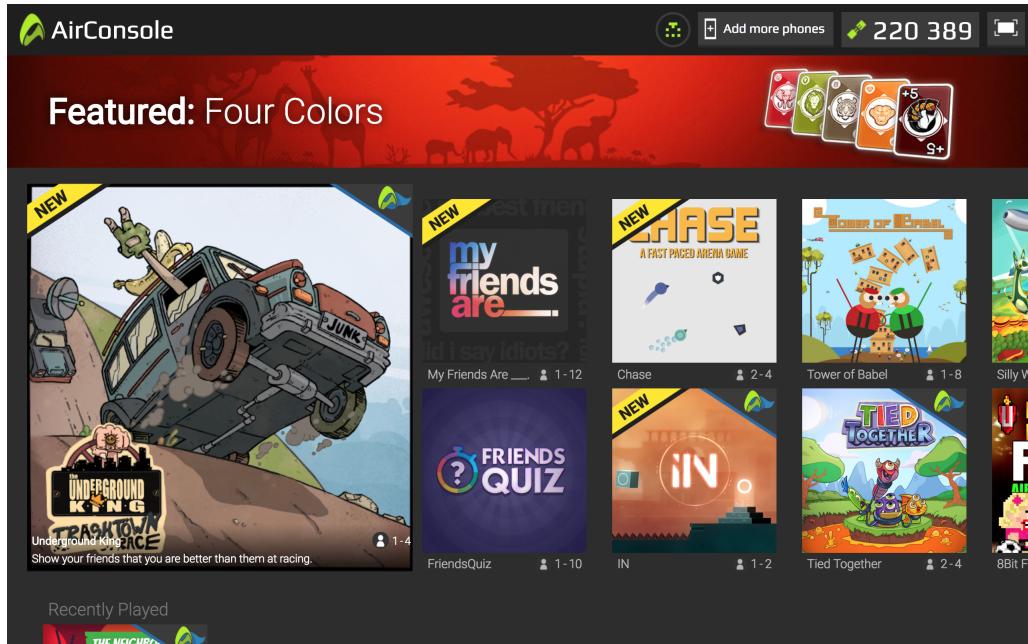
Rancangan antarmuka halaman *character* pada *PC* dapat dilihat pada Gambar 4.9.



Gambar 4.9: Halaman pada *PC* yang menunjukkan berbagai permainan yang dapat dipilih.

Smartphone

Halaman ini akan menampilkan daftar karakter yang dapat dimainkan oleh pemain. Komponen halaman ini terdiri dari daftar karakter, dan tombol *choose*. Rancangan antarmuka halaman *character* pada *smartphone* dapat dilihat pada Gambar 4.10.

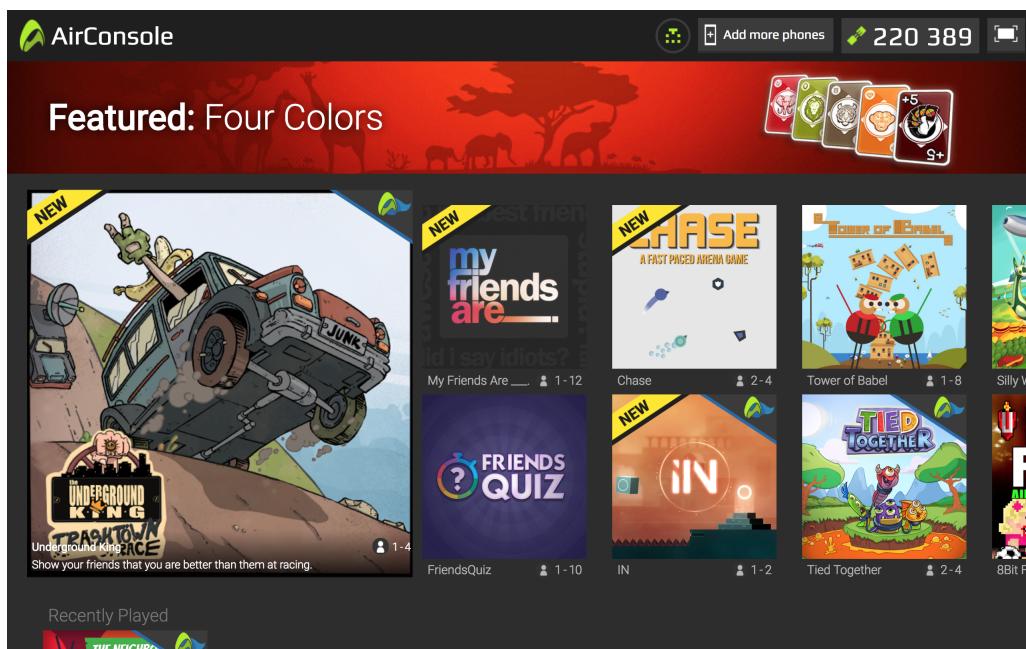


Gambar 4.10: Halaman pada *PC* yang menunjukkan berbagai permainan yang dapat dipilih.

4. Antarmuka halaman *gameplay*

PC

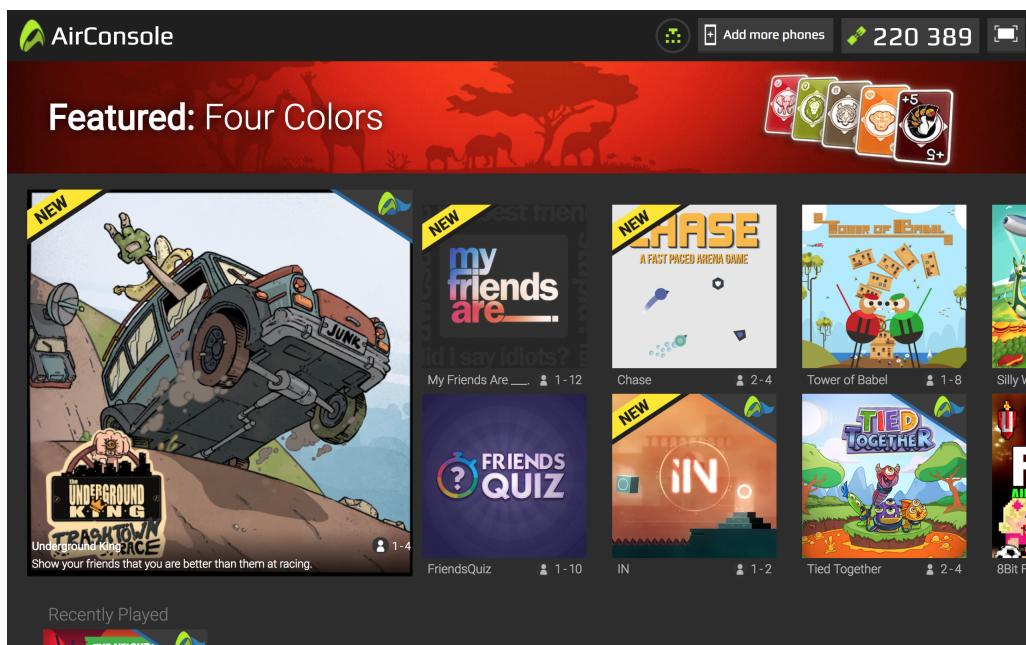
Halaman ini menampilkan arena permainan untuk para pemain. Komponen halaman ini terdiri dari suatu gambar lintasan lari, dan karakter yang telah dipilih pada halaman sebelumnya. Rancangan antarmuka halaman *gameplay* pada *PC* dapat dilihat pada Gambar 4.11.



Gambar 4.11: Halaman pada *PC* yang menunjukan berbagai permainan yang dapat dipilih.

Smartphone

Halaman ini berfungsi sebagai *controller* untuk para pemain. Komponen halaman ini terdiri dari dua buah gambar telapak kaki yang berfungsi sebagai tombol. Pemain dapat menekan tombol berulang kali untuk menggerakan karakter yang ada pada halaman *gameplay* di *PC*. Rancangan antarmuka halaman *gameplay* pada *smartphone* dapat dilihat pada Gambar 4.12.

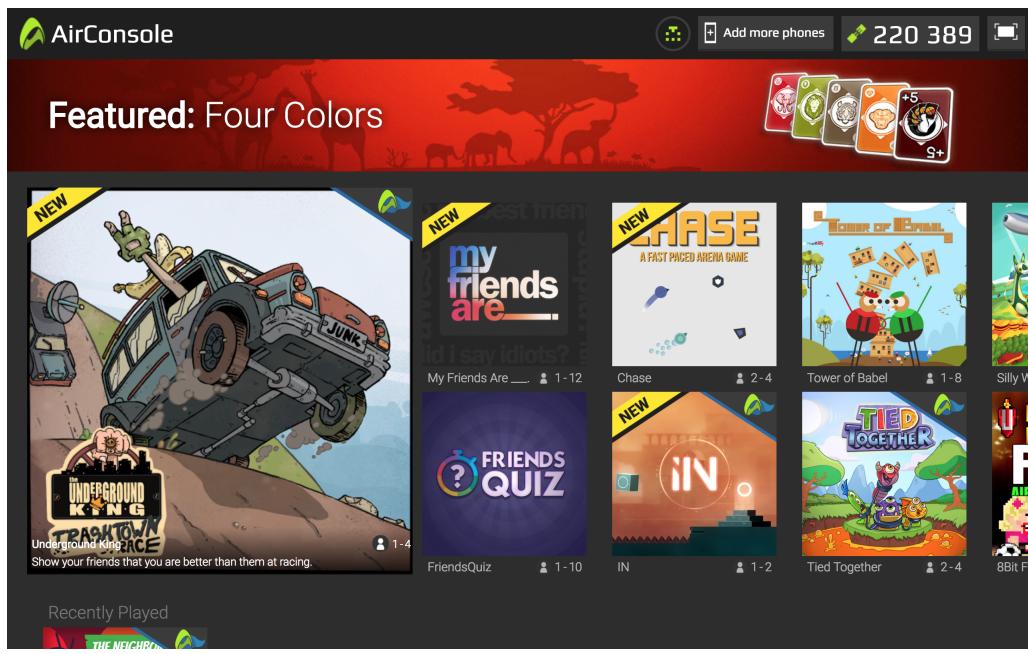


Gambar 4.12: Halaman pada *PC* yang menunjukan berbagai permainan yang dapat dipilih.

5. Antarmuka halaman *gameover*

PC

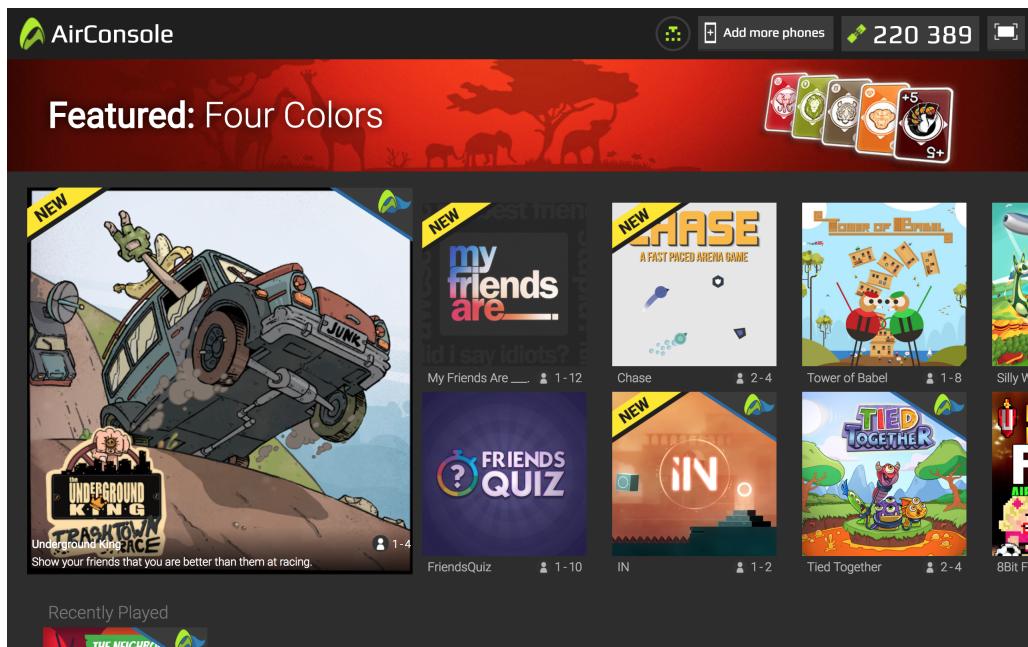
Halaman ini menampilkan para pemenang yang telah berhasil menyelesaikan permainan. Setelah pemain selesai bermain, pemain dapat menekan tombol *back* untuk kembali ke halaman utama dan mengakhiri permainan. Rancangan antarmuka halaman *gameover* pada *PC* dapat dilihat pada Gambar 4.13.



Gambar 4.13: Halaman pada *PC* yang menunjukkan berbagai permainan yang dapat dipilih.

Smartphone

Halaman ini menampilkan teks yang menandakan bahwa permainan telah selesai. Rancangan antarmuka halaman *gameover* pada *smartphone* dapat dilihat pada Gambar 4.14.



Gambar 4.14: Halaman pada *PC* yang menunjukan berbagai permainan yang dapat dipilih.

4.3 Perancangan Struktur Direktori

Bagian ini akan membahas mengenai perancangan struktur direktori untuk membangun web. Struktur direktori meliputi beberapa folder dan *file* yang memiliki fungsi berbeda-beda. Folder beserta *file* yang digunakan akan dijelaskan sebagai berikut:

1. bin

Folder ini berisi bagian-bagian yang berperan sebagai *server*. Pada folder ini terdapat folder lain dan berkas yang berfungsi untuk menangani jalannya koneksi pada saat web diakses. Berikut merupakan isi dari folder ini:

(a) Folder **utils**

Folder ini menyimpan berkas yang membantu proses berjalannya koneksi untuk *server*. Isi dari folder ini adalah sebagai berikut:

- Berkas **users.js** Berkas ini berfungsi untuk menyimpan seluruh *client* yang terkoneksi pada *socket.io server*. Seluruh data pengguna yang akan memainkan permainan web akan disimpan dan diatur oleh berkas ini.

(b) Berkas **www**

Berkas ini berfungsi sebagai *server*. Seluruh koneksi yang tersambung dan terputus akan diatur oleh berkas ini. Komunikasi antara *client* pun akan diatur oleh *server*.

2. public

3. routes

4. views

5. app.js

DAFTAR REFERENSI

- [1] Dahl, R. (2009) Node.js. <https://nodejs.org/en/>. [Online; diakses 7-Oktober-2017].
- [2] Holowaychuk, T. (2010) Express.js. <https://expressjs.com/>. [Online; diakses 7-Oktober-2017].
- [3] Mozilla (2011) WebSockets. https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Online; diakses 7-Oktober-2017].
- [4] Rauch, G. (2011) Socket.io. <https://socket.io/>. [Online; diakses 7-Oktober-2017].
- [5] Rauch, G. (2011) Socket.io Server API. <https://socket.io/docs/server-api/>. [Online; diakses 7-Oktober-2017].
- [6] Mozilla (2004) Canvas API. https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API. [Online; diakses 7-Oktober-2017].

LAMPIRAN A

KODE PROGRAM

Listing A.1: MyCode.c

```

1 // This does not make algorithmic sense,
2 // but it shows off significant programming characters.
3
4 #include<stdio.h>
5
6 void myFunction( int input, float* output ) {
7     switch ( array[1] ) {
8         case 1: // This is silly code
9             if ( a >= 0 || b <= 3 && c != x )
10                 *output += 0.005 + 20050;
11             char = 'g';
12             b = 2^n + ~right_size - leftSize * MAX_SIZE;
13             c = ( -aaa + &daa ) / ( bbb++ - ccc % 2 );
14             strcpy(a,"hello_@?");
15         }
16     count = ~mask | 0x00FF00AA;
17 }
18
19 // Fonts for Displaying Program Code in LATEX
20 // Adrian P. Robson, nepswb.co.uk
21 // 8 October 2012
22 // http://nepswb.co.uk/docs/progfonts.pdf
23

```

Listing A.2: MyCode.java

```

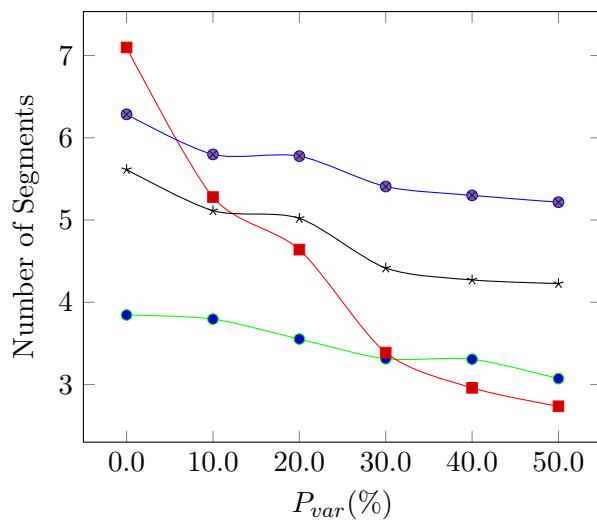
1 import java.util.ArrayList;
2 import java.util.Collections;
3 import java.util.HashSet;
4
5 //class for set of vertices close to furthest edge
6 public class MyFurSet {
7     protected int id;                                //id of the set
8     protected MyEdge FurthestEdge;                   //the furthest edge
9     protected HashSet<MyVertex> set;                //set of vertices close to furthest edge
10    protected ArrayList<ArrayList<Integer>> ordered; //list of all vertices in the set for each trajectory
11    protected ArrayList<Integer> closeID;           //store the ID of all vertices
12    protected ArrayList<Double> closeDist;          //store the distance of all vertices
13    protected int totaltrj;                          //total trajectories in the set
14
15    /*
16     * Constructor
17     * @param id : id of the set
18     * @param totaltrj : total number of trajectories in the set
19     * @param FurthestEdge : the furthest edge
20     */
21    public MyFurSet(int id,int totaltrj,MyEdge FurthestEdge) {
22        this.id = id;
23        this.totaltrj = totaltrj;
24        this.FurthestEdge = FurthestEdge;
25        set = new HashSet<MyVertex>();
26        ordered = new ArrayList<ArrayList<Integer>>();
27        for (int i=0;i<totaltrj;i++) ordered.add(new ArrayList<Integer>());
28        closeID = new ArrayList<Integer>(totaltrj);
29        closeDist = new ArrayList<Double>(totaltrj);
30        for (int i = 0;i < totaltrj;i++) {
31            closeID.add(-1);
32            closeDist.add(Double.MAX_VALUE);
33        }
34    }
35}
36

```

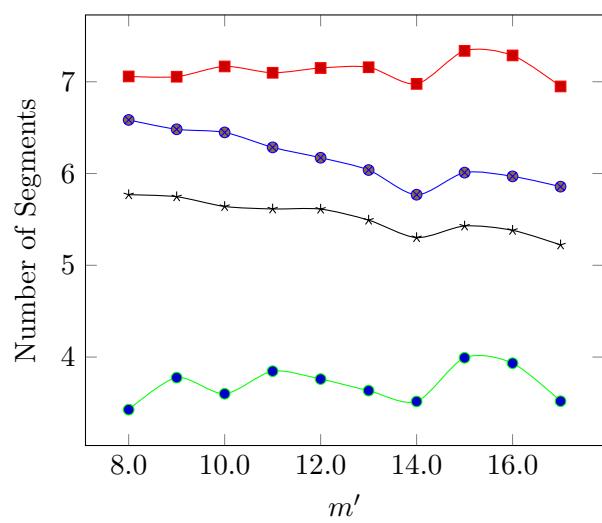

LAMPIRAN B

HASIL EKSPERIMENT

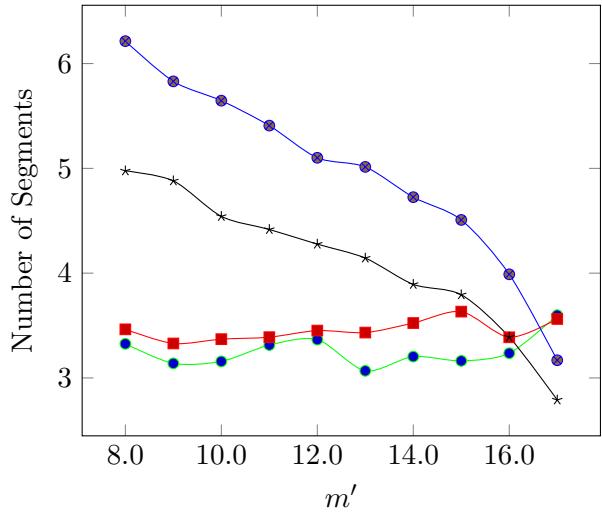
Hasil eksperimen berikut dibuat dengan menggunakan TIKZPICTURE (bukan hasil excel yg diubah ke file bitmap). Sangat berguna jika ingin menampilkan tabel (yang kuantitasnya sangat banyak) yang datanya dihasilkan dari program komputer.



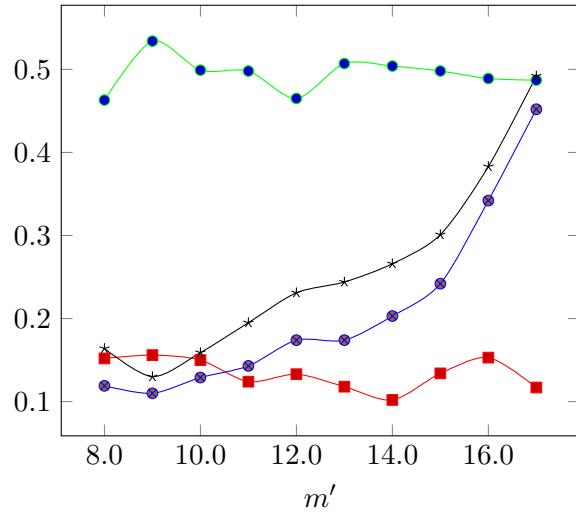
Gambar B.1: Hasil 1



Gambar B.2: Hasil 2



Gambar B.3: Hasil 3



Gambar B.4: Hasil 4