TUGAS AKHIR - KI141502

RANCANG BANGUN CLASS LIBRARY UNTUK IMPLEMENTASI ATURAN MAIN DARI SOCIAL GAME HEART MEISTER UNTUK PERANGKAT ANDROID DENGAN MENGGUNAKAN UNITY

Remy Giovanny Mangowal
NRP 5110100 177

Dosen Pembimbing
Imam Kuswardayan, S.Kom., M.T.
Abdul Munif, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember
Surabaya 2015

FINAL PROJECT - KI141502

CLASS LIBRARY DESIGN ARCHITECTURE FOR MAIN GAMEPLAY IMPLEMENTATION FROM HEART MEISTER SOCIAL GAME FOR ANDROID USING UNITY

Remy Giovanny Mangowal
NRP 5110100 177

Advisor
Imam Kuswardayan, S.Kom., M.T.
Abdul Munif, S.Kom., M.Sc.

DEPARTMENT OF INFORMATICS
Faculty of Information Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2015

LEMBAR PENGESAHAN

Rancang Bangun *Class Library* untuk Implementasi Aturan Main dari *Social Game Heart Meister* untuk Perangkat Android dengan Menggunakan Unity

TUGAS AKHIR

Diajukan Guna Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Interaksi Grafis dan Seni
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh :
REMY GIOVANNY MANGOWAL
NRP : 5110 100 177

Disetujui oleh Dosen Pembimbing tugas akhir :

IMAM KUSWARDAYAN, S.Kom., M.T. .....
NIP: 197411232006041001 (pembimbing 1)

ABDUL MUNIF, S.Kom., M.Sc. .....
NIP: 5100201301005 (pembimbing 2)

SURABAYA
JANUARI 2015

Rancang Bangun *Class Library* untuk Implementasi Aturan Main dari *Social Game Heart Meister* untuk Perangkat Android dengan Menggunakan Unity

Nama Mahasiswa	: Remy Giovanny Mangowal
NRP	: 5110100177
Jurusan	: Teknik Informatika FTIf-ITS
Dosen Pembimbing 1	: Imam Kuswardayan, S.Kom., M.T.
Dosen Pembimbing 2	: Abdul Munif, S.Kom, M.Sc.

ABSTRAK

Perangkat mobile sudah menjadi bagian keseharian sebagian besar penduduk. Berbagai macam perangkat lunak pun bermunculan untuk memanfaatkan kesempatan tersebut, termasuk perangkat lunak berbentuk permainan. Untuk mempermudah pengembangan permainan pada perangkat mobile salah satu caranya adalah dengan menggunakan class library dalam pengembangan permainan tersebut.

Class library dapat membantu penyeragaman modul-modul yang berbeda sehingga dapat membantu pengimplementasian aturan main dari permainan ke modul-modul yang berbeda. Class library diuji dengan menggunakan pengujian fungsionalitas untuk menguji seluruh fungsi yang ada.

Seluruh fungsi dapat dijalankan dengan baik, namun proses integrasi modul web service dan sosial ke class library mengalami kesulitan karena Unity mengharuskan beberapa komponen class library tersambung dengan objek Unity. Hal ini membuat class library tidak dapat dipisah menjadi independen dari Unity sehingga modul web service dan sosial tidak dapat menggunakan class library dengan sempurna. Modul epic battle dapat mengimplementasikan fitur transisi dan aturan main pertarungan dari class library karena modul tersebut juga berbasis Unity sehingga tidak mengharuskan pemisahan class library.

Kata Kunci**: **Class Library, C#, Game, Social Game, Unity.

Class Library Design Architecture for Main Gameplay Implementation from Heart Meister Social Game for Android Using Unity

Student Name	: Remy Giovanny Mangowal
Student ID	: 5110100177
Major	: Teknik Informatika FTIf-ITS
Advisor 1	: Imam Kuswardayan, S.Kom., M.T.
Advisor 2	: Abdul Munif, S.Kom, M.Sc.

ABSTRACT

Mobile device has become a part of the mass' daily lives. A lot of softwares appeared to use that chance, including game softwares. To ease the development of games in mobile devices, one of the solution is to use class library in the development of the game.

Class library can help to uniform the different modules, so it can help the implementation of the main gameplay to the different modules. Class library is tested with functionality testing to test all the available functions.

All functions can be run smoothly, but the integration process from the web service module and the social module experienced some difficulties because Unity needs to have some class library components attached to the Unity objects. This makes the class library inseparable into an independent entity from Unity so that the web service module and the social module cannot use the class library perfectly. The epic battle module can implement the transition feature and the battle gameplay from the class library because that module is also Unity-based, hence needing no separation from the class library.

Keyword**: **Class Library, C#, Game, Social Game, Unity.

KATA PENGANTAR

Penulis memanjatkan puji dan syukur kepada Tuhan Yang Maha Esa karena berkat segala rahmat-Nya penulis dapat menyelesaikan tugas akhir yang berjudul:

"Rancang Bangun *Class Library* untuk Implementasi Aturan Main dari *Social Game Heart Meister* untuk Perangkat Android dengan Menggunakan Unity"

Semoga apa yang tertulis dalam buku tugas akhir ini dapat memberikan manfaat kepada pengembangan sejenis nantinya.

Penulis ingin mengucapkan terima kasih kepada:

1. Ayah Nico, Ibu Indri, dan Kak Stella karena penulis hanya dapat mencapai jenjang sejauh ini dengan bantuan penuh kasih selama ini.
2. Bapak Imam Kuswardayan dan Bapak Abdul Munif selaku dosen pembimbing yang telah sangat membantu dalam berbagai aspek pengerjaan tugas akhir ini, baik berupa saran, masukan, tambahan, dan nasihat.
3. Bapak, Ibu dosen Jurusan Teknik Informatika ITS yang telah membimbing penulis selama masa studi di Teknik Informatika ITS.
4. Staf dan karyawan Teknik Informatika ITS yang telah membantu penulis dari sisi administrasi akademik.
5. Tim teman-teman yang tergabung dalam proyek SQS-ITS Project dan teman-teman kelas E yang selalu memberikan bantuan disaat diperlukan.
6. Angkatan 2010 Teknik Informatika ITS yang berjuang bersama selama ini.
7. Dan pihak-pihak lain yang turut membantu dalam pengerjaan tugas akhir ini.

Surabaya, Januari 2015

Remy Giovanny Mangowal

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

BIODATA PENULIS

Remy Giovanny Mangowal, lahir di Jakarta pada tanggal 13 Desember 1992, merupakan anak kedua dari dua bersaudara. Penulis menempuh pendidikan dasar mulai kelas 1 hingga kelas 3 di SDK Kristoforus, Jakarta. Dilanjutkan dengan pendidikan dasar mulai kelas 4 hingga kelas 6 di SDK Santa Maria, Banjarmasin. Penulis melanjutkan pendidikan menengah di SMPK Santa Maria, Banjarmasin. Kemudian penulis melanjutkan pendidikan di SMAN 1 Banjamarsin. Selanjutnya penulis melanjutkan pendidikan sarjana di Jurusan Teknik Informatika, Institut Teknologi Sepuluh Nopember Surabaya.

Di Jurusan Teknik Informatika, penulis mengambil bidang minat Interaksi Grafik dan Seni (IGS) . Penulis dapat dihubungi melalui alamat email remy.mangowal@yahoo.co.id

BAB I
PENDAHULUAN

Pada bab ini akan dipaparkan mengenai garis besar tugas akhir yang meliputi latar belakang, tujuan, rumusan dan batasan permasalahan, metodologi pembuatan tugas akhir, dan sistematika penulisan.

1.1. Latar Belakang

Maraknya penggunaan perangkat *mobile* seperti *smartphone* membuka peluang yang sangat besar bagi perkembangan perangkat lunak di bidang *mobile*. Salah satunya adalah perangkat lunak berbentuk *game*, terutama *social game* yang sangat cocok untuk diaplikasikan pada perangkat tersebut, misalnya Android. Pada perangkat ini, baik pemain kasual maupun pemain yang menekuni *game* dapat saling berinteraksi dengan mudah, sehingga membuka pasar pemain yang luas. Banyaknya pembajakan aplikasi *game* berbasis *offline* juga memberi kesempatan keuntungan yang lebih besar bagi *game* berbasis *online*. Ini dikarenakan *game* berbasis *online* seperti *social game* lebih mengutamakan keuntungan sedikit demi sedikit lewat *microtransactions* daripada penjualan *game* itu sendiri. Kecocokan dari *social game* dengan perangkat *mobile* itu sendiri disebabkan oleh kemudahan integrasi perangkat *mobile* dengan fitur-fitur jejaring sosial seperti Facebook, Twitter, dan sejenisnya, yang dapat diintegrasikan lebih lanjut untuk menunjang *gameplay* dari *social game* tersebut.

Pemrograman dalam pengembangan *game* tersebut dapat dipermudah dengan pemakaian *class library*. *Class library* adalah kumpulan dari kelas-kelas yang memiliki daya pemakaian ulang tinggi, dan sangat baik untuk mengurangi pengerjaan dan tingkat redundansi dari sebuah aplikasi. Pada tugas akhir ini sebuah *class library* yang digunakan oleh *social game* untuk perangkat Android yang menggunakan bahasa pemrograman C# telah dibuat.

1.2. Tujuan

Tujuan dari pembuatan tugas akhir ini adalah membuat *class library* yang dapat digunakan oleh *social game Heart Meister*.

1.3. Rumusan Permasalahan

Rumusan masalah dari tugas akhir ini adalah sebagai berikut.

1. Bagaimana membuat *class library* yang memuat implementasi aturan main yang dapat dipakai oleh seluruh permainan?
2. Bagaimana mengintegrasikan modul-modul yang ada pada *game*?

1.4. Batasan Permasalahan

Batasan masalah dari tugas akhir ini adalah permainan dibuat dengan menggunakan bahasa pemrograman C#.

1.5. Metodologi

Langkah-langkah yang ditempuh dalam pengerjaan tugas akhir ini yaitu:

1. **Studi literatur**

   Pada tahap ini dilakukan pengumpulan dan penggalian informasi dan literatur yang diperlukan dalam proses perancangan dan implementasi sistem yang akan dibangun. Literatur yang digunakan adalah teknik pemrograman untuk pengembangan permainan dua dimensi menggunakan Unity dengan bahasa pemrograman C#.

2. **Analisis dan perancangan sistem**

   Pada tahap ini dilakukan analisis dan pendefinisian kebutuhan sistem untuk masalah yang sedang dihadapi. Selanjutnya, dilakukan perancangan sistem dengan beberapa tahap sebagai berikut:

   a. analisis aktor yang terlibat didalam sistem;
   b. perancangan proses aplikasi;
   c. perancangan antar muka sistem; dan
   d. perancangan diagram kelas sistem.

3. Implementasi

Pada tahap ini dilakukan pembuatan elemen perangkat lunak. Sistem yang dibuat berpedoman pada rancangan yang telah dibuat pada proses perancangan dan analisis sistem.

Perincian tahap ini adalah sebagai berikut:

a. implementasi pembuatan *class library*; dan
b. implementasi cara penggunaan *class library*.

4. Pengujian dan evaluasi

Pada tahap ini akan dilakukan pengujian terhadap perangkat lunak menggunakan skenario yang telah disiapkan sebelumnya. Uji coba dan evaluasi dilakukan untuk mencari masalah yang memiliki kemungkinan muncul, mengevaluasi apakah jalan program sesuai dengan keinginan, dan melakukan perbaikan ketika kekurangan ditemukan.

Pengujian bersifat *black-box* akan dilakukan pada tahap ini. Metode yang akan digunakan dalam pengujian adalah *functional testing*. *Functional testing* dilakukan untuk menguji apakah modul yang dibuat dapat bekerja dengan sebagaimana mestinya.

5. Penyusunan buku tugas akhir

Pada tahap ini dilakukan pendokumentasian dan pelaporan dari seluruh konsep, dasar teori, implementasi, proses yang telah dilakukan, dan hasil-hasil yang telah didapatkan selama pengerjaan tugas akhir.

1.6. Sistematika Penulisan

Buku tugas akhir ini bertujuan untuk mendapatkan gambaran dari pengerjaan tugas akhir ini. Selain itu, diharapkan dapat berguna untuk pembaca yang tertarik untuk melakukan pengembangan lebih lanjut. Secara garis besar, buku tugas akhir terdiri atas beberapa bagian seperti berikut ini.

Bab I Pendahuluan

Bab ini berisi latar belakang masalah, tujuan dan manfaat pembuatan tugas akhir, permasalahan, batasan masalah, metodologi yang digunakan, dan sistematika penyusunan tugas akhir.

Bab II Dasar Teori

Bab ini membahas beberapa teori penunjang yang berhubungan dengan pokok pembahasan dan mendasari pembuatan tugas akhir ini.

Bab III Analisis dan Perancangan Sistem

Bab ini membahas mengenai perancangan perangkat lunak. Perancangan perangkat lunak meliputi perancangan data, arsitektur, proses dan perancangan antarmuka pada kakas.

Bab IV Implementasi

Bab ini berisi implementasi dari perancangan perangkat lunak.

Bab V Pengujian dan Evaluasi

Bab ini membahas pengujian dengan metode pengujian subjektif untuk mengetahui penilaian aspek kegunaan (*usability*) dari perangkat lunak dan pengujian hasil analisis kakas.

Bab VI Kesimpulan

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan. Bab ini membahas saran-saran untuk pengembangan sistem lebih lanjut.

Daftar Pustaka

Merupakan daftar referensi yang digunakan untuk mengembangkan tugas akhir.

Lampiran

Merupakan bab tambahan yang berisi kode sumber yang penting pada aplikasi ini.

BAB II
DASAR TEORI

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan tugas akhir. Teori-teori tersebut meliputi Unity, *class library*, C#, *social game*, *object oriented design*, Mono Framework, Mono Behaviour, Android, dan Android SDK.

2.1. Unity

Unity merupakan sebuah ekosistem pengembangan permainan yang terintegrasi, dan kaya akan alat atau perlengkapan yang sangat berguna untuk membangun permainan interaktif seperti *lighting*, *special effect*, *animation* dan *physics engine*. Unity dapat digunakan untuk membangun permainan dengan grafis tiga dimensi atau dua dimensi. Unity juga dapat digunakan untuk melakukan perubahan maupun pengujian permainan yang dibuat, dan ketika sudah siap permainan dapat dipublikasikan ke berbagai macam perangkat yang mendukung Unity seperti Mac, PC, Linux, Windows Phone, Android, Blackberry, Wii U, PS3, dan Xbox 360 [1].

Gambar 2.1. Tampilan Editor Unity

2.1.1. *Scene*

Setiap *object* di dalam permainan yang dibuat oleh Unity diletakkan di dalam sebuah *scene*. *Scene* digunakan untuk membuat menu utama, level, dan lain sebagainya. *Scene* juga dapat dianalogikan sebagai level. Di dalam sebuah *scene*, penulis dapat meletakkan entitas, dekorasi permainan dan lain sebagainya untuk dibangun menjadi sebuah permainan.

2.1.2. *GameObject*

GameObject adalah bagian terkecil dari sebuah permainan yang dibangun menggunakan Unity. *Programmer* dapat menambahkan atribut dan juga *class* program ke dalam sebuah *GameObject*. Setiap *GameObject* pada Unity memiliki fungsi yang berbeda - beda.

2.1.3. *Prefabs*

Prefabs adalah sebuah *GameObject* yang dapat digunakan pada beberapa *scene*. Sebuah *prefabs* juga dapat digunakan berulangkali pada sebuah *scene* yang sama tanpa merubah sifat asli dari *prefabs* tersebut. Sebuah *prefabs* yang dipanggil melalui kode program akan memiliki akhiran *"clone"* pada nama *prefabs*. Didalam jendela *Hierarchy*, *prefabs* dilambangkan dengan *GameObject* yang memiliki teks berwarna biru.

2.1.4. *Resources*

Resources merupakan kumpulan aset permainan baik berupa *AudioClip*, *GameObject*, *Image* ataupun animasi. *Resources* dapat dikatakan mirip dengan sebuah *prefabs*, yang mana tujuan penggunaannya adalah untuk membuat sebuah *template* objek sehingga dapat dipanggil dan diduplikasi berulang-ulang. Perbedaan mendasar dari *resources* dan *prefabs* adalah cara penggunaannya. *Resources* bisa dipanggil secara

langsung melalui *code*, sedangkan *prefabs* digunakan langsung melalui Unity Editor. Perbedaan lain adalah *resources* harus ditempatkan di *folder Resources* sedangkan *prefabs* bisa diletakkan dimana saja.

2.2. *Class Library*

Class library adalah kumpulan dari *class* yang sudah ditulis sebelumnya, yang bisa dipakai oleh seorang pemrogram ketika mengembangkan sebuah aplikasi. Pemrogram dapat memilih ingin memakai *class* yang mana yang diperlukan oleh aplikasi tersebut. Akses dan penggunaan *class library* sangat memudahkan pekerjaan pemrogram karena pemrogram tidak perlu menulis ulang kode yang sudah dibuat sebelumnya untuk setiap pemakaian [2].

2.3. C#

C# adalah bahasa pemrograman buatan Microsoft yang dikembangkan untuk bersaing dengan bahasa pemrogaman Java milik Sun. C# adalah sebuah bahasa pemrogaman berbasis objek yang digunakan dengan *web service* pada *platform* .NET dan dirancang untuk meningkatkan produktivitas pada pengembangan aplikasi jaringan. C# memiliki fitur-fitur seperti *type-safety*, *garbage collection*, *simplified type declarations*, *versioning* dan *scalability support* [3].

- *Type safety* adalah pencegahan *error* yang dikarenakan adanya perbedaan tipe data atau penggunaan tipe data yang tidak tepat [4].
- *Garbage collection* adalah sebuah bentuk manajemen memori yang terotomatisasi. *Garbage collector* mendealokasi memori yang digunakan oleh objek yang tidak sedang digunakan oleh program [5].
- *Versioning* adalah proses evolusi sebuah komponen seiring berjalannya waktu sambil menjaga keselarasan komponen. Sebuah versi baru disebut *source compatible* dengan versi sebelumnya apabila kode yang bekerja dengan versi

sebelumnya dapat bekerja dengan versi baru setelah program disusun ulang [6].

- *Scalability* adalah kemampuan program dalam meningkatkan sumber daya untuk meningkatkan layanan. Ciri khas aplikasi yang *scalable* adalah beban kerja tambahan hanya memerlukan sumber daya tambahan tanpa perubahan ekstensif dalam aplikasi [7].

2.4. *Social Game*

Social game adalah sebuah jenis permainan yang menggunakan interaksi sosial sebagai penambah daya tarik dan pengalaman bermain permainan tersebut. *Social game* bergantung pada interaksi sosial antara seorang pemain dengan pemain lainnya, misalnya persaingan dan kerja sama antar pemain.

Social game pada umumnya menerapkan mekanisme permainan *multiplayer* dan *asynchronous*. *Social game* pada umumnya diimplementasikan sebagai sebuah *browser game* tetapi bisa juga diimplementasikan di *platform* lain.

- *Asynchronous*, memungkinkan pemain untuk bermain dengan pemain lain secara tanpa harus bermain bersamaan di saat yang sama.
- Tidak ada kondisi menang, permainan tidak memiliki akhir dan tidak ada pemain yang dinyatakan sebagai pemenang.
- *Virtual Currency*, sebuah mata uang virtual yang bisa dibeli menggunakan mata uang asli dan digunakan untuk membeli *upgrade* [8].

Gambar 2.2. *Social Game* The Sims Social

2.5. *Object Oriented Design*

Object Oriented Design adalah konsep yang memaksa pemrogram untuk merencanakan kode yang ingin dibuat terlebih dahulu sebelum membuatnya untuk mendapatkan sebuah program yang memiliki aliran proses yang lebih baik. Beberapa konsep dasar yang dianut oleh OOD antara lain adalah enkapsulasi, penurunan sifat, antarmuka, dan *polymorphism*.

- Enkapsulasi digunakan untuk menyembunyikan nilai suatu data dalam objek untuk mencegah akses langsung dari pihak luar.
- Penurunan sifat adalah pembuatan sebuah *class* yang didasarkan pada atau menggunakan implementasi yang sama dengan *class* lain.

- Antarmuka mendeskripsikan aksi apa saja yang bisa dilakukan sebuah objek.
- *Polymorphism* adalah pola pemrograman dalam OOD di mana sebuah *class* bisa diperlakukan sebagai *class* lain [9].

2.6. Mono Framework

Mono adalah sebuah proyek *open source* dan gratis yang disponsori oleh Xamarin. Mono Framework adalah *development framework* yang setara dengan *framework* .NET milik Microsoft. Karena Mono memiliki *runtime* tersendiri yaitu Mono Runtime yang menjalankan program yang memiliki arsitektur prosesor, Unity yang menggunakan Mono dapat mendukung berbagai macam *platform*.

Mono Framework mendukung berbagai macam bahasa pemrograman, yaitu C#, JavaScript dan Boo. C# yang tersedia pada Unity setara dengan yang tersedia pada Visual Studio 2005 [10].

2.7. *MonoBehaviour*

Setiap kelas yang diimplementasikan ke *game object* pada Unity pasti merupakan turunan dari *MonoBehaviour*, karena *MonoBehaviour* adalah kelas induk dari *game object* pada Unity. Kelas turunan *MonoBehaviour* dapat menggunakan fungsi-fungsi seperti *Start*, *Awake*, *Update*, *FixedUpdate*, serta *OnGUI*. Pada C# dan Boo *MonoBehaviour* harus diturunkan secara eksplisit, namun pada Java setiap kelas akan otomatis merupakan turunan dari *MonoBehaviour*.

Checkbox yang ada pada *editor* akan mengatur berjalannya fungsi yang ada dan tidak akan mempengaruhi berjalannya fungsi dasar lain yang ada pada *MonoBehaviour* [11].

2.8. Android

Android merupakan sistem operasi berbasis Linux yang dikembangkan secara khusus untuk perangkat *touchscreen* seperti *smartphone*. Sistem operasi ini pertama kali dikembangkan oleh Android, Inc. hingga pada akhirnya Google mengambil alih pada

tahun 2005. Dan pada Oktober 2008, telepon genggam pertama yang menggunakan Android diluncurkan, yaitu T-Mobile G1. Meskipun berjalan pada *kernel* Linux, Android bukanlah merupakan sebuah *distro* karena sudah meninggalkan banyak komponen dari Linux seperti *editor*, *framework*, GUI, *library*, dan *shell* [12].

2.9. Android SDK

Android SDK (*Software Development Kit*) merupakan sebuah *tools* API (*Application Programming Interface*) yang digunakan untuk memulai mengembangkan aplikasi Android dengan menggunakan Java. Sampai saat ini, telah disediakan banyak Android SDK untuk berbagai macam versi Android, mulai dari versi pertama hingga versi terbaru. Pada Unity3D Android SDK digunakan untuk menerjemahkan program yang telah dibuat dalam bahasa C#, JavaScript maupun Boo sehingga dapat dibaca dan dijalankan oleh Android [13].

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. 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 dan deskripsi umum sistem.

3.1.1. Analisis Permasalahan

Permasalahan utama yang diangkat dalam pengerjaan tugas akhir ini adalah bagaimana membuat *class library* yang memuat implementasi aturan main yang dapat dipakai oleh *game Heart Meister*. Permasalahan kedua adalah bagaimana cara merancang *class library* yang didasarkan pada *object oriented design*.

3.1.2. Deskripsi Umum

Sistem yang akan dibuat berupa *class library. Class library* ini dapat digunakan oleh *game Heart Meister* untuk mengatur hal-hal universal, misalnya pengaturan volume suara, penggunaan *web service*, penyimpanan data, dan pemberian efek pergantian layar. *Class library* ini akan dibuat dengan menggunakan bahasa pemrograman C# dan *MonoLibrary* agar *class library* ini mudah berinteraksi dengan Unity. Untuk efek pengaturan suara, dengan pemakaian *class library* ini pengguna dapat menyambungkan suara latar yang diputar ketika terjadi pergantian *scene* yang memiliki *background music* yang sama sehingga tidak akan terjadi perpotongan *file* audio. Pada efek pengaturan penggunaan *web service*, seluruh lalu lintas *web service* akan diatur melewati *library*

tersebut terlebih dahulu sehingga memudahkan pembatasan. Penyimpanan data dapat dilakukan untuk menyimpan data untuk proses *offline* sehingga permainan tidak akan menuntut penggunaan koneksi *internet* yang berlebihan. Pemberian efek pergantian layar memberikan transisi *scene* yang halus sehingga tidak mengagetkan pemain dan memberi waktu tambahan pemuatan aset-aset yang diperlukan *scene*.

Permainan *Heart Meister* sendiri adalah sebuah permainan dimana pemain berusaha menjadi *Heart Meister* terhebat. *Heart Meister* adalah sebutan bagi para pelatih monster yang disebut *Heart*. Pemain dapat bertualang menantang berbagai macam musuh untuk melatih *Heart* milik mereka untuk menjadi lebih kuat. Pemain dibekali toko untuk membeli *Heart* baru yang dapat dilatih.

3.1.3. Analisis *Class Library*

Class library yang digunakan oleh *game Heart Meister* dapat dibagi menjadi tiga kelompok, kelompok kelas yang termasuk dalam bagian *MainGameplay* yang dapat mengakses kelas-kelas inti dari permainan, kelompok kelas *HMCL* yang merupakan kelas-kelas yang mengatur transisi antar *scene* Unity, dan kelas yang tidak termasuk dalam *namespace* tersebut.

Pada kelompok kelas yang termasuk dalam bagian *MainGameplay*, yang merupakan kelas-kelas yang terhubung dengan kelas inti dari *game Heart Meister*, terdapat kelas-kelas yang dibutuhkan agar *game* dapat berjalan sesuai dengan aturan main yang telah dibuat dan logika permainan lainnya. Kelompok kelas *HMCL* adalah kelas yang mengatur perpindahan antar *scene* Unity, sehingga efek *fade in* dan *fade out* dapat diberikan dan musik latar dapat disesuaikan apabila terjadi kesamaan. Kelas yang terdapat diluar *namespace* adalah kelas-kelas yang berhubungan secara langsung dengan *GameObject* yang ada pada Unity dan kelas-kelas yang tidak mengakses kelas-kelas utama dari permainan.

3.1.4. Fungsional Sistem

Fungsi-fungsi yang dimiliki oleh c*lass library game Heart Meister* dapat dilihat pada Tabel 1.

Tabel 1. Daftar Fungsi pada *Class Library*

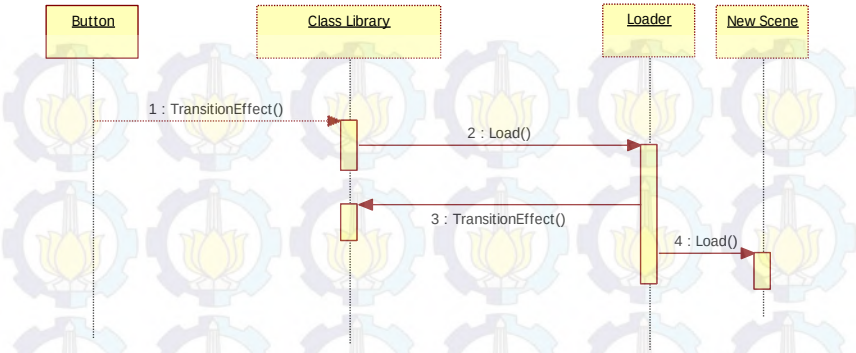
No	Fungsi
1	Pengatur transisi antar *scene* Unity
2	Pengatur transisi antar musik latar
3	*Interface* antara Unity dengan *web service*
4	Penyimpanan data *game*
5	Pengatur aturan main pertarungan
6	Pengatur aturan main batasan monster
7	Pengatur aturan main *energy* pemain

3.1.4.1. Pengatur Transisi Antar *Scene* Unity

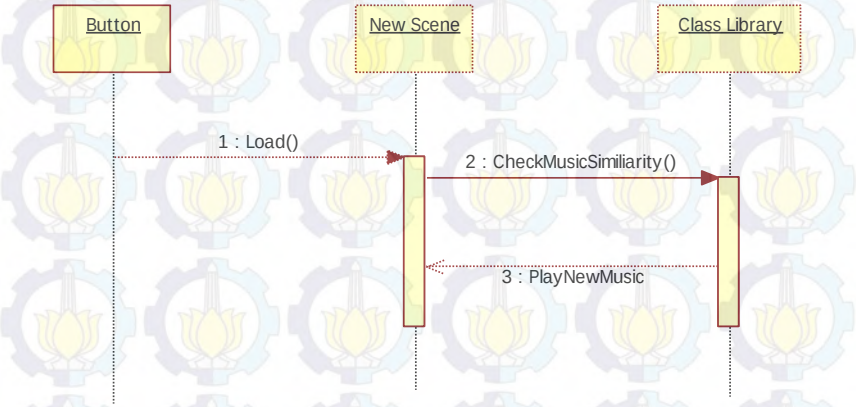
Pada saat pemain melakukan perpindahan dari sebuah *scene* ke *scene* lainnya di Unity, Unity akan langsung memindahkan tampilan ke *scene* baru. *Class library* akan membuat transisi antar *scene* lebih halus dengan pemberian efek *fade in* dan *fade out* setiap kali terjadi transisi *scene* pada Unity. Setelah pemain memulai perpindahan *scene*, *class library* akan memberikan efek transisi berupa *fade out* lalu *fade in* kedalam *scene Loader*, yang lalu akan meminta efek transisi sekali lagi sebelum berubah menjadi *scene* tujuan. Diagram *sequence* untuk fungsi pengatur transisi antar *scene* Unity dapat dilihat pada Gambar 3.1.

3.1.4.2. Pengatur Transisi Antar Musik Latar

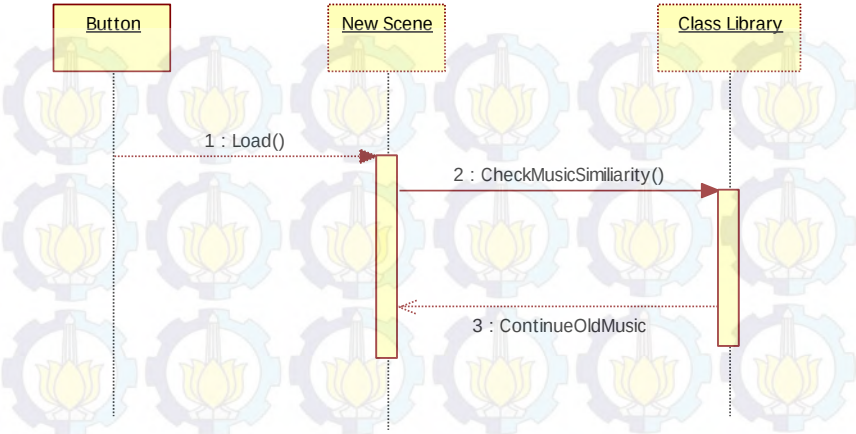
Ketika perpindahan *scene* dalam Unity terjadi dan masing-masing *scene* memiliki musik latar yang sama, maka Unity akan mengulang musik latar tersebut dari awal. *Class library* akan membuat *game* mampu mengenali ketika musik latar dari *scene* yang akan didatangi sama dengan sebelumnya dan membuat musik latar terus dilanjutkan. Diagram *sequence* untuk fungsi pengatur transisi musik latar dapat dilihat pada Gambar 3.2 dan Gambar 3.3.



Gambar 3.1. Diagram *Sequence* Fungsi Pengatur Transisi *Scene* Unity



Gambar 3.2. Diagram *Sequence* Fungsi Pengatur Transisi Musik Latar Ketika Musik Latar Berbeda dengan Sebelumnya



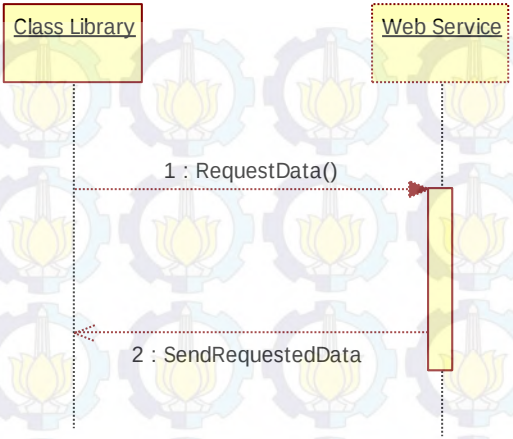
Gambar 3.3. Diagram *Sequence* Fungsi Pengatur Transisi Musik Latar Ketika Musik Latar Sama dengan Sebelumnya

3.1.4.3. *Interface* Antara Unity Dengan *Web Service*

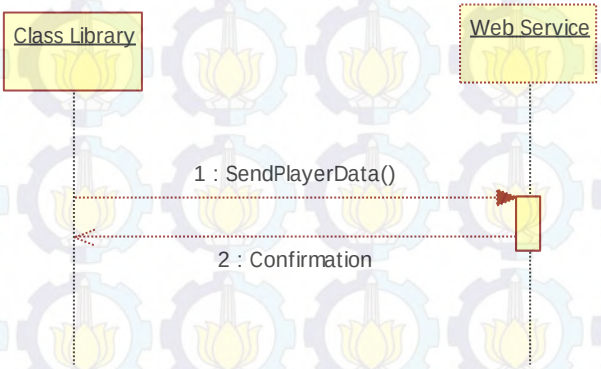
Class library memiliki bagian yang akan berfungsi sebagai perantara antara *game* dengan *web service*. Seluruh komunikasi antara *game* dan *web service* akan dilakukan lewat *class library*. *Class library* mampu menerima maupun mengirim data ke *web service* untuk mengakses *database*. Diagram *sequence* penerimaan dan pengiriman data dapat dilihat pada Gambar 3.4. dan 3.5.

3.1.4.4. Penyimpanan Data *Game*

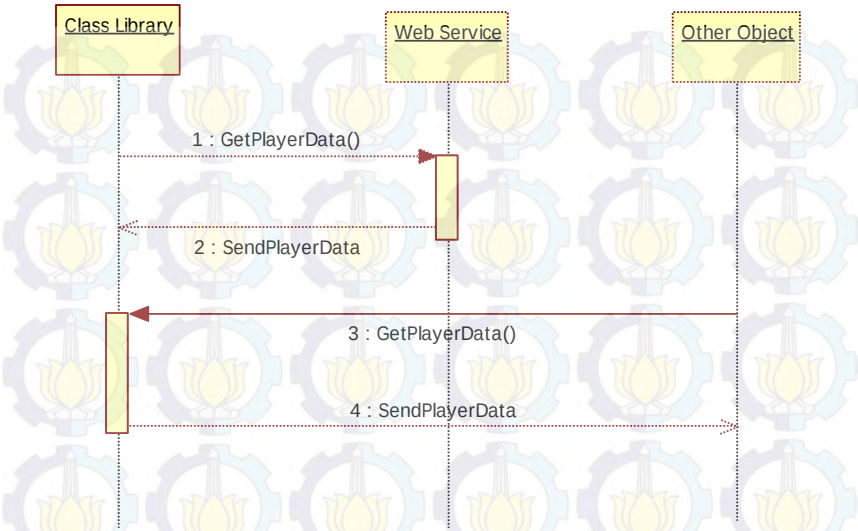
Data yang digunakan oleh *game* dapat disimpan oleh *class library* sehingga program tidak harus selalu terhubung dengan *web service* pada poin-poin tertentu. Ketika sebuah bagian permainan menginginkan data pemain yang sudah tersimpan dalam *class library*, maka *class library* hanya perlu memberikan data tersebut tanpa memerlukan koneksi ke *web service* lagi. Diagram *sequence* dari penyimpanan data *game* dapat dilihat pada Gambar 3.6.



Gambar 3.4. Diagram *Sequence* Permintaan Data ke *Web Service*



Gambar 3.5. Diagram *Sequence* Pengiriman Data ke *Web Service*



Gambar 3.6. Diagram *Sequence* Penyimpanan Data dari *Web Service* dalam *Class Library*

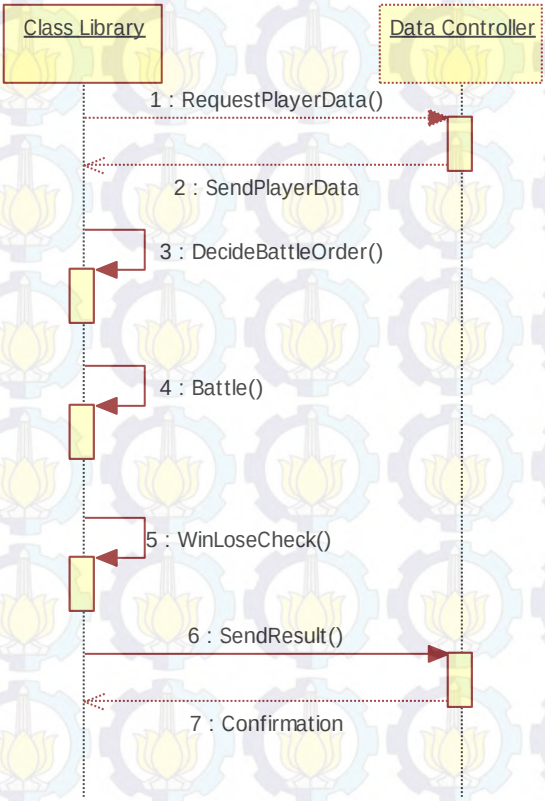
3.1.4.5. Pengatur Aturan Main Pertarungan

Class library mengatur jalannya pertarungan dalam *Heart Meister*. Pertarungan dalam permainan *Heart Meister* bersifat otomatis. Ketika pertarungan dimulai, *class library* akan mengambil data monster milik pemain dan musuh dari penyimpanan data pemain. *Class library* menentukan urutan giliran pertarungan dari masing-masing monster tersebut menggunakan kecepatan dari masing-masing monster. Pertarungan akan dijalankan melewati giliran tersebut hingga salah satu tim kehabisan seluruh *health point* monsternya. Diagram *sequence* aturan main ini dapat dilihat pada Gambar 3.7.

3.1.4.6. Pengatur Aturan Main Batasan Monster

Dalam permainan *Heart Meister* pemain harus membawa tiga monster setiap saatnya, dan hanya boleh memiliki maksimal

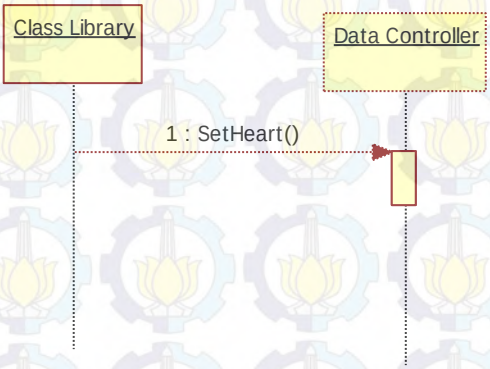
enam monster. Hal ini tidak diberitahu atau ditekankan secara spesifik oleh permainan, tapi berbagai aspek permainan membuat pemain selalu membawa tiga monster dan memiliki paling banyak enam monster.



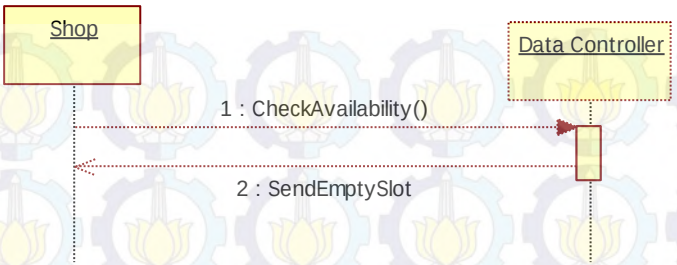
Gambar 3.7. Diagram *Sequence* Pengatur Aturan Main Pertarungan

Aspek paling utama adalah karena ketika pemain memulai permainan, pemain otomatis mendapatkan tiga monster dalam *party*. Pemain tidak dapat mengeluarkan monster dari *party*

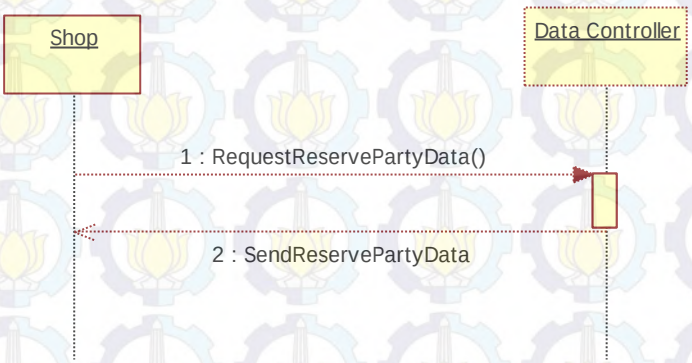
utama sehingga pemain selalu membawa tiga monster. Layar penjualan monster hanya memperbolehkan penjualan monster yang ada pada kolom cadangan, sehingga pemain tetap membawa tiga monster. Layar pembelian dan undian monster hanya dapat memberikan monster baru apabila ada tempat kosong di kolom cadangan pemain, sehingga pemain hanya dapat memiliki paling banyak enam monster. Diagram *sequence* pada Gambar 3.8. menggambarkan pemberian tiga *Heart* yang menjadi batas awal sehingga pemain tidak bisa membawa lebih dari tiga *Heart*. Diagram *sequence* pada Gambar 3.9 menunjukkan proses pembelian *Heart* dari *Shop* sehingga pemain tidak akan melewati batas enam *Heart* yang dimiliki. Diagram *sequence* pada Gambar 3.10 menunjukkan proses penjualan *Heart* di *Shop*, yang membuat pemain tidak dapat menjual *Heart* yang ada di *party* dan sekaligus membuat *party* tiga.



Gambar 3.8. Diagram *Sequence* Pemberian Tiga *Heart* Saat Pendaftaran



Gambar 3.9. Diagram *Sequence* Pembelian *Heart*

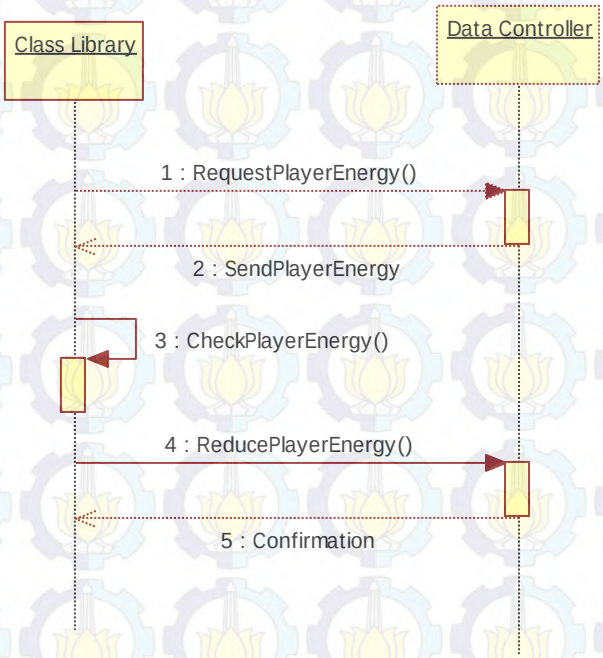


Gambar 3.10. Diagram *Sequence* Penjualan *Heart*

3.1.4.7. Pengatur Aturan Main *Energy* Pemain

Dalam permainan *Heart Meister* pemain memiliki sebuah sumber daya yang disebut *energy*. *Energy* digunakan ketika pemain ingin memasuki *dungeon* atau *epic battle*. Ketika pemain ingin memasuki *dungeon*, permainan akan memeriksa apakah *energy* pemain cukup untuk memasuki *dungeon*. Bila cukup barulah pemain dapat memasukinya. Begitu juga dengan *epic battle*. Pemain membutuhkan lima *energy* untuk memasuki *dungeon* dan sepuluh untuk *epic battle*. Jumlah *energy* yang dimiliki pemain diambil dari

server yang mengkalkulasi *energy* yang dimiliki pemain. Diagram *sequence* milik pengatur aturan main *energy* pemain dapat dilihat pada Gambar 3.11.



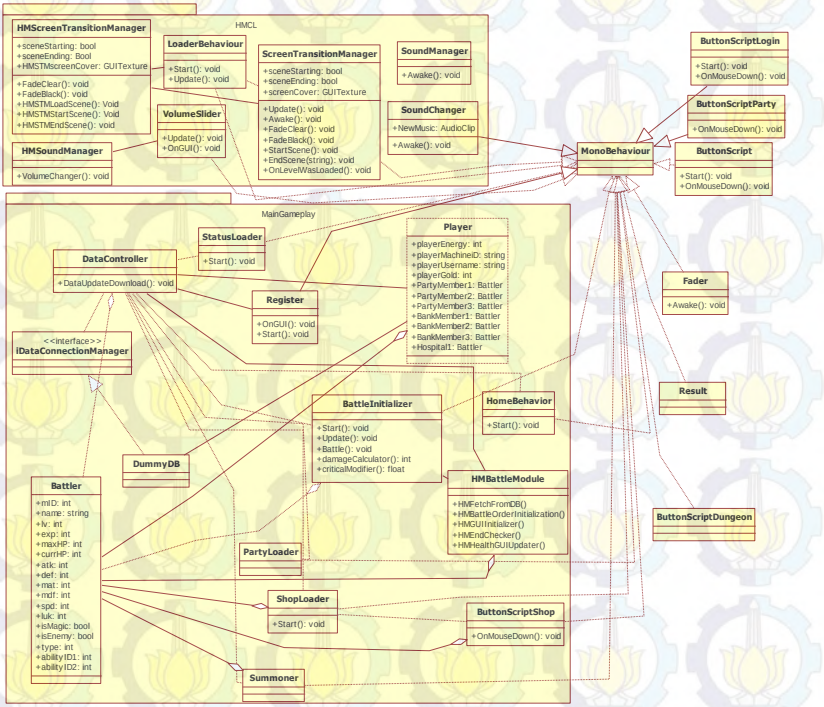
Gambar 3.11. Pengatur Aturan Main *Energy* Pemain

3.2. Perancangan Sistem

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

3.2.1. Perancangan Diagram Kelas

Perancangan diagram kelas berisi rancangan dari kelas-kelas yang digunakan untuk membangun permainan. Pada subbab ini, hubungan dan perilaku antar kelas digambarkan dengan lebih jelas. Tiga kelompok kelas yang terdapat pada arsitektur ini adalah kelompok kelas *MainGameplay*, kelompok kelas *HMCL*, dan kelas yang tidak termasuk dalam *namespace* tersebut.



Gambar 3.5. Desain Diagram Kelas Seluruh Sistem

3.2.1.1. Kelompok kelas bagian *MainGameplay*

Kelompok kelas bagian *MainGameplay* mengandung kelas-kelas yang berhubungan dengan kelas-kelas utama permainan dan kelas-kelas yang mengatur logika permainan. Terdapat kelas

Battlers yang merupakan kelas *template* dari semua monster yang ada dalam permainan. Kelas *BattleInitializer* adalah kelas yang mengatur logika permainan pada layar pertarungan, yang termasuk perhitungan kerusakan serangan, pengacakan target serangan, urutan giliran penyerangan dan lainnya. Kelas *DataController* adalah kelas yang meyimpan seluruh data pemain agar kelas-kelas lain tidak perlu mengakses *web service* setiap kali memerlukan data pemain. Antarmuka *IDataConnectionManager* merupakan antarmuka yang menjadi perantara antara sistem dan *web service* yang diwakilkan oleh kelas *DummyDB* dengan pola *dependency injection*.

3.2.1.2. Kelompok kelas bagian *HMCL*

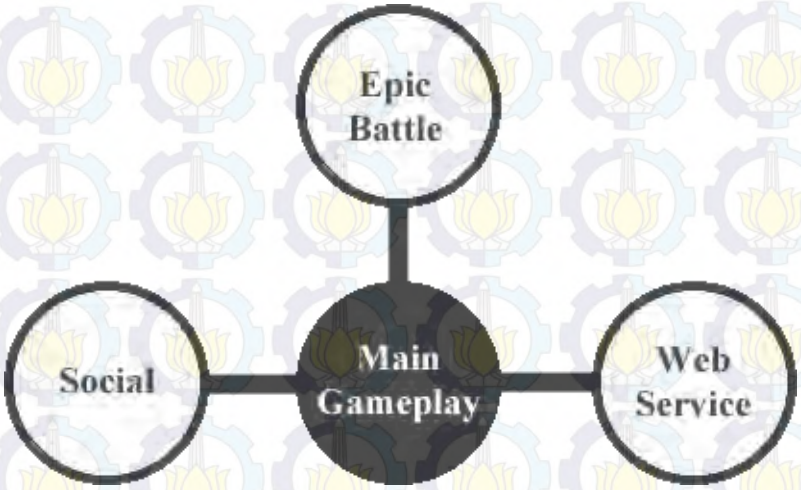
Kelompok kelas bagian *HMCL* adalah kelompok kelas yang berisikan kelas-kelas yang mengatur transisi antar *scene* pada Unity. Anggota kelas tersebut antara lain adalah *ScreenTransitionManager* yang mengatur pemberian efek perpindahan *scene* yaitu *fade in* dan *fade out*. Anggota berikutnya adalah *SoundManager* yang bertugas memeriksa musik latar setiap terjadi perpindahan *scene* sehingga apabila musik latar yang akan diputar oleh *scene* tujuan sama dengan *scene* awal, maka musik latar tersebut akan dilanjutkan tanpa terputus. Hal ini dibantu oleh anggota ketiga yaitu *SoundChanger*.

3.2.1.3. Kelas-kelas tanpa kelompok kelas

Terdapat juga kelas-kelas yang tidak termasuk dalam kelompok kelas apapun karena kelas-kelas tersebut adalah kelas yang diimplementasikan oleh *GameObject* Unity tanpa fungsi lainnya. Kelas-kelas tersebut antara lain adalah kelas *Fader* yang bertugas melindungi citra penutup yang dijadikan *fade in* dan *fade out*, kelas *VolumeSlider* yang menjalankan *scene* perubah volume suara menggunakan *slider*, dan kelas *ButtonScript* yang diimplementasikan pada hampir semua tombol umum dalam permainan.

3.2.2. Perancangan Integrasi Sistem

Pada bagian ini akan dijelaskan tentang rencana integrasi yang akan dilakukan pada modul-modul milik *game Heart Meister*. Modul yang akan diintegrasikan antara lain adalah modul *MainGameplay* sebagai modul aturan main utama yang mengandung *class library*. Modul *MainGameplay* adalah modul yang menyertakan *class library* ini. Modul berikutnya adalah modul *web service* yang bekerja sebagai perantara *MainGameplay* dengan *server*. Modul ini dibuat oleh Febi Nur Salsabila [14]. Modul *social* adalah modul yang bertugas mengatur interaksi data pemain yang satu dengan pemain yang lainnya. Modul ini dibuat oleh Hizkia Adi Surya [15]. Dan modul *epic battle* yang bertugas mengatur *event* tersendiri dalam permainan yang bernama *epic battle*, yang dibuat oleh Nabilah [16]. Semua modul terhubung ke modul *MainGameplay* sebagai pusat dari permainan *Heart Meister*. Hubungan dari keempat modul tersebut dapat dilihat pada Gambar 3.7.



Gambar 3.12. Hubungan dari Modul *Game Heart Meister*

BAB IV
IMPLEMENTASI

Bab ini membahas tentang implementasi dari perancangan *class library game Heart Meister*. Proses implementasi dari setiap kelas yang ada pada *class library* yang dibuat dengan mengacu pada rancangan yang telah dibahas sebelumnya akan dipaparkan pada bab ini. Bahasa pemrograman yang digunakan untuk *class library* adalah bahasa pemrograman C#.

4.1. Implementasi Kelompok *HMCL*

Kelompok kelas *HMCL* adalah kelompok kelas yang bertugas mengatur transisi antar *scene* pada Unity sehingga efek *fade in, fade out,* dan penyambungan musik latar dapat dilakukan. Kelas-kelas ini tidak berhubungan dengan logika dan aturan permainan. Pada bagian ini implementasi dari rancangan kelompok kelas HMCL akan dijelaskan.

4.1.1. Kelas *ScreenTransitionManager.cs*

Kelas *ScreenTransitionManager.cs* adalah kelas yang bertugas mengecek setiap perpindahan *scene* yang dilakukan dalam Unity, memberikan efek *fade out* pada saat pemain meninggalkan sebuah *scene*, dan memberikan efek *fade in* pada saat pemain memasuki sebuah *scene*.

Pada fungsi *Update* kelas akan mengecek apakah *state* dari *scene* saat ini adalah saat awal masuk *scene* atau saat akhir ingin meninggalkan *scene* tersebut. Apabila kondisi masuk *scene* (*sceneStarting*) maka kelas akan menjalankan fungsi *StartScene*. Fungsi *StartScene* sekaligus menjalankan fungsi *FadeClear* yang perlahan demi perlahan seiring dengan *Update* Unity membuat bagian *Fader* atau lapisan hitam untuk *fade in* dan *fade out* memudar sehingga menciptakan efek *fade in*. Di dalam fungsi ini juga terdapat bagian yang mengecek apabila layar sudah bersih dari *Fader* atau

belum, dan jika sudah akan kembali meng-*enable* tombol yang sebelumnya di *disable* oleh fungsi *EndScene*. Apabila kondisi meninggalkan *scene* (*sceneEnding*) dicapai, maka *Update* akan memanggil fungsi *FadeBlack* dan mulai menghitamkan *Fader* hingga layar hitam untuk menciptakan efek *fade out*. Setelah layar menjadi hitam, barulah *scene* perantara *LoadingScreen* dipanggil untuk menggantikan *scene* sebelumnya. Fungsi *OnLevelWasLoaded* akan melakukan pengecekan setiap *scene* baru di *load*, dan me-*reset state sceneStarting* dan *sceneEnding* untuk *scene* tersebut.

4.1.2. Kelas *HMScreenTransitionManager.cs*

Kelas ini adalah modifikasi kelas *ScreenTransitionManager* yang dirancang agar dapat bekerja sama dengan *ScreenTransitionManager* dalam mengeluarkan efek transisi *scene*. Kelas menerima variabel-variabel dari *ScreenTransitionManager* seperti target perpindahan *scene*. Setelah itu kelas akan menjalankan efek transisi layar berupa *fade out* dan *fade in* bersama dengan *ScreenTransitionManager*.

4.1.3. Kelas *SoundManager.cs*

Kelas *SoundManager.cs* adalah kelas yang bertugas membuat *instance* musik latar yang tidak bisa diduplikasi sehingga musik latar tidak akan bertumpuk satu dengan yang lainnya. Pada fungsi *Awake*, terdapat bagian yang memeriksa penduplikasian *instance SoundManager* dan membuat kelas ini tidak di *destroy* saat pemain melakukan pergantian *scene.* Hal ini membuat objek yang mengimplementasi kelas *SoundManager.cs* tetap berada dalam permainan walaupun *scene* telah berganti menjadi *scene* selanjutnya, sehingga objek ini mampu terus menjalankan *file* musik latar yang diinginkan selama permainan.

4.1.4. Kelas *SoundChanger.cs*

Kelas ini adalah kelas yang bekerjasama dengan *SoundManager.cs* dalam mengatur transisi musik latar. Di berbagai *scene* yang memiliki musik latar, kelas ini akan dipasangkan ke objek yang akan memainkan musik tersebut sebagai musik latar yang baru. Ketika fungsi *Awake* dari kelas ini dijalankan, kelas akan mencari *RaeySoundManager* yang merupakan objek pembawa musik latar dalam Unity. Objek *RaeySoundManager* juga membawa kelas *SoundManager.cs* yang terus menerus memainkan *file* musik latar yang dipasangkan pada objek yang sama. Kelas *SoundChanger.cs* melakukan pemeriksaan apakah musik latar yang sedang dimainkan oleh objek tadi berbeda dengan musik latar dari *scene* tempat *SoundChanger.cs* berasal, lalu menggantinya dengan *file* musik latar yang baru apabila ditemui perbedaan. Dengan ini ketika Unity menemui dua musik latar yang sama dari *scene* yang berbeda Unity tidak akan memutus lagu tersebut di tengah-tengah dan memulainya kembali dari awal seperti biasanya, namun melanjutkan musik latar tersebut.

4.1.5. Kelas *HMSoundManager.cs*

Kelas *HMSoundManager.cs* adalah kelas yang dirancang untuk dapat membantu *VolumeSlider.cs* dalam mengatur volume dalam permainan walaupun tidak dipasangkan ke *GameObject* ataupun merupakan turunan dari *MonoBehaviour*.

4.1.6. Kelas *LoaderBehavior.cs*

Kelas *LoaderBehavior.cs* adalah kelas yang berada di objek Unity yang mengatur *scene LoadingScreen*. Kelas ini akan mengambil nama target perpindahan *scene* dari *ScreenTransitionManager.cs* lalu kembali menjalankan fungsi perpindahan milik kelas tersebut sehingga efek *fade out* dan *fade in* terjadi sebelum *scene* berpindah. Kelas ini merupakan kelas perantara sehingga *scene* berikutnya dapat dimuat terlebih dahulu sebelum dibuka.

4.1.7. Kelas *VolumeSlider.cs*

Kelas *VolumeSlider.cs* adalah kelas yang mengimplementasikan fungsi *OnGUI* untuk menampilkan sebuah *slider* horisontal yang dapat digunakan untuk mengatur volume dari *game* itu sendiri. Bersama dengan *HMSoundManager.cs* kelas ini dapat mengatur volume dalam permainan.

4.2. Implementasi Kelompok *MainGameplay*

Kelompok kelas *MainGameplay* adalah kelompok kelas yang mengakses kelas-kelas utama dalam logika permainan *Heart Meister*. Termasuk di dalam kelas ini adalah kelas-kelas logika permainan, seperti kelas yang mengatur pertarungan, kelas yang mengatur undian, toko, dan lain-lain. Pada bagian ini implementasi dari rancangan kelompok kelas *MainGameplay* akan dijelaskan.

4.2.1. Kelas *Battler.cs*

Kelas *Battler.cs* adalah salah satu kelas utama dalam *Heart Meister* dimana kelas ini merupakan *template* dari setiap monster yang ada di permainan, baik milik pemain maupun musuh. Dalam kelas ini terdapat semua aspek monster, mulai dari kekuatan serangan sampai nama dan kecepatannya. Kelas ini dipanggil oleh semua kelas yang berhubungan dengan monster.

Kekuatan serangan monster terbagi menjadi dua, yaitu *attack* dan *magic attack*. *Attack* merupakan kekuatan serangan monster dalam serangan bertipe fisik, sedangkan *magic attack* merupakan kekuatan serangan monster dalam serangan bertipe magis. Masing-masing tipe serangan memiliki unit pertahanan yang berbeda, dengan *defense* untuk pertahanan dari serangan berbasis fisik (*attack*), dan *magic defense* untuk pertahanan dari serangan berbasis magis (*magic attack*).

Aspek *speed* dari monster adalah aspek yang menentukan monster mana yang dapat menyerang lebih dahulu dalam pertarungan. Tentunya monster dengan *speed* lebih tinggi dapat

menyerang terlebih dahulu. *HP*, singkatan dari *health point*, adalah aspek monster yang menentukan seberapa banyak serangan yang dapat diterima oleh monster tersebut sebelum monster tersebut mengalami kekalahan. *Luck* adalah aspek monster yang menentukan seberapa sering monster dapat melancarkan serangan yang lebih kuat dari biasanya atau c*ritical hit* kepada musuh. Semakin tinggi nilai *luck* dari sebuah monster, semakin sering monster tersebut dapat melancarkan *critical hit* kepada musuh yang diserangnya.

Aspek terakhir dari monster adalah *Type*, dimana monster memiliki tiga jenis tipe, yaitu *Humanoid*, *Cool*, dan *Cute*. Ketiga tipe ini menentukan seberapa banyak tambahan atau pengurangan serangan yang akan dimasukan kedalam perhitungan saat monster menyerang monster lainnya.

4.2.2. Kelas *BattleInitializer.cs*

Kelas *BattleInitializer.cs* adalah salah satu kelas utama dalam logika permainan. Kelas ini mengatur seluruh kejadian yang ada pada pertarungan pemain melawan musuh. Pada fungsi *Start* ada pemanggilan beberapa fungsi untuk mempersiapkan pertarungan. Pertama-tama ada fungsi *FetchFromDB* yang bertugas mengambil data para petarung yang ada dari penyimpanan sementara *game*. Setelah data petarung diambil, maka data tersebut dimasukkan ke *Battler* yang sudah disiapkan pada kelas ini untuk mempermudah penyimpanan dan pemanggilan status monster nantinya. Fungsi berikutnya adalah *BattleOrderInitialization* yang bertugas mengurutkan giliran petarung sesuai dengan kecepatan masing-masing petarung. Fungsi *InitialGUI* memperlihatkan *health point* awal para petarung pada tampilan yang dapat dilihat oleh pemain. Di fungsi *Update* yang berjalan terus menerus terdapat fungsi *HealthGUIUpdater* yang berfungsi memperbaharui tampilan *health point* setiap kali update berjalan. Fungsi *EndChecker* mengecek apakah kondisi akhir pertarungan sudah tercapai, antara seluruh monster pemain telah kalah atau seluruh monster musuh telah kalah.

Fungsi *Battle* yang merupakan fungsi utama pada kelas ini adalah fungsi yang mengatur alur pertarungan tiap gilirannya, mulai dari siapakah yang harus menyerang, siapakah yang akan diserang, menganimasikan serangan, mengkalkulasikan kerusakan dari serangan, serta mengecek apakah target serangan kalah atau tidak. Variabel *DamageCalculator* yang tadi dipergunakan untuk fungsi *Battle* didapatkan setelah kalkulasi status *attack* atau *magic attack* penyerang terhadap *defense* atau *magic defense* target serangan dan juga kemungkinan terjadinya *critical hit*. *Critical hit* dihitung dengan menggunakan variabel *criticalModifier* yang sekaligus menampilkan di layar pemain apabila terjadi *critical hit* atau serangan yang lebih kuat dari biasanya. *HitAnimation* adalah *CoRoutine* yang dijalankan bersama-sama dengan fungsi *Battle*, karena *CoRoutine* adalah semacam *thread* milik Unity.

4.2.3. Kelas *HMBattleModule.cs*

Kelas *HMBattleModule.cs* adalah kelas yang membantu kelas *BattleInitializer.cs* dalam mengatur aturan main pertarungan. Kelas *BattleInitializer.cs* diimplementasikan ke objek Unity, sedangkan kelas ini bekerja di belakang layar tanpa diimplementasikan ke objek Unity. Kelas ini memberikan parameter milik monster pemain dan musuh dan menentukan urutan pertarungan, lalu memberikannya ke *BattleInitializer.cs* yang menjalankan kalkulasi pertarungan.

4.2.4. Kelas *DataController.cs*

Kelas *DataController.cs* adalah kelas yang menyimpan seluruh data pemain. Kelas ini akan meminta data pemain pada *web service* dan menyimpannya sehingga *client* tidak akan terlalu sering berhubungan dengan *web service.* Setiap kali permainan membutuhkan data yang tidak bersifat merubah, misalkan hanya ingin menunjukkan jumlah uang yang dimiliki oleh pemain pada saat ini, maka kelas lain tersebut dapat mendapatkan data tersebut secara langsung dengan mengakses data pemain yang tersimpan pada

DataController.cs. Bagian terpenting dari *DataController.cs* adalah bagian fungsi *DataUpdateDownload* yang menjalankan fungsi pengambilan data pemain dan menyimpannya dalam permainan untuk digunakan nantinya. Contoh data yang disimpan dalam kelas ini misalnya adalah berapa banyak uang atau *gold* yang dimiliki oleh pemain, siapa sajakah monster yang dimiliki oleh pemain, siapakah *username* pemain, apakah *machine ID* pemain, dan aspek pemain lainnya.

4.2.5. Interface *IDataConnectionManager.cs*

Interface IDataConnectionManager.cs adalah *interface* untuk menghubungkan *DataController* dengan *web service*. *Interface* ini merupakan realisasi dari *pattern dependency injection.* Kelas ini berisi fungsi-fungsi yang menghubungkan *DataController.cs* dengan *DummyDB.cs.* Dengan menggunakan fungsi milik *interface* ini, *DataController.cs* dapat mengambil berbagai properti milik pemain yang tersimpan baik di dalam *server* maupun di dalam *DummyDB.cs*.

4.2.6. Kelas *DummyDB.cs*

Kelas *DummyDB.cs* merupakan kelas untuk menggantikan fungsi *web service* selama *game* berjalan secara *offline*. Kelas ini menyimpan data-data pemain, seperti jumlah uang, jenis monster, *username*, dan lain-lain. Sebagai turunan dari *interface IDataConnectionManager.cs*, *DummyDB.cs* memiliki semua fungsi yang dimiliki oleh *IDataConnectionManager.cs*. Fungsi-fungsi tersebut dapat digunakan untuk mengakses variabel-variabel yang ada dalam kelas *DummyDB.cs*.

4.2.7. Kelas *PartyLoader.cs*

Kelas *PartyLoader.cs* adalah kelas yang diimplementasikan di *scene PartyScreen*, dan berfungsi mengatur pertukaran monster pada *party* yang terjadi dalam *scene* tersebut. *PartyLoader.cs*

mengambil data monster milik pemain dari *DataController.cs*, lalu menempatkannya dalam tampilan yang mudah dimengerti oleh pemain sambil menampilkan atribut-atribut yang dimiliki oleh monster tersebut. Ketika pemain memilih monster yang ada, atribut milik monster tersebut akan ditampilkan dibawah layar. Jika pemain memilih monster lain setelahnya, maka dengan mudah pemain dapat melakukan pemindahan posisi monster.

4.2.8. Kelas *ShopLoader.cs*

Kelas *ShopLoader.cs* adalah kelas yang diimplementasikan di *scene Shop* dan mengatur tampilan serta tombol-tombol yang ada pada *scene* tersebut.

4.2.9. Kelas *Summoner.cs*

Kelas *Summoner.cs* adalah kelas yang diimplementasikan di *scene Summoner* dan mengatur pengacakan sistem undian yang ada di *scene* tersebut. Ketika pemain menekan tombol dan mengaktifkan undian, jika pemain memiliki sebuah *slot* kosong atau cukup uang untuk mengikuti undian, maka pemain akan mendapatkan satu monster yang akan terpilih secara acak dari kumpulan monster yang bisa didapatkan disini.

4.2.10. Kelas *HomeBehavior.cs*

Kelas *HomeBehavior.cs* adalah sebuah kelas yang diimplementasikan pada *scene Home* untuk menunjukkan jumlah *energy* pemain yang tersisa. Kelas mengambil data *energy* ke *DataController* dan menampilkannya pada sebuah *text mesh* di *Home*.

4.2.11. Kelas *Register.cs*

Kelas *Register.cs* adalah kelas yang mengatur *scene Register* dalam permainan. Kelas ini memberikan pemain sebuah tampilan

berjenis *OnGUI* yang menyediakan sebuah *text box* yang dapat diisi untuk memilih *username* saat pemain masuk ke dalam *game* untuk pertama kalinya. Dengan menekan tombol *confirm* yang telah disediakan, pemain dapat menyimpan *username* yang telah dimasukkan ke dalam *text box* menjadi *username* milik pemain tersebut.

4.2.12. Kelas *Result.cs*

Kelas *Result.cs* adalah kelas yang mengatur tampilan dan pengiriman data setelah pemain mengakhiri pertarungan dengan kemenangan. Pada kelas ini, pemain dapat melihat hasil dari pertarungan, mulai dari berapa poin pengalaman yang didapatkan monster hingga berapa *gold* yang didapatkan oleh pemain.

4.3. Kelas-kelas Lainnya

Selain kelas-kelas yang berada pada *namespace HMCL* dan *namespace MainGameplay*, ada beberapa kelas yang tidak termasuk dalam *namespace* manapun. Kelas-kelas tersebut adalah kelas yang hanya memiliki satu fungsi yang spesifik yang tidak melibatkan logika permainan ataupun transisi *scene*. Pada bagian ini implementasi dari kelas-kelas yang tidak termasuk *namespace* apapun tersebut akan dijelaskan.

4.3.1. Kelas *ButtonScript.cs*

Kelas *ButtonScript.cs* adalah kelas yang paling banyak digunakan di berbagai macam objek pada *game Heart Meister*. Kelas ini memberikan efek *tween* pada objek sehingga pemain dapat dengan lebih mudah memastikan tombol manakah yang mereka tekan berdasarkan animasi tersebut. Selain itu dengan fungsi *SendMessage* yang dimiliki kelas ini, tombol yang mengimplementasikan kelas ini dapat memberikan perintah kepada kelas yang diimplementasikan oleh objek Unity lain yang berada pada *scene* yang sama untuk melaksanakan salah satu *method* yang dimiliki kelas tersebut. Pemakaian fungsi *SendMessage* ini

diterapkan untuk memerintahkan kelas *ScreenTransitionManager.cs* untuk melakukan pemberian efek transisi antar *scene* Unity yaitu efek *fade out* dan efek *fade in*. Perpindahan *scene* baru dilakukan setelah efek transisi *fade out* dan *fade in* selesai dilakukan.

4.3.2. Kelas *ButtonScriptDungeon.cs*

Kelas *ButtonScriptDungeon.cs* adalah kelas yang diimplementasikan pada tombol di *scene dungeonMenu* agar tombol dapat meminta *DataController* mengakses data pemain sebelum memasuki *dungeon*. Kelas ini juga memerintahkan kelas *DataController.cs* untuk menyimpan data musuh yang akan dihadapi oleh pemain di tempat yang dituju.

4.3.3. Kelas *ButtonScriptShop.cs*

Kelas *ButtonScriptShop.cs* adalah kelas yang diimplementasikan pada tombol di *scene Shop* agar tombol dapat meminta layar *Shop* tambahan yang memuat monster yang dapat dibeli dan dijual di *Shop*. Tombol ini mengaktifkan bagian kedua dari *scene Shop* sehingga tombol untuk membeli dan menjual monster dapat tampil pada layar pemain. Terdapat tiga *state* pada kelas ini, *normalState*, *buyState*, dan *sellState*. *State normalState* akan memicu pengaktifan tombol-tombol yang berada pada layar awal *Shop*. *State buyState* akan memicu pengaktifan tombol-tombol yang berada pada layar pembelian monster baru. Sedangkan *state sellState* akan memicu pengaktifan tombol-tombol yang berada pada layar penjualan monster milik pemain.

4.3.4. Kelas *ButtonScriptLogin.cs*

Kelas ini adalah sebuah kelas yang diimplementasikan pada *game object* Unity yang berupa tombol di layar *title*, sehingga ketika pemain *login* ke dalam *game* pemain sekaligus mengambil data pemain dari *database*. Kelas ini akan memeriksa apakah pemain sudah memiliki *username* atau belum. Apabila pemain belum

memiliki *username* maka pemain akan diarahkan menuju layar *Register*. Jika pemain telah memiliki *username* maka pemain akan langsung diarahkan menuju *scene Home* untuk memulai permainan setelah data milik pemain tersebut diambil dari penyimpanan.

4.3.5. Kelas *Fader.cs*

Kelas *Fader.cs* adalah sebuah kelas yang diimplementasikan pada *game object* Unity yang memiliki render hitam untuk keperluan *fade in* dan *fade out* dari kelas *ScreenTransitionManager.cs* yang telah dijelaskan pada bagian sebelumnya. Objek yang mengimplementasikan *Fader.cs* tidak akan di-*destroy* walaupun terjadi pergantian *scene* karena bagian *DontDestroyOnLoad* milik *Fader.cs*, karena itu objek ini digunakan di sepanjang permainan sebagai tirai *fade in* dan *fade out* untuk layar.

4.4. Implementasi Antarmuka Pengguna

Implementasi tampilan pengguna dibuat pada Unity dengan menggunakan fitur-fitur tampilan Unity seperti *scene*. Berikut ini akan dijelaskan mengenai implementasi tampilan antarmuka yang ditampilkan oleh permainan *Heart Meister*.

4.4.1. Tampilan *Scene Home*

Scene Home merupakan *scene* yang pertama ditemui pemain setelah pemain memasuki permainan. Pada *scene* ini terdapat beberapa tombol untuk memasuki *scene* lainnya, yaitu *scene Status*, *scene Shop*, *scene Volume*, dan *scene Dungeon*. Tampilan *scene Home* dapat dilihat pada Gambar 4.1.

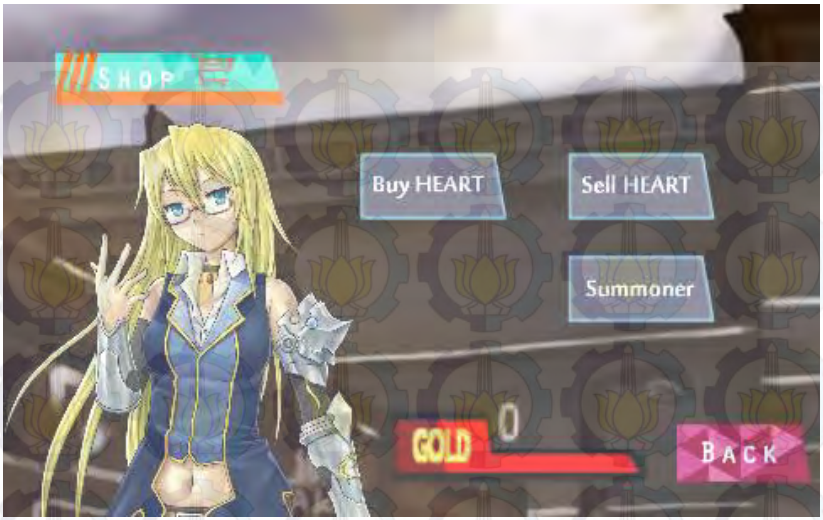
Gambar 4.1. Tampilan *Scene Home*

4.4.2. Tampilan *Scene Shop*

Scene Shop adalah *scene* dimana pemain bisa membeli monster, menjual monster dan mencoba undian untuk mendapatkan monster baru. Pada *scene* ini uang pemain juga ditunjukkan. Tampilan *scene Shop* dapat dilihat pada Gambar 4.2.

4.4.3. Tampilan *Scene Battle*

Scene Battle adalah tempat dimana pemain dapat melihat pertarungan antara monster milik pemain dengan monster musuh yang bergerak secara otomatis. Di kiri bawah layar terdapat *text box* yang berisi pengumuman tentang apa yang terjadi pada layar pertarungan. Tampilan *scene Battle* dapat dilihat pada Gambar 4.3.



Gambar 4.2. Tampilan *Scene Shop*

Gambar 4.3. Tampilan *Scene Battle*

4.5. Cara Pemakaian *Class Library*

Dalam *project* Unity yang ingin menggunakan *class library* ini, pengembang hanya perlu memasukan seluruh *folder MainGameplay* yang berisi kode *class library* ke dalam folder *Assets* miliknya. Pada layar pertama yang akan dibuka oleh Unity, diperlukan empat objek Unity untuk jalannya *class library*. Keeempat objek tersebut harus dinamakan *RaeyDummyDB*, *RaeyScreenManager*, *RaeySoundManager*, dan *ScreenFader*.

Script DummyDB.cs diimplementasikan pada objek *RaeyDummyDB*. *Script ScreenTransitionManager.cs* diimplementasikan pada objek *RaeyScreenManager*. *SoundManager.cs* diimplementasikan pada objek *RaeySoundManager*. Sebuah *GUITexture* sebesar layar berwarna hitam diberikan pada objek *ScreenFader*, dilengkapi dengan *script Fader.cs*. Setiap *scene* selain *scene* pertama harus memiliki objek *BGM* yang diisikan *script SoundChanger.cs* beserta lagu milik *scene* tersebut. Untuk tombol perpindahan digunakan *ButtonScript.cs*. *BattleInitializer.cs* diimplementasikan pada *scene* yang memuat pertarungan. Pada layar *LoadingScreen* yang dikehendaki digunakan kode *LoaderBehavior.cs*. Setiap kali objek dalam permainan ingin mengakses data pemain atau monster cukup hubungi kelas *DataController.cs* untuk mendapatkan data pemain. Untuk layar *Shop*, *script* yang digunakan adalah *ShopLoader.cs*. Tombol-tombol *Shop* menggunakan *ButtonScriptShop.cs*.

Pada setiap *ButtonScript* masukkan *scene* tujuan yang ingin dicapai. Dengan memilih target *EndScene*, maka setiap kali *ButtonScript* melakukan transisi layar efek transisi akan dilaksanakan.

BAB V
PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dan evaluasi pada *class library* yang dikembangkan. Kelas-kelas yang akan diuji adalah kelas pada kelompok *HMCL*. Pengujian fungsionalitas mengacu pada kasus penggunaan pada bab tiga. Pengujian yang dilakukan adalah pengujian terhadap kebutuhan fungsionalitas sistem dengan menggunakan metode *black box*. Hasil evaluasi menjabarkan tentang rangkuman hasil pengujian pada bagian akhir bab ini.

5.1. Lingkungan Pengujian

Lingkungan pengujian sistem pada pengerjaan tugas akhir ini dilakukan pada lingkungan dan alat kakas sebagai berikut:

Jenis Device : ACER Aspire 4755G
Prosesor : Intel i7-2670QM Processor
Memori : 4GB RAM
Kartu Grafis : Intel HD Graphic
Sistem Operasi : Windows 7

5.2. Skenario Pengujian

Pada bagian ini akan dijelaskan tentang skenario pengujian yang dilakukan. Pengujian yang dilakukan adalah pengujian kebutuhan fungsionalitas dengan menggunakan metode *black box testing*.

5.2.1. Pengujian Fungsionalitas

Pengujian kebutuhan fungsionalitas sistem dilakukan dengan menyiapkan sejumlah skenario pengujian sebagai tolak ukur keberhasilan pengujian. Subbab 3.1.4. yang telah menjabarkan fungsional sistem dijadikan acuan untuk pengujian ini. Pengujian pada kebutuhan fungsionalitas dijabarkan pada subbab berikut.

5.2.1.1. Pengujian Fitur Pengatur Transisi Antar *Scene* Unity

Pengujian fitur pengatur transisi antar *scene* Unity bertujuan menguji apakah *ScreenTransitionManager* dapat melakukan transisi antar *scene* dengan baik sambil memberikan efek *fade out* dan *fade in* pada perpindahan. Skenario pengujian adalah dengan cara menekan tombol *Shop* pada *scene Home*, lalu memperhatikan perpindahan *scene* dari *Home* ke *Shop* hingga selesai. Melalui pengujian ini dapat diuji apakah kelas *HMScreenTransitionManager*, *ScreenTransitionManager*, *LoaderBehavior*, dan *ButtonScript* berjalan sebagaimana mestinya. Pada tahap ini dilihat apakah perpindahan *scene* tersebut disertai oleh pemberian efek *fade out* dan *fade in* sehingga menghasilkan transisi yang benar. Hasil pengujian fitur ini dapat dilihat pada Tabel 5.1.

Tabel 5.1. Pengujian Fitur Pengatur Transisi Antar *Scene* Unity

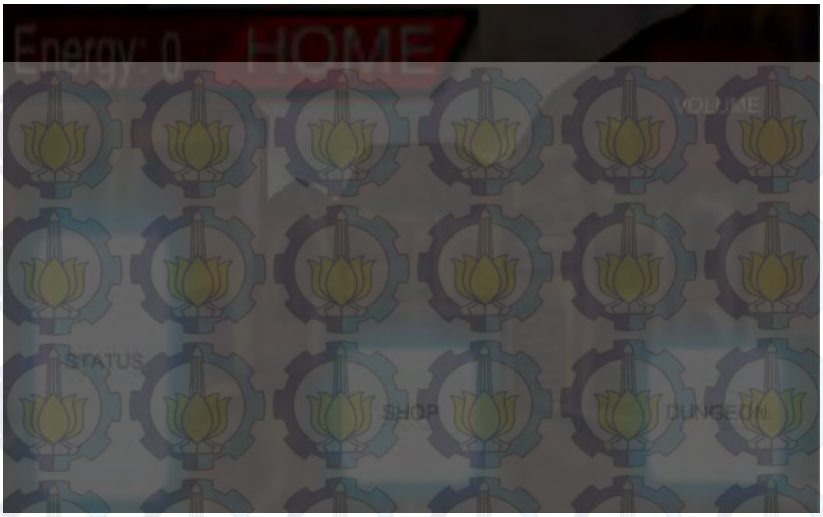
ID	**UJ.HM-001**
Referensi Fungsionalitas	1
Nama	Pengujian Fitur Pengatur Transisi Antar *Scene* Unity
Tujuan Pengujian	Menguji fitur pengatur transisi antar *scene* Unity beserta hasilnya.
Skenario 1	Pengguna menekan tombol *Shop* pada *scene Home*.
Kondisi Awal	Aplikasi menampilkan menu dari *scene Home* seperti ditunjukan pada Gambar 5.1.
Masukan	Sentuhan pada layar.
Hasil yang Diharapkan	Efek *fade out* dan *fade in* dapat diperlihatkan pada saat transisi *scene* dilaksanakan.
Hasil yang Didapat	Efek *fade out* dan *fade in* dapat diperlihatkan pada saat transisi scene dilaksanakan.

Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil pengujian dapat dilihat pada Gambar 5.2., Gambar 5.3., Gambar 5.4. dan Gambar 5.5.

Gambar 5.1. Kondisi Awal Saat Pengguna Berada pada *Scene Home*

5.2.1.2. Pengujian Fitur Pengatur Transisi Antar Musik Latar

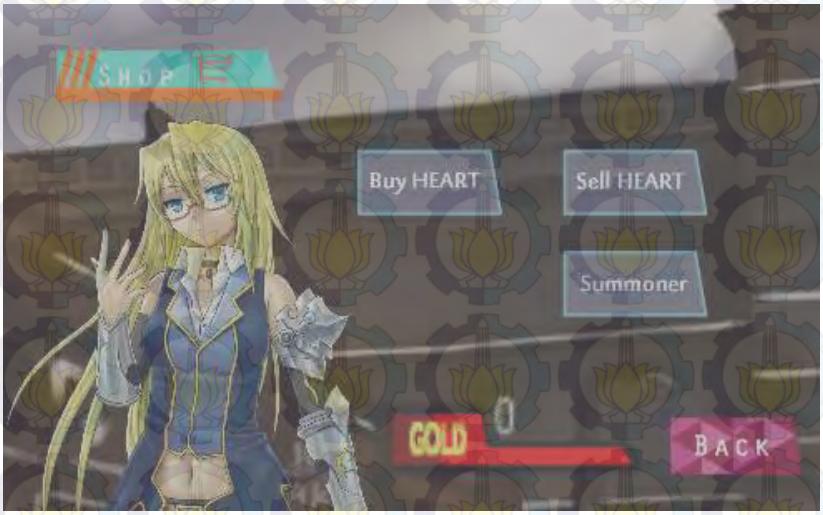
Pengujian fitur pengatur transisi antar musik latar bertujuan menguji apakah *SoundManager* dan *SoundChanger* dapat melakukan transisi antar musik latar dengan baik sambil memeriksa apakah musik latar yang baru sama atau tidak. Jika sama maka musik latar lama akan tetap diputar tanpa diputus. Skenario pengujian adalah dengan cara menekan tombol *Dungeon* pada scene *Home*, lalu memperhatikan perpindahan scene dari *Home* ke *DungeonMenu* hingga selesai. Melalui pengujian ini kebenaran dari kerja kelas *SoundManager* dan *SoundChanger* dapat diuji.



Gambar 5.2. ***Fade Out*** **Saat Perpindahan** ***Scene*** **Dimulai**

Gambar 5.3. ***Fade In*** **Saat Memasuki** ***Loading Screen***

Gambar 5.4. *Fade Out* Saat Meninggalkan *Loading Screen*



Gambar 5.5. *Fade In* Saat Tiba di *Scene Shop*, Transisi Berhasil

Pada tahap ini dilihat apakah perpindahan scene tersebut disertai oleh transisi musik latar yang benar karena musik latar dari *scene Home* dan musik latar dari *scene DungeonMenu* berbeda. Hasil pengujian fitur ini dapat dilihat pada Tabel 5.2.

Tabel 5.2. Pengujian Fitur Pengatur Transisi Antar Musik Latar

ID	**UJ.HM-002**
Referensi Fungsionalitas	2
Nama	Pengujian Fitur Pengatur Transisi Antar Musik Latar
Tujuan Pengujian	Menguji fitur pengatur transisi antar musik latar beserta hasilnya.
Skenario 2	Pengguna menekan tombol *Dungeon* pada *scene Home*.
Kondisi Awal	Aplikasi menampilkan menu dari *scene Home* seperti ditunjukan pada Gambar 5.6.
Masukan	Sentuhan pada layar.
Hasil yang Diharapkan	Musik latar berubah karena musik latar *scene* asal dan tujuan berbeda.
Hasil yang Didapat	Musik latar berubah karena musik latar scene asal dan tujuan berbeda.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil pengujian dapat dilihat pada Gambar 5.7.

Gambar 5.6. *Inspector* Menunjukan Musik Latar *Home* di *Scene Home*

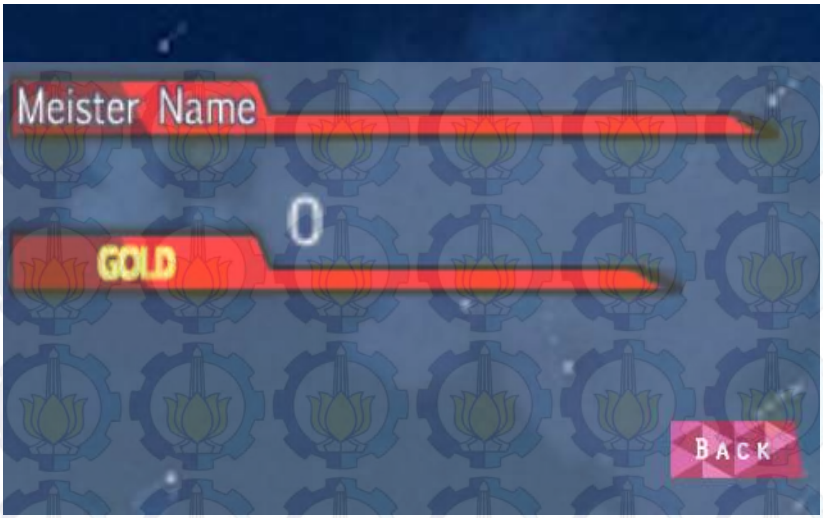
Gambar 5.7. *Inspector* Menunjukkan Perubahan Musik Latar di *Scene* Tujuan

5.2.1.3. Pengujian Fitur *Interface* Antara Unity dan *Web Service*

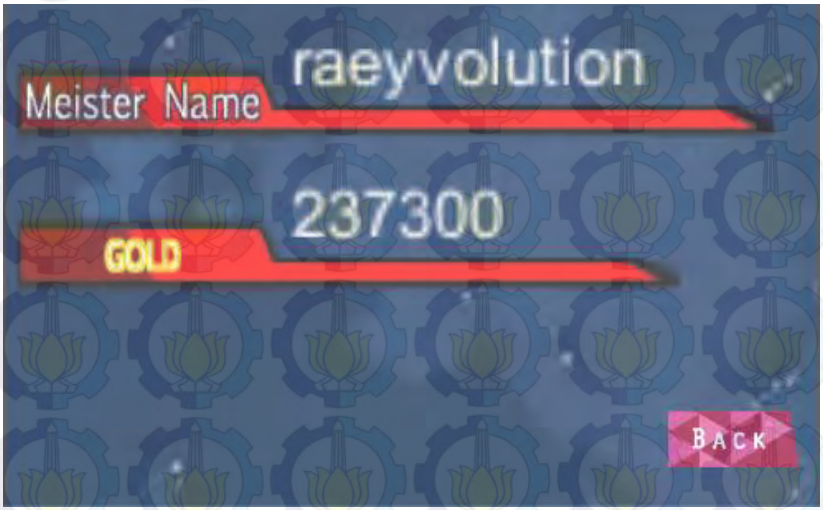
Pengujian fitur *interface* antara Unity dan *web service* bertujuan menguji apakah *IDataConnectionManager* dapat menyampaikan data pemain dari *web service* yang digantikan oleh *DummyDB* atau tidak. Jika berhasil maka pemain akan mempunyai *username* dan uang seperti tertera pada *DummyDB*. Skenario pengujian adalah dengan cara menjalankan aplikasi ketika *IDataConnectionManager* terhubung dengan *DataController* dan sebaliknya. Melalui pengujian ini dapat diuji kerja dari kelas *DataController*, *IDataConnectionManager*, *DummyDB*, dan *StatusLoader*. Pada tahap ini dilihat apakah terdapat perbedaan tampilan pada *scene Status* ketika *IDataConnectionManager* dihubungkan dengan *DataController* atau tidak. Hasil pengujian fitur ini dapat dilihat pada Tabel 5.3.

Tabel 5.3. Pengujian Fitur *Interface* Antara Unity dan *Web Service*

ID	**UJ.HM-003**
Referensi Fungsionalitas	3
Nama	Pengujian Fitur *Interface* Antara Unity dan *Web Service*
Tujuan Pengujian	Menguji fitur *interface* antara Unity dan *web service* agar data terhubung.
Skenario 3	*IDataConnectionManager* tidak dihubungkan dengan *DataController*, lalu dihubungkan.
Kondisi Awal	*IDataConnectionManager* tidak terhubung dengan *DataController*, pengguna ada di layar *Status*.
Masukan	Pengeksekusian fungsi *DataUpdateDownload* pada *DataController*.
Hasil yang Diharapkan	*Username* dan *gold* berubah karena sudah mendapat data dari *web service*.
Hasil yang Didapat	*Username* dan *gold* berubah karena sudah mendapat data dari *web service*.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil pengujian dapat dilihat pada Gambar 5.9.



Gambar 5.8. Data Kosong karena Belum Mendapat Data Pemain dari *Web Service*



Gambar 5.9. Data Pemain Berhasil Ditampilkan Setelah Terhubung

5.2.1.4. Pengujian Fungsi Pengubah Volume Suara

Pengujian fungsi pengubah volume suara bertujuan menguji apakah *VolumeSlider* dan *HMSoundManager* dapat mengubah volume suara dengan tampilan dari *VolumeSlider* dengan benar. Jika volume suara yang dihasilkan oleh permainan secara umum menguat atau melemah sesuai dengan perubahan nilai yang dicantumkan pada *VolumeSlider*, maka pengujian dinyatakan berhasil. Skenario percobaan adalah dengan mengubah posisi *slider* pada *VolumeSlider* lalu mendengarkan volume suara yang dihasilkan. Melalui pengujian ini kerja *VolumeSlider* dan *HMSoundManager* dalam mengubah volume suara dapat diuji. Hasil pengujian fungsi ini dapat dilihat pada Tabel 5.4.

Tabel 5.4. Pengujian Fungsi Pengubah Volume Suara

ID	**UJ.HM-004**
Nama	Pengujian Fungsi Pengubah Volume Suara
Tujuan Pengujian	Menguji fungsi *HMSoundManager* dan *VolumeSlider* dalam mengubah volume.
Skenario 4	Posisi *slider* pada *VolumeSlider* dipindah dan perubahan pada suara didengarkan.
Kondisi Awal	Berada di *scene VolumeSlider*.
Masukan	Posisi *slider* pada *VolumeSlider*.
Hasil yang Diharapkan	Volume musik pada permainan berubah seiring perubahan *slider*.
Hasil yang Didapat	Volume musik pada permainan berubah seiring perubahan *slider*.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil pengujian dapat dilihat pada Gambar 5.11.

Gambar 5.10. Posisi Awal *Slider* pada *VolumeSlider*

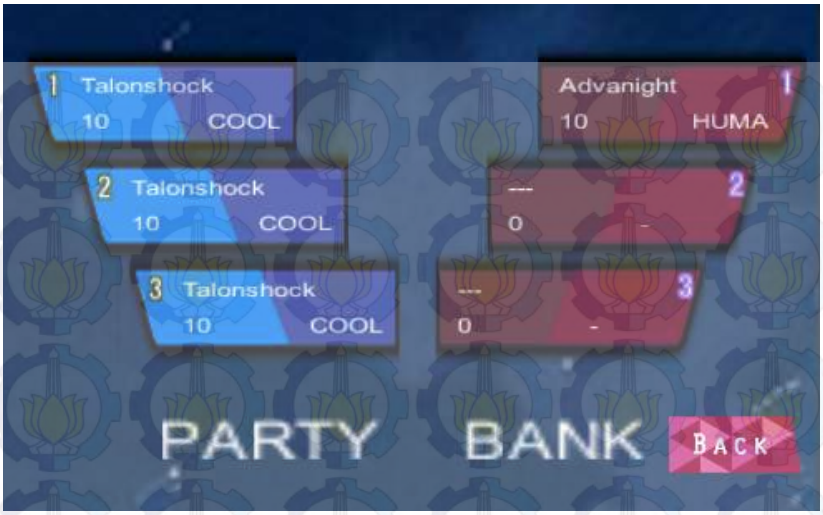
Gambar 5.11. Volume Melemah Setelah *Slider* Digeser ke Kiri

5.2.1.5. Pengujian Fungsi Perubahan Anggota

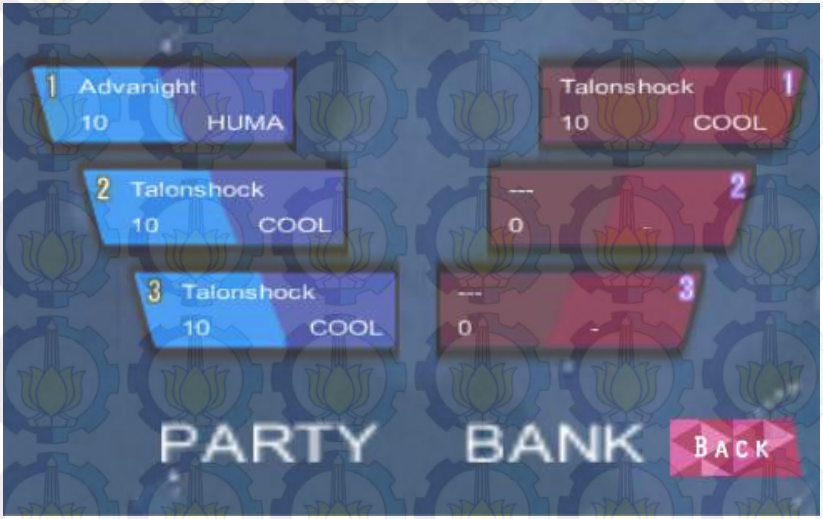
Pengujian fungsi perubahan anggota bertujuan menguji apakah *PartyLoader.cs* dapat menjalankan fungsi untuk mengubah anggota dengan benar. Skenario pengujian ini adalah dengan memilih salah satu anggota, lalu menukarnya dengan anggota yang tersimpan. Pengujian dinyatakan berhasil apabila setelah pemindahan posisi anggota yang ingin ditukar berpindah sesuai perintah pemain. Pengujian ini menguji kerja dari *PartyLoader.cs*. Hasil pengujian fungsi ini dapat dilihat pada Tabel 5.5.

Tabel 5.5. Pengujian Fungsi Perubahan Anggota

ID	**UJ.HM-005**
Nama	Pengujian Fungsi Perubahan Anggota
Tujuan Pengujian	Menguji fungsi *PartyLoader* dalam mengubah posisi anggota.
Skenario 5	Memilih salah satu anggota lalu anggota lainnya untuk memindahkannya.
Kondisi Awal	Berada di *scene PartyScreen*.
Masukan	Posisi monster pertama dan kedua.
Hasil yang Diharapkan	Posisi monster pertama dan kedua ditukar.
Hasil yang Didapat	Posisi monster pertama dan kedua ditukar.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil pengujian dapat dilihat pada Gambar 5.13.



Gambar 5.12. Layar *PartyScreen* dan Anggota Monster Pemain



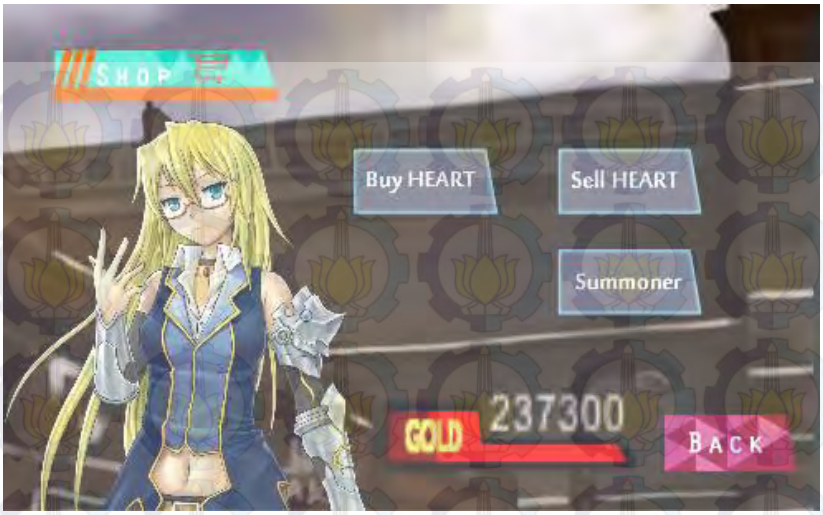
Gambar 5.13. Pemindahan Anggota Berhasil Dilakukan

5.2.1.6. Pengujian Fungsi Layar *Shop*

Pengujian fungsi layar *Shop* memiliki tujuan untuk menguji apakah layar *Shop* bekerja dengan sebagaimana mestinya. Skenario pengujian adalah dengan membuka layar pembelian dan penjualan monster lalu mencoba penggunaan fungsi masing-masing. Pengujian dinyatakan berhasil apabila pembelian dan penjualan berhasil dilakukan oleh pengguna. Yang diuji pada pengujian ini adalah kerja dari *ShopLoader* dan *ButtonScriptShop*. Hasil pengujian fungsi ini dapat dilihat pada Tabel 5.6.

Tabel 5.6. Pengujian Fungsi Layar *Shop*

ID	**UJ.HM-006**
Nama	Pengujian Fungsi Layar *Shop*
Tujuan Pengujian	Menguji fungsi *Shop* dan *ButtonScriptShop* dalam menjalankan layar *Shop*.
Skenario 6	Mencoba fungsi *Buy* dan *Sell* dari layar *Shop*.
Kondisi Awal	Berada di *scene Shop*.
Masukan	Monster yang ingin dibeli atau dijual.
Hasil yang Diharapkan	Monster baru dapat dibeli dan monster lama dapat dijual.
Hasil yang Didapat	Monster baru dapat dibeli dan monster lama dapat dijual.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil pengujian dapat dilihat pada Gambar 5.15 dan Gambar 5.16.



Gambar 5.14. Layar Awal *Shop*

Gambar 5.15. Layar *Buy* dari *Shop*

Gambar 5.16. Layar *Sell* dari *Shop*

5.2.1.7. Pengujian Fungsi Layar *Battle*

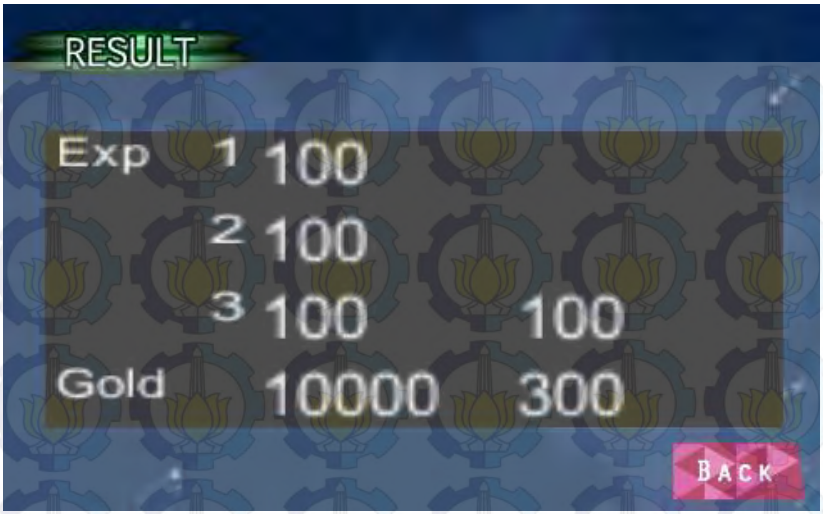
Pengujian fungsi layar *Battle* bertujuan untuk menguji apakah layar *Battle* dapat bekerja sesuai dengan desain awal. Skenario pengujian adalah dengan masuk ke layar *DungeonMenu* lalu masuk ke layar *Battle* hingga mencapai layar *Result*. Pengujian dinyatakan berhasil apabila pertarungan berlangsung hingga selesai dan layar *Result* berhasil dicapai. Pengujian ini menguji kerja dari *ButtonScriptDungeon*, *BattleInitializer*, dan *Result*. Hasil pengujian fungsi ini dapat dilihat pada Tabel 5.7.

Tabel 5.7. Pengujian Fungsi Layar *Battle*

ID	**UJ.HM-007**
Nama	Pengujian Fungsi Layar *Battle*
Tujuan Pengujian	Menguji fungsi *ButtonScriptDungeon*, *BattleInitializer*, dan *Result* dalam menampilkan pertarungan.

Skenario 7	Mencoba fungsi *Battle* hingga layar hasil.
Kondisi Awal	Berada di *scene DungeonMenu*.
Masukan	Data monster pemain dan musuh.
Hasil yang Diharapkan	Pertarungan berjalan hingga layar hasil.
Hasil yang Didapat	Pertarungan berjalan hingga layar hasil.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil pengujian dapat dilihat pada Gambar 5.18.

Gambar 5.17. Layar Pertarungan



Gambar 5.18. Layar Hasil Pertarungan

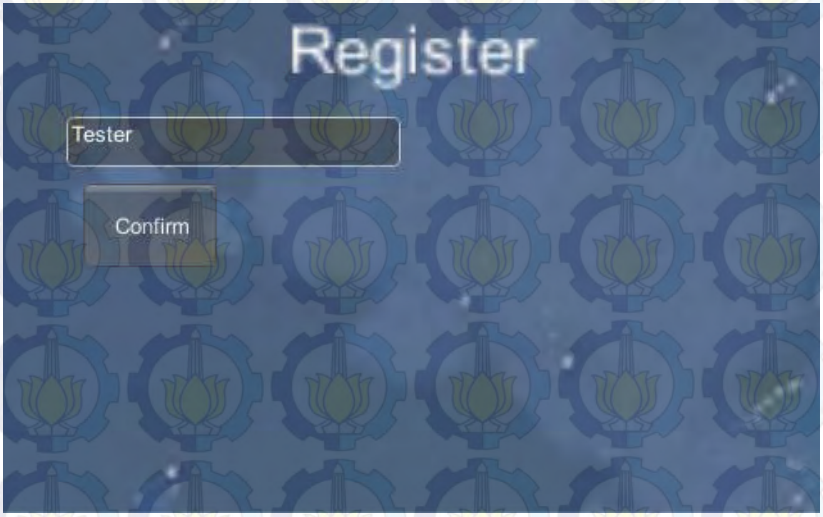
5.2.1.8. Pengujian Fungsi *Register*

Pengujian fungsi *Register* memiliki tujuan untuk menguji apakah kelas *Register* dapat mendaftarkan *username* pemain. Skenario pengujian adalah dengan memasukkan *username* pada layar *Register* lalu melihat apakah *username* milik pemain telah berubah seperti yang diinginkan. Pengujian dinyatakan berhasil apabila *username* di *scene Register* berubah sesuai dengan *username* yang dimasukkan pada layar *Register* sebelumnya. Pengujian ini menguji kerja dari *ButtonScriptLogin* dan *Register*. Hasil pengujian fungsi ini dapat dilihat pada Tabel 5.8.

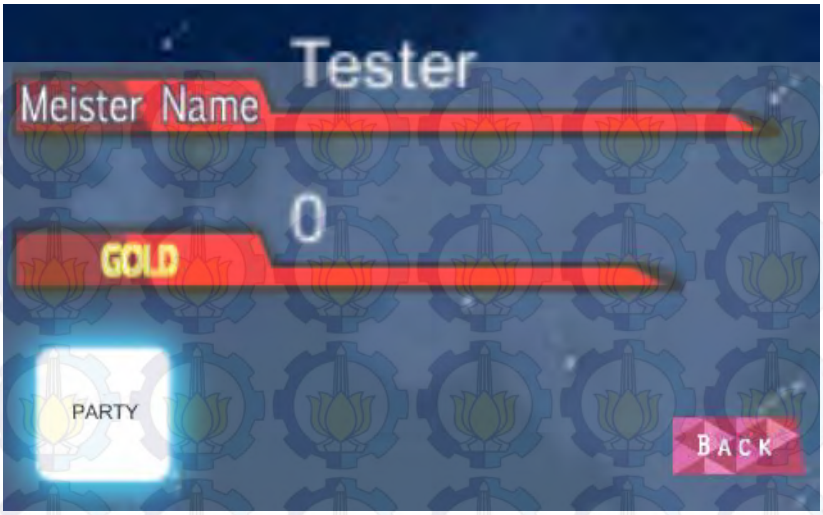
Tabel 5.8. Pengujian Fungsi *Register*

ID	**UJ.HM-008**
Nama	Pengujian Fungsi *Register*
Tujuan Pengujian	Menguji fungsi *Register* dalam mendaftarkan *username*.

Skenario 8	Mencoba fungsi *Register* dan melihat hasilnya pada halaman *Status*.
Kondisi Awal	Berada di *scene Register*.
Masukan	*Username* yang diinginkan pemain.
Hasil yang Diharapkan	*Username* yang dipilih dapat disimpan dan ditampilkan pada layar *Status*.
Hasil yang Didapat	*Username* yang dipilih dapat disimpan dan ditampilkan pada layar *Status*.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil pengujian dapat dilihat pada Gambar 5.20.



Gambar 5.19. Layar *Register* dan *Username* yang Dimasukkan



Gambar 5.20. Layar *Status* Menunjukkan *Username* Sesuai dengan yang Dimasukkan Sebelumnya

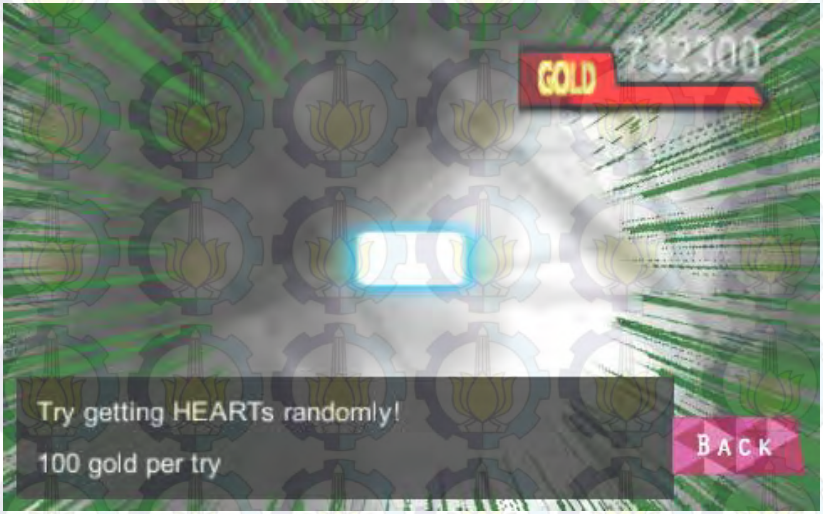
5.2.1.9. Pengujian Fungsi *Summoner*

Pengujian fungsi *Summoner* bertujuan untuk menguji apakah kelas *Summoner* dapat membuat undian untuk mendapatkan monster.seperti yang diharapkan. Skenario pengujian adalah dengan menekan tombol yang ada pada layar *Summoner*, lalu melihat hasil yang didapatkan. Pengujian dinyatakan berhasil apabila *Summoner* berhasil memberikan sebuah monster baru secara acak pada pemain. Pengujian ini menguji kerja dari *Summoner*. Hasil pengujian fungsi ini dapat dilihat pada Tabel 5.9.

Tabel 5.9. Pengujian Fungsi *Summoner*

ID	**UJ.HM-009**
Nama	Pengujian Fungsi *Summoner*
Tujuan Pengujian	Menguji fungsi *Summoner* dalam mengundi monster.

Skenario 9	Menjalankan fungsi mengundi monster milik *Summoner*.
Kondisi Awal	Berada di *scene Summoner*.
Masukan	Jumlah *gold* dan *slot* kosong yang dimiliki pemain.
Hasil yang Diharapkan	Undian berhasil menambahkan monster.
Hasil yang Didapat	Undian berhasil menambahkan monster.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan hasil pengujian dapat dilihat pada Gambar 5.22.



Gambar 5.21. Tampilan Awal *Scene Summoner*

Gambar 5.22. Pemain Mendapatkan *Thunderbird* dari Undian

5.2.1.10. Pengujian Pengatur Aturan Main Pertarungan

Pengujian pengatur aturan main pertarungan bertujuan menguji apakah *class library* dapat menjalankan aturan main pertarungan dengan baik dan benar. Aturan main yang akan diuji disini antara lain adalah aturan main pertarungan berjalan otomatis, aturan main kalkulasi *damage*, dan aturan main menang kalah pertarungan. Pengujian dilakukan dengan pembuatan kelas *test* secara manual yang memiliki fungsi-fungsi untuk mengetes aturan main tersebut.

Pengujian terhadap aturan main pertarungan berjalan otomatis dilakukan dengan cara penjalankan layar *BattleTest* dan mencoba memberikan input berupa sentuhan dari pengguna di berbagai lokasi pada layar selain pada tombol *test*. Pengujian dianggap sukses apabila seluruh input sentuhan dari pemain tidak mempengaruhi jalannya pertarungan. Hasil pengujian dari bagian aturan main pertarungan berjalan secara otomatis ini dapat dilihat pada Tabel 5.10.

Tabel 5.10. Pengujian Aturan Main Pertarungan Otomatis

ID	**UJ.HM-0010**
Nama	Pengujian Aturan Main Pertarungan Otomatis
Tujuan Pengujian	Menguji aturan main pertarungan berjalan secara otomatis.
Skenario 10	Menjalankan *scene BattleTest* dan memberikan input sentuhan di tengah pertarungan.
Kondisi Awal	Berada di *scene BattleTest*.
Masukan	Sentuhan pada layar.
Hasil yang Diharapkan	Pertarungan tidak terpengaruh masukan sentuhan.
Hasil yang Didapat	Pertarungan tidak terpengaruh masukan sentuhan.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan *scene BattleTest* dapat dilihat pada Gambar 5.23.

Pengujian terhadap aturan main kalkulasi *damage* dilakukan dengan menambahkan *Debug.Log* yang menunjukkan nilai dari variabel yang diinginkan selama *debugging* dalam *editor* Unity. *Debug.Log* disini ditambahkan untuk menampilkan nilai dari *damage*, penyerang dan target serangan, untuk menguji apakah yang ditampilkan pada tampilan *scene Battle* benar seperti yang dikalkulasikan dalam kode sumber. Pengujian dianggap berhasil apabila kalkulasi, penyerang dan target serangan yang ditampilkan oleh *scene* sama seperti yang ditampilkan oleh *Debug.Log*. Hasil dari pengujian ini dapat dilihat pada Tabel 5.11.

Gambar 5.23. Pertarungan Terus Berjalan pada *Scene BattleTest*

Tabel 5.11. Pengujian Aturan Main Kalkulasi *Damage*

ID	**UJ.HM-0011**
Nama	Pengujian Aturan Main Kalkulasi *Damage*
Tujuan Pengujian	Menguji kebenaran hitungan dan tampilan aturan main kalkulasi *damage*.
Skenario 11	Menjalankan *scene BattleTest* dan mengecek hasil *output* dari *Debug.Log*.
Kondisi Awal	Berada di *scene BattleTest*.
Masukan	Status monster penyerang dan diserang.
Hasil yang Diharapkan	Tampilan pada *scene* dan *Debug.Log* sama.
Hasil yang Didapat	Tampilan pada *scene* dan *Debug.Log* sama.

Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan *scene BattleTest* dan *Debug.Log* dapat dilihat pada Gambar 5.24.

Gambar 5.24. Perbandingan Tampilan *Scene* dengan *Debug.Log*

Pengujian terhadap aturan main menang dan kalah pertarungan dilakukan dengan menggunakan dua tombol yang dibuat di tengah *scene BattleTest*. Pengecekan menang kalah dilakukan pada saat salah satu tim kehabisan seluruh *health point* atau *HP* milik semua monsternya. Tombol *win* akan mengubah *HP* dari seluruh musuh menjadi nol, yang seharusnya memicu kemenangan pemain. Sebaliknya tombol *lose* akan mengubah *HP* dari seluruh monster pemain menjadi nol, yang seharusnya memicu kekalahan pemain. Hasil dari pengujian terhadap aturan main kondisi menang kalah pertarungan dapat dilihat pada Tabel 5.12.

Tabel 5.12. Pengujian Kondisi Menang Kalah Pertarungan

ID	UJ.HM-0012
Nama	Pengujian Aturan Main Kondisi Menang Kalah Pertarungan
Tujuan Pengujian	Menguji kebenaran pemeriksaan kondisi menang kalah pada pertarungan.
Skenario 12	Menekan tombol *win*, lalu menekan tombol *lose* pada pertarungan berikutnya.
Kondisi Awal	Berada di *scene BattleTest*.
Masukan	Tombol *win* dan *lose* yang mengubah *HP* monster dalam pertarungan.
Hasil yang Diharapkan	Pemain menang ketika *win* ditekan, pemain kalah ketika *lose* ditekan.
Hasil yang Didapat	Pemain menang ketika *win* ditekan, pemain kalah ketika *lose* ditekan.
Hasil Pengujian	Berhasil
Kondisi Akhir	Tampilan menang dapat dilihat pada Gambar 5.25. Tampilan kalah dapat dilihat pada Gambar 5.26.

5.2.1.11. Pengujian Aturan Main Batasan Monster

Pengujian aturan main batasan monster bertujuan menguji apakah pemain dapat memiliki kurang dari tiga monster dalam *party* dan memiliki lebih dari enam monster. Skenario pengujian pertama adalah dengan mencoba mengeluarkan salah satu anggota *party* ke kolom cadangan. Skenario dianggap berhasil apabila pemain tidak dapat menukar monster dengan *slot* kosong pada kolom cadangan. Skenario berikutnya adalah dengan mencoba menjual monster yang ada pada *slot party*. Skenario ini dianggap berhasil apabila pemain tidak bisa menjual, sehingga jumlah monster di *party* tetap tiga.

Gambar 5.25. Pemain Menang Setelah Tombol *Win* Ditekan

Gambar 5.26. Pemain Kalah Setelah Tombol *Lose* Ditekan

Berikutnya adalah skenario pemain berusaha membeli monster baru lewat layar *Shop* atau *Summoner*. Skenario dianggap berhasil apabila tombol pembelian monster di *Shop* dan *Summoner* tidak responsif sehingga jumlah monster pemain tidak bertambah.

Setelah pengujian skenario pertama, didapatkan bahwa pemain tidak bisa menukar monster dalam party dengan *slot* kosong dalam kolom cadangan. Skenario kedua mendapatkan hasil bahwa pemain tidak dapat menjual monster dalam *party* karena layar penjualan hanya menampilkan monster pada kolom cadangan. Kedua skenario ini membuktikan bahwa pemain pasti memiliki tiga monster dalam *party* miliknya. Hasil pengujian kedua skenario ini dapat dilihat pada Tabel 5.13 dan 5.14.

Setelah pengujian skenario ketiga didapatkan hasil bahwa tombol pembelian monster pada *Shop* dan *Summoner* tidak responsif ketika pemain telah memiliki enam monster. Hal ini membuktikan bahwa pemain tidak dapat memiliki lebih dari enam monster seperti yang diterakan dalam aturan main permainan sebelumnya. Hasil pengujian dapat dilihat pada Tabel 5.15.

Tabel 5.13. Pengujian Aturan Main Batasan *Party* Tiga Monster

ID	**UJ.HM-0013**
Nama	Pengujian Aturan Main Batasan *Party* Tiga Monster
Tujuan Pengujian	Menguji kebenaran *party* pemain selalu terdiri dari tiga monster.
Skenario 13	Mencoba menukar monster *party* dengan *slot* kosong pada kolom cadangan.
Kondisi Awal	Berada di *scene PartyScreen*, terdapat *slot* kosong pada kolom cadangan pemain.
Masukan	Penekanan tombol monster *party* dan *slot* kosong dalam kolom cadangan.
Hasil yang Diharapkan	Tombol *slot* kosong non-responsif sehingga monster tidak keluar dari *party*.

Hasil yang Didapat	Tombol *slot* kosong non-responsif sehingga monster tidak keluar dari *party*.
Hasil Pengujian	Berhasil
Kondisi Akhir	Monster tidak keluar *party* dapat dilihat pada Gambar 5.27.

Gambar 5.27. Monster Tidak Dapat Dipindahkan ke *Slot* Kosong

Tabel 5.14. Pengujian Aturan Main Batasan *Party* Penjualan Monster

ID	**UJ.HM-0014**
Nama	Pengujian Aturan Main Batasan *Party* Penjualan Monster
Tujuan Pengujian	Menguji kebenaran *party* pemain selalu terdiri dari tiga monster dengan mencoba menjual monster.
Skenario 14	Mencoba menjual monster yang ada pada *party* pemain.

Kondisi Awal	Berada di *scene Shop*.
Masukan	Penekanan tombol *Sell* untuk masuk ke layar penjualan monster di *Shop*.
Hasil yang Diharapkan	Tidak bisa dijualnya monster *party*.
Hasil yang Didapat	Tidak bisa dijualnya monster *party*.
Hasil Pengujian	Berhasil
Kondisi Akhir	Monster yang dapat dijual hanyalah monster cadangan seperti dapat dilihat pada Gambar 5.28.

Gambar 5.28. Layar Penjualan Monster Hanya Menampilkan Cadangan

Tabel 5.15. Pengujian Aturan Main Batasan *Party* Pembelian Monster

ID	UJ.HM-0015
Nama	Pengujian Aturan Main Batasan *Party* Pembelian Monster
Tujuan Pengujian	Menguji kebenaran *party* pemain paling banyak enam monster karena tidak bisa membeli monster baru lagi.
Skenario 15	Mencoba membeli dan mengundi monster baru saat sudah memiliki enam monster.
Kondisi Awal	Berada di *scene Shop*.
Masukan	Penekanan tombol *Buy* untuk masuk ke layar pembelian monster dan *Summoner* di *Shop*.
Hasil yang Diharapkan	Tidak bisa membeli monster baru karena tombol non-responsif.
Hasil yang Didapat	Tidak bisa membeli monster baru karena tombol non-responsif.
Hasil Pengujian	Berhasil
Kondisi Akhir	Ketika sudah memiliki enam monster tombol *Buy* dan *Summoner* non-responsif seperti dapat dilihat pada Gambar 5.29 dan Gambar 5.30.

5.2.1.12. Pengujian Aturan Main *Energy* Pemain

Pengujian aturan main *energy* pemain ditujukan untuk menguji apakah aturan main pemakaian *energy* telah diterapkan dengan benar. *Energy* akan terpakai sebanyak 5 apabila pemain memasuki *dungeon*, dan terpakai sebanyak 10 apabila pemain memasuki *epic battle*. Skenario dianggap berhasil apabila setelah pemain memilih *epic battle* jumlah *energy* pemain berkurang sebanyak 10. Hasil pengujian dapat dilihat pada Tabel 5.16.

Gambar 5.29. Tombol Pembelian Non-Responsif karena Pemain Sudah Memiliki Enam Monster

Gambar 5.30. Tombol *Summoner* Non-Responsif karena Pemain Sudah Memiliki Enam Monster

Tabel 5.16. Pengujian Aturan Main *Energy* Pemain

ID	**UJ.HM-0016**
Nama	Pengujian Aturan Main *Energy* Pemain
Tujuan Pengujian	Menguji implementasi aturan main penggunaan *energy* pemain.
Skenario 16	Memasuki *epic battle* dan kembali ke *Home* untuk melihat sisa *energy*.
Kondisi Awal	Berada di *scene Home* dengan 50 *energy*.
Masukan	Penekanan tombol *epic battle*.
Hasil yang Diharapkan	*Energy* pemain tersisa 40 karena 10 dipakai untuk masuk *epic battle*.
Hasil yang Didapat	*Energy* pemain tersisa 40 karena 10 dipakai untuk masuk *epic battle*.
Hasil Pengujian	Berhasil
Kondisi Akhir	*Energy* pemain tersisa 40 seperti dapat dilihat pada Gambar 5.31.

5.2.1.13. Pengujian Efek Transisi dan Aturan Main Pertarungan pada Modul *Epic Battle*

Pengujian ini ditujukan untuk menguji apakah efek transisi dan aturan main dari *class library* dapat diintegrasikan ke modul *epic battle* yang dikerjakan oleh Nabilah. Skenario pengujian adalah dengan memasuki modul *epic battle* dari *scene Home* milik modul *MainGameplay*. Jika efek transisi layar dan aturan main pertarungan dapat dihubungkan ke modul *epic battle* maka pengujian dinyatakan berhasil. Hasil pengujian dapat dilihat pada Tabel 5.17.

Gambar 5.31. *Energy* Pemain Tersisa 40 Setelah Memasuki *Epic Battle*

Tabel 5.17. Pengujian Efek Transisi dan Aturan Main Pertarungan pada Modul *Epic Battle*

ID	**UJ.HM-0017**
Nama	Pengujian Efek Transisi dan Aturan Main Pertarungan pada Modul *Epic Battle*
Tujuan Pengujian	Menguji pengintegrasian modul *epic battle* ke *class library*.
Skenario 17	Memasuki *epic battle* dan kembali ke *Home* setelah memasuki pertarungan.
Kondisi Awal	Berada di *scene Home*.
Masukan	Penekanan tombol *epic battle*.
Hasil yang Diharapkan	Efek *fade out* dan *fade in* serta pertarungan dapat dijalankan dengan baik pada modul *epic battle*.

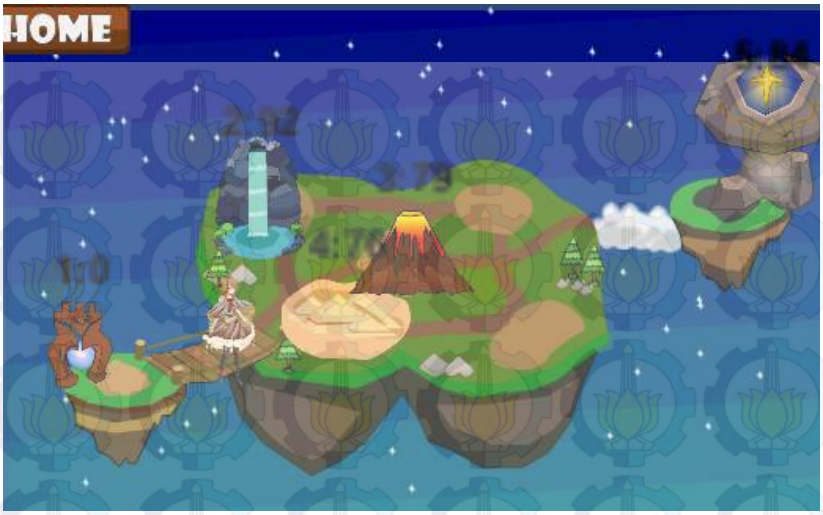
Hasil yang Didapat	Efek *fade out* dan *fade in* serta pertarungan dapat dijalankan dengan baik pada modul *epic battle*.
Hasil Pengujian	Berhasil
Kondisi Akhir	Gambar 5.32 sampai Gambar 5.35 menunjukkan modul *epic battle* berjalan lancar.

Gambar 5.32. Efek *Fade In* Berhasil Diterapkan pada Modul *Epic Battle*

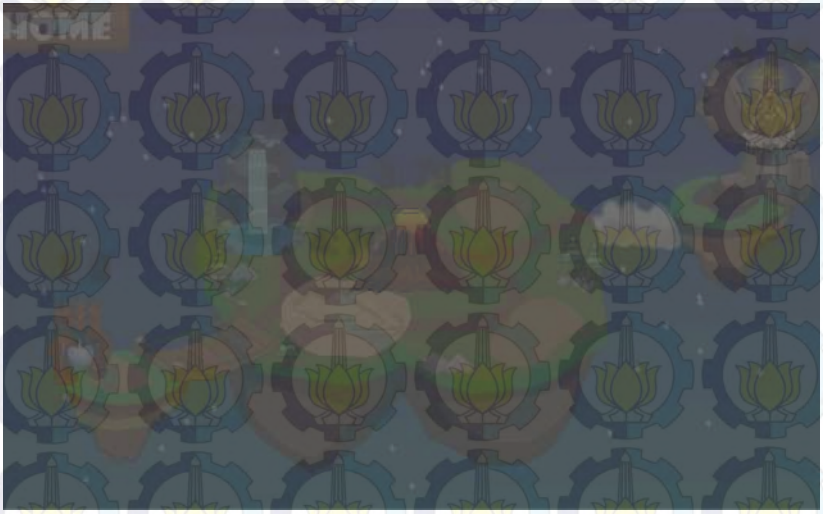
5.3. Evaluasi Hasil Pengujian

Pada subbab ini hasil dari pengujian yang telah dilakukan pada fungsi-fungsi yang ada pada *class library* akan dibahas. Hasil pengujian ini merupakan hasil pengujian secara *black box*.

1. Pengujian fungsi transisi menunjukkan hasil sesuai seperti yang diharapkan. Pada pengujian tersebut transisi antar *scene* Unity berlangsung dengan adanya tambahan *fade in* dan *fade out* dari *ScreenTransitionManager*.



Gambar 5.33. Modul *Epic Battle* Berjalan Normal



Gambar 5.34. Efek *Fade Out* Berhasil Diterapkan Pada Modul *Epic Battle*

Gambar 5.35. *Scene* Berisi Aturan Main Pertarungan Berhasil Dimasuki Lewat Modul *Epic Battle*

2. Pengujian fungsi transisi musik latar menunjukkan hasil sesuai seperti yang diharapkan. Pada pengujian tersebut transisi antar musik latar berlangsung dengan baik pada objek yang sama karena kedua *scene* memiliki musik latar yang berbeda.
3. Pengujian fungsi *interface* antara Unity dengan *web service* berhasil sesuai dengan yang diharapkan. Pada pengujian tersebut dapat terlihat jelas perbedaan yang terjadi ketika *interface IDataConnectionManager* tidak terhubung dengan *DataController* selaku penyimpan data.
4. Pengujian fungsi-fungsi yang termasuk pada aturan main telah sesuai yang diharapkan. Pada seluruh pengujian aturan main dapat dilihat bahwa sebagian besar terhubung kepada kelas-kelas utama permainan *Heart Meister*.
5. Pengujian integrasi modul *epic battle* ke *class library* berjalan lancar. Modul *epic battle* dapat menggunakan fitur transisi layar dan *scene* berisi aturan main pertarungan tanpa kendala.

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

Dalam proses pengerjaan tugas akhir ini dimulai dari pendahuluan, kajian pustaka, analisis, perancangan, implementasi, hingga akhirnya pengujian diperoleh kesimpulan sebagai berikut.

1. *Class library* dapat dipergunakan untuk memuat implementasi aturan main untuk sebuah permainan dengan baik, seperti aturan main pertarungan, batasan monster, dan *energy* pemain.
2. Implementasi dan integrasi modul *epic battle* ke *class library* lebih mudah dicapai dibandingkan modul lainnya karena modul *epic battle* sama-sama berbasis objek Unity.
3. Implementasi dan integrasi modul selain *epic battle* ke *class library* terhambat oleh kesulitan pengimplementasian *class library* sebagai modul terpisah dari Unity. Kesulitan ini disebabkan oleh sebagian besar kelas yang harus diimplementasikan ke objek Unity melalui *MonoBehavior*, sehingga tidak dapat dipisah menjadi modul tersendiri.

6.2. Saran

Berikut ini adalah saran-saran yang ditujukan untuk pengembangan dan perbaikan sistem yang mirip atau sejenis di masa yang akan datang.

1. *Class library* dapat mempermudah pembuatan sebuah permainan, sekaligus pemeliharaan permainan tersebut untuk ke depannya.

2. Unity adalah alat kakas yang terus berkembang, karena itu penggunaan versi yang lebih baru untuk percobaan selanjutnya sangat direkomendasikan agar fitur-fitur baru yang belum ada pada versi sebelumnya dapat mempermudah percobaan tersebut.
3. Ketika ingin mencari referensi tentang bahasa pemrograman yang digunakan pada Unity, pastikan untuk menyertakan kata kunci Unity pada pencarian, karena tidak semua metode yang biasa dipakai pada pemrograman biasa dapat diimplementasikan oleh Unity.
4. Unity hanya mendukung pemakaian *framework* .NET hingga versi 2.0. Pemakaian kelas atau modul yang menggunakan *framework* .NET dengan versi lebih tinggi tidak dianjurkan.

LAMPIRAN

```
using UnityEngine;
using System.Collections;
namespace HMCL
{
    public class HMScreenTransitionManager
    {
        public static string targetName;
        float fadeSpeed = 2f;
        public bool sceneStarting = true;
        public bool sceneEnding = false;
        private GUITexture HMSTMScreenCover;

        public HMScreenTransitionManager()
        {
        }

        public HMScreenTransitionManager(GUITexture
screenCover)
        {
            this.HMSTMScreenCover = screenCover;
        }

        public void FadeClear()
        {
            HMSTMScreenCover.color =
Color.Lerp(HMSTMScreenCover.color, Color.clear, fadeSpeed
* Time.deltaTime);
        }

        public void FadeBlack()
        {
            HMSTMScreenCover.enabled = true;
```

```
            HMSTMScreenCover.color =
Color.Lerp(HMSTMScreenCover.color, Color.black, fadeSpeed
* Time.deltaTime);
        }

        public void HMSTMLoadScene()
        {
            if (HMSTMScreenCover.color.a >= 0.99f)
            {
                Application.LoadLevel("LoadingScreen");
            }
        }

        public void HMSTMStartScene()
        {
            FadeClear();
            if (HMSTMScreenCover.color.a <= 0.01f)
            {
                HMSTMScreenCover.color = Color.clear;
                HMSTMScreenCover.enabled = false;
                sceneStarting = false;
                ButtonScript.buttonsEnabled = true;
            }
        }

        public void HMSTMEndScene(string target)
        {
            targetName = target;
            sceneEnding = true;
            ButtonScript.buttonsEnabled = false;
        }
    }
}
```

Kode Sumber A. Kelas *HMScreenTransitionManager.cs*

```
using UnityEngine;
using System.Collections;
namespace HMCL
{
    public class ScreenTransitionManager : MonoBehaviour
    {
        public static string targetName;
        public GUITexture screenCover;

        public HMScreenTransitionManager hmstm;

        public GameObject stmGO;

        // Use this for initialization
        void Start()
        {
            hmstm = new
HMScreenTransitionManager(screenCover);
        }

        // Update is called once per frame
        void Update()
        {
            if (hmstm.sceneStarting)
            {
                hmstm.HMSTMStartScene();
            }
            if (hmstm.sceneEnding)
            {
                hmstm.FadeBlack();
                hmstm.HMSTMLoadScene();
            }
        }
```

```
    void FadeBlack()
    {
        hmstm.FadeBlack();
    }

    void Awake()
    {
        DontDestroyOnLoad(transform.gameObject);
        screenCover.pixelInset = new Rect(0f, 0f,
Screen.width, Screen.height);
    }

    void OnLevelWasLoaded(int level)
    {
        if (level != 0)
        {
            hmstm.sceneStarting = true;
            hmstm.sceneEnding = false;
        }
    }

    public void EndScene(string target)
    {
        hmstm.HMSTMEndScene(target);
    }
  }
}
```

Kode Sumber B. Kelas *ScreenTransitionManager.cs*

```
using UnityEngine;
using System.Collections;
namespace HMCL
```

```
{

    public class VolumeSlider : MonoBehaviour
    {
        float s = 1.0F;
        public TextMesh amount;

        public HMSoundManager hmsm;

        // Use this for initialization
        void Start()
        {
            hmsm = new HMSoundManager();
        }

        void Update()
        {
            hmsm.VolumeChanger(s, amount);
        }

        void OnGUI()
        {
            s = GUI.HorizontalSlider(new Rect(50, 150, 384, 40), s,
0.0F, 1.0F);
        }
    }
}
```

Kode Sumber C. Kelas *VolumeSlider.cs*

```
public void VolumeChanger(float position, TextMesh amount)
    {
        AudioListener.volume = position;
        amount.text = ((int)(position * 100)).ToString();
    }
```

Kode Sumber D. Fungsi *VolumeChanger* milik *HMSoundManager.cs*

```
public static void DataUpdateDownload()
    {
        playerEnergy = db.PlayerData.playerEnergy;
                        playerMachineID =
db.PlayerData.playerMachineID;
                        playerUsername =
db.PlayerData.playerUsername;
                        playerGold = db.PlayerData.playerGold;
        PartyMember1 = db.PlayerData.PartyMember1;
                        PartyMember2 =
db.PlayerData.PartyMember2;
                        PartyMember3 =
db.PlayerData.PartyMember3;
                        BankMember1 =
db.PlayerData.BankMember1;
                        BankMember2 =
db.PlayerData.BankMember2;
                        BankMember3 =
db.PlayerData.BankMember3;
        EnemyMember1 = db.Enemy1;
        EnemyMember2 = db.Enemy2;
        EnemyMember3 = db.Enemy3;
        PartySlot[0] = PartyMember1;
        PartySlot[1] = PartyMember2;
        PartySlot[2] = PartyMember3;
        PartySlot[3] = BankMember1;
```

```
        PartySlot[4] = BankMember2;
        PartySlot[5] = BankMember3;
    }
```

Kode Sumber E. Fungsi *DataUpdateDownload* pada *DataController.cs*

```
using UnityEngine;
using System.Collections;
namespace MainGameplay
{

    public interface IDataConnectionManager
    {
        int PlayerEnergy
        {
            get;
            set;
        }
        string AndroidMachineID
        {
            get;
            set;
        }
        string StatusPlayerName
        {
            get;
            set;
        }
        int CurrentGold
        {
            get;
            set;
        }
        Battler Member1
```

```
        {
            get;
            set;
        }
        Battler Member2
        {
            get;
            set;
        }
        Battler Member3
        {
            get;
            set;
        }
        Battler Bank1
        {
            get;
            set;
        }
        Battler Bank2
        {
            get;
            set;
        }
        Battler Bank3
        {
            get;
            set;
        }
    }
}
```

Kode Sumber F. ***Interface IDataConnectionManager.cs***

DAFTAR PUSTAKA

[1] Unity, "Unity - create games," Unity3D, [Online]. Available: http://unity3d.com/pages/create-games. [Accessed 1 March 2014].

[2] M. Rouse, "What is class library? - Definition from WhatIs.com," August 2005. [Online]. Available: http://whatis.techtarget.com/definition/class-library. [Accessed 1 March 2014].

[3] Webopedia, "C# Definition," [Online]. Available: http://www.webopedia.com/TERM/C/C_sharp.html. [Accessed 1 March 2014].

[4] Microsoft, "Microsoft Developer Network," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/hbzz1a9a%28v=vs.110%29.aspx. [Accessed 13 December 2014].

[5] Microsoft, "Microsoft Developer Network," Microsoft, April 2003. [Online]. Available: http://msdn.microsoft.com/en-us/library/ms973837.aspx. [Accessed 13 December 2014].

[6] Microsoft, "Microsoft Developer Network," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/aa664632%28v=vs.71%29.aspx. [Accessed 13 December 2014].

[7] Microsoft, "Microsoft Developer Network," Microsoft, [Online]. Available: http://msdn.microsoft.com/en-us/library/aa292172%28v=vs.71%29.aspx. [Accessed 13 December 2014].

[8] N. Lovell, "What is a social game? - Gamesbrief," GAMESbrief, [Online]. Available: http://www.gamesbrief.com/2011/01/what-is-a-social-game/. [Accessed 1 March 2014].

[9] Select Business Solustions, "What is object oriented design," Select Business Solustions, [Online]. Available: http://www.selectbs.com/process-maturity/what-is-object-oriented-design. [Accessed 1 March 2014].

[10] Xamarin, "About Mono | Mono," Xamarin, [Online]. Available: http://www.mono-project.com/docs/about-mono/. [Accessed 13 December 2014].

[11] Unity, "Unity Documentation," Unity, [Online]. Available: http://docs.unity3d.com/ScriptReference/MonoBehaviour.html. [Accessed 13 December 2014].

[12] Verizon, "Android OS FAQs," Verizon, 2014. [Online]. Available: http://www.verizonwireless.com/support/android-os-faqs/. [Accessed 13 December 2014].

[13] Android, "Tools Help," Android, [Online]. Available: http://developer.android.com/tools/help/index.html. [Accessed 13 December 2014].

[14] F. N. Salsabila, Rancang Bangun Web Service untuk Implementasi Aturan Main dan Manajemen Transaksi dalam Permainan Sosial Heart Meister pada Perangkat Android, Surabaya: Teknik Informatika ITS, 2015.

[15] H. A. Surya, Implementasi Fitur Sosial untuk Permainan Sosial Heart Meister pada Perangkat Android, Surabaya: Teknik Informatika ITS, 2015.

[16] Nabilah, Implementasi Modul Random Dungeon Route Generator pada Game Sosial Heart Meister, Surabaya: Teknik Informatika ITS, 2015.