TUGAS AKHIR – KI141502

Pembuatan *World Generator* pada *Game Little Star for Little Wars*

UMA PATU RAMA
NRP 5111100094

Dosen Pembimbing
Imam Kuswardayan, S.Kom., M.T.
Dr.Eng. Nanik Suciati, S.Kom., M.Kom.

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2016

(Halaman Ini Sengaja Dikosongkan)

FINAL PROJECT– KI141502

Creating World Generator in Game Little Star for Little Wars

UMA PATU RAMA
NRP 5111100094

Advisor
Imam Kuswardayan, S.Kom., M.T.
Dr.Eng. Nanik Suciati, S.Kom., M.Kom.

DEPARTMENT OF INFORMATICS
FACULTY OF INFORMATION TECHNOLOGY
SEPULUH NOPEMBER INSTITUTE OF TECHNOLOGY
SURABAYA
2016

(Halaman Ini Sengaja Dikosongkan)

KATA PENGANTAR

Segala puji dan syukur kepada Tuhan Yang Maha Esa yang telah memberikan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul "Pembuatan *World Generator* pada *game Little Star for Little Wars*".

Pengerjaan tugas akhir ini adalah momen bagi penulis untuk mengeluarkan seluruh kemampuan, hasrat, dan keinginan yang terpendam di dalam hati mulai dari masuk kuliah hingga lulus sekarang ini, lebih tepatnya di kampus Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya.

Dalam pelaksanaan dan pembuatan tugas akhir ini tentunya sangat banyak bantuan-bantuan yang penulis terima dari berbagai pihak. Melalui lembar ini, penulis ingin secara khusus menyampaikan ucapan terima kasih kepada:

1. Tuhan Yang Maha Esa yang telah memberikan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan Tugas Akhir dengan baik
2. Orang tua dari penulis yang selalu mendukung penulis.
3. Bapak Imam Kuswardayan selaku dosen pembimbing Tugas Akhir pertama dan yang telah memberikan arahan dalam pengerjaan Tugas Akhir ini.
4. Ibu Nanik Suciati selaku dosen pembimbing yang selama ini memberikan bantuan kepada penulis.
5. Bapak Radityo Anggoro selaku dosen koordinator Tugas Akhir yang telah membantu penulis atas segala sesuatu mengenai syarat-syarat dan terlaksananya sidang Tugas Akhir.
6. Bapak Darlis Herumurti selaku Kepala Jurusan Teknik Informatika ITS.
7. Dosen-dosen Teknik Informatika yang dengan sabar mendidik dan memberikan pengalaman baru kepada penulis selama di Teknik Informatika.
8. Staf TU Teknik Informatika ITS yang senantiasa memudahkan segala urusan penulis di jurusan.

9. Rekan-rekan dan pengelola Laboratorium Interaksi, Grafik, dan Seni yang telah memberikan fasilitas dan kesempatan melakukan riset atas Tugas Akhir yang dikerjakan penulis.
10. Rekan-rekan dan sahabat-sahabat penulis angkatan 2011 yang memberikan dorongan motivasi dan bantuan kepada penulis.
11. Pihak-pihak lain yang tidak sengaja terlewat dan tidak dapat penulis sebutkan satu per satu.

Penulis telah berusaha sebaik mungkin dalam menyusun tugas akhir ini, namun penulis mohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya.

Surabaya, Januari 2016
Penulis

Uma Patu Rama

LEMBAR PENGESAHAN

PEMBUATAN WORLD GENERATOR PADA GAME LITTLE STAR FOR LITTLE WARS

Tugas Akhir

Diajukan Untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Rumpun Mata Kuliah Interaksi, Grafika, dan Seni
Program Studi S-1 Jurusan Teknik Informatika
Fakultas Teknologi Informasi
Institut Teknologi Sepuluh Nopember

Oleh:
UMA PATU RAMA
NRP. 5111 100 094

Disetujui oleh Dosen Pembimbing Tugas Akhir:

Imam Kuswardayan, S.Kom., M.T. .....
NIP: 19761215 200312 1 001 (pembimbing 1)

Dr.Eng. Nanik Suciati, S.Kom., M.Kom. .....
NIP: 19710428 199412 2 001 (pembimbing 2)

SURABAYA
JANUARI, 2016

(Halaman Ini Sengaja Dikosongkan)

PEMBUATAN WORLD GENERATOR PADA GAME LITTLE STAR FOR LITTLE WARS

Nama Mahasiswa : Uma Patu Rama
NRP : 5111 100 094
Jurusan : Teknik Informatika FTIf-ITS
Dosen Pembimbing I : Imam Kuswardayan, S.Kom., M.T.
Dosen Pembimbing II :Dr.Eng. Nanik Suciati, S.Kom., M.Kom.

ABSTRAK

Saat ini game memiliki perkembangan yang sangat pesat. Kebanyakan game yang dibuat sampai saat ini menggunakan level manual atau level yang sudah ditentukan oleh pembuat game tersebut. Saat pemain telah menyelesaikan game yang dimainkan, pemain tersebut kemungkinan akan merasa bosan karena jika pemain mengulangi permainan tersebut dari awal, keadaannya akan sama berdasarkan tingkat kesulitan yang sudah dimainkan sebelumnya.

Dalam tugas akhir ini dibuat World Generator yang digunakan untuk membuat world yang berbeda dengan tingkat kesulitan yang hampir sama dalam suatu level, sehingga pemain dapat bermain berulang – ulang dengan keadaan world yang berbeda pada level yang sama. World yang dihasilkan oleh World Generator juga harus seimbang agar tidak terlalu menguntungkan atau merugikan pemain maupun lawan.

Metode World Generator yang dibuat di game Little Star for Little Wars, terdiri dari tiga bagian. Yang pertama yaitu menentukan nilai jumlah planet, banyak musuh, dan kecepatan regenerasi musuh untuk setiap level dari game Little Star for Little Wars. Bagian selanjutnya yaitu pembuatan world yang terdiri dari penentuan posisi setiap planet dan pembuatan edge. Bagian yang ketiga yaitu penentuan world yang adil yang terdiri dari penentuan planet awal untuk pemain dan musuh, dan

penentuan nilai untuk planet – planet yang belum dikuasai oleh pemain atau musuh.

Pengujian dilakukan dengan memainkan game di level yang sama beberapa kali untuk menunjukkan world yang dinamis dan seimbang pada level yang dipilih. Pengujian kecepatan pembuatan world juga dilakukan di level yang sama sebanyak lima kali untuk mendapatkan kecepatan rata – rata pembuatan world di setiap level.

Hasil dari tugas akhir ini adalah world yang dinamis dan seimbang dalam game Little Star for Little Wars dengan menggunakan world generator. Dan hasil yang lain adalah rata – rata hasil kecepatan pembuatan world untuk setiap level dalam game Little Star for Little Wars.

Kata kunci : World Generator, Level, Game

CREATING WORLD GENERATOR IN GAME LITTLE STAR FOR LITTLE WARS

Student Name : Uma Patu Rama
NRP : 5111 100 094
Major : Teknik Informatika FTIf-ITS
Advisor I : Imam Kuswardayan, S.Kom., M.T.
Advisor II : Dr.Eng. Nanik Suciati, S.Kom., M.Kom.

ABSTRACT

Game today has very rapid development. Most of the game that ever made until today use manual level or level that already decided by the game maker. When player has finish the game, the player may feel bored because if player repeat the game from the start, the situation will be the same depend on the level that already played before.

In this final project, World Generator is made which used to make different world with degree of difficulty that almost the same in some level, so that player can play repeatedly with different world in the same level. World that creted with world generator need to be balance so that neither player or enemy get too advantage or disadvantage.

World Generator method which made in game Little Star for Little Wars, consist of three parts. The first one is determine value of planet, enemies, and the speed of enemy regeneration of every level. Next part is making world that consist of determining postition of every planets and creating edge. The third part is determine the fair world that consist of determining first planet for player and enemies, and determine value for planets that yet ruled by player or enemies.

Testing is done by playing the game in the same level a few times to show dynamic and balanced world in the choosen level. Creating world speed testing is done in the same level by five times to get the average speed of creating world.

Result of this final project is dynamic world and balanced world in game Little Star for Little Wars using world generator. And the other result is the average result of creating world speed for every level in game Little Star for Little Wars.
Keywords: World Generatror, Level, Game

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

(Halaman Ini Sengaja Dikosongkan)

BAB I
PENDAHULUAN

Bab ini memaparkan garis besar tugas akhir yang meliputi latar belakang, tujuan dan manfaat pembuatan, rumusan dan batasan permasalahan, metodologi pembuatan tugas akhir, dan sistematika penulisan.

1.1 Latar Belakang

Perkembangan teknologi komputer pada saat ini sangat pesat, dimana yang telah kita ketahui dalam instansi pemerintahan maupun swasta, lebih mengutamakan menggunakan teknologi komputer dalam menyelesaikan pekerjaannya, dan juga bagi para pelajar atau mahasiswa serta dosen dalam mempermudahkan proses belajar mengajar.

Salah satu hasil dari perkembangan teknologi ini adalah game. Game merupakan sebuah alat rekreasi yang sangat populer dimasyarakat. Hampir seluruh lapisan masyarakat saat ini pernah memainkan sebuah *game*. Pada perkembanganya selain bisa dimainkan secara bersama–sama, game juga dituntut untuk bisa dimainkan sendirian dengan tingkat kesenangan dan tantangan yang tidak kalah jika dimainkan bersama-sama. Sehingga dibutuhkan *game* yang dapat menghasilkan *level* yang dinamis untuk menambah unsur adiktif pada *game* yang dibuat.

Kebanyakan *game* yang dibuat sampai saat ini menggunakan *level* statis atau *level* yang sudah ditentukan oleh pembuat *game* tersebut. Saat pemain menyelesaikan *game* tersebut, pemain kemungkinan akan merasakan kebosanan karena jika mengulang permainan tersebut dari awal, tingkat kesulitannya akan sama dengan ketika pemain pertama kali bermain *game* tersebut.

Tugas akhir ini lebih difokuskan pada *world generator* sehingga meskipun *game* yang akan dibuat dimainkan berulang – ulang, akan menghasilkan *world* yang berbeda pada tingkat kesulitan yang sama.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini adalah sebagai berikut:

1. Bagaimana membuat *world* yang dinamis di setiap *level* pada *game Little Star of Little Wars*?
2. Bagaimana membuat *world* yang seimbang untuk pemain dan musuh?

1.3 Batasan Masalah

Permasalahan yang dibahas dalam tugas akhir ini memiliki beberapa batasan, di antaranya sebagai berikut:

1. Aplikasi yang dibangun dengan menggunakan *IDE Unity* dan bahasa pemrograman c#
2. Aplikasi yang dibangun melalui komponen objek dua dimensi
3. Jumlah maksimal planet dalam *game Little Star of Little Wars* untuk setiap *level* adalah 20
4. Jumlah maksimal AI dalam *game Little Star of Little Wars* untuk setiap *level* adalah 2
5. *Level* maksimal yang bisa dimainkan adalah 15.

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah untuk membuat sebuah *World Generator* yang nantinya akan digunakan dalam game *Little Star of Little Wars*.

1.5 Manfaat

Manfaat dari hasil pembuatan tugas akhir ini antara lain:

1. Dapat membuat *world* yang dinamis di setiap *level* pada *game little star for little wars.*
2. Dapat membuat *world* yang seimbang bagi pemain dan musuh.

1.6 Metodologi

Pembuatan tugas akhir dilakukan menggunakan metodologi sebagai berikut:

A. Studi literatur
   Tahap Studi Literatur merupakan tahap pembelajaran dan pengumpulan informasi yang digunakan untuk mengimplementasikan Tugas Akhir. Tahap ini diawali dengan pengumpulan literatur, diskusi, eksplorasi teknologi dan pustaka, serta pemahaman dasar teori yang digunakan pada topik tugas akhir. Literatur-literatur yang dimaksud disebutkan sebagai berikut:
   1. Penggunaan *Unity Game Engine*;
   2. Algoritma Djikstra.

B. Perancangan perangkat lunak
   Pada tahap ini dilakukan analisa awal dan pendefinisian kebutuhan sistem untuk mengetahui permasalahan yang sedang dihadapi. Selanjutnya, dirumuskan rancangan sistem yang dapat memberi solusi terhadap permasalahan tersebut. Langkah yang akan digunakan pada tahap ini adalah sebagai berikut:
   1. Pencarian dan pendataan materi yang akan digunakan dalam *game Little Star for Little Wars*.
   2. Perancangan sistem dan mekanisme *game Little Star for Little Wars*.
   3. Analisis kebutuhan non fungsional.
   4. Analisis algoritma dan formula dalam membangun *World Generator* dalam *game Little Star for Little Wars.*
   5. Perancangan algoritma dan formula dalam membangun *World Generator* dalam *game Little Star for Little Wars*

C. Implementasi dan pembuatan sistem
   Tahap implementasi merupakan tahap untuk membangun aplikasi permainan beserta sistem yang terkait. Aplikasi ini akan dibangun dengan bahasa pemrograman C# dengan IDE Unity.

D. Uji coba dan evaluasi
   Pada tahap ini akan dilakukan pengujian terhadap perangkat lunak menggunakan data atau skenario yang telah dipersiapkan sebelumnya yakni sebagai berikut:
   1. Pengujian blackbox
      Pengujian blackbox adalah pengujian yang berfokus pada spesifikasi fungsional dari perangkat lunak, tester dapat mendefinisikan kumpulan kondisi input dan melakukan pengetesan pada spesifikasi fungsional program. Pengujian ini dilakukan untuk menguji apakah proses kinerja aplikasi *game* ini sudah sesuai dengan kebutuhan pengguna atau tidak.
   2. Pengujian usabilitas
      Pengujian usabilitas dilakukan dengan cara melakukan survei ke pengguna yaitu beberapa pengguna yang suka bermain *game* di sekitar lingkungan Teknik Informatika ITS. Survei dilakukan untuk mengukur tingkat kegunaan dari aplikasi yang dibuat dalam membantu pengguna.

E. Penyusunan laporan tugas akhir
   Pada tahap ini dilakukan penyusunan laporan yang berisi dasar teori, dokumentasi dari perangkat lunak, dan hasil-hasil yang diperoleh selama pengerjaan tugas akhir.

1.7 Sistematika Penulisan

Buku tugas akhri ini terdiri dari beberapa bab, yang dijelaskan sebagai berikut.

BAB I PENDAHULUAN

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

BAB II TINJAUAN PUSTAKA

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

BAB III ANALISIS DAN PERANCANGAN

Bab ini membahas analisis dari sistem yang dibuat meliputi analisis permasalahan, deskripsi umum perangkat lunak, spesifikasi kebutuhan, dan identifikasi pengguna. Kemudian membahas rancangan dari sistem yang dibuat meliputi rancangan skenario kasus penggunaan, arsitektur, data, dan antarmuka.

BAB IV IMPLEMENTASI

Bab ini membahas implementasi dari rancangan sistem yang dilakukan pada tahap perancangan. Penjelasan implementasi meliputi implementasi pembangkitan area permainan, dan antarmuka permainan.

BAB V PENGUJIAN DAN EVALUASI

Bab ini membahas pengujian dari aplikasi yang dibuat dengan melihat keluaran yang dihasilkan oleh aplikasi dan evaluasi untuk mengetahui kemampuan aplikasi.

BAB VI PENUTUP

Bab ini berisi kesimpulan dari hasil pengujian yang dilakukan serta saran untuk pengembangan aplikasi selanjutnya.

(Halaman Ini Sengaja Dikosongkan)

BAB II
TINJAUAN PUSTAKA

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan tugas akhir. Teori-teori tersebut adalah Unity Game Engine*, C#,* Permainan, Algoritma Djikstra, dan Keseimbangan *Game*.

2.1 Unity Game Engine

Unity 3D adalah sebuah *game engine* yang *berbasis cross-platform*. Unity dapat digunakan untuk membuat sebuah *game* yang bisa digunakan pada perangkat komputer, ponsel pintar android, iPhone, PS3, dan bahkan Xbox.

Unity adalah sebuah sebuah alat yang terintegrasi untuk membuat *game*, arsitektur bangunan dan simulasi. Unity bisa untuk *PC game* dan. *Online game* Untuk *Online game* diperlukan sebuah plugin, yaitu Unity Web Player, sama halnya dengan *Flash Player* pada *Browser*.

Unity tidak dirancang untuk proses desain atau *modelling*, dikarenakan unity bukan alat untuk mendesain. Jika ingin mendesain, pergunakan 3D *editor* lain seperti 3dsmax atau Blender. Banyak hal yang bisa dilakukan dengan unity, ada fitur *audio reverb zone, particle effect*, dan *Sky Box* untuk menambahkan langit.

Fitur scripting yang disediakan, mendukung 3 bahasa pemrograman, JavaScript, C#, dan Boo. Fleksibel dan *EasyMoving*, *rotating*, *dan scaling objects* hanya perlu sebaris kode. Begitu juga dengan *Duplicating*, *removing*, dan *changing properties*. *Visual Properties Variables* yang di definisikan dengan *scripts* ditampilkan pada *Editor*. Bisa digeser, *drag* and *drop*, bisa memilih warna dengan *color picker*. Berbasis .NET, artinya penjalanan program dilakukan dengan *Open Source .NET platform*, *Mono*.

Pada setiap project Unity terdapat sebuah *Assets folder*. Isi dari *Assets folder* ditampilkan dalam bentuk *panel project* dalam *editor* unity. *Assets folder* adalah tempat untuk menyimpan semua

komponen dari *game* seperti tingkatan *game* (*level scenes*), *scripts*, 3D *models*, *teksture*, dan *file audio*.

Untuk menambahkan *assets* ke dalam *project*, cukup dengan menarik (*drag*) file yang ingin ditambahkan ke dalam *panel project*. Atau dengan memilih menu *Assets->Import New Asset*. Untuk membuat *scene* baru, gunakan tombol *Control-N* (pada *keyboard*). Untuk menyimpan *scene* yang sedang aktif, gunakan *Control-S* (pada *keyboard*).

Panel *Hierarchy* menampung semua *GameObject* yang terdapat di *Scene* yang sedang aktif. Beberapa dari *GameObject* tersebut berhubungan langsung ke *asset* seperti objek 3D. Objek yang terdapat pada hierarchy dapat di seleksi dan dihapus. Jika objek dihapus atau ditambahkan pada *scene*, maka objek tersebut juga akan hilang atau muncul pada *hierarchy*.

Unity menggunakan sebuah konsep yang disebut *Parenting*. Ini digunakan untuk membuat sebuah *GameObject* menjadi anak dari *GameObject* yang lain. Tarik sebuah *GameObject* dan pindahkan tepat di atas tulisan *GameObject* yang akan dijadikan parent dalam *hierarchy*. *GameObject* yang terdapat dalam sebuah *GameObject* lainnya akan mengikuti perpindahan dan perputaran ketika *GameObject* parent mengalami perubahan posisi.

2.2 Permainan

Menurut Scott Rogers permainan adalah sebuah aktivitas yang memerlukan paling sedikit satu pemain, mempunyai aturan, dan kondisi menang. Bermain dengan bola tangan hanya membuang waktu, namun dapat menjadi sebuah permainan jika terdapat peraturan dan tujuan. Si pemain hanya dapat melempar dengan tangan kanan dan menangkap dengan tangan kiri, dan tidak boleh menjatuhkan bola. Kondisi menang terjadi ketika si pemain berhasil menangkap bola sebanyak sepuluh kali tanpa jatuh. Kondisi gagal terjadi ketika si pemain melakukan pelanggaran terhadap peraturan atau kondisi menang. Sebuah aktivitas permainan tercipta ketika semua kriteria terpenuhi.

Hal yang pertama kali harus dilakukan oleh pengembang untuk membangun sebuah permainan yaitu menemukan ide. Dari penemuan ide tersebut, pengembang harus mengembangkannya menjadi cerita atau alur. Alur permainan dapat membantu pengembang melakukan perancangan sistem. Sistem tersebut meliputi "Tiga C" yaitu *character, camera* dan *control.* Selain "Tiga C", pengembang juga harus melakukan perancangan HUD dan *icon.* Semua aspek tersebut harus dilakukan secara sistematis agar tercipta suatu permainan yang baik dan menyenangkan.

2.3 *RTS (Real Time Strategy)*

Real Time Strategy adalah salah satu *genre* dari permainan yang memiliki ciri khas berupa pemain mengontrol banyak unit pada waktu yang bersamaan dan dilakukan dalam waktu nyata.

Berbeda dengan *TBS(Turn Based Strategy)* dimana pemain diberikan waktu yang tidak terbatas untuk secara leluasa memberikan perintah kepada setiap unit yang dikendalikannya, dalam *RTS* semua pemain diharapkan mampu melakukan pengendalian unit dan pemberian perintah pada waktu yang hampir bersamaan dengan pemain lainya.

2.4 C#

Microsoft C# (disebut C sharp) adalah sebuah bahasa pemrograman yang didesain untuk membangun jangkauan aplikasi enterprise yang berjalan di atas framework .NET. Sebuah evolusi Microsoft C dan Microsoft C++, C# sederhana, modern, aman dan Object Oriented. C# dikenal sebagai visual C# dalam Visual Studio .Net. Dukungan untuk Visual C# termasuk proyek template, desainer, halaman poperti, kode, model objek dan fitur lain dari lingkungan pengembangan. Library untuk pemrograman visual c# adalah .NET Framework [4]

Bahasa pemrograman ini dibuat berbasiskan bahasa C++ yang telah dipengaruhi oleh aspek-aspek ataupun fitur bahasa yang terdapat pada bahasa-bahasa pemrograman lainnya seperti *Java, Delphi, Visual Basic*, dan lain-lain) dengan beberapa penyederhanaan. Menurut standar ECMA-334 *C# Language*

Specification, nama C# terdiri atas sebuah huruf Latin C (U+0043) yang diikuti oleh tanda pagar yang menandakan angka # (U+0023). Tanda pagar # yang digunakan memang bukan tanda kres dalam seni musik (U+266F), dan tanda pagar # (U+0023) tersebut digunakan karena karakter kres dalam seni musik tidak terdapat di dalam keyboard standar.

Bahasa pemrograman C# ditujukan untuk digunakan dalam mengembangkan komponen perangkat lunak yang mampu mengambil keuntungan dari lingkungan terdistribusi. C# ditujukan agar cocok digunakan untuk menulis program aplikasi baik dalam sistem klien-server (*hosted system*) maupun sistem embedded (*embedded system*), mulai dari perangkat lunak yang sangat besar yang menggunakan sistem operasi yang canggih hingga kepada perangkat lunak yang sangat kecil yang memiliki fungsi-fungsi terdedikasi. Meskipun aplikasi C# ditujukan agar bersifat 'ekonomis' dalam hal kebutuhan pemrosesan dan memori komputer, bahasa C# tidak ditujukan untuk bersaing secara langsung dengan kinerja dan ukuran perangkat lunak yang dibuat dengan menggunakan bahasa pemrograman C dan bahasa rakitan.

2.5 Keseimbangan *Game*

Dalam mendesain *game*, keseimbangan adalah konsep dan percobaan kesesuaian dari aturan *game*, biasanya digunakan untuk mencegah jika ada bagian – bagian dari sistem yang tidak efektif atau tidak diinginkan ketika dibandingkan dengan bagian yang lain. Sistem yang tidak seimbang menunjukkan pengunaan sumber daya yang sia – sia.

Keseimbangan tidak berarti membuat *game* yang adil. Hal ini terbilang benar terutama untuk *action game.* Jaime Griesemer, pimpinan desain di Bungie, memberi kuliah ke disainer yang lain bahwa “setiap pertarungan di *Halo* tidak adil”. Ketidakadilan dapat membuat ketidakpastian, yang membawa ke ketegangan dan kesenangan yang ingin diberikan oleh setiap *action game*.

BAB III
ANALISIS DAN PERANCANGAN

Pada bab ini akan dibahas mengenai analisis dan perancangan yang akan digunakan untuk menyelesaikan tugas akhir. Bab ini dibagi menjadi dua bagian. Pertama membahas mengenai pembuatan *game* secara umum. Bagian ini berisi tentang perancangan permainan.

Bagian kedua berisi tentang penarapan *world generator* dalam *game.* Pada bagian ini dibahas tentang tahap – tahap dalam pembuatan *world generator* serta algoritma untuk optimisasi dari metode *world generator* yang digunakan.

3.1 Analisis Sistem

Permainan video berkembang sangat pesat akibat tuntutan perkembangan zaman. Masyarat berminat terhadap permainan video yang tentunya menyenangkan. Salah satu faktor yang mempengaruhi tingkat kesenangan pemain dari permainan video yaitu level permainan. Level permainan yang beragam tentunya membutuhkan variasi tingkat kesulitan pada permainan. Salah satu cara untuk meningkatkan tingkat kesulitan permainan adalah dengan menyediakan lawan yang menantang.

Aplikasi ini dibangun dengan tujuan pemain dapat memainkan *game* berulang – ulang di *level* yang sama dengan *world* yang dihasilkan berbeda setiap kali dimainkan. *Level* pada sistem permainan dibangun secara dinamis berdasarkan tingkat kesulitan masing – masing. Data yang digunakan untuk membangun *level* adalah pembobotan setiap *level*. Semakin tinggi *level* yang dipilih maka nilai pembobotan juga akan semakin tinggi. Aplikasi ini diharapkan dapat memberikan unsur adiktif kepada pemainnya dan mengurangi tingkat kebosanan dari *game* yang menggunakan *level* secara manual.

Penulis menggunakan *IDE* Unity Game Engine dan bahasa pemrograman C# untuk memfasilitasi pengembangan permainan.

3.2 Perancangan Permainan

3.2.1 Deskripsi Umum Perangkat Lunak

Tugas akhir yang akan dikembangkan adalah sebuah permainan 2D bergenre RTS atau strategi waktu nyata. Pengguna utama dari *game* ini adalah semua orang yang ingin bermain. Pemain berperan sebagai pemimpin pasukan dari koloni sebuah bangsa yang ingin menguasai satu buah galaksi atau kumpulan planet. Pemain dapat memilih *level* yang akan dimainkan pada **Layar Level Selection**. Pada **Layar Level Selection** disediakan beberapa *level* yang dapat dimainkan oleh pemain. Semakin tinggi *level* yang dipilih, kesulitan yang dihadapi pemain untuk menyelesaikan *level* tersebut menjadi bertambah sulit. Serta pemain diharuskan memilih *level* pertama terlebih dahulu dan harus memenangkan *level* tersebut untuk bisa memilih *level* selanjutnya.

Dalam sistem permainan ini, pemain melakukan manajemen terhadap sumber daya yang ada pada sebuah planet untuk menyerang atau bertahan dari serangan. Pemain dinyatakan menang ketika semua lawan pada galaksi sudah berhasil dikalahkan.

3.2.2 Perancangan Kontrol *Game*

Kontrol *game* yang dibuat membutuhkan alat masukan berupa *keyboard* dan *mouse*. Tombol yang diperlukan adalah klik kiri dan klik kanan pada *mouse.* Pada *keyboard* tombol yang digunakan adalah tombol *Escape.*

3.2.3 Perancangan Alur *Game*

Alur *game* merupakan serangkaian proses yang harus diikuti pemain untuk memperoleh kemenangan. Dalam *game* ini pemain berperan sebagai pemimpin pasukan militer dari sebuah koloni makhluk hidup disebuah planet diluar angkasa.

Perkembangan teknologi dari ras makhluk hidup ini sangatlah cepat. Pada suatu waktu karena banyaknya penghuni dari planet asal pemain, pemain mengemban misi untuk mencari planet-planet lain untuk ditinggali.

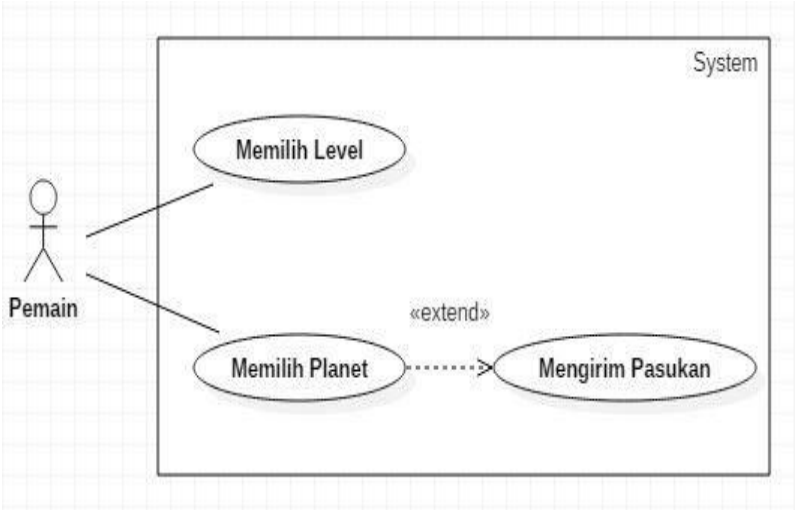
Dalam perjalanan mencari planet baru untuk ditinggali, pemain kadang bertemu ras makhluk hidup lain yang juga mencari tempat tinggal baru. Karena ketidakcocokan akhirnya pemain dan ras lain tersebut berseteru untuk memperebutkan planet dan galaksi baru untuk menjadi tempat tinggal. Pemain dapat dikatakan menang jika pemain sudah menguasai semua planet yang dimiliki oleh pemain lawan pada sebuah galaksi.

3.2.4 Spesifikasi Kebutuhan Fungsional

Dalam aplikasi tugas akhir ini, terdapat tiga kasus penggunaan. Tiga kasus penggunaan ini diantaranya adalah memilih *level*, memilih planet, dan mengirim pasukan. Pengguna atau entitas luar dari sistem adalah pemain yang memainkan permainan.

3.2.5 Perancangan Skenario Kasus Penggunaan

Kasus penggunaan yang terdapat didalam sistem dicantumkan pada Gambar 3.1.



Gambar 3.1 Diagram Kasus Aplikasi

Penjelasan dari masing-masing kasus penggunaan dicantumkan pada Tabel 3.1 . Tabel tersebut berisi penjelasan skenario yang akan dilakukan ketika pengujian.

Tabel 3.1 Skenario Kasus Penggunaan

No	Kode Kasus Penggunaan	Nama Kasus Penggunaan	Keterangan
1	UC-001	Memilih Level	Untuk memilih tingkat kesulitan level yang akan dimainkan pemain.
2	UC-002	Memilih Planet	Untuk memilih planet mana yang akan diberi perintah.
3	UC-003	Mengirim Pasukan	Untuk memilih planet mana yang akan dikirimi pasukan dari planet awal yang sudah dipilih

3.2.5.1 Kasus Penggunaan Permainan

Penjelasan kasus penggunaan permainan untuk skenario UC-001 yakni Memilih *level* dijelaskan pada Tabel 3.2.

Tabel 3.2 Skenario Kasus Penggunaan Memilih *Level*

Nama Kasus Penggunaan	Memilih level
Kode	UC-001
Deskripsi	Kasus penggunaan dimana aktor memilih level kesulitan yang akan dimainkan
Aktor	Pemain.
Kondisi Awal	Pemain sudah masuk ke aplikasi dan masuk ke layar pemilihan level
Alur Normal	1. Pemain memilih tombol ‘START’ dan menekan klik. 2. Sistem menampilkan layar **Pemilihan Level**. 3. Pemain memilih level kesulitan mana yang akan dimainkan

	4. Sistem menampilkan layar **Permainan** sesuai dengan pilihan yang dibuat pemain

Selanjutnya penjelasan kasus penggunaan permainan untuk skenario UC-002 yakni Memilih Planet dijelaskan pada Tabel 3.3.

Tabel 3.3 Skenario Kasus Penggunaan Memilih Planet

Nama Kasus Penggunaan	Memilih Planet
Kode	UC-002
Deskripsi	Kasus penggunaan dimana aktor memilih planet mana yang akan diberi perintah.
Aktor	Pemain.
Kondisi Awal	Pemain sudah masuk ke aplikasi, memilih tingkat kesulitan yang akan dimainkan dan planet yang dipilih sudah dikuasai oleh pemain
Alur Normal	1. Pemain memilih level yang akan dimainkan. 2. Sistem menampilkan layar **Permainan**. 3. Pemain memilih planet yang sudah dikuasai untuk diberi perintah. 4. Sistem memberikan indikator planet mana yang dipilih

Kemudian penjelasan kasus penggunaan permainan untuk skenario UC-003 yakni Mengirim Pasukan dijelaskan pada Tabel 3.4.

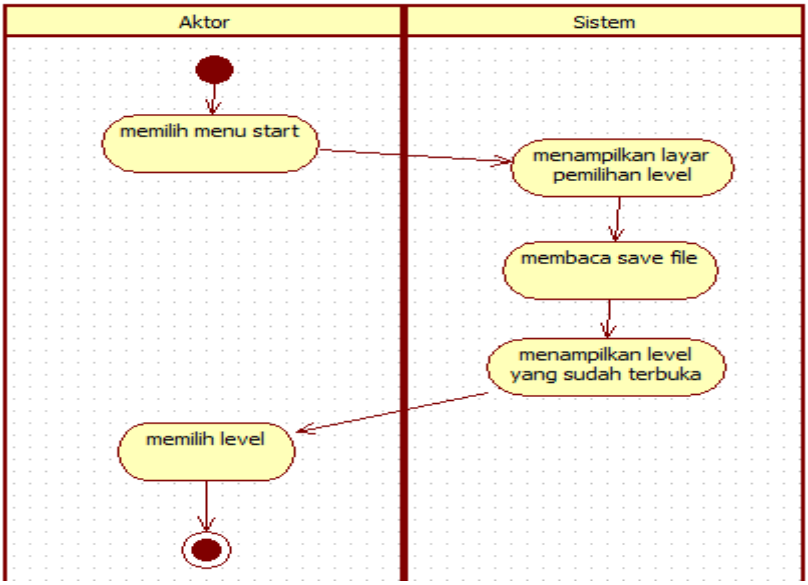
Tabel 3.4 Skenario Kasus Penggunaan Mengirim Pasukan

Nama Kasus Penggunaan	Mengirim Pasukan
Kode	UC-003
Deskripsi	Kasus penggunaan dimana aktor memilih planet target yang akan dikirimi pasukan
Aktor	Pemain.
Kondisi Awal	Pemain sudah memilih planet awal yang akan mengirim pasukan

Alur Normal	1. Pemain memilih salah satu planet yang dikuasai untuk mengirim pasukan. 2. Sistem memberikan indikator planet mana yang dipilih 3. Pemain memilih planet mana yang akan dikirimi pasukan. 4. Sistem membuat unit penyerang dari planet awal menuju planet tujuan

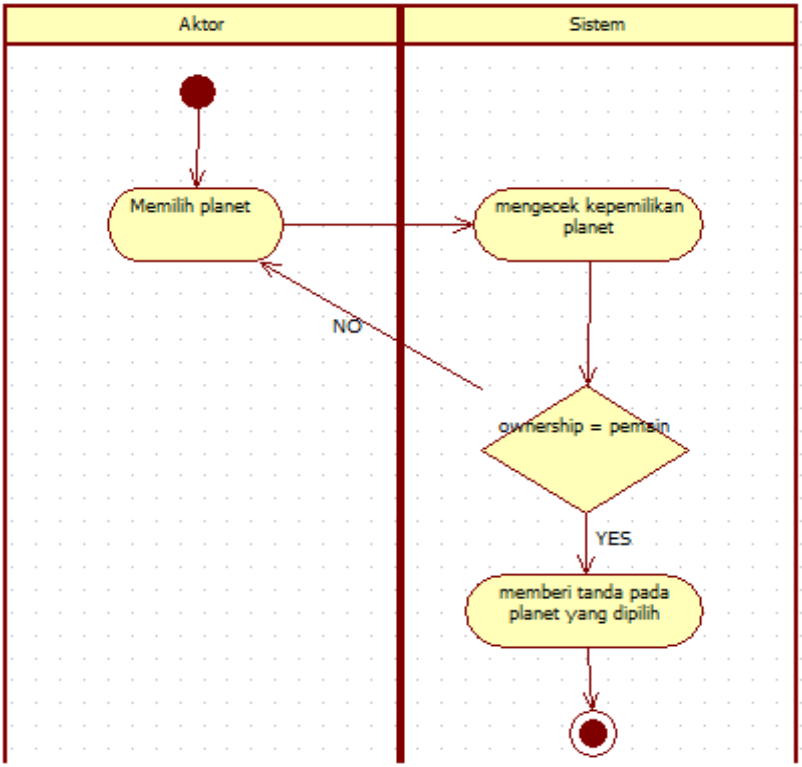
3.2.5.2 Diagram Aktivitas

Diagram aktivitas menampilkan langkah-langkah normal yang harus dilakukan pemain untuk menjalankan studi kasus permainan dimulai dari awal permainan hingga kondisi akhir. Diagram aktivitas untuk dari kasus penggunaan UC-001 yakni Memilih *Level* dijelaskan pada Gambar 3.2.



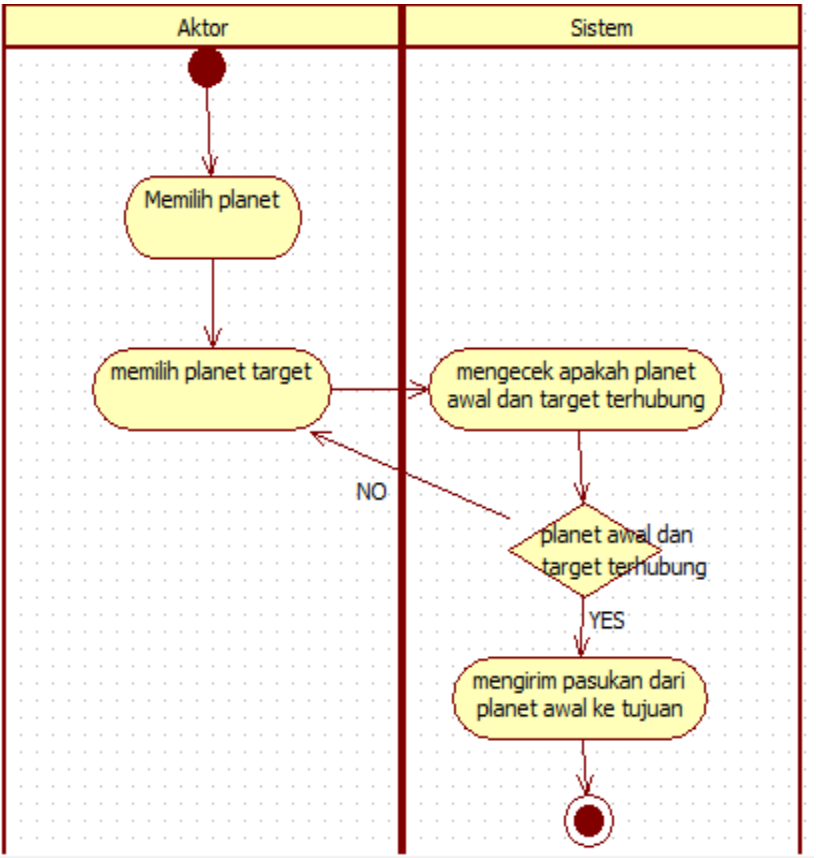
Gambar 3.2 Diagram Aktivitas Memilih *Level*

Diagram aktivitas untuk dari kasus penggunaan UC-002 yakni Memilih Planet dijelaskan pada Gambar 3.3. Setelah pemain memilih level yang ingin dimainkan sistem akan membawa pemain ke layar permainan sesuai dengan level yang telah dipilih. Pada layar permainan pemain dapat memilih planet yang sudah dikuasai untuk memberikan perintah. Pemain tidak dapat memilih planet yang belum dia kuasai. Jika pemain mencoba untuk memilih planet yang tidak dia kuasai makan sistem akan mengabaikan masukan yang dilakukan oleh pemain.



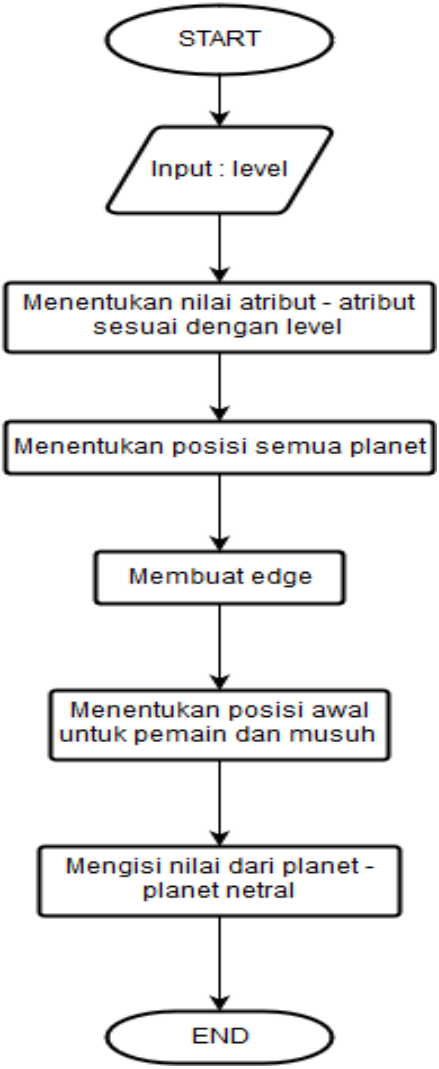
Gambar 3.3 Diagram Aktivitas Memilih Planet

Kemudian diagram aktivitas untuk dari kasus penggunaan UC-003 yakni Mengirim Pasukan dijelaskan pada Gambar 3.4. Setelah pemain memilih salah satu planet yang sudah dikuasai, pemain dapat memberikan perintah kepada planet tersebut untuk mengirimkan pasukan ke planet yang terhubung dengan planet yang dipilih. Jika pemain memilih planet yang tidak terhubung maka sistem akan mengabaikan masukan yang dilakukan oleh pemain.



Gambar 3.4 Diagram Aktivitas Mengirim Pasukan

3.2.6 Perancangan Skenario *World Generator*



Gambar 3.5 Diagram Alur Skenario *World Generator*

Skenario pembuatan *world generator* dapat dilihat pada Gambar 3.5. Dalam skenario menjalankan *world generator,* sistem akan mendapatkan masukkan nilai *level* dari pemain. Dari nilai *level* yang didapatkan dari pemain, ditentukan nilai – nilai dari semua atribut untuk membuat *world*. Atribut – atribut yang ditentukan adalah jumlah planet, jumlah musuh, dan kecepatan regenerasi dari planet – planet yang dikuasai oleh musuh.

Kemudian sistem akan membuat *world* sesuai dengan nilai atribut – atribut yang telah ditentukan. Pembuatan *world* diawali dengan menentukan posisi semua planet. Posisi planet ditentukan secara random dan tidak boleh terlalu dekat dengan planet yang lainnya. Setelah menentukan posisi semua planet, sistem kemudian akan membuat edge.

Setelah selesai membuat *world,* sistem akan menentukan posisi awal untuk pemain dan musuh.. Posisi awal untuk pemain dan AI ditentukan dengan mencari 2 atau 3 planet yang memiliki jarak paling jauh dari satu sama lain. Jika AI ada 2 maka digunakan rumus pembobotan untuk menentukan posisi planet awal agar jaraknya sama satu sama lain.

Planet netral adalah planet yang tidak dikuasai oleh musuh atau oleh pemain. Untuk menguasai planet netral pemain atau musuh harus mengirimkan pasukan dengan jumlah yang sama atau lebih besar dari planet netral yang akan dikuasai. Nilai awal planet netral ditentukan berdasarkan jarak dari planet awal pemain dan planet awal musuh.

3.2.6.1 *World* Dinamis

World yang dibuat dengan menggunakan *world generator* harus dinamis. Faktor yang membuat *world* dalam suatu *level* dinamis adalah :

1. Nilai atribut – atribut setiap kali memainkan *level* yang sama.
2. Hasil pembuatan *world* secara visual.

Agar *world* yang dibuat dinamis, nilai atribut – atribut yang menentukan suatu *level* sebaiknya berbeda setiap setiap kali *level*

tersebut dimainkan. Meskipun nilai atributnya berbeda setiap kali dimainkan tapi nilai tingkat kesulitannya tidak boleh melebihi dari nilai tingkat kesulitan yang sudah ditentukan.

Jika nilai atribut di *level* sama berbeda maka *world* yang dihasilkan akan terlihat berbeda secara visual. Tetapi meskipun atribut yang dihasilkan di *level* tersebut selalu sama, maka untuk membuat *world* yang dinamis *world* yang dihasilkan tetap harus berbeda secara visual. Hal yang membuat *world* berbeda secara visual karena meskipun nilai atributnya sama, tetapi posisi setiap planet, posisi awal pemain dan musuh, dan nilai pasukan planet netral berbeda.

3.2.6.2 *World* yang Seimbang

World yang dibuat dengan menggunakan *world generatror* harus seimbang. *World* yang seimbang diterapkan ketika sistem mengisi nilai dari planet – planet netral. Faktor yang menyebabkan keseimbangan dalam *world* adalah :

1. Jarak planet netral ke planet awal terdekat.
2. Nilai pasukan di planet netral.

Hal yang mempengaruhi kemudahan dalam menguasai planet lain adalah jarak. Jika jarak sebuat planet yang ingin dikuasai dari planet milik pemain atau AI semakin besar maka akan semakin sulit bagi pemain atau AI untuk menguasai planet tersebut. Jika jarak sebuah planet yang ingin dikuasai dari planet pemain atau musuh semakin pendek, maka akan semakin mudah bagi pemain atau musuh untuk menguasai planet tersebut.

Untuk membuat *world* yang seimbang, nilai pasukan di planet netral berbeda satu sama lain. Nilai pasukan planet netral berbeda karena jarak setiap planet netral dari planet awal pemain atau musuh berbeda. Sehingga agar *world* yang dibuat bisa seimbang, nilai pasukan planet netral dibuat berdasarkan jarak planet netral tersebut ke planet awal pemain atau musuh. Semakin besar jarak planet netral dari planet awal pemain atau musuh, maka nilai pasukan di planet netral tersebut akan semakin kecil. Semakin

pendek jarak planet netral dari planet awal pemain atau musuh, maka nilai pasukan di planet netral tersebut akan semakin besar.

3.2.7 Spesifikasi Kebutuhan Non-Fungsional

Terdapat beberapa kebutuhan non-fungsional yang apabila dipenuhi, dapat meningkatkan kualitas dari permainan ini. Berikut daftar kebutuhan non-fungsional.

1. FrameRate

Permainan ini harus mampu dimainkan secara lancar, tidak ada *lag* dan nyaman di mata. Sebagian permainan 2D biasanya optimal pada *Frame Rate* 24-30 fps (*Frame per Second*). Namun untuk pembangkitan level secara dinamis dapat mempengaruhi kelancaran *Frame Rate* di awal pembentukan. Serta kelancaran *Frame Rate* dipengaruhi dan tergantung oleh spesifikasi komputer yang akan digunakan.

2. Kebutuhan Grafis

Kenyamanan bermain berbanding lurus dengan kualitas grafis yang disajikan dalam permainan. Efek seperti animasi merupakan salah satu daya tarik dalam suatu permainan. Efek-efek ini bisa membuat *drop rate fps* dan permainan melambat (*lag*), karena membutuhkan tambahan komputasi.

3.2.8 Karakteristik Pengguna

Berdasarkan deskripsi umum diatas, maka dapat diketahui bahwa pengguna yang akan menggunakan aplikasi ini ada dua orang, yaitu pemain yang memainkan permainan, dan pengembang. Karakteristik pengguna tercantum dalam Tabel 3.5.

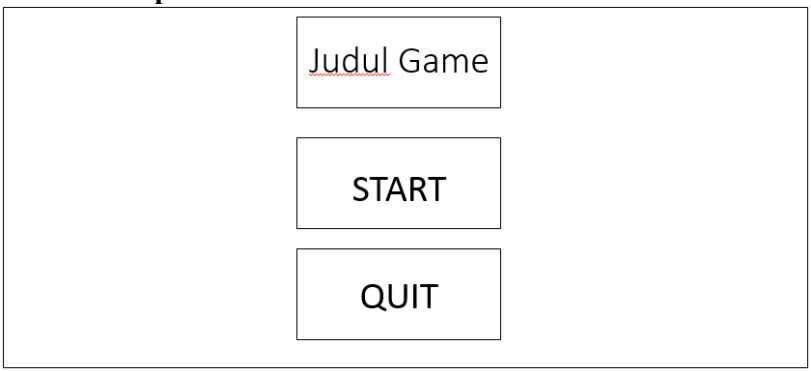
Tabel 3.5 Karakteristik Pengguna

Nama Aktor	Tugas	Hak Akses Aplikasi	Kemampuan yang harus dimiliki
Pemain	Pihak luar yang memainkan permainan.	Memainkan permainan	Tidak ada

3.2.9 Perancangan Antarmuka Pengguna

Subbab ini membahas bagaimana rancangan antarmuka pengguna yang akan digunakan untuk tugas akhir. Rancangan antarmuka yang dibahas meliputi ketentuan masukan dan rancangan jendela tampilan. Dalam aplikasi ini terdapat beberapa tampilan, yaitu **Main Menu**, **Level Selection**, **Game Play**, **Game Over,** dan **Tutorial**.

3.2.9.1 Tampilan Main Menu

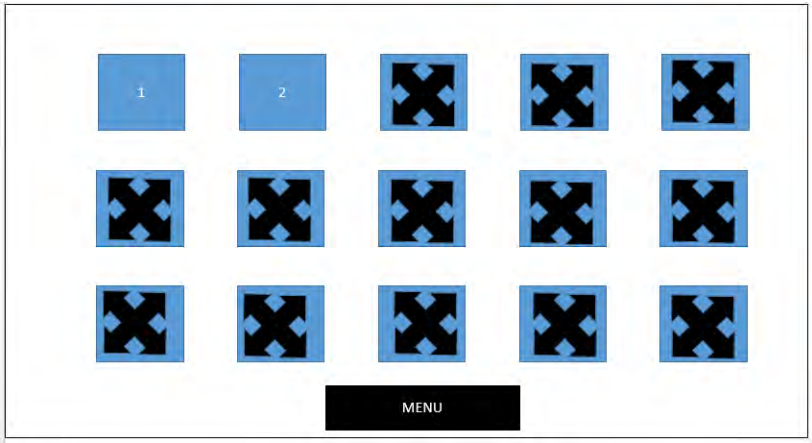


Gambar 3.6 Tampilan Main Menu

Tampilan Main Menu merupakan tampilan yang pertama kali muncul ketika aplikasi dijalankan pertama. Rancangan tampilan **Main Menu** dapata dilihat pada Gambar 3.6.

3.2.9.2 Tampilan Level Selection

Layar **Level Selection** ditampilkan sesudah pemain menekan tombol START. Dalam tampilan ini dapat dilihat *level – level* yang sudah dibuka oleh pemain. Pada saat aplikasi permainan pertama kali dijalankan pilihan *level* yang terbuka hanya *level* 1 saja. Seiring pemain mengalahkan *level-level* yang ada, *level-level* yang lebih tinggi akan terbuka. Ikon *level* yang masih belum terbuka akan tertutup gambar silang dan tidak dapat dipilih. Tampilan layar **Level Selection** ditampilkan pada Gambar 3.7.



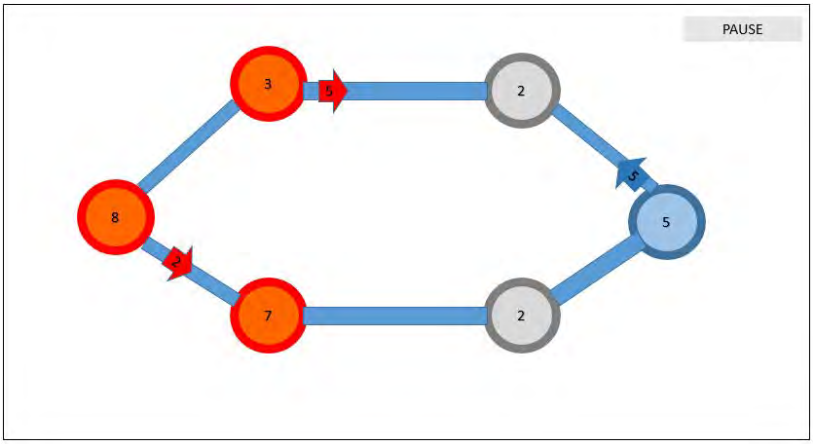
Gambar 3.7 Tampilan Level Selection

3.2.9.3 Tampilan Game Play

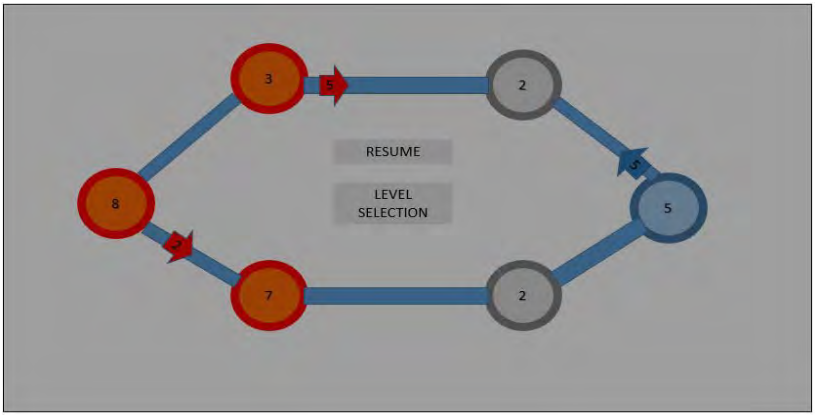
Tampilan **Game Play** merupakan tampilan dimana pemain memainkan *game*. Dalam tampilan ini akan ditunjukan beberapa bulatan yang melambangkan planet serta garis-garis yang menghubungkan planet-planet tersebut. Pasukan hanya bisa dikirim antara planet-planet yang terhubung dengan garis. Tampilan **Game Play** dapat dilihat pada Gambar 3.8.

Saat bermain pemain dapat menghentikan permainan dengan menekan tombol *pause* yang ada di sebelah kanan atas layar atau

menekan tombol *esc* pada *keyboard*. Tampilan permainan dapat dilihat pada Gambar 3.9.



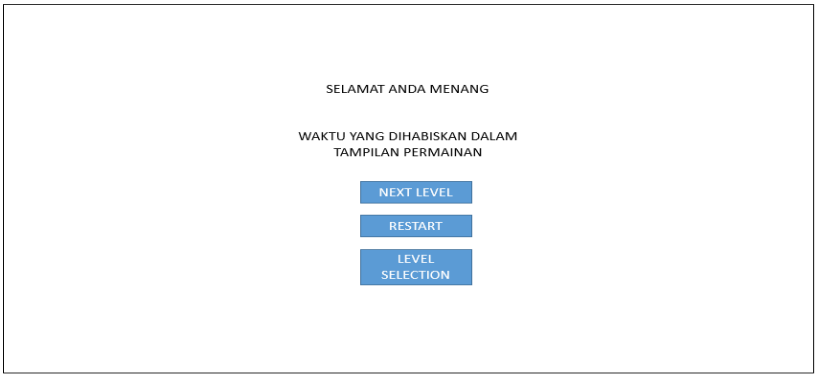
Gambar 3.8 Tampilan Game Play



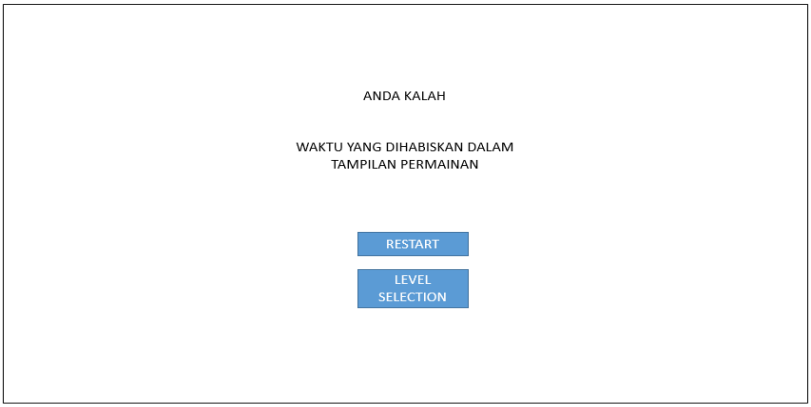
Gambar 3.9 Tampilan Game Play saat Pause

3.2.9.4 Tampilan Game Over

Tampilan ini merupakan tampilan dari **Game Over**. Setelah pengguna bermain dalam sebuah *level*, ketika permainan berakhir pengguna akan dibawa ke tampilan ini. Tampilan yang ditampilkan tergantung dari hasil permainan. Tampilan **Game Over** ditunjukkan seperti pada Gambar 3.10 dan Gambar 3.11.



Gambar 3.10 Tampilan Game Over saat Menang



Gambar 3.11 Tampilan Game Over saat Kalah

3.2.9.5 Tampilan Tutorial

Tampilan ini merupakan tampilan untuk memberikan bantuan kepada pemain untuk memahami cara bermain dari permainan ini. Tampilan **Tutorial** ditunjukkan seperti pada Gambar 3.12.

Gambar 3.12 Tampilan Tutorial

3.3 Perancangan World Generator

Dalam subab ini dibahas perancangan *world generator* yang akan digunakan dalam permainan dan algoritma yang akan digunakan untuk melakukan optimasi *world generator* tersebut.

3.3.1 Perancangan Level

Dalam Tabel 3.6 menunjukkan salah satu kondisi yang mencantumkan detil perancangan tiap *level*, dengan kolom yang berisi :

- *Level,* berisi tingkat kesulitan yang ada pada menu *level selection*.
- Jumlah Planet, Jumlah planet di setiap *level*. Jumlah planet di setiap *level* ditentukan secara acak kecuali untuk *level* 1. Semakin

besar jumlah planet yang didapat maka nilai bobot *level* akan semakin besar.

- Jumlah AI, Jumlah musuh yang dihadapi di setiap *level*. Mulai dari *level* 1 sampai 9 jumlah AI adalah 1. Untuk *level* 10 ke atas jumlah AI yang akan dihadapai oleh pemain adalah 2.
- *Regen* AI, kecepatan regenerasi sumber daya planet yang sudah dikuasai oleh AI. Nilai regen AI untuk *level* 1 adalah 3, sedangkan untuk *level* 2 sampai 15, nilai regen AI diacak dari 1 sampai 3. Semakin kecil nilai regen AI maka kecepatan regen akan semakin cepat dan level tersebut akan semakin sulit.
- Bobot *level*, nilai yang digunakan untuk menentukan tingkat kesulitan suatu *level*. Semakin tinggi nilai sebuah *level* maka nilai bobot di *level* tersebut juga semakin besar. Nilai bobot dipengaruhi dari jumlah planet, jumlah musuh, dan kecepatan regenerasi AI. Rumus 3.1 digunakan untuk mendapatkan nilai suatu bobot. Jika nilai bobot melebihi atau kurang dari batas bobot yang telah ditentukan, maka perhitungan untuk mencari nilai bobot di *level* tersebut akan terus dilakukan dengan mengacak nilai dari jumlah planet dan kecepatan regenerasi AI.

$$B = \frac{p - minP}{maxP - minP} bP + \frac{AI - minAI}{maxAI - minAI} bAI + \frac{r - minR}{maxR - minR} \quad (3.1)$$

B	=	Nilai bobot dari suatu *level*.
p	=	hasil acak dari jumlah planet yang dimainkan di *level* tersebut.
minP	=	nilai minimal jumlah planet yang bisa dimainkan yaitu 5.
maxP	=	nilai maksimal jumlah planet yang bisa dimainkan yaitu 20.
bP	=	bobot dari tingkat kesulitan jumlah planet yang bernilai 0,25.
AI	=	jumlah AI di *level* yang dimainkan semakin banyak maka bobot *level* semakin besar .

minAI = jumlah minimal AI yaitu 1.
maxAI = jumlah maksimal AI yaitu 3.
bAI = bobot dari tingkat kesulitan jumlah AI yang bernilai 0,25.
r = nilai acak kecepatan regenerasi planet yang dikuasai oleh AI. Semakin kecil nilai regen AI semakin tinggi nilai bobot *level*.
minR = nilai minimal kecepatan regenerasi yaitu 1.
maxR = nilai maksimal kecepatan regenerasi yaitu 2.
bR = bobot dari tingkat kesulitan regenerasi AI yang bernilai 0,5. Nilai bobot regen AI lebih besar dari bobot jumlah planet dan bobot AI karena pengaruh nilai regen AI lebih besar dari atribut yang lainnya.

Tabel 3.6 Perancangan Level

Level	Jumlah Planet	Jumlah AI	Regen AI	Bobot *level*
1	5	1	3	0
2	6 – 16	1	1 – 3	0,03 – 0,06
3	6 – 16	1	1 – 3	0,095 – 0,125
4	6 – 16	1	1 – 3	0,16 – 0,19
5	6 – 16	1	1 – 3	0,225 – 0,255
6	6 – 16	1	1 – 3	0,29 – 0,32
7	6 – 16	1	1 – 3	0,355 – 0,385
8	6 – 16	1	1 – 3	0,42 – 0,45
9	6 – 16	1	1 – 3	0,485 – 0,515
10	15 – 20	2	1 – 3	0,55 – 0,58
11	15 – 20	2	1 – 3	0,615 – 0,645
12	15 – 20	2	1 – 3	0,68-0,71
13	15 – 20	2	1 – 3	0,745 – 0,775
14	15 – 20	2	1 – 3	0,81 – 0,84
15	15 - 20	2	1 – 3	0,875 – 0,905

3.3.2 Perancangan Pembuatan Planet

Hal pertama yang dilakukan dalam membuat *world* untuk sebuah *level* adalah membuat planet – planet yang jumlahnya sudah ditentukan. Penentuan posisi planet dilakukan secara acak. Meskipun penempatan planet dilakukan secara acak, planet tidak boleh saling bertabrakan atau terlalu dekat satu sama lain, Sehingga dibutuhkan jarak minimal antar planet.

3.3.3 Perancangan Pembuatan Edge

Setelah menentukan posisi planet secara acak, hal yang dilakukan selanjutnya adalah membuat *edge*. *Edge* digunakan untuk menghubungkan planet – planet yang telah dibuat sebelumnya. Tahap – tahap yang dilakukan dalam pembuatan *edge* :

1. Membuat *edge* yang tidak memotong planet yang lain.
2. Mengeliminasi *edge* yang berpotongan dengan *edge* yang lain.

3.3.3.1 Perancangan Membuat *Edge* yang Tidak Memotong Planet

Cara yang digunakan untuk menentukan jika sebuah *edge* memotong atau bersinggungan dengan planet lain adalah dengan mencari nilai diskriminan. Nilai diskriminan didapat dari menggabungkan rumus persamaan garis dan rumus lingkaran.

Nilai dari rumus persamaan garis didapat setelah mendapatkan nilai dari rumus kemiringan garis. Rumus kemiringan garis membutuhkan nilai sumbu x dan sumbu y dari kedua planet yang dihubungkan oleh garis (*edge*) tersebut. Rumus persamaan garis adalah sebagai berikut.

$$m = \frac{y2 - y1}{x2 - x1} \tag{3.2}$$

m = nilai kemiringan garis
y2 = nilai sumbu y dari planet kedua
y1 = nilai sumbu y dari planet pertama
x2 = nilai sumbu x dari planet kedua
x1 = nilai sumbu x dari planet pertama

Setelah mendapatkan nilai dari kemiringan garis, nilai tersebut dapat dimasukkan ke dalam rumus persamaan garis. Rumus persamaan garis adalah sebagai berikut.

$$y = m\,x + c \tag{3.3}$$

y = nilai sumbu y
m = nilai kemiringan garis
x = nilai sumbu x
c = konstanta

Nilai konstanta pada rumus 3.3 dapat dicari dengan memasukan nilai x dan y sesuai dengan nilai sumbu x dan y dari salah satu planet yang digunakan untuk mencari nilai kemiringan garis. Rumus persamaan lingkaran adalah sebagai berikut.

$$x^2 + y^2 + Ax + By + C = 0 \tag{3.4}$$

$$A = -2\,x_3$$
$$B = -2\,y_3$$

Nilai dari x_3 dan y_3 adalah titik pusat planet yang ingin ditentukan jika planet tersebut memotong garis dari persamaan

sebelumnya. Dari rumus persamaan lingkaran di atas, dimasukkan rumus persamaan garis sebagai berikut.

$$x^2 + y^2 + Ax + By + C = 0$$

$$x^2 + (mx + c)^2 + Ax + B(mx + c) + C = 0$$

$$x^2 + (mx)^2 + 2mxc + c^2 + Ax + Bmx + Bc + C = 0$$

$$(m + 1)x^2 + (2mc + A + Bm)x + c^2 + Bc + C = 0 \quad (3.5)$$

Dari rumus gabungan persamaan garis dan persamaan lingkaran di rumus 3.5, dapat dicari nilai diskriminannya. Rumus diskriminan dapat dilihat di rumus 3.6 berikut.

$$D = b'^2 - 4a'c' \quad (3.6)$$

$$a' = m^2 + 1$$
$$b' = 2mc + A + Bm$$
$$c' = c^2 + Bc + C$$

Jika nilai diskriminan lebih kecil dari 0, maka garis berada di luar lingkaran. Jika nilai diskriminan adalah 0, maka garis menyinggung lingkaran. Jika nilai diskriminan lebih besar dari 0, maka garis memotong lingkaran di dua titik yang berbeda.
Jika ada *edge* yang memotong sebuah lingkaran, maka garis itu tidak akan dibuat. Sedangkan jika *edge* tidak memotong satupun planet, maka *edge* tersebut akan dibuat.

3.3.3.2 Perancangan Mengeliminasi *Edge*

Tahap selanjutnya dalam pembuatan *edge* yaitu mengeliminasi *edge* yang berpotongan dengan *edge* yang lain. Untuk mengetahui jika sebuah *edge* berpotongan dengan *edge* yang lain

adalah dengan mencari nilai kemiringan kedua *edge* tersebut menggunakan rumus 3.2. Jika nilai kemiringan kedua *edge* tersebut tidak sama atau kemiringan *edge* pertama dikali dengan kemiringan *edge* yang kedua sama dengan -1, maka kedua *edge* tersebut berpotongan satu sama lain.

Jika *edge* saling berpotongan satu sama lain, maka harus ditentukan titik potong dari kedua *edge* tersebut. Rumus yang digunakan untuk menentukan titik potong kedua *edge* tersebut adalah rumus kemiringan garis seperti rumus 3.2 dan rumus persamaan garis seperti rumus 3.3. Jika sudah ditemukan titik potong dari kedua *edge* tersebut, dicari apakah titik potongnya berada di wilayah dalam dari keempat planet yang dihubungkan oleh salah satu dari kedua *edge* yang berpotongan. Jika titik potongnya berada di wilayah dalam maka salah satu dari *edge* yang berpotongan tersebut harus dieliminasi. *Edge* yang dieliminasi adalah *edge* yang nilai jaraknya paling besar. Jika titik potongnya berada di wilayah luar dari keempat planet yang terhubung oleh salah satu *edge* yang berpotongan, maka kedua *edge* tersebut tidak perlu dieliminasi.

3.3.4 Algoritma Djikstra

Algoritma Djikstra pada *game* yang dibuat ini, digunakan untuk mencari jarak terpendek sebuah *graph* atau *world* yang sudah dibuat. Algoritma Djikstra digunakan untuk menemukan jarak terpendek dari dua planet. Selain mentukan jarak terpedek, algoritma djikstra juga dapat menetukan planet – planet yang harus dilewati sebelum sampai ke planet yang dituju.

Dalam pembuatan *world generator*, algoritma djikstra digunakan untuk mendapatkan data jarak terpendek suatu planet ke semua planet dan jumlah planet yang harus dilewati untuk sampai ke sebuah planet. Data ini nantinya akan digunakan untuk inisialisasi planet awal dan inisialisasi nilai awal pasukan dari planet netral.

3.3.5 Inisialisasi Planet Awal

Inisialisasi Planet Awal dilakukan untuk menentukan planet pertama yang dimiliki oleh pemain dan musuh. Inisialisasi planet awal menggunakan data yang dihasilkan oleh algoritma djikstra. Data yang diambil yaitu data jumlah planet yang harus dilewati untuk sampai ke planet yang lain. Nilai yang pertama diambil dari nilai yang paling besar dari jumlah minimal planet yang harus dilewati. Jika ada beberapa nilai yang memiliki jumlah yang sama, maka dicari nilai jarak terpendek yang paling besar. Hal ini dilakukan jika pemain lawan hanya satu. Jika pemain lawan ada dua, maka dibutuhkan rumus 3.7, rumus 3.8, dan rumus 3.9.

$$avg = \frac{jp1 + jp2 + jp3}{3} \tag{3.7}$$

avg = rata – rata jarak ketiga planet
jp1 = jumlah planet yang harus dilewati dari planet 1 ke planet 2
jp2 = jumlah planet yang harus dilewati dari planet 1 ke planet 3
jp3 = jumlah planet yang harus dilewati dari planet 2 ke planet 3

$$totalDelta = delta1 + delta2 + delta3 \tag{3.8}$$

$$delta1 = |jp1 - avg|$$
$$delta2 = |jp2 - avg|$$
$$delta3 = |jp3 - avg|$$

Delta1, delta2, delta3 merupakan selisih jarak antar setiap planet awal dengan rata – rata jarak antar setiap planet awal. Rumus 3.8 digunakan untuk mendapatkan jumlah dari delta1, delta2, dan

delta3 yang nantinya digunakan untuk mendapatkan nilai bobot dari ketiga planet awal.

$$B[i,j,k] = \frac{totalDelta - minDelta}{maxDelta - minDelta}\ 0.6 + \frac{avg - minAvg}{maxAvg - minAvg}\ 0.4 \quad (3.9)$$

B[i, j, k] = nilai bobot jika posisi awal di planet – i, planet – j, dan planet – k. Nilai bobot yang terbesar dijadikan tolak ukur untuk menentukan posisi awal dari ketiga planet awal yaitu i, j, k.

maxDelta = nilai terbaik yang didapat dari total delta yaitu 0 karena semakin kecil nilai totalDelta, maka semakin besar pengaruhnya dalam bobot B.

minDelta = nilai maksimal total selisih ketiga planet dengan jarak rata – rata. Jika nilai totalDelta semakin besar maka pengaruhnya dalam bobot di rumus 3.9 semakin kecil. Sehingga minDelta diberi nilai 20.

maxAvg = nilai maksimal jarak rata – rata ketiga planet diberi nilai 15. Karena nilai maksimal jarak rata – rata ketiga planet tidak melebihi 15.

minAvg = nilai jarak rata – rata ketiga planet yang paling kecil yaitu 0.

Rumus 3.9 digunakan untuk mencari jarak ketiga planet awal yang paling seimbang. Dalam implementasinya, rumus 3.9 dilakukan berulang – ulang kali. Nilai bobot yang paling tinggi

adalah nilai yang dipilih. Dari nilai ini posisi planet awal adalah planet – i, planet – j, dan planet – k dari rumus 3.9.

3.3.6 Inisialisasi Nilai Pasukan Planet Netral

Semua planet yang tidak dikuasai oleh pemain maupun musuh disebut dengan planet netral. Planet netral memiliki nilai pasukan tersendiri sehingga jika pemain atau musuh ingin menguasai sebuah planet netral, maka pemain atau musuh tersebut harus mengirimkan pasukan yang memiliki nilai lebih tinggi atau sama dari planet netral tersebut.

Untuk dapat membuat *world* yang seimbang, maka diperlukan perhitungan untuk menentukan jumlah pasukan dari planet – planet netral. Perhitungan dimulai dengan mengisi pasukan planet – planet netral yang terhubung dengan planet pertama yang dimiliki oleh pemain dan musuh.

Planet netral yang diberikan nilai pasukan terlebih dahulu adalah planet – planet netral yang terhubung langsung dengan dengan planet awal pemain dan planet awal musuh. Planet – planet ini disebut tetangga satu karena langsung terhubung dengan planet awal pemain dan musuh sedangkan planet netral yang harus melewati sebuah planet lain untuk sampai ke planet awal pemain atau musuh disebut tetangga dua dan seterusnya. Rumus yang digunakan adalah sebagai berikut.

$$tPs = 6pt + \left(\frac{avgP}{totalDt} - \frac{pDt}{totalDt}\right) totalPt \tag{3.10}$$

tPs = total pasukan untuk semua tetangga planet awal pemain atau musuh.
pt = jumlah planet tetangga untuk satu pemain atau musuh.
totalDt = total jarak semau planet awal ke tetangganya.
avgP = totalDt dibagi dengan total pemain.
pDt = jarak planet awal pemain atau musuh ke

semua tetangganya.

totalPt = total jumlah planet tentangga dari planet awal semua pemain.

Rumus 3.10 digunakan untuk mencari total pasukan dari planet tetangga pemain atau musuh. Rumus 3.10 melibatkan jarak planet awal ke planet tetangga dan juga total planet tetangga. Hal tersebut diperlukan untuk membuat planet netral yang memiliki jarak semakin panjang dari planet awal, nilai pasukannya semakin kecil.

$$tPs = tPs - 3\ (maxPt - pt) \tag{3.11}$$

maxPt = jumlah planet tetangga yang paling besar antara pemain atau musuh.

Rumus 3.11 digunakan untuk mengurangi total pasukan dari planet tentangga yang jumlah tetangganya lebih sedikit dari pada jumlah tetangga planet awal dari pemain yang lain.

$$Ps[i] = avgPs + tPs(\frac{pDt}{pt} - \frac{dt[i]}{pDt}) \tag{3.12}$$

Ps[i] = nilai pasukan planet netral di planet – i.

avgPs = Total pasukan untuk semua tetangga suatu planet awal dibagi dengan jumlah planet tetangga.

dt = jarak sebuah planet awal ke salah satu planet tetangganya yaitu planet - i.

Rumus 3.12 digunakan untuk menentukan nilai pasukan sebuah planet netral. Semakin jauh jarak planet tersebut dari planet awal terdekat maka nilai pasukan di planet netral tersebut akan semakin kecil.

(Halaman Ini Sengaja Dikosongkan)

BAB IV
IMPLEMENTASI

Pada bab ini akan dibahas mengenai implementasi dari perancangan perangkat lunak. Di dalamnya mencakup proses penerapan dan pengimplementasian algoritma, dan antar muka yang mengacu pada rancangan yang telah dibahas sebelumnya.

4.1 Lingkungan Implementasi

Lingkungan implementasi dari tugas akhir dijelaskan pada Tabel 4.1.

Tabel 4.1 Lingkungan Implementasi Perangkat Lunak

Perangkat Keras	Prosesor: Intel(R) Core(TM) i5 – 4210U CPU @ 1.70GHz 2.40GHz Memori: 8 GB
Perangkat Lunak	Sistem Operasi: Microsoft Windows 8.1 64-bit Perangkat Pengembang: Unity Game Engine v5.0.0f4 Direct X Version : DirectX 11

4.2 Implementasi Permainan

Implementasi dari masing-masing fungsi utama dituliskan menggunakan *pseudocode* berdasarkan antarmuka utama yang ada pada permainan. Penjelasan implementasi hanya berupa antar muka yang berhubungan dengan keseimbangan *game* dan fitur – fitur pembuatan *world* diantaranya **Level Selection** dan **Game Play**.

4.2.1 Implementasi Layar Level Selection

Pada layar **Level Selection** terdapat sebuah tombol menu dan 15 tombol berbentuk kotak yang di masing – masing kotak terdapat angka yang merepresentasikan *level*. Tombol menu digunakan jika pemain ingin pergi ke layar **Main Menu**. Tombol

lain yang bebentuk kotak digunakan jika pemain ingin memulai permainan sesuai dengan *level* yang diinginkan. Jika pemain tidak pernah memainkan *game* ini sebelumnya, maka pemain harus memulai permainan dari *level* 1.

Fungsi 'generateMap' pada Gambar 4.1 dijalankan saat pemain memilih *level* dan digunakan untuk menentukan jumlah planet, jumlah musuh, dan kecepatan regenerasi musuh sesuai dengan *level* yang dipilih.

```
// fungsi MapGenerator //
weight_prev <- 0.065 * (level - 2)
if (level != 1)
   do {
      if (weight_prev >= 0.52)
         planet <- random (15 , maxPlanet)
         ai = 2
      else
         planet <- random(minPlanet + level – 1, 17)
         ai = 1
      aiRegen <- random (minAiRegen, maxAiRegen)
      weight_now <- ((planet – minPlanet) / (maxPlanet
      - minPlanet) * planet_weight ) + ((aiRegen -
      minAiRegen) / (maxAiRegen - minAiRegen) *
      aiRegen_weight) + ((ai - minAi) / (maxAi -
      minAi) * ai_weight)
   }
   while (weight_now – weight_prev < 0.03 ||
   weight_now – weight_prev > 0.06)
```

Gambar 4.1 Pseudocode perhitungan Atribut Level

4.2.2 Implementasi Layar Game Play

Fungsi – fungsi yang digunakan untuk membuat *world* yang adil terdapat pada layar **Game Play**. Fungsi 'creatingPlanet' pada Gambar 4.2 merupakan fungsi yang digunakan untuk menentukan posisi semua planet yang ada di *level* yang dimainkan oleh pemain. Penentuan posisi semua planet ini dilakukan secara acak. Posisi planet ditentukan dengan mengambil nilai acak dari sumbu x atau sumbu y terlebih dahulu.

Setelah mengambil nilai acak, kemudian ditentukan planet – planet yang jaraknya terlalu dekat dengan sumbu x atau y dari nilai acak tersebut. Kemudian cari nilai yang tidak terlalu dekat dengan planet lain dengan sumbu berlawanan dari sumbu yang dipilih. Kemudian nilai tersebut dipilih secara acak untuk menetukan nilai dari sumbu berlawanan dengan sumbu yang pertama kali dipilih.

```
// Fungsi creatingPlanets() //
for i = 1 to numberOfPlanets
   axis <- random (x, y)
   if (axis = x)
      valx = random (xMin, xMax)
      clashPlanets <- findCrashedPlanets()
      emptySpaces <- findEmptySpaces()
      valy = random (emptySpaces)
   else
      valy = random (yMin, yMax)
      clashPlanets <- findCrashedPlanets()
      emptySpaces <- findEmptySpaces()
      valx = random (emptySpaces)
   planets(i).x <- valx
   planets(i).y <- valy
```

Gambar 4.2 Pseudocode Pembuatan Planet

Fungsi ‘creatingEdges()’ pada Gambar 4.3 digunakan untuk membuat *edge* yang saling menghubungkan beberapa planet.

```
// Fungsi creatingEdges() //
eliminateFirstEdges()
eliminateSecondEdges()
```

Gambar 4.3 Pseudocode Pembuatan Edge

Fungsi ‘eliminataFirstEdges’ pada Gambar 4.4 digunakan untuk membuat planet yang tidak memotong atau terlalu dekat dengan planet lain. Cara agar *edge* yang terlalu dekat dengan

sebuah planet tidak dibuat adalah dengan memperbesar jari – jari dari titik pusat suatu planet walaupun secara visual jari – jari planet lebih kecil.

```
// Fungsi eliminateFirstEdges() //
for i = 1 to numberOfPlanets – 1
   for j = i + 1 to numberOfPlanets
      innerP <- innerPlanet(i, j)
      makeEdge <- true
      for each planet in innerP
         m    <-    planets[i].y    –    planets[j]    /
         planets[i].x – planets[j].x
         c <- planets[i].y – (m * planets[i].x)
         A <- -2 * planet.x
         B <- -2 * planet.y
         C <- (planet.x)² + (planet.y)² - 4
         D <- ((2 * m * c) + (B * m)) * ((2 * m * c)
         + A + (B * m)) – (4 * (m² + 1) * (c² + (B *
         c) + C))
         if D > 0
            makeEdge <- false
   if(makeEdge = false)
      edge[i, j] <- false
   else
      edgeList[n, 0] <- i
      edgeList[n, 1] <- j
      n++
```

Gambar 4.4 Pseudocode Eliminasi Edge Pertama

Fungsi ‘eliminateSecondEdges’ pada Gambar 4.5 digunakan untuk mengetahui *edge* yang saling berpotongan dan menghapus *edge* yang memiliki jarak paling besar dari *edge* yang berpotongan tersebut. Di dalam fungsi ini juga terdapat fungsi ‘innerCross’. Fungsi ‘innerCross’ digunakan untuk menetukan letak dari titik potong dari kedua *edge* yang saling berpotongan. Fungsi ‘innerCross’ akan memberikan nilai *true* jika titik potongnya berada di wilayah dalam dari keempat planet yang terhubung oleh salah satu *edge* yang berpotongan. Dan fungsi

'innerCross' akan memberikan nilai *false* jika titik potong berada di wilayah luar dari keempat titik yang terhubung oleh salah satu *edge* yang berpotongan.

```
// Fungsi eliminateSecondEdges() //
for i = 1 to n
   for j = i + 1 to n
      m1 <- edgeList[i, 0].y - edgeList[i, 1].y /
      edgeList[i, 0].x - edgeList[i, 1].x
      m2 <- edgeList[j, 0].y - edgeList[j, 1].y /
      edgeList[j, 0].x - edgeList[j, 1].x
      if (m1 != m2 && innerCross())
          edge[i, j] <- false
```

Gambar 4.5 Pseudocode Eliminasi Edge Kedua

Fungsi 'initMap' pada Gambar 4.6 digunakan untuk menentukan planet awal yang dimiliki oleh pemain dan musuh. Dalam fungsi ini juga ditentukan nilai pasukan yang dimiliki oleh planet – planet netral. Tujuan dari fungsi ini agar *world* yang telah dibuat menjadi lebih adil bagi pemain dan musuh.

```
// Fungsi initMap() //
Djikstra()
initPosition()
fillPlanetValue()
```

Gambar 4.6 Pseudocode untuk menentukan Keadaan Awal Level

Fungsi 'Djikstra' digunakan untuk mendapatkan data – data seperti jarak terpendek ke planet lain dan jumlah planet yang harus dilewati untuk sampai ke planet yang lain dengan menggunakan algoritma djikstra.

```
// Fungsi Djikstra //
for each planet in Planets
   dist[v] <- infinity
   add v to Q
while Q is not empty
   u <- vertex in Q with min dist[u]
   remove u from Q
      for each neighbor v of u
         totalDist <- dist[u] + length(u, v)
         if (totalDist < pDist[v])
            pDist[v] = totalDist
            pDist2[v] = pDist2 + 1
```

Gambar 4.7 Pseudocode untuk Algoritma Djikstra

Fungsi 'initPosition' pada Gambar 4.8 digunakan untuk menentukan posisi awal planet yang dimiliki pemain dan musuh. Jika musuh hanya ada satu. Maka planet awal yang dipilih adalah dua planet yang memiliki jumlah planet yang harus dilewati lebih banyak. Jika ada beberapa planet dengan jumlah yang harus dilewati sama dengan jumlah yang harus dilewati terbanyak, maka akan dipilih dua planet dengan jarak yang paling jauh.

Jika musuh ada dua maka total planet awal dalam *world* yang dibuat ada tiga. Planet awal yang dipilih adalah planet yang selisih jarak planet 1 ke planet 2, planet 1 ke planet 3, dan planet 2 ke planet 3 lebih sedikit dan yang jaraknya lebih jauh.

```
// fungsi initPosition()
maxDist <- 0
maxDist2 <- 0
if (ai = 1)
   for i = 0 to numberOfPlanets
      for j = i + 1 to numberOfPlanets
         if (pDist2[i, j] > maxDist2 || (pDist2[i, j]
         == maxDist2 && pDist[i, j] > maxDist ))
            maxDist2 <- pDist2[i, j]
            maxDist <- pDist[i, j]
            inits[0] <- i
            inits[1] <- j
else if (ai = 2)
```

```
maxBobot <- 0
for i = 0 to numberOfPlanets
   for j = i + 1 to numberOfPlanets
      for k = j + 1 to numberOfPlanets
         totD <- pDist2[i, j] + pDist2[i, k] +
         pDist2[j, k]
         avg <- totD / (ai + 1)
         delta1 <- absolute(pDist2[i, j] - avg)
         delta2 <- absolute(pDist2[i, k] - avg)
         delta3 <- absolute(pDist2[k, j] - avg)
         totDelta <- delta1 + delta2 + delta3
         bobot <- ((totDelta - minTotDlta) /
         (maxTotDlta - minTotDlta) * ratio1) +
         ((totD - minTotD) / (maxTotD - minTotD) *
         ratio2)
         if (bobot > maxBobot)
            maxBobot = bobot
            inits[0] <- i
            inits[1] <- j
            inits[2] <- k
```

Gambar 4.8 Pseudocode menentukan planet awal

Fungsi 'fillPlanetValue' pada Gambar 4.9 digunakan untuk menentukan jumlah pasukan dari planet – planet netral untuk membuat *game* yang adil. Jumlah pasukan di planet netral tergantung pada jarak dan jumlah tetangga dari planet awal. Semaking panjang jarak antara planet awal dengan planet netral, maka jumlah pasukan yang dimiliki planet netral tersebut akan semakin lebih sedikit. Jika jumlah tetangga pertama planet awal pemain dan musuh sama, maka total jumlah pasukan dari semua planet netral tetangga planet awal akan sama. Tetapi jika jumlah tentangga planet awal pemain dan musuh berbeda, jumlah pasukan pemain dengan total tetangga paling sedikit akan berkurang.

```
// fungsi fillPlanetValue //
tier <- 1
while (allPlanetsFilled = false)
   p1t <- totalPlanets(tier, inits[0])
```

```
p1Tiers <- listPlanets(tier, inits[0])
p1Dist <- totalLenghtPlanets(tier, inits[0])
p2t <- totalPlanets(tier, inits[1])
p2Tiers <- listPlanets(tier, inits[1])
p2Dist <- totalLenghtPlanets(tier, inits[1])
maxPt <- maxTotalPlanet(p1t, p2t)
tierTotalD <- p1Dist + p2Dist
tierTotalP <- 6 * (p1t + p2t)
if (ai = 2)
   p3t <- totalPlanets(tier, inits[2])
   p3Tiers <- listPlanets(tier, inits[2])
   maxPt <- maxTotalPlanet(p1t, p2t, p3t)
   p3Dist <- totalLenghtPlanets(tier, inits[2])
   tierTotalD <- p1Dist + p2Dist + p3Dist
   tierTotalP <- 6 * (p1t + p2t + p3t)
avg <- tierTotalD / (ai + 1)
b1 <- (6 * p1t) + (((avg / tierTotalD) - (p1Dist /
tierTotalD))* tierTotalP)
baseRes1 <- b1 / p1t
for i = 1 to p1t
   newRes <- (((p1Dist / p1t) - pDist[inits[0],
   p1Tiers[i]) / p1Dist) * b1
   p1Tiers[i].resource = baseRes1 + newRes
b2 <- (6 * p2t) + (((avg / tierTotalD) - (p2Dist /
tierTotalD))* tierTotalP)
baseRes2 <- b2 / p2t
for i = 1 to p2t
   newRes <- (((p2Dist / p2t) - pDist[inits[1],
   p2Tiers[i]) / p2Dist) * b2
   p2Tiers[i].resource = baseRes2 + newRes


if (ai = 2)
   b3 <- (6 * p3t) + (((avg / tierTotalD) -
   (p3Dist / tierTotalD))* tierTotalP)
   baseRes3 <- b3 / p3t
   for i = 1 to p3t
      newRes <- (((p3Dist / p3t) - pDist[inits[2],
      p3Tiers[i]) / p3Dist) * b3
      p3Tiers[i].resource = baseRes3 + newRes
tier++
```

Gambar 4.9 Pseudocode Pengisian Nilai Planet Netral

BAB V
PENGUJIAN DAN EVALUASI

Pada bab ini akan dijelaskan mengenai rangkaian uji coba dan evaluasi yang dilakukan. Proses pengujian dilakukan menggunakan metode kotak hitam berdasarkan skenario yang telah ditentukan dan pengujian dilakukan dengan survei langsung kepada pengguna.

5.1 Lingkungan Uji Coba

Lingkungan pelaksanaan uji coba meliputi perangkat keras dan perangkat lunak yang akan digunakan pada sistem ini. Spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam rangka uji coba perangkat lunak ini dicantumkan pada Tabel 5.1.

Tabel 5.1 Lingkungan Uji Coba Perangkat Lunak

Perangkat Keras	Prosesor: Intel(R) Core(TM) i5-4210U CPU@ 1.70 GHz 2.40GHz Memori: 8 GB
Perangkat Lunak	Sistem Operasi: Microsoft Windows 8.1 64-bit Unity Game Engine v5.0.0f4 Direct X Version : DirectX 11

5.2 Pengujian Fungsionalitas dengan metode black-box

Pengujian ini dilakukan untuk menguji apakah fungsionalitas yang diidentifikasi pada tahap kebutuhan benar-benar diimplementasikan dan bekerja semestinya. Selain itu juga untuk mengetahui kesesuaian keluaran dari setiap tahapan atau langkah penggunaan fitur terhadap skenario yang dipersiapkan. Pengujian dilakukan dengan menggunakan metode *black-box.*

5.2.1 Skenario Pengujian Fungsionalitas

Skenario Pengujian fungsionalitas digunakan untuk memberikan tahap-tahap dalam pengujian sistem. Skenario ini tertera pada Tabel 5.2.

Tabel 5.2 Pengujian Permainan

	UFG01
Kondisi Awal	Pengguna berada pada layar Main Menu.
Prosedur Pengujian	Pengguna memainkan permainan hingga selesai dari beberapa fungsionalitas yang ingin diuji
Hasil yang diharapkan	Pengguna berhasil menyelesaikan permainan dan fungsionalitas permainan berjalan dengan lancar.
Hasil yang diperoleh	Pengguna berhasil menyelesaikan permainan dan fungsionalitas berjalan lancar.
Kesimpulan.	Pengujian berhasil.

5.2.1.1 Pengujian Layar Main Menu

Pengujian dimulai ketika pemain maasuk ke layar **Main Menu** seperti pada Gambar 5.1. Pemain dapat memilih tombol ‘START’ untuk masuk ke layar **Level Selection**. Pemain juga dapat memilih tombol ‘TUTORIAL’ untuk masuk ke layar **Help Page** atau memilih tombol ‘QUIT’ untuk keluar dari permainan.

Gambar 5.1 Tampilan Layar Main Menu

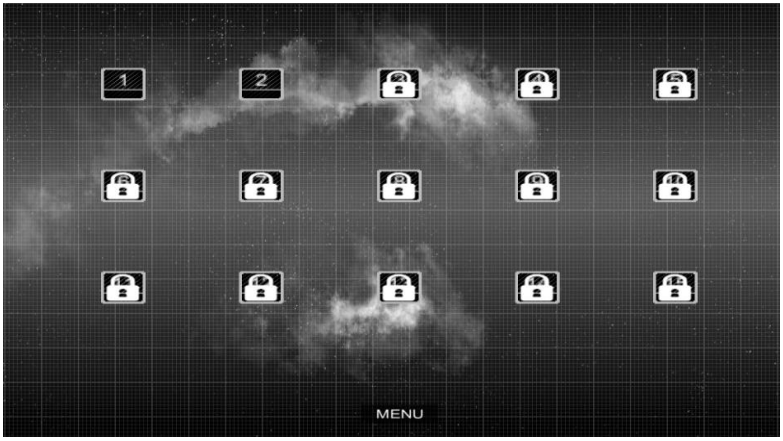
Setelah memulai permainan, layar Menu Utama akan muncul. Tampilan dan semua tombol pada tampilan ini berfungsi dengan baik.

5.2.1.2 Pengujian Layar Level Selection

Pengujian dimulai ketika pemain masuk ke layar Pemilihan Level dengan cara menekan tombol START pada layar menu utama. Pada tampilan ini disajikan level-level yang dapat dimainkan oleh pemain. Pada awalnya level yang terbuka hanya *level* yang pertama saja. Pemain hanya dapat memilih level yang terbuka saja. Tampilan layar **Level Selection** saat pertama kali bermain *game* ini dapat dilihat pada gambar Gambar 5.2. Tampilan layar **Level Selection** saat ada *level* yang terbuka dapat dilihat pada Gambar 5.3.

Gambar 5.2 Tampilan Layar Level Selection

.

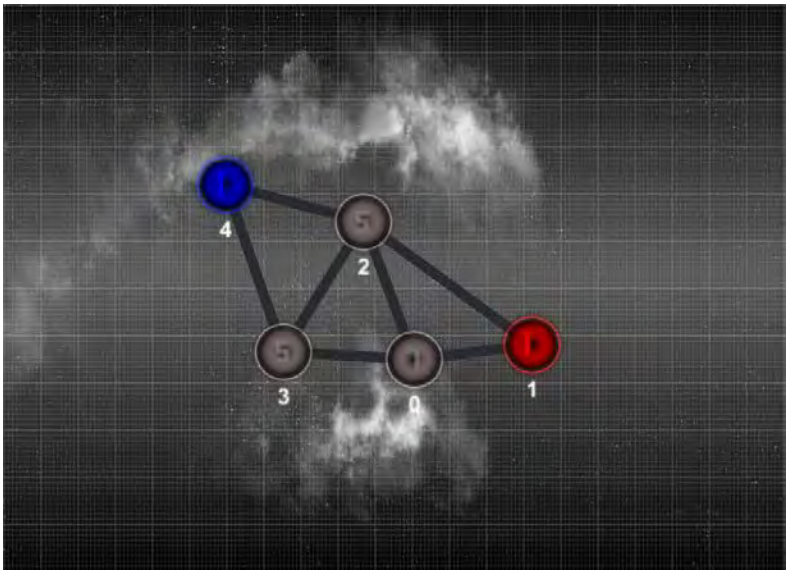


Gambar 5.3 Tampilan Layar Level Selection Setelah ada Level yang Terbuka

Kesimpulan fungsionalitas pada layar ini ialah berhasil dikarenakan tombol, sistem, dan skenario sudah diuji dan hasilnya sesuai dengan yang diharapkan.

5.2.1.3 Pengujian Game Play

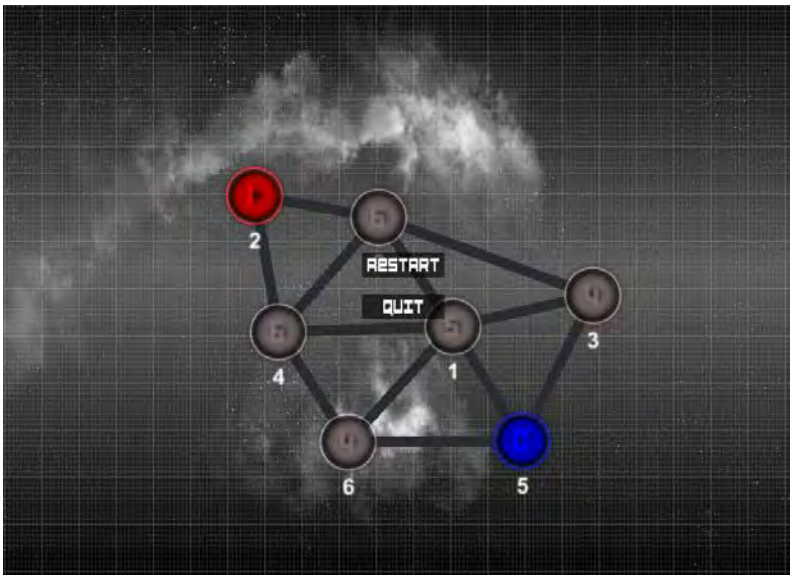
Skenario pengujian dimulai ketika pemain sudah memilih level yang ingin dimainkan. Pemain akan dibawa ke layar permainan seperti terlihat pada Gambar 5.4. Tulisan angka di bagian bawah planet adalah nomor planet tersebut.



Gambar 5.4 Tampilan Layar Game Play

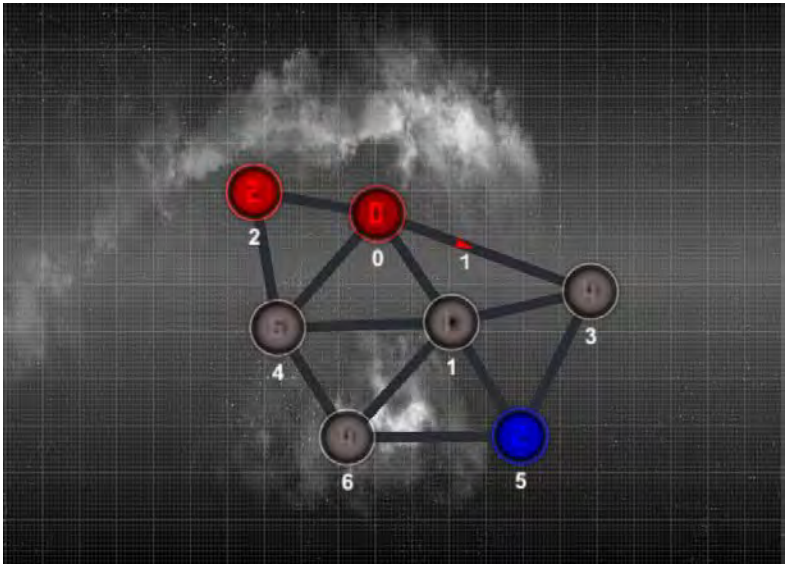
Jika pemain menekan tombol ‘PAUSE’ maka permainan akan berhenti sehingga baik pemain maupun AI tidak akan beregenerasi pasukan dan tidak bisa mengirim pasukan. Dan juga muncul dua tombol baru yaitu tombol ‘RESTART’ dan tombol ‘QUIT’ seperti yang ditunjukkan di Gambar 5.5. Tombol ‘RESTART’ digunakan jika pemain ingin memulai kembali di

level yang sama dengan *world* yang berbeda. Tombol 'QUIT' digunakan jika pemain ingin kembali ke layar **Level Selection**.



Gambar 5.5 Tampilan Layar Game Play saat Pause

Di layar **Game Play** planet yang dikuasai pemain merupakan planet yang berwarna merah seperti yang ditunjukkan pada Gambar 5.6. Planet netral ditunjukkan dengan planet yang berwarna abu – abu. Semua planet yang tidak berwarna abu – abu dapat melakukan regenerasi pasukan. Pemain dapat menekan planet yang berwarna merah untuk mengirimkan pasukan ke planet lain seperti Gambar 5.6. Jika pemain mengirimkan pasukan ke planet yang tidak berwarna merah maka pasukan pemain akan berkurang. Jika pemain mengirimkan pasukan ke planet yang berwarna merah, maka pasukan planet tersebut akan bertambah sesuai dengan jumlah pasukan yang dikirim.

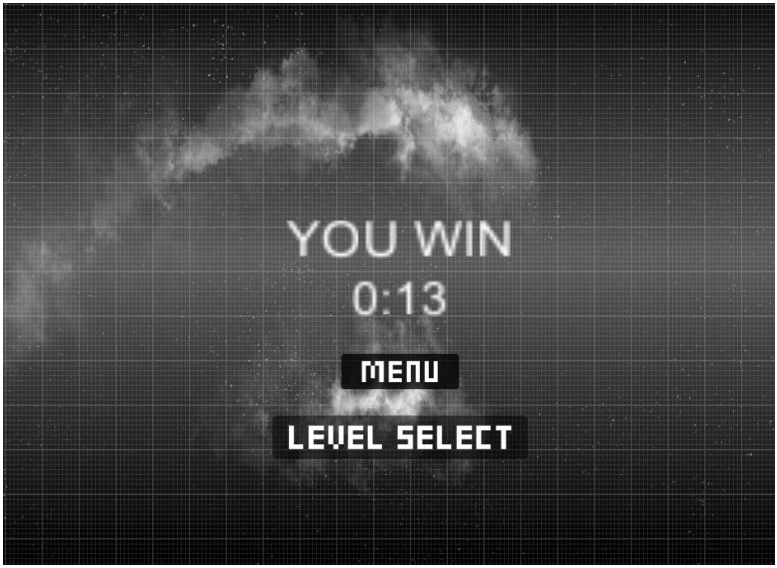


Gambar 5.6 Mengirim Pasukan dan Menguasai Planet

Kesimpulan untuk fungsionalitas pada layar **Permainan** ialah berhasil dikarenakan tombol, sistem, teks, dan skenario sudah diuji dan hasilnya sesuai dengan yang diharapkan.

5.2.1.4 Pengujian Layar Game Over

Pada layar **Game Over** terdapat 2 jenis tampilan yaitu ketika pemain menang seperti Gambar 5.7 dan ketika pemain kalah seperti Gambar 5.8. Di layar **Game Over** ditampilkan waktu yang dihabiskan pemain ketika bermain di *level* yang dimainkan. Di layar **Game Over** terdapat dua tombol yaitu tombol ‘MENU’ yang digunakan untuk masuk ke layar **Main Menu** dan tombol ‘LEVEL SELECT’ yang digunakan untuk masuk ke layar **Level Selection**.



Gambar 5.7 Tampilan Hasil Permainan ketika Menang

Gambar 5.8 Tampilan Hasil Permainan ketika Kalah

5.2.1.5 Hasil Pengujian Fungsional

Hasil uji fungsionalitas yang sudah dilakukan berdasarkan pada pengujian layar **Main Menu**, pengujian layar **Level Selection**, pengujian layar **Game Play**, pengujian layar **Game Over**. Empat uji coba yang telah dilakukan dilakukan menunjukkan bahwa semua fungsionalitas permainan berjalan dengan baik dan sesuai dengan sebagaimana mestinya skenario dan alur yang telah dibuat pada perancangan. Rekap hasil pengujian fungsionalitas dicantumkan pada Tabel 5.3.

Tabel 5.3 Hasil Pengujian Fungsionalitas

No	Kode Pengujian	Hasil Pengujian
1.	Pengujian layar main menu	Berhasil
2.	Pengujian layar level selection	Berhasil
3.	Pengujian layar Game Play	Berhasil
4.	Pengujian layar Game Over	Berhasil

5.3 Pengujian Performa World Generator

Pengujian dilakukan untuk menguji seberapa cepat sistem dalam membangkitkan *world*. Pengujian dilakukan dengan menambahkan *code* pada *script* untuk memudahan perhitungan waktu pembangkitan *world*.

5.3.1 Skenario dan Data Uji Coba Performa Kecepatan Pembuatan World

Skenario Pengujian kecepatan performa digunakan untuk memberikan tahap-tahap dalam pengujian sistem dalam hal kecepatan. Skenario ini tertera pada Tabel 5.4.

Tabel 5.4 Skenario Pengujian Performa Pembuatan World

Deskripsi	Bertujuan untuk mengetahui performa sistem dalam membuat *world*
Langkah pengujian	1. Tambahkan code untuk menghitung waktu pembangkitan pada script. 2. Jalankan scene game pada project di unity. 3. Pada *console* Unity akan menampilkan waktu pembangkitan world. 4. Ulangi langkah ke-1 sampai ke-4. 5. Cari rata-rata waktu kecepatan pembangkitan world.

```
// menghitung waktu generateMap di Level Selection//
waktuAwal <- Time.realtimeSinceStartUp
mapGenerator()
Application.LoadLevel("GamePlay")
// menghitung waktu generateMap di Game Play //
creatingPlanet()
creatingEdges()
initMap()
waktuAkhir <-realtimeSinceStartUp
print(waktuAkhir - waktuAwal)
```

Gambar 5.9 Menghitung Waktu dalam Pembuatan World

Pada Gambar 5.9, di kode sumber dalam layar **Level Selection** dan **Game Play** ditambah kode untuk menghitung

waktu mulai dari fungsi ‘mapGenerator’ hingga ke fungsi – fungsi untuk membuat *world*.

5.3.2 Hasil Pengujian Performa Kecepatan Pembuatan World

Hasil pengujian performa dikhususkan untuk menghitung waktu yang dibutuhkan dalam membangkitkan *world* dimana dijelaskan pada Tabel 5.5.

Tabel 5.5 Hasil Uji Coba Performa Pembuatan World

Level	Uji Coba ke-1	Uji Coba ke-2	Uji Coba ke-3	Uji Coba ke-4	Uji Coba ke-5	Rata-rata
1	0.10	0.09	0.10	0.10	0.09	0.096
2	0.10	0.10	0.08	0.09	0.09	0.092
3	0.09	0.09	0.09	0.09	0.09	0.09
4	0.09	0.08	0.11	0.09	0.11	0.096
5	0.14	0.10	0.10	0.10	0.13	0.114
6	0.10	0.10	0.13	0.10	0.11	0.108
7	0.12	0.13	0.11	0.10	0.12	0.116
8	0.10	0.10	0.09	0.08	0.10	0.094
9	0.10	0.10	0.12	0.10	0.11	0.106
10	0.09	0.10	0.10	0.10	0.09	0.096
11	0.13	0.11	0.10	0.11	0.10	0.11
12	0.13	0.10	0.10	0.09	0.10	0.104
13	0.12	0.09	0.10	0.10	0.09	0.1
14	0.10	0.10	0.10	0.09	0.10	0.098
15	0.12	0.09	0.09	0.10	0.10	0.1
Rata - rata						0.101

Berdasarkan Tabel 5.5 dapat disimpulkan kecepatan pembuatan *world* membutuhkan waktu rata-rata 0.101 detik. Waktu eksekusi yang cukup singkat untuk membangun sebuah *world*.

5.3.3 Skenario Pengujian World yang Dinamis

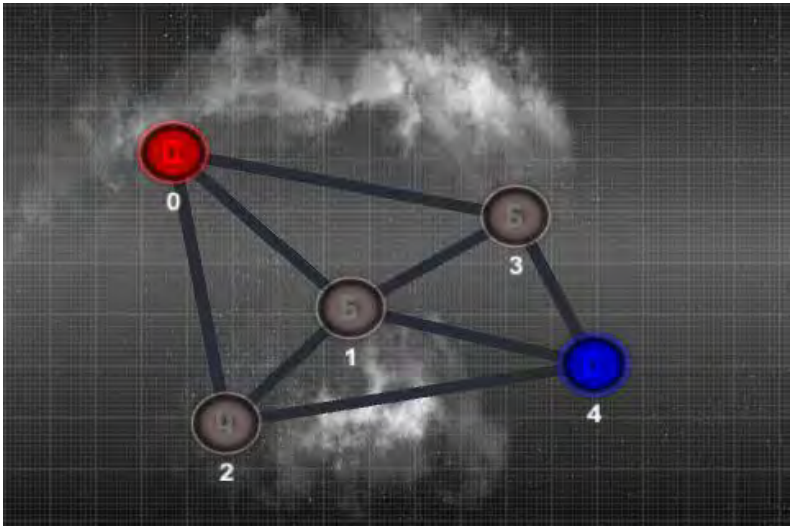
Skenario pengujian *world* yang dinamis digunakan untuk memberikan tahap - tahap dalam pengujian sistem dalam membuktikan *world* yang dinamis. Skenario ini tertera pada Tabel 5.6.

Tabel 5.6 Skenario Pengujian World yang Dinamis

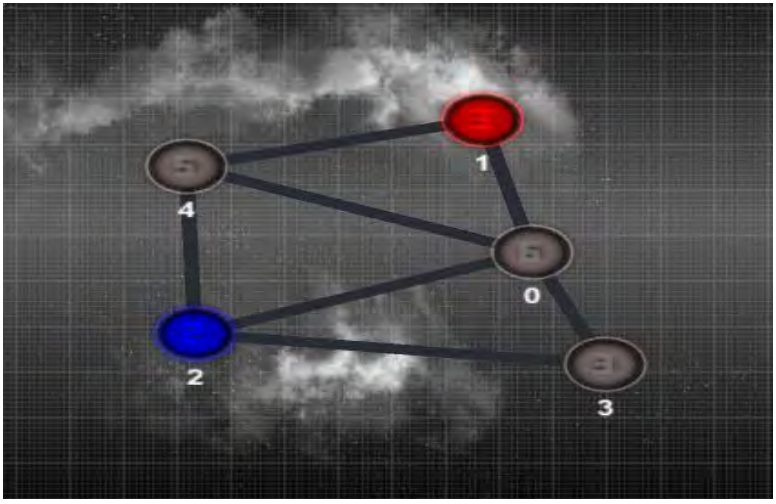
Deskripsi	Bertujuan untuk mengetahui jika *world* yang dibuat sudah dinamis
Langkah pengujian	1. Jalankan *game* sampai layar **Level Selection**. 2. Pilih level yang diinginkan 3. Klik tombol *pause* dan pilih *quit* 4. Ulangi langkah 2 sampai 4 5. Tampilkan beberapa *world* secara visual di level yang sama

5.3.4 Hasil Pengujian World yang Dinamis

Gambar 5.10 dan Gambar 5.11 merupakan hasil pembuatan *world* di *level* 1. Kedua gambar tersebut menunjukkan bahwa dengan menggunakan *world generator* sistem barhasil membuat *world* yang dinamis di *level* 1 karena secara visual *world* yang dihasilkan berbeda satu sama lain meskipun sama – sama dimainkan di *level* 1. Dan juga meskipun atribut – atribut di *level* 1 sama *world* yang dihasilkan secara visual tetap berbeda. Di Gambar 5.10 dan Gambar 5.11 planet yang dikuasai pemain berwarna merah dan planet yang dikuasai oleh AI adalah planet yang berwarna biru. Tulisan angka di bawah setiap planet menujukkan nomor dari planet tersebut.

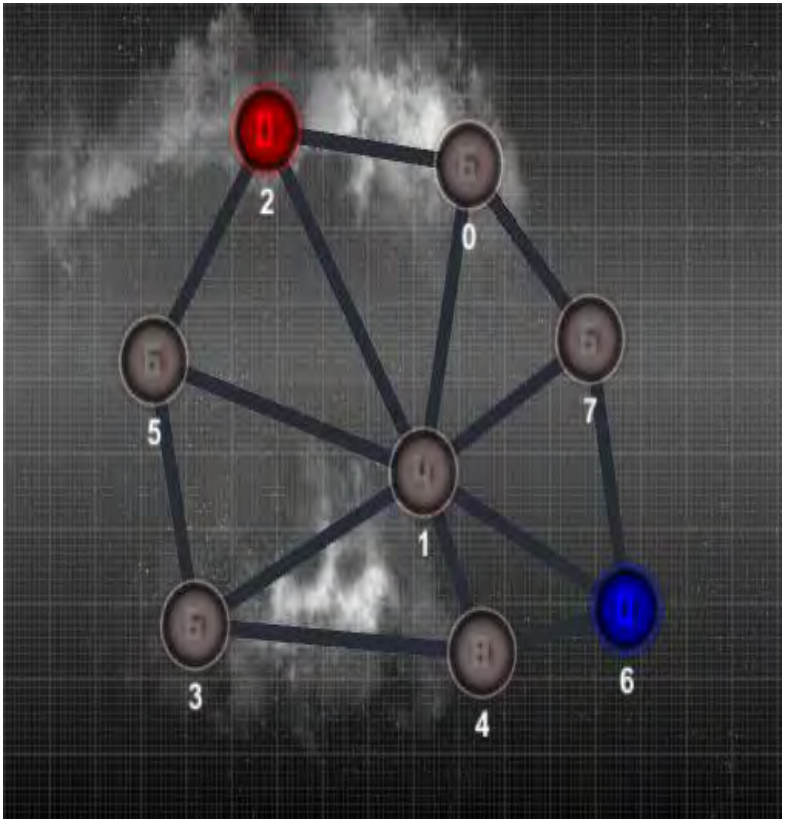


Gambar 5.10 Hasil Pembuatan World Level 1 (Bagian 1)

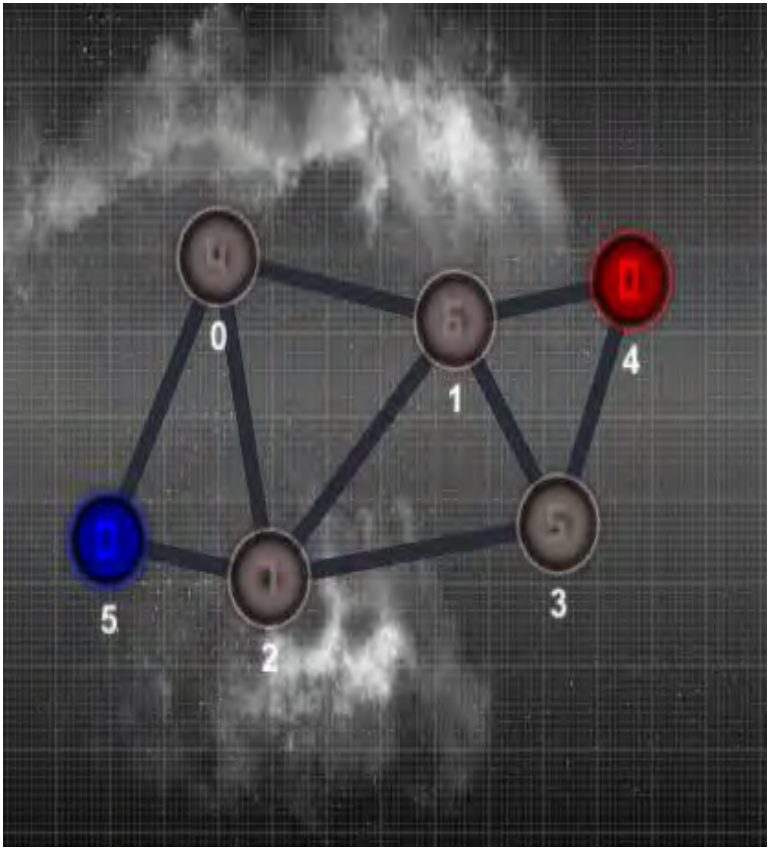


Gambar 5.11 Hasil Pembuatan World Level 1 (Bagian 2)

Gambar 5.12 dan Gambar 5.13 merupakan hasil pembuatan *world* di *level* 2. Jumlah planet di Gambar 5.12 berbeda dengan jumlah planet di Gambar 5.13 sehingga secara visual *world* yang dihasilkan pasti berbeda. Sehingga metode *world generator* berhasil membuat *world* yang dinamis di *level* 3.

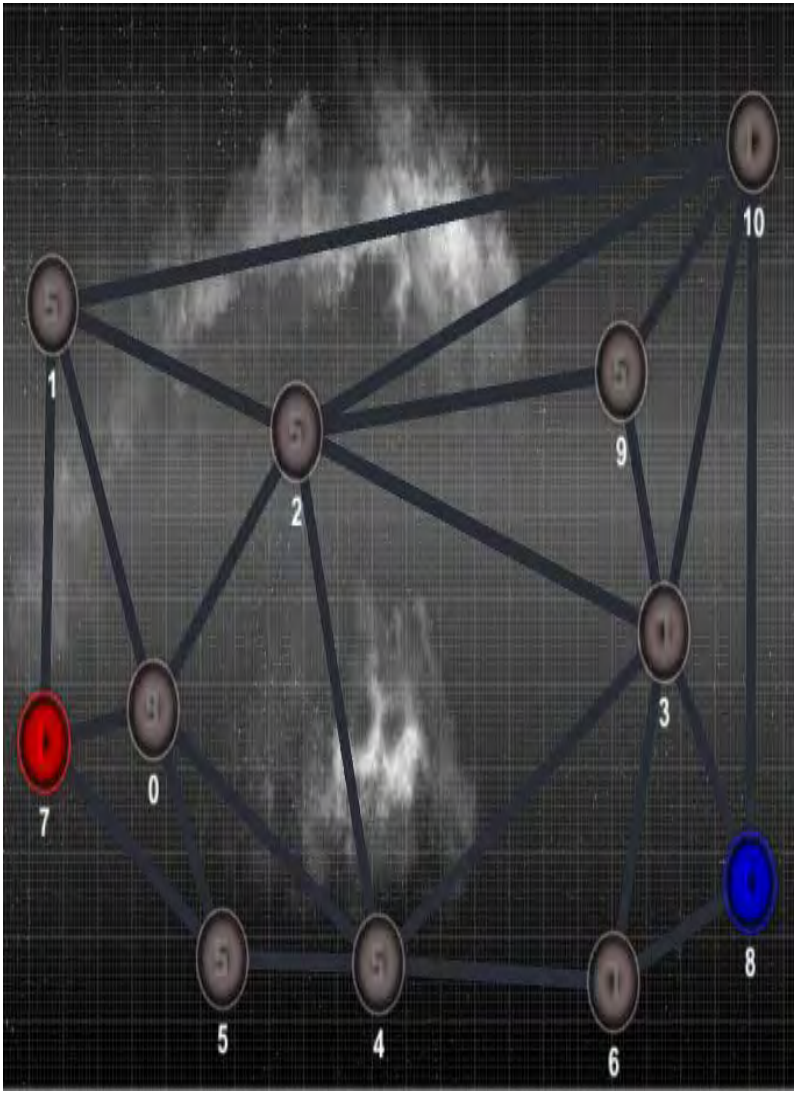


Gambar 5.12 Hasil Pembuatan World Level 2 (Bagian 1)

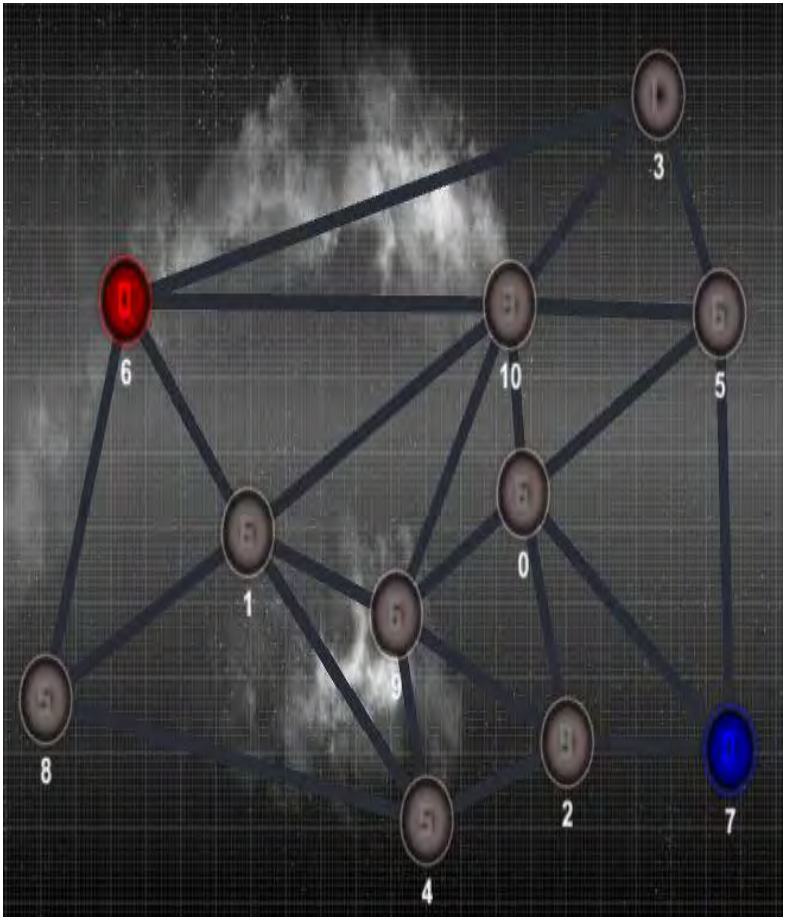


Gambar 5.13 Hasil Pembuatan World Level 2 (Bagian 2)

Gambar 5.14 dan Gambar 5.15 adalah hasil pembuatan *world* di *level* 3. *World* yang dihasilkan secara visual berbeda. Sehingga untuk *level* 3, metode *world generator* berhasil membuat *world* yang dinamis untuk *level* 3 berhasil dibuat.

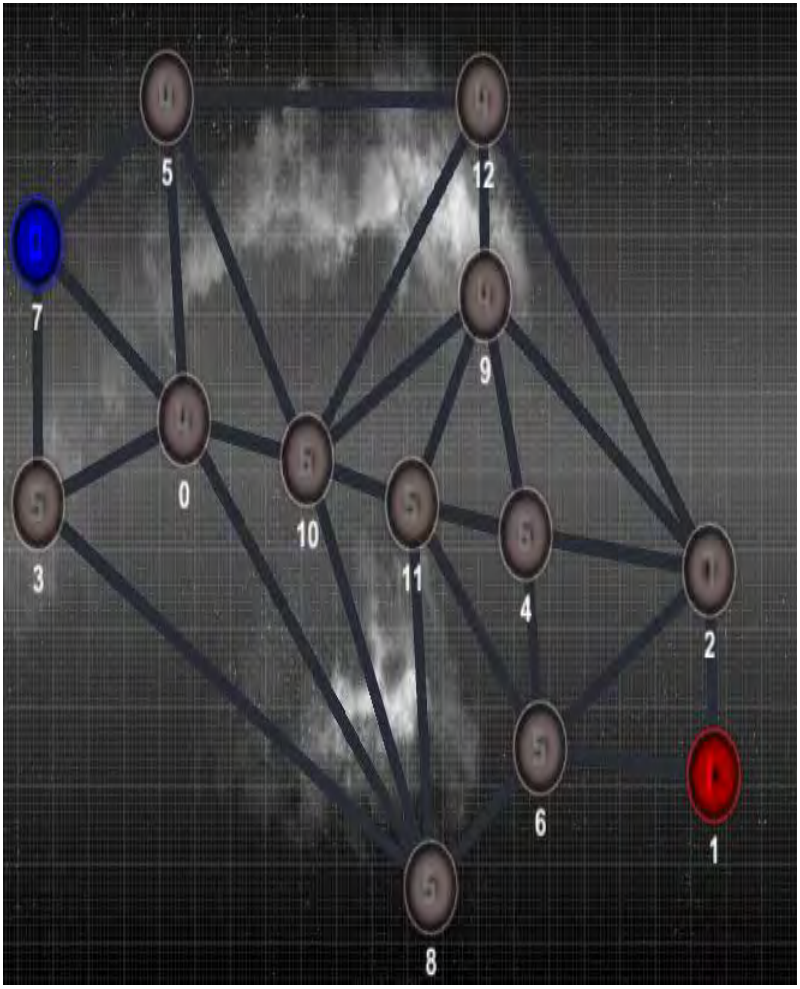


Gambar 5.14 Hasil Pembuatan World Level 3 (Bagian 1)

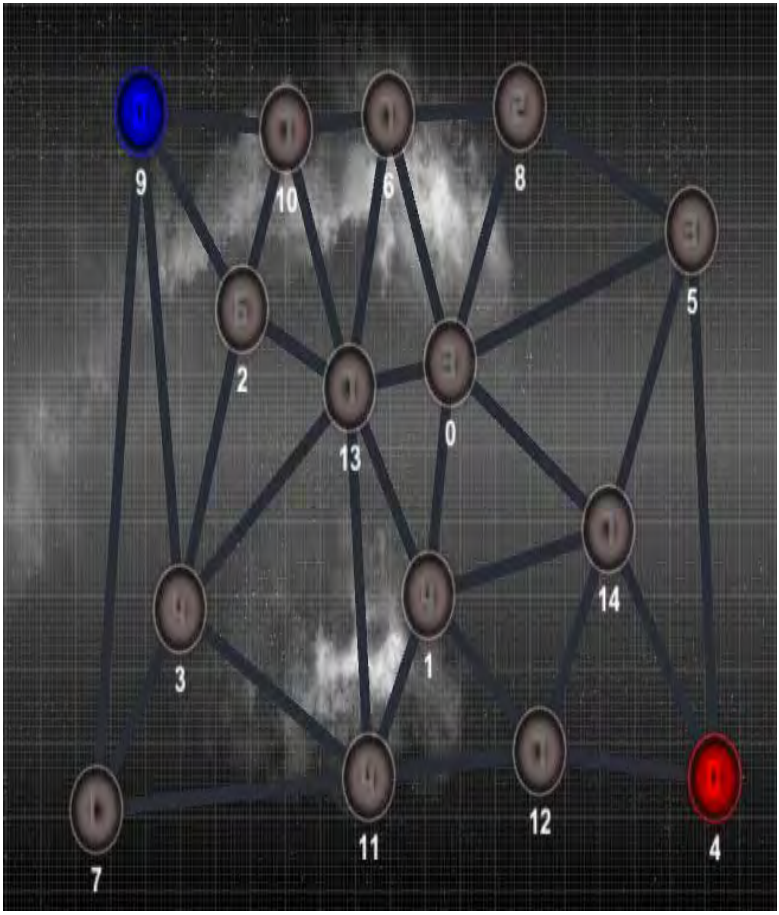


Gambar 5.15 Hasil Pembuatan World Level 3 (Bagian 2)

Gambar 5.16 dan Gambar 5.17 merupakan hasil pembuatan *world* secara visual di *level* 4. Karena hasil dari *world* di *level* 4 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 4 berhasil.

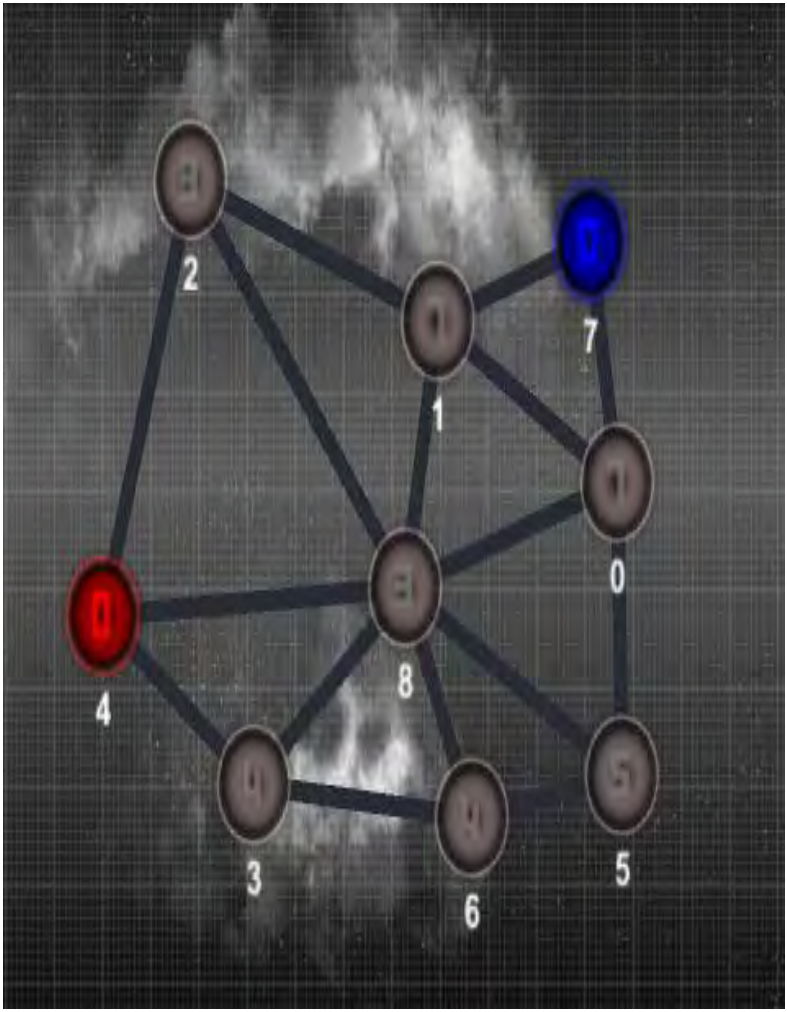


Gambar 5.16 Hasil Pembuatan World Level 4 (Bagian 1)

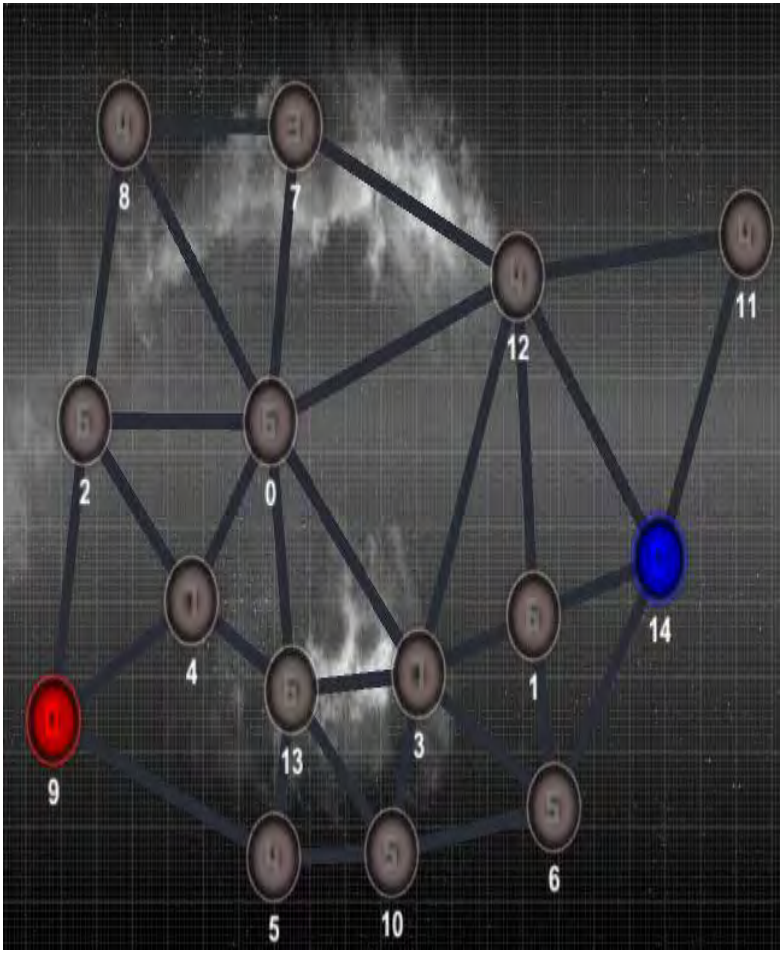


Gambar 5.17 Hasil Pembuatan World Level 4 (Bagian 2)

Gambar 5.18 dan Gambar 5.19 adalah hasil pembuatan *world* di *level* 5. Karena hasil dari *world* di *level* 5 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 5 berhasil.

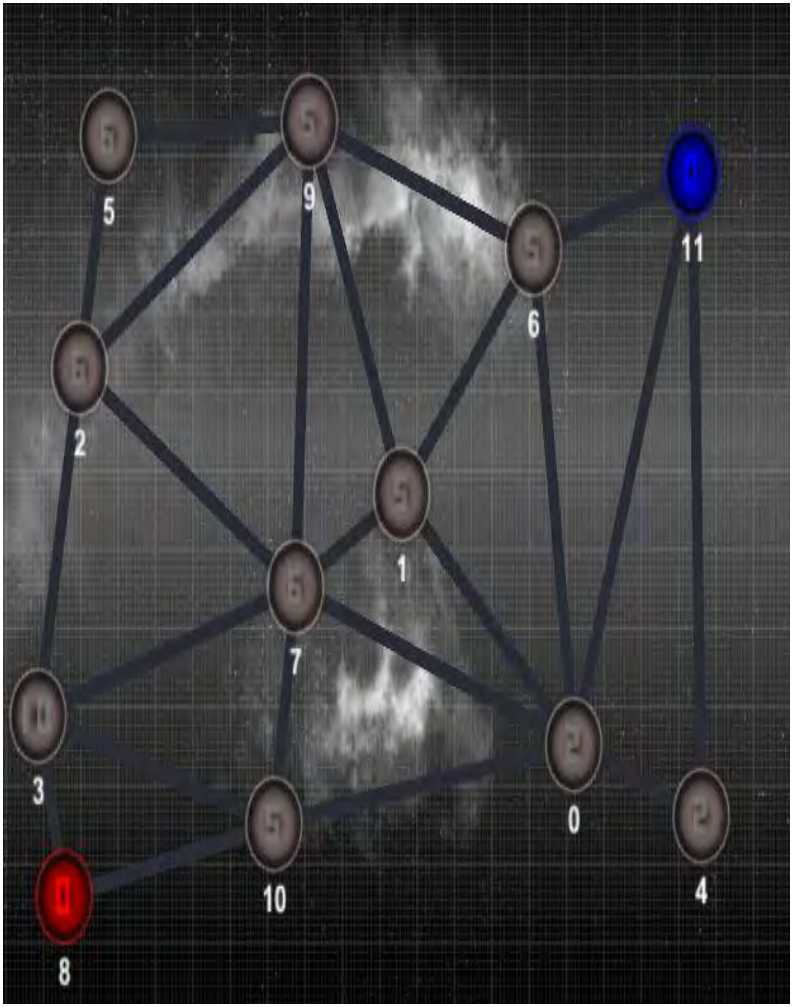


Gambar 5.18 Hasil Pembuatan World Level 5 (Bagian 1)

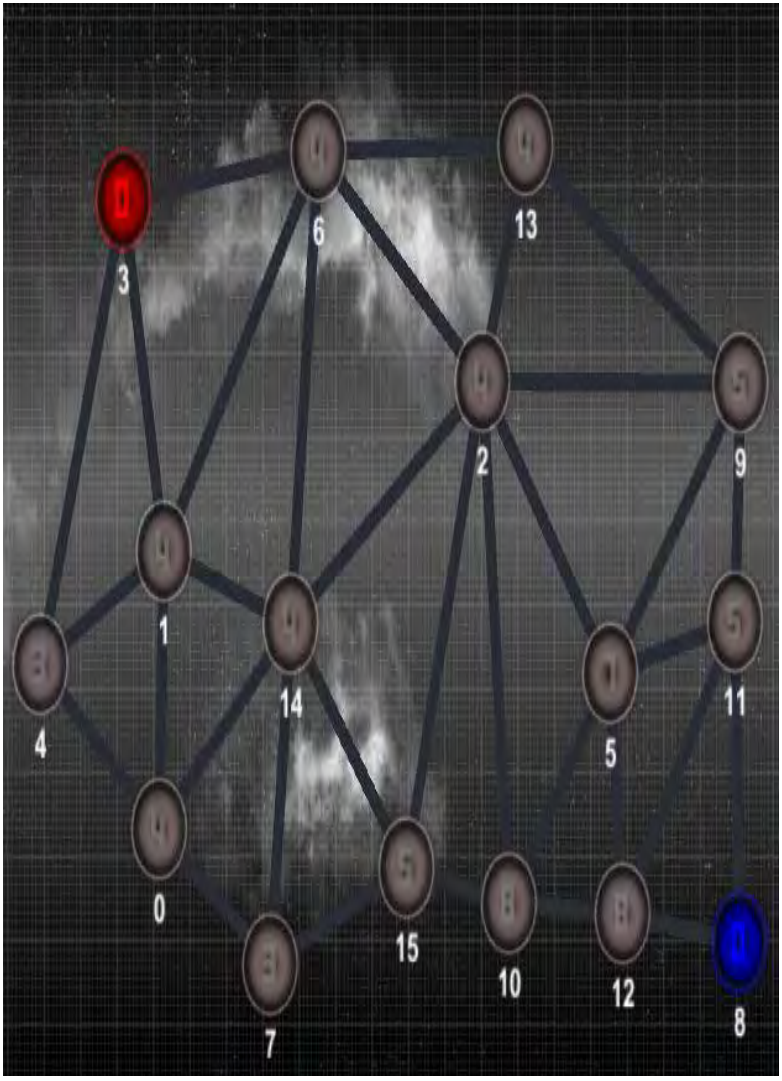


Gambar 5.19 Hasil Pembuatan World Level 5 (Bagian 2)

Gambar 5.20 dan Gambar 5.21 adalah hasil pembuatan *world* di *level* 6. Karena hasil dari *world* di *level* 6 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 6 berhasil.

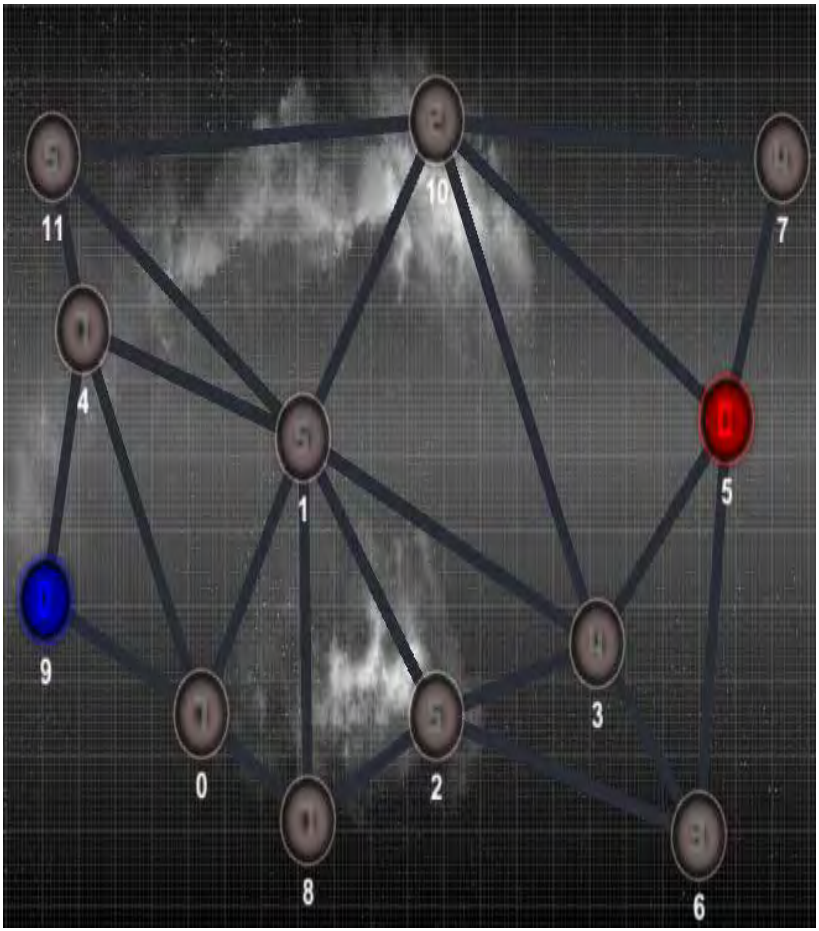


Gambar 5.20 Hasil Pembuatan World Level 6 (Bagian 1)

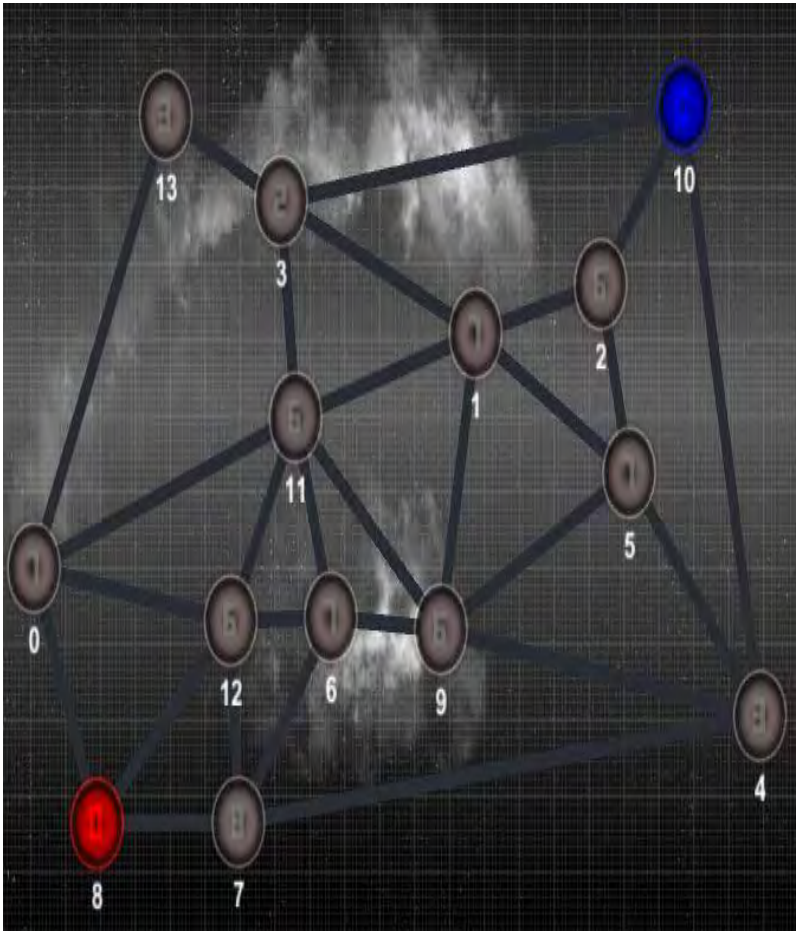


Gambar 5.21 Hasil Pembuatan World Level 6 (Bagian 2)

Gambar 5.22 dan Gambar 5.23 adalah hasil pembuatan *world* di *level* 7. Karena hasil dari *world* di *level* 7 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 7 berhasil.

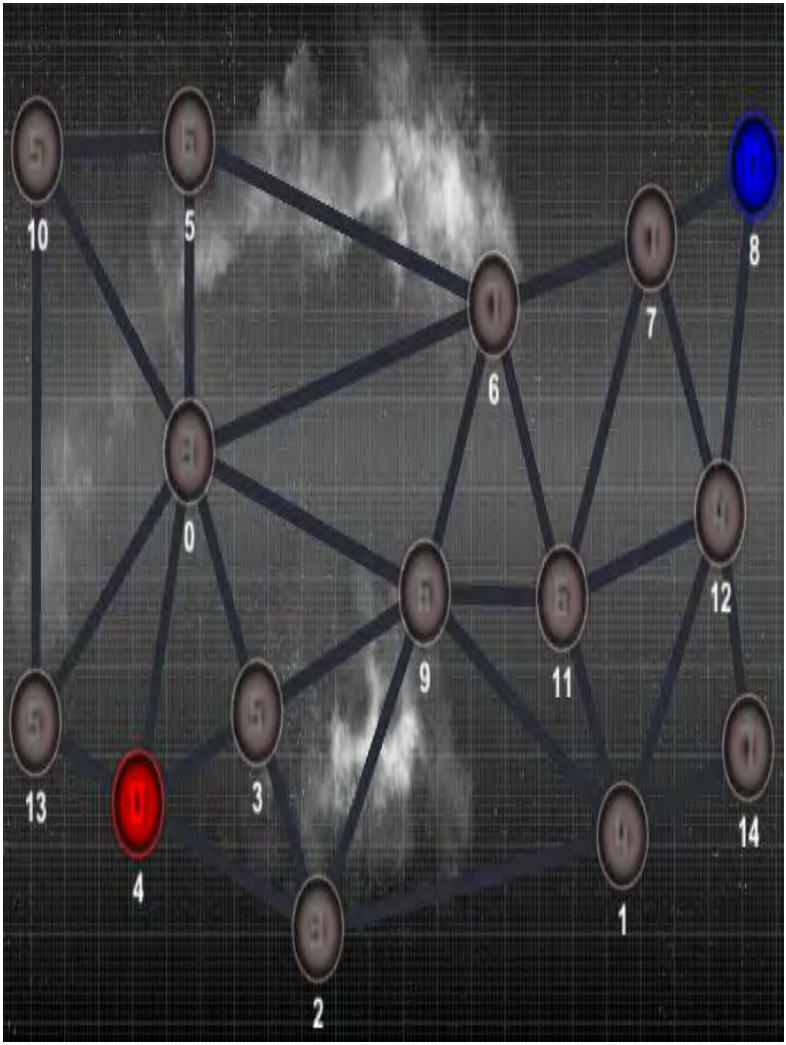


Gambar 5.22 Hasil Pembuatan World Level 7 (Bagian 1)

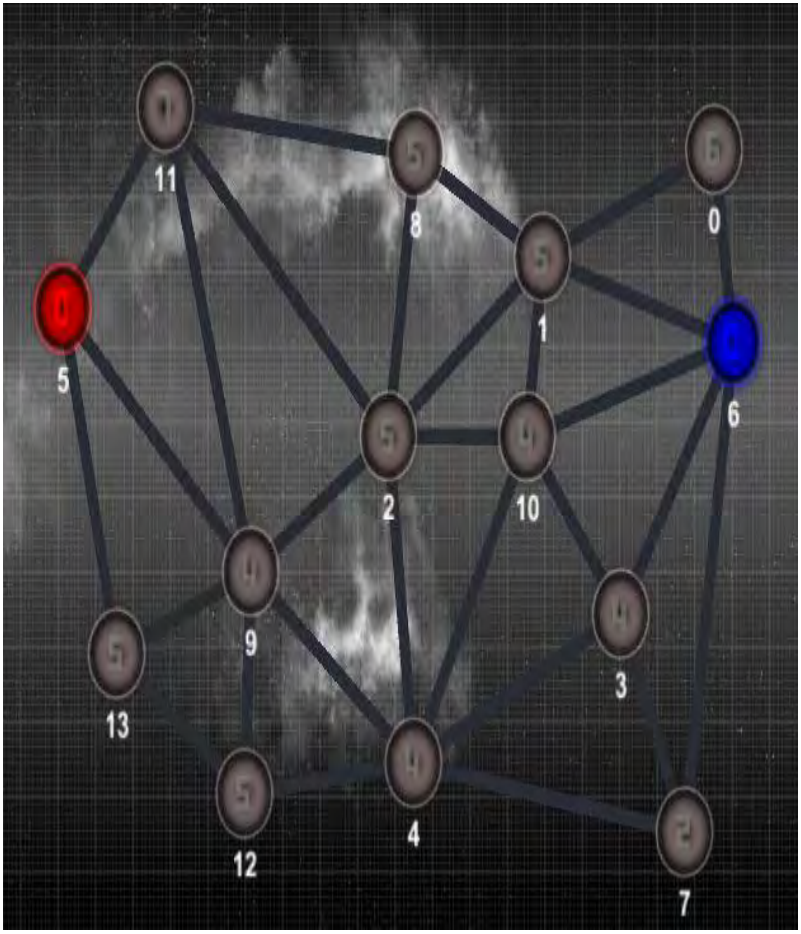


Gambar 5.23 Hasil Pembuatan World Level 7 (Bagian 2)

Gambar 5.24 dan Gambar 5.25 adalah hasil pembuatan *world* di *level* 8. Karena hasil dari *world* di *level* 8 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 8 berhasil.

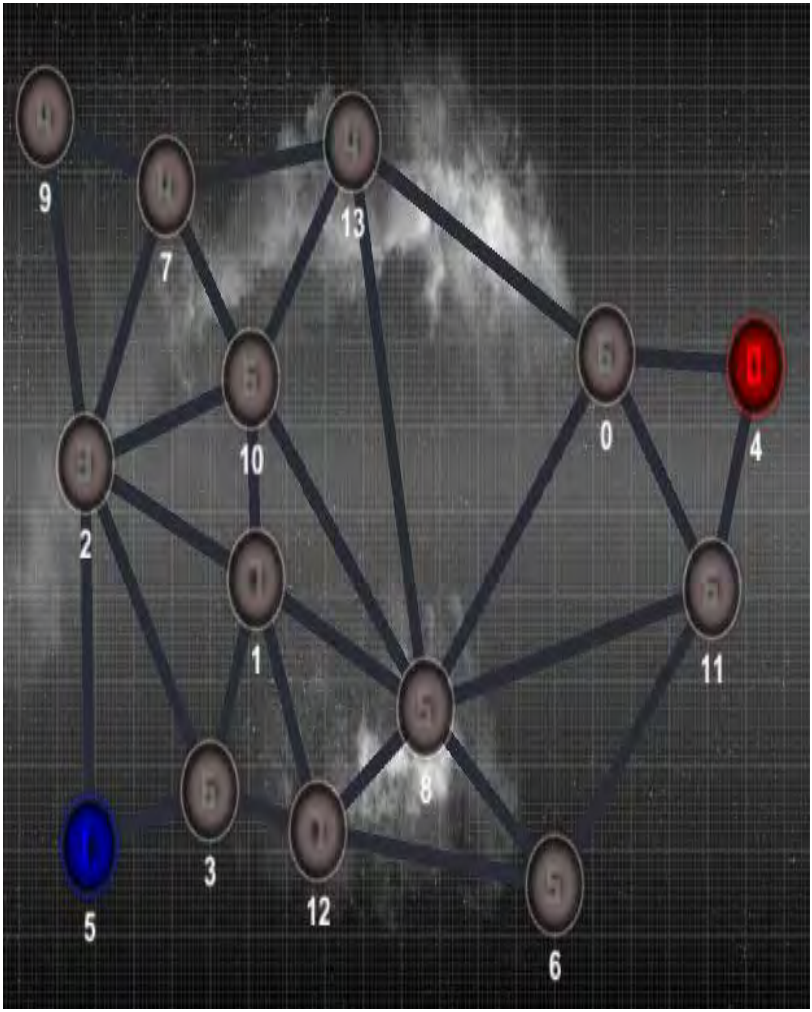


Gambar 5.24 Hasil Pembuatan World Level 8 (Bagian 1)

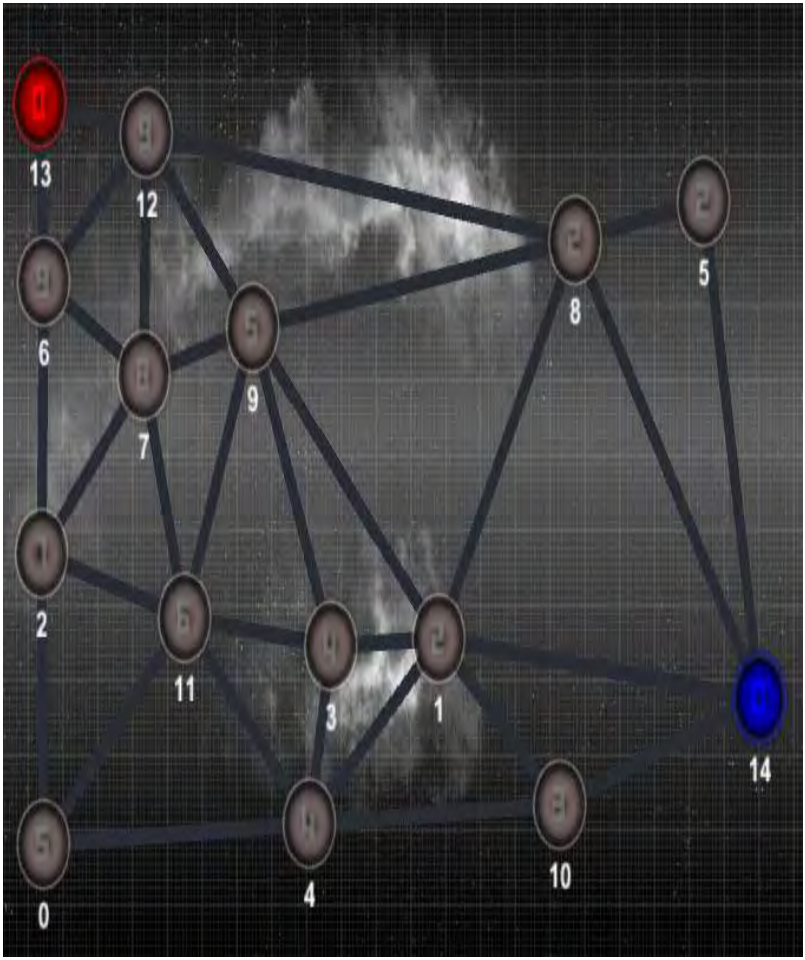


Gambar 5.25 Hasil Pembuatan World Level 8 (Bagian 2)

Gambar 5.26 dan Gambar 5.27 adalah hasil pembuatan *world* di *level* 9. Karena hasil dari *world* di *level* 9 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 9 berhasil.



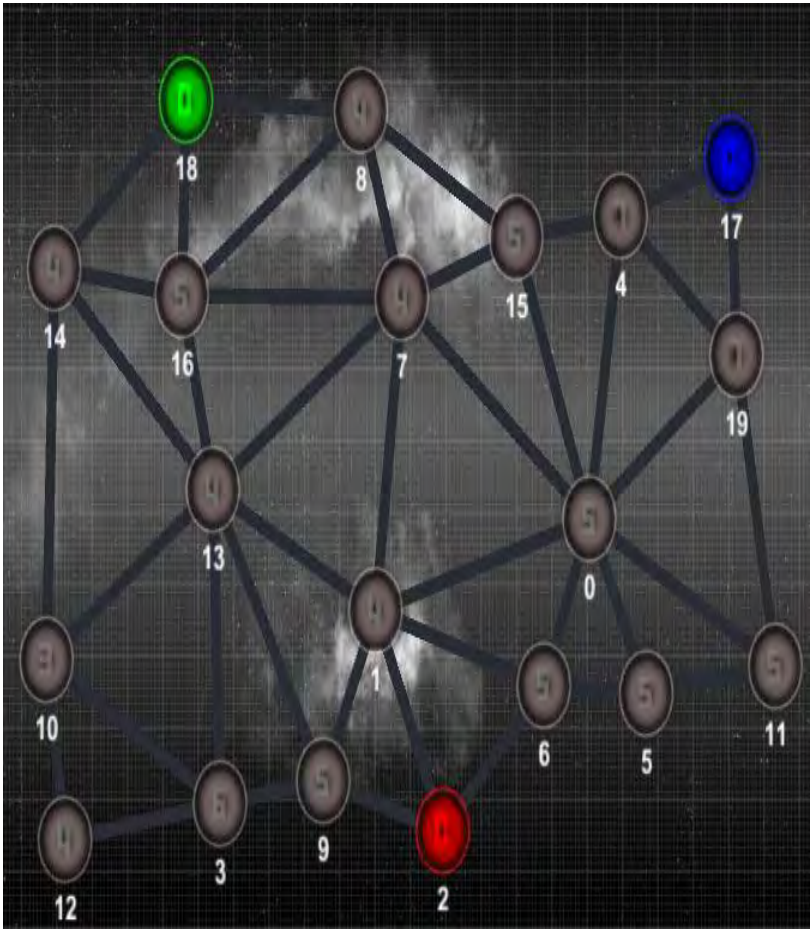
Gambar 5.26 Hasil Pembuatan World Level 9 (Bagian 1)



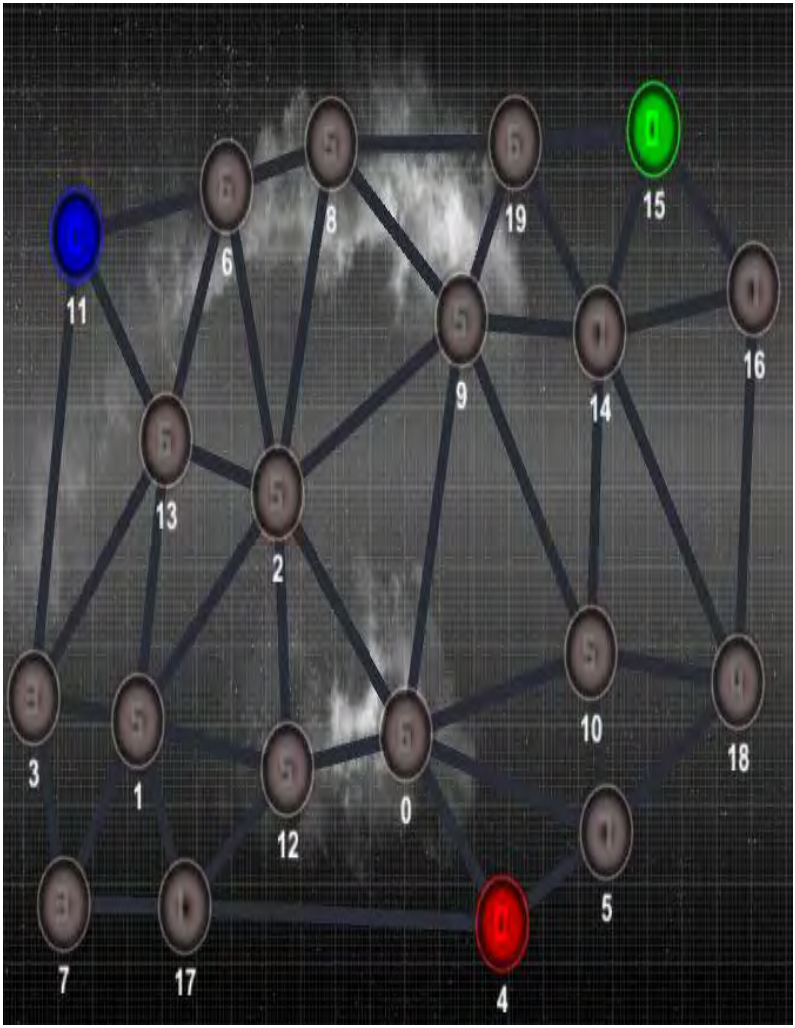
Gambar 5.27 Hasil Pembuatan World Level 9 (Bagian 2)

Untuk *level* 10 keatas jumlah AI adalah 2. Sehingga total planet awal dalam suatu *world* ada 3. Planet yang dimiliki oleh pemain adalah planet yang berwarna merah. Sedangkan planet yang berwana hijau dan biru adalah planet yang dimiliki oleh AI.

Gambar 5.28 dan Gambar 5.29 adalah hasil pembuatan *world* di *level* 10. Karena hasil dari *world* di *level* 10 berbeda secara visual sehingga pembuatan *world* yang dinamis di *level* 10 berhasil.

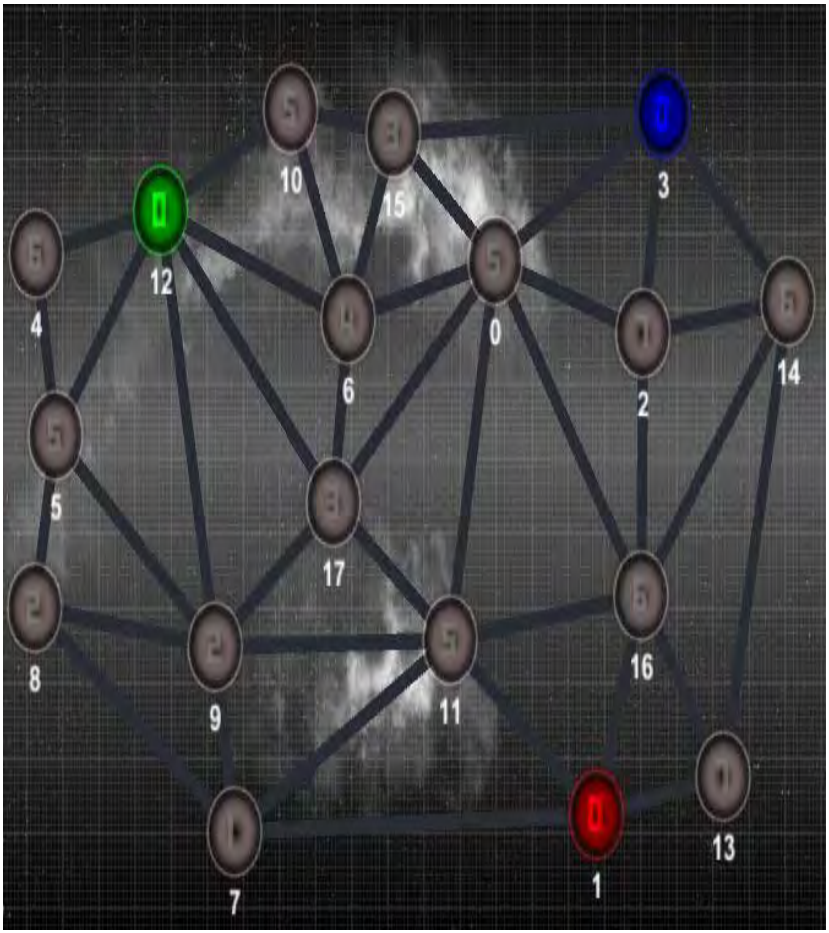


Gambar 5.28 Hasil Pembuatan World Level 10 (Bagian 1)

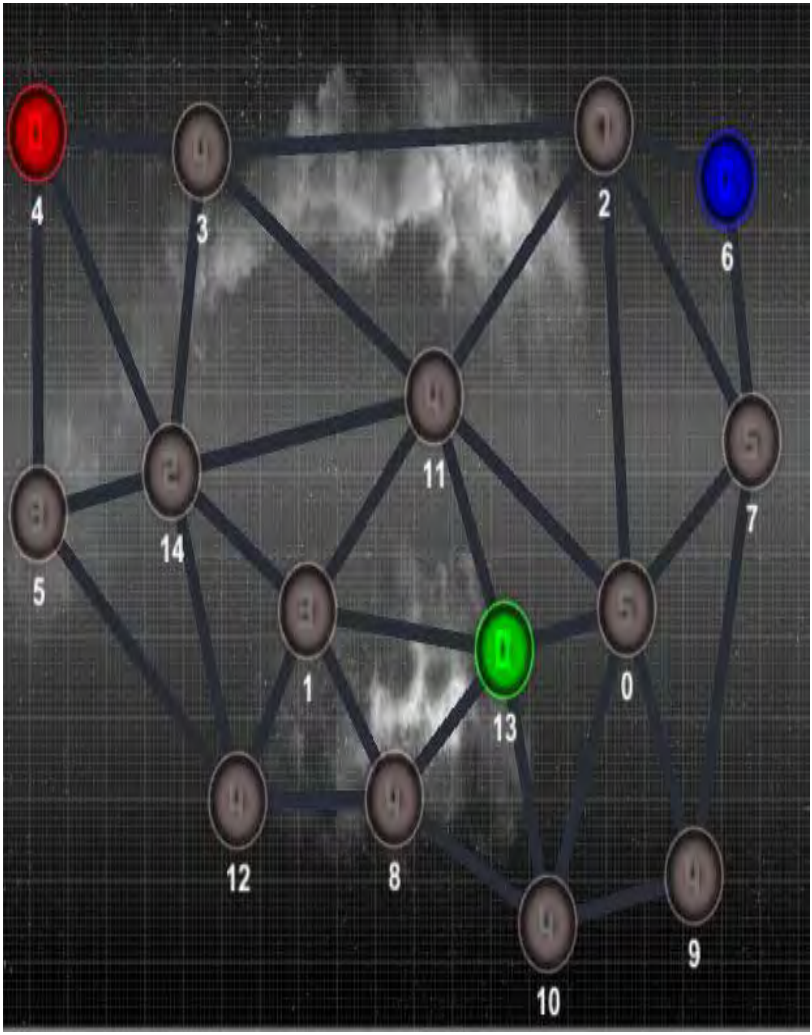


Gambar 5.29 Hasil Pembuatan World Level 10 (Bagian 2)

Gambar 5.30 dan Gambar 5.31 adalah hasil pembuatan *world* di *level* 11. Karena hasil dari *world* di *level* 11 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 11 berhasil.

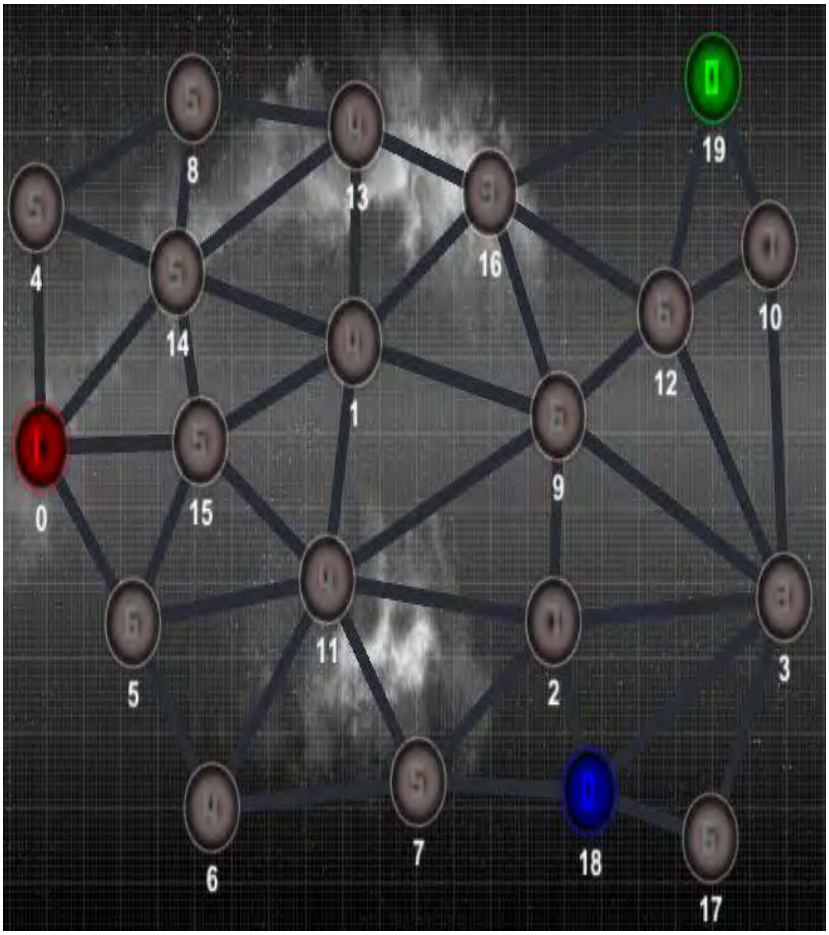


Gambar 5.30 Hasil Pembuatan World Level 11 (Bagian 1)

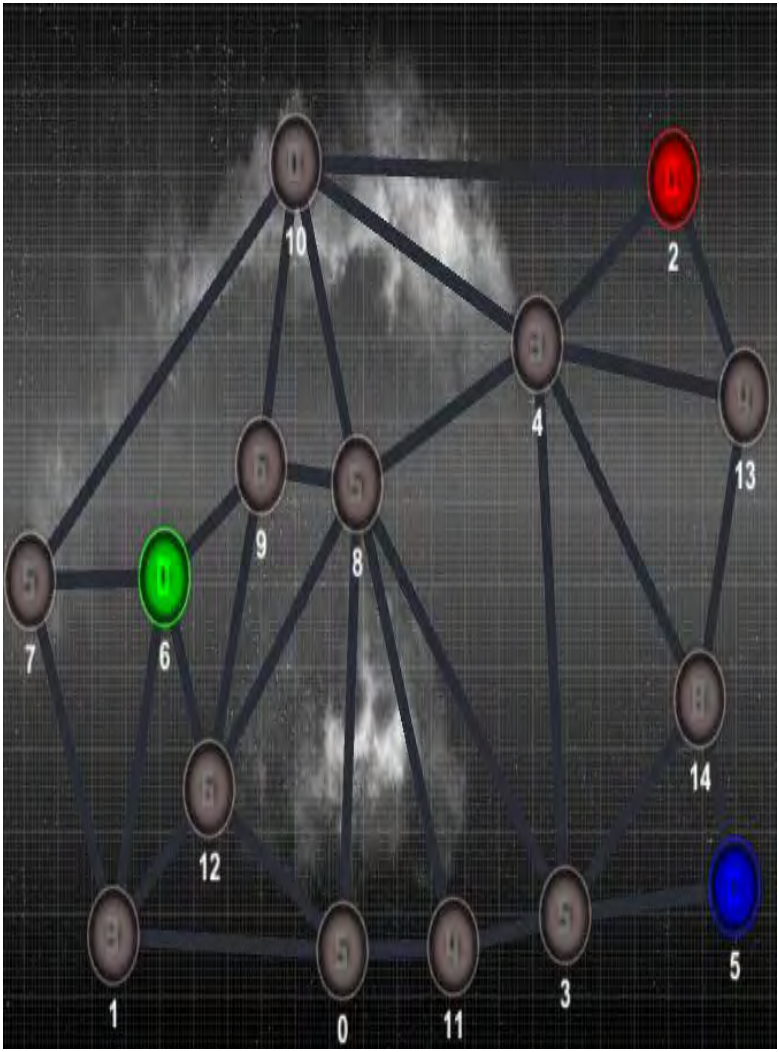


Gambar 5.31 Hasil Pembuatan World Level 11 (Bagian 2)

Gambar 5.32 dan Gambar 5.33 adalah hasil pembuatan *world* di *level* 12. Karena hasil dari *world* di *level* 12 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 12 berhasil.

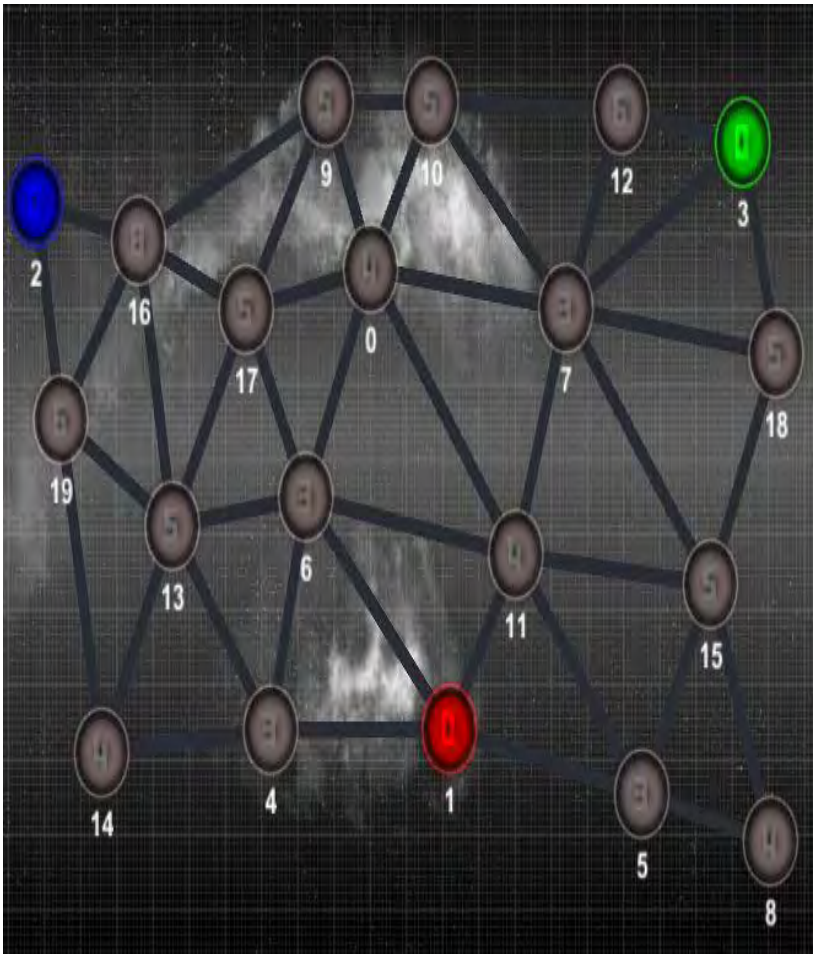


Gambar 5.32 Hasil Pembuatan World Level 12 (Bagian 1)

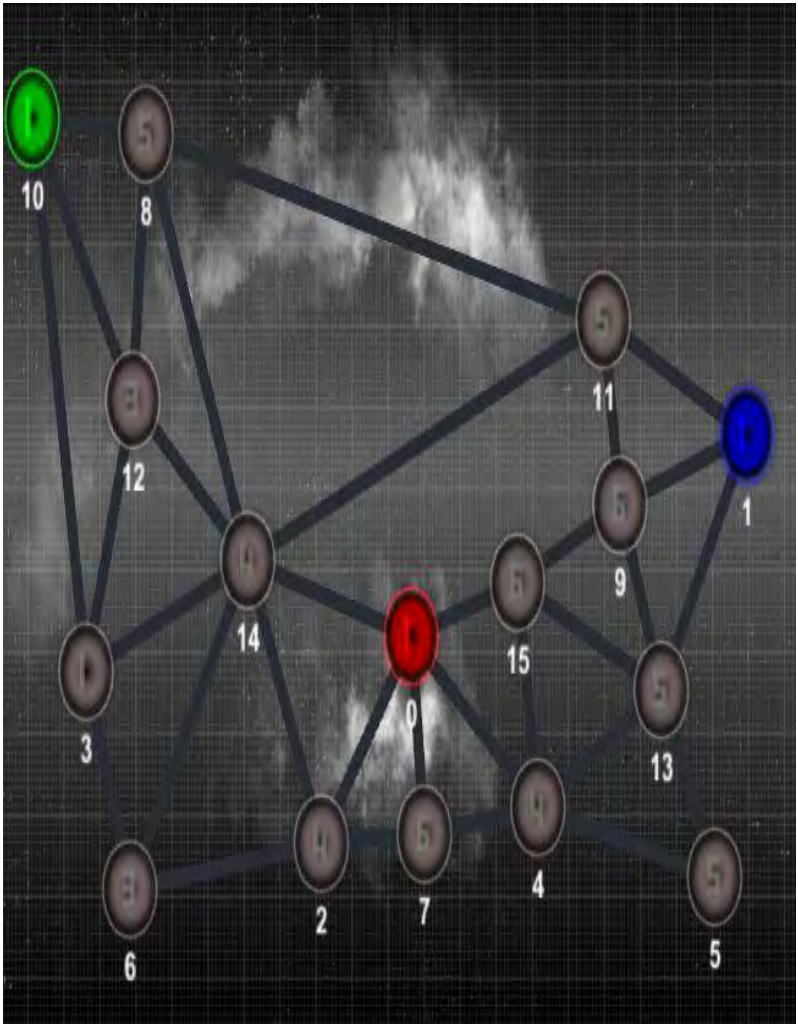


Gambar 5.33 Hasil Pembuatan World Level 12 (Bagian 2)

Gambar 5.34 dan Gambar 5.35 adalah hasil pembuatan *world* di *level* 13. Karena hasil dari *world* di *level* 13 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 13 berhasil.

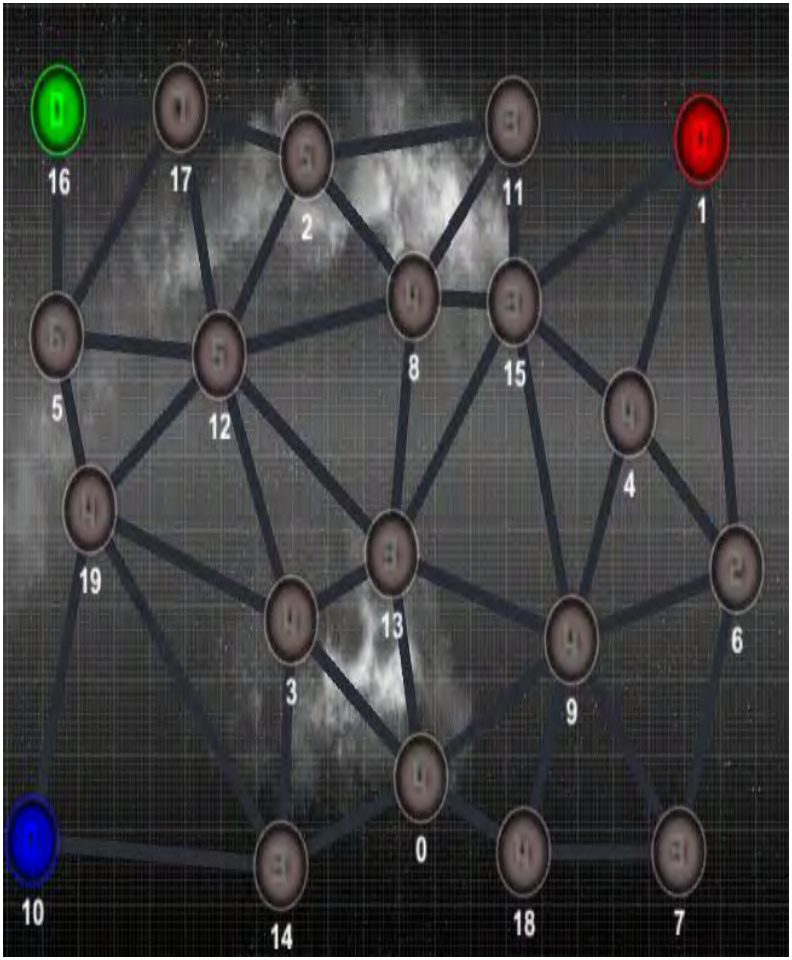


Gambar 5.34 Hasil Pembuatan World Level 13 (Bagian 1)

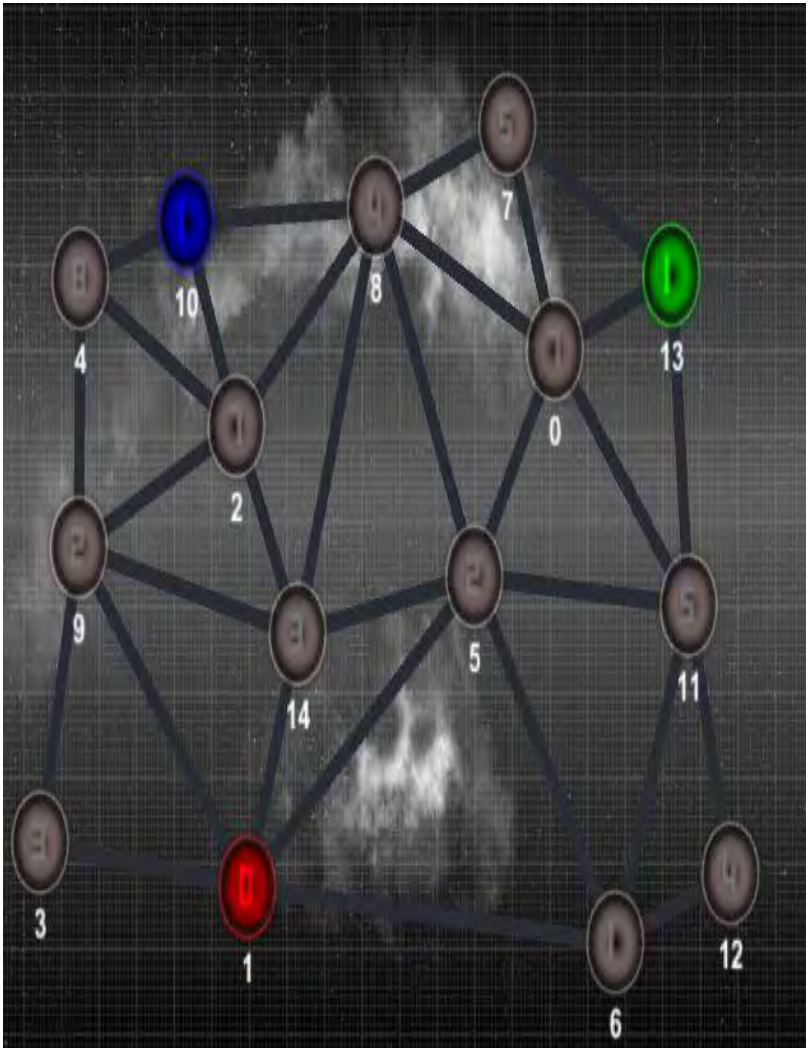


Gambar 5.35 Hasil Pembuatan World Level 13 (Bagian 2)

Gambar 5.36 dan Gambar 5.37 adalah hasil pembuatan *world* di *level* 14. Karena hasil dari *world* di *level* 14 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 14 berhasil.

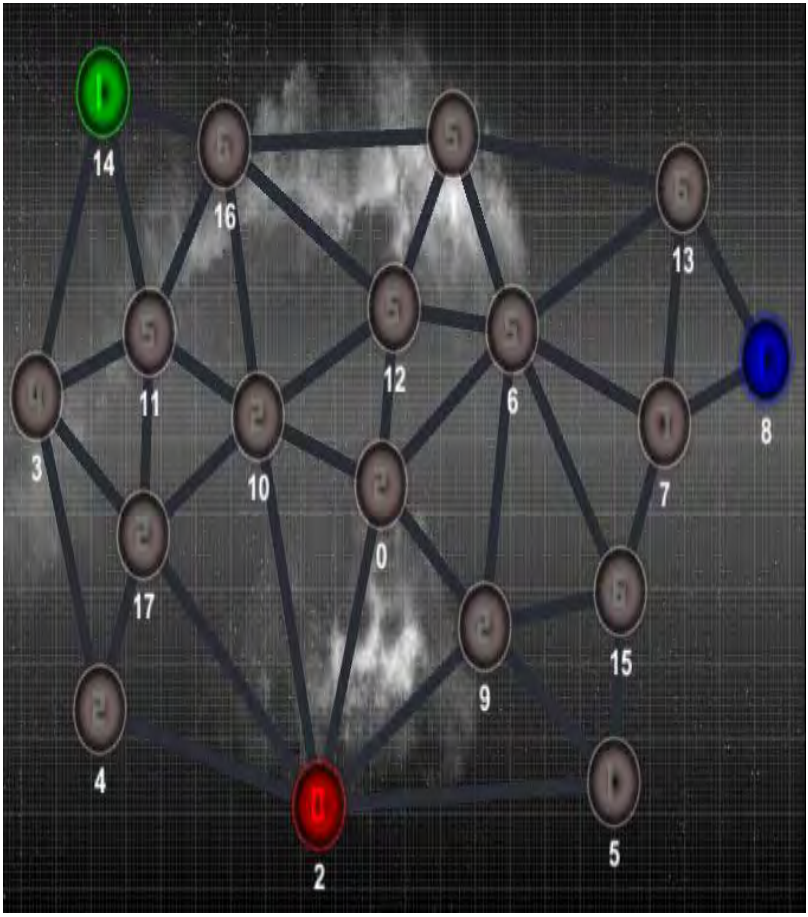


Gambar 5.36 Hasil Pembuatan World Level 14 (Bagian 1)

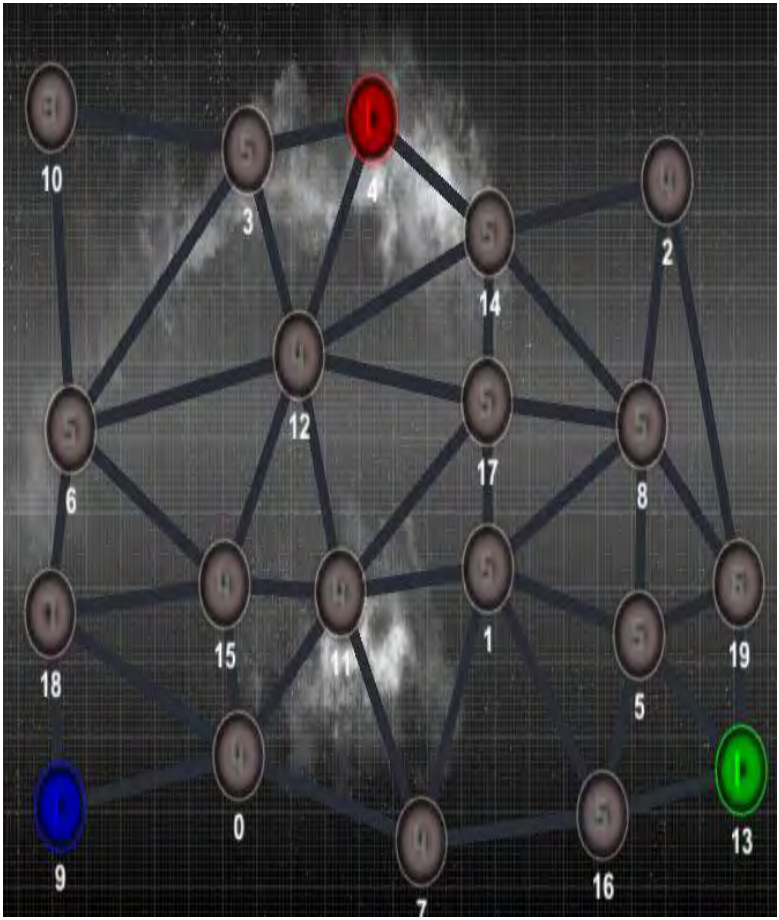


Gambar 5.37 Hasil Pembuatan World Level 14 (Bagian 2)

Gambar 5.38 dan Gambar 5.39 adalah hasil pembuatan *world* di *level* 15. Karena hasil dari *world* di *level* 15 berbeda secara visual dan jumlah planetnya juga berbeda sehingga pembuatan *world* yang dinamis di *level* 15 berhasil.



Gambar 5.38 Hasil Pembuatan World Level 15 (Bagian 1)



Gambar 5.39 Hasil Pembuatan World Level 15 (Bagian 2)

Tabel 5.7 menunjukkan nilai atribut – atribut dan bobot yang menentukan keadaan *world* di setiap *level*. Dapat dilihat di Tabel 5.7 walaupun nilai dari atribut – atribut setiap *level* seperti planet, AI, dan AI regen berbeda tapi bobot di *level* yang sama bernilai kurang dari 0,03. Sehingga pembuatan *world* yang dinamis dapat dikatakan berhasil.

Tabel 5.7 Hasil Pengujian World Dinamis

Level	Planet	AI	AI Regen	Bobot
1	5	1	3	0
1	5	1	3	0
2	8	1	2,9965	0,0509
2	6	1	2,8271	0,0599
3	11	1	2,9273	0,1182
3	11	1	2,9531	0,1117
4	13	1	2,8177	0,1789
4	15	1	2,9515	0,1788
5	9	1	2,3443	0,2306
5	15	1	2,7516	0,2288
6	12	1	2,3047	0,2905
6	16	1	2,4837	0,3124
7	12	1	1,9743	0,3731
7	14	1	2,0782	0,3804
8	15	1	1,9514	0,4288
8	14	1	1,8523	0,4369
9	14	1	1,6084	0,4979
9	15	1	1,6720	0,4987
10	20	2	2,7865	0,5534
10	20	2	2,7969	0,5508
11	18	2	2,3287	0,6344
11	15	2	2,0876	0,6448
12	20	2	2,2095	0,6976
12	15	2	1,8402	0,7066
13	20	2	1,9992	0,7502
13	16	2	1,6848	0,7621
14	20	2	1,7534	0,8116
14	15	2	1,3419	0,8312
15	18	2	1,2517	0,9037
15	20	2	1,4625	0,8844

5.3.5 Skenario Pengujian World yang Seimbang

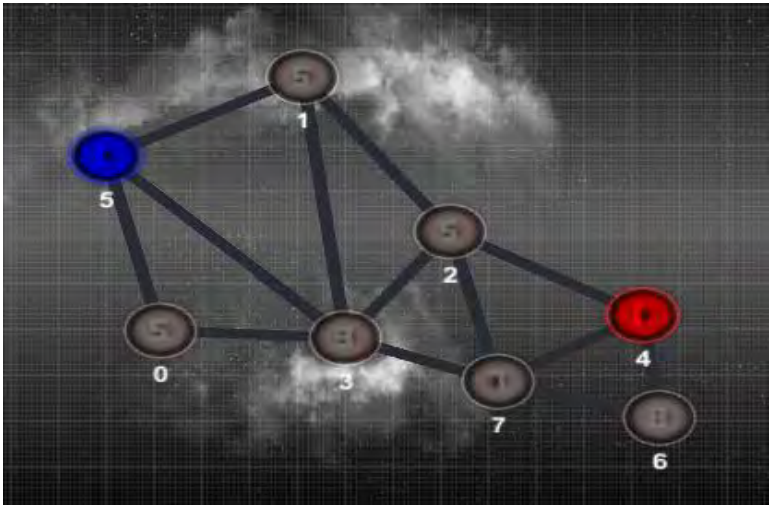
Skenario pengujian *world* yang seimbang digunakan untuk memberikan tahap - tahap dalam pengujian sistem dalam membuktikan *world* yang seimbang. Skenario ini tertera pada Tabel 5.8.

Tabel 5.8 Skenario Pengujian World yang Dinamis

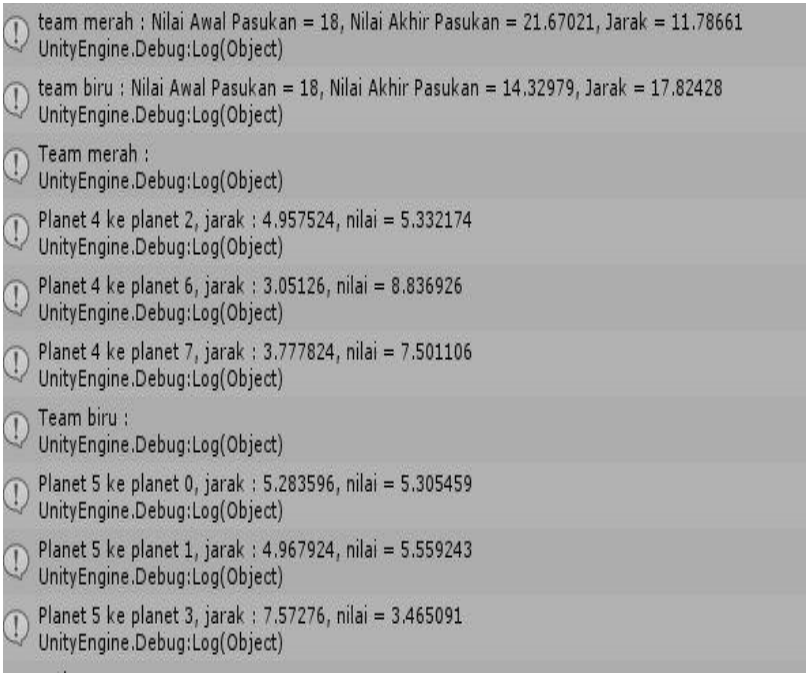
Deskripsi	Bertujuan untuk mengetahui jika *world* yang dibuat sudah dinamis
Langkah pengujian	1. Jalankan *game* sampai layar **Level Selection**. 2. Pilih level yang diinginkan

5.3.6 Hasil Pengujian World yang Seimbang

Gambar 5.40 adalah salah satu hasil dari pembuatan *world* secara visual. Tulisan angka yang berada di bawah planet adalah nomor dari planet tersebut. Planet merah adalah planet milik pemain dan planet biru adalah planet milik AI.



Gambar 5.40 Hasil World yang Seimbang



Gambar 5.41 Keterangan dari Hasil Pembuatan World

Gambar 5.41 menunjukkan keterangan dari *world* yang telah dibuat seperti di Gambar 5.40. Di Gambar 5.41 nilai awal pasukan untuk tim merah dan biru bernilai sama yaitu 18. Tetapi karena total jarak ke semua tetangga pertama dari tim merah dan biru berbeda maka nilai akhir pasukan memiliki hasil yang berbeda. Karena jarak dari tim merah lebih kecil dari tim biru, nilai akhir pasukan tim merah menjadi lebih besar dari tim biru.

Kemudian nilai akhir pasukan dibagikan ke setiap planet netral disekitar planet awal untuk tim merah dan biru. Semakin besar jarak dari planet awal suatu tim ke planet netral maka nilai pasukan di planet tersebut akan semakin kecil. Seperti yang ditunjukkan pada Gambar 5.41, karena jarak planet 4 ke planet 2 lebih besar dari pada jarak planet 4 ke planet 6 dan jarak planet 4

ke planet 7, maka jarak planet 4 ke planet 2 memiliki nilai pasukan di planet 2 jauh lebih kecil. Sedangkan karena jarak planet 4 ke planet 7 sedikit lebih besar dari jarak planet 4 ke planet 6, maka nilai pasukan planet 7 sedikit lebih kecil daripada nilai pasukan di planet 6. Sehingga hasil dari pembuatan *world* dikatakan sudah seimbang karena dapat menentukan nilai pasukan planet netral berdasarkan dari jarak planet netral tersebut dari planet awal salah satu tim pemain atau musuh.

5.4 Pengujian Pengguna

Pengujian pada perangkat lunak yang dibangun tidak hanya dilakukan pada fungsionalitas yang dimiliki, tetapi juga pada pengguna untuk mencoba secara langsung. Pengujian ini berfungsi sebagai pengujian subjektif yang bertujuan untuk mengetahui tingkat keberhasilan aplikasi yang dibangun dari sisi pengguna. Hal ini dapat dicapai dengan meminta penilaian dan tanggapan dari pengguna terhadap sejumlah aspek perangkat lunak yang ada.

5.4.1 Skenario Uji Coba Pengguna

Dalam melakukan pengujian perangkat lunak, penguji diminta mencoba menggunakan perangkat lunak untuk mencoba semua fungsionalitas dan fitur yang ada. Serta untuk mencoba materi-materi dan isi dari soal-soal edukasi yang ada pada permainan.

Pengujian aplikasi oleh pengguna dilakukan dengan sebelumnya memberikan informasi seputar aplikasi, kegunaan, dan fitur-fitur yang dimiliki. Setelah informasi tersampaikan, pengguna kemudian diarahkan untuk langsung mencoba aplikasi dengan spesifikasi lingkungan yang sama dengan yang telah diuraikan pada uji coba fungsionalitas.

Jumlah pengguna yang terlibat dalam pengujian perangkat lunak sebanyak tiga orang. Dalam melakukan pengujian, pengguna melakukan percobaan lebih dari satu kali penggunaan untuk masing-masing pengguna.

Dalam memberikan penilaian dan tanggapan, penguji diberikan formulir pengujian perangkat lunak. Formulir pengujian perangkat lunak ini memiliki beberapa aspek penilaian dan pada bagian akhir terdapat saran untuk perbaikan fitur.

5.4.2 Daftar Penguji Perangkat Lunak

Pada subbab ini ditunjukkan daftar pengguna yang bertindak sebagai penguji coba aplikasi yang dibangun. Daftar nama penguji aplikasi ini dapat dilihat pada Tabel 5.9.

Tabel 5.9 Daftar Penguji Aplikasi

Nomor	Nama	Pekerjaan
1	Muhamad Rizal Prihandoko	Mahasiswa Teknik Informatika ITS
2	Irooyan Alfi Aziz	Mahasiswa Teknik Informatika ITS
3	Yuan Akbarsyah Pandu	Mahasiswa Teknik Informatika ITS
4	Mulia Lovendo Aprin Cigy	Mahasiswa Teknik Informatika ITS
5	Stieven Wirakarsa	Mahasiswa Teknik Informatika ITS
6	Alfian Maulana Azhari	Mahasiswa Teknik Informatika ITS
7	Nyoman Bagus P	Mahasiswa Teknik Informatika ITS
8	Firdaus Nutrihadi	Mahasiswa Teknik Informatika ITS
9	Sandhi Ading W	Mahasiswa Teknik Informatika ITS
10	Muhammad Rohman	Mahasiswa Teknik Informatika ITS

5.4.3 Hasil Uji Coba Pengguna

Uji coba yang dilakukan terhadap beberapa pengguna memiliki beberapa aspek yang dipisahkan berdasarkan antarmuka dan fungsionalitas yang dimiliki. Sistem penilaian didasarkan pada skala penghitungan satu sampai empat di mana skala satu menunjukkan nilai terendah dan skala empat menunjukkan skala tertinggi. Penilaian akhir kemudian dilakukan dengan menghitung berapa banyak penguji yang memilih suata skala tertentu dan kemudian dicari nilai rata-ratanya. Hasil uji coba dipaparkan secara lengkap dengan disertai tabel yang dapat dilihat pada subbab.

5.4.3.1 Hasil Penilaian Antarmuka

Penilaian antarmuka difokuskan pada penilaian pengguna terhadap kemudahan penggunaan antarmuka dan sifat-sifat lain yang perlu dimiliki. Sepuluh orang yang namanya terdaftar di Tabel 5.9 akan memberikan penilaian terhadap antarmuka aplikasi di Tabel 5.10 dengan nilai mulai dari 1 sampai 4 dimana nilai 1 adalah buruk, 2 adalah cukup baik, 3 adalah baik, dan 4 adalah sangat baik dan kemudian dicari nilai rata – rata terhadap antarmuka *game* yang telah dibuat.

Tabel 5.10 Penilaian Antarmuka

No.	Antarmuka	Penilaian				Rata-Rata
		1	2	3	4	
1	Kemudahan Penggunaan	0	1	9	0	2,9
2	Kelengkapan Menu	0	2	8	0	2,8
3	Keindahan Tampilan	0	8	2	0	2,2
4	Ketertarikan Bermain	0	0	7	3	3,3
Nilai Akhir						**2,8**

5.4.3.2 Hasil Penilaian Performa Sistem

Penilaian performa sistem difokuskan pada penilaian pengguna terhadap kemampuan aplikasi dalam menghasilkan performa dari interaksi pengguna. Penilaian ini juga ditujukan untuk mendapatkan tingkat kecepatan dan kelancaran sistem atas interaksi yang dibuat oleh pengguna. Sepuluh orang yang namanya terdaftar di Tabel 5.9 akan memberikan penilaian terhadap performa sistem di **Tabel 5.11** dengan nilai mulai dari 1 sampai 4 dimana nilai 1 adalah buruk, 2 adalah cukup baik, 3 adalah baik, dan 4 adalah sangat baik dan kemudian dicari nilai rata – rata terhadap performa sistem *game* yang telah dibuat.

Tabel 5.11 Penilaian Performa Sistem

No.	Performa Sistem	Penilaian				Rata-Rata
		1	2	3	4	
1	Performa atau kinerja pada permainan	0	0	5	5	3,5
2	Kesulitan dari kecerdasan buatan	0	0	3	7	3,7
3	Kelancaran animasi	0	0	10	0	3
4	Kesesuaian animasi dengan sistem permainan	0	5	5	0	2,5
5	Kesesuaian interaksi dengan hasil yang seharusnya	0	0	2	8	3,8
Nilai Akhir						**3,3**

5.5 Hasil Pengujian Pengguna

Evaluasi pengujian pengguna dilakukan dengan menampilkan data rekapitulasi perangkat lunak yang telah dipaparkan. Dalam hal ini, rekapitulasi disusun dalam bentuk tabel yang dapat dilihat pada Tabel 5.12. Dari data diketahui bahwa aplikasi telah memenuhi unsur yang seharusnya dimana

nilai prosentase untuk antarmuka adalah 70% dan nilai prosentase untuk performa sistem memiliki nilai 84%.

Tabel 5.12 Rekapitulasi Hasil Uji Coba Pengguna

No.	Nama Pengujian		Rata-Rata	Nilai	Nilai (%)
1	Penilaian Antarmuka	Kemudahan Penggunaan	2,9	**2,8**	**70**
		Kelengkapan Menu	2,8		
		Keindahan Tampilan	2,2		
		Ketertarikan Bermain	2,7		
2	Performa Sistem permainan	Performa atau kinerja pada permainan	3,5	**3,3**	**82,5**
		Kesulitan dari kecerdasan buatan	3,7		
		Kelancaran animasi	3		
		Kesesuaian animasi dengan sistem permainan	2,5		
		Kesesuaian interaksi dengan hasil yang seharusnya	3,8		

(Halaman Ini Sengaja Dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang dapat diambil dari tujuan pembuatan perangkat lunak dan hasil uji coba yang telah dilakukan sebagai jawaban dari rumusan masalah yang dikemukakan. Selain kesimpulan, terdapat pula saran yang ditujukan untuk pengembangan perangkat lunak lebih lanjut.

6.1. Kesimpulan

Dalam proses pengerjaan tugas akhir mulai dari tahap analisis, desain, implementasi, hingga pengujian didapatkan kesimpulan sebagai berikut:

1. Berdasarkan uji performa, aplikasi berhasil membangkitkan *world* dengan waktu minimal 0.08 detik dan maksimal sampai 0,14 detik. Sehingga dapat diasumsikan bahwa pembangkitan *world* di setiap *level* dengan maksimal level sebanyak 15 memiliki rata-rata 0,1 detik.
2. Aplikasi berhasil membuat *world* yang dinamis karena setiap memainkan *game* di *level* yang sama jumlah planet, posisi planet, kecepatan regenerasi planet, dan jumlah musuh berrubah.
3. Aplikasi berhasil membangun permainan menggunakan *world generator* dan berhasil membuat *world* yang seimbang bagi pemain dan musuh karena dapat mengisi nilai pasukan dari planet netral berdasarkan jarak planet netral tersebut dari planet awal.

6.2. Saran

Berikut merupakan beberapa saran untuk pengembangan sistem di masa yang akan datang, berdasarkan pada hasil perancangan, implementasi dan uji coba yang telah dilakukan.

1. Penambahan *level* dalam permainan menggunakan lebih banyak variasi atribut yang dibangkitkan.

2. Optimisasi perhitungan terhadap nilai pasukan di planet netral dan penentuan posisi awal pemain dan musuh jika AI ada 2 agar lebih seimbang bagi semua pemain.

DAFTAR PUSTAKA

[1] "Unity Manual," Unity Documentation, [Online]. Available: http://docs.unity3d/Manual/index.html. [Accessed 17 12 2014].

[2] "GooglePlay," Little Star for Little Wars, [Online]. Available: https://play.google.com/store/apps/details?id=com.mkarpenko.lsflw2. [Accessed 17 12 2014].

[3] "Yahoo Games Network," Yahoo, 24 5 2013. [Online]. Available: https://gamesnet.yahoo.net/documentation/yahoo/. [Accessed 18 9 2014].

[4] "Black Box Techniques," Qualitance, [Online]. Available: http://www.qualitance.com/blog/black-box-techniques. [Accessed 18 9 2014].

[5] "Secret Game Mechanics Playdeck," TechCrunch, 25 7 2010. [Online]. Available: http://techcrunch.com/2010/08/25/scvngr-game-mechanics/. [Accessed 26 9 2014].

[6] M. Yan, "Djikstra Algorithm," [Online]. Available: math.mit.edu/~rothvoss/18.304.3PM/Presentations/1-Melissa.pdf. [Accessed 14 11 2015].

[7] "C# .NET Programming," [Online]. Available: http://brainmatics.com/c-net-programming/. [Accessed 21 12 2014].

[8] J. Laird and S. Jamin, "Game Theory and Game Balance," 30 October 2006. [Online]. [Accessed 27 October 2015].

[9] "The Anatomy of a Design Document," Gamasutra, 19 October 1999. [Online]. Available: http://www.gamasutra.com/view/feature/3384/the_anatomy_of_a_design_document_.php. [Accessed 26 9 2015].

[10] A. Laskov, "Level Generation System," 2009. [Online].

(Halaman Ini Sengaja dikosongkan)

BIODATA PENULIS

Penulis dilahirkan di Pontianak, 30 Desember 1992, merupakan anak pertama dari 2 bersaudara. Penulis telah menempuh pendidikan formal yaitu SDK Maria Fatima 3 Jember (1999-2005), SMPK Maria Fatima Jember (2005-2008), SMA Negeri 2 Jember (2008-2011), dan mahasiswa S1 Jurusan Teknik Informatika Institut Teknologi Sepuluh Nopember Surabaya rumpun mata kuliah Interaksi, Grafika, dan Seni (2011-2015).

Selama menjadi pelajar, penulis pernah mengikuti lomba Pertolongan Pertama pada Kecelakaan (2005). Saat kuliah penulis pernah mengikuti kegiatan UKM Jujitsu. Penulis dapat dihubungi melalui surel ramatcb11@gmail.com.