Rancang Bangun dan Implementasi Modul Pertarungan Pemain Lawan Pemain pada Aplikasi Permainan Card Warlock Saga Berbasis Android

Nama Mahasiswa	: Misbahul Munir
NRP	: 5110100009
Jurusan	: Teknik Informatika FTIf-ITS
Dosen Pembimbing 1	: Imam Kuswardayan, S.Kom., M.T.
Dosen Pembimbing 2	: Ridho Rahman H., S.Kom., M.Sc.

ABSTRAK

Card Warlock Saga adalah aplikasi permainan turn-based collectible card yaitu bertema turn-based yang menggunakan kartu sebagai mekanisme pertarungan permainan. Pemain memiliki satu hero yang disebut magician dan set kartu sebagai mekanisme pertarungan. Terkadang permainan serasa membosankan bila dimainkan sendiri. Dikarenakan permainan Card Warlock Saga yang berbasis permainan sosial maka dirancang suatu sistem agar pemain dapat bersosialisasi lewat pertarungan antar pemain.

Modul pertarungan ini dibuat sebagai pendukung aplikasi permainan utama Card Warlock Saga. Sistem aturan main yang diterapkan pada modul pertarungan pemain lawan pemain akan diterapkan juga sebagai aturan main utama permainan pada mode offline.

Modul pertarungan ini dibangun dengan menerapkan konsep desain pattern. Konsep desain pattern diperlukan dikarenakan perancangan aplikasi yang berbasis modul. Diharapkan dengan digunakannya rancangan konsep desain pattern memudahkan pengembangan pemainan menjadi lebih modular dan meningkatkan tingkat reusability code.

***Kata kunci**: Android, Game, Networking, Unity, Web Service.*

Design and Implementation of Player Versus Players Battle Module on Warlock Saga Card Game-Based Android

Student Name	: Misbahul Munir
Student ID	: 5110100009
Major	: Teknik Informatika FTIf-ITS
Advisor 1	: Imam Kuswardayan S.Kom., M.T.
Advisor 2	: Ridho Rahman H., S.Kom., M.Sc.

ABSTRACT

Card Warlock Saga is a turn-based game application that is themed collectible card that uses a turn-based card battle game as a mechanism. Players have a hero called magician and card set as the fight mechanism. Sometimes the game seemed boring when played alone. Due Warlock Saga Card Game-based social game then designed a system so that players can socialize through the battle between players.

The module is designed to support the main game applicationt. The system of rules that applied to the module players opposing players will be applied as well as the main rules of the game in offline mode

This fight module is built by applying the concept of design pattern. The concept of design pattern is needed because module-based application design. It is expected that the use of the design on the concept of design pattern facilitates the development of the game due to become more modular and increase the level of code reusability.

Keyword: Android, Game, Networking, Unity, Web Service.

DAFTAR KODE SUMBER

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan Tugas Akhir. Teori-teori tersebut meliputi *Social Game*, *Web Service,* Kryonet, *Android Development Tools* (ADT), dan Unity.

2.1 *Social Game*

Social Game adalah permainan yang dimainkan dengan teman-teman yang berada pada suatu jaringan sosial sehingga pemain bisa mengundang atau mengajak teman untuk melakukan aktivitas tertentu dalam permainan. *Social Game* pada umumnya bersifat *asynchronous* di mana untuk memainkannya pemain tidak perlu *online* secara bersama–sama [1].

2.2 *Web Service*

Web service adalah metode untuk berkomunikasi antar perangkat elektronik melalui *World Wide Web* [2]. *Web service* menyediakan fungsi–fungsi yang dibutuhkan oleh perangkat lunak dengan menggunakan *network address* melalui jaringan internet. Contoh-contoh format *web service* di antaranya adalah WSDL, SOAP, dan serialisasi XML.

2.3 Kryonet

Kryonet adalah sebuah kerangka kerja yang menyediakan fungsi-fungsi *networking* untuk melakukan komunikasi *client server*. Kryonet menggunakan Kryo *serialization library* yang secara efisien mengirimkan objek melalui jaringan. Kryonet ideal untuk aplikasi *client server* apapun, khususnya *game*. Kryonet juga berguna pada komunikasi antar proses.

Kryonet mengimplementasi fungsi *thread runnable* milik Java di mana pada Java *thread* dijalankan melalui *high priority thread.* Semua *thread* pada Java mempunyai prioritas nilai antara

1 sampai dengan 10. Di mana *thread-thread* yang mempunyai nilai *high priority* dieksekusi terlebih dahulu.

Pada penggunaannya Kryonet hanya perlu melakukan implementasi terhadap kelas *listener* milik Kryonet di mana akan dimplementasikan empat buah fungsi yaitu `Receive`, `Connect`, dan `disconnect`. Setiap fungsi yang terdapat pada kelas *listener* akan mendapatkan *update* dari *thread* milik Java [3].

2.4 *Transmission Control Protocol* (TCP)

Transmission Control Protocol (TCP) adalah suatu program yang berfungsi menangani transmisi dan penerimaan data. TCP secara bergiliran dilayani oleh *packet switch* yang terhubung pada *host* yang menggunakan transmisi TCP [4].

2.5 ADT (*Android Development Tool*)

Eclipse juga mendukung berbagai pemrograman lainnya seperti Android. *Android Development Tool* (ADT) merupakan sebuah *project* eclipse yang menyediakan IDE untuk pengembangan aplikasi Android . Terdapat dua bagian utama pada ADT yaitu *view user interface* yang didefinisikan pada bahasa XML dan kode utama yang didefinisikan pada bahasa pemrograman Java [5].

2.6 *Design Pattern*

Design Pattern adalah deskripsi atau *template* yang digunakan untuk menyelesaikan masalah dalam banyak situasi. Pada pemrograman yang berorientasi objek, *design pattern* digunakan sebagai *template* untuk menunjukkan bagaimana relasi antar kelas di sebuah *source code* dapat bekerja dengan tujuan agar *source code* yang digunakan memiliki tingkat *reusability* yang tinggi. *Design pattern* terdiri atas beberapa bagian [6].

1) ***Pattern name*** adalah penamaan yang bisa kita gunakan untuk menggambarkan desain untuk suatu masalah dan solusinya. Dengan adanya penamaan *pattern* memudahkan

kita untuk membicarakannya dengan kolega sehingga memudahkan kita untuk saling berinteraksi mengenai *pattern.*

2) ***Problem*** adalah deskripsi untuk menunjukkan masalah-masalah apa saja yang bisa diselesaikan dengan *design pattern. Problem* juga mendeskripsikan desain masalah yang spesifik seperti bagaimana merepresentasikan suatu algoritma ke dalam struktur objek.
3) ***Solutions*** adalah penggambaran mengenai *element* yang akan digunakan dalam mendesain solusi. Dikarenakan *pattern* adalah *template* yang dapat diterapkan pada banyak situasi, *pattern* tidak menggambarkan solusi secara langsung ataupun memberikan implementasi, namun hanya memberikan deskripsi abstrak dan desain masalah.
4) ***Consequences*** adalah hasil dan timbal balik yang didapatkan saat mengimplementasikan *pattern*. *Consequences* penting untuk mengetahui memahami *cost* dan keuntungan ketika menerapkan *design pattern*, dikarenakan *reusability* merupakan hal yang penting pada *Object Oriented Programming* (OOP) .

2.6.1 *Strategy Pattern*

Strategy pattern adalah salah satu bagian dari *design pattern* yang mengenkapsulasi algoritma-algoritma yang kompleks pada sebuah hirarki, sehingga memudahkan *programmer* untuk menjalankan algoritma yang dinginkan saat program sedang berjalan. *Strategy pattern* biasanya dibentuk dalam hirarki abstraksi atau *interface* [6].

2.6.2 *State Pattern*

State pattern adalah salah satu bagian dari *design pattern* yang memungkinkan suatu objek untuk mengubah *behavior* ketika *internal state* berubah. Objek yang diimplementasikan dengan *state pattern* akan terlihat berubah *behavior*-nya [6].

2.6.3 *Singleton*

Singleton adalah objek statis yang memungkinkan satu kelas hanya memiliki satu *instance*, memberikan akses global bagi objek tersebut. *Singleton* digunakan sebagai objek statis yang dapat diakses dari kelas lain sehingga dapat bertukar *value* antar variabel tanpa menggunakan variabel yang bersifat statis [6].

2.6.4 *Command Pattern*

Command pattern adalah *template* objek yang mengenkapsulasi *request* ke dalam sebuah objek, sehingga memungkinkan memberikan parameter kepada objek-objek pada klien dengan *request* yang berbeda-beda [6].

2.7 JMeter

JMeter adalah aplikasi *open source* yang berjalan pada *desktop*. JMeter didesain untuk melakukan *load test* dan mengukur performa dari aplikasi *web*. Pada awalnya JMeter hanya digunakan pada aplikasi *web* namun sejalan dengan perkembangan, JMeter dapat melakukan pengujian bukan saja pada aplikasi *web* namun juga pada TCP, *native command*, dan *shell script* [7].

2.8 Unity

Unity merupakan sebuah ekosistem pengembangan permainan yang terintegrasi dan kaya akan alat atau perlengkapan yang sangat berguna untuk membangun permainan interaktif seperti pencahayaan, efek khusus, animasi, dan mesin fisika. Unity dapat digunakan untuk membangun permainan dengan dukungan grafis tiga dimensi atau dua dimensi. Unity juga dapat dilakukan secara bersamaan untuk mengetes maupun mengubah permainan yang dibuat. Ketika sudah siap, permainan dapat dipublikasikan pada berbagai macam platform seperti Mac, PC, Linux, Windows Store, iOS, Android, Windows Phone 8, Blackberry 10, Wii U, PS3, dan Xbox 360 [8].

2.9 Sinkronisasi

Sinkronisasi adalah proses pengaturan jalannya proses secara bersamaan. Tujuan utama sinkronisasi adalah menghindari terjadinya inkonsistensi data karena pengaksesan oleh beberapa proses yang berbeda, serta untuk mengatur urutan jalannya proses-proses sehingga dapat berjalan dengan lancar dan terhindar dari *deadlock*, atau *starvation* [9].

2.10 Java *Thread Scheduling*

Thread Scheduling pada JVM menggunakan prioritas tertinggi pada penjadwalan eksekusi *thread. Thread* dengan prioritas tertinggi memiliki nilai 10 sedangkan *thread* dengan prioritas terendah memiliki nilai 1. Pada proses eksekusinya apabila terdapat dua buah *thread* yang memiliki nilai prioritas yang sama, maka akan dilakukan pemilihan *thread* berdasarkan FIFO (*First In First Out*) [10].

BAB III
ANALISIS DAN PERANCANGAN SISTEM

Bab ini membahas tahap analisis permasalahan dan perancangan dari sistem yang akan dibangun. Analisis permasalahan membahas permasalahan yang diangkat dalam pengerjaan Tugas Akhir. Analisis kebutuhan mencantumkan kebutuhan-kebutuhan yang diperlukan perangkat lunak. Selanjutnya dibahas mengenai perancangan sistem yang dibuat. Pendekatan yang dibuat dalam perancangan ini adalah pendekatan berorientasi objek. Perancangan direpresentasikan dengan diagram UML (*Unified Modelling Language*).

3.1 Analisis

Tahap analisis dibagi menjadi beberapa bagian antara lain cakupan permasalahan, deskripsi umum sistem, kasus penggunaan sistem, dan kebutuhan perangkat lunak.

3.1.1 Analisis Permasalahan

Permasalahan utama yang diangkat adalah pembuatan aturan main modul pertarungan yang sesuai dengan konsep dan tema yang telah ditetapkan yaitu *role playing game*. Bagaimana menentukan protokol komunikasi agar setiap perangkat *mobile* yang terhubung dengan *server* dapat saling berkomunikasi?

State pattern digunakan untuk menentukan *state* yang dilakukan ketika pemain melakukan eksekusi efek kartu, melakukan *draw*, menyerang, dan mengakhiri giliran. Dengan adanya penggunaan *state patern*, dapat dirancang sebuah aturan main yang sesuai dengan tema permainan *role playing game*. Pada *role playing game* pemain melakukan permainan dengan bergilir. *Singleton* dan *command pattern* digunakan pada komunikasi *client server*. Pada komunikasi *client server* pemain akan mengirimkan protokol melalui *singleton* dan hasil kembalian akan diterjemahkan oleh *client* menggunakan *command pattern.*

3.1.2 *Deskripsi* Umum Sistem

Card Warlock Saga adalah aplikasi permainan *turn-based collectible card* bertema *turn-based*, yang menggunakan kartu sebagai mekanisme pertarungan permainan. Pemain memiliki satu *hero* yang disebut *magician* dan *set* kartu sebagai mekanisme pertarungan.

Permainan ini merupakan gabungan dari strategi, *role playing game*, dan *card battle*, sehingga dengan penggabungan tersebut memberikan suatu sensasi baru dalam permainan. Di mana selain pemain melakukan eksplorasi di dalam dunia permainan, pemain juga harus mengatur strategi kartu apa saja yang dimasukkan ke dalam *deck* dan dipakai untuk bertarung.

Tidak seperti permainan *role playing game* lainnya, permainan ini tidak memiliki kondisi tamat. Agar tidak bosan, permainan ini dibekali dengan *avatar*, kartu, *dungeon*, dan banyak *monster* yang beragam. Selain itu ditambahkan unsur sosial agar pemain dapat berinteraksi dengan pemain lain seperti *trading card, battle, invitation request*, serta s*end gift*.

Aplikasi yang dibangun adalah modul pertarungan dari aplikasi Card Warlock Saga yang akan diintegrasikan dengan fungsi-fungsi yang berada pada *web service* di mana kembalian fungsi–fungsi tersebut nantinya akan dikonsumsi oleh aplikasi dalam bentuk XML. Lebih lanjut modul pertarungan ini akan diintegrasikan pula dengan modul-modul lainnya berupa modul sosial agar menjadi satu aplikasi yang utuh.

Yang dimaksud dengan modul pertarungan pemain lawan pemain adalah modul yang berfungsi untuk menangani pertarungan pemain lawan pemain dengan menggunakan komunikasi *socket* yang terhubung antara satu *server* dan banyak klien. lampiran B Gambar B. 1 merupakan potongan dari *block diagram* sistem yang diimplementasikan nantinya. *Block diagram* yang ditandai dengan kotak warna merah adalah representasi dari

modul pertarungan yang nantinya akan diimplementasikan di dalam *permainan*.

Permasalahan lain yaitu pemilihan protokol yang sesuai dengan kebutuhan sistem. Protokol TCP dipilih karena perancangan aturan main yang tidak selalu melakukan *update* terhadap klien dan membutuhkan protokol yang memiliki tingkat *reliability* pengiriman data yang tinggi. Protokol TCP memiliki tingkat *reliability* pengiriman data yang tinggi berbeda dengan protokol lain seperti UDP, TCP memiliki mekanisme *three way handshake* di mana pada proses tersebut klien dan *server* bersama-sama mengirimkan *acknowledgment* menandakan bahwa paket data dapat saling bertukar data antar *client server*. Kelebihan lainnya pada pemilihan protokol TCP adalah jika terjadi kegagalan koneksi protokol TCP dapat memberi tahu *server* ataupun klien. Hal ini berbeda dengan UDP di mana jika ada kegagalan koneksi masih harus di tangani pada tingkat aplikasi. Adapun kelemahan pada pemilihan protokol TCP adalah memerlukan waktu koneksi yang lebih lama dikarenakan sistem *three way handshake* untuk memastikan paket benar-benar sampai di tujuan.

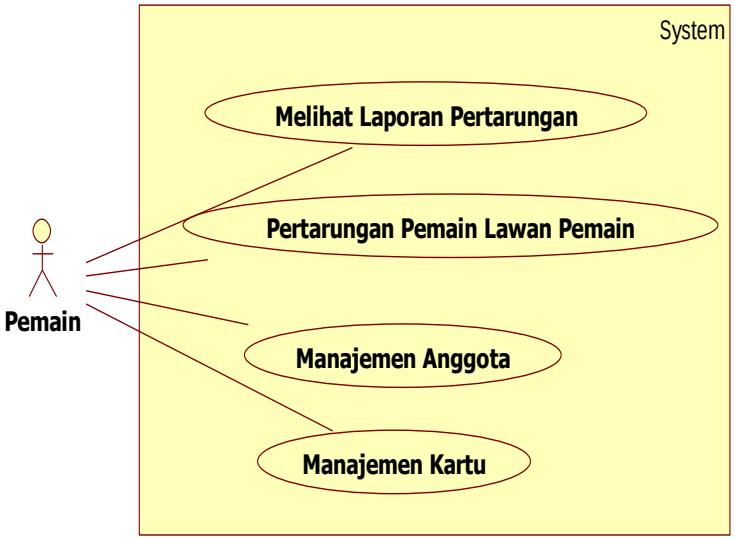
Permasalahan selanjutnya adalah adanya limitasi dari Unity di mana pada Unity kelas *library socket* milik .NET tidak dapat digunakan kecuali pada produk Unity yang berlisensi profesional. Adanya batasan terhadap *library socket* milik .NET menyebabkan dicarinya alternatif lain pada pengiriman data melalui Java *plugin*. Lebih lanjut hal ini akan dibahas pada perancangan arsitektur sistem.

3.1.3 Aktor

Aktor mendefinisikan entitas-entitas yang terlibat dan berinteraksi langsung dengan sistem. Entitas ini bisa berupa manusia maupun sistem atau perangkat lunak yang lain. Aktor yang terdapat pada sistem ini hanya memiliki sebuah peran yaitu sebagai pemain Card Warlock Saga.

3.1.4 Kasus Penggunaan

Berdasarkan analisis spesifikasi kebutuhan fungsional dan analisis aktor dari sistem dibuat kasus penggunaan sistem. Kasus-kasus penggunaan dalam sistem ini akan dijelaskan secara rinci. Kasus penggunaan digambarkan dalam sebuah diagram kasus penggunaan. Kasus penggunaan dapat dilihat pada Gambar 3.1.



Gambar 3.1 Diagram Kasus Penggunaan Modul Pertarungan Pemain Lawan Pemain

Tabel 3.1 Spesifikasi Kasus Penggunaan

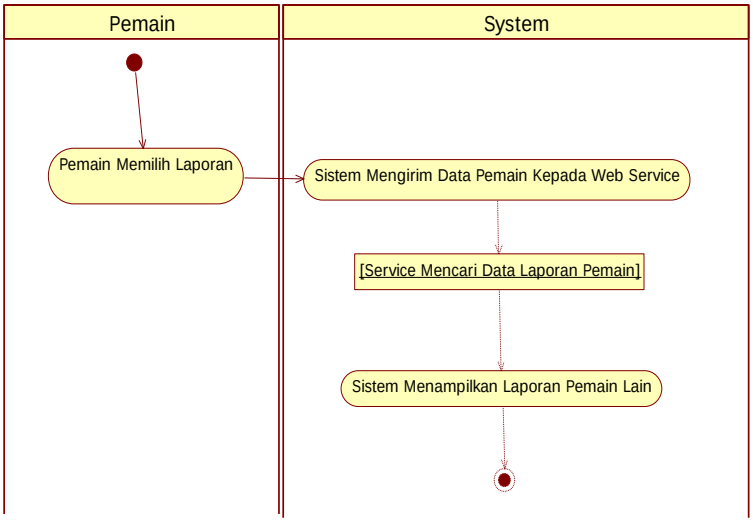
Kode Kasus Penggunaan	Nama
UC-0001	Melihat Laporan Pertarungan
UC-0002	Pertarungan Pemain Lawan Pemain
UC-0003	Manajemen Kartu
UC-0004	Manajemen Anggota

3.1.4.1 *Menampilkan Laporan Pertarungan*

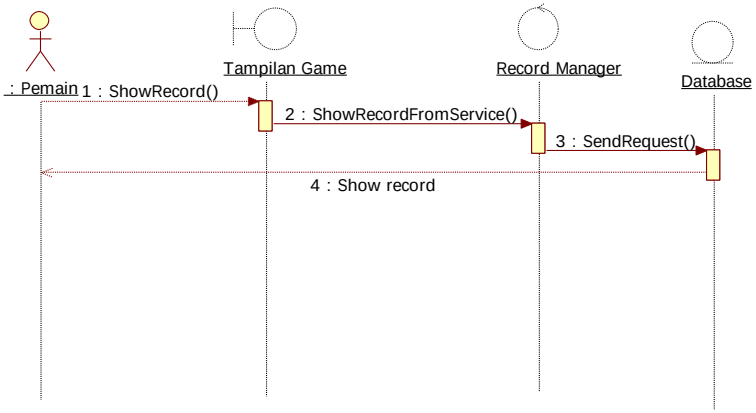
Sistem dapat menampilkan laporan hasil pertarungan pemain lawan pemain dari *record* yang terdapat di dalam *web service*. Sistem menerima masukan berupa nama pemain dan *record* yang dipilih untuk ditampilkan. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 3.2. Diagram aktivitas dan sekuen dari kasus penggunaan ini bisa dilihat pada Gambar 3.2 dan Gambar 3.3.

Tabel 3.2 Spesifikasi Kasus Penggunaan Menampilkan Hasil Pertarungan

Nama	Menampilkan laporan pertarungan
Kode	UC-0001
Deskripsi	Menampilkan laporan pertarungan pemain lawan pemain yang dipilih untuk ditampilkan. Kode program berupa keseluruhan kelas tempat *method* yang dideteksi berada.
Tipe	Fungsional
Pemicu	Pengguna memilih *record* yang ingin ditampilkan
Aktor	Pemain
Kondisi Awal	Pengguna sudah berada pada halaman *lobby*
Aliran: - **Kejadian Normal**	1. Pengguna memilih *record* yang ingin ditampilkan. 2. Sistem mengambil *record* pertarungan dari kelas *entity*. 3. Sistem memasukkan data ini ke dalam bagian tampilan kode sumber sistem.
- **Kejadian Alternati**	Tidak ada
Kondisi Akhir	Sistem menampilkan laporan pemain lawan pemain
Kebutuhan Khusus	Tidak ada



Gambar 3.2 Diagram Aktivitas Menampilkan Laporan Pertarungan



Gambar 3.3 Diagram Sekuen Menampilkan Laporan Pertarungan

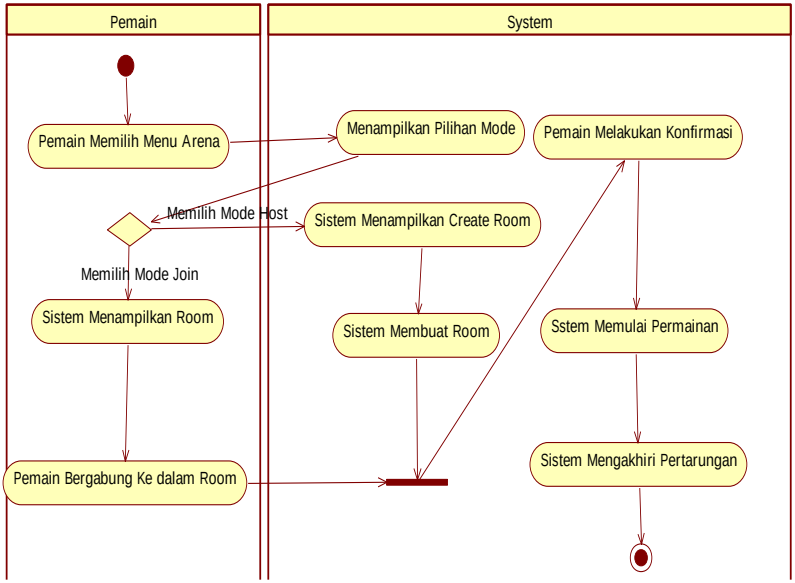
3.1.4.2 *Pertarungan Pemain Lawan Pemain*

Pada kasus penggunaan ini akan diimplementasikan sebuah aturan main yang memungkinkan adanya pertarungan pemain lawan pemain dengan menggunakan *state pattern*. Sistem akan menerima masukan dari salah satu pemain, kemudian pesan disebarkan secara *end-to-end* dengan menggunakan jaringan *client server*. Pada sisi *server* pesan akan diterjemahkan menggunakan protokol sedangkan pada sisi klien akan diterjemahkan menggunakan *command pattern*. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 3.3. Diagram aktivitas dan sekuen dari kasus penggunaan ini bisa dilihat pada Gambar 3.4 dan lampiran B Gambar B. 6.

Tabel 3.3 Spesifikasi Kasus Pertarungan Pemain Lawan Pemain

Nama	Pertarungan Pemain Lawan Pemain
Kode	UC-0002
Deskripsi	Penentuan aturan main dari kasus penggunaan pemain lawan pemain dengan menggunakan *pattern* dan pengiriman pesan lewat jaringan.
Tipe	Fungsional
Pemicu	Pengguna memilih menu *colloseum*
Aktor	Pemain
Kondisi Awal	Pemain sudah *login* ke dalam *server*
Aliran: – **Kejadian Normal**	1. Pemain melakukan *hosting room* A1 Pemain memilih *mode join room* 2. Pemain saling memberikan konfirmasi kepada *server* bahwa permainan boleh dimulai. 3. *Server* memberikan pesan pemicu berupa pesan *Start game.* 4. Pemain memulai pertarungan. 5. Ketika pemain *disconnect* atau *health*-nya kosong pada permainan maka dianggap kalah.

Nama	Pertarungan Pemain Lawan Pemain
– **Kejadian Alternati**	6. Ketika pemain menghabiskan *health* lawan maka dianggap menang. A1 Pemain memilih *mode join room.* 1. Pemain melakukan *discovery room*. 2. Pemain melakukan *join room.* 3. Kembali ke aliran kejadian normal 2.
Kondisi Akhir	Sistem menampilkan pemain menang atau kalah.
Kebutuhan Khusus	Koneksi ke *Server*



Gambar 3.4 Diagram Aktivitas Pemain Lawan Pemain

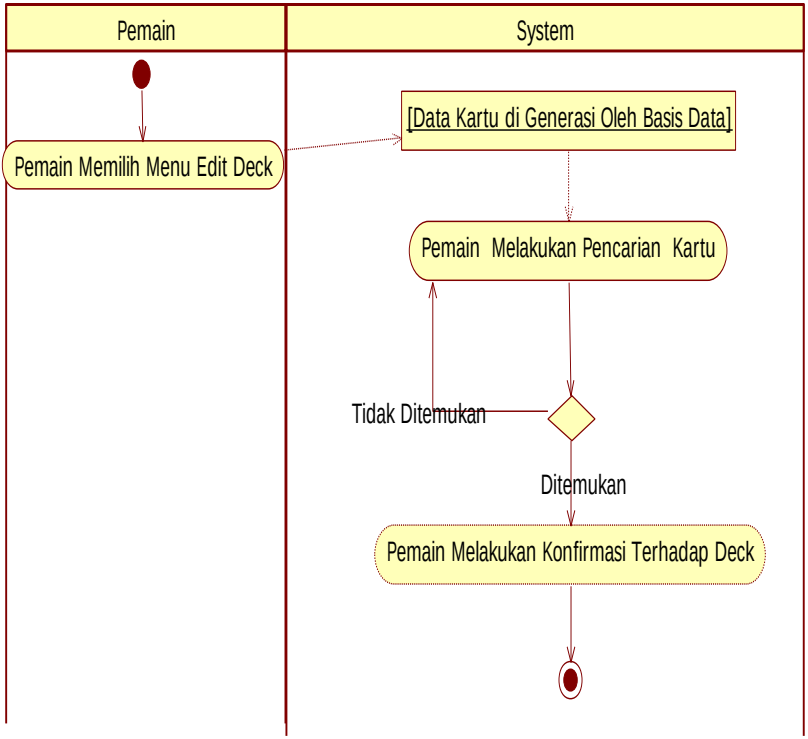
3.1.4.3 *Manajemen Kartu*

Pada kasus penggunaan ini, diimplementasikan sebuah editor untuk manajemen kartu yang memungkinkan pemain untuk mengubah susunan kartu miliknya. Pemain dapat menambahkan dan mengurangi kartu pada *deck*-nya. Pada sistem turut disertakan pencarian agar pemain dapat mencari kartu yang terdapat pada *trunk* kartunya ke dalam *deck.* Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 3.4. Diagram aktivitas dan sekuen dari kasus penggunaan ini bisa dilihat pada Gambar 3.5 dan Gambar 3.6.

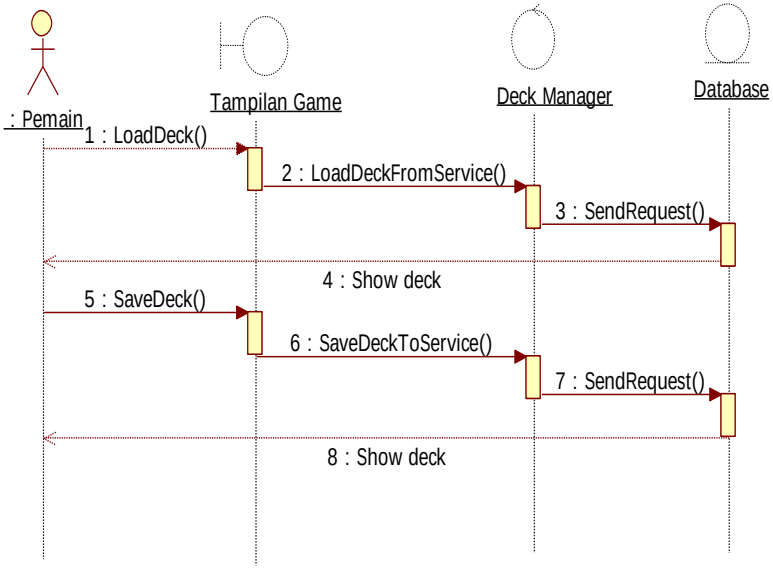
Tabel 3.4 Spesifikasi Kasus Manajemen Kartu

Nama	Manajemen Kartu
Kode	UC-0003
Deskripsi	Pada kasus penggunaan manajemen kartu pengguna dapat menambahkan atau mengurangi jumlah kartu yang terdapat di dalam *deck* miliknya. Pemain juga dapat mencari kartu di dalam *trunk.*
Tipe	Fungsional
Pemicu	Pengguna memilih menu *edit deck*
Aktor	Pemain
Kondisi Awal	Pemain sudah *login* ke dalam permainan
Aliran: – **Kejadian Normal**	1. Pemain memilih menu *edit deck*. 2. Pemain dapat melakukan *drag and drop* pada kartu untuk memindahkan kartu dari *trunk* ke dalam *deck*. 3. Pemain melakukan konfirmasi dengan menekan tombol *confirm*.

Nama	Manajemen Kartu
– **Kejadian Alternatif**	-
Kondisi Akhir	Sistem menampilkan pesan bahwa konfirmasi *deck* telah berhasil dilakukan.
Kebutuhan Khusus	Tidak ada



Gambar 3.5 Diagram Aktivitas Manajemen Kartu



Gambar 3.6 Diagram Sekuen Manajemen Kartu

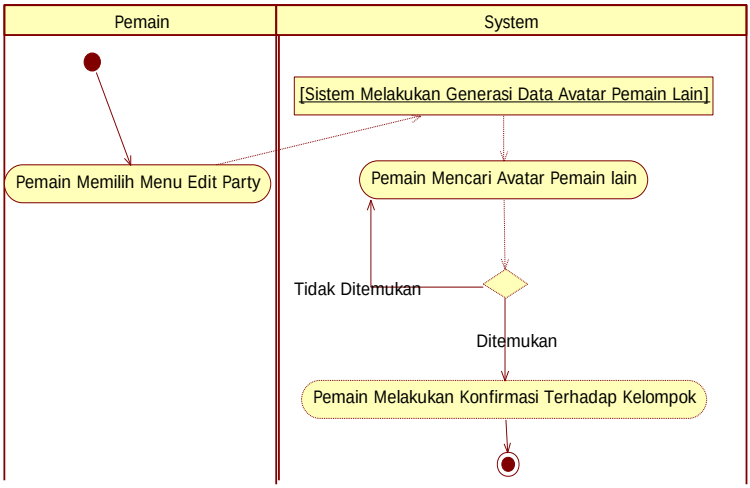
3.1.4.4 *Manajemen Pemain*

Pada kasus penggunaan ini, diimplementasikan sebuah editor untuk manajemen pemain yang memungkinkan pemain untuk mengubah susunan dari *party* saat ingin memulai permainan. Pemain dapat menambahkan *avatar* milik temannya yang terdaftar pada basis data untuk bertarung bersama pada *dungeon*. Spesifikasi kasus penggunaan ini dapat dilihat pada Tabel 3.5. Diagram aktivitas dan sekuen dari kasus penggunaan ini bisa dilihat pada Gambar 3.7 dan Gambar 3.8.

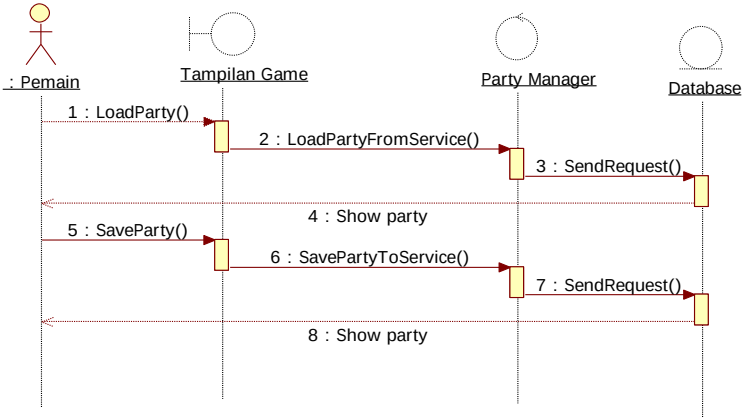
Tabel 3.5 Spesifikasi Kasus Manajemen Pemain

Nama	Manajemen Pemain
Kode	UC-0004

Nama	Manajemen Pemain
Deskripsi	Pengguna dapat mengubah susunan *party* yang akan dibawa saat bertarung.
Tipe	Fungsional
Pemicu	Pengguna memilih menu *edit party.*
Aktor	Pemain
Kondisi Awal	Sistem sudah terhubung dengan *web service.*
Aliran: – **Kejadian Normal**	1. Pengguna mencari *icon* dari *avatar* yang akan ditambahkan ke dalam *party* 2. Pengguna menekan tombol c*onfirm*. .
– **Kejadian Alternatif**	Tidak ada
Kondisi Akhir	Sistem menampilkan pesan bahwa konfirmasi *party* telah sukses dilakukan.
***Kebutuhan* Khusus**	Tidak ada



Gambar 3.7 Diagram Aktivitas Manajemen Pemain



Gambar 3.8 Diagram Sekuen Manajemen Pemain

3.1.5 Kebutuhan Fungsional

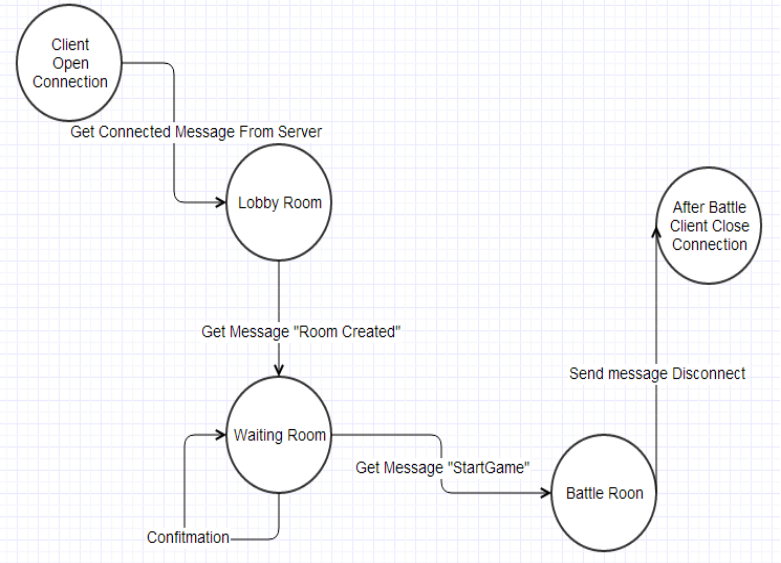
Kebutuhan fungsional berisi proses-proses yang harus dimiliki sistem. Kebutuhan fungsional mendefinisikan layanan yang harus disediakan dan reaksi sistem terhadap masukan atau pada situasi tertentu. Daftar kebutuhan fungsional dapat dilihat pada Tabel 3.6. Adapun kebutuhan fungsional nantinya akan diimplementasikan sebagai fitur yang terdapat pada aplikasi. Implementasi lebih lanjut dari kebutuhan fungsional dapat dilihat pada bab 3.

Tabel 3.6 Daftar Kebutuhan Fungsional Modul Pertarungan

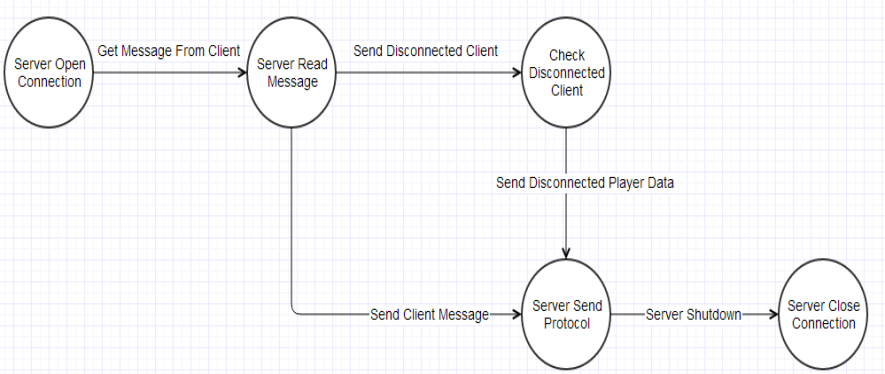
Kode Kebutuhan Fungsional	Kebutuhan Fungsional	Deskripsi
F-001	Menampilkan laporan pertarungan	Pengguna dapat melihat laporan pertarungan yang telah dilalui
F-002	Menjalankan pertarungan pemain lawan pemain	Pengguna dapat melakukan pertarungan antar pemain
F-003	Manajemen pemain	Pengguna dapat melakukan pergantian *party*
F-004	Manajemen kartu	Pengguna dapat melakukan pergantian kartu

3.1.6 Analisis Kegagalan Sinkronisasi

Pada subbab ini akan digambarkan komunikasi *server* dan klien hal ini guna mengidentifikasi kegagalan-kegagalan apa saja yang dapat terjadi ketika melakukan komunikasi antara *server* dan klien.



Gambar 3.9 *State Machine Diagram* Klien Pemain Lawan Pemain



Gambar 3.10 *State Machine Diagram* Server Pemain Lawan Pemain

Dari hasil analisis pada *state machine diagram* pada gambar Gambar 3.9 dan Gambar 3.10 memungkinkan terjadinya kegagalan pada aliran sinkronisasi data. Kegagalan-kegagalan tersebut di antaranya adalah kegagalan koneksi dengan *server* dan pemain terputus dari *server* ketika bermain.

3.1.6.1 *Kegagalan Koneksi Dengan Server*

Kegagalan di saat koneksi *server* dapat diakibatkan oleh hilang atau lambatnya koneksi antara *server* dan klien. Kegagalan tersebut menyebabkan pesan yang harusnya diterima oleh klien tidak sampai pada tujuan. Untuk mengatasi adanya kegagalan pada pengiriman pesan akan diimplementasikan fungsi untuk melakukan pengecekan terhadap jumlah waktu yang dibutuhkan untuk melakukan koneksi antara klien dan *server*, bila klien tidak bereaksi pada jangka waktu yang ditentukan, maka akan dianggap *timeout*. Untuk mengatasi permasalahan tersebut akan dirancang suatu fungsi di mana pada fungsi tersebut akan dilakukan pengecekan terhadap waktu koneksi apabila waktu berjalan lebih dari lima belas detik, maka akan dianggap *timeout* dan dimunculkan pesan bahwa pemain gagal terhubung dengan *server.*

3.1.6.2 *Pemain Terputus dari Server*

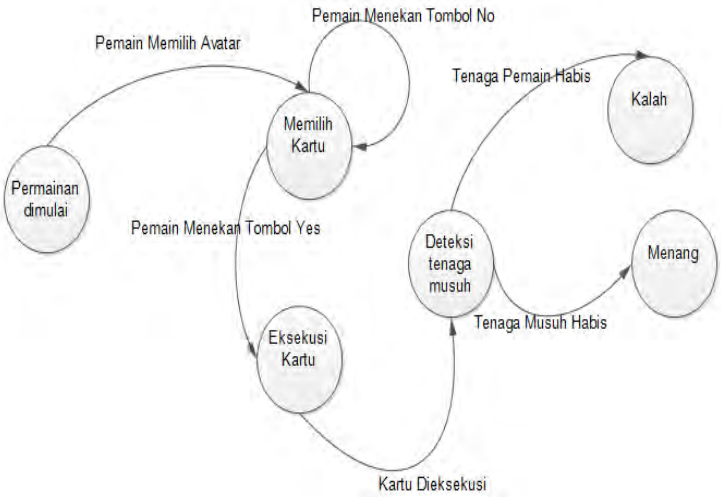
Kegagalan selanjutnya adalah apabila klien yang terhubung dengan *server* terputus di saat permainan dimulai. Pemain yang terputus dari koneksi akan dianggap kalah sedangkan pemain yang masih terhubung dengan *server* akan dianggap menang.

3.2 Perancangan Sistem

Penjelasan tahap perancangan perangkat lunak dibagi menjadi beberapa bagian, yaitu perancangan aturan main, perancangan diagram kelas, dan perancangan antarmuka.

3.2.1 Perancangan Aturan Main

Perancangan aturan main berisi mengenai rancangan dari permainan modul pertarungan. Pada perancangan aturan main akan dirancang suatu aturan main yang sesuai dengan tema dari permainan Card Warlock Saga. *State machine diagram* dapat dilihat pada Gambar 3.11.



Gambar 3.11 *State Machine Diagram* Aturan Main Card Warlock Saga

Pemain bermain dengan cara membawa sejumlah kartu pada *deck* miliknya. Pemain dapat mengajak *avatar* lain milik temannya pada saat akan menjalankan pertarungan. Pemain dianggap kalah ketika tenaga yang dimilikinya habis atau bila kartu yang ada pada *deck*-nya habis. Pemain menang bila berhasil mengalahkan musuh.

3.2.2 Perancangan *Server*

Pada subbab ini akan dijelaskan mengenai perancangan *server* yang ada pada *game*. *Server* dan klien diimplementasikan dengan menggunakan kerangka kerja Kryonet. Kerangka kerja Kryonet merupakan kerangka kerja yang berfungsi sebagai instansiasi kelas *server*. Kelas *server* yang diinstansiasi pada *server* adalah salah satu kelas *library* milik kerangka kerja Kryonet. Pada pengimplementasiannya akan dibuatkan sebuah kelas *listener* pada *server* yang bertugas untuk menangani *state* yang terjadi ketika *connect, disconnect,* dan *receive message*. Kerangka kerja Kryonet

sudah bersifat *multiclient* sehingga pada penggunaannya tidak perlu melakukan konfigurasi terhadap *thread* secara manual.

3.2.3 Perancangan Diagram Kelas

Perancangan diagram kelas berisi rancangan dari kelas-kelas yang digunakan untuk membangun sistem. Pada subbab ini, hubungan dan perilaku antar kelas digambarkan dengan lebih jelas. Tiga lapisan pada arsitektur ini terdiri dari lapisan antarmuka, kontrol, dan *listener*. Lapisan kontrol merupakan penghubung antara lapisan antarmuka dengan lapisan model. Subbab ini dibagi menjadi tiga bagian, yaitu diagram kelas untuk lapisan model, kontrol, dan *listener*.

3.2.3.1 *Diagram Kelas Lapisan Model*

Pada diagram kelas lapisan model terdapat kelas-kelas yang dijadikan acuan untuk implementasi obyek pada permainan. Pada modul pertarungan Card Warlock Saga dibutuhkan perancangan kelas model yang memungkinkan pemain melawan musuh yang memiliki *element* yang beraneka ragam, maka dirancang satu kelas turunan dari kelas *enemy* berupa *fire enemy*, *water enemy*, dan *wind enemy*. Untuk memenuhi kebutuhan permainan, satu profesi hanya bisa membawa kartu dari salah satu *element saja*. Oleh sebab itu dirancang kelas-kelas profesi yang merupakan turunan dari kelas `player`. Kelas-kelas itu adalah `sorcerer`, `wizard`, `magician`, `grandmagus`, dan `warlock`. Kelas `DamageReceiver` dibuat untuk proses kalkulasi *hit points* pada permainan. Gambar lapisan kelas model dapat dilihat pada lampiran B Gambar B. 3.

Kelas-kelas lapisan kontrol adalah kelas-kelas yang dirancang untuk menginstansiasi kelas-kelas pada lapisan model. Kelas lapisan kontrol nantinya tidak akan diinstansiasi namun ditempelkan sebagai komponen pada Unity, dikarenakan Unity yang berbasis komponen. Tugas dari kelas kontrol adalah melakukan perpindahan data dari komponen ke dalam model.

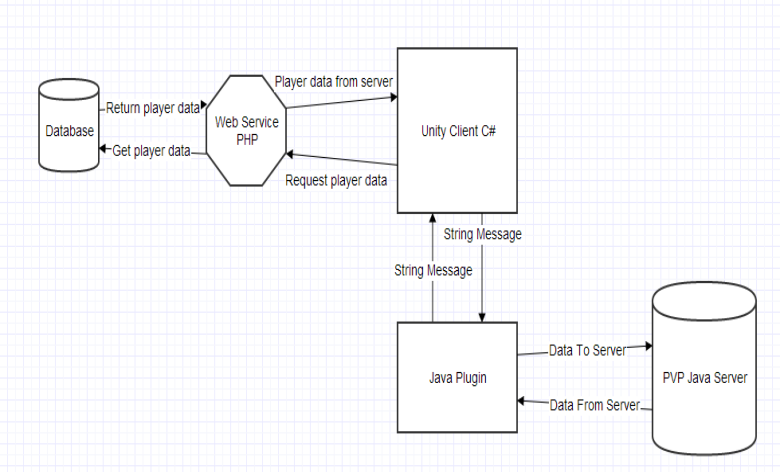
Unity merupakan *game engine* yang berbasis komponen sehingga penerapan OOP pada Unity harus disesuaikan dengan basis komponen. Lapisan kelas kontrol merupakan turunan dari kelas utama milik Unity yaitu `MonoBehaviour` yang berfungsi agar kelas-kelas lapisan kontrol dapat ditambahkan sebagai komponen pada *game object*. Hirarki pada kelas-kelas lapisan kontrol dibentuk layaknya pada kelas model. Hal ini memudahkan untuk melakukan instansiasi kelas pada Unity yang berbasis komponen *scripting.* Rancangan diagram kelas lapisan model dapat dilihat pada lampiran B Gambar B. 4.

3.2.3.2 *Diagram Kelas Lapisan Listener*

Kelas lapisan *listener* adalah kelas-kelas yang berfungsi menerjemahkan pesan yang dikirimkan oleh *server*. Pesan akan dikirimkan dari *server* dalam bentuk *string* untuk selanjutnya diterjemahkan menggunakan *command pattern* untuk mengeksekusi perintah yang sesuai dengan pesan yang diterima. Diagram kelas lapisan *listener* dapat dilihat pada lampiran B Gambar B. 5.

3.2.4 Perancangan Arsitektur Perangkat Lunak

Bagian ini membahas mengenai perancangan arsitektur sistem, keterkaitan penggunaan bahasa C#, Java, dan PHP, dan alur aplikasi sistem. Agar perangkat *mobile* dapat berkomunikasi dengan *server* dibutuhkan kelas *library socket* pada .NET, namun dengan adanya limitasi terhadap penggunaan *library socket* milik .NET menyebabkan adanya penggunaan *library socket* milik Java. *Library socket* milik Java digunakan sebagai *plugin* pada Unity. Ketika permainan dijalankan, secara otomatis *plugin* yang berada pada program berjalan pada *background*. Penggambaran alur yang lebih jelas dapat dilihat pada Gambar 3.12.



Gambar 3.12 Arsitektur Perangkat Lunak Modul Pemain Lawan Pemain

Dapat dilihat bahwa sistem mengatasi keterbatasan penggunaan *socket* .NET dengan penggunaan Java *plugin*, yang di dalamnya terdapat kerangka kerja Kryonet yang mempunyai Java *socket.* Sementara itu PHP *web service* digunakan untuk mengambil data yang berasal dari *database server* yang berbeda. Sistem menggunakan dua buah *server* yang berbeda. *Server* yang pertama bertugas sebagai penyimpan data-data yang berhubungan dengan *game* dan yang kedua berfungsi untuk melakukan *broadcast* data pemain lawan pemain. Java *plugin* berfungsi menerjemahkan data objek menjadi pesan bertipe data *string*. Adapun kekurangan penggunaan banyak bahasa pada sistem adalah susah untuk di-*maintenance* dikarenakan penggunaan arsitektur bahasa yang berbeda-beda.

3.2.4.1 *Unity Client*

Unity *client* adalah aplikasi utama yakni permainan yang dibangun menggunakan Unity dengan bahasa pemrograman C#. Pada permainan utama akan dilakukan pertukaran data protokol

berupa *string message* dengan Java *plugin* menggunakan kelas `AndroidUnityListener`. Selain data protokol Unity *client* juga bertukar data pemain melalui *web service* PHP yang terhubung dengan *database* sehingga Unity *client* dapat menginstansiasi pemain yang terdaftar pada *database.*

3.2.4.2 *Java Plugin*

Java *plugin* adalah aplikasi pendukung yang berfungsi menerjemahkan data objek dari *server* menjadi pesan yang bertipe data *string* dan mengirimkannya ke Unity *client.* Java *plugin* berada pada *background* aplikasi permainan Unity *client*, Java *plugin* diimplementasikan dengan kelas `AndroidUnityListener` dan kerangka kerja Kryonet.

3.2.4.3 *Java Server*

Java *server* adalah *server* permainan pemain lawan pemain, yang berfungsi melakukan *broadcast* protokol pada pemain. *Server* ini diimplementasikan dengan kerangka kerja Kryonet.

3.2.5 Perancangan Protokol *Client Server*

Bagian ini membahas rancangan protokol antara *client* dan *server*. Agar perangkat *mobile* dapat berkomunikasi dengan *server* maka dibutuhkan suatu protokol. Protokol berfungsi untuk mengontrol alur permainan ketika pemain memilih mode *online*. Daftar protokol yang digunakan adalah sebagai berikut.

1.	Nama protokol	:	*Create room*
	Syntax	:	CreateRoom|||[room name]|||[player name]
	Parameter	:	*Player name, Room name*
	Fungsi	:	Menciptakan *room* pada *server*
	Kembalian	:	CreatedRoomSuccess
	Tipe data kembalian	:	*String*

2. Nama protokol : *Join room*
   Syntax : JoinRoom||[room name]||[player name]
   Parameter : *Room name*, *Player name*
   Fungsi : Melakukan *join room*
   Kembalian : JoinedRoom[room name]
   Tipe data kembalian : *String*

3. Nama protokol : *Confirmation*
   Syntax : Confirmation||[room name]||[player name]
   Parameter : *Player name*,Room *name*
   Fungsi : Melakukan konfirmasi sebelum permainan dimulai
   Kembalian : ConfirmationForPlay
   Tipe data kembalian : *String*

4. Nama protokol : *Start Game*
   Syntax : StartGame||[room name]||[player name]
   Parameter : *Player name, Room name*
   Fungsi : Memulai *game* setelah dilakukan konfirmasi
   Kembalian : StartGame

5. Nama protokol : *Get player list*
   Syntax : GetPlayerlist||[room name]
   Parameter : *Room Dame*
   Fungsi : Melihat daftar pemain pada *rim*
   Kembalian : PlayerList||[player name]
   Tipe data kembalian : *String*

6. Nama protokol : *Get room list*
   Syntax : GetRoomList

	Parameter	:	Tidak ada
	Fungsi	:	Melihat daftar *room* yang ada
	Kembalian	:	RoomList||[room name]
7.	Nama protokol	:	*Card effect*
	Syntax	:	CardEffect||[card name]||[target name]
	Parameter	:	*Card name, Target Name*
	Fungsi	:	Mengirimkan nama kartu untuk diterjemahkan sebagai efek pada *game*
	Kembalian	:	CardEffect||[card name]||[targert name]
	Tipe data kembalian	:	*String*
8.	Nama protokol	:	*End turn*
	Syntax	:	EndTurn||[room name]||[player name]
	Parameter	:	*Room name, Player name*
	Fungsi	:	Menginformasikan pemain lain bahwa pemain tersebut telah mengakhiri gilirannya
9.	Nama protokol	:	*Chat*
	Syntax	:	Chat-[room name]||[player name]||[message]
	Parameter	:	*Room name, Player name, Message*
	Fungsi	:	Melakukan *chatting* antar pemain
	Kembalian	:	Chat-[room name]||[player name]||[message]
	Tipe data kembalian		*String*
10.	Nama protokol	:	*Disconnect*

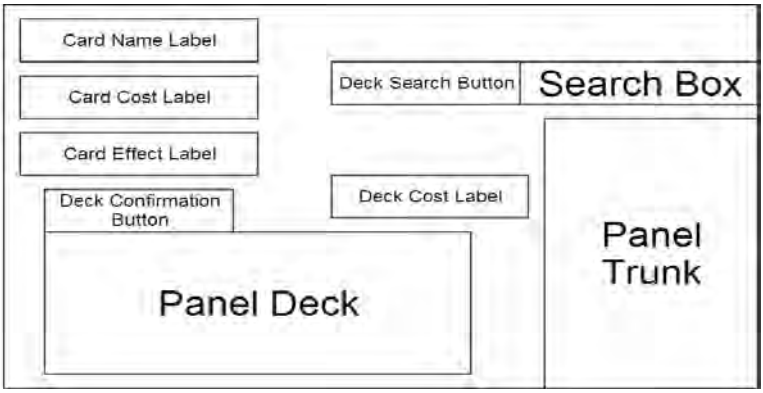
Syntax	:	Disconnect||[Room] name||[Player name]
Parameter	:	*Room name, Player Name*
Fungsi	:	Menginformasikan pada klien lain jika ada klien yang terputus dari jaringan
Kembalian	:	Disconnect||[Room] name||[Player name]
Tipe data kembalian	:	*String*

3.2.6 Perancangan Antarmuka Pengguna

Bagian ini membahas rancangan tampilan antarmuka pada sistem. Pada sistem terdapat banyak tampilan yang digunakan pemain untuk mengubah susunan kartu, susunan pemain, dan tampilan utama modul pertarungan.

3.2.6.1 *Halaman Tampilan Manajemen Kartu*

Halaman ini merupakan tampilan yang muncul ketika pemain memilih mode *edit deck*. Pada halaman ini terdapat panel *deck* dan panel *trunk*. Pengguna dapat memindahkan kartu dari dalam panel *trunk* ke dalam panel *deck*, kemudian pemain dapat melakukan konfirmasi. Posisi dan besar ukuran halaman dapat diubah sesuai dengan ukuran perangkat bergerak pengguna. Fitur lainnya adalah pencarian kartu pemain dapat mencari kartu miliknya dari dalam *trunk* dengan mengetik nama dari kartu. Rancangan tampilan dapat dilihat pada Gambar 3.13. Spesifikasi atribut antar muka tampilan dapat dilihat pada Tabel 3.7.



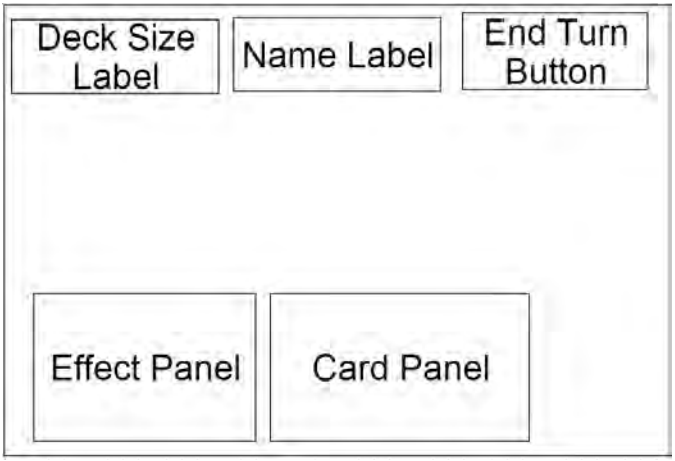
Gambar 3.13 Rancangan Antarmuka Halaman Manajemen Kartu

Tabel 3.7. Spesifikasi Atribut Antarmuka Tampilan Manajemen Kartu

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan
1	Panel *Trunk*	*Grid*	Menampilkan seluruh kartu yang dimiliki pemain
2	Panel *Deck*	*Grid*	Menampilkan kartu yang ada pada *dek*
3	*Search Box*	*Label*	Masukan pencarian kartu
4	*Card Name Label*	*Label*	Menampilkan nama dari kartu
5	*Card Cost Label*	*Label*	Menampilkan *cost* dari kartu
6	*Deck Cost Label*	*Label*	Menampilkan *deck cost* dari kartu
7	*Card Effect Label*	*Label*	Menampilkan efek dari kartu
8	*Deck Confirmation Button*	*Button*	Melakukan konfirmasi terhadap *deck*
9	*Deck Search Button*	*Button*	Melakukan fungsi pencarian

3.2.6.2 *Halaman Tampilan Pertarungan*

Halaman tampilan pertarungan adalah halaman utama pada modul pertarungan pemain lawan pemain. Gambar 3.14 merupakan tampilan yang muncul ketika pemain melakukan pertarungan. Pada halaman ini terdapat panel yang berisi keterangan mengenai efek kartu dan panel yang berfungsi untuk menampilkan kartu, selebihnya terdapat label yang menunjukkan informasi mengenai *health player*, *deck size, soul points*, dan nama pemain. Pemain dapat menyerang musuh dengan menekan salah satu kartu yang terdapat pada panel. Tombol *end turn* ditekan ketika pemain hendak mengakhiri gilirannya. Spesifikasi atribut antarmuka tampilan dapat dilihat pada Tabel 3.8.



Gambar 3.14 Rancangan Antarmuka Pertarungan

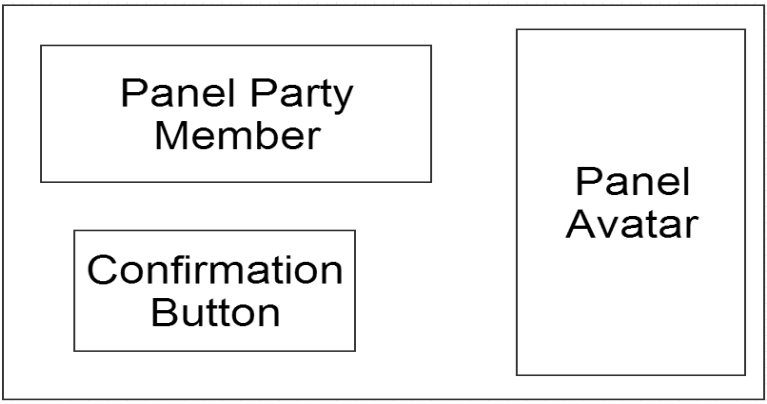
Pemain dapat menyerang musuh dengan menekan salah satu kartu yang terdapat pada panel. Tombol *end turn* ditekan ketika pemain hendak mengakhiri gilirannya. Spesifikasi atribut antarmuka tampilan dapat dilihat pada Tabel 3.8.

Tabel 3.8 Spesifikasi Atribut Antarmuka Tampilan Pertarungan

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan
1	*Card Panel*	*Grid*	Menampilkan seluruh kartu yang dimiliki oleh pemain
2	*Effect Panel*	*Grid*	Menampilkan efek kartu yang dipilih pemain
3	*Deck Size Label*	*Label*	Menampilkan jumlah sisa kartu pada *deck*
4	*Name Label*	*Label*	Menampilkan nama pemain
5	*End Turn Button*	*Button*	Mengakhiri giliran pemain

3.2.6.3 *Halaman Tampilan Manajemen Pemain*

Halaman tampilan manajemen pemain adalah halaman yang berfungsi sebagai editor untuk menambah atau mengurangi *party member*. Halaman tersebut dapat dilihat pada Gambar 3.15.



Gambar 3.15 Rancangan Antar Muka Manajemen Pemain

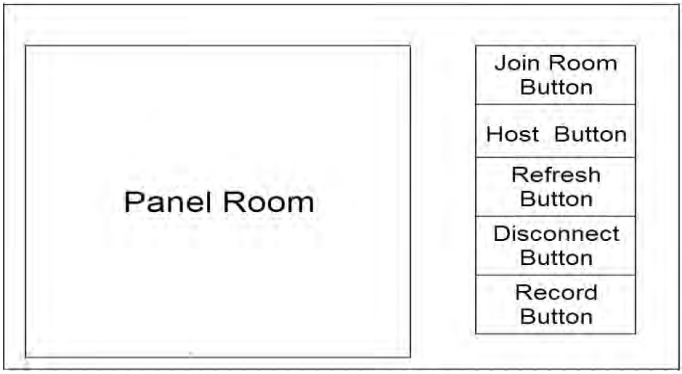
Gambar 3.15 merupakan rancangan halaman yang muncul ketika pemain memilih mode *edit party*. Pemain dapat melihat *avatar* milik temannya untuk ditambahkan ke dalam *party* member dengan cara *drag and drop*. Spesifikasi atribut antarmuka tampilan dapat dilihat pada Tabel 3.9.

Tabel 3.9 Spesifikasi Atribut Antarmuka Ruang Manajemen Pemain

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan
1	Panel *Avatar*	*Grid*	Menampilkan seluruh *avatar* milik teman
2	Panel *Party Member*	*Grid*	Bergabung dengan *room* yang masih aktif
3	*Confirmation Button*	*Button*	Melakukan konfirmasi *party*

3.2.6.4 *Halaman Tampilan Ruang Lobi*

Halaman tampilan ruang lobi adalah halaman yang berfungsi sebagai menu utama pada modul pertarungan pemain lawan pemain. Halaman tersebut dapat dilihat pada Gambar 3.16.



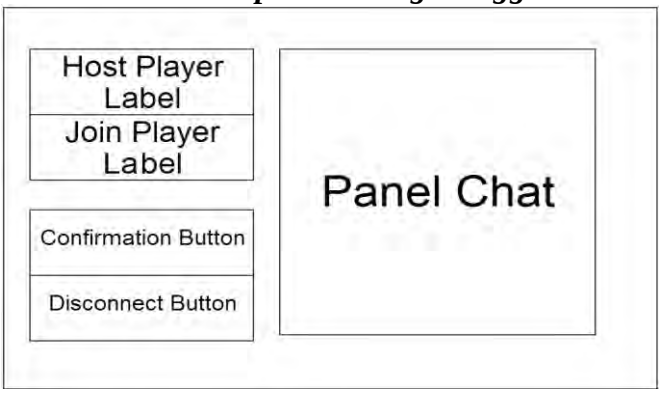
Gambar 3.16 Rancangan Tampilan Ruang Lobi

Gambar 3.16 merupakan tampilan yang muncul ketika pemain memilih mode pemain lawan pemain. Halaman ini bertujuan menunjukkan *room* yang aktif pada saat pemain ingin melakukan pertarungan pemain lawan pemain. Spesifikasi atribut antarmuka tampilan dapat dilihat pada Tabel 3.10.

Tabel 3.10 Spesifikasi Atribut Antarmuka Ruang Lobi

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan
1	Panel *Room*	*Grid*	Menampilkan seluruh *room* yang sedang aktif
2	*Join Room Button*	*Button*	Bergabung dengan *room* yang masih aktif
2	*Refresh Button*	*Button*	Mendapatkan *room list* terbaru
4	*Host Button*	*Button*	Berpindah ke dalam halaman *host*
5	*Disconnect Button*	*Button*	Mengakhiri koneksi ke *server*
6	*Record Button*	*Button*	Berpindah ke halaman *record*

3.2.6.5 *Halaman Tampilan Ruang Tunggu*



Gambar 3.17 Rancangan Tampilan Ruang Tunggu

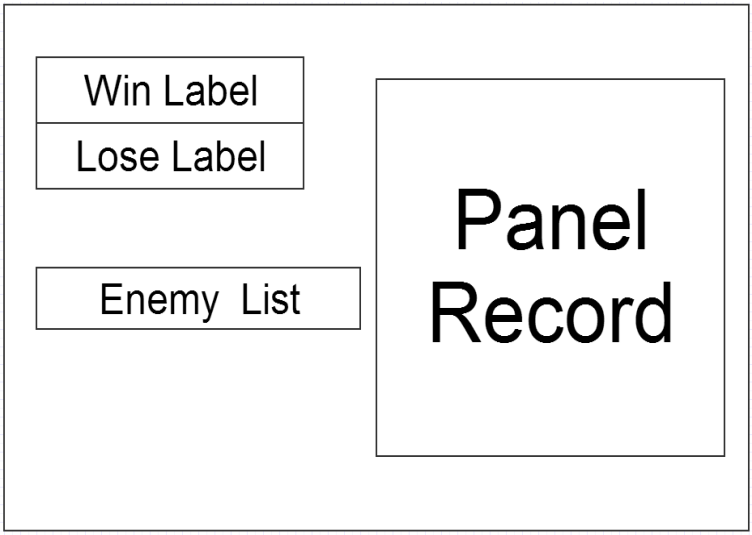
Halaman tampilan ruang tunggu adalah halaman yang berfungsi sebagai ruang tunggu pada modul pertarungan pemain lawan pemain. Gambar 3.17 merupakan tampilan ruang tunggu pemain sebelum pemain melakukan pertarungan pemain lawan pemain. Pemain yang bertindak sebagai *host* dapat memulai permainan, dengan syarat kedua pemain telah menekan tombol konfirmasi terlebih dahulu. Spesifikasi atribut antarmuka tampilan dapat dilihat pada Tabel 3.11.

Tabel 3.11 Spesifikasi Atribut Antarmuka Ruang Tunggu

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan
1	Panel *Chat*	*Grid*	Menampilkan pesan dari *server* dan *chat* pemain lain
2	*Host Player Label*	*Label*	Menampilkan nama pemain yang menjadi *Hot*
3	*Join Player Label*	*Label*	Menampilkan nama pemain yang menjadi *join player*
4	*Disconnect Button*	*Button*	Mengakhiri koneksi ke *server*
5	*Confirmation Button*	*Button*	Melakukan konfirmasi untuk memulai pertarungan

3.2.6.6 *Halaman Tampilan Record Pemain*

Gambar 3.18 adalah rancangan tampilan yang muncul ketika pemain memilih menu *record*. Halaman ini berfungsi menampilkan lima orang *ranking* teratas di dalam pertarungan pemain lawan pemain. Selain itu halaman ini juga berfungsi untuk menampilkan jumlah menang kalah pemain melawan pemain lain. Spesifikasi atribut antarmuka tampilan dapat dilihat pada Tabel 3.12.



Gambar 3.18 Rancangan Tampilan Halaman *Record*

Tabel 3.12 Spesifikasi Atribut Antarmuka Ruang *Record*

No	Nama Atribut Antarmuka	Jenis Atribut	Kegunaan
1	Panel *Record*	*Grid*	Menampilkan lima pemain dengan rekor tertinggi
2	*Win Label*	*Label*	Menampilkan jumlah pemain memenangkan pertarungan dengan pemain lain
3	*lose Label*	*Label*	Menampilkan jumlah kekalahan pemain dengan pemain lain
4	*Disconnect Button*	*Button*	Mengakhiri koneksi ke *serve*r
5	Enemy List	*Pop up List*	Menampilkan nama pemain yang telah dikalahkan pemain

BAB IV
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan sistem. Bab ini berisi proses implementasi dari setiap kelas pada semua modul. Bahasa pemrograman yang digunakan adalah bahasa pemrograman Java dan C#.

4.1 Implementasi Lapisan Model

Lapisan model merupakan lapisan kelas model yang bertugas sebagai entitas pada saat permainan berjalan. Pada bagian ini akan dijelaskan implementasi dari rancangan lapisan model.

4.1.1 Kelas `DamageReceiver`

Kelas ini merupakan kelas abstrak yang digunakan untuk perhitungan *damage* pada entitas objek permainan dengan menggunakan fungsi `ReceiveDamage` pada lampiran A Kode Sumber A. 1. Objek-objek yang terdapat pada kelas ini merupakan objek milik kelas Unity *engine*.

```
public virtual void ReceiveDamage(int damage)
{  if ((character.CurrentHealth - damage)<=0)
{
  character.CurrentHealth = 0;
}
else {
character.CurrentHealth -= damage;
     }
}
```

Kode Sumber 4.1.Fungsi `DamageReceiver`

4.1.2 Kelas `Player`

Kelas ini merupakan lapisan kelas abstrak yang berfungsi sebagai entitas objek pemain di dalam permainan. Kelas `Player` digunakan sebagai dasar dalam membangun kelas-kelas `sorcerer`, `warlock`, `magician`, `grandmagus`, dan `wizard`.

4.1.3 Kelas `Enemy`

Kelas ini merupakan kelas abstrak yang digunakan sebagai dasar dalam membangun kelas `FireEnemy`, `WaterEnemy`, `WindEnemy`, `EarthEnemy`, dan `ThunderEnemy`.

4.2 Implementasi Kasus Penggunaan

Pada bagian bagian ini dijelaskan mengenai implementasi kasus penggunaan dalam modul pertarungan pemain lawan pemain.

4.2.1 Implementasi Pertarungan Pemain Lawan Pemain

Agar kasus penggunaan pemain lawan pemain dapat diimplementasikan, dibutuhkan rancangan kelas yang dapat berkomunikasi dengan protokol pada *server*. Oleh karena itu diimplementasikan sebuah kelas yang berasal dari *state pattern* untuk menjalankan alur pertarungan.

4.2.1.1 *Kelas BattleState*

Kelas `BattleState` adalah kelas abstrak yang berfungsi sebagai cetak biru *state* yang dialami pemain di saat bermain. Fungsi utama kelas ini adalah sebagai penentu alur permainan. Rincian dapat dilihat pada lampiran A Kode Sumber A. 7.

4.2.1.2 *Kelas BattleObjectLoader*

Kelas ini berfungsi untuk melakukan *load* objek-objek yang berkaitan dengan pertarungan. Objek-objek tersebut

adalah *character* pemain, musuh, dan *background*. Rincian dapat dilihat pada lampiran A Kode Sumber A. 15.

4.2.1.3 *Kelas AbstractFactory*

Kelas ini berfungsi sebagai cetak biru kelas-kelas yang berfungsi untuk menginstansiasi objek di dalam permainan. Kelas-kelas yang mengimplementasikan `AbstractFactory` adalah `EnemyFactory`, `PlayerFactory`, dan `OnlineEnemyFactory`.

4.2.1.4 *Kelas BattleStateManager*

Kelas `BattleStateManager` adalah kelas kontrol yang berhubungan dengan kelas `BattleState`. Kelas ini memegang *current state* alur permainan yang dimiliki oleh kelas `BattleState`. Kelas `BattleManager` melakukan perubahan *state* sesuai dengan alur permainan. Rincian dapat dilihat pada lampiran A Kode Sumber A. 8.

4.2.1.5 *Kelas NetworkManager*

Kelas `NetworkManager` merupakan *singleton pattern* yang berfungsi menghubungkan Unity dengan aplikasi Android yang berjalan di *background process*. Kelas `NetworkManager` memiliki fungsi yang berinteraksi dengan *socket* yang berada pada aplikasi *background*.

```
public void Connect()
        {
            var args = new string[3];
            args[0] = host;
            args[1] = tcpPort;
            args[2] = udpPort;
            UnityPlayer = new
AndroidJavaClass("com.Unity3d.player.UnityPlayer");
```

```
            activity =
UnityPlayer.GetStatic<AndroidJavaObject>("currentAc
tivity");
            activity.Call("runOnUiThread", new
AndroidJavaRunnable(() =>
            {
                playerClient = new
AndroidJavaObject("com.its.warlocksaga.AndroidUnity
Listener", args);

            }));
        }
        public void Disconnect()
        {
            PlayerClient.Call("CloseConnection");
        }
```

Kode Sumber 4.2 Fungsi Koneksi pada Kelas `NetworkSingleton`

Fungsi `Connect` pada Kode Sumber 4.2 berfungsi menjalankan aplikasi `AndroidUnityListener` sekaligus mengirimkan parameter berupa alamat IP, TCP *port*, dan UDP *port*.

4.2.1.6 *Kelas InvokerListener*

Kelas `InvokerListener` pada Kode Sumber 4.3 berfungsi untuk menerima protokol yang dikirimkan oleh *server*. Protokol kemudian akan diinstansiasi kelas-kelas implementasi *command pattern* sehingga menjadi sebuah perintah yang di implementasikan di dalam permainan.

```
private void Update()
        {
            if
(NetworkSingleton.Instance().ServerMessage != null)
            {
                var serverMessage =
NetworkSingleton.Instance().ServerMessage;
```

```
                try
                {
                    string[] message =
serverMessage.Split('-');
                    if
(serverMessage.Contains("CardEffect"))
                    {

text.GetComponent<UILabel>().text = serverMessage;
                        _invoke = new Invoker();
                        _cmd =
message[1].ToLower().Equals(GameManager.Instance().
PlayerId.ToLower())
                            ? new
CardExecuteCommand(message[2], "enemy")
                            : new
CardExecuteCommand(message[2], "player");
                        _invoke.AddCommand(_cmd);
                        _invoke.RunCommand();


NetworkSingleton.Instance().ServerMessage = "";
                    }
                    else if
(serverMessage.Contains("EndTurn"))
                    {

text.GetComponent<UILabel>().text = serverMessage;
                        _invoke = new Invoker();
                        try
                        {

battleStateManager.GetComponent<BattleStateManager>
().endButton.SetActive(true);
                            _cmd = new
EndPhaseCommand(battleStateManager.GetComponent<Bat
tleStateManager>());

                    _invoke.AddCommand(_cmd);
                        _invoke.RunCommand();
                      }
                      catch (Exception e)
```

```
                        {

                            Debug.Log("Endturn
Error"+e.Message);
                        }
NetworkSingleton.Instance().ServerMessage = "";
                    }
                    else if
(serverMessage.Contains("Chat"))
                    {

text.GetComponent<UILabel>().text = serverMessage;

textList.GetComponent<UITextList>().Add(message[1])
;

NetworkSingleton.Instance().ServerMessage = "";
                    }
                }
                catch (Exception e)
                {
        Debug.Log("errorInvokerListener" +
e.Message);
                }
            }
```

Kode Sumber 4.3 Fungsi `Update` pada Kelas `InvokerListener`

4.2.1.7 *Kelas AndroidUnityListener*

Kelas `AndroidUnitiListener` merupakan kelas yang melakukan koneksi dengan *server*. `AndroidUnityListener` menginisialisasi kelas *client* yang dimiliki oleh kerangka kerja Kryonet dan mengirim pesan kepada *server* dengan menggunakan fungsi `sendMessage` pada Kode Sumber 4.4.

```
public boolean sendMessage(String Message)
      {
            try
            {
                  Log.i(TAG,"Sending Message to
server, Message ="+Message);
                  pm= new PacketMessage();
                  pm.setMessage(Message);
                  client.sendTCP(pm);
                  return true;
            }
            catch(Exception i)
            {
                  Log.e(TAG,"unable to send
message"+i.getLocalizedMessage());
                  _listener.interrupt();
                  return false;
            }
      }
```

Kode Sumber 4.4 Fungsi *sendMessage* pada Kelas `AndroidUnityListener`

4.2.1.8 *Kelas ServerApplication*

Kelas ini adalah implementasi dari *server* modul pertarungan pemain lawan pemain. Pada kelas `ServerApplication` terdapat *framework* Kryonet yang berfungsi menginisiasi *server*. Fungsi utama dari kelas `ServerApplication` adalah melakukan *broadcast* protokol. Fungsi yang digunakan untuk *broadcast* protokol adalah fungsi `SendToRoom`.

```
public void
SendToRoom(List<Player>PlayerRoom,String
PlayerName,String Message,String Protocol)
    {
       for(Player p : PlayerRoom)
       {   if(Protocol.equals("CardEffect"))
         Protocol+="-"+PlayerName;
```

```
            pmessage.setMessage(Protocol+"-
"+Message);
            System.out.println(Message);
            p.connection.sendTCP(pmessage);
        }
    }
```

Kode Sumber 4.5 Fungsi `SendToRoom` pada Kelas `ServerApplication`

Fungsi `SendToRoom` pada Kode Sumber 4.5 berfungsi menerima parameter berupa daftar dari pemain pada *room*, nama pemain, pesan yang ingin dikirim, dan protokol yang digunakan.

4.2.2 Implementasi Manajemen Kartu

Pada bagian ini akan dijelaskan mengenai kelas-kelas yang berkaitan dengan implementasi kasus penggunaan manajemen kartu.

4.2.2.1 *Kelas DeckLoadManager*

Kelas `DeckLoadManager` berfungsi mengunduh data *deck* pemain dari *web service*. data tersebut diinstansiasi menjadi *game object* di dalam permainan melalui fungsi `LoadCardFromService`.

```
public void LoadCardFromService(string method,
GameObject grid)
        {
            List<string> list = new List<string>();
            Boolean _isEmpty = false;
            try
            {

                TextReader textReader = new
StreamReader(Application.persistentDataPath + "/" +
method + GameManager.Instance().PlayerId + ".xml");
                _xmlDoc.Load(textReader);
                _nameNodes =
 xmlDoc.GetElementsByTagName("Name");
```

```
                _quantityNodes =
_xmlDoc.GetElementsByTagName("Quantity");

                for (int i = 0; i <
_nameNodes.Count; i++)
                {
                    for (int j = 0; j <
int.Parse(_quantityNodes[i].InnerXml); j++)
                    {

list.Add(_nameNodes[i].InnerXml);
                    }
                }
            }
            catch
            {
                _isEmpty = true;
            }
            if(!_isEmpty) AddToGrid(grid, list);
        }
        public void ShowPlayerDP()
        {
            TextReader textReader = new
StreamReader(Application.persistentDataPath +
"/player_profile_" +
GameManager.Instance().PlayerId + ".xml");
            _xmlDoc.Load(textReader);
            _nameNodes =
_xmlDoc.GetElementsByTagName("MaxDP");

playerDeckPoint.GetComponent<UILabel>().text =
_nameNodes[0].InnerXml;
        }
    }
```

Kode Sumber 4.6 Fungsi `LoadCardFromService` pada Kelas `DeckLoadManager`

Fungsi `LoadCardFromService` pada Kode Sumber 4.6 berfungsi membaca *file* dengan format XML. *File* tersebut di baca sesuai dengan *tag* yang ada dan diinstansiasi.

4.2.2.2 *Kelas ConfirmDeck*

Ketika pemain selesai mengatur *deck*-nya. Pemain melakukan konfirmasi terhadap kartu yang dibawa. Apabila kartu yang dibawa *cost*-nya melebihi batasan *cost* yang dimiliki pemain maka konfirmasi tidak dapat dilanjutkan.

```
public void OnClick()
        {
            totalDeckCost = 0;
            cardList = new List<string>();
            cardQuantity = new List<int>();
            int DPCost =
int.Parse(deckPointCost.GetComponent<UILabel>().tex
t);
            int DPLeft =
int.Parse(playerDP.GetComponent<UILabel>().text);


            foreach (Transform t in grid.transform)
            {
                string s = t.name.Split('(')[0];
                bool is_distinguish = true;
                for(int i=0;i<cardList.Count;i++)
                {
                    if (cardList[i] == s)
                    {
                        is_distinguish = false;
                        cardQuantity[i]++;
                        totalDeckCost +=
t.GameObject.GetComponent<CardsEffect>().CardCost;
                        break;
                    }
                }
                if (is_distinguish)
                {
                    cardList.Add(s);
                    cardQuantity.Add(1);
                    totalDeckCost +=
t.GameObject.GetComponent<CardsEffect>().CardCost;
                }
```

```
            }

            if (DPCost <= DPLeft)
            {

WebServiceSingleton.GetInstance().ProcessRequest("c
lear_deck", id);

Debug.Log(WebServiceSingleton.GetInstance().respons
eFromServer);
                for (int i = 0; i < cardList.Count;
i++)
                {

WebServiceSingleton.GetInstance().ProcessRequest("i
nsert_to_deck", id + "|" + cardList[i] + "|" +
cardQuantity[i]);

Debug.Log(WebServiceSingleton.GetInstance().respons
eFromServer);
                }

                for (int i = 1; i <= 2; i++)
                {
                    try
                    {
                        string param = "";
                        if (i == 1) param = "deck";
                        else param = "trunk";
                        WebClient webClient = new
WebClient();

WebServiceSingleton.GetInstance().ProcessRequest("g
et_player_" + param, id);
                        Debug.Log("response : " +
WebServiceSingleton.GetInstance().responseFromServe
r);

Debug.Log(WebServiceSingleton.GetInstance().Downloa
dFile("get_player_" + param, id));
                    }
                    Catch
```

```
                        {
                            Debug.Log("Error connecting
to CWS server");
                        }
                    }

Application.LoadLevel("BeforeBattle");
                }
                else
                {
                    Debug.Log("Not Enough DP");
                }
            }
```

Kode Sumber 4.7 Fungsi Konfirmasi *Deck*

Fungsi konfirmasi *deck* pada Kode Sumber 4.7 secara *real time* melakukan pengecekan terhadap jumlah *cost deck* yang dimiliki pemain.

4.2.3 Implementasi Manajemen Pemain

Pada bagian ini akan dibahas mengenai kelas-kelas yang berkaitan dengan kasus penggunaan manajemen pemain.

4.2.3.1 *Kelas PartyEditorManager*

Kelas `PartyEditorManager` merupakan kelas yang berfungsi menampilkan daftar *avatar* milik pemain lain melalui fungsi `GetAvatarList`.

```
public void GetAvatarList(string id)
        {
            friendList = new List<string>();
            xmlFromServer = new XmlDocument();
            TextReader textReader = new
StreamReader(Application.persistentDataPath +
"/friends_of_" + GameManager.Instance().PlayerId +
".xml");
            xmlFromServer.Load(textReader);
            attributeNodes =
xmlFromServer.GetElementsByTagName("Name");
```

```
            for (int i = 0; i <
attributeNodes.Count; i++)
            {

friendList.Add(attributeNodes[i].InnerXml);
            }
        }
```

Kode Sumber 4.8 Fungsi `GetAvatarList` pada Kelas `PartyEditor`

Fungsi `GetAvatarList` pada Kode Sumber 4.8 membaca dokumen XML yang berasal dari *web service*. Setiap data *avatar* diinstansiasi menjadi *list* sehingga dapat ditampilkan pada layar permainan. Kemudian dilakukan proses konfirmasi melalui fungsi `Confirmparty` pada Kode Sumber 4.9.

```
private void ConfirmParty()
        {
            GameManager.Instance().PartyId = new
List<string>();
            foreach (Transform T in grid.transform)
            {
                if
(T.GetComponent<Avatar>().PlayerName !=
GameManager.Instance().PlayerId)
                {

GameManager.Instance().PartyId.Add(T.GetComponent<A
vatar>().PlayerName);
                }
            }
            foreach (string s in
GameManager.Instance().PartyId)
            {
WebServiceSingleton.GetInstance().ProcessRequest("g
et profile", s);
```

```
Debug.Log(WebServiceSingleton.GetInstance().Downloa
dFile("get_profile", s));

WebServiceSingleton.GetInstance().ProcessRequest("g
et_player_deck", s);

Debug.Log(WebServiceSingleton.GetInstance().Downloa
dFile("get_player_deck", s));
                }

                }
            }
            Application.LoadLevel("BeforeBattle")
```

Kode Sumber 4.9 Fungsi `ConfirmParty`

Ketika pemain telah yakin dengan formasi pemain. Pemain akan menekan tombol *confirm*. Pemain lain secara otomatis akan ditambahkan ke dalam *party list* untuk diinstansiasi pada pertarungan.

4.2.4 Implementasi Melihat Laporan Pertarungan

Pada bagian ini dijelaskan mengenai kelas-kelas yang berkaitan dengan kasus penggunaan melihat laporan pertarungan.

4.2.4.1 *Kelas PVPRecordManager*

Kelas `PVPRecordManager` berfungsi membaca laporan pertarungan pemain yang terdapat pada *file* XML melalui fungsi `GetOpponentsName` dan `GenerateRooster` dapat dilihat pada Kode Sumber 4.10 dan Kode Sumber 4.11.

```
public void GetOpponentsName()
        {
            try
            {
                XmlSerializer deserializer = new
XmlSerializer(typeof(BattleRankFromService));
```

```
                TextReader textReader = new
StreamReader(Application.persistentDataPath +
"/battle_rank_of_" +
GameManager.Instance().PlayerId + ".xml");
                object obj =
deserializer.Deserialize(textReader);
                var  Opponents =
(BattleRankFromService)obj;
                textReader.Close();
                current.items= new List<string>();
                opponentList= new
List<EnemyPlayerFromService>();
                foreach (var op in
Opponents.enemyPlayer)
                {
                    current.items.Add(op.Name);
                    Debug.Log(op.Name);
                    opponentList.Add(op);
                }
            }
        }
```

Kode Sumber 4.10 Kode Sumber Fungsi `GetOpponentsName`

Fungsi `GetOpponentsname` berfungsi untuk mendapatkan nama-nama dari pemain lain yang sudah dilawan oleh pemain, Fungsi `GetOpponentRooster` berfungsi untuk mendapatkan jumlah menang dan kalah saat pemain melawan pemain.

```
public void GenerateRooster()
        {
            try
            {
                XmlSerializer deserializer = new
XmlSerializer(typeof(PlayerRankingFromService));
                TextReader textReader = new
StreamReader(Application.persistentDataPath +
"/player_rank.xml");
                object obj =
deserializer.Deserialize(textReader);
```

```
                var Opponents =
(PlayerRankingFromService)obj;
                textReader.Close();
                current.items = new List<string>();
                opponentList = new
List<EnemyPlayerFromService>();
                foreach (var op in
Opponents.playerDetail)
                {
                    label.GetComponent<UILabel>().text
= op.Name + " Battle Won =" + op.BattleWon;
                    NGUITools.AddChild(grid, label);
                }
            }
            catch (Exception e)
            {
                Debug.Log(e);
            }

        }
```

Kode Sumber 4.11 Fungsi `GetOppnentRooster`

4.3 Implementasi Penanganan Kegagalan Sinkronisasi

Pada bagian ini akan membahas mengenai implementasi penaganan kegagalan sinkronisasi pada modul pertarungan pemain lawan pemain.

4.3.1 Implementasi Kegagalan Koneksi

Untuk mengatasi terjadinya kegagalan koneksi terhadap *server* maka diimplementasikan sebuah fungsi yang berfungsi untuk mengetahui apakah koneksi mengalami *timeout.*

```
private void CheckUpdate()
        {

            if
(NetworkSingleton.Instance().ServerMessage != null)
            {
                if
(NetworkSingleton.Instance().ServerMessage.Contains
("Connected-to-server"))
                {
                    StopAllCoroutines();

Application.LoadLevel("LobbyRoom");

NetworkSingleton.Instance().ServerMessage = "";

                }
            }
            if (flag2 == true)
            {
                t += Time.deltaTime;
            }
            if (t > 15)
            {
                if (flag == false)
                {
                    var obj = new object[2];
                    obj[0] = "Time is up";
                    obj[1] = "connection failed
cannot contact server";

msgBox.SendMessage("SetMessage", obj);

msgBox.SendMessage("ShowMessageBox");
                    StopAllCoroutines();
```

```
                    t = 0.0f;
                }
                flag = true;
                flag2 = false;
                button1.SetActive(true);
                button2.SetActive(true);
                //input.SetActive(true);
                loading.SetActive(false);
            }
            //Debug.Log(t);
        }
```

Kode Sumber 4.12 Fungsi `CheckUpdate`

Fungsi `CheckUpdate` pada Kode Sumber 4.12 berfungsi untuk melakukan pengecekan terhadap *request* yang diberikan kepada *server*. Apabila waktu kembalian melebihi waktu yang ditentukan yaitu lima belas detik, maka pesan akan dianggap *timeout* dan koneksi gagal dilakukan.

4.3.2 Implementasi Penanganan Pemain Terputus dari *Server*

Untuk mengatasi terjadinya pemain yang terputus dari *server* akan diimplementasikan, sebuah fungsi yang berfungsi untuk membaca pesan dari *server* bahwa klien terputus dari *server*.

```
public void CheckWinorLose()
        {
            if
(GameManager.Instance().Enemies.Count <= 0)
            {
                this.currentstate= new
WinState(this);
                this.currentstate.Action();
            }
            else if
(GameManager.Instance().Players.Count <= 0)
            {
```

```
                this.currentstate= new
LoseState(this);
                this.currentstate.Action();
            }
            if (GameManager.Instance().GameMode ==
"pvp")
            {
                string serverMessage =
NetworkSingleton.Instance().ServerMessage;
                Debug.Log(serverMessage);
                var message =
serverMessage.Split('-');
                if
(!serverMessage.Contains("Disconnected")) return;
                if
(GameManager.Instance().PlayerId.Equals(message[1])
)
                {
                   this.currentstate= new
LoseState(this);
                   this.currentstate.Action();
                }
                else
                {
                    this.currentstate= new
WinState(this);
                    this.currentstate.Action();

NetworkSingleton.Instance().ServerMessage = "";
                }
            }
        }
```

Kode Sumber 4.13 Fungsi CheckWinorLose

Fungsi `CheckWinorLose` pada Kode Sumber 4.13 berfungsi untuk membaca pesan dari *server* apabila pada klien terputus dari *server*, *server* akan melakukan *broadcast* pesan bahwa klien terputus dari *server*. Apabila klien tersebut terputus pada saat permainan berlangsung maka pemain akan dianggap kalah.

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada modul yang dikembangkan. Pengujian yang dilakukan adalah secara *white box* dan *black box* yaitu pengujian fungsionalitas. Pengujian dilakukan dengan kombinasi pengujian manual terhadap modul yang dikembangkan dengan *unit testing.* Pengujian dilakukan untuk mengetahui apakah modul sudah sesuai dengan kasus penggunaan pada bab 3.

5.1 Lingkungan Pengujian

Lingkungan pengujian sistem pada pengerjaan Tugas Akhir ini dilakukan pada lingkungan dan alat kakas sebagai berikut.

Jenis Perangkat :*Handphone* Andromax Z
Prosesor :Quad-core 1.5 GHz MT6589E.
Sistem Operasi :Android Jelly Bean 4.2.1.
Memori :1.00 GB.

Jenis Perangkat :*Handphone* Nexus 4.
Prosesor :Qualcomm APQ8064 SnapdragonQuad-core 1.5 GHz.
Sistem Operasi :Android Kitkat 4.3.1.
Memori :2.00 GB.

5.2 Skenario Pengujian

Pada bagian ini akan dijelaskan tentang skenario pengujian yang dilakukan. Pengujian yang dilakukan adalah pengujian fungsionalitas dan *white box*.

5.2.1 Pengujian Fungsionalitas

Pengujian fungsionalitas sistem dilakukan dengan menyiapkan sejumlah skenario sebagai tolok ukur keberhasilan pengujian. Pengujian fungsionalitas dilakukan dengan mengacu pada kasus penggunaan yang telah dijelaskan pada subbab 3.1.5. Pengujian kebutuhan fungsionalitas akan dijabarkan pada subbab berikut.

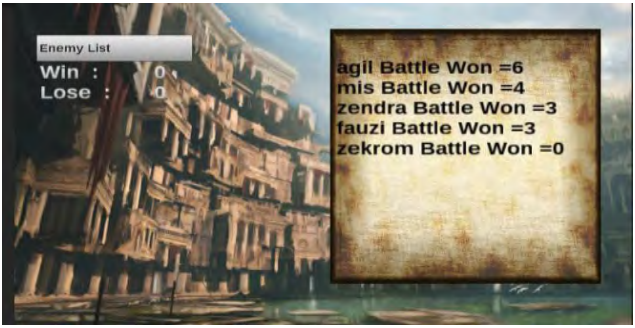
5.2.1.1 *Pengujian Fitur Menampilkan Laporan Pertarungan*

Pengujian fitur menampilkan laporan pertarungan bertujuan untuk menguji apakah aplikasi dapat menampilkan laporan pertarungan pemain di arena sesuai dengan pertarungan yang dilaluinya. Skenario pengujian adalah dengan cara pemain memilih nama lawan yang pernah dilawan pada pertarungan untuk mengetahui jumlah kemenangan dan kekalahan.

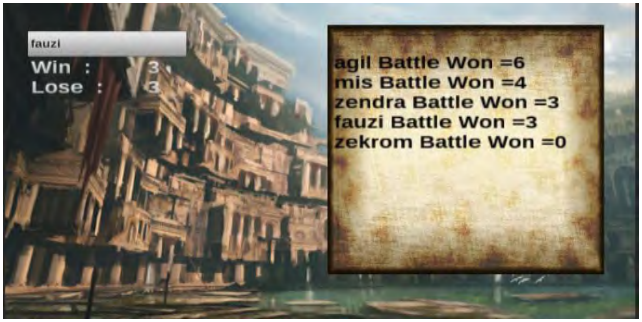
Tabel 5.1 Pengujian Menampilkan Laporan Pertarungan

ID	UJ.UC-0001
Referensi Kasus Penggunaan	UC-0001
Nama	Pengujian fitur melihat laporan pertarungan.
Tujuan Pengujian	Menguji apakah pemain dapat bermain dengan *player* lain sesuai dengan alur permainan.
Skenario 1	Pengguna menekan menu *host* pada layar ruang *lobby*.
Kondisi Awal	Pengguna sudah ada pada layar ruang *lobby*.
Masukan	Sentuhan layar

Hasil Yang Diharapkan	Pemain dapat melihat laporan pertarungan.
Hasil Yang Didapat	Sistem menampilkan jumlah menang dan kalah pemain.
Hasil Pengujian	Berhasil
Kondisi Akhir	Pemain dapat melihat laporan pertarungan. Hasil akhir pengujian dapat dilihat pada Gambar 5.2.



Gambar 5.1 Kondisi Awal Sebelum Pemain Memilih Pemain



Gambar 5.2 Kondisi Setelah Pemain Memilih Pemain

Detail dari skenario dapat dilihat pada Tabel 5.1 sedangkan hasil uji dapat dilihat pada Gambar 5.1 dan Gambar 5.2. Berdasarkan pengujian aplikasi dapat menampilkan jumlah menang dan kalah pemain. Hal ini membuktikan bahwa pengujian berhasil dilakukan.

5.2.1.2 *Pengujian Fitur Pemain Lawan Pemain*

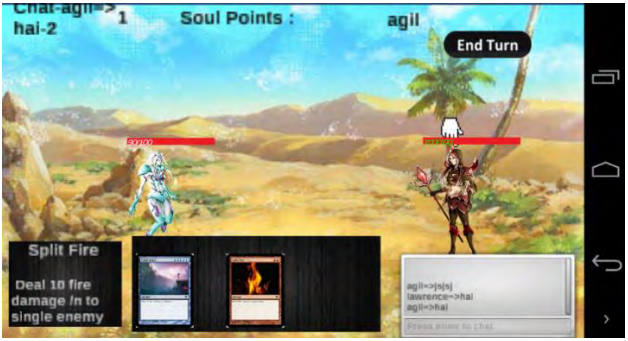
Pengujian fitur pemain lawan pemain dilakukan dengan dua skenario. Skenario pertama adalah kondisi pemain sebagai *host*. Skenario kedua adalah kondisi pemain sebagai *join player* pemain pengujian dilakukan untuk mengetahui apakah alur permainan sudah berjalan sesuai yang diharapkan.

Tabel 5.2 Pengujian Pertarungan Pemain Lawan Pemain

ID	UJ.UC-0002
Referensi Kasus Penggunaan	UC-0002
Nama	Pengujian fitur pemain lawan pemain
Tujuan Pengujian	Menguji apakah pemain dapat bermain dengan *player* lain sesuai dengan alur permainan.
Skenario 1	Pengguna menekan menu *host* pada layar ruang *lobby*.
Kondisi Awal	Pengguna sudah ada pada layar ruang *lobby*.
Masukan	Sentuhan layar
Hasil Yang Diharapkan	Pemain dapat melaksanakan permainan sesuai alur permainan.

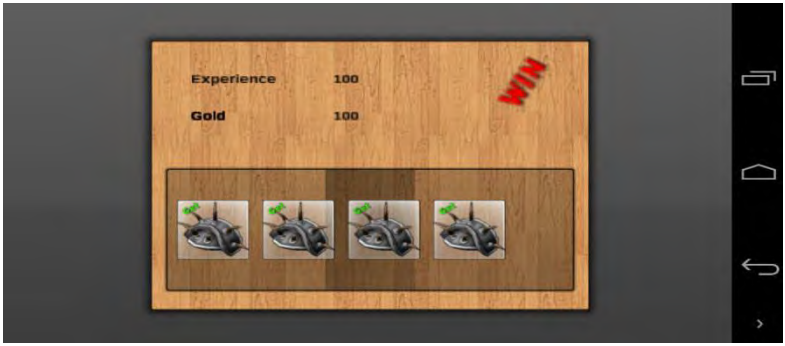
Hasil Yang Didapat	Pemain berhasil menang atau kalah.
Hasil Pengujian	Berhasil
Kondisi Akhir	Pemain dapat terhubung dengan *server*. Hasil akhir pengujian dapat dilihat pada

Gambar 5.3 Kondisi Awal Pada Saat Pemain Memilih Mode *Host*



Gambar 5.4 Kondisi Pertarungan Antar Pemain

Detail dari skenario dapat dilihat pada Tabel 5.2 sedangkan hasil uji dapat dilihat pada Gambar 5.3 Berdasarkan pengujian dapat dilihat bahwa pemain yang bertindak sebagai *host* dapat terhubung dengan pemain lain. Hanya pemain yang bertindak sebagai *host* yang dapat memulai permainan dapat dilihat pada Gambar 5.4. Pemain dapat menjalankan permainan sebagaimana mestinya terlihat pada Gambar 5.5. Hal ini membuktikan bahwa pengujian berhasil dilakukan.

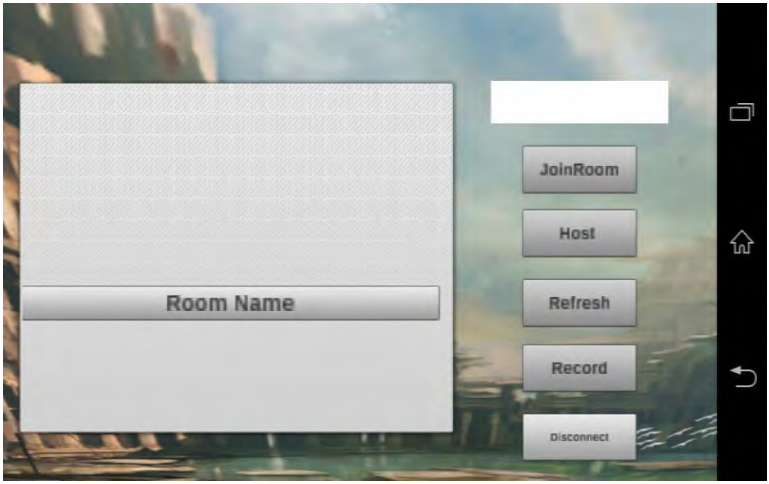


Gambar 5.5 Hasil Akhir Pengujian Pemain Lawan Pemain Skenario 1

Tabel 5.3 Pengujian Fitur Pertarungan Pemain Lawan Pemain Skenario kedua

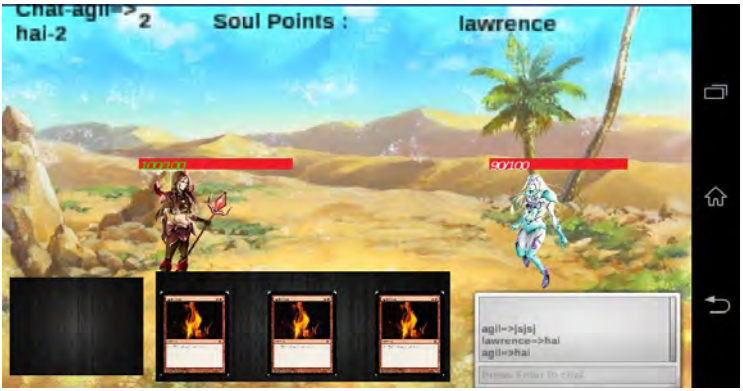
ID	UJ.UC-0002
Referensi Kasus Penggunaan	UC-0002
Nama	Pengujian fitur pemain lawan pemain
Tujuan Pengujian	Menguji apakah pemain dapat bermain dengan *player* lain sesuai dengan alur permainan.
Skenario 2	Pengguna menekan menu *join* pada layar *lobby*

Kondisi Awal	Pengguna sudah ada pada layar ruang *lobby*
Masukan	Sentuhan layar
Hasil Yang Diharapkan	Pemain dapat melaksanakan permainan sesuai alur permainan
Hasil Yang Didapat	Pemain berhasil menang atau kalah.
Hasil Pengujian	Berhasil
Kondisi Akhir	Pemain dapat terhubung dengan *server*. Hasil akhir dapat dilihat pada Gambar 5.8



Gambar 5.6 Kondisi Awal Pemain lawan Pemain Memilih Mode *Join*

Gambar 5.7 Kondisi Ruang Tunggu Mode *Join*



Gambar 5.8 Kondisi Pertarungan Mode *Join*

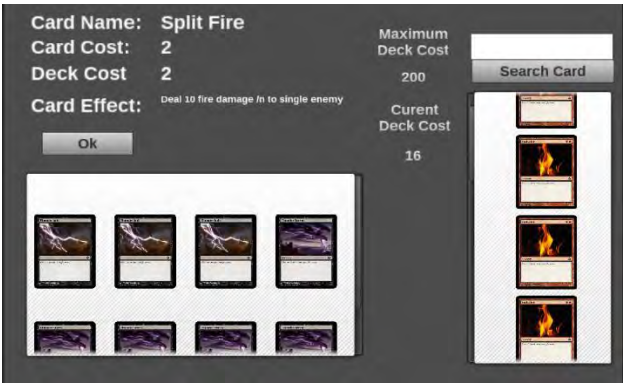
Detail dari skenario dapat dilihat pada Tabel 5.2 sedangkan hasil uji dapat dilihat pada Gambar 5.7 dan Gambar 5.8. Berdasarkan pengujian dapat dilihat bahwa pemain yang bertindak sebagai *join client* dapat terhubung dengan pemain lain dan hanya pemain yang bertindak sebagai *host* yang dapat memulai permainan dapat dilihat pada Gambar 5.7. Pemain dapat menjalankan permainan seperti yang terlihat pada Gambar 5.8. Hal ini membuktikan bahwa pengujian berhasil dilakukan.

5.2.1.3 *Pengujian Fitur Manajemen Kartu*

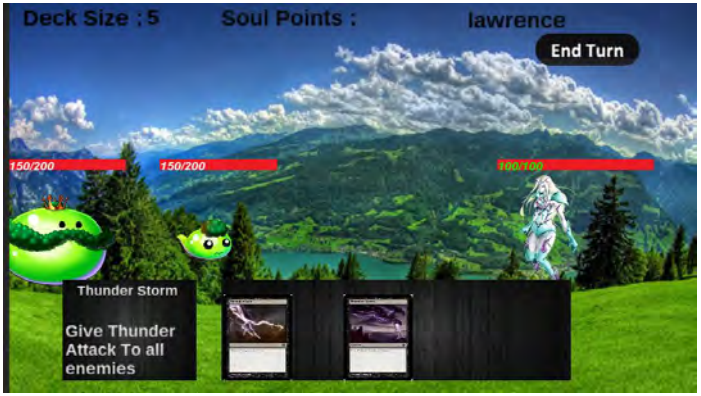
Pengujian fitur manajemen kartu berfungsi untuk mengetahui apakah kartu dapat dipindahkan dari dalam tempat penyimpanan ke dalam *deck*.

Tabel 5.4 Pengujian fitur manajemen kartu

ID	UJ.UC-0003
Referensi Kasus Penggunaan	UC-0003
Nama	Pengujian fitur manajemen kartu.
Tujuan Pengujian	Menguji apakah kartu dapat dipindahkan dari dalam tempat penyimpan ke dalam *deck*.
Skenario	Pengguna menekan tombol *edit deck* pada layar.
Kondisi Awal	Pengguna sudah ada pada layar utama.
Masukan	Sentuhan layar.
Hasil Yang Diharapkan	Pengguna dapat menyimpan data kartu dari dalam *trunk* ke dalam *deck* dan diinstansiasi pada saat pertarungan.
Hasil Yang Didapat	Kartu berhasil diinstansiasi
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pemain dapat mengganti isi *deck*. Hasil akhir pengujian dapat dilihat pada Gambar 5.10.



Gambar 5.9 Kondisi Manajemen Awal Manajemen Kartu



Gambar 5.10 Kondisi Kartu Terinstansiasi

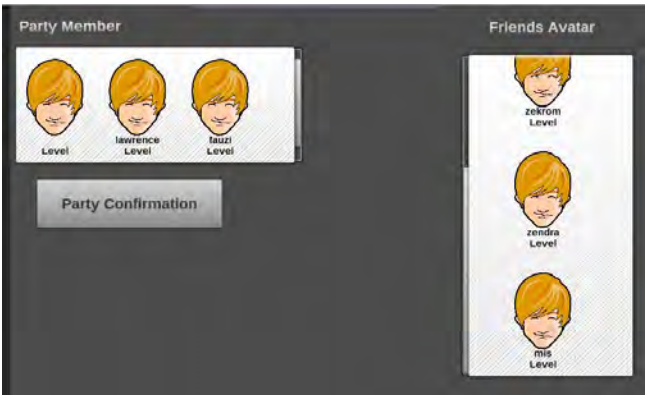
Detail dari skenario dapat dilihat pada Tabel 5.4 sedangkan hasil uji dapat dilihat pada Gambar 5.9 dan Gambar 5.10. Berdasarkan pengujian aplikasi menginstansiasi kartu yang terdapat pada panel *deck* yaitu pada Gambar 5.9 panel sebelah kiri. Hal ini membuktikan bahwa pengujian berhasil dilakukan.

5.2.1.4 *Pengujian Fitur Manajemen Pemain*

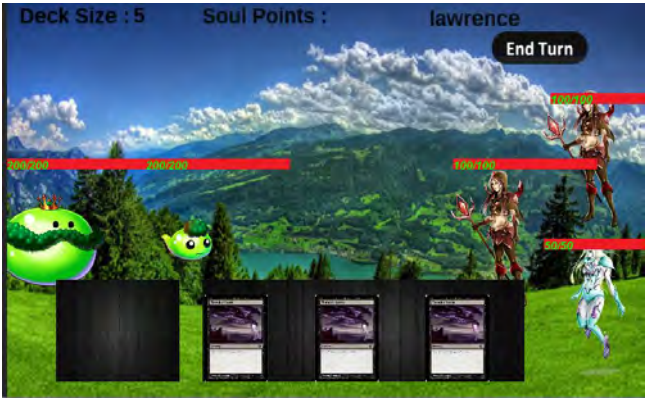
Pengujian fitur manajemen pemain berfungsi untuk mengetahui apakah *avatar* milik pemain lain diinstansisasi pada pertarungan.

Tabel 5.5 Pengujian Fitur Manajemen Pemain

ID	UJ.UC-0004
Referensi Kasus Penggunaan	UC-0004
Nama	Pengujian fitur manajemen pemain
Tujuan Pengujian	Menguji *avatar* pemain milik teman dapat diinstansiasi pada pertarungan.
Skenario	Pengguna menekan tombol *edit party* pada layar
Kondisi Awal	Pengguna sudah ada pada layar utama
Masukan	Sentuhan layar
Hasil Yang Diharapkan	Pengguna dapat memilih *avatar* milik teman dan diinstansiasi pada permainan.
Hasil Yang Didapat	*Avatar* milik pemain lawan berhasil diinstansias pada pertarungan.
Hasil Pengujian	Berhasil
Kondisi Akhir	Pemain dapat menginstansiasi *avatar* pemain lain . Hasil akhir pengujian dapat dilihat pada Gambar 5.12.



Gambar 5.11 Memindahkan *Avatar* Milik Pemain Lain Ke Dalam *Party Member*



Gambar 5.12 *Avatar* Milik Pemain Lain Berhasil Diinstansiasi Pada Pertarungan

Detail dari skenario dapat dilihat pada Tabel 5.5, sedangkan hasil uji dapat dilihat pada Gambar 5.11 dan Gambar 5.12. Berdasarkan pengujian aplikasi dapat menginstansiasi *avatar* pemain lain yang terdapat pada *party member*. Hal ini membuktikan bahwa pengujian berhasil dilakukan.

5.2.2 Pengujian *White Box*

Pengujian *white box* dilakukan dengan cara melakukan pengujian terhadap fungsi-fungsi yang ada pada modul pertarungan dengan menggunakan Unity Test Tool. Pengujian dilakukan dengan cara membandingkan hasil keluaran fungsi apakah sesuai dengan hasil yang diharapkan.

5.2.2.1 *Unit Testing*

Unit testing adalah pengujian yang dilakukan dengan membandingkan kesesuaian hasil keluaran fungsi-fungsi pada permainan dengan hasil yang ingin dicapai.

Tabel 5.6 Daftar Fungsi Uji *Unit Testing*

No.	Nama Fungsi Unit *Testing*	Kegunaan	Hasil yang diharapkan	Status Pengujian
1	*Test Card Effect Protocol.*	Melakukan pengujian terhadap protokol yang mengirimkan efek kartu.	Efek kartu dapat dijalankan dengan sukses.	Berhasil
2	*Test End Phase Protocol.*	Melakukan pengujian terhadap protokol yang mengirimkan tanda akhiri giliran.	Dapat berganti *state* dengan sukses.	Berhasil
3	*Damage Receiver Thunder Test.*	Melakukan pengujian terhadap efek kartu petir terap objek.	*Damage* yang diterima berjumlah dua kali lipat dan setengah dari *damage* biasa.	Berhasil

No.	Nama Fungsi Unit *Testing*	Kegunaan	Hasil yang diharapkan	Status Pengujian
4	*Damage Receiver Water Test.*	Melakukan pengujian terhadap efek kartu air terhadap objek.	*Damage* yang diterima berjumlah dua kali lipat dan setengah dari *damage* biasa.	Berhasil
5	*Damage Receiver Earth Test.*	Melakukan pengujian terhadap efek kartu tanah terhadap objek.	*Damage* yang diterima berjumlah dua kali lipat dan setengah dari *damage* biasa.	Berhasil
7	*Damage Receiver Fire Test.*	Melakukan pengujian terhadap efek kartu api terhadap objek.	*Damage* yang diterima berjumlah dua kali lipat dan setengah dari *damage* biasa	Berhasil
8	*Damage Receiver Wind Test.*	Melakukan pengujian terhadap efek kartu angin terhadap objek.	*Damage* yang diterima berjumlah dua kali lipat dan setengah dari *damage* biasa.	Berhasil

Gambar 5.13 Hasil *Unit Testing* Fungsi-Fungsi pada Modul Pertarungan

Gambar 5.13 merupakan hasil dari *unit testing* fungsi-fungsi utama pada modul pertarungan. Rincian kode pengujian dapat dilihat pada lampiran Kode Sumber A. 16.

5.2.3 Pengujian Protokol

Pengujian protokol dilakukan untuk mengetahui ukuran penerimaan pesan balasan protokol dari *server* ke *client* dalam hitungan *millisecond*. Pemanggilan protokol dilakukan pada *server* yang berada pada jaringan lokal. Daftar protokol dan hasil uji dapat dilihat pada Tabel 5.7.

Tabel 5.7 Daftar Pengujian Protokol

No	Protokol	Balasan	Hasil	Waktu
1	GetRoomList	Data *room*	Berhasil	5 ms
2	JoinRoom	Pesan "JoinedRoom"	Berhasil	6 ms
3	GetPlayerList	Data pemain yang terhubung	Berhasil	7 ms
4	CardEffect	Nama kartu	Berhasil	6 ms

No	Protokol	Balasan	Hasil	Waktu
5	EndTurn	Pesan "EndTurn"	Berhasil	8 ms
6	Chat	Pesan "Chat"	Berhasil	8 ms
7	Disconnect	Pesan "Disconnected"	Berhasil	7 ms
8	Confirmation	Pesan "Confirmed"	Berhasil	9 ms
9	StartGame	Pesan "GameStart"	Berhasil	10 ms
10	CreateRoom	Pesan "RoomCreatedSuccess"	Berhasil	60 ms

5.2.4 *Stress Testing*

Stress testing bertujuan untuk mengetahui ketahanan *server* jika terhubung dengan jumlah pengguna yang ditentukan. Untuk mengetahui ketahanan *server* maka dilakukan pengujian menggunakan TCP *sampler* milik JMeter. Pengujian dilakukan terhadap *server* yang berada pada jaringan lokal dengan asumsi terhubung dengan banyak pengguna.

Skenario pengujian dilakukan dengan menggunakan skenario yang dirundingkan bersama dengan *stakeholder* aplikasi Card Warlock Saga yaitu *system administrator* dari perusahaan Square-Enix. Dua buah komputer yang menjalankan Apache JMeter pada *port* 9999 dan IP *server* 10.151.43.150 secara bersama-sama. Dengan IP masing-masing komputer adalah 10.151.63.151 dan 10.151.63.6. Pengujian dilakukan dengan menggunakan Apache JMeter untuk melihat batasan pengguna. Pengujian dilakukan dengan kelipatan 1000 mulai dari 1000 sampai dengan 10000 pengguna untuk mengetahui reaksi *server* ketika menerima jumlah klien yang besar dengan waktu *timeout* sebesar 5000 ms.

Tabel 5.8 *Strees Testing* Tahap 1

ID	UJISTRESS.UC-0001
Nama	Pengujian *stress testing* dengan 1000 pengguna
Tujuan Pengujian	Menguji apakah jumlah pengguna yang banyak pada saat mengakses *server*, menyebabkan *timeout* atau *deadlock*.

Skenario	Penguji menjalankan Apache JMeter dengan menjalankan 1000 *thread* menggunakan dua buah komputer.
Kondisi Awal	Penguji terhubung dengan jaringan.
Masukan	-
Hasil Yang Diharapkan	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Yang Didapat	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Thread* Ke-1000 dapat terhubung dengan sukses.

Tabel 5.9 *Strees Testing* Tahap 2

ID	UJISTRESS.UC-0002
Nama	Pengujian *stress testing* dengan 2000 pengguna
Tujuan Pengujian	Menguji apakah jumlah pengguna yang banyak pada saat mengakses *server*, menyebabkan *timeout* atau *deadlock*.
Skenario	Penguji menjalankan Apache JMeter dengan menjalankan 2000 *thread* menggunakan dua buah komputer.
Kondisi Awal	Penguji terhubung dengan jaringan.
Masukan	-
Hasil Yang Diharapkan	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Yang Didapat	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Pengujian	Berhasil

Kondisi Akhir	*Thread* Ke-2000 dapat terkoneksi dengan sukses.

Tabel 5.10 *Strees Testing* Tahap 3

ID	UJISTRESS.UC-0003
Nama	Pengujian *stress testing* dengan 3000 pengguna
Tujuan Pengujian	Menguji apakah jumlah pengguna yang banyak pada saat mengakses *server*, menyebabkan *timeout* atau *deadlock*.
Skenario	Penguji menjalankan Apache JMeter dengan menjalankan 3000 *thread* menggunakan dua buah komputer.
Kondisi Awal	Penguji terhubung dengan jaringan.
Masukan	-
Hasil Yang Diharapkan	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Yang Didapat	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Thread* Ke-3000 dapat terhubung dengan sukses.

Tabel 5.11 *Strees Testing* Tahap 4

ID	UJISTRESS.UC-0004
Nama	Pengujian *stress* testing dengan 4000 pengguna
Tujuan Pengujian	Menguji apakah jumlah pengguna yang banyak pada saat mengakses *server*, menyebabkan *timeout* atau *deadlock*.
Skenario	Penguji menjalankan Apache JMeter dengan menjalankan 4000 *thread* menggunakan dua buah komputer.

Kondisi Awal	Penguji terhubung dengan jaringan.
Masukan	-
Hasil Yang Diharapkan	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Yang Didapat	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Thread* Ke-4000 dapat terhubung dengan sukses.

Tabel 5.12 *Strees Testing* Tahap 5

ID	UJISTRESS.UC-0005
Nama	Pengujian *stress testing* dengan 5000 pengguna
Tujuan Pengujian	Menguji apakah jumlah pengguna yang banyak pada saat mengakses *server*, menyebabkan *timeout* atau *deadlock*.
Skenario	Penguji menjalankan Apache JMeter dengan menjalankan 5000 *thread* menggunakan dua buah komputer.
Kondisi Awal	Penguji terhubung dengan jaringan.
Masukan	-
Hasil Yang Diharapkan	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Yang Didapat	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Thread* Ke-5000 dapat terhubung dengan sukses.

Tabel 5.13 *Strees Testing* Tahap 6

ID	UJISTRESS.UC-0006
Nama	Pengujian *stress testing* dengan 6000 pengguna
Tujuan Pengujian	Menguji apakah jumlah pengguna yang banyak pada saat mengakses *server*, menyebabkan *timeout* atau *deadlock*.
Skenario	Penguji menjalankan Apache JMeter dengan menjalankan 6000 *thread* menggunakan dua buah komputer.
Kondisi Awal	Penguji terhubung dengan jaringan.
Masukan	-
Hasil Yang Diharapkan	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Yang Didapat	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Thread* Ke-6000 dapat terhubung dengan sukses.

Tabel 5.14 *Strees Testing* Tahap 7

ID	UJISTRESS.UC-0007
Nama	Pengujian *stress testing* dengan 7000 pengguna
Tujuan Pengujian	Menguji apakah jumlah pengguna yang banyak pada saat mengakses *server*, menyebabkan *timeout* atau *deadlock*.
Skenario	Penguji menjalankan Apache JMeter dengan menjalankan 7000 *thread* menggunakan dua buah komputer.

Kondisi Awal	Penguji terhubung dengan jaringan.
Masukan	-
Hasil Yang Diharapkan	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Yang Didapat	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Thread* Ke-7000 dapat terhubung dengan sukses.

Tabel 5.15 *Strees Testing* Tahap 8

ID	UJISTRESS.UC-0008
Nama	Pengujian *stress testing* dengan 8000 pengguna
Tujuan Pengujian	Menguji apakah jumlah pengguna yang banyak pada saat mengakses *server*, menyebabkan *timeout* atau *deadlock*.
Skenario	Penguji menjalankan Apache JMeter dengan menjalankan 8000 *thread* menggunakan dua buah komputer.
Kondisi Awal	Penguji terhubung dengan jaringan.
Masukan	-
Hasil Yang Diharapkan	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Yang Didapat	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Thread* Ke-8000 dapat terhubung dengan sukses.

Tabel 5.16 *Strees Testing* Tahap 9

ID	UJISTRESS.UC-0009
Nama	Pengujian *stress testing* dengan 9000 pengguna
Tujuan Pengujian	Menguji apakah jumlah pengguna yang banyak pada saat mengakses *server*, menyebabkan *timeout* atau *deadlock*.
Skenario	Penguji menjalankan Apache JMeter dengan menjalankan 9000 *thread* menggunakan dua buah komputer.
Kondisi Awal	Penguji terhubung dengan jaringan.
Masukan	-
Hasil Yang Diharapkan	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Yang Didapat	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Thread* Ke-9000 dapat terhubung dengan sukses.

Tabel 5.17 *Strees Testing* Tahap 10

ID	UJISTRESS.UC-0010
Nama	Pengujian *stress testing* dengan 10000 pengguna
Tujuan Pengujian	Menguji apakah jumlah pengguna yang banyak pada saat mengakses *server*, menyebabkan *timeout* atau *deadlock*.
Skenario	Penguji menjalankan Apache JMeter dengan menjalankan 10000 *thread* menggunakan dua buah komputer.
Kondisi Awal	Penguji terhubung dengan jaringan.

Masukan	-
Hasil Yang Diharapkan	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Yang Didapat	Tidak ada *thread* yang *deadlock*, ataupun *timeout* pada saat pengujian.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Thread* Ke-10000 dapat terhubung dengan sukses.

Tabel 5.8 sampai dengan Tabel 5.17 merupakan spesifikasi dari skenario pengujian *stress testing*. Dari hasil pengujian pada *host* pertama, dapat dilihat pada lampiran C Gambar C. 1 sampai dengan Gambar C. 10 bahwa pengujian dapat dilakukan dengan sukses tanpa adanya masalah. Sedangkan pada pengujian pada *host* kedua, dapat dilihat pada lampiran C Gambar C. 11 sampai dengan Gambar C. 20 juga tidak mengalami masalah.

5.2.5 Pengujian Integrasi dengan Modul Lain

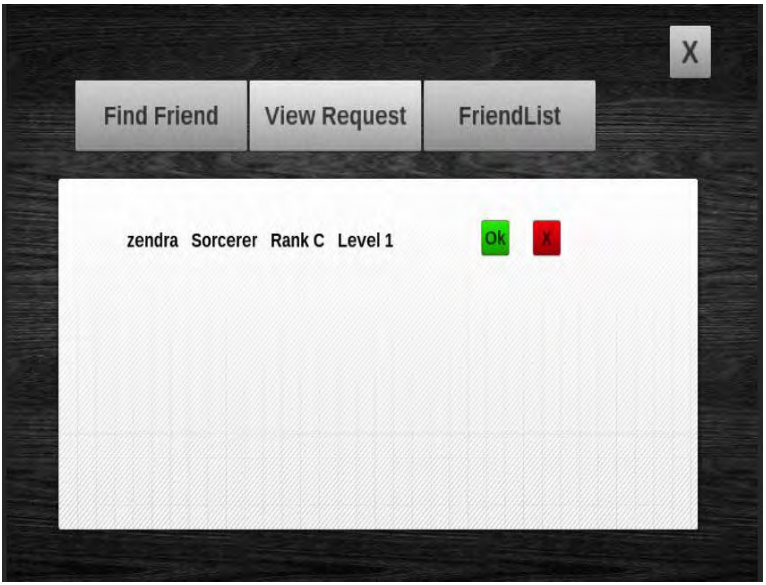
Pengujian integrasi dengan modul lain dilakukan dengan menyiapkan skenario sebagai tolok ukur keberhasilan pengujian dan setiap hasil dari skenario akan dilihat dari modul lain untuk melihat apakah integrasi antar modul telah tereksekusi dengan benar.

5.2.5.1 *Pengujian Integrasi dengan Modul Sosial*

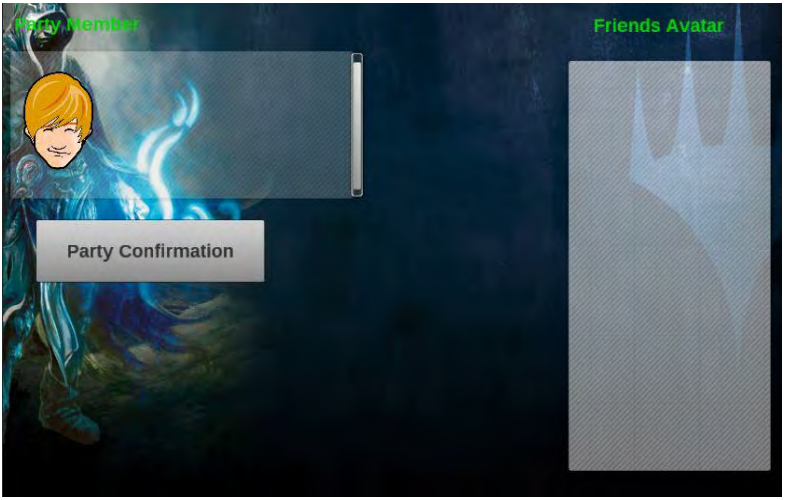
Pengujian integrasi dengan modul sosial dilakukan untuk menguji apakah kegiatan yang dilakukan pada modul sosial mempengaruhi modul sosial. Yang diuji adalah *database* yang terdapat pada sosial dan yang diuji pada modul sosial adalah fitur mengirim permintaan pertemanan. Detail skenario pengujian dapat dilihat pada Tabel 5.18.

Tabel 5.18 Pengujian Integrasi dengan Modul Sosial

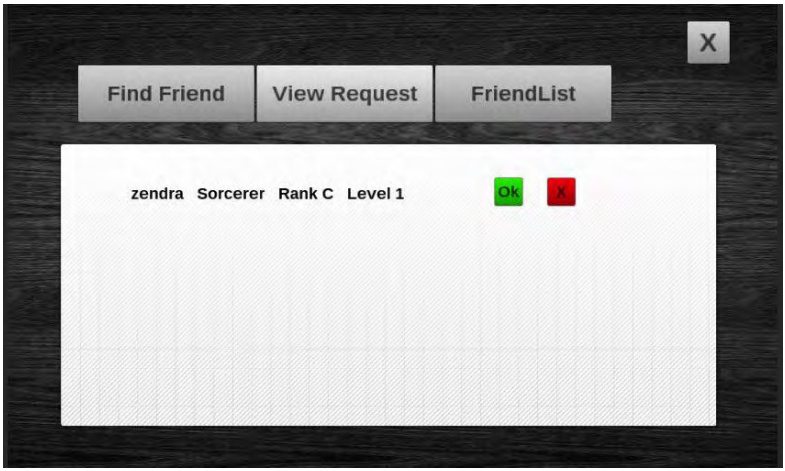
ID	UJI.UC-0001
Nama	Pengujian integrasi dengan modul pertarungan
Tujuan Pengujian	Menguji apakah salah satu fitur pada modul sosial yaitu fitur menerima permintaan pertemanan mempengaruhi daftar teman pada fitur *party editor* di modul pertarungan.
Skenario	Pengguna menerima permintaan pertemanan.
Kondisi Awal	Pengguna sudah mendapat permintaan pertemanan dari pengguna lain.
Masukan	Sentuhan layar
Hasil Yang Diharapkan	Daftar teman pada fitur *party editor* di modul pertarungan terintegrasi dengan daftar teman pada modul sosial.
Hasil Yang Didapat	Daftar teman pada fitur *party editor* di modul pertarungan terintegrasi dengan daftar teman pada modul sosial.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Database* pada fitur *party editor* modul pertarungan ter-*update*.



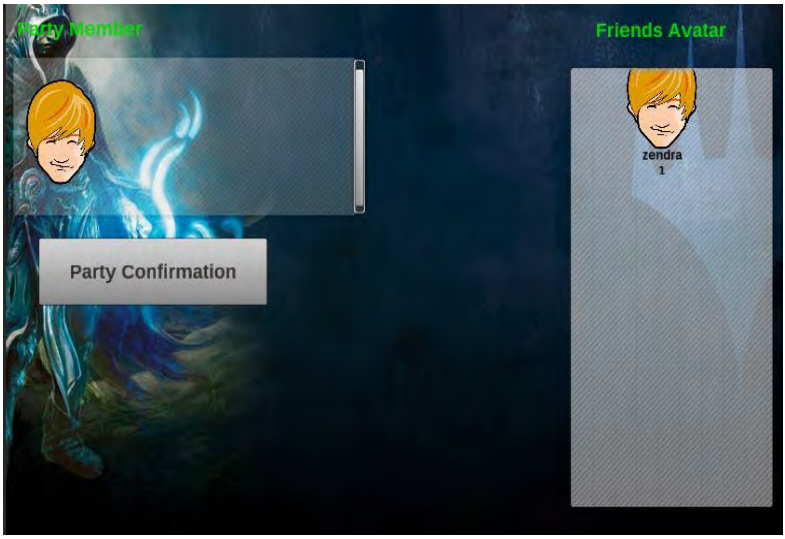
Gambar 5.14 Kondisi Awal Pengujian Integrasi dengan Modul Sosial



Gambar 5.15 Kondisi Awal Daftar Teman pada *Party Editor*



Gambar 5.16 Tampilan Modul Sosial pada Skenario Pengujian UJI.UC-0001



Gambar 5.17 Tampilan Hasil Pengujian pada *Party Editor*

Kondisi awal pengujian adalah ketika pemain telah mendapat permintaan pertemanan seperti dapat dilihat pada tampilan modul sosial pada Gambar 5.14. Sedangkan kondisi awal *party editor* pada modul pertarungan sebelum modul sosial menerima permintaan pertemanan, terlihat seperti pada Gambar 5.15 di mana panel daftar teman di sebelah kanan tampilan masih kosong.

Saat skenario pengujian dilakukan yaitu permintaan diterima, maka daftar teman akan bertambah dan terlihat seperti tampilan pada modul sosial di Gambar 5.16. Setelah permintaan pertemanan diterima pada modul sosial, maka daftar teman pada modul sosial dan modul pertarungan akan bertambah seperti terlihat seperti Gambar 5.17. Dari hasil pengujian, disimpulkan bahwa pengujian integrasi modul sosial dengan modul pertarungan berhasil.

5.2.6 Pengujian Kegagalan Sinkronisasi

Pengujian kegagalan sinkronisasi dilakukan untuk menguji modul pertarungan pemain lawan pemain, bila menghadapi kasus-kasus yang berhubungan dengan kegagalan sinkronisasi. Pengujian yang akan dilakukan adalah pengujian kegagalan koneksi dengan *server* dan pemain terputus dari *server* pada saat bermain.

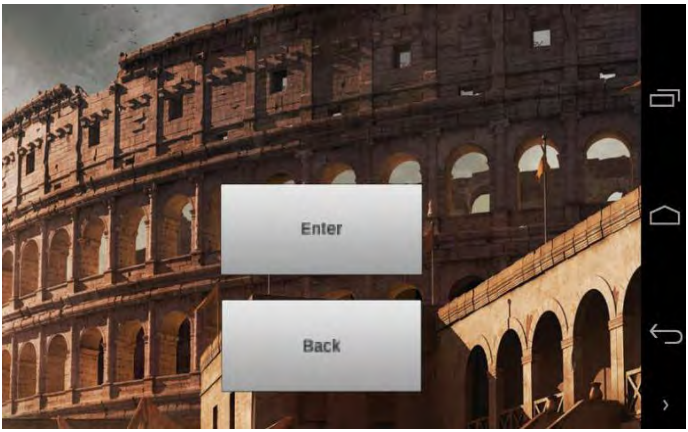
5.2.6.1 *Pengujian Kegagalan Koneksi Dengan Server*

Pengujian kegagalan koneksi dengan *server* dilakukan untuk mengetahui, bagaimana sistem dapat bereaksi terhadap kegagalan koneksi dengan *server*. Spesifikasi lebih lanjut untuk kasus pengujian ini dapat dilihat pada Tabel 5.19.

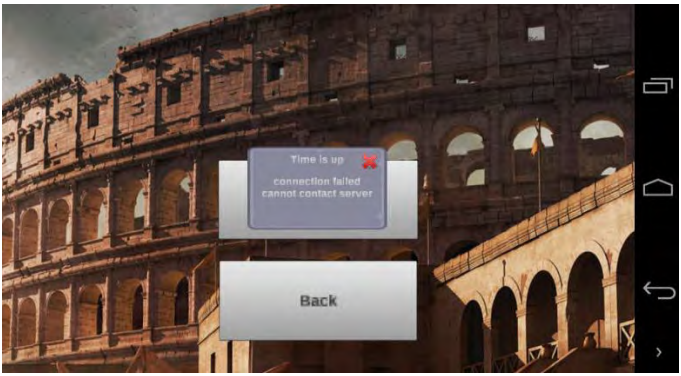
Tabel 5.19 Tabel Pengujian Kegagalan Koneksi Dengan *Server*

ID	UJISINKRONISASI.UC-0001
Nama	Pengujian kegagalan koneksi dengan *server*.
Tujuan Pengujian	Menguji apakah sistem dapat mendeteksi apabila terjadi kegagalan koneksi terhadap *server*.

Skenario	Pengguna menekan tombol *enter*.
Kondisi Awal	Pengguna sudah memasuki halaman *colloseum*.
Masukan	Sentuhan layar.
Hasil Yang Diharapkan	Sistem mengeluarkan pesan bahwa koneksi dengan *server* gagal dilakukan.
Hasil Yang Didapat	Sistem mengeluarkan pesan bahwa koneksi dengan *server* gagal dilakukan.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pemain menerima pesan bahwa koneksi gagal dilakukan.



Gambar 5.18 Kondisi Awal Sebelum Melakukan Koneksi



Gambar 5.19 Kondisi Pada Saat Gagal Melakukan Koneksi Dengan *Server*

Pada hasil yang ditunjukkan Gambar 5.19 dapat terlihat bahwa sistem berhasil mendeteksi adanya kegagalan koneksi dan memberikan pesan kepada pengguna.

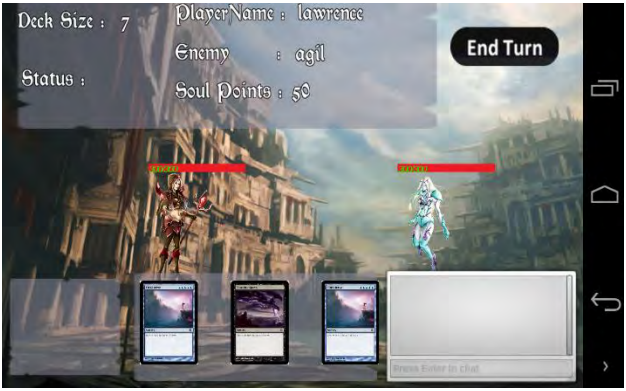
5.2.6.2 *Pengujian Pemain Terputus dari Server*

Pengujian dilakukan untuk mengetahui bagaimana sistem dapat mengatasi kegagalan sinkronisasi. Pada kasus ini adalah terputusnya pemain dari *server*. Spesifikasi lebih lanjut untuk kasus pengujian ini dapat dilihat pada Tabel 5.20.

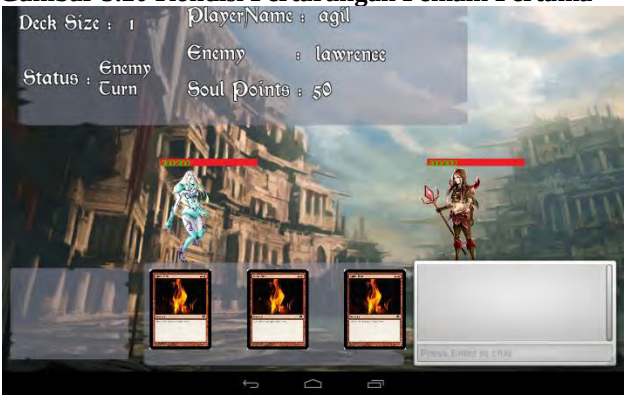
Tabel 5.20 Tabel Pengujian Terputusnya Pemain dari *Server*

ID	UJISINKRONISASI.UC-0002
Nama	Pengujian pemain terputus dari *server*.
Tujuan Pengujian	Menguji apakah sistem dapat menangani kegagalan sinkronisasi yaitu terputus dari *server*, ketika bermain..
Skenario	Menggunakan dua buah perangkat bergerak. Di mana salah satu perangkat bergerak dipaksa terputus dari *server*.
Kondisi Awal	Pemain sudah memasuki halaman pertarungan.

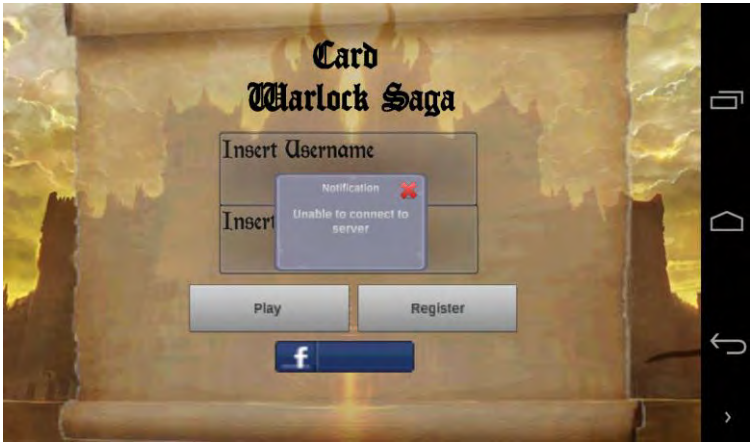
Masukan	Sentuhan layar.
Hasil Yang Diharapkan	Sistem mengeluarkan pesan bahwa pemain yang tidak terputus menang.
Hasil Yang Didapat	Sistem mengeluarkan pesan bahwa pemain yang tidak terputus menang.
Hasil Pengujian	Berhasil.
Kondisi Akhir	Pemain yang masih terhubung menerima pesan bahwa pemain menang.



Gambar 5.20 Kondisi Pertarungan Pemain Pertama



Gambar 5.21 Kondisi Pertarungan Pemain Kedua



Gambar 5.22 Kondisi Pemain Kedua Terputus dari *Server*

Gambar 5.23 Kondisi Pesan Menang Muncul Pada Pemain Pertama

Permainan dijalankan dengan menggunakan dua buah perangkat bergerak. Perangkat bergerak pertama menjalankan pengguna dengan nama agil yang dapat dilihat pada Gambar 5.20. Sedangkan untuk perangkan bergerak kedua menggunakan nama lawrence yang dapat dilihat pada Gambar 5.21. Pengujian dilaksanakan dengan memutus koneksi pada salah satu pengguna

yang dapat dilihat pada Gambar 5.22 terlihat bahwa pengguna yang terputus akan dikembalikan ke halaman login untuk kemudian diberikan notifikasi bahwa koneksi internet terputus. Untuk mengetahui apakah pemain yang masih terhubung dianggap menang. Hasil akhir pengujian dapat dilihat pada Gambar 5.23 yang menandakan bahwa pengujian berhasil dilaksanakan.

5.2.7 Evaluasi Pengujian

Evaluasi pengujian dibagi berdasarkan tiga pengujian yaitu pengujian fungsional, *unit testing*, dan *stress testing*. Pada pengujian fungsional terdapat beberapa evaluasi. Evaluasi pengujian fungsional kasus penggunaan dapat dilihat pada Tabel 5.21.

Tabel 5.21 Tabel Evaluasi Pengujian Fungsional Kasus Penggunaan

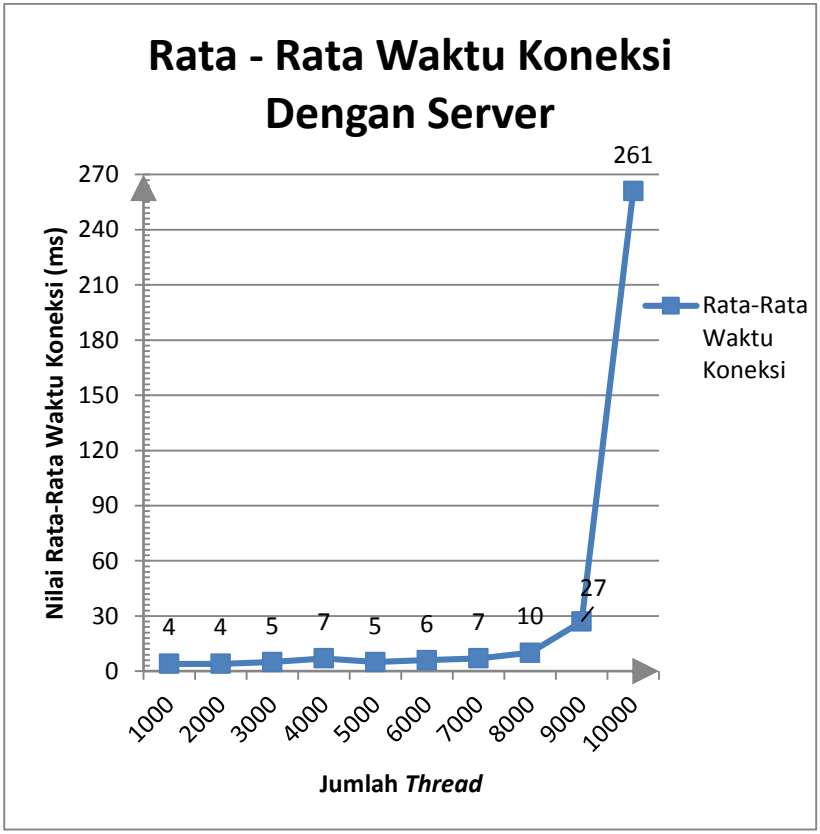
Nomor Kasus Penggunaan	Nama Kasus Penggunaan	Evaluasi
UC0001	Melihat laporan pertarungan	Berhasil
UC0002	Pertarungan pemain lawan pemain	Berhasil
UC0003	Manajemen kartu	Berhasil
UC0004	Manajemen pemain	Berhasil

Pada Tabel 5.21 dapat dilihat bahwa pengujian fungsional pada tiap-tiap kasus penggunaan berhasil dilakukan dan dapat berjalan dengan baik. Selanjutnya adalah evaluasi mengenai unit testing berikut adalah evaluasi hasil pengujian *unit testing.* Dapat dilihat pada pengujian Tabel 5.6 bahwa fungsi-fungsi utama pada modul pertarungan dapat dijalankan dengan baik.

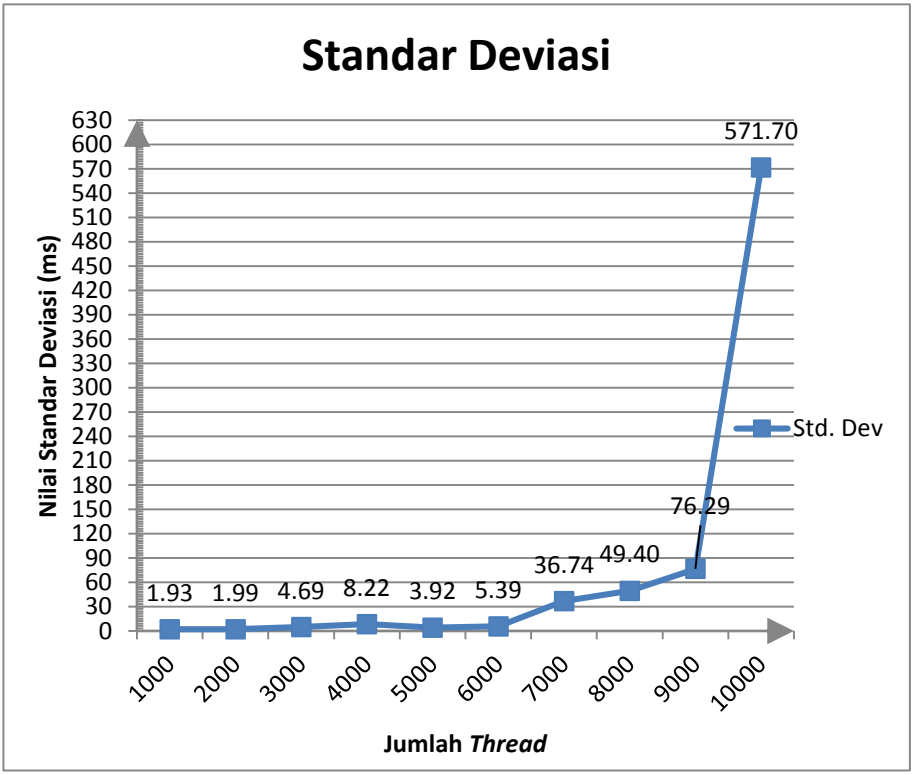
5.2.8 Evaluasi Pengujian *Stress Testing*

Pada bagian ini akan dibahas mengenai evaluasi dari *stress testing*. Rata-rata waktu koneksi adalah rata-rata waktu yang dibutuhkan data berpindah dari klien menuju *server* untuk setiap

sampel pengujian. Semakin kecil nilai rata-rata koneksi, semakin rendah waktu yang dibutuhkan untuk mengirimkan data dari klien ke *server*. Pada Gambar 5.24 dapat dilihat bahwa nilai rata-rata waktu koneksi meningkat sejalan dengan banyaknya koneksi. Peningkatan nilai rata-rata waktu koneksi paling signifikan terjadi pada saat perpindahan koneksi dari jumlah koneksi 8000 sampai dengan 10000. Hal ini menandakan adanya *bottleneck* pada sisi *server* dikarenakan jumlah koneksi yang tinggi.



Gambar 5.24 Grafik Rata-Rata Waktu Koneksi Dengan *Server*

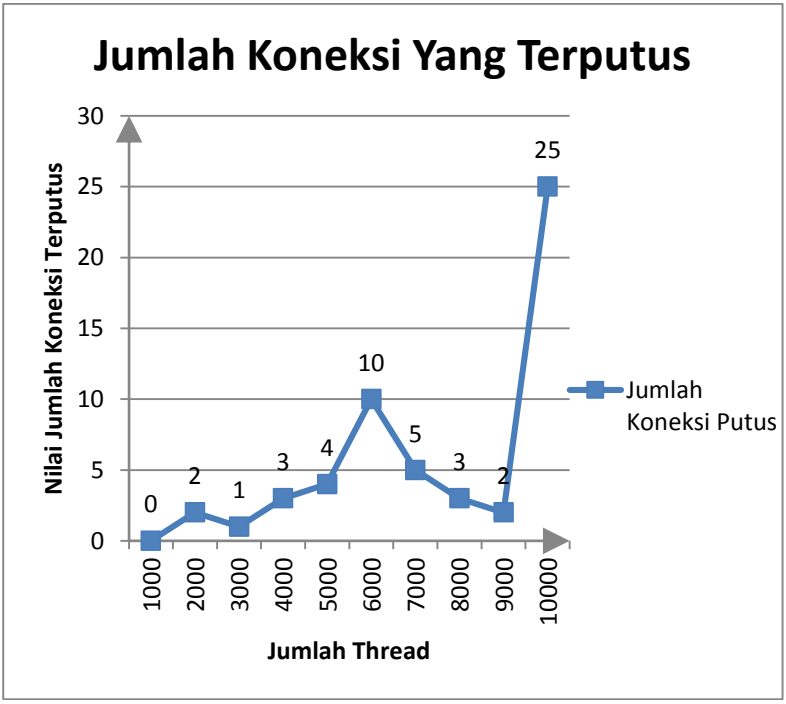


Gambar 5.25 Grafik nilai Standar Deviasi Waktu Koneksi Dengan *Server*

Standar deviasi adalah ukuran keberagaman data. Semakin besar nilai standar deviasi maka semakin beragam data yang dimiliki, semakin kecil nilai standar deviasi maka semakin homogen data yang dimiliki. Pada Gambar 5.25 dapat dilihat bahwa persebaran nilai standar deviasi dari jumlah 1000 koneksi sampai dengan 9000 koneksi bernilai relatif kecil. Dapat disimpulkan bahwa interval waktu pengiriman data antar sampel relatif kecil. Dapat dilihat bahwa trend waktu koneksi dan standar deviasi pada grafik dikisaran koneksi 1000 sampai dengan 8000 bersifat linear sedangkan pada kisaran 8000 sampai dengan 10000

grafik berubah menjadi eksponensial. Hal ini dapat disebabkan oleh *bottleneck* dari sisi *server* sehingga memperbesar waktu koneksi yang dibutuhkan.

Gambar 5.26. merupakan jumlah koneksi yang terputus dari klien. Dapat terlihat bahwa jumlah koneksi putus terbanyak adalah dikisaran 10000 dengan jumlah koneksi putus sebanyak 25. Hal ini disebabkan oleh jumlah waktu yang dibutuhkan paket menuju *server* mengalami *timeout.*



Gambar 5.26 Jumlah Koneksi *Timeout*

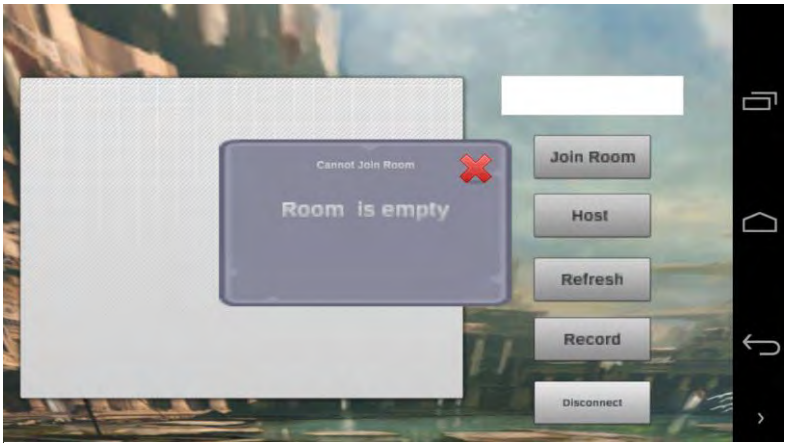
Dari hasil uji coba dapat diketahui bahwa rata-rata waktu koneksi dan standar deviasi paling besar, adalah 261 ms dan 571.70 ms pada jumlah koneksi sebesar 10000 *thread.*

5.3 *Error Handling*

Error handling adalah tindakan pencegahan yang dilakukan oleh sistem untuk mengatasi kesalahan penggunaan pada sistem. *Error handling* memberikan pesan kepada pengguna bahwa suatu tindakan menyalahi kasus penggunaan. Adapun *error handling* yang diimplementasikan pada modul pertarungan pemain lawan pemain adalah sebagai berikut.

5.3.1 Pesan Bahwa *Room* Tidak Ada

Ketika pemain melakukan *join room* tanpa memilih salah satu *room* pada *roomlist.* Sistem akan menampilkan pesan bahwa pengguna tidak dapat melakukan *join room* dikarenakan *room* belum dipilih oleh pengguna. Detail *error handling* dapat dilihat pada Gambar 5.27.



Gambar 5.27 Pesan Bahwa Tidak Ada *Room* Yang Dipilih

5.3.2 Pesan Bahwa Permainan Tidak Dapat Dimulai Dikarenakan Jumlah Pemain Tidak Memadai

Untuk memulai permainan pemain lawan pemain jumlah pemain yang terhubung pada *room* haruslah berjumlah dua pemain. Jika pemain yang terhubung dengan *room* jumlahnya kurang dari

dua, sistem akan menampilkan pesan kepada pengguna bahwa jumlah pemain yang ada tidak memadai untuk memulai permainan. Detail *error handling* dapat dilihat pada Gambar 5.28.

Gambar 5.28 Pesan Bahwa Salah Satu Pemain Ada Yang Kosong

5.3.3 Pesan Bahwa Jumlah Pemain Pada Kelompok Melebihi Jumlah Maksimal

Jumlah *avatar* yang dapat dibawa termasuk dengan *avatar* pemain pada mode *dungeon* adalah tiga orang. Ketika pemain mencoba menambahkan *avatar* melewati jumlah semestinya, sistem akan menampilkan pesan pada pengguna bahwa *avatar* yang dibawa pemain melebihi jumlah yang ditentukan. Detail *error handling* dapat dilihat pada Gambar 5.29.

BAB VI
KESIMPULAN DAN SARAN

Pada bab ini akan diberikan kesimpulan yang diambil selama pengerjaan Tugas Akhir serta saran-saran tentang pengembangan yang dapat dilakukan terhadap Tugas Akhir ini di masa yang akan datang.

6.1 Kesimpulan

Dari hasil selama proses perancangan, implementasi, serta pengujian dapat diambil kesimpulan sebagai berikut.

1. Sistem dapat melakukan sinkronisasi data antar perangkat bergerak dengan menggunakan kerangka kerja Kryonet dan arsitektur jaringan bertipe *client server*.
2. Modul pertarungan pemain lawan pemain dapat terintegrasi dengan modul lainnya dengan menerapkan konsep *design pattern*.
3. Penentuan aturan main secara *realtime* dapat terimplementasikan dengan menggunakan *room*.
4. Rata-rata waktu koneksi terbesar adalah 261 ms dengan nilai standar deviasi terbesar adalah 571.70 ms.
5. *Bottleneck* koneksi pada sisi *server* terjadi pada koneksi *thread* berjumlah 10000 dengan peningkatan rata-rata waktu koneksi menjadi 261 ms.

6.2 Saran

Berikut saran-saran untuk pengembangan dan perbaikan sistem di masa yang akan datang. Di antaranya adalah sebagai berikut.

1. Dapat ditambahkan *mode* kompetisi layaknya turnamen pada pertarungan pemain lawan pemain.
2. Dapat ditambahkan *password* agar *room* tidak dimasuki sembarang orang.

LAMPIRAN A. KODE SUMBER

```
public abstract class DamageReceiver:VisitableObject
    {
        private int maxHealth;

        public int MaxHealth
        {
            get { return maxHealth; }
            set { maxHealth = value; }
        }
        private int currentHealth;

        public int CurrentHealth
        {
            get { return currentHealth; }
            set { currentHealth = value; }
        }

        private string element;

        public void ReceiveDamage(int MaxHealth, int
Damage)
        {


        }

        private string name;

        public string Name
        {
            get { return name; }
            set { name = value; }
        }

        public string Element
        {
            get { return element; }
            set { element = value; }
        }
```

```
        public DamageReceiver(int MaxHealth)
        {
            this.MaxHealth = MaxHealth;
        }
        public DamageReceiver(int MaxHealth, int
CurrentHealth)
            : this(MaxHealth)
        {
            this.CurrentHealth = CurrentHealth;
        }
        public DamageReceiver(int MaxHealth, int
CurrentHealth, string Name)
            : this(MaxHealth, CurrentHealth)
        {
            this.Name = Name;
        }
        public DamageReceiver()
        {
        }
    }
```

Kode Sumber A. 1 `Kelas Model DamageReceiver`

```
public abstract class Enemy : DamageReceiver
    {
        public Enemy(int MaxHealth, int
CurrentHealth, string Name)
            : base(MaxHealth, CurrentHealth, Name)
        {
        }

        public Enemy(int MaxHealth, int
CurrentHealth, string Name, int gold,int exp)
            : this(MaxHealth, CurrentHealth, Name)
        {
            this.GoldForPlayer = gold;
            this.ExpForPlayer = exp;
        }

    public Enemy()
```

```
        {

        }

        private int _expForPlayer;
        private int _goldForPlayer;
        private List<String> _playerGetItem;
        public int ExpForPlayer
        {
            get { return _expForPlayer; }
            set { _expForPlayer = value; }
        }

        public int GoldForPlayer
        {
            get { return _goldForPlayer; }
            set { _goldForPlayer = value; }
        }

        public List<string> PlayerGetItem
        {
            get { return _playerGetItem; }
            set { _playerGetItem = value; }
        }
    }
```

Kode Sumber A. 2 `Kelas Model Enemy`

```
public abstract class Player : DamageReceiver
    {
        private int deckCostPoint;
        private int level;
        private string gender;
        private string job;
        private List<string> deckList;

        public List<string> DeckList
        {
            get { return deckList; }
            set { deckList = value; }
        }
```

```
        private int currentSoulPoints;

        public int CurrentSoulPoints
        {
            get { return currentSoulPoints; }
            set { currentSoulPoints = value; }
        }
        private int maxSoulPoints;

        public int MaxSoulPoints
        {
            get { return maxSoulPoints; }
            set { maxSoulPoints = value; }
        }

        private int handCapacity;

        public int HandCapacity
        {
            get { return handCapacity; }
            set { handCapacity = value; }
        }
        private int experience;

        public int Experience
        {
            get { return experience; }
            set { experience = value; }
        }
        private int gold;

        public int Gold
        {
            get { return gold; }
            set { gold = value; }
        }

        private string rank;

        public string Rank
        {
```

```
            get { return rank; }
            set { rank = value; }
        }

        public int DeckCostPoint
        {
            get { return deckCostPoint; }
            set { deckCostPoint = value; }
        }

        public int Level
        {
            get { return level; }
            set { level = value; }
        }

        public string Gender
        {
            get { return gender; }
            set { gender = value; }
        }

        public string Job
        {
            get { return job; }
            set { job = value; }
        }

        public Player(int MaxHealth, int
CurrentHealth, string Name, int HandCapacity)
            : base(MaxHealth, CurrentHealth, Name)
        {
            this.HandCapacity = HandCapacity;
        }
        public Player(int MaxHealth, int
CurrentHealth, string Name, int HandCapacity, int
MaxSoulPoints, int CurrentSoulPoints)
            : this(MaxHealth, CurrentHealth, Name,
HandCapacity)
        {
            this.maxSoulPoints = MaxSoulPoints;
```

```
            this.currentSoulPoints =
CurrentSoulPoints;
        }

        public Player(int MaxHealth, int
CurrentHealth, string Name, int MaxSoulPoints, int
CurrentSoulPoints, int HandCapacity, int Experience,
int Gold)
            : this
                (MaxHealth, CurrentHealth, Name,
HandCapacity, MaxSoulPoints, CurrentSoulPoints)
        {
            this.Experience = Experience;
            this.Gold = Gold;
            this.Rank = Rank;
        }

        public Player()
        {
        }
```

Kode Sumber A. 3 Kelas Model Player

```
public  abstract class
DamageReceiverAction:MonoBehaviour
      {
        private BuffState _currenState;
        private Boolean isEnemy;

        public Boolean IsEnemy
        {
            get { return isEnemy; }
            set { isEnemy = value; }
        }
        private DamageReceiver character;

        public DamageReceiver Character
        {
            get { return character; }
            set { character = value; }
        }
```

```
        public BuffState CurrenState
        {
            get { return _currenState; }
            set { _currenState = value; }
        }

          private List<Visitor> visitors;
          private List<DamageReceiver>
damageReceivers;

          public void Instantiate()
          {
              visitors= new List<Visitor>();
            visitors.Add(new visitWaterElement());
            visitors.Add(new VisitEarthElement());
            visitors.Add(new VisitThunderElement());
            visitors.Add(new VisitFireElement());
            visitors.Add(new VisitWIndElement());
            damageReceivers= new
List<DamageReceiver>();
          }
          public virtual void
ReceiveDamage(DamageReceiver
damageReceiver,CardsEffect damageGiver,int damage)
          {
              Instantiate();
              foreach (var visit in visitors)
              {

damageReceiver.AcceptVisitor(visit,damageGiver,damag
e);
              }

          }
      }
```

Kode Sumber A. 4 Kelas `DamageReceiverAction`

```
   public abstract class PlayerAction :
DamageReceiverAction
    {
        private Deck deck;
        public Deck Deck
        {
            get { return deck; }
            set { deck = value; }
        }
        public List<GameObject> cards;

        public List<GameObject> Cards
        {
            get { return cards; }
            set { cards = value; }
        }
        private GameObject sceneObject;

        public GameObject SceneObject
        {
            get { return sceneObject; }
            set { sceneObject = value; }
        }
        private List<GameObject> currentHand;


        public int handSize;

        public int HandSize
        {
            get { return handSize; }
            set { handSize = value; }
        }


        public List<GameObject> CurrentHand
        {
            get { return currentHand; }
            set { currentHand = value; }
        }
        public void FirstHand()
        {
```

```
            for (int c = 0; c < handSize; c++)
            {
                if (Deck.Card.Count > 0)
                {

this.CurrentHand.Add(Deck.Draw());

                }
            }

            if
(GameManager.Instance().CurrentPawn.GetComponent<Pla
yerAction>().Character.Name ==
this.Character.Name&&GameManager.Instance().PlayerId
!=null)
            {

GameObject.Find("Objcetloader").GetComponent<BattleO
bjectLoader>().LoadDisplayedCards(this.sceneObject);
            }
        }
        public void Draw()
        {
            if (Deck.Card.Count > 0 &&
currentHand.Count < 3)
            {

                currentHand.Add(Deck.Draw());
            }
        }
        public void InstanceCard()
        {
            cards = new List<GameObject>();
            Debug.Log(this.Character.Name);
            var player = this.Character as Player;
            if (player != null && player.DeckList !=
null)
            {
                foreach (string t in (this.Character
as Player).DeckList)
                {
```

```
                    Cards.Add((GameObject)
Resources.Load("RealCard/"+t, typeof (GameObject)));
                }
            }
        }
    }
```

Kode Sumber A. 5 Kelas `PlayerAction`

```
public  abstract class
EnemyAction:DamageReceiverAction
      {
          private Enemy enemy;

          public Enemy Enemy
          {
              get { return enemy; }
              set { enemy = value; }
          }

          public virtual void AttackAction()
        {

        }

        public override void
ReceiveDamage(DamageReceiver damageReceiver,
CardsEffect damageGiver, int damage)
        {
            base.ReceiveDamage(damageReceiver,
damageGiver, damage);
            if (this.enemy.CurrentHealth <= 0)
            {
                Destroy(this.GameObject);
                GameManager.Instance().PlayerGold +=
enemy.GoldForPlayer;
                GameManager.Instance().PlayerExp +=
enemy.ExpForPlayer;
            }
```

```
Debug.Log(GameManager.Instance().Enemies.Count);
        }
      }
```

Kode Sumber A. 6 Kelas `EnemyAction`

```
public abstract class BattleState : MonoBehaviour
    {
        private GameObject objectLoader;
        private GameObject selectedCard;

        public GameObject SelectedCard
        {
            get { return selectedCard; }
            set { selectedCard = value; }
        }
        private List<GameObject> players;

        public List<GameObject> Players
        {
            get { return players; }
            set { players = value; }
        }
        private BattleStateManager _battleManager;

        public BattleStateManager BattleManager
        {
            get { return _battleManager; }
            set { _battleManager = value; }
        }


        public GameObject ObjectLoader
        {
            get { return objectLoader; }
            set { objectLoader = value; }
        }

        private GameObject currentPlayer;
```

```
        public GameObject CurrentPlayer
        {
            get { return currentPlayer; }
            set { currentPlayer = value; }
        }
        private List<GameObject> enemy;

        public List<GameObject> Enemy
        {
            get { return enemy; }
            set { enemy = value; }
        }
        public abstract void Action();
        public BattleState(GameObject CurrentPlayer)
        {
            this.CurrentPlayer = CurrentPlayer;
        }
        public BattleState(List<GameObject> Players,
List<GameObject> Enemy, BattleStateManager
BattleManager)
        {
            this.Enemy = Enemy;
            this.BattleManager = BattleManager;

        }
        public BattleState(GameObject CurrentPlayer,
GameObject ObjectLoader, BattleStateManager
BattleManager)
            : this(CurrentPlayer)
        {
            this.ObjectLoader = ObjectLoader;
            this.BattleManager = BattleManager;
        }

        public BattleState(GameObject CurrentPlayer,
BattleStateManager BattleManager)
            : this(CurrentPlayer)
        {
            this.BattleManager = BattleManager;
        }
```

```
        public BattleState(GameObject CurrentPlayer,
BattleStateManager BattleManager, GameObject
SelectedCard)
            : this(CurrentPlayer, BattleManager)
        {
            this.SelectedCard = SelectedCard;
        }

        public BattleState(BattleStateManager
battleStateManager)
        {
           BattleManager= battleStateManager;
        }
    }
```

Kode Sumber A. 7 Kelas `BattleState`

```
public class BattleStateManager : MonoBehaviour
    {

        // Use this for initialization
        private GameObject hitObj;
        private GUIStyle style;
        private BattleState currentstate;

        public BattleState Currentstate
        {
            get { return currentstate; }
            set { currentstate = value; }
        }
        public GameObject objectLoader;
        public GameObject Cursor;
        public GameObject endButton;

        public void SelectPawn()
        {
            if (!(this.currentstate is
CardExcutionState) && !(this.currentstate is
PvpEnemyState))
            {
                hitObj = HitCollider();
```

```
                if (hitObj != null &&
hitObj.GetComponent<PlayerAction>() != null &&
hitObj.GetComponent<PlayerAction>().IsEnemy ==
false)
                {
                    Cursor.transform.position = new
Vector3(hitObj.rigidbody2D.transform.position.x,
hitObj.rigidbody2D.transform.position.y +
(hitObj.transform.GetChild(0).renderer.bounds.size.y
/ 2), 0f);
                    GameObject obj =
GameManager.Instance().CurrentPawn = hitObj;
                    currentstate = new
ChangePlayerState(obj, objectLoader, this);
                    currentstate.Action();
                }
            }
        }

        public void SelectEnemy()
        {
            if (!(this.currentstate is
CardExcutionState) && !(this.currentstate is
PvpEnemyState))
            {
                hitObj = HitCollider();
                if (hitObj != null &&
hitObj.GetComponent<EnemyAction>() != null)
                {
                    Cursor.transform.position = new
Vector3(hitObj.rigidbody2D.transform.position.x,
hitObj.rigidbody2D.transform.position.y +
(hitObj.renderer.bounds.size.y/ 2), 0f);
                    Debug.Log(hitObj.name);

GameManager.Instance().CurrentEnemy = hitObj;
                }
            }
        }
        public void EndPlayerTurn()
        {
            hitObj = HitCollider();
```

```
            if (hitObj != null &&
hitObj.name.ToLower().Contains("endbutton"))
            {
                endButton = hitObj;
                if (GameManager.Instance().GameMode
== "pvp")
                {
                    try
                    {
                        var succses = false;
                        succses =
NetworkSingleton.Instance().PlayerClient.Call<bool>(
"sendMessage", "EndTurn-" +
NetworkSingleton.Instance().RoomName + "-" +
GameManager.Instance().PlayerId);
                        Debug.Log(succses ? "send
succes" : "send false");
                        currentstate = new
PvpEnemyState(this);
                        currentstate.Action();
                    }

                    catch (Exception e)
                    {
                        Debug.Log("eror
state" + e.Message);
                    }
                }
                else
                {
                    currentstate = new
EnemyState(GameManager.Instance().Players,
GameManager.Instance().Enemies, this);
                    currentstate.Action();
                }
            }
        }

        public void DrawCursor()
        {
            if (GameManager.Instance().CurrentPawn
!= null)
```

```
                Cursor.transform.position = new
Vector3(GameManager.Instance().CurrentPawn.rigidbody
2D.transform.position.x,
GameManager.Instance().CurrentPawn.rigidbody2D.trans
form.position.y +

(GameManager.Instance().CurrentPawn.GameObject.rende
rer.bounds.size.y / 2), 0f);

        }

        public void CheckWinorLose()
        {

            if (GameManager.Instance().Enemies.Count
<= 0)
            {
                this.currentstate= new
WinState(this);
                this.currentstate.Action();
            }
            else if
(GameManager.Instance().Players.Count <= 0)
            {
                this.currentstate= new
LoseState(this);
                this.currentstate.Action();
            }
            if (GameManager.Instance().GameMode ==
"pvp")
            {
                string serverMessage =
NetworkSingleton.Instance().ServerMessage;
                Debug.Log(serverMessage);
                var message = serverMessage.Split('-
');

                if
(!serverMessage.Contains("Disconnected")) return;
                if
(GameManager.Instance().PlayerId.Equals(message[1]))
                {
```

```
                    this.currentstate= new
LoseState(this);
                    this.currentstate.Action();
                }
                else
                {
                    this.currentstate= new
WinState(this);
                    this.currentstate.Action();

NetworkSingleton.Instance().ServerMessage = "";
                }


            }


        }
        void Start()
        {
if (GameManager.Instance().GameMode == "pvp")
            {
                ChekHost();
            }
        }


        // Update is called once per frame
        private void Update()
        {
                EndPlayerTurn();
                SelectPawn();
                SelectEnemy();
                CheckWinorLose();
                GameManager.Instance().BattleState =
currentstate;
            //}

        }

        void OnGUI()
        {
```

```
            if (!(currentstate is
CardExcutionState)) return;

GameManager.Instance().CurrentPawn.GetComponent<Anim
ator>().SetBool("IsAttack", false);
            GUI.Box(new Rect((Screen.width / 2) -
50, (Screen.height / 2) - 75, 100, 150), "Execute
Effect");
            if (GameManager.Instance().CurrentEnemy
== null )
            {
                GameManager.Instance().CurrentEnemy
= GameManager.Instance().Enemies[0];
            }
            if (GUI.Button(new Rect((Screen.width /
2) - 50, (Screen.height / 2) - 25, 100, 50), "Yes"))
            {


                currentstate.Action();

            }

            if (!GUI.Button(new Rect((Screen.width /
2) - 50, ((Screen.height / 2) - 25) + 50, 100, 50),
"No")) return;
            var obj =
GameManager.Instance().CurrentPawn;
            currentstate = new
ChangePlayerState(obj, objectLoader, this);
            currentstate.Action();
        }
        public GameObject HitCollider()
        {
            GameObject obj = null;
            RaycastHit2D hit =
Physics2D.Raycast(Camera.main.ScreenToWorldPoint(Inp
ut.mousePosition), Vector2.zero);
            if (Input.GetMouseButtonUp(0))
            {
                if (hit.collider != null)
                {
```

```
                    obj = hit.collider.GameObject;
                }
            }
            return obj;
        }
        public void ChekHost()
        {
            if
(NetworkSingleton.Instance().HostPlayer !=
GameManager.Instance().PlayerId)
            {
                try
                {
                    currentstate = new
PvpEnemyState(this);

GameManager.Instance().BattleState = currentstate;
                    currentstate.Action();
                }
                catch (Exception e)
                {
                    Debug.Log("errorcheckhoststate"
+ e.Message);
                }

            }
        }
    }
```

Kode Sumber A. 8 Kelas `BattleStateManager`

```
public class CardExcutionState : BattleState
    {

        private bool isCardConfimed;

        public CardExcutionState(GameObject
CurrentPlayer, BattleStateManager BattleManager,
GameObject SelectedCard)
            : base(CurrentPlayer, BattleManager,
SelectedCard)
```

```
        {

        }

        public override void Action()
        {

GameManager.Instance().CurrentPawn.GetComponent<Anim
ator>().SetBool("IsAttack", true);
            if (GameManager.Instance().GameMode ==
"pvp")
            {
                bool succses = false;
                succses =
NetworkSingleton.Instance().PlayerClient.Call<bool>(
"sendMessage", "SendMessage-"
+NetworkSingleton.Instance().RoomName+"-
"+GameManager.Instance().PlayerId+"-
"+"CardEffect"+"-
"+GameManager.Instance().CurrentCard.GetComponent<Ca
rdsEffect>().CardName);
                Debug.Log(succses ? "send succes" :
"send false");

            }
            else
            {

SelectedCard.GetComponent<CardsEffect>().SetTarget("
enemy");

SelectedCard.GetComponent<CardsEffect>().Effect();

            }

            var gobj =
GameManager.Instance().CurrentPawn.GetComponent<Play
erAction>().CurrentHand.FirstOrDefault(obj =>
this.SelectedCard.name == obj.name + "(Clone)");

GameManager.Instance().CurrentPawn.GetComponent<Play
erAction>().CurrentHand.Remove(gobj);
```

```
            Destroy(this.SelectedCard);
            this.BattleManager.Currentstate = new
ChangePlayerState(this.CurrentPlayer,
            this.BattleManager.objectLoader,
this.BattleManager);

this.BattleManager.Currentstate.Action();
        }
    }
```

Kode Sumber A. 9 Kelas `CardExcecutionState`

```
   public class ChangePlayerState : BattleState
    {
        public ChangePlayerState(GameObject
CurrentPlayer, GameObject ObjectLoader,
BattleStateManager BattleManager)
            : base(CurrentPlayer, ObjectLoader,
BattleManager)
        {
        }
        public override void Action()
        {
            BattleManager.endButton.SetActive(true);
            BattleManager.Cursor.renderer.enabled =
true;

this.ObjectLoader.GetComponent<BattleObjectLoader>()
.DestroyDisplayedCards();

this.ObjectLoader.GetComponent<BattleObjectLoader>()
.LoadDisplayedCards(this.CurrentPlayer);
        }
    }
```

Kode Sumber A. 10 Kelas `ChangePlayerState`

```
public class DrawState : BattleState
    {
```

```
        public DrawState(GameObject CurrentPlayer,
GameObject Objectloader, BattleStateManager
BattleManager)
            : base(CurrentPlayer, Objectloader,
BattleManager)
        {
        }

        public override void Action()
        {
            int totalCards=0;
            foreach (GameObject obj in
GameManager.Instance().Players)
            {
                totalCards +=
obj.GetComponent<PlayerAction>().Deck.Card.Count +

obj.GetComponent<PlayerAction>().CurrentHand.Count;
                var emptyhand =
obj.GetComponent<PlayerAction>().HandSize -
obj.GetComponent<PlayerAction>().CurrentHand.Count()
;
                for (int c = 0; c < emptyhand; c++)
                {

obj.GetComponent<PlayerAction>().Draw();
                }
            }
            if (totalCards == 0)
            {
                BattleManager.Currentstate= new
LoseState(BattleManager);
                BattleManager.Currentstate.Action();
            }
            BattleManager.Currentstate = new
ChangePlayerState(CurrentPlayer,
BattleManager.objectLoader, BattleManager);
            BattleManager.Currentstate.Action();
        }
    }
```

Kode Sumber A. 11 `Kelas DrawState`

```
public class EnemyState : BattleState
    {
        public EnemyState(List<GameObject> Players,
List<GameObject> Enemy, BattleStateManager
BattleManager)
            : base(Players, Enemy, BattleManager)
        {

        }
        public override void Action()
        {
            foreach (GameObject obj in this.Enemy)
            {

obj.GetComponent<EnemyAction>().AttackAction();
            }

            BattleManager.endButton.SetActive(true);
            BattleManager.Cursor.renderer.enabled =
true;
            BattleManager.Currentstate = new
DrawState(GameManager.Instance().CurrentPawn,
BattleManager.objectLoader, BattleManager);
            BattleManager.Currentstate.Action();

        }
    }
```

Kode Sumber A. 12 Kelas `EnemyState`

```
public class LoseState:BattleState
      {
        public LoseState(BattleStateManager
battleStateManager) : base(battleStateManager)
        {
        }
        public override void Action()
        {
            if (GameManager.Instance().GameMode ==
"pvp")
            {
```

```
                var succses = false;
                succses =
NetworkSingleton.Instance().PlayerClient.Call<bool>(
"sendMessage", "SendMessage-" +
NetworkSingleton.Instance().RoomName + "-" +
GameManager.Instance().PlayerId + "-" +
"Disconnected" + "-" + "Disconnected");
                Debug.Log(succses ? "send succes" :
"send false");
            }
      GameManager.Instance().PermainanStatus =
"lose";
            Application.LoadLevel("AfterBattle2");
        }
    }
```

Kode Sumber A. 13 Kelas `LoseState`

```
public class WinState: BattleState
      {
        public WinState(BattleStateManager
battleStateManager) : base(battleStateManager)
        {
        }
        public override void Action()
        {  GameManager.Instance().PermainanStatus =
"win";
            Application.LoadLevel("AfterBattle2");
        }
    }
```

Kode Sumber A. 14 Kelas `WinState`

```
public class BattleObjectLoader : MonoBehaviour
    {

        // Use this for initialization
        private int pawnsnumber;
        public int enemyCount;
        public List<GameObject> pawnsPosisition;
        public GameObject grid;
        private List<GameObject> DisplayedCards;
```

```
        private int currentPawnNumber;
        private AbstractFactory factory;
        public List<GameObject> enemyPosition;
        public GameObject enemyOnlinePosisiton;

        public void LoadBackground()
        {
            if (GameManager.Instance().GameMode !=
"pvp")
            {
                factory = new BackgroundFactory();
                factory.InstantiateObject();

factory.CreateBackground(TextureSingleton.Instance()
.TextureTiles);
                Debug.Log("Bg " +
TextureSingleton.Instance().TextureTiles);
            }
        }
        public void LoadPlayer()
        {
            factory = new PlayerFactory();
            factory.InstantiateObject();
            if (GameManager.Instance().GameMode ==
"pvp")
            {

factory.CreatePlayer(GameManager.Instance().PlayerId
, pawnsPosisition[0]);
            }
            else
            {

factory.CreatePlayer(GameManager.Instance().PlayerId
, pawnsPosisition[0]);
                if (GameManager.Instance().PartyId
!= null)
                {
                    LoadParty();
                }
            }
```

```
        }

        public void LoadParty()
        {
            int temp = 1;
            foreach (string party in
GameManager.Instance().PartyId)
            {
                factory = new PlayerFactory();
                factory.InstantiateObject();
                factory.CreatePlayer(party,
pawnsPosisition[temp]);
                temp++;
                Debug.Log(party);
            }
            temp = 0;
        }

        public void LoadDisplayedCards(GameObject
pawn)
        {
            if (pawn != null)
            {

                foreach (GameObject t in
pawn.GetComponent<PlayerAction>().CurrentHand)
                {
                    var obj =
NGUITools.AddChild(grid, t);
                    DisplayedCards.Add(obj);
                }

            }
        }

        public void DestroyDisplayedCards()
        {
            foreach (GameObject obj in
DisplayedCards)
            {
                Destroy(obj);
            }
```

```
        }

        public void LoadEnemy()
        {
            if (GameManager.Instance().GameMode ==
"pvp")
            {
                factory = new OnlineEnemyFactory();
                factory.InstantiateObject();
                factory.CreatePlayer(

GameManager.Instance().PlayerId.ToLower() !=
NetworkSingleton.Instance().HostPlayer
                        ?
NetworkSingleton.Instance().HostPlayer
                        :
NetworkSingleton.Instance().JoinPlayer,
                    enemyOnlinePosisiton);
            }
            else
            {
                if
(TextureSingleton.Instance().TextureTiles ==
"0_Lavaland")
                {
                    if
(TextureSingleton.Instance().IdButton == "@Fire_0")
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("FireSlime", enemyPosition[0]);

factory.CreateEnemy("FireKingSlime",
enemyPosition[1]);

                    }
                  else if
(TextureSingleton.Instance().IdButton == "@Fire_1")
                  {
```

```
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("FireNymph", enemyPosition[0]);

factory.CreateEnemy("FireNymph", enemyPosition[1]);
                    }
                    else
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("FireDragon", enemyPosition[0]);

factory.CreateEnemy("FireDragon", enemyPosition[1]);
                    }
                }
                else if
(TextureSingleton.Instance().TextureTiles ==
"0_Greenland")
                {
                    if
(TextureSingleton.Instance().IdButton == "@Earth_0")
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("EarthSlime", enemyPosition[0]);

factory.CreateEnemy("EarthKingSlime",
enemyPosition[1]);
                    }
                    else if
(TextureSingleton.Instance().IdButton == "@Earth_1")
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();
```

```
factory.CreateEnemy("EarthNymph", enemyPosition[0]);

factory.CreateEnemy("EarthNymph", enemyPosition[1]);
                    }
                    else if
(TextureSingleton.Instance().IdButton == "@Earth_2")
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("Mandrake", enemyPosition[0]);

factory.CreateEnemy("Mandragora", enemyPosition[1]);
                    }
                    else
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("Canotre", enemyPosition[0]);

factory.CreateEnemy("Treant", enemyPosition[1]);
                    }
                }
                else if
(TextureSingleton.Instance().TextureTiles ==
"0_Iceland")
                {
                    if
(TextureSingleton.Instance().IdButton == "@Water_0")
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("WaterSlime", enemyPosition[0]);

factory.CreateEnemy("WaterKingSlime",
enemyPosition[1]);
```

```
                    }
                    else if
(TextureSingleton.Instance().IdButton == "@Water_1")
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("WaterNymph", enemyPosition[0]);

factory.CreateEnemy("WaterNymph", enemyPosition[1]);
                    }
                    else
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("WaterKingSlime",
enemyPosition[0]);

factory.CreateEnemy("WaterKingSlime",
enemyPosition[1]);
                    }
                }
                else if
(TextureSingleton.Instance().TextureTiles ==
"0_Wetland")
                {
                    if
(TextureSingleton.Instance().IdButton ==
"@Thunder_0")
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("ThunderSlime",
enemyPosition[0]);

factory.CreateEnemy("ThunderKingSlime",
enemyPosition[1]);
```

```
                    }
                    else if
(TextureSingleton.Instance().IdButton ==
"@Thunder_1")
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("ThunderNymph",
enemyPosition[0]);

factory.CreateEnemy("ThunderNymph",
enemyPosition[1]);
                    }
                    else
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("ThunderKingSlime",
enemyPosition[0]);

factory.CreateEnemy("ThunderKingSlime",
enemyPosition[1]);
                    }
                }
                else if
(TextureSingleton.Instance().TextureTiles ==
"0_Windyhill")
                {
                    if
(TextureSingleton.Instance().IdButton == "@Wind_0")
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("WindSlime", enemyPosition[0]);
```

```
factory.CreateEnemy("WindKingSlime",
enemyPosition[1]);
                    }
                    else if
(TextureSingleton.Instance().IdButton == "@Wind_1")
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("WindNymph", enemyPosition[0]);

factory.CreateEnemy("WindNymph", enemyPosition[1]);
                    }
                    else
                    {
                        factory = new
EnemyFactory();
                        factory.InstantiateObject();

factory.CreateEnemy("WindKingSlime",
enemyPosition[0]);

factory.CreateEnemy("WindKingSlime",
enemyPosition[1]);
                    }
                }
            }

        }
        void Awake()
        {
            LoadBackground();
            DisplayedCards = new List<GameObject>();
            Screen.orientation =
ScreenOrientation.LandscapeLeft;
            LoadEnemy();
            LoadPlayer();


        }
```

```
        void Start()
        {
            FinshihLoadObject();
        }
        void Update()
        {

grid.GetComponent<UIGrid>().Reposition();
        }
        void FinshihLoadObject()
        {
            if (GameManager.Instance().GameMode ==
"pvp")
            {
                var succses = false;
                succses =
NetworkSingleton.Instance().PlayerClient.Call<bool>(
"sendMessage", "LoadFinish" +"-"+
NetworkSingleton.Instance().RoomName + "-" +
GameManager.Instance().PlayerId);
                Debug.Log(succses ? "send succes" :
"send false");
            }
        }

    }
```

Kode Sumber A. 15 Kelas `BattleObjectLoader`

```
[Test]
    public void DamageReceiverFireTest()
    {
        var sorcerer= new Sorcerer();
        sorcerer.MaxHealth = 200;
        sorcerer.CurrentHealth = 200;
        sorcerer.HandCapacity = 2;
        sorcerer.MaxSoulPoints = 30;
        sorcerer.CurrentSoulPoints = 20;
        var action = new WarlockAction();
        action.Character = sorcerer;
```

```
        action.ReceiveDamage(action.Character, new
EarthCard(), 100);

Assert.LessOrEqual(action.Character.CurrentHealth,15
0);
        action.ReceiveDamage(action.Character, new
WaterCard(), 50);

Assert.LessOrEqual(action.Character.CurrentHealth,10
0);


    }

    [Test]
    public void DamageReceiverWaterTest()
    {
        var sorcerer = new Warlock();
        sorcerer.MaxHealth = 200;
        sorcerer.CurrentHealth = 200;
        sorcerer.HandCapacity = 2;
        sorcerer.MaxSoulPoints = 30;
        sorcerer.CurrentSoulPoints = 20;
        var action = new WarlockAction();
        action.Character = sorcerer;
        action.ReceiveDamage(action.Character, new
ThunderCard(), 100);

Assert.LessOrEqual(action.Character.CurrentHealth,
150);
        action.ReceiveDamage(action.Character, new
EarthCard(), 50);

Assert.LessOrEqual(action.Character.CurrentHealth,
100);
    }
    [Test]
    public void DamageReceiverEarthTest()
    {
        var sorcerer = new Magician();
        sorcerer.MaxHealth = 200;
        sorcerer.CurrentHealth = 200;
```

```
        sorcerer.HandCapacity = 2;
        sorcerer.MaxSoulPoints = 30;
        sorcerer.CurrentSoulPoints = 20;
        var action = new WarlockAction();
        action.Character = sorcerer;
        action.ReceiveDamage(action.Character, new
WindCard(), 100);

Assert.LessOrEqual(action.Character.CurrentHealth,
150);
        action.ReceiveDamage(action.Character,new
FireCard(), 50);

Assert.LessOrEqual(action.Character.CurrentHealth,
100);
    }
    [Test]
    public void DamageReceiverThunderTest()
    {
        var sorcerer = new Wizard();
        sorcerer.MaxHealth = 200;
        sorcerer.CurrentHealth = 200;
        sorcerer.HandCapacity = 2;
        sorcerer.MaxSoulPoints = 30;
        sorcerer.CurrentSoulPoints = 20;
        var action = new WarlockAction();
        action.Character = sorcerer;
        action.ReceiveDamage(action.Character, new
WaterCard(), 100);

Assert.LessOrEqual(action.Character.CurrentHealth,
150);
        action.ReceiveDamage(action.Character, new
WindCard(), 50);

Assert.LessOrEqual(action.Character.CurrentHealth,
100);
    }
    [Test]
    public void DamageReceiverWindTest()
    {
        var sorcerer = new GrandMagus();
```

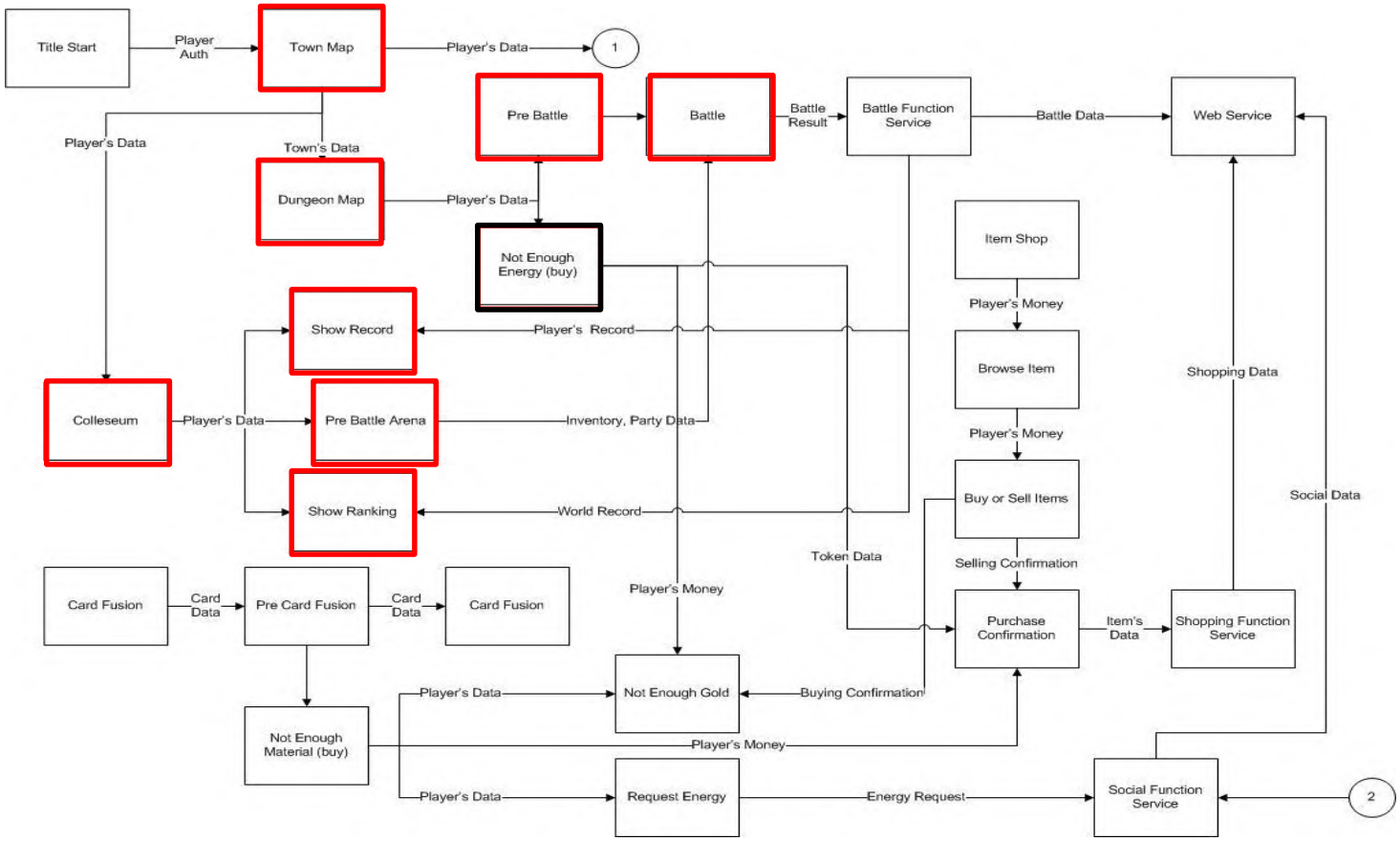
```
        sorcerer.MaxHealth = 200;
        sorcerer.CurrentHealth = 200;
        sorcerer.HandCapacity = 2;
        sorcerer.MaxSoulPoints = 30;
        sorcerer.CurrentSoulPoints = 20;
        var action = new WarlockAction();
        action.Character = sorcerer;
        action.ReceiveDamage(action.Character, new
ThunderCard(), 100);

Assert.LessOrEqual(action.Character.CurrentHealth,
150);
        action.ReceiveDamage(action.Character, new
EarthCard(), 50);

Assert.LessOrEqual(action.Character.CurrentHealth,
100);
    }
    [Test]
    public void TestCardEffectProtocol()
    {   var sorcerer= new GrandMagus();
        sorcerer.MaxHealth = 200;
        sorcerer.CurrentHealth = 200;
        sorcerer.HandCapacity = 2;
        sorcerer.MaxSoulPoints = 30;
        sorcerer.CurrentSoulPoints = 20;
        GameObject action= new GameObject();
        action.AddComponent("WarlockAction");

action.GetComponent<WarlockAction>().Character =
sorcerer;
        GameManager.Instance().AddPlayer(action);
        GameManager.Instance().AddEnemy(action);
        Invoker invoke = new Invoker();
        invoke.AddCommand(new
CardExecuteCommand("SplitFire","enemy"));
        invoke.RunCommand();
        GameObject.DestroyImmediate(action);
    }


    [Test]
```

```
    public void TestEndPhaseProtocol()
    {
        Sorcerer sorcerer = new Sorcerer();
        sorcerer.MaxHealth = 200;
        sorcerer.CurrentHealth = 200;
        sorcerer.HandCapacity = 2;
        sorcerer.MaxSoulPoints = 30;
        sorcerer.Name = "agil";
        sorcerer.CurrentSoulPoints = 20;
        sorcerer.DeckList= new List<string>()
        {
            {"SplitFire"},
            {"TidalWave"}
        };
        GameObject action = new GameObject();
        action.AddComponent("WarlockAction");

action.GetComponent<WarlockAction>().Character =
sorcerer;

action.GetComponent<WarlockAction>().HandSize = 3;
        GameManager.Instance().AddPlayer(action);
        GameManager.Instance().AddEnemy(action);
        GameManager.Instance().CurrentPawn = action;
        Invoker invoke = new Invoker();
        invoke.AddCommand(new EndPhaseCommand(new
BattleStateManager()));
        GameObject.DestroyImmediate(action);
    }
```

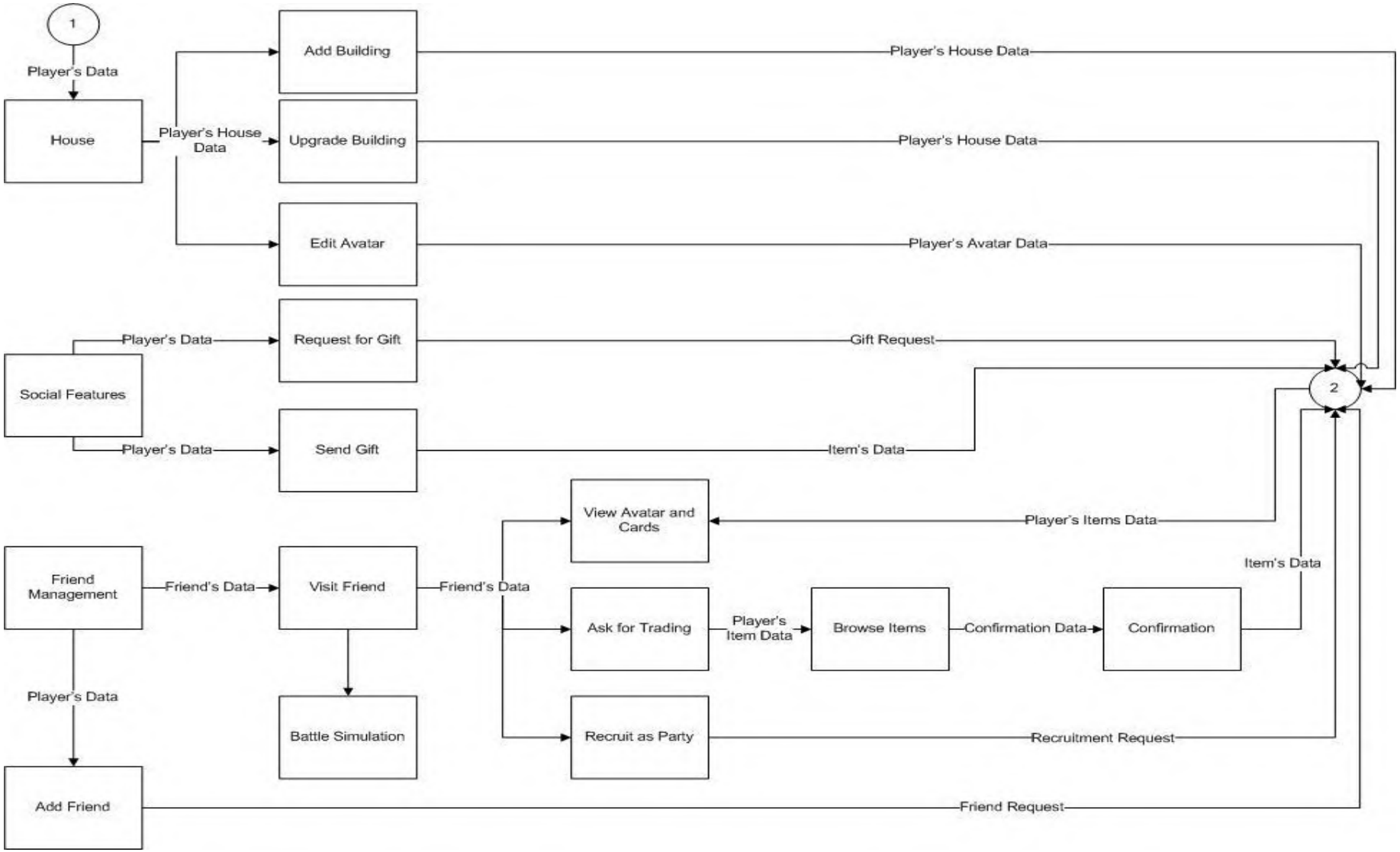
Kode Sumber A. 16 Fungsi-Fungsi Pengujian *Unit Testing*

[Halaman ini sengaja dikosongkan]

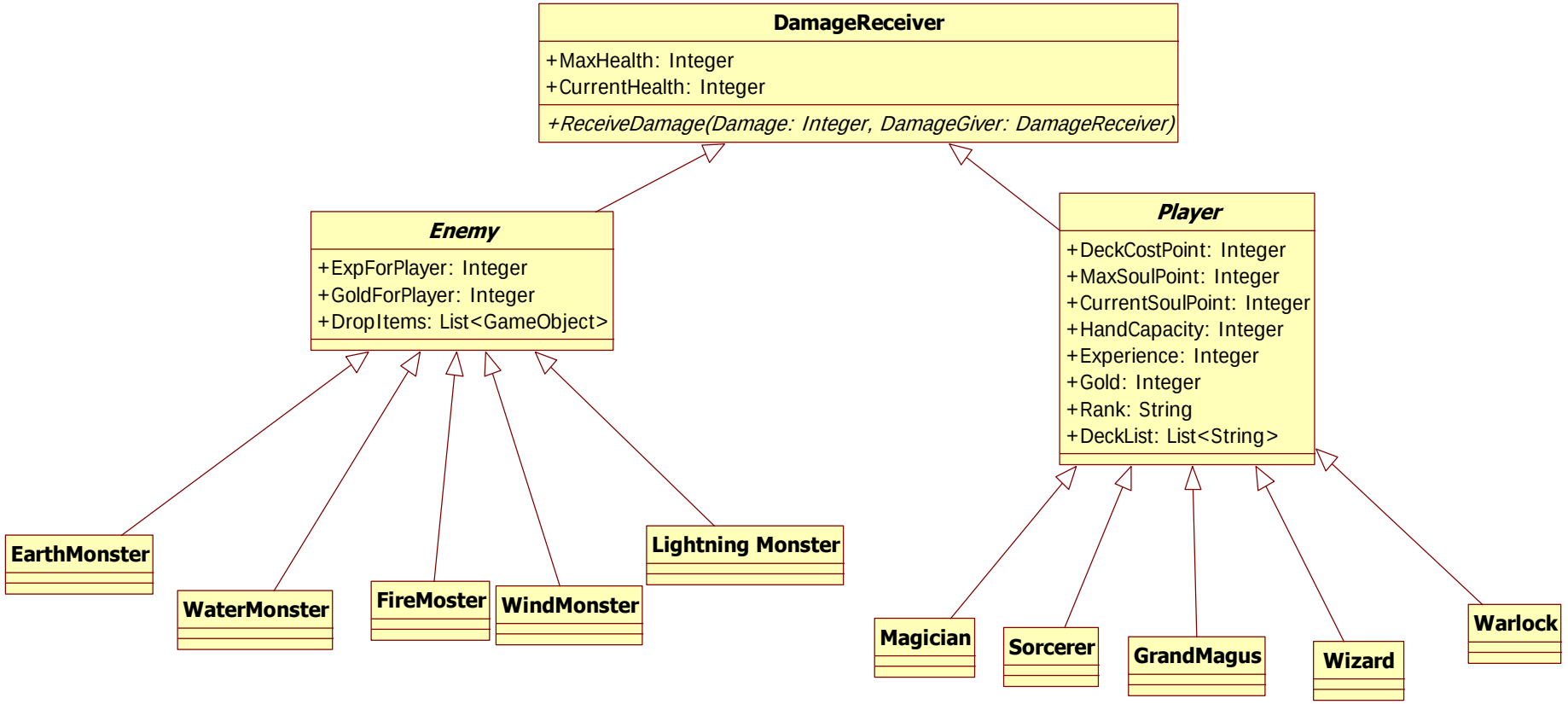
LAMPIRAN B. DIAGRAM



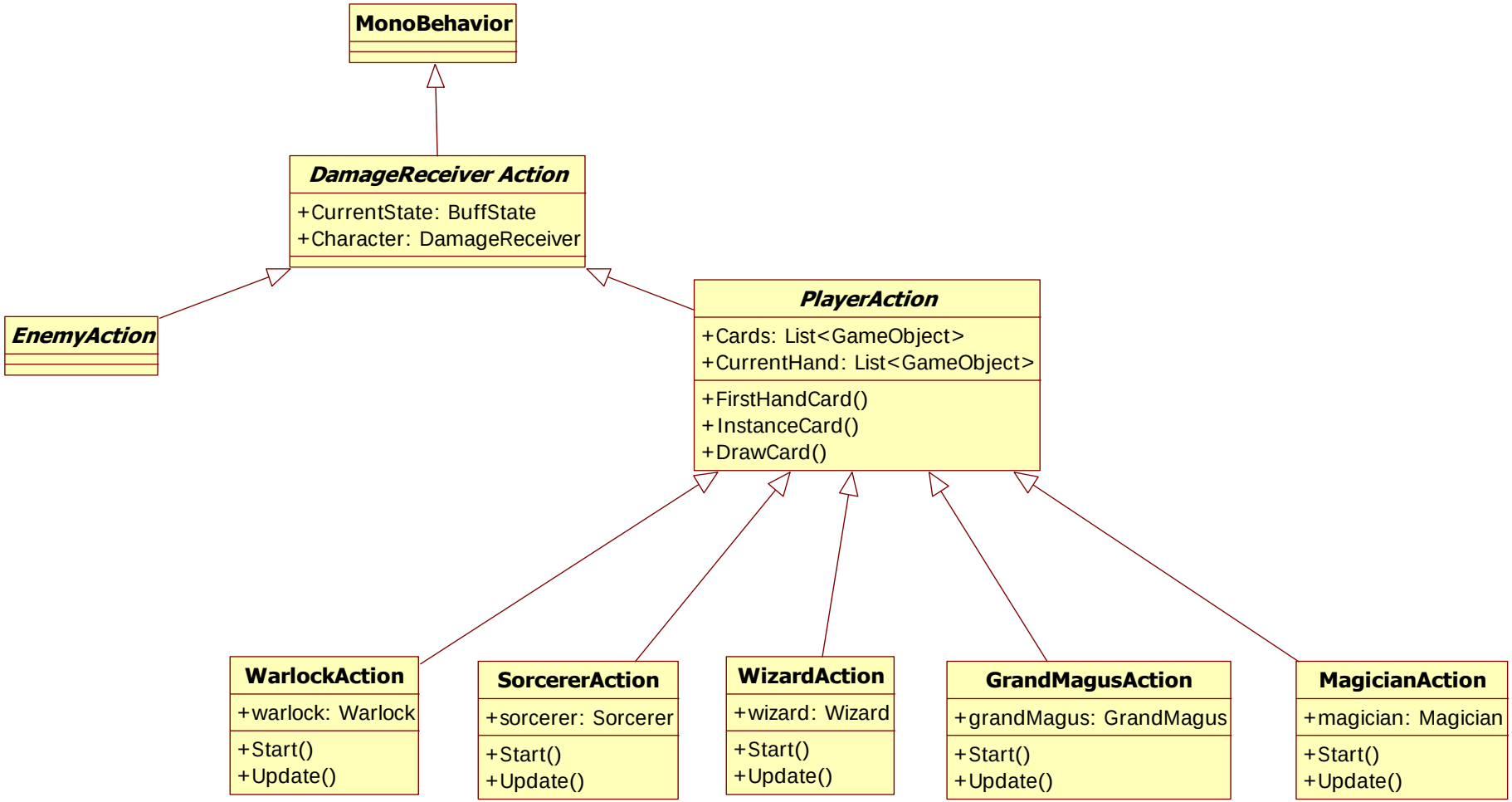
Gambar B. 1 Diagram Blok Card Warlock Saga Bagian 1



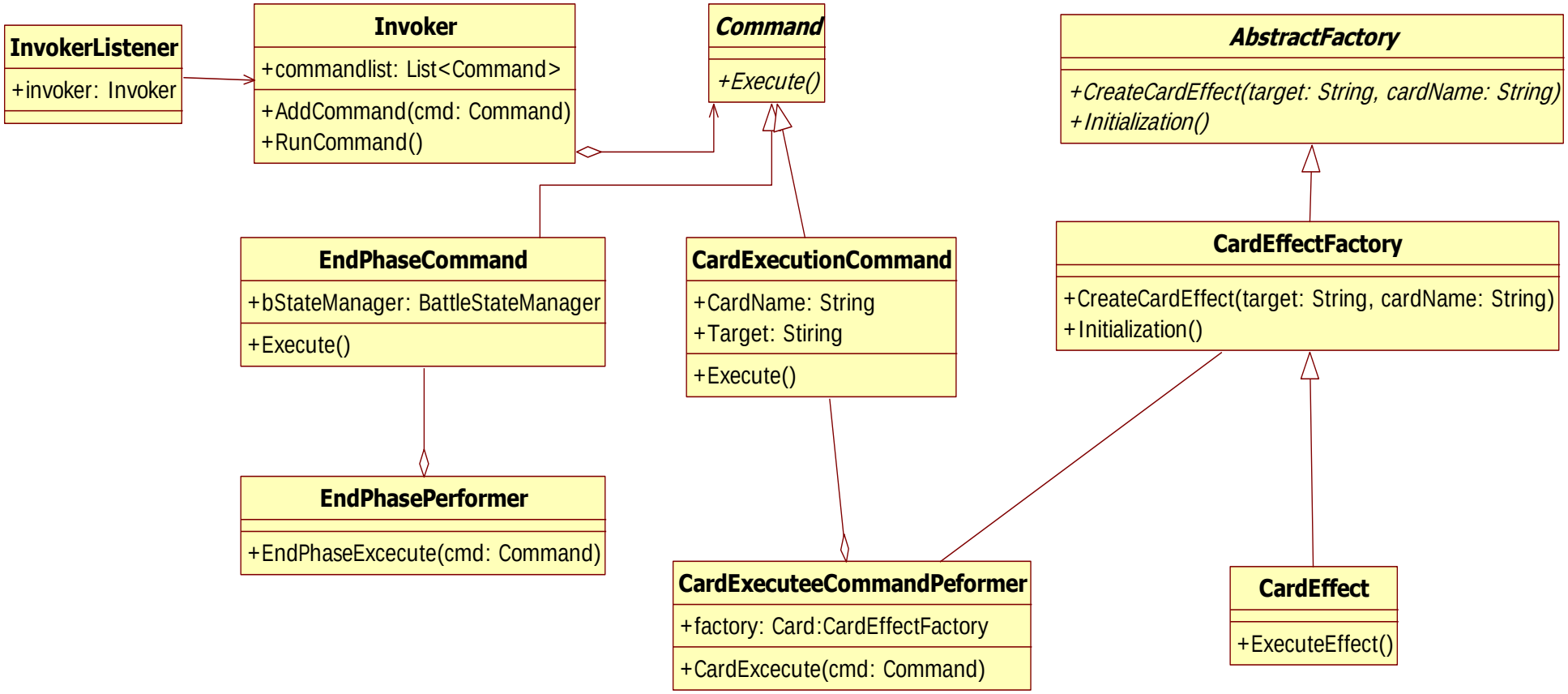
Gambar B. 2 Diagram Blok Card Warlock Saga Bagian 2



Gambar B. 3 Diagram Kelas Lapisan Model

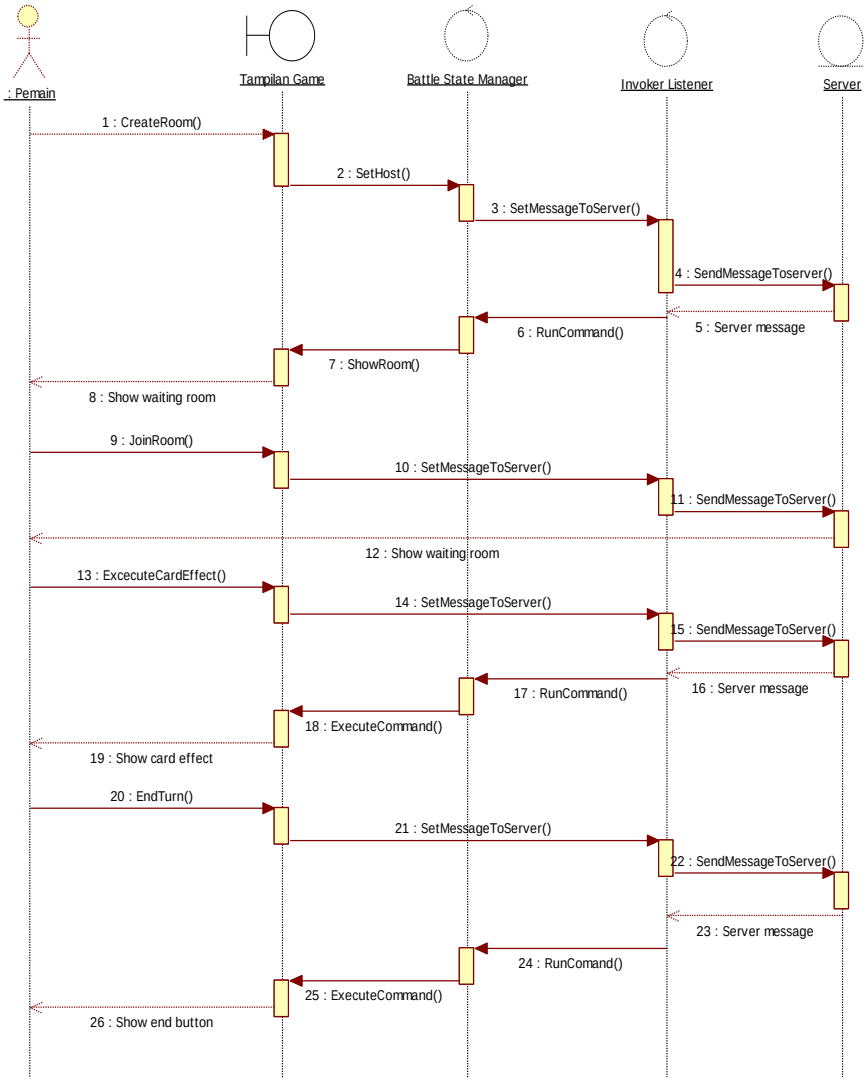


Gambar B. 4 Diagram Kelas Lapisa Kontrol



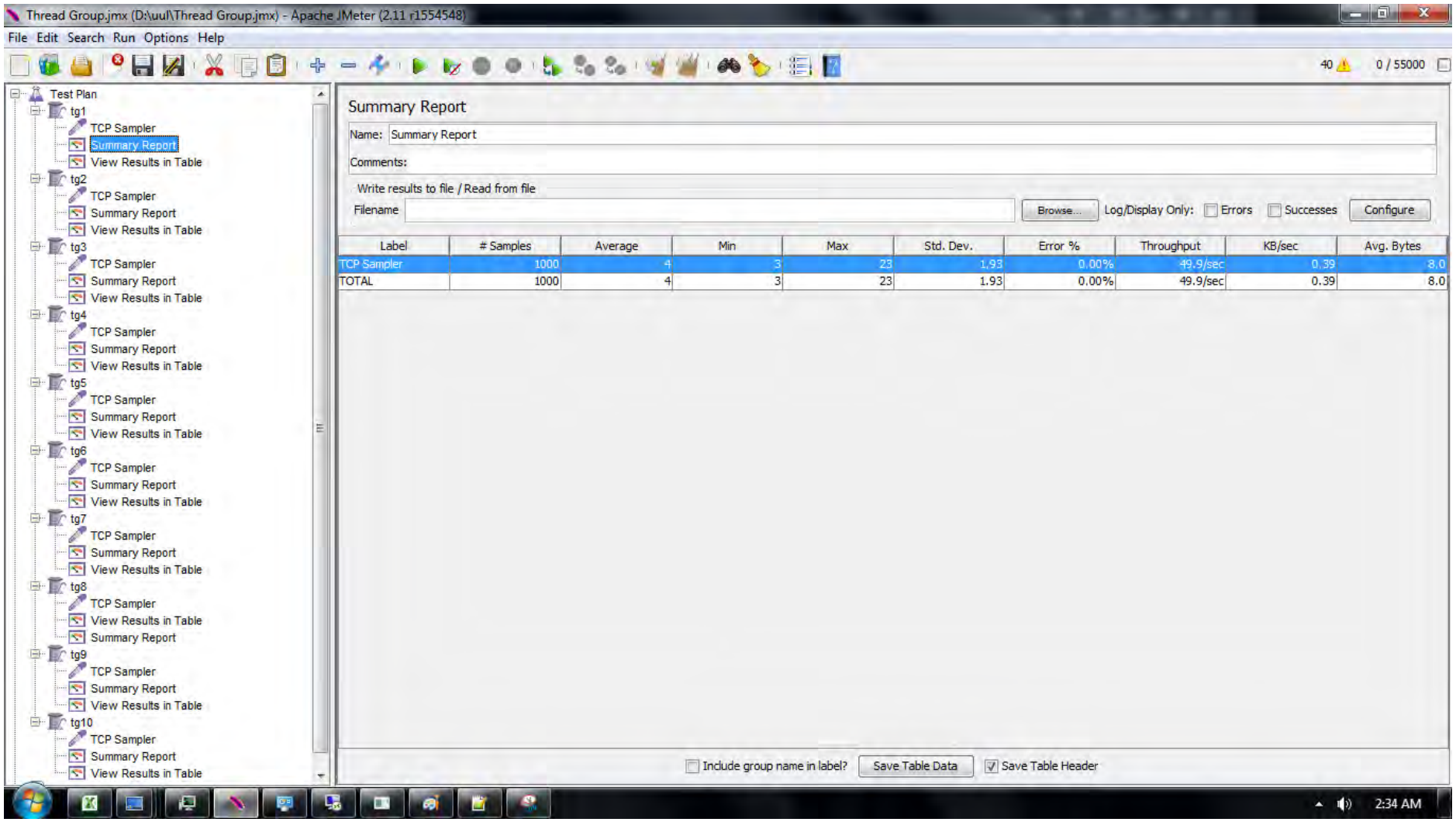
Gambar B. 5 Diagram Kelas Lapisan Listener

[Halaman ini sengaja dikosongkan]

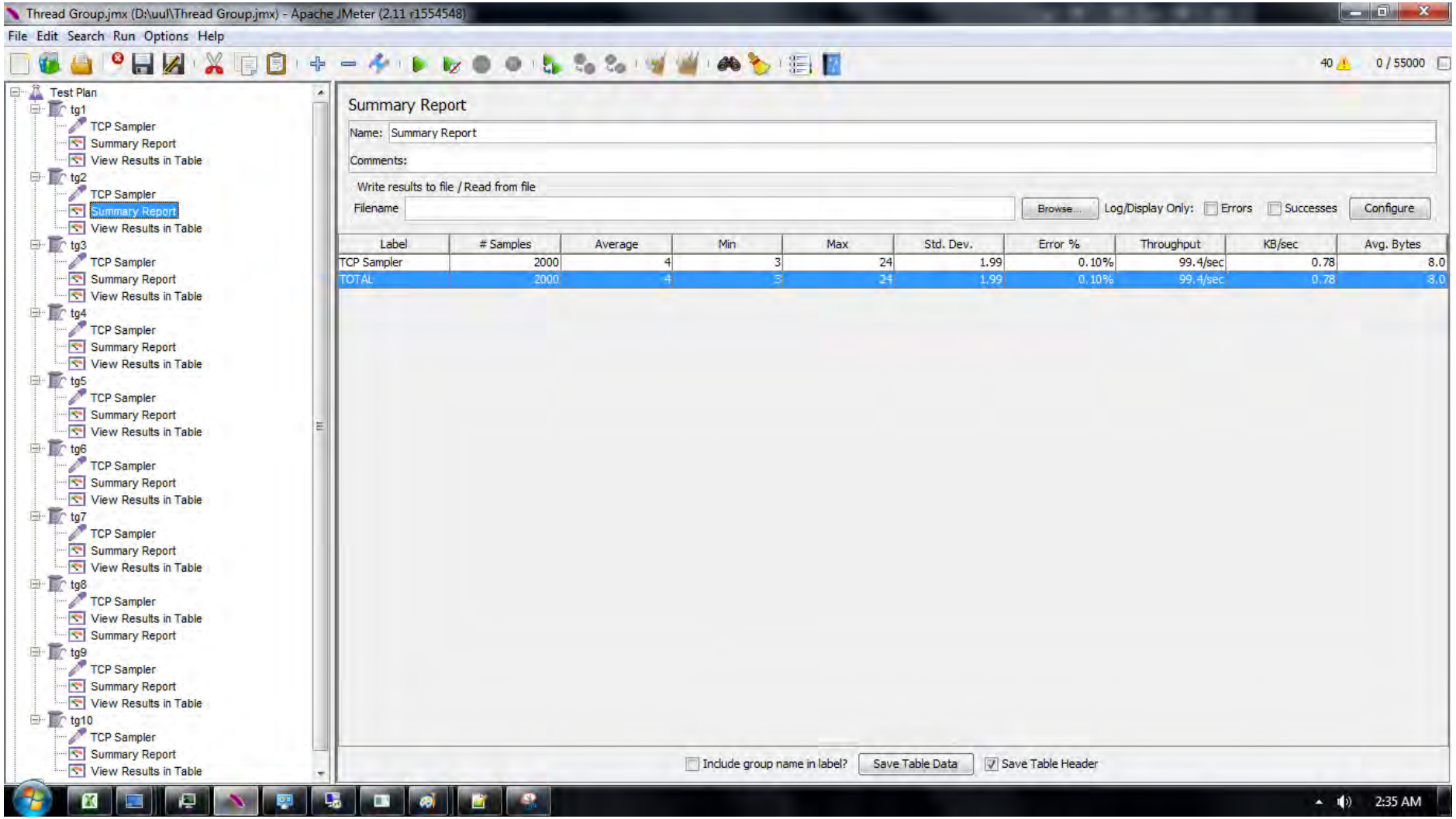


Gambar B. 6 Diagram Sekuen Pemain Lawan Pemain

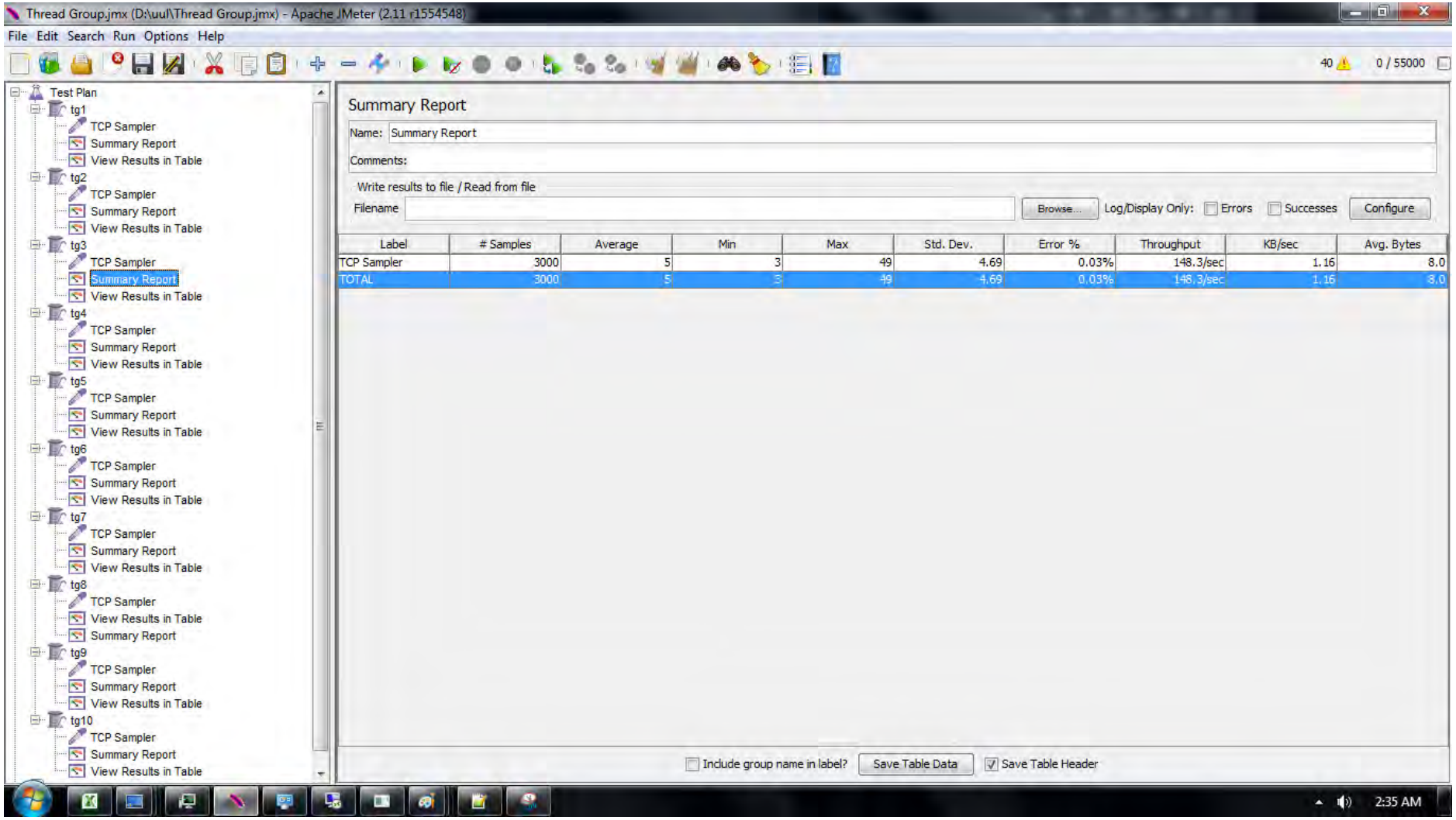
[Halaman ini sengaja dikosongkan]



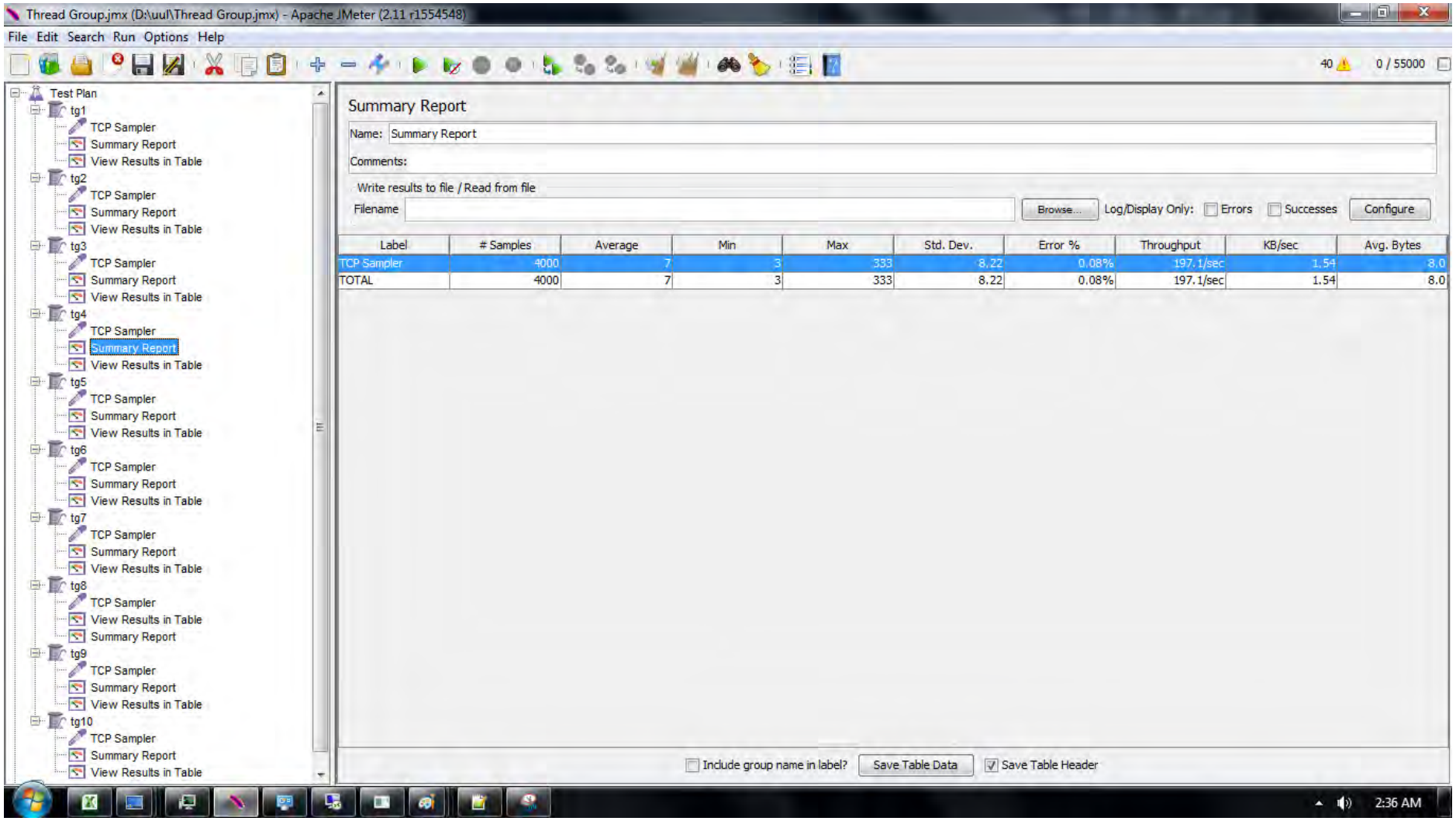
Gambar C. 1 Hasil Uji Sampai *Thread* Ke-1000 *Host* Pertama



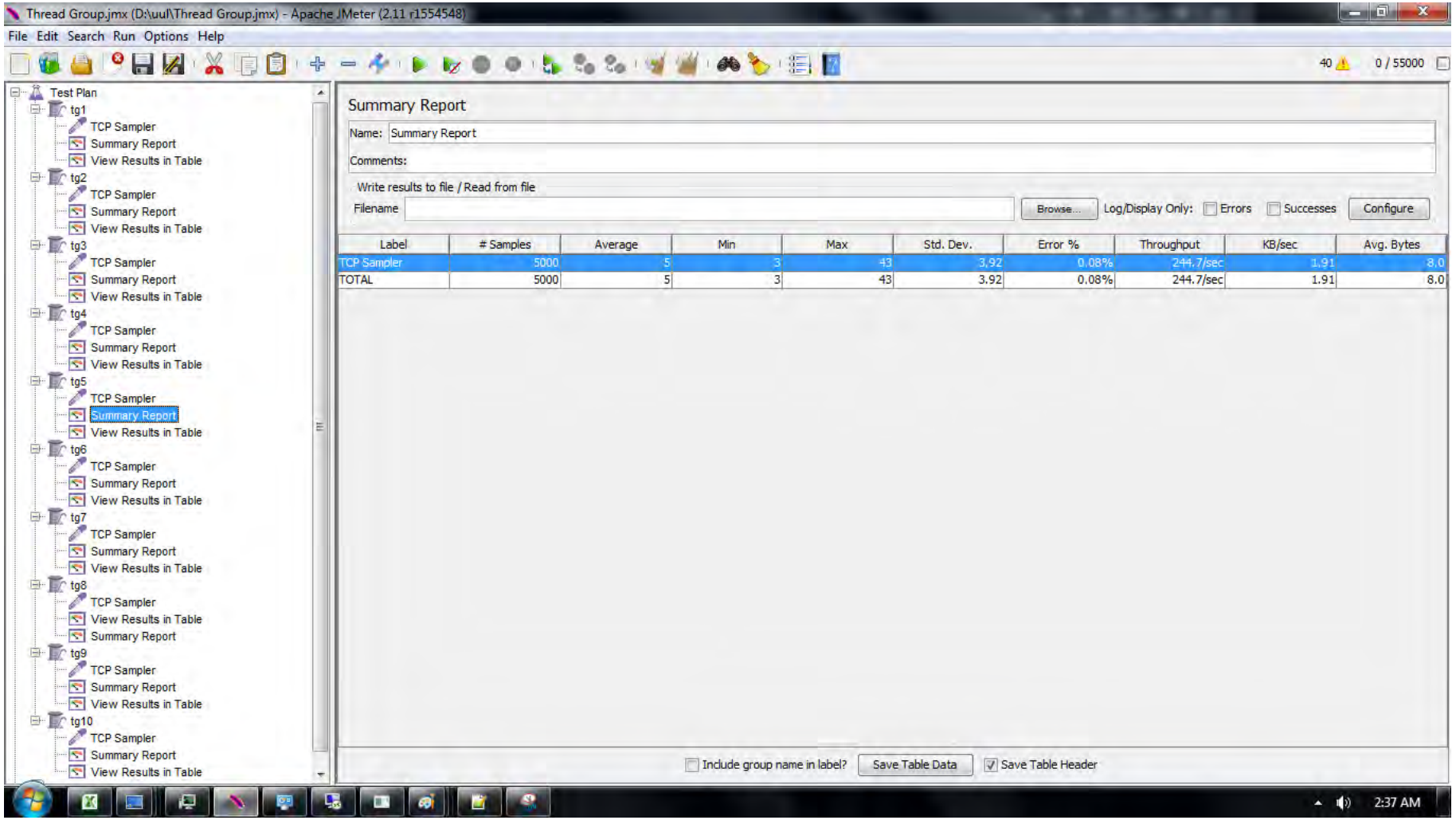
Gambar C. 2 Hasil Uji Sampai *Thread* Ke-2000 *Host* Pertama



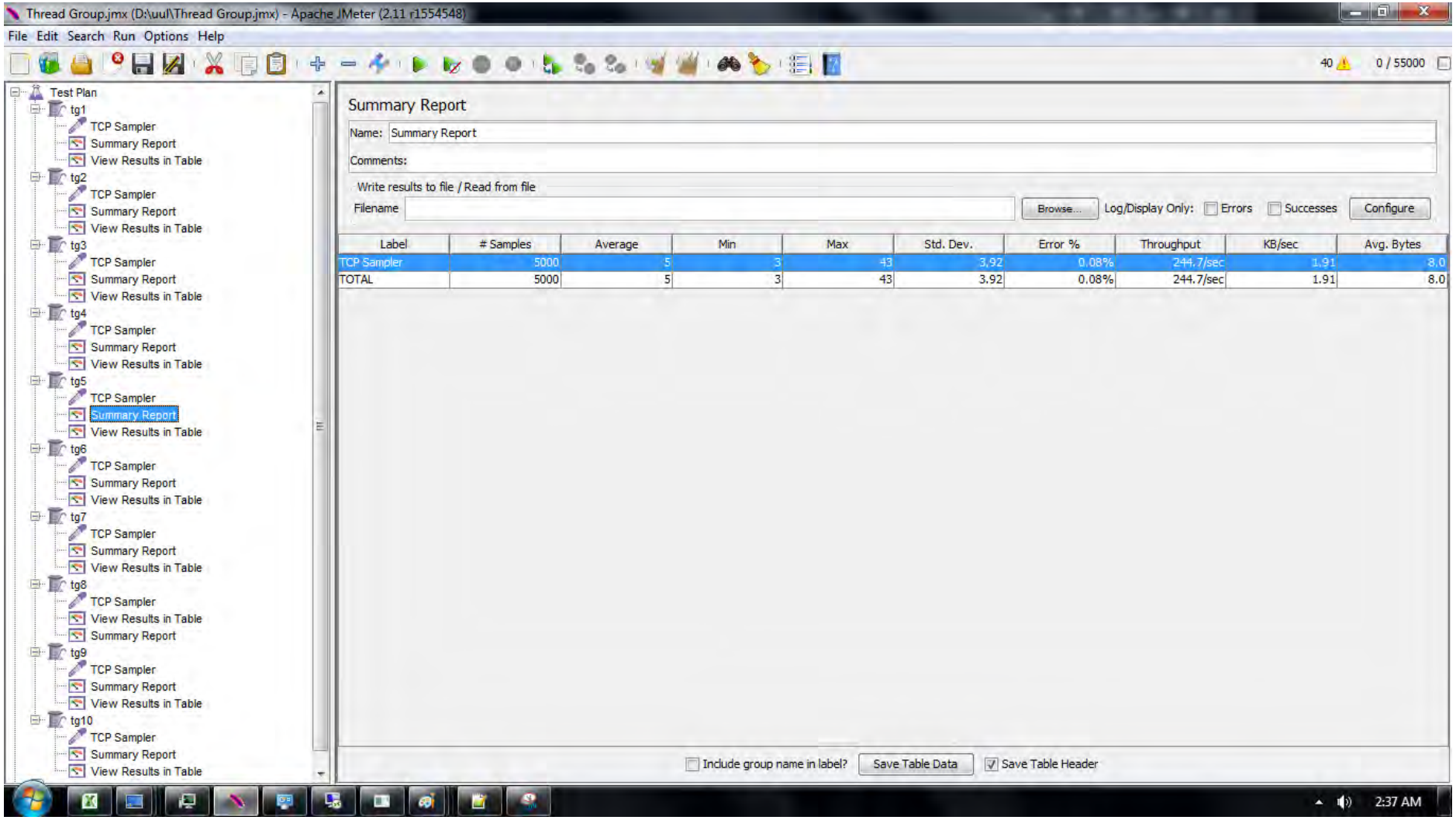
Gambar C. 3 Hasil Uji Sampai *Thread* Ke-3000 *Host* Pertama



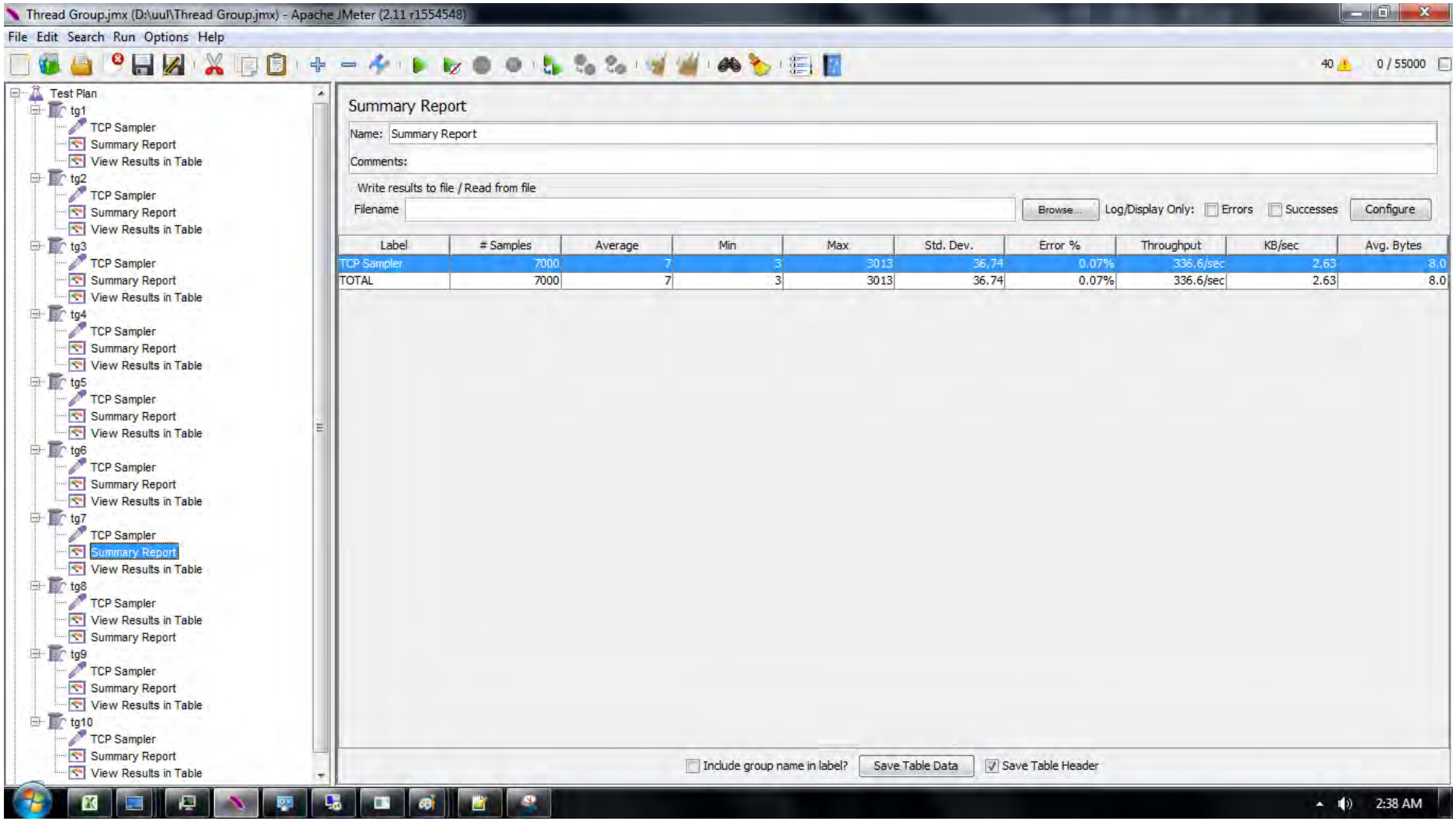
Gambar C. 4 Hasil Uji Sampai *Thread* Ke-4000 *Host* Pertama



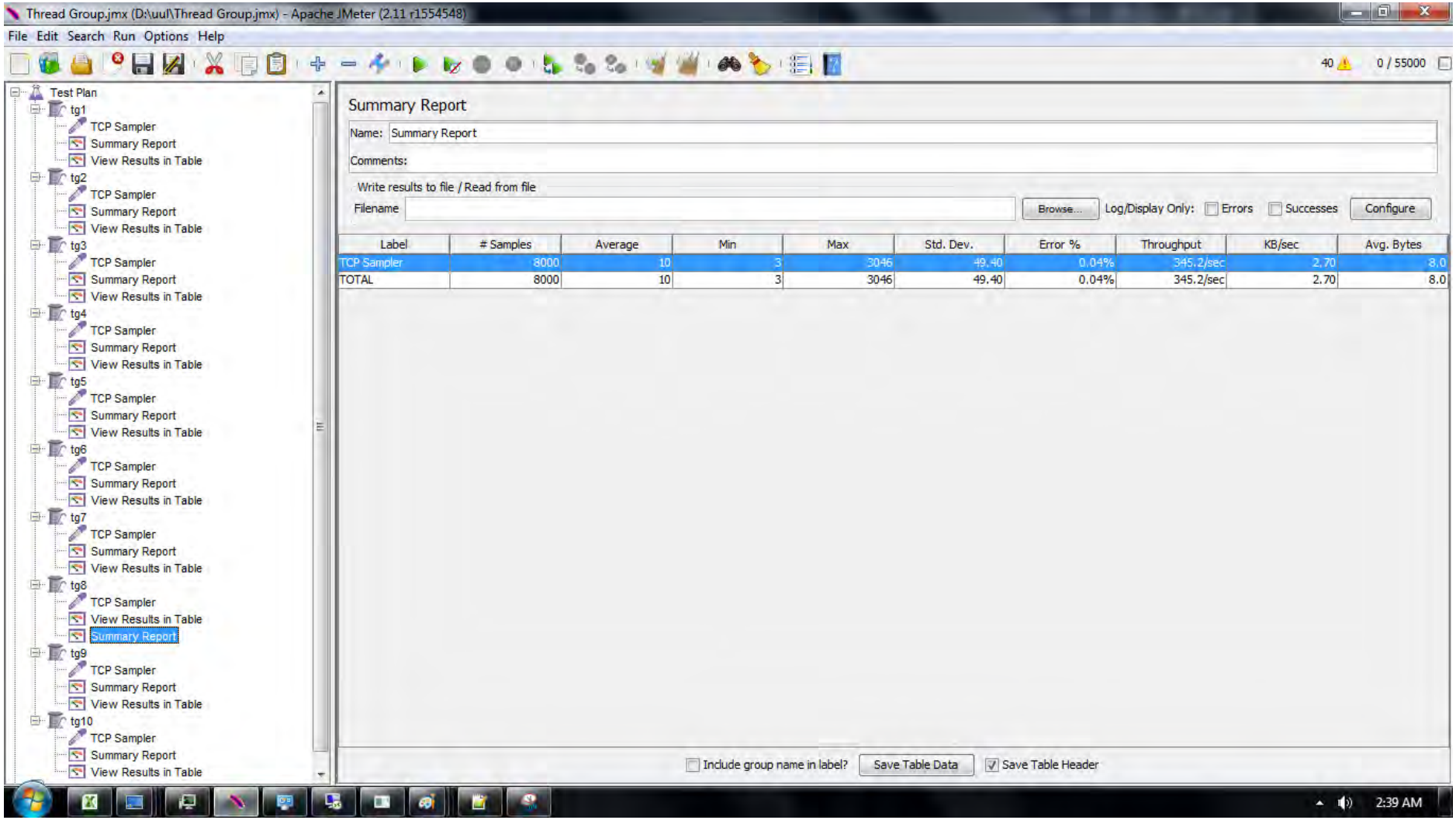
Gambar C. 5 Hasil Uji Sampai *Thread* Ke-5000 *Host* Pertama



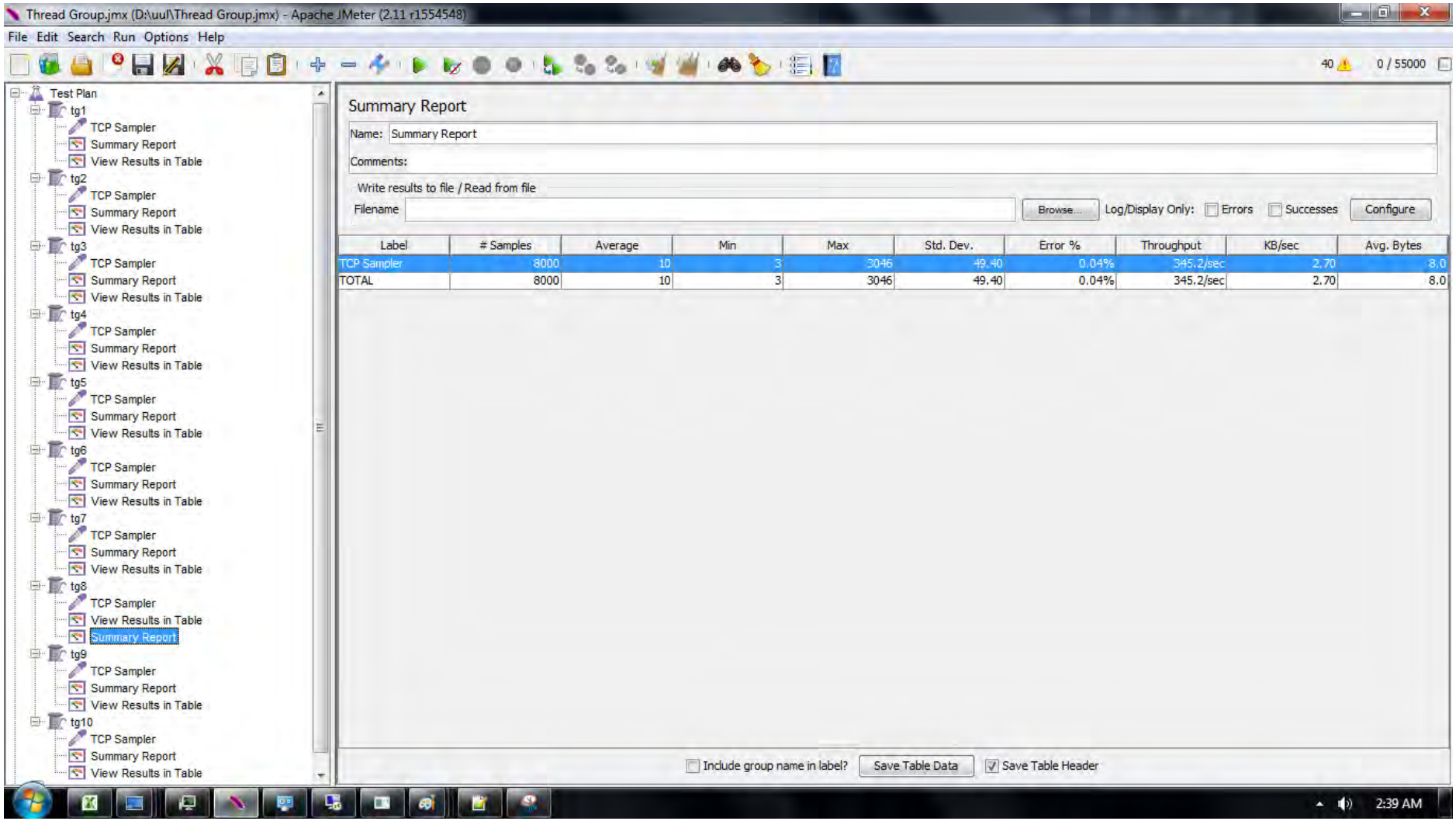
Gambar C. 6 Hasil Uji Sampai *Thread* Ke-6000 *Host* Pertama



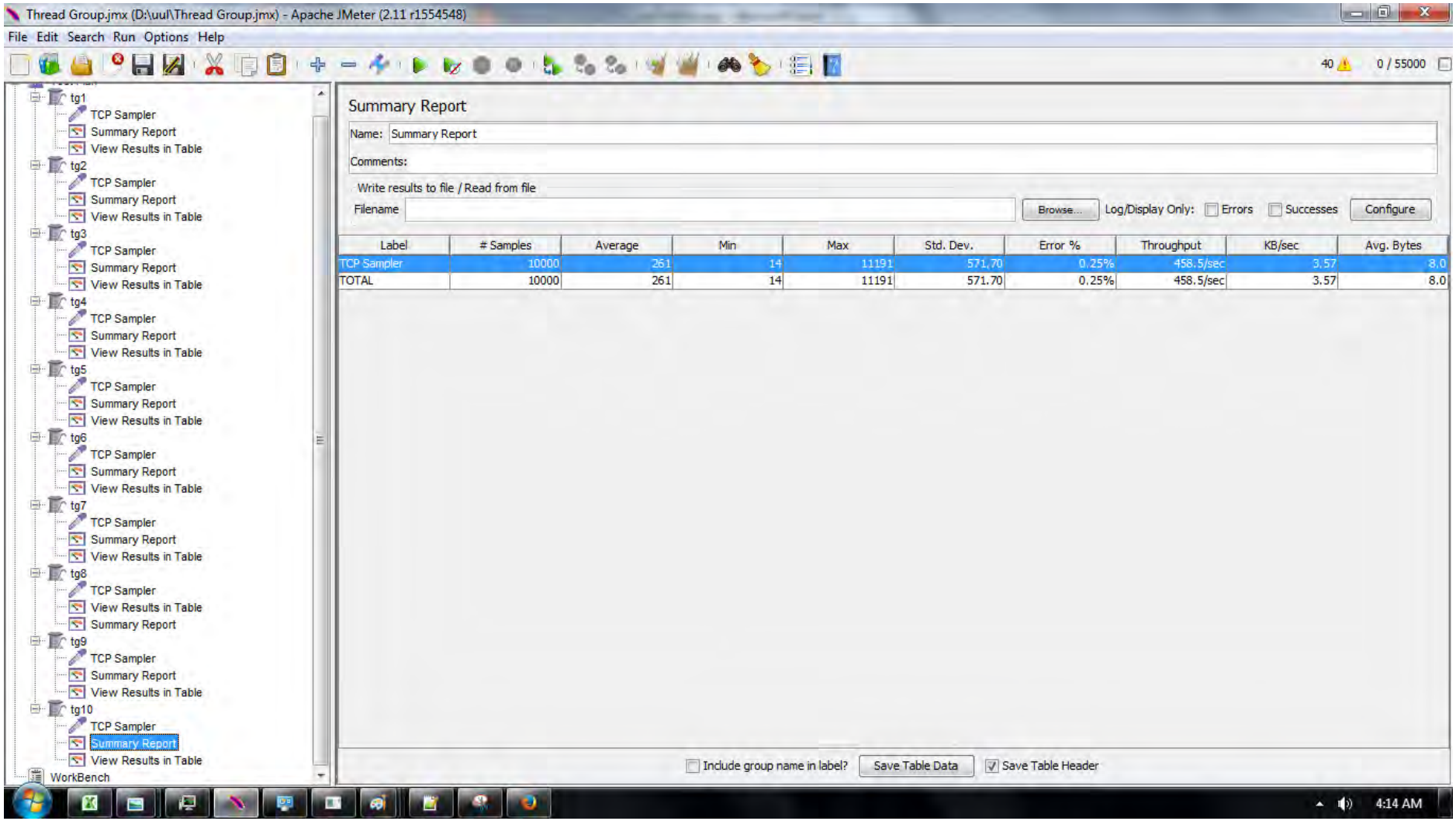
Gambar C. 7 Hasil Pengujian Sampai *Thread* Ke-7000 *Host* Pertama



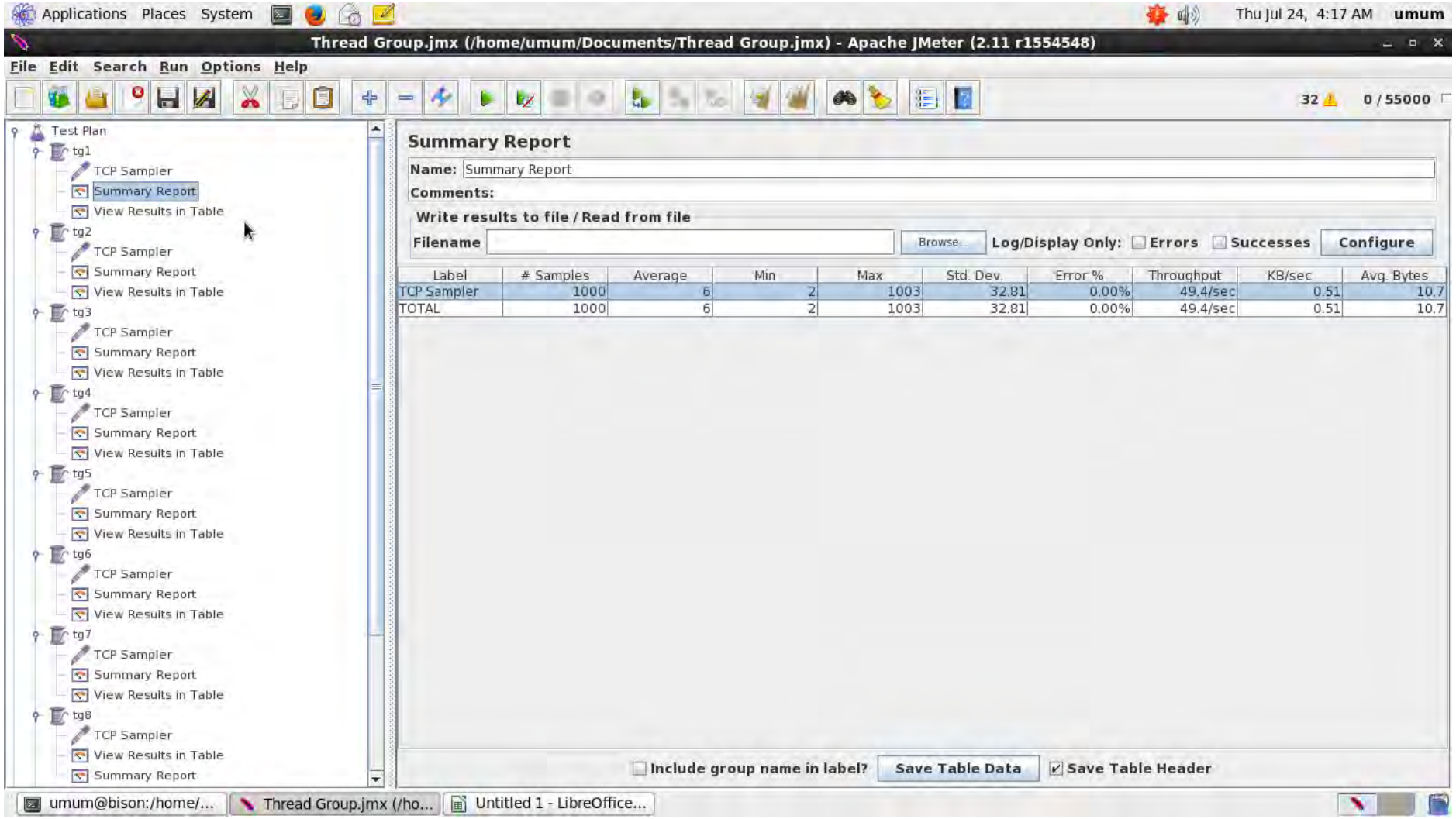
Gambar C. 8 Hasil Pengujian Sampai *Thread* Ke-8000 *Host* Pertama



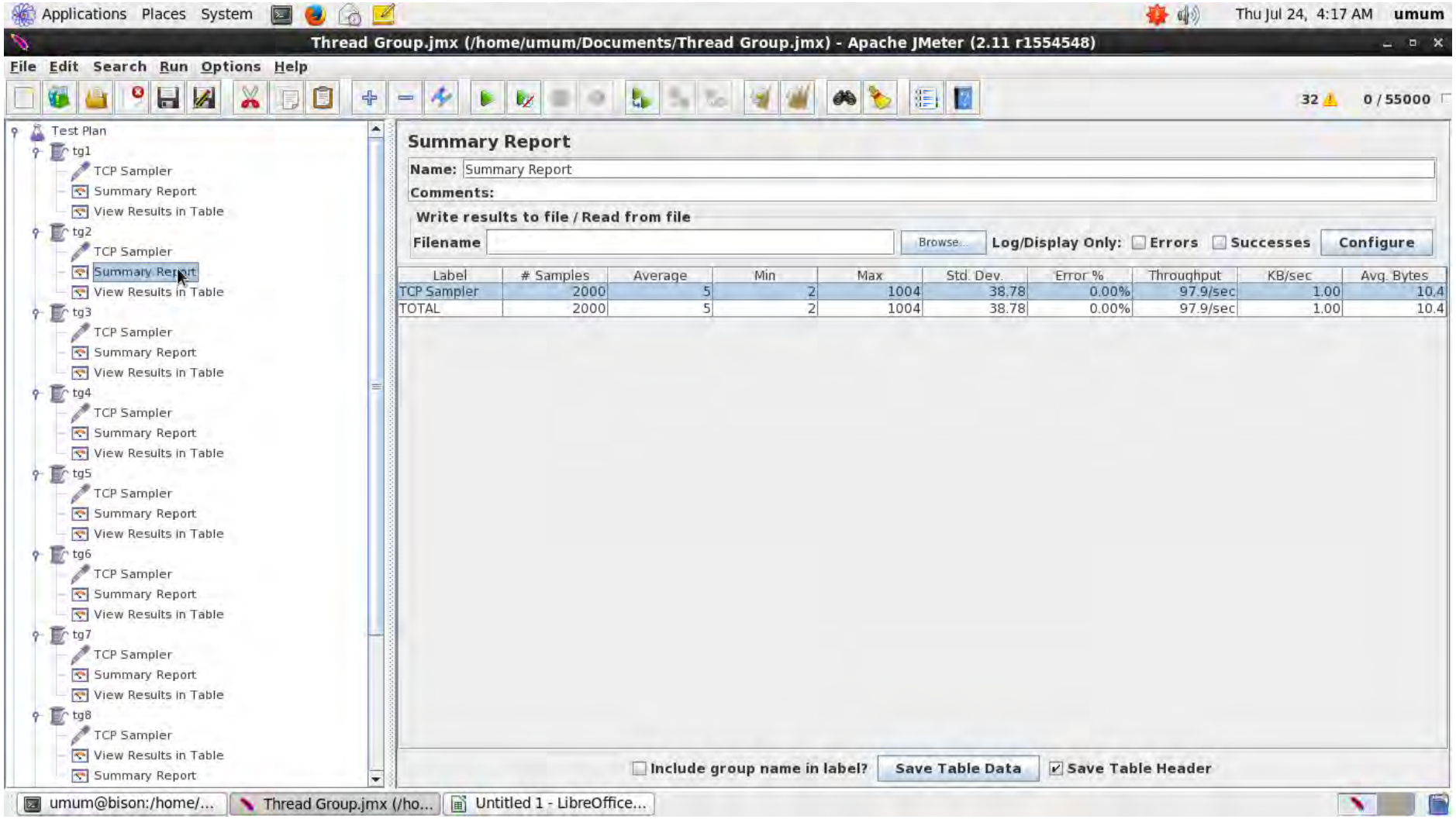
Gambar C. 9 Hasil Pengujian Sampai *Thread* Ke-9000 *Host* Pertama



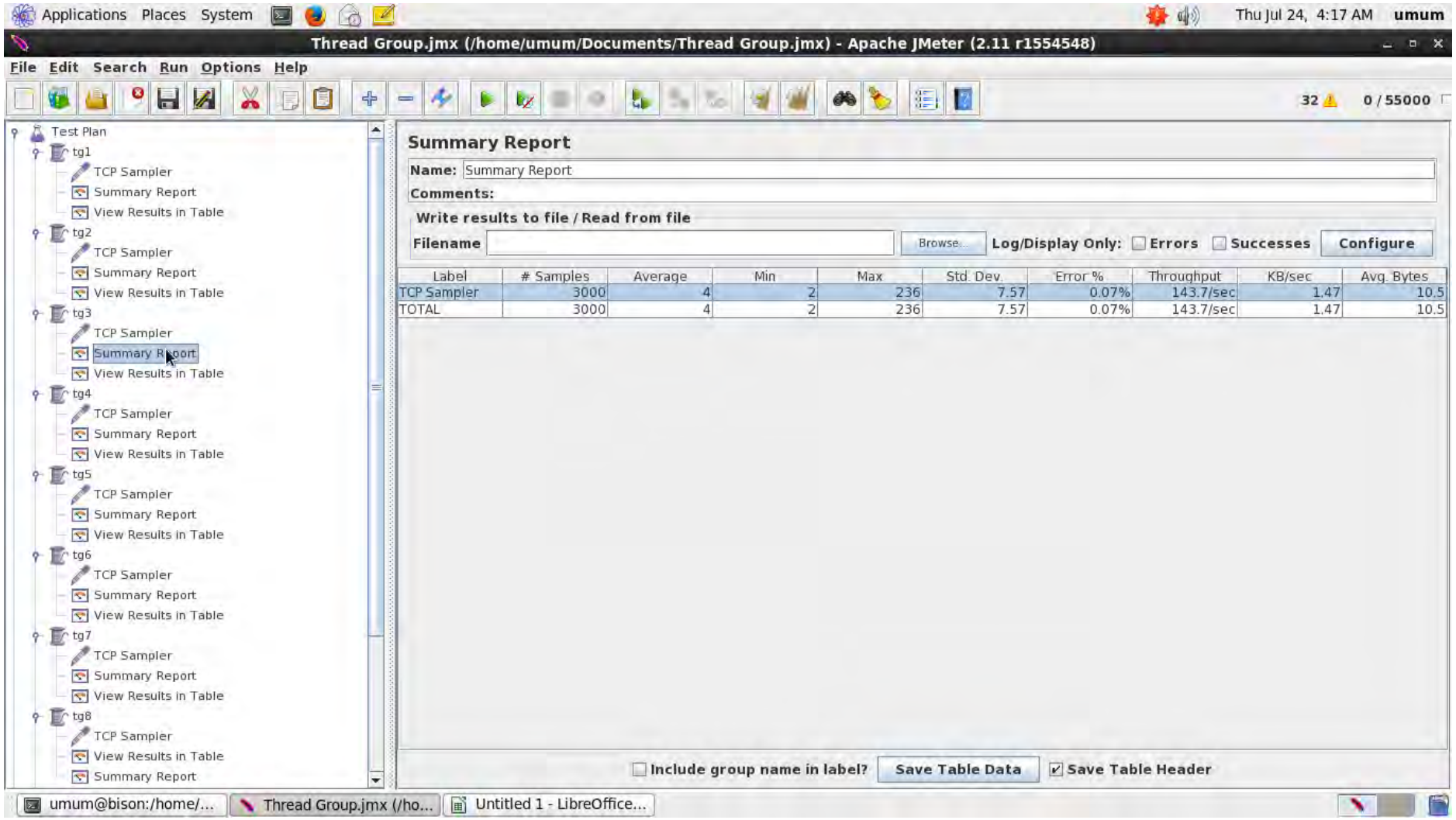
Gambar C. 10 Hasil Pengujian Sampai *Thread* Ke 10000 *Host* Pertama



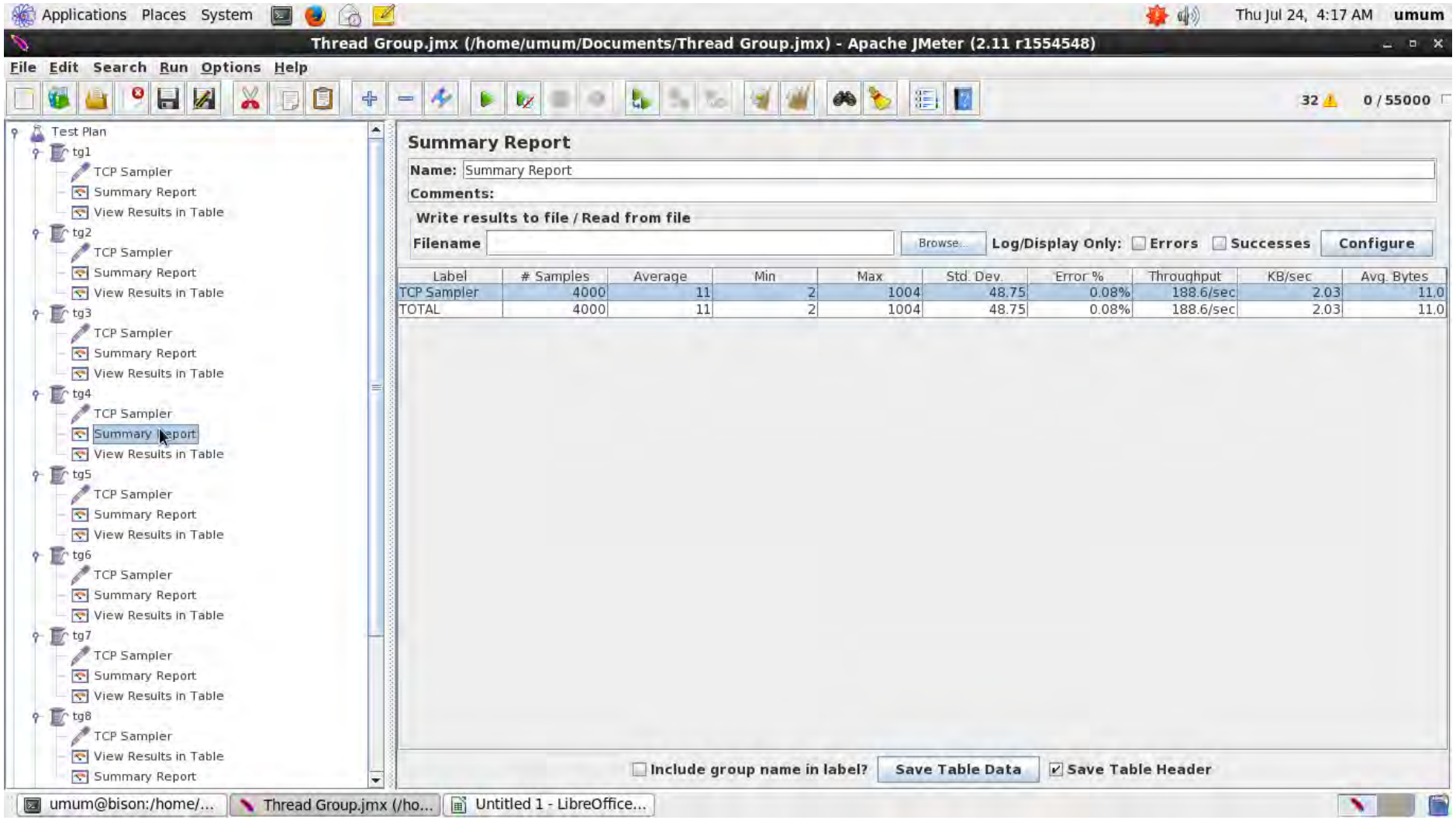
Gambar C. 11 Hasil Pengujian Sampai *Thread* Ke-1000 *Host* Kedua



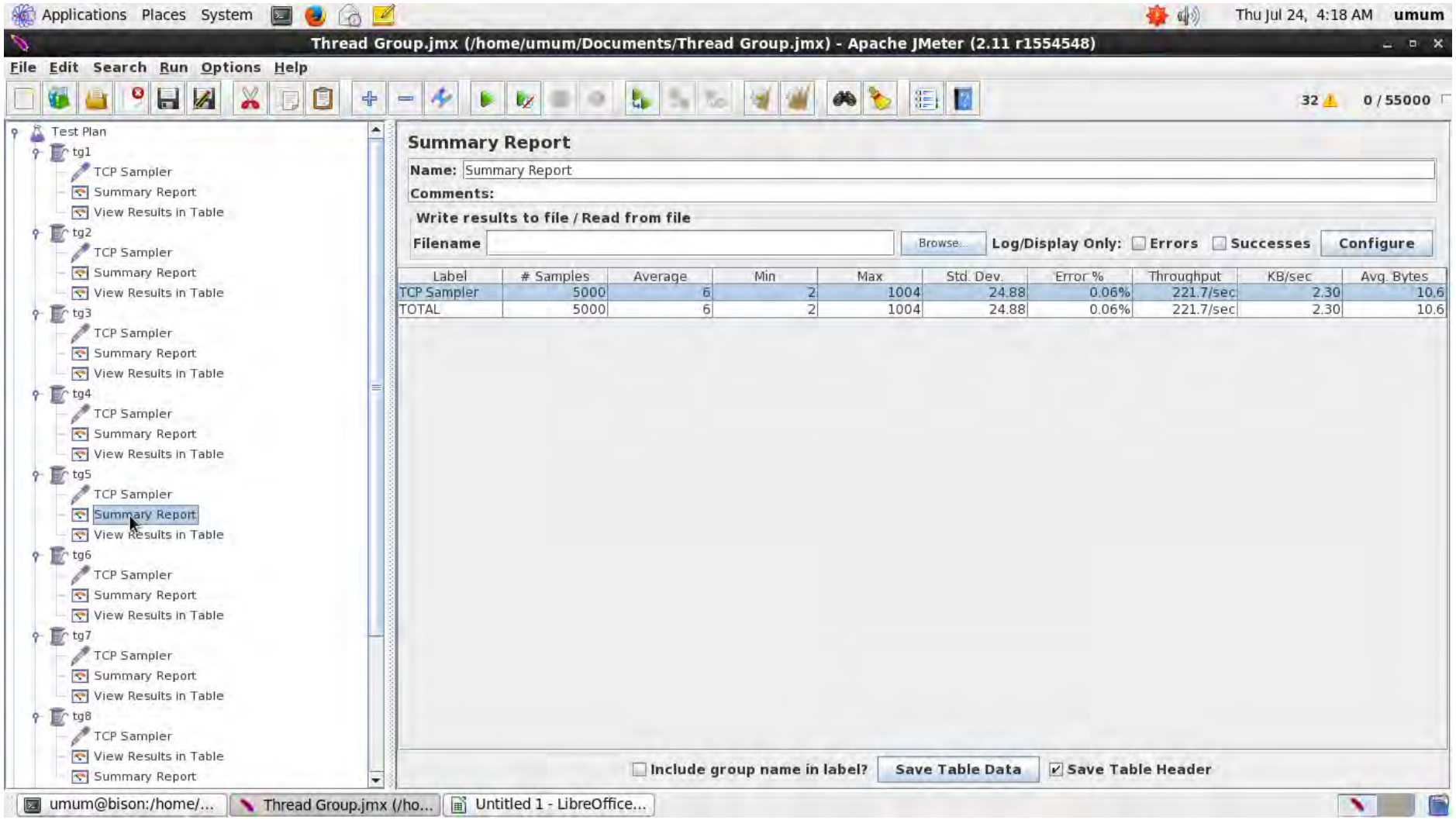
Gambar C. 12 Hasil Uji Sampai *Thread* Ke-2000 *Host* Kedua



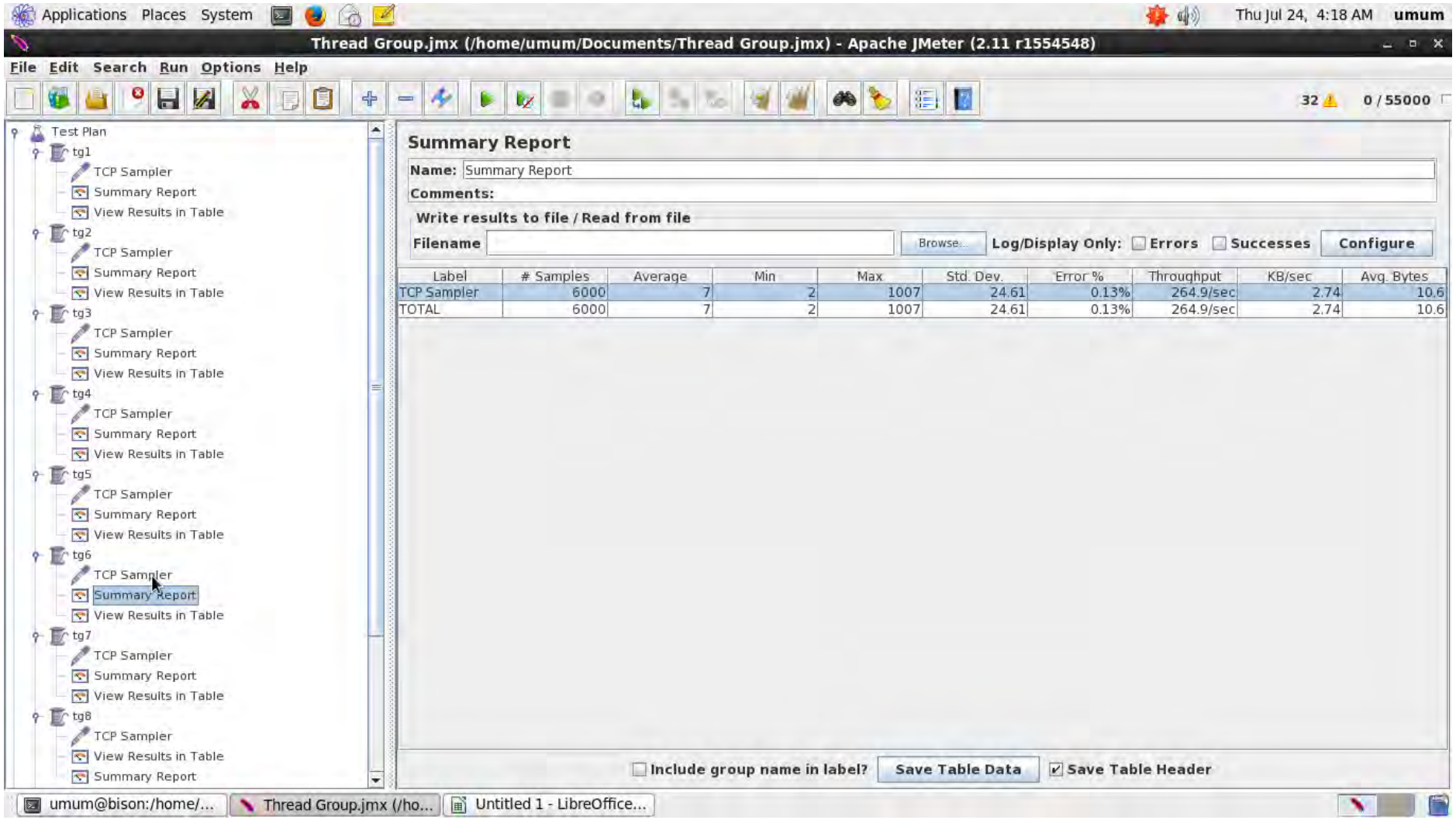
Gambar C. 13 Hasil Uji Sampai *Thread* Ke-3000 *Host* Kedua



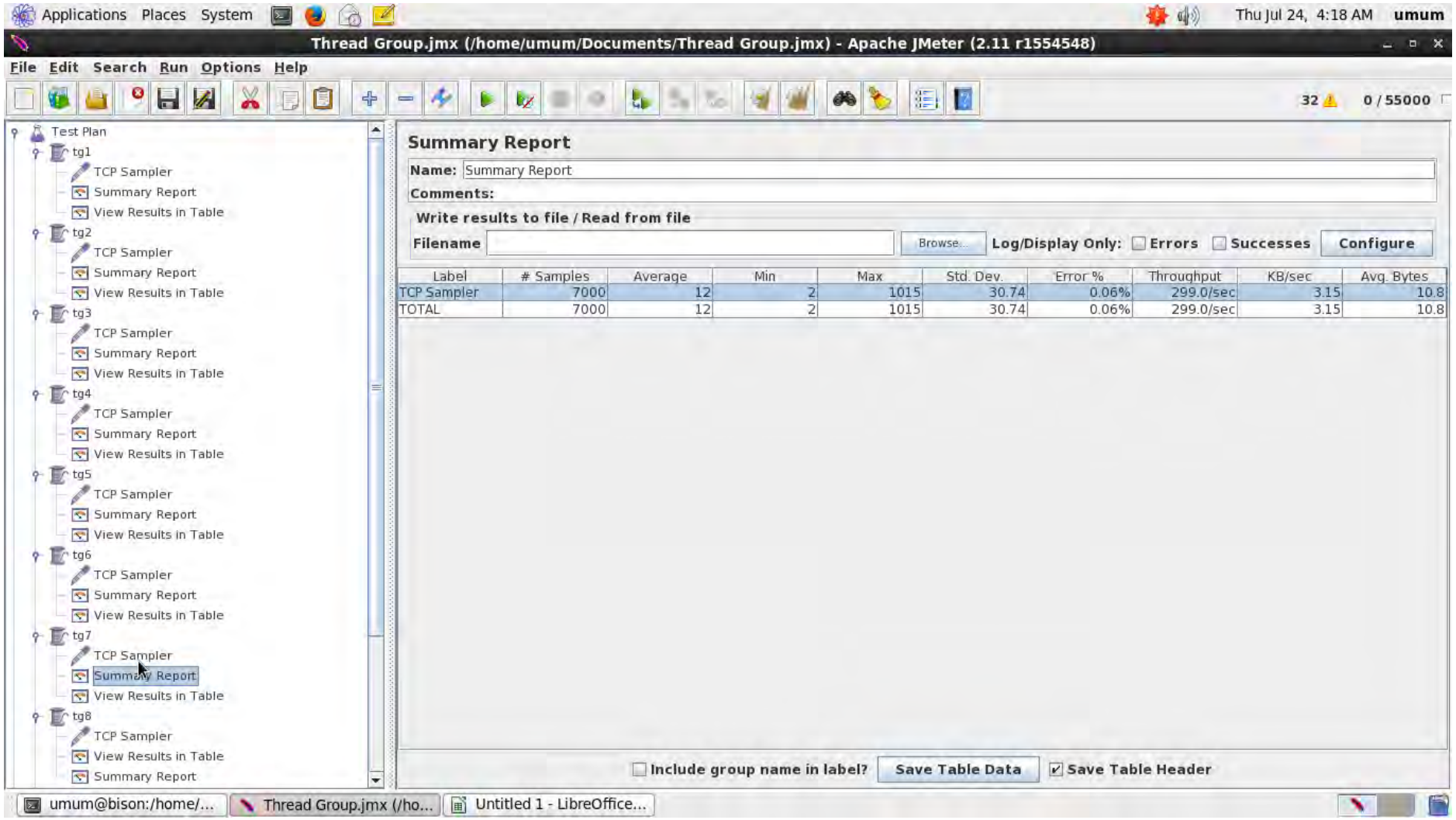
Gambar C. 14 Hasil Uji Sampai *Thread* Ke-4000 *Host* Kedua



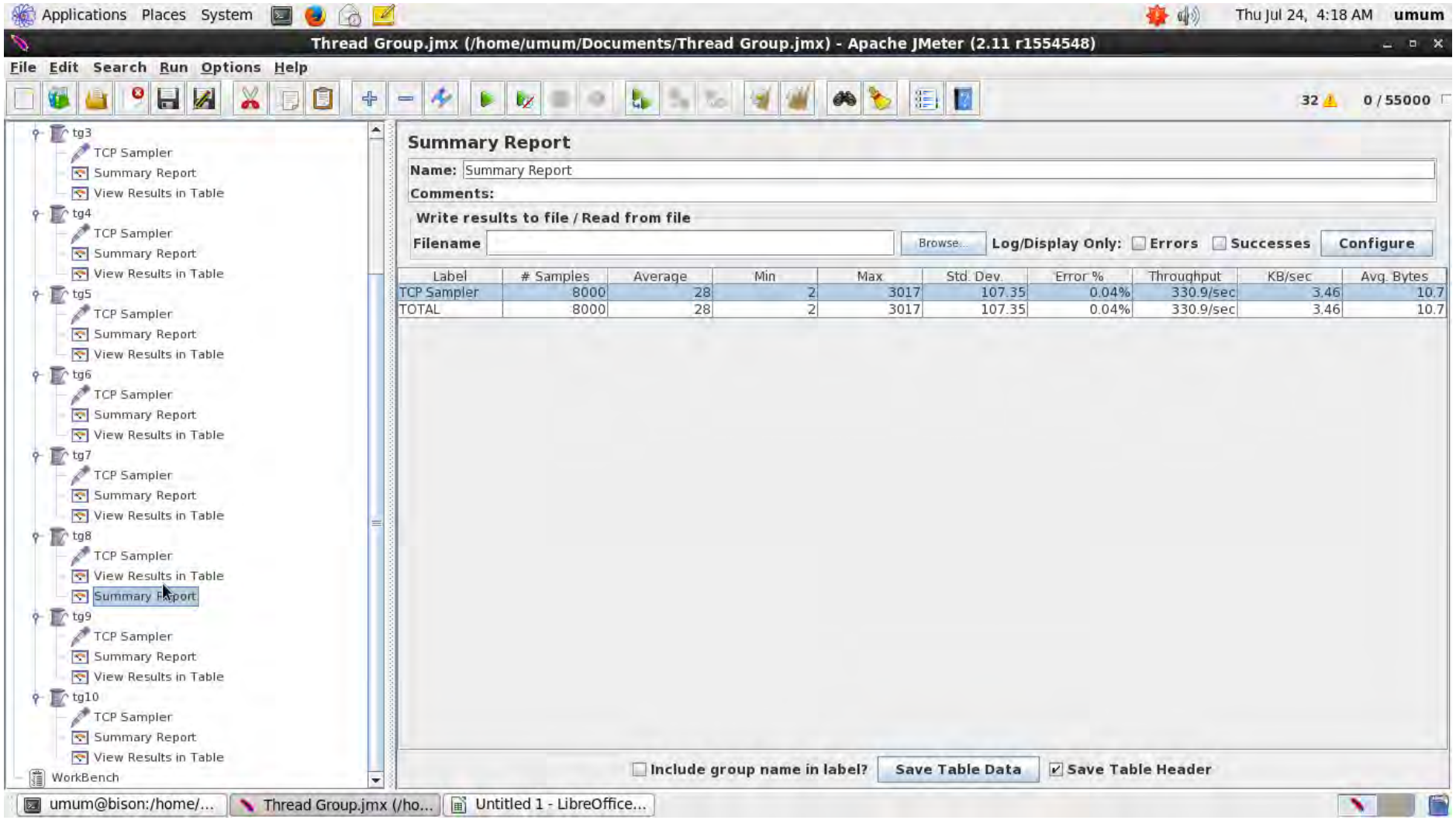
Gambar C. 15 Hasil Uji Sampai *Thread* Ke-5000 *Host* Kedua



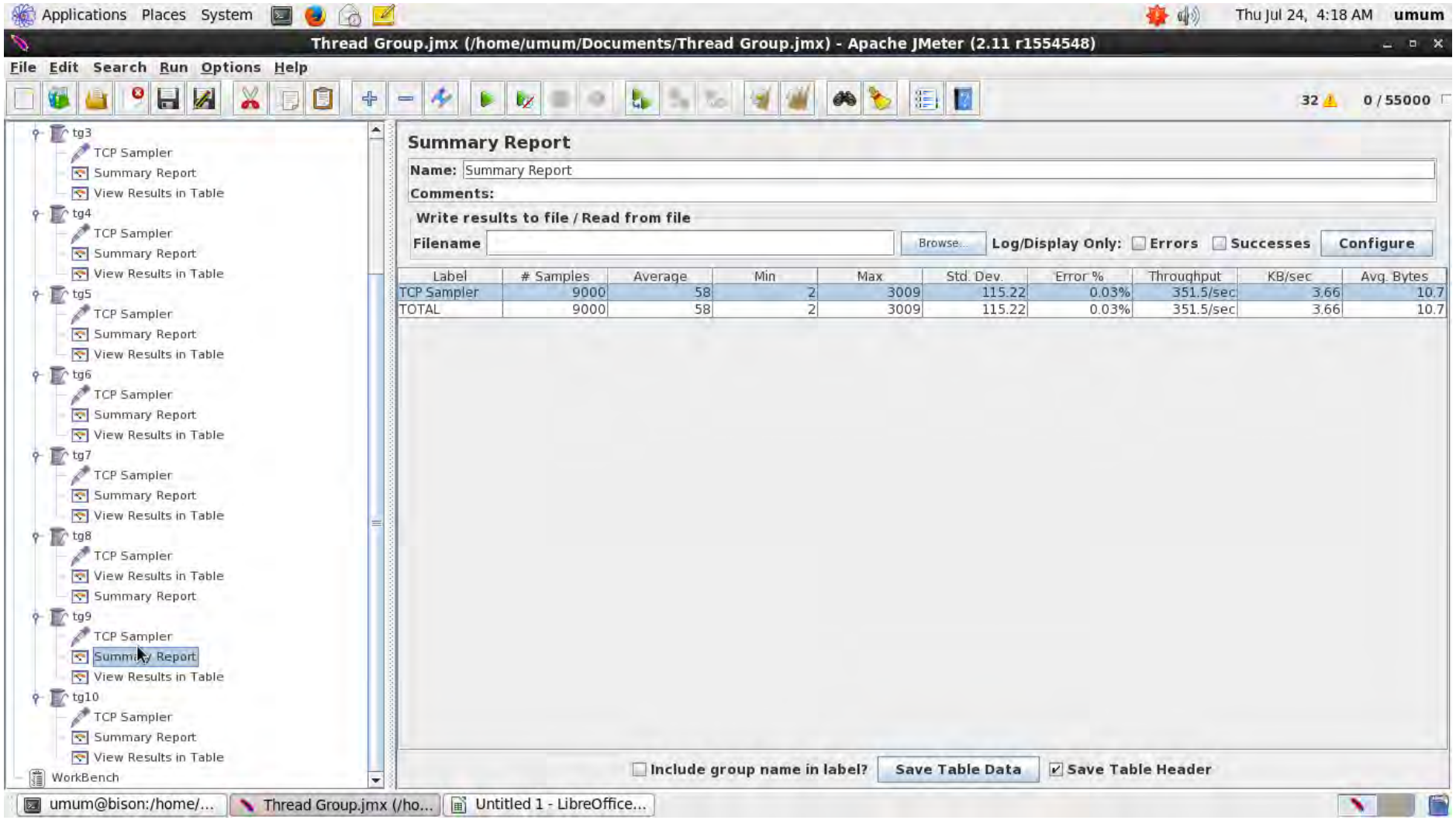
Gambar C. 16 Hasil Uji Sampai *Thread* Ke-6000 *Host* Kedua



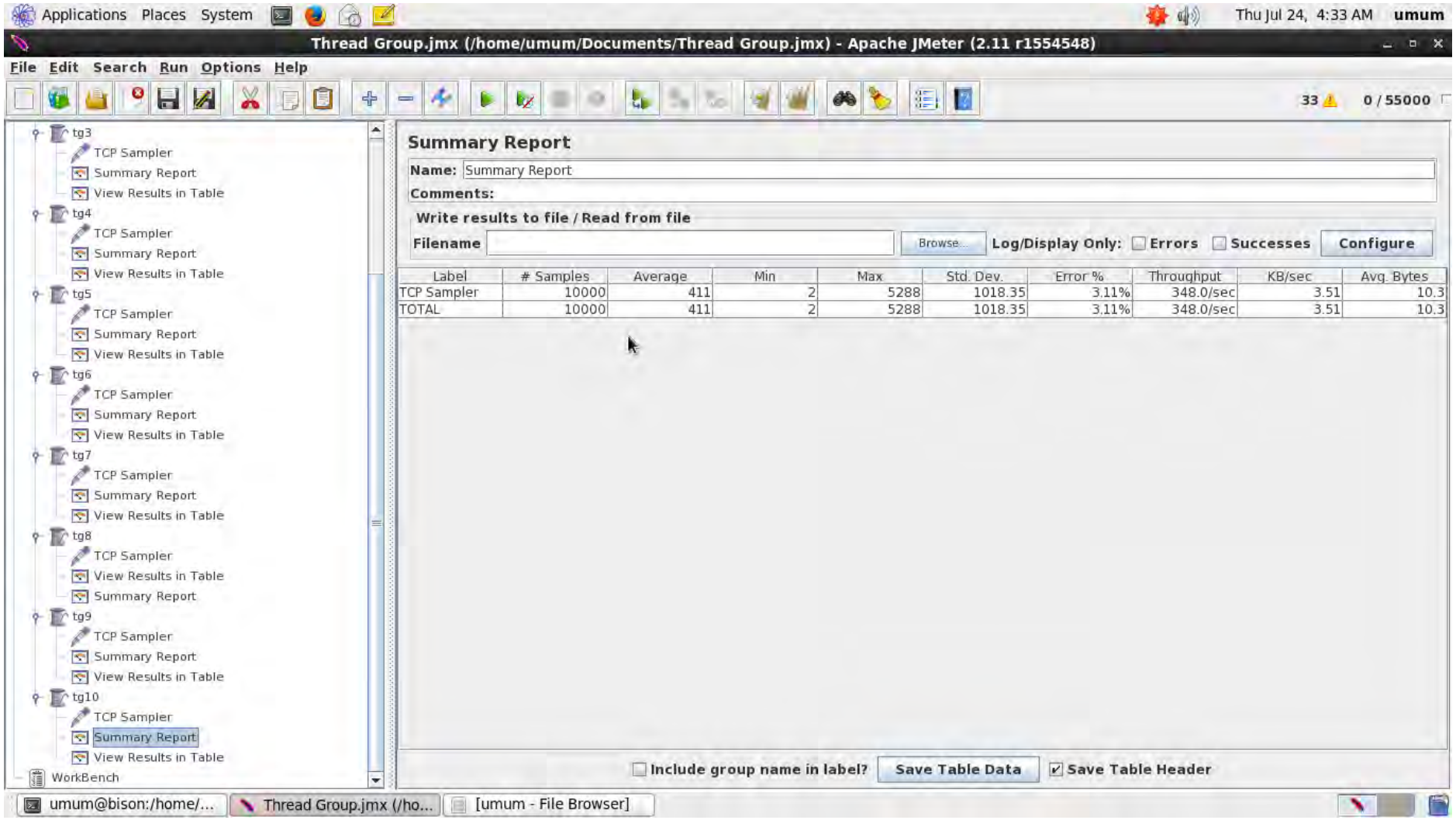
Gambar C. 17 Hasil Uji Sampai *Thread* Ke-7000 *Host* Kedua



Gambar C. 18 Hasil Uji Sampai *Thread* Ke-8000 *Host* Kedua



Gambar C. 19 Hasil Uji Sampai *Thread* Ke-9000 *Host* Kedua



Gambar C. 20 Hasil Uji Sampai *Thread* Ke-10000 *Host* Kedua