TUGAS AKHIR – KI141502

RANCANG BANGUN MODUL *PROCEDURAL CONTENT GENERATION* PADA PERMAINAN FRIEND AND FOE

ANDRIE PRASETYO UTOMO
NRP 5111100081

Dosen Pembimbing
Imam Kuswardayan, S.Kom., M.T.
Ridho Rahman Hariadi, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2015

TUGAS AKHIR – KI141502

RANCANG BANGUN MODUL *PROCEDURAL CONTENT GENERATION* PADA PERMAINAN FRIEND AND FOE

ANDRIE PRASETYO UTOMO
NRP 5111100081

Dosen Pembimbing
Imam Kuswardayan, S.Kom., M.T.
Ridho Rahman Hariadi, S.Kom., M.Sc.

JURUSAN TEKNIK INFORMATIKA
FAKULTAS TEKNOLOGI INFORMASI
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
SURABAYA
2015

RANCANG BANGUN MODUL PROCEDURAL CONTENT GENERATION PADA PERMAINAN FRIEND AND FOE

Nama Mahasiswa : Andrie Prasetyo Utomo
NRP : 5111 100 081
Jurusan : Teknik Informatika FTIf-ITS
Dosen Pembimbing I : Imam Kuswardayan, S.Kom., M.T.
Dosen Pembimbing II : Ridho Rahman Hariadi, S.Kom., M.Sc.

ABSTRAK

Saat ini perkembangan permainan digital (game) telah berkembang cukup pesat. Sudah banyak jenis permainan yang telah beredar di pasaran. Oleh karena itu, perlu adanya faktor dari permainan yang mampu menjaga pemainnya agar mau tetap bermain. Salah satu faktor tersebut adalah dengan memberikan variasi skenario yang berbeda-beda sehingga pemain tidak cepat merasa bosan ketika memulai permainan kembali.

Dalam Tugas Akhir ini dibangun permainan yang mampu membangkitkan peta permainan secara bervariasi ketika permainan dimulai dari awal. Metode yang digunakan untuk membangkitkan peta dinamis tersebut adalah dengan menerapkan Procedural Content Generation. Pembangkitan peta diperoleh dengan membuat sekumpulan peta permainan secara acak yang kemudian akan dievaluasi untuk dijadikan solusi. Di dalam permainan juga akan dijaga keseimbangannya. Artinya permainan dapat menyesuaikan dengan kemampuan pemain. Pengaturan keseimbangan permainan dilakukan dengan pengecekan fungsi evaluasi terhadap komponen permainan yang sudah ada dalam peta permainan.

Pengujian peta permainan dilakukan dengan cara membangkitkan peta permainan melalui beberapa kali percobaan. Dari hasil uji coba, diketahui bahwa Procedural Content Generation mampu membentuk peta permainan secara dinamis, di mana bentuk peta akan selalu berubah setiap

permainan dimulai dari awal. Pengujian keseimbangan dilakukan dengan cara melihat bobot objek yang akan dicari dengan kesesuaian tingkat kemahiran pemain. Dari hasil uji coba didapatkan bahwa Procedural Content Generation mampu digunakan untuk mengatur objek-objek yang muncul dalam permainan untuk disesuaikan dengan kemampuan pemain, yaitu ketika pemain memiliki poin perlengkapan yang cukup besar, maka tingkat kesulitan permainan akan dinaikkan. Sedangkan ketika poin kesehatan pemain dirasa cukup kecil, maka tingkat bantuan akan dinaikkan.

Kata kunci**: **Pembangkit Level Dinamis, Pembangkit Peta Dinamis, Procedural Content Generation, Dynamic Balancing.

DESIGN AND IMPLEMENTATION OF PROCEDURAL CONTENT GENERATION IN "FRIEND AND FOE" GAME

Student Name	: Andrie Prasetyo Utomo
NRP	: 5111 100 081
Major	: Teknik Informatika FTIf-ITS
Advisor I	: Imam Kuswardayan, S.Kom., M.T.
Advisor II	: Ridho Rahman Hariadi, S.Kom., M.Sc.

ABSTRACT

Nowadays, the development of video games has been growing rapidly. There are so many games that have been developed. Therefore, to keep the players in order to saty in the game, the game should have a different scenario and environtment whenever the game started.

This final project built a game that is able to generate a map of the game that is varied when the game starts from the beginning. The method used to generate dynamic maps is the practice of Procedural Content Generation. The generation of the map obtained by making a set of random map which will be evaluated to be the solution. The game also will be kept in balance. This means that the difficulty of the game can customized dynamically depends on the player's ability. Game balancing is done by checking the evaluation function of the difficulty of the game at the time.

Tests done by generating the maps through several experiments. From the test results, it is known that the Procedural Content Generation is able to form a map of the game dynamically, in which the shape of the map will always change every game starts from the beginning. Balance testing is done by looking at the weight of the object to be searched with the suitability of the players skill level. From the test results showed that the Procedural Content Generation can be used to organize the objects that appear in the game to suit the player's ability,

that is, when the player has points equipment is large enough, then the game difficulty level will be raised. Meanwhile, when the player's health are considered small, then the level of aid will be increased.

Keywords: Dynamic Level Generator, Education, Mobile Device, Android OS.

LEMBAR PENGESAHAN

RANCANG BANGUN MODUL PROCEDURAL CONTENT GENERATION PADA PERMAINAN FRIEND AND FOE

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:
ANDRIE PRASETYO UTOMO
NRP. 5111 100 084

Disetujui oleh Dosen Pembimbing Tugas Akhir:

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

Ridho Rahman Hariadi, S.Kom., M.Sc. .....
NIP: 19870213 201404 1 001 (pembimbing 2)

SURABAYA
JUNI, 2015

KATA PENGANTAR

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

Segala puji dan syukur, kehadirat Allah Subhanahu wa ta'ala yang telah memberikan rahmat dan hidayah-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul "RANCANG BANGUN MODUL PROCEDURAL CONTENT GENERATION PADA PERMAINAN FRIEND AND FOE".

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. Allah SWT atas limpahan rahmat dan rezeki-Nya sehingga penulis dapat menyelesaikan Tugas Akhir.
2. Ayah penulis, Nur Gutomo, dan Ibu penulis, Toenik, yang selalu memberikan dukungan, doa, perhatian, dan kasih sayang.
3. Bapak Imam Kuswardayan selaku dosen pembimbing Tugas Akhir pertama dan yang telah memberikan arahan dalam pengerjaan Tugas Akhir ini.
4. Bapak Ridho Rahman Hariadi selaku dosen pembimbing Tugas Akhir kedua yang dengan sabar membimbing penulis dalam pengerjaan Tugas Akhir ini.
5. Bapak Dwi Sunaryono, selaku dosen wali semester satu sampai enam yang berkenan memberi motivasi dan arahan selama penulis menjalani studi S1.
6. Bapak Radityo Anggoro selaku dosen koordinator Tugas Akhir yang telah membantu penulis atas segala sesuatu mengenai syarat-syarat dan terlaksananya sidang Tugas Akhir.

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. Sarianti Pundi Abdyantary yang selalu membantu, memberikan semangat dan omelan demi terselesainya tugas akhir ini.
10. The Contracant Company (Ruslan, Tommy, Rahman, Faris, Punggi, Besta, Dapik, Yunus, Tev, Risal, Toto) yang selalu rusuh.
11. Rekan-rekan dan pengelola Laboratorium Interaksi, Grafika, dan Seni yang telah memberikan fasilitas dan kesempatan melakukan riset atas Tugas Akhir yang dikerjakan penulis.
12. Rekan-rekan dan sahabat-sahabat penulis angkatan 2011 yang memberikan dorongan motivasi dan bantuan kepada penulis.
13. 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, Juni 2015
Penulis

Andrie Prasetyo Utomo

DAFTAR ISI

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR KODE SUMBER

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

Survival merupakan sebuah aliran di dalam permainan digital yang termasuk di bawah aliran *action-adventure*. Dalam aliran ini, pemain diharuskan untuk bertahan selama mungkin hingga mencapai batasan di mana pemain dinyatakan telah kalah.

Permainan digital dengan aliran *survival* menuntut pemain untuk menjelajah dan memahami peta permainan untuk dapat bertahan hidup. Namun, jika peta permainan setiap kali dimainkan selalu sama, maka permainan akan menjadi kurang menarik karena pemain yang telah lama bermain akan dengan mudah menghafal bagian-bagian di setiap peta permainan.

Oleh karena itu, melalui tugas akhir ini akan dibangun sebuah pembangkit peta permainan secara dinamis menggunakan *Procedural Content Generation* (PCG) untuk membangun bentuk yang berbeda-beda setiap kali permainan dimainkan dari awal.

Peta permainan yang berubah setiap permainan dimulai dari awal akan membuat permainan terkadang sangat sulit atau sangat mudah. Oleh karena itu, untuk menjaga agar permainan tetap mampu dimainkan, maka dibutuhkan skenario dinamis yang akan menjaga keseimbangan permainan di dalam permainan.

1.2 Rumusan Masalah

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

1. Bagaimana merancang aturan main dari permainan Friend and Foe

2. Bagaimana membuat area permainan yang dapat dihasilkan secara otomatis
3. Bagaimana membuat permainan tetap seimbang antara sumberdaya untuk bertahan hidup dengan bahaya yang ada di dalam satu peta permainan.

1.3 Batasan Masalah

Permasalahan pada tugas akhir ini dibatasi pada perancangan peta permainan dinamis dan skenario keseimbangan menggunakan implementasi *Procedural Content Generation* (PCG).

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah untuk membangun peta secara dinamis beserta skenario keseimbangan permainan sebagai modul dalam permainan Friend and Foe agar peta permainan dan lingkungan di dalamnya bisa berganti setiap kali permainan dimulai dari awal.

1.5 Manfaat

Manfaat yang ingin diperoleh dari hasil pembuatan tugas akhir ini antara lain:

1. Memberikan referensi kepada pengembang permainan dalam perancangan peta di dalam permainan secara otomatis
2. Memberikan permainan yang seimbang baik dari sumber daya maupun bahaya kepada pemain Friend and Foe.

1.6 Metodologi

Pembuatan tugas akhir dilakukan menggunakan metodologi sebagai berikut:

A. Studi literatur

Untuk membatu kelancaran pengerjaan tugas akhir ini, beberapa literatur yang perlu dipelajari adalah sebagai berikut:

1. Pemakaian Unity sebagai *Game Engine* dan *Editor*

2. PCG sebagai pembangkit komponen permainan secara dinamis
3. Implemetasi Algoritma Genetika sebagai fungsi evaluasi dalam PCG
4. *Dynamic Game Balancing* sebagai skenario pengatur keseimbangan.

B. Perancangan perangkat lunak
Pada tahap ini dilakukan analisis dan pendefinisian kebutuhan sistem untuk masalah yang sedang dihadapi. Selanjutnya, dilakukan perancangan sistem dengan beberapa tahap sebagai berikut:
1. Spesifikasi Kebutuhan fungsional
2. Perancangan kasus penggunaan sistem
3. Perancangan diagram aktivitas sistem
4. Perancangan diagram kelas sistem
5. Perancangan proses aplikasi

C. Implementasi dan pembuatan sistem
Pada tahap ini dilakukan pembuatan elemen perangkat lunak. Sistem yang dibuat berpedoman pada rancangan yang telah dibuat pada proses perancangan dan analisis s*item*. Perincian tahap ini adalah sebagai berikut:
1. Implementasi pembangkit peta permainan dinamis
2. Implementasi skenario keseimbangan permainan.

D. Uji coba dan evaluasi
Metode pengujian yang akan dilakukan adalah dengan menggunakan metode *blackbox testing*. *Blackbox testing* adalah pengujian yang berfokus pada spesifikasi fungsional aplikasi. Pengujian aplikasi akan berfokus pada nilai *Correctness* untuk mengetahui bahwa aplikasi yang dibuat telah sesuai dengan spesifikasi dan kebutuhan fungsional.

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 akhir ini terdiri dari beberapa bab yang akan 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, dan antarmuka.

BAB IV IMPLEMENTASI

Bab ini membahas implementasi dari rancangan sistem yang dilakukan pada tahap perancangan. Penjelasan implementasi meliputi implementasi pembangkit 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 KESIMPULAN DAN SARAN

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

BAB VII DAFTAR PUSTAKA

Bab ini merupakan daftar yang digunakan dalam mengembangkan Tugas Akhir.

BAB II
TINJAUAN PUSTAKA

Pada bab ini akan dibahas mengenai teori-teori yang menjadi dasar dari pembuatan tugas akhir. Teori-teori tersebut adalah Algoritma Genetika, *Procedural Content Generation*, *Search-Based Procedural Content Generation*, Permainan Don't Starve, dan *Dynamic Game Balancing*.

2.1 Algoritma Genetika

Algoritma Genetika merupakan salah satu model komputasi yang didasarkan pada proses evolusi [1]. Algoritma ini didasarkan pada proses perkembangan generasi di dalam sebuah populasi yang alami, secara lambat laun mengikuti prinsip seleksi alam, "Siapa yang kuat, dia yang bertahan". Dengan meniru teori evolusi ini, Algoritma Genetika dapat digunakan untuk mencari solusi permasalahan-permasalahan dalam dunia nyata.

Hal-hal yang harus dilakukan untuk menerapkan algoritma genetika adalah sebagai berikut:

1. Mendefinisikan bentuk kromosom individu/*genotype*

   Individu menyatakan salah satu solusi yang mungkin. Individu bisa dikatakan sebagai kromosom (*genotype*). Kromosom ini dapat bernilai biner, float, maupun kombinatorial. Contoh individu biner ditunjukkan pada Gambar 2.1.

Gambar 2.1 Individu

2. Mendefinisikan nilai fitness

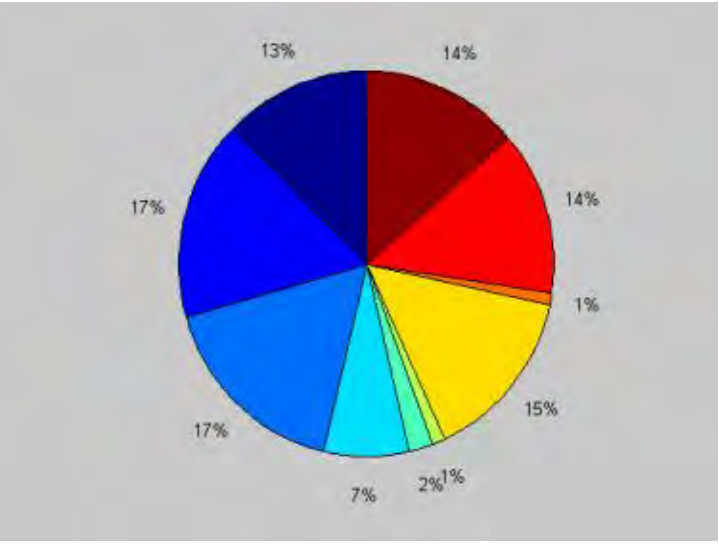
   Nilai fitness merupakan suatu nilai yang dapat menyatakan bahwa suatu individu bernilai baik atau tidak. Nilai fitness berisi aturan dan penilaian berupa poin terhadap suatu individu.

3. Menentukan pembangkitan populasi awal

   Populasi merupakan sekumpulan individu yang akan diproses bersama dalam satu siklus evolusi. Setiap permasalahan bisa diwakili dengan individu yang berbeda-beda. Oleh karena itu, pembangkitan populasi awal berperan untuk membentuk individu-individu yang cocok dengan permasalahan.

4. Menentukan proses seleksi

   Seleksi adalah proses pemilihan calon induk. Terdapat beberapa metode yang dapat digunakan untuk melakukan seleksi, yaitu Roulette Wheel, Competition, atau Tournament. Metode yang paling banyak dipakai adalah metode Roulette Wheel. Dalam metode ini, setiap individu dalam populasi akan memiliki prosentase terpilih yang berbeda-beda sesuai dengan nilai fitness yang dimiliki. Gambaran prosentase tiap individu digambarkan pada Gambar 2.2.

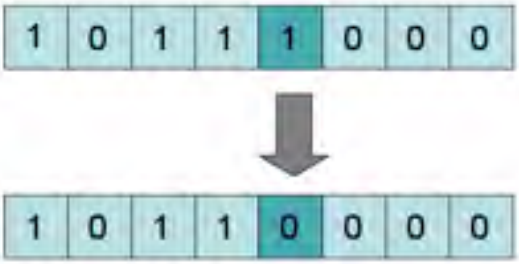


Gambar 2.2 Roulette Wheel

5. Menentukan proses *crossover* dan mutasi

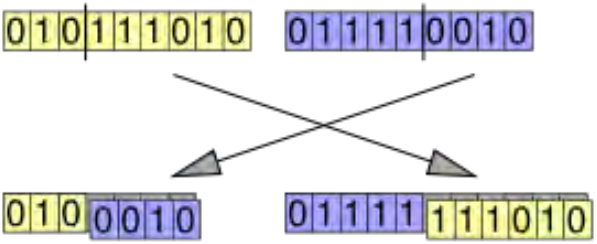
Crossover dan mutasi adalah suatu cara yang digunakan untuk menghasilkan individu yang baru. Individu baru tersebut akan dibandingkan dengan induknya. Jika individu tersebut memiliki nilai fitness yang lebih baik, maka individu tersebut akan menggantikan posisi induknya di dalam populasi.

Mutasi berfokus pada satu induk saja. Pada Gambar 2.3 dijelaskan bagaimana proses mutasi mampu menghasilkan individu baru dengan mengganti salah satu nilai dari *genotype* dengan nilai yang lain.



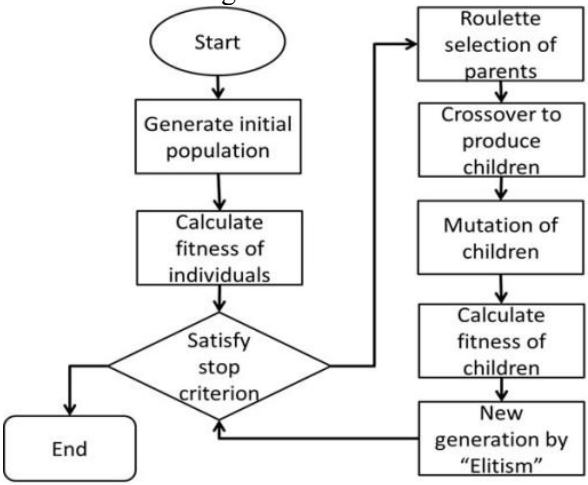
Gambar 2.3 Proses Mutasi

Crossover berfokus pada dua induk. Pada Gambar 2.4 digambarkan proses crossover dalam menyilangkan sebagian nilai dari satu induk untuk menggantikan sebagian nilai dari induk yang lain untuk menghasilkan individu yang baru.



Gambar 2.4 Proses Crossover

Proses Genetika Algoritma akan dijalankan terus menerus hingga ditemukan individu dengan nilai terbaik atau ketika total generasi telah mencapai maksimal. Menjelaskan proses Algoritma Genetika dalam bentuk diagram alir.

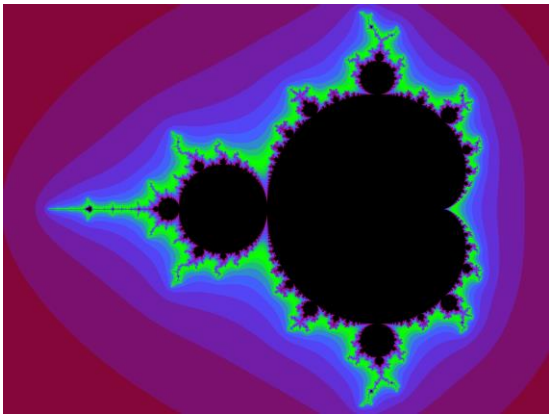


Gambar 2.5 Diagram Alir Proses Algoritma Genetika

2.2 *Procedural Content Generation*

Procedural Content Generation (PCG) mengacu pada pembuatan kontent *game* secara otomatis melalui cara-cara algoritmik [2]. konten *game* berarti seluruh aspek yang mempengaruhi alur permainan, kecuali perilaku *non-player character* (NPC) atau *game engine*. Aspek yang dimaksud bisa berarti *terrain*, peta, *level*, cerita, dialog, misi, sudut pandang kamera, maupun senjata.

Istilah *Procedural* mengacu pada proses untuk menghitung fungsi tertentu. Contohnya, Fraktal pada Gambar 2.6 merupakan salah satu contoh dari penerapan *Procedural Generation*. Fraktal memiliki bentuk dimana seluruh bagan dibentuk dengan komputasi matematika secara geometri.



Gambar 2.6 Geometri Fraktal

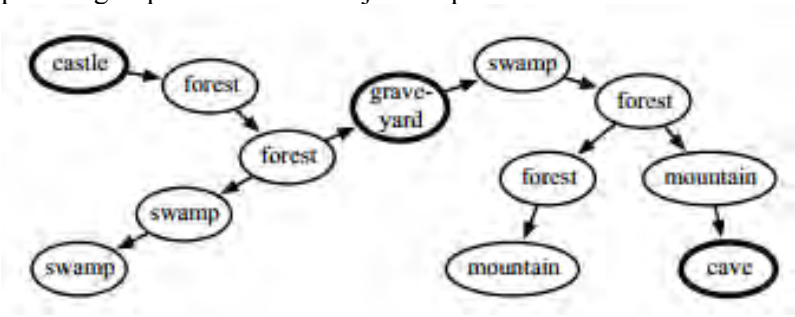
2.3 *Search-Based Procedural Content Generation*

Search-Based Procedural Content Generation (SBPCG) merupakan PCG yang melibatkan algoritma pencarian untuk menentukan model komponen yang akan dibangkitkan [3]. Contoh komponen yang dapat dibangkitkan menggunakan SBPCG adalah dekorasi lingkungan virtual, *level* permainan, maupun misi permainan. Ada beberapa proses yang harus dilakukan untuk membangunn SBPCG, yaitu menentukan *genotype representation*, menentukan *fitness function*, menentukan algoritma pencarian, dan membangun *graphical realization*.

2.3.1 Menentukan *Genotype Representation*

Genotype Representation merupakan suatu cara yang dapat dilakukan untuk menggambarkan komponen yang ingin dibentuk ke dalam bentuk data. *Genotype* harus berupa struktur data sederhana sehingga proses pencarian solusi dalam algoritma pencarian bisa dihasilkan dengan lebih cepat. Sebagai contoh pada kasus pembangkit peta dinamis, peta permainan dapat digambarkan ke dalam *space tree. Space tree* merupakan struktur data tree di mana tiap-tiap *node* merupakan representasi dari bioma, dan tiap-tiap panah yang menunjuk bioma menyatakan

keterhubungan antar bioma. Contoh *space tree* untuk kasus pembangkit peta dinamis ditunjukkan pada Gambar 2.7.



Gambar 2.7 Space Tree

2.3.2 Menentukan *Fitness Function*

Fitness function adalah sebuah fungsi yang dapat digunakan untuk menentukan kesesuaian *genotype* yang telah dibuat dengan model komponen yang ingin dibentuk. *Fitness function* berisi batasan-batasan yang harus diikuti oleh *genotype* untuk menentukan nilai *genotype* tersebut. Pada kasus pembangkit peta dinamis, contoh batasan-batasan yang dapat diterapkan adalah sebagai berikut:

1. Penalti 5 poin untuk setiap bioma yang tidak ada dalam *genotype*
2. Penalti 1 poin jika Bioma Swamp tersambung langsung dengan Bioma Forest.

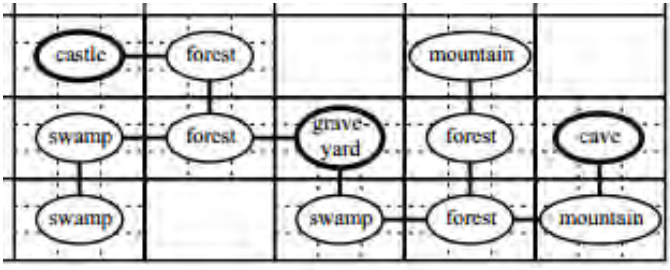
Setiap batasan yang dilanggar akan memberikan nilai penalti untuk *genotype* tersebut. *Genotype* terbaik adalah *genotype* dengan nilai penalti nol atau paling minimal di antara semua *genotype* dalam populasi.

2.3.3 Implementasi Algoritma Pencarian

Algoritma pencarian yang digunakan pada SBPCG pada umumnya adalah Algoritma Genetika. Algoritma Genetika dipilih karena algoritma ini cenderung dapat digunakan untuk mencari solusi atas banyak permasalahan.

2.3.4 *Graphical Realization*

Graphical Realization merupakan suatu cara yang dapat digunakan untuk mengubah model data menjadi bentuk objek nyata di lingkungan virtual. Dalam kasus pembangkit peta dinamis, solusi yang sudah didapatkan akan dipetakan ke dalam *2D tiled-grid* untuk memudahkan proses peletakan objek ketika dilakukan *rendering*. *2D tiled-grid* merupakan struktur data berupa matriks yang dapat digunakan untuk mewakili penggambaran dunia permainan secara keseluruhan di mana tiap petak dapat menggambarkan suatu daerah permainan. Gambar 2.8 menggambarkan space tree yang telah dibuat dipetakan ke dalam bentuk *tiled-grid*.



Gambar 2.8 Space Tree Yang Dipetakan Ke Dalam *Tiled* Grid.

2.4 Don't Starve

Don't Starve merupakan permainan digital dengan jenis aliran *survival*-horror. Permainan ini dikembangkan oleh sebuah perusahaan *indie* bernama Klei Entertainment. Don't Starve menerima ulasan yang cukup positif dengan poin mencapai 79,06 % pada GameRangkings [4]. Don't Starve terjual hingga satu juta salinan pada akhir 2013 [5]. Gambar 2.9 menunjukkan tampilan permainan dari Don't Starve.

Gambar 2.9 Don't Starve

Di dalam permainannya, peta permainan beserta elemen untuk bertahan hidup akan dibuat secara acak dan otomatis. Akan terdapat beberapa jenis bioma dengan karakteristiknya masing-masing yang saling terhubung dan membentuk suatu daratan, di mana rangkaian bioma ini akan berubah posisinya setiap permainan dimulai dari awal. Gambar 2.10 menampilkan peta permainan Don't Starve dalam mode tampil atas.

Gambar 2.10 Peta Permainan Don't Starve

Ada dua jenis mode dalam permainan yang dapat diikuti, yaitu mode *survival* atau *adventure*. Mode *survival* mengharuskan pemain untuk mampu bertahan selama mungkin dengan mengatur sebaik mungkin tingkat kelaparan, kegilaan, dan kesehatan. Pemain harus sebaik mungkin mendirikan sebuah tempat berlindung untuk meyimpan makanan, perlengkapan keselamatan, berlindung dari hewan buas, maupun mengatasi kegilaan, serta empat musim yang masing-masing memiliki karakteristik masing-masing. Sedangkan permainan dengan mode adventure akan membawa pemain untuk fokus mengalahkan lawan bernama Maxwell yang terperangkap di dalam sebuah gua.

Di dalam tugas akhir ini, akan diimplementasikan bagaimana membuat peta petualangan yang mampu dihasilkan secara dinamis dengan rangkaian bioma yang posisinya dapat berganti-ganti seperti permainan Don't Starve. Selain itu, akan ada skenario dinamis yang mampu menjaga kesimbangan sumber daya dalam peta permainan.

2.5 *Dynamic Game Balancing*

Dynamic Game Balancing merupakan sebuah proses yang dilakukan agar permainan dapat menyesuaikan dengan tingkat kemampuan pemain. Menyeimbangkan permainan berarti mengubah parameter-parameter, skenario, dan tingkah laku permainan untuk mencegah pemain merasa frutasi karena permainan yang terlalu sulit atau mencegah pemain merasa bosan karena permainan yang terlalu mudah [6]. Tujuan adanya keseimbangan adalah untuk menjaga pemain tetap bertahan dari awal hingga akhir permainan [6].

Implementasi sederhana dari *Dynamic Game Balancing* adalah adanya sebuah sistem yang mampu membantu pemain ketika permainan terlalu sulit, dan sistem untuk menaikkan tingkat kesulitan permainan ketika permainan terlalu mudah.

BAB III
ANALISIS DAN PERANCANGAN

Bab ini membahas tahap analisis dan perancangan dari permainan yang akan dibangun. Analisis sistem membahas permasalahan yang yang diangkat dan deskripsi umum mengenai perangkat lunak. Selanjutnya dibahas mengenai perancangan sistem yang dibuat. Pendekatan yang dibuat dalam perancangan ini adalah pendekatan berorientasi objek. Perancangan direpresentasikan dengan diagram UML (*Unified Modelling Language*).

3.1 Analisis Sistem

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

3.1.1 Analisis Permasalahan

Permasalahan utama yang diangkat dalam pembuatan tugas akhir ini adalah bagaimana membuat peta permainan secara dinamis, sehingga peta permainan yang dibangkitkan selalu bervariasi.

Permasalahan kedua yang akan dibahas adalah bagaimana menentukan keseimbangan kesulitan selama permainan. Keseimbangan permainan berarti bahwa sistem mampu membantu pemain dengan cara yang tidak langsung ketika pemain merasa kesulitan, dan mampu menambah tingkat kesulitan permainan ketika permainan terasa terlalu mudah.

Permasalahan terakhir yang akan dibahas dalam tugas akhir ini adalah bagaimana merancang aturan permainan. Aturan permainan meliputi bagaimana proses menang dan kalah dalam permainan, bagaimana perilaku sistem selama permainan, serta *user action* yang dapat dilakukan oleh pemain.

3.1.2 Deskripsi Umum Perangkat Lunak

Tugas akhir yang akan dikembangkan adalah sebuah permainan dengan jenis *survival-horror*. Tujuan utama permainan ini adalah untuk bertahan hidup pada lingkungan permainan dalam batas waktu tertentu.

Permainan ini akan dikembangkan dengan modul *Procedural Content Generation* sehingga lingkungan permainan akan berubah-ubah setiap kali permainan dimulai dari awal. *Procedural Content Generation* juga akan digunakan untuk melakukan kontrol terhadap isi dari peta permainan. Isi yang dimaksud adalah objek-objek yang dapat muncul dalam peta permainan. Dengan adanya kontrol objek, maka keseimbangan permainan dapat dijaga.

3.2 Perancangan Sistem

Penjelasan tahap perancangan sistem dibagi menjadi beberapa subbab, yaitu spesifikasi kebutuhan fungsional, kasus penggunaan, perancangan aturan permainan, perancangan pembangkit peta dinamis, dan perancangan skenario keseimbangan.

3.2.1 Spesifikasi Kebutuhan Fungsional

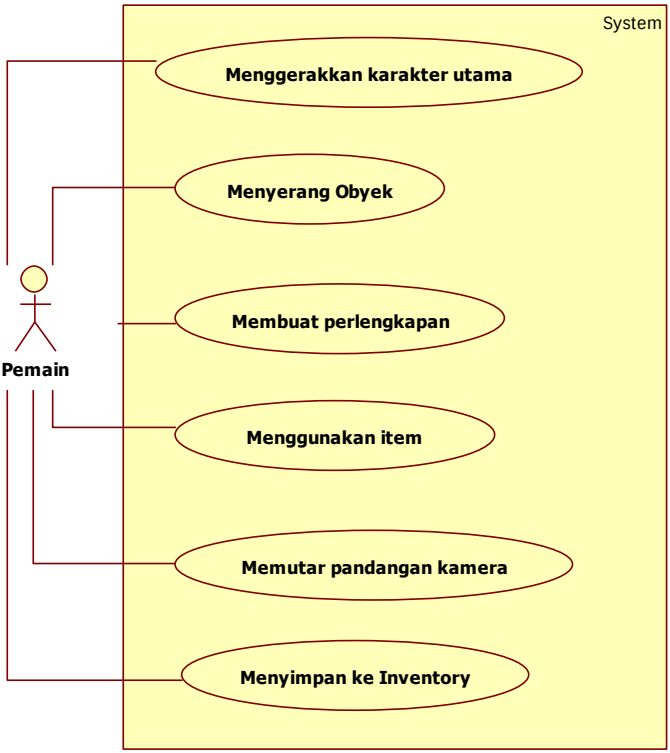
Kebutuhan fungsional berisi proses-proses yang harus dimiliki sistem. Kebutuhan fungsional mendefinisikan layanan yang harus disediakan dan reaksi sistem terhadap masukan atau pada situasi tertentu. Daftar kebutuhan fungsional dapat dilihat pada Tabel 3.1.

Tabel 3.1 Kebutuhan Fungsional

Kode	Deskripsi
F-0001	Sistem dapat membangkitkan peta dinamis
F-0002	Pengguna dapat bermain hingga permainan selesai
F-0003	Sistem mampu mengatur keseimbangan permainan

3.2.2 Kasus Penggunaan

Kasus-kasus penggunaan dalam sistem ini akan dijelaskan secara rinci pada subbab ini. Terdapat enam kasus penggunaan dalam sustem ini, yaitu menggerakkan karakter utama, menyerang objek, membuat perlengkapan, menggunakan *item*, memutar pandangan kamera, dan menyimpan *item* ke *inventory*. Penjelasan mengenai kasus penggunaan akan dijelaskan dalam Gambar 3.1 dan Tabel 3.2.



Gambar 3.1 Diagram Kasus Aplikasi

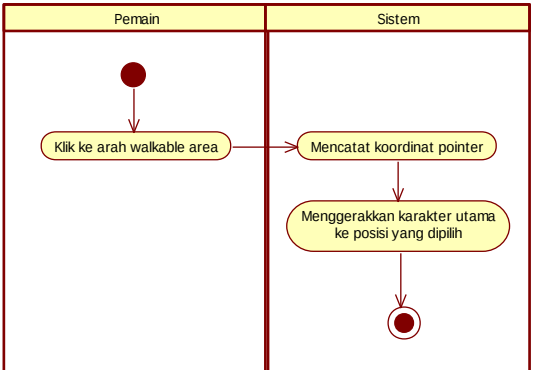
Tabel 3.2 Daftar Kasus Penggunaan

No	Kode	Nama Kasus Penggunaaan
1	UC-0001	Menggerakkan karakter utama
2	UC-0002	Menyerang objek
3	UC-0003	Membuat perlengkapan
4	UC-0004	Menggunakan *item*
5	UC-0005	Memutar pandangan kamera
6	UC-0006	Menyimpan *item* ke *inventory*

3.2.2.1 Menggerakkan Karakter Utama

Kasus penggunaan dengan kode UC-0001 digunakan untuk membantu pemain dalam menggerakkan karakter utama sesuai dengan posisi masukan yang diberikan oleh pemain. Spesifikasi kasus penggunaan menggerakkan karakter utama dijelaskan melalui Tabel 3.3.

Alur kasus penggunaan menggerakkan pemain diawali dengan kondisi ketika karakter utama telah memasuki area permainan. Di dalam permainan, terdapat area-area yang bisa dilewati atau tidak oleh karakter utama. Pemain harus mengarahkan *pointer* dan melakukan klik pada area yang dapat dilewati oleh karakter utama. Selanjutnya, sistem akan menggerakkan karakter utama ke area tempat *pointer*. Diagram aktivitas mengenai kasus penggunaan menggerakkan karakter dijelaskan pada Gambar 3.2.



Gambar 3.2 Diagram Aktivitas Menggerakkan karakter utama

Tabel 3.3 Spesifikasi Kasus Penggunaan Menggerakkan Karakter

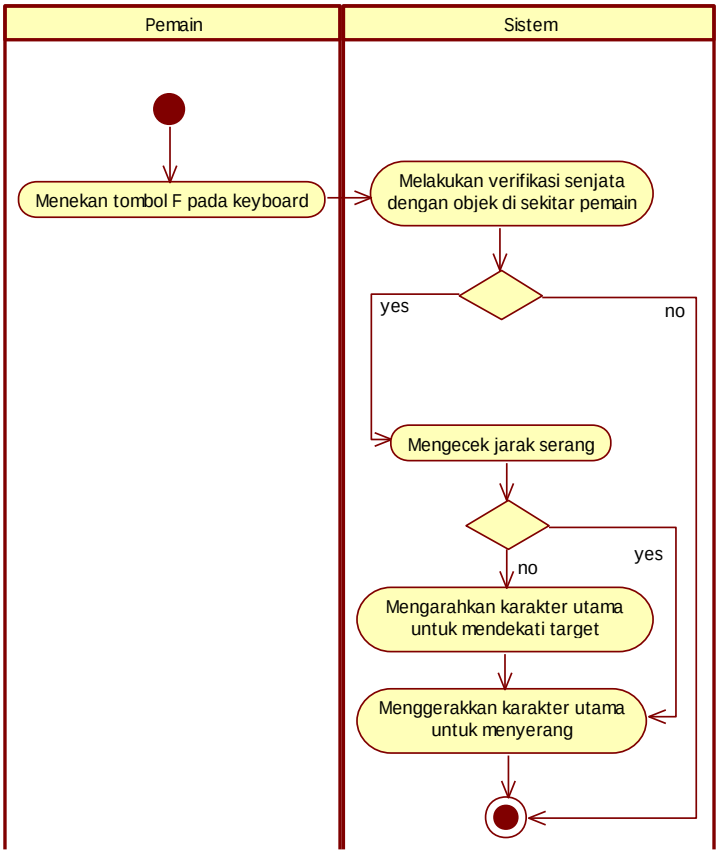
Nama	Menggerakkan karakter utama
Kode	UC-0001
Deskripsi	Pemain menggerakkan karakter utama di area permainan
Aktor	Pemain
Kondisi Awal	Karakter utama sudah memasuki area permainan
Alur Normal	1. Pemain melakukan klik *pointer* ke *walkable area*. 2. Sistem mencatat koordinat *pointer* 3. Sistem menggerakkan karakter utama ke posisi yang dipilih
Alur Alternatif	-
Kondisi Akhir	Karakter utama berada di posisi koordinat yang dipilih

3.2.2.2 Menyerang Objek

Tabel 3.4 Spesifikasi Kasus Menyerang Objek

Nama	Menyerang objek
Kode	UC-0002
Deskripsi	Pemain menyerang target dalam permainan
Aktor	Pemain
Kondisi Awal	Pemain telah memakai senjata dan berada di sekitar musuh
Alur Normal	1. Pemain menekan tombol F pada *keyboard*. 2. Sistem melakukan verifikasi kecocokan antara senjata dengan objek di sekitar pemain. 2.A. Tidak ada objek yang cocok dengan senjata terpakai. 3. Sistem menandai objek yang cocok sebagai target. 4. Sistem mengecek jika target di dalam jarak serang. 3.A. Jarak serang lebih besar dari jarak karakter utama ke target. 5. Sistem menyerang target.
Alur Alternatif	2.A Tidak ada objek yang cocok dengan senjata terpakai 2.A.1 Sistem mengakhiri perintah serangan 3.A Jarak serang lebih besar dari jarak karakter ke target 3.A.1 Sistem mengarahkan karakter utama untuk mendekati target
Kondisi Akhir	Karakter utama menyerang target

Kasus penggunaan dengan kode UC-0002 digunakan untuk menyerang *attackable object* yang terdapat di dalam permainan. *Attackable object* bisa berupa hewan, bebatuan, pepohonan, dll. Spesifikasi kasus penggunaan menyerang objek dijelaskan pada Tabel 3.4**.**



Gambar 3.3 Diagram Aktivitas Menyerang Objek

Alur kasus penggunaan menyerang objek dimulai dengan kondisi ketika pemain telah memilih senjata dan berada di sekitar *attackable object*. Kemudian pemain harus menekan tombol F pada *keyboard*. Kemudian sistem akan megecek jarak serang dan kecocokan antara senjata dengan objek di sekitar pemain. Objek di sekitar pemain yang cocok dengan senjata akan disebut target. Selanjutnya sistem akan mengarahan karakter utama untuk

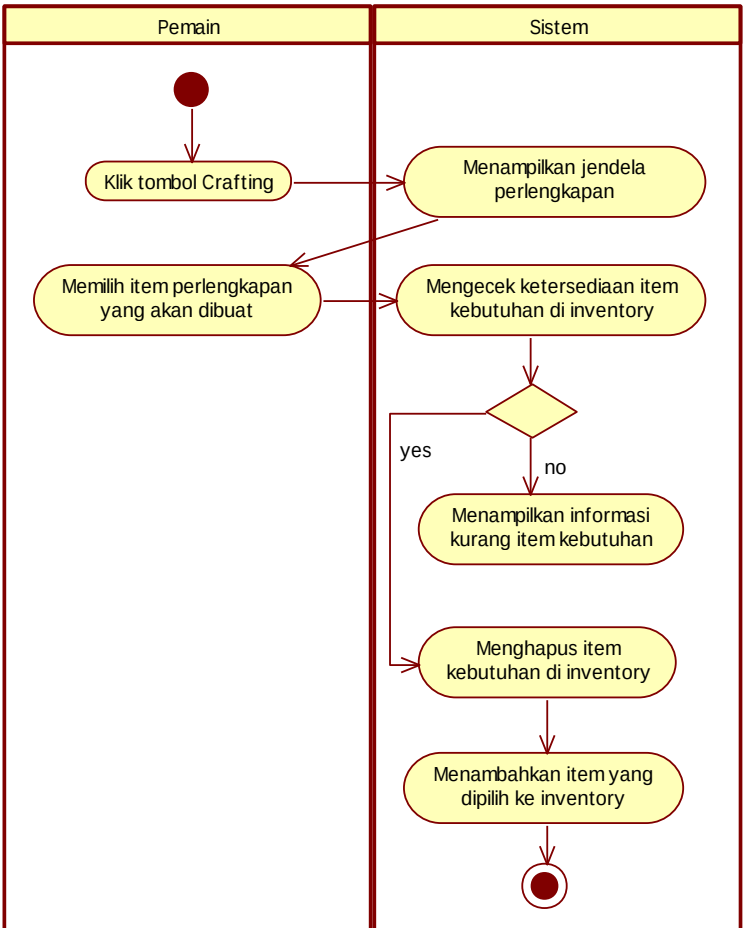
mendekati target, atau langsung menyerang target jika jarak serang yang dibutuhkan telah dicapai. Diagram Aktivitas kasus penggunaan menyerang objek dijelaskan pada Gambar 3.3**.**

3.2.2.3 Membuat Perlengkapan

Tabel 3.5 Spesifikasi Penggunaan Membuat Perlengkapan

Nama	Membuat perlengkapan
Kode	UC-0003
Deskripsi	Pemain membuat perlengkapan untuk bertahan hidup
Aktor	Pemain
Kondisi Awal	Pemain memiliki *item* di *inventory*
Alur Normal	1. Pemain melakukan klik tombol Crafting**.** 2. Sistem menampilkan jendela perlengkapan. 3. Pemain memilih *item* perlengkapan yang akan dibuat. 4. Sistem mengecek ketersediaan *item* kebutuhan perlengkapan di *inventory*. 4.A. kebutuhan perlengkapan tidak tersedia. 5. Sistem menghapus semua *item* kebutuhan perlengkapan di *inventory*. 6. Sistem menambahkan *item* perlengkapan ke *inventory*.
Alur Alternatif	4.A kebutuhan perlengkapan tidak tersedia. 4.A.1 Sistem menampilkan informasi bahwa *item* kebutuhan tidak lengkap.
Kondisi Akhir	Perlengkapan yang dipilih pemain ditambahkan ke *inventory*.

Kasus penggunaan dengan kode UC-0003 digunakan untuk membuat perlengkapan yang digunakan untuk bertahan hidup, seperti palu, kapak, api unggun, dll. Perlengkapan merupakan *item* yang dapat dibentuk dari beberapa *item* lain di dalam permainan. Spesifikasi kasus penggunaan membuat perlengkapan dijelaskan pada Tabel 3.5**.**



Gambar 3.4 Diagram Aktivitas Membuat Perlengkapan

Alur kasus penggunaan membuat perlengkapan dimulai dengan pemain melakukan klik pada tombol Crafting. Kemudian pemain harus memilih salah satu perlengkapan yang ingin dibuat. Setelah itu, sistem akan mengecek apakah ketersediaan *item* yang dibutuhkan untuk membuat *item* perlengkapan di *inventory*. Ketika *item* kebutuhan dinyatakan tersedia, *item* tersebut akan

dihapus dari *inventory* dan digantikan dengan *item* perlengkapan yang ingin dibuat. Diagram aktivitas mengenai kasus penggunaan membuat perlengkapan dijelaskan melalui Tabel 3.5**.**

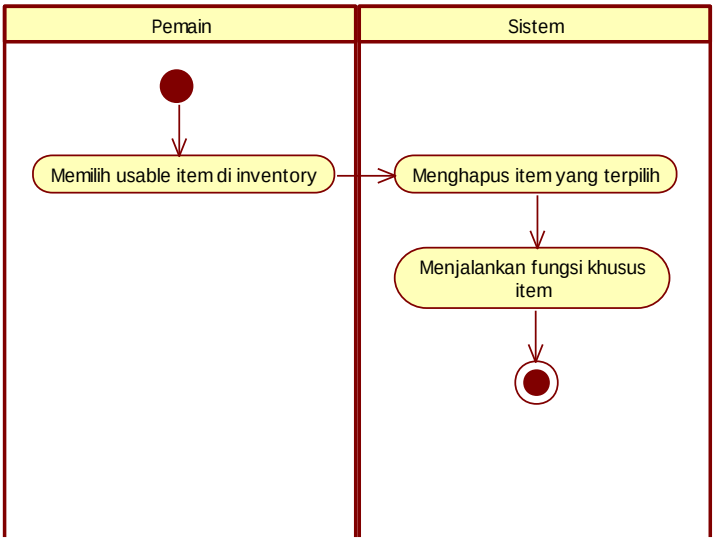
3.2.2.4 Menggunakan *Item*

Kasus penggunaan dengan kode UC-0004 digunakan untuk memakai *usable item* dalam *inventory*. *Usable item* adalah *item* yang memiliki fungsi tertentu ketika dipakai. Spesifikasi kasus penggunaan untuk menggunakan *item* dijelaskan melalui Tabel 3.6**.**

Tabel 3.6 Spesifikasi Kasus Penggunaan Menggunakan *Item*

Nama	Menggunakan *item*
Kode	UC-0004
Deskripsi	Pemain menggunakan *usable item* yang disimpan dalam *inventory*
Aktor	Pemain
Kondisi Awal	Pemain memiliki *usable item* di *inventory*
Alur Normal	1. Pemain memilih *usable item* di *inventory*. 2. Sistem menghapus *item* terpilih dari *inventory*. 3. Sistem menjalankan perintah sesuai dengan item yang digunakan
Alur Alternatif	-
Kondisi Akhir	*Item* yang digunakan hilang dari *inventory* dan sistem menjalankan fungsi khusu item yang digunakan.

Alur kasus penggunaan menggunakan *item* dimulai dengan kondisi ketika pemain telah memiliki *usable item* di dalam *inventory*. Pemain harus memilih *item* tersebut dengan melakukan klik pada *item* tersebut. Setiap *item* akan memiliki fungsi khusus masing-masing. Setelah *item* dipilih, maka sistem akan menjalankan fungsi khusus tersebut. Diagram Aktivitas kasus penggunaan menggunakan *item* dijelaskan pada Tabel 3.6**.**



Gambar 3.5 Diagram Aktivitas Menggunakan *Item*

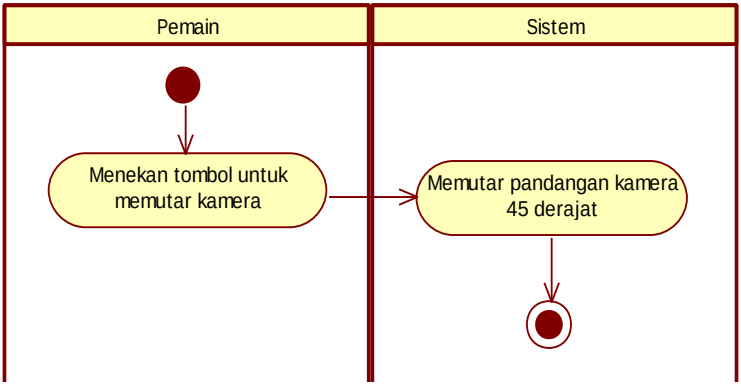
3.2.2.5 Memutar Pandangan Kamera

Kasus penggunaan dengan kode UC-0007 digunakan untuk memutar kamera dengan sumbu-y (atas) untuk membantu pemain dalam melihat kondisi di sekitar karakter utama. Spesifikasi kasus penggunaan memutar pandangan kamera dijelaskan pada Tabel 3.7.

Alur kasus penggunaan untuk memutar pandangan kamera dimulai ketika karakter utama telah memasuki area permainan dan permainan siap dimulai. Selanjutnya pemain harus menekan tombol untuk memutar kamera sehingga sistem akan memutar kamera sebesar 45 derajat sesuai dengan tombol putar kiri atau tombol putar kanan. Diagram aktivitas memutar pandangan kamera dijelaskan pada Gambar 3.6**.**

Tabel 3.7 Spesifikasi Kasus Penggunaan Memutar Pandangan Kamera

Nama	Memutar pandangan kamera
Kode	UC-0007
Deskripsi	Pemain memutar pandangan kamera dengan sumbu-y
Aktor	Pemain
Kondisi Awal	Karakter utama memasuki area peramainan
Alur Normal	1. Pemain menekan tombol untuk memutar kamera (Q atau E). 2. Sistem memutar kamera sebesar 45 derajat.
Alur Alternatif	-
Kondisi Akhir	Pandangan kamera berputar 45 derajat.



Gambar 3.6 Diagram Aktivitas Memutar Pandangan Kamera

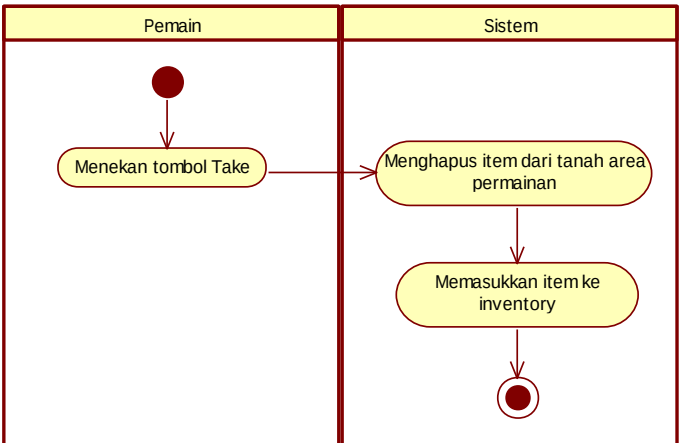
3.2.2.6 Menyimpan *Item* ke *Inventory*

Kasus penggunaan dengan kode UC-0008 digunakan untuk menyimpan *takeable item* ke dalam *inventory*. Spesifikasi kasus penggunaan untuk menyimpan *item* ke *inventory* dijelaskan melalui Tabel 3.8**.**

Tabel 3.8 Spesifikasi Kasus Penggunaan Menyimpan *Item* ke *Inventory*

Nama	Menyimpan ke *Inventory*
Kode	UC-0008
Deskripsi	Pemain menyimpan *takable item* ke dalam *inventory*
Aktor	Pemain
Kondisi Awal	Karakter utama berada di sekitar *takable item.*
Alur Normal	1. Pemain menekan tombol Take. 2. Sistem menghapus *item* di tanah area permainan. 3. Sistem memasukkan *item* ke *inventory.*
Alur Alternatif	-
Kondisi Akhir	*Item* berpindah dari tanah ke *inventory* karakter utama

Alur penggunaan menyimpan *item inventory* diawali dengan kondisi ketika karakter utama telah memasuki area permainan. Karakter utama sebelumnya harus berada di sekitar *takable item.* Kemudian pemain harus menekan tombol Take yang ada pada layar. Selanjutnya sistem akan menghapus *item* yang ada di area tanah permainan, dan memasukkan *item* tersebut ke dalam *inventory* pemain. Diagram Aktivitas kasus penggunaan menyimpan *item* ke *inventory* dijelaskan pada Gambar 3.7.



Gambar 3.7 Diagram Aktivitas Memasukkan *Item* ke *Inventory*

3.2.3 Perancangan Aturan Permainan

Pada permulaan permainan, sistem akan membangkitkan peta permainan secara dinamis. Kemudian karakter utama akan ditempatkan pada salah satu bioma yang disebut *spawn point* dari karakter utama. Bioma yang menjadi *spawn point* tersebut adalah bioma yang memiliki karakteristik paling aman.

Karakter utama memiliki dua status kesehatan. Status pertama adalah *health status*. *Health status* memiliki peran paling penting karena jika health status mencapai nilai nol, maka pemain dinyatakan kalah. Sedangakan status kedua adalah *hunger status*. *Hunger status* memiliki peran untuk menjaga *health status*. Jika *hunger status* mencapai nilai nol, maka *health status* pemain akan berkurang terus menerus.

Pemain dapat mengumpulkan sumber daya yang tersebar di area permainan. Sumber daya merupakan seluruh benda yang ada dalam peta permainan yang memiliki nilai guna untuk membantu pemain dapat bertahan. Beberapa sumber daya memerlukan alat khusus untuk dapat dimanfaatkan. Alat-alat yang dimaksud seperti: palu untuk memecah batu besar, kapak untuk memotong pohon, atau tombak untuk membunuh hewan buas. Barang yang

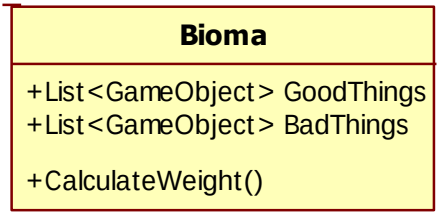
dikumpulkan pemain dapat dikombinasikan untuk membentuk barang baru dengan nilai guna yang lebih tinggi.

Selain untuk bertahan hidup, permainan ini juga mengharuskan pemain untuk mengalahkan satu musuh yang menentukan kemenangan pemain. Musuh tersebut merupakan batu bertuah yang selalu berada di area Bioma Swamp. Setiap dua kali malam, batu tersebut akan memanggil satu hewan buas di dalam peta permainan untuk dijadikan pelindung batu tersebut.

Tugas pemain adalah menjaga agar batu tersebut tetap sendirian ketika waktu menunjukkan malam hari dengan mengalahkan hewan-hewan buas di sekitarnya. Ketika batu tersebut sendirian pada malam hari, maka nilai kerusakan batu tersebut akan bertambah. Namun ketika batu tersebut berhasil mengumpulkan hewan buas yang cukup banyak, batu tersebut akan mampu mengendalikan semua hewan buas tersebut untuk menyerang pemain secara serentak. Batu tersebut akan menyisahkan satu hewan untuk tetap menjaga dia dari kerusakan. Pemain dapat memenangkan permainan jika pemain dapat membuat *health point* dari telur tersebut mencapai nol.

3.2.4 Perancangan Pembangkit Peta Dinamis

Sistem permainan telah menyediakan enam jenis bioma. Masing-masing bioma memiliki bobot sesuai dengan perhitungan objek yang bisa dimunculkan di dalam bioma tersebut. Desain Kelas Bioma ditunjukkan melalui Gambar 3.8. Kelas Bioma berisi dua List<GameObject> yang akan digunakan untuk menampung objek-objek yang dapat dimunculkan di posisi bioma tersebut. Sementara fungsi CalculateWeight() digunakan untuk menghitung bobot yang dimiliki bioma tersebut.



Gambar 3.8 Kelas Bioma

Tabel 3.9 menjelaskan bagaimana perilaku Kelas Bioma dalam menghitung bobot yang dimiliki Kelas Bioma. List GoodThings maupun List BadThings akan diisi dengan objek yang juga memiliki bobot masing-masing. Fungsi CalculateWeight() menghitung hasil jumlah dari rata-rata bobot objek dalam GoodThings dengan rata-rata bobot objek dalam BadThings.

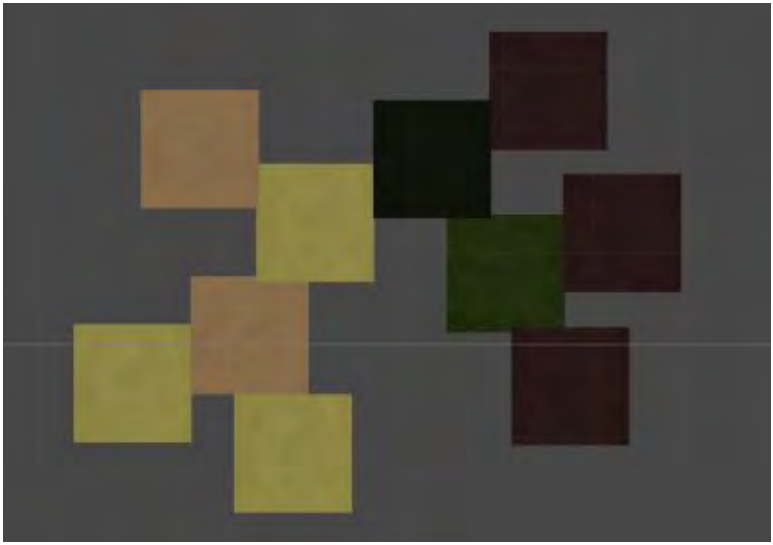
Tabel 3.9 Spesifikasi Bobot Bioma

ID	Nama	Avg (GoodThings)	Avg (BadThings)	Calculate Weight()
0	Desert	1	-3	-2
1	Forest	2	-1	1
2	Graveyard	2	-3	-1
3	Greenland	2	0	2
4	Savana	2	0	2
5	Swamp	0	0	0

Peta permainan merupakan kumpulan bioma yang saling terhubung sehingga membentuk sebuah pulau. Jika bioma tersebut disambung secara acak, maka akan mengurangi nilai guna bioma tersebut. Sebagai contoh, jika permainan membutuhkan Bioma Forest sebagai tempat munculnya pepohonan namun ketika peta permainan dibangkitkan secara acak, ada kemungkinan bahwa Bioma Forest sama sekali tidak

ada di dalam peta permainan. Dengan begitu, pemakaian peta tersebut akan mengganggu jalannya permainan.

Pembangkit peta permainan secara acak memungkinkan peta permainan kurang menarik karena memungkinkan adanya ketimpangan kesulitan di salah satu sisi. Hal tersebut akan membuat pemain enggan menjelajah daerah tersebut.



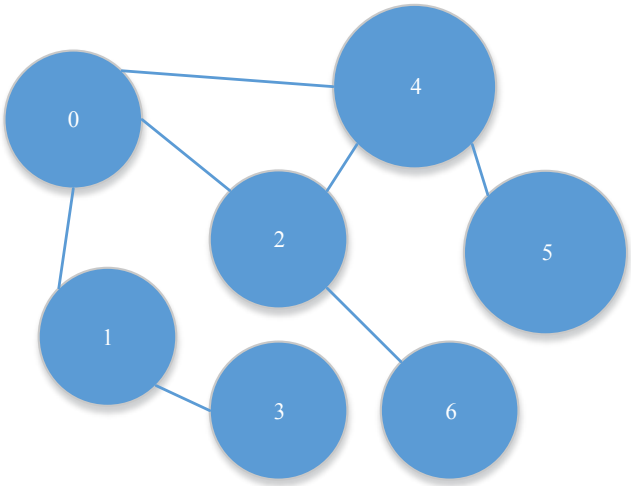
Gambar 3.9 Peta Permainan dengan Bobot Tidak Seimbang

Gambar 3.9 menunjukkan gambaran peta permainan yang tidak seimbang. Dengan peta permainan tersebut, maka pemain cenderung lebih sering berada di daerah sebelah kiri (bobot positif) daripada menjelajah daerah sebelah kanan (bobot negatif).

Untuk dapat membangkitkan peta permainan dengan kombinasi bioma yang sesuai dengan kebutuhan, maka peta dinamis akan dibangkitkan menggunakan metode SBPCG. Perancangan pembangkit peta dinamis akan mengikuti serangkaian proses di dalam SBPCG. Proses-proses tersebut adalah menentukan *genotype representation*, menentukan *fitness function*, implementasi Algoritma Genetika, dan *graphical realization*.

3.2.4.1 Menentukan *Genotype Representation*

Genotype representation merupakan struktur data yang dapat digunakan untuk mewakili bentuk asli peta permainan. Pada pembangkit peta permainan, struktur data tersebut dapat digambarkan melalui sebuah *space tree,* yaitu sebuah tree di mana tiap node dapat mewakili sebuah porsi dari peta permainan. Space tree tersebut kemudian akan dibagi ke dalam dua struktur data yang lebih sederhana. Struktur data yang pertama adalah bentuk buta dari peta permainan yang dibentuk. Struktur buta yang dimaksud adalah bahwa peta tersebut hanya berisi informasi mengenai *node* yang saling terhubung, namun tidak ada informasi mengenai jenis bioma atas *node* tersebut. Struktur data yang kedua berisi daftar mengenai bioma yang akan dipakai dalam peta permainan, namun tidak berisi informasi mengenai hubungan antar bioma. Satu peta permainan utuh didapatkan dengan menggabungkan antara struktur data pertama dengan struktur data yang kedua.



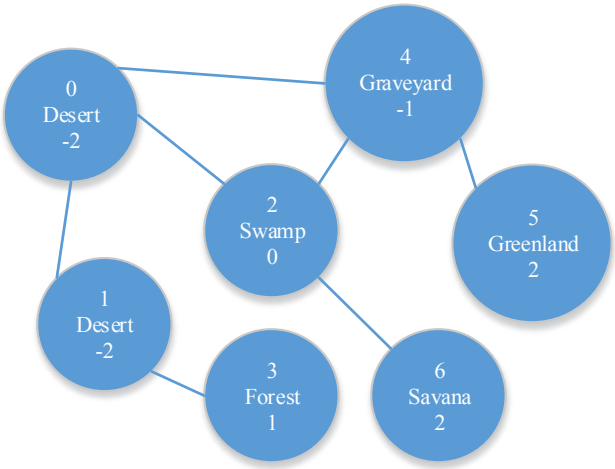
Gambar 3.10 Struktur Data Tanpa Informasi Bioma

Sebagai contoh pembangkitan peta permainan dengan tujuh bioma, Gambar 3.10 menunjukkan struktur data pertama yang

berupa *tree,* yaitu struktur data yang digambarkan menyerupai pohon. Di dalam *tree* tersebut ditunjukkan hubungan keterkaitan antar *node.* Sedangkan informasi bioma yang bersesuaian ditunjukkan pada Gambar 3.11 melalui struktur data berupa *list*, yaitu struktur data yang deretan baris.

Gambar 3.11 *List* Informasi Bioma

Kombinasi dari Gambar 3.10 dengan Gambar 3.11 akan menghasilkan informasi relasi bioma yang lebih jelas yang disebut *space tree*. *Space tree* tersebut ditunjukkan pada Gambar 3.12.



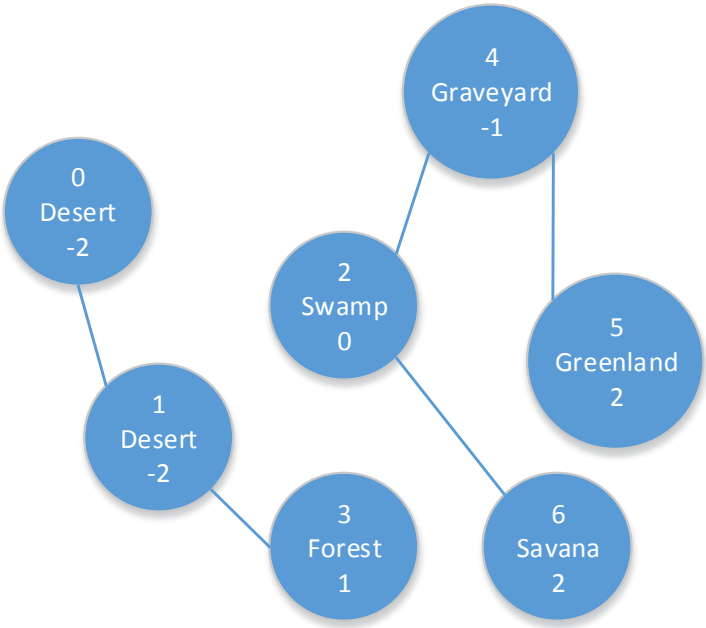
Gambar 3.12 Rancangan Space Tree

Pemisahan *space tree* menjadi dua bertujuan agar proses Implementasi Algoritma Genetika menjadi lebih cepat. Hal itu disebabkan karena dengan hanya mencari kombinasi dari *list*

intformasi bioma, maka implementasi Algoritma Genetika hanya memerlukan struktur data satu dimensi. Sementara *tree* relasi *node* akan dibentuk secara acak, namun dengan batasan sebagai berikut:

1. Semua *node* harus saling terhubung
2. Maksimal tetangga suatu *node* adalah tiga.

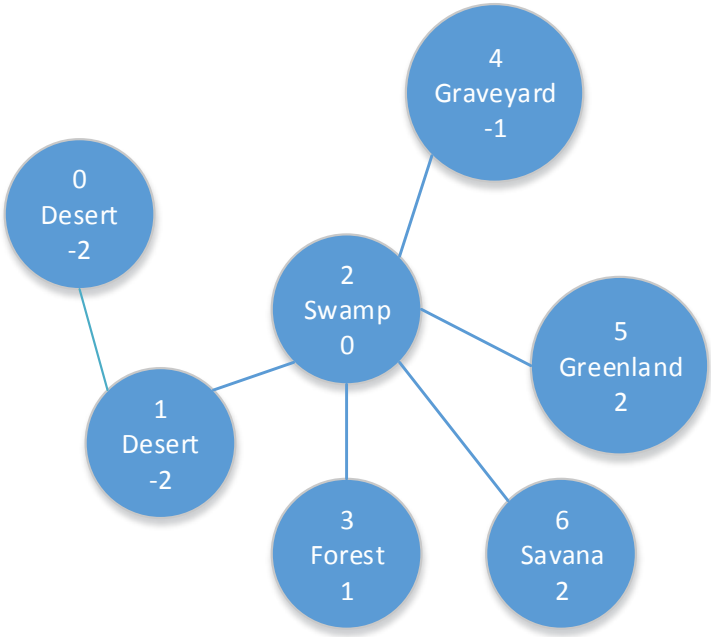
Batasan pertama bertujuan agar tidak ada bioma yang tidak bisa dilewati oleh pemain. Jika ada bioma yang saling tidak terhubung ke semua bioma yang lain, maka akan menyebabkan terbentuknya dua atau lebih pulau permainan seperti ditunjukkan pada Gambar 3.13.



Gambar 3.13 Peta Permainan dengan Bioma Terpisah

Batasan kedua bertujuan agar tidak dibentuk peta permainan yang terlalu terpusat. Peta permainan yang terpusat dianggap kurang menarik karena cenderung menjadikan area permainan

terlalu kecil. Contoh peta permainan yang terpusat ditunjukkan pada Gambar 3.14.



Gambar 3.14 Peta Permainan yang Terlalu Terpusat

3.2.4.2 Menentukan *Fitness Function*

Fitness function adalah sebuah fungsi yang dapat digunakan untuk mengecek kesesuaian *genotype* yang terbentuk dengan kriteria yang diinginkan. Untuk menghitung nilai *fitness*, maka harus didefinisikan batasan dan nilai penalti untuk tiap-tiap batasan yang dilanggar. Besar nilai penalti untuk masing-masing batasan ditentukan berdasarkan tingkat kepentingan batasan yang dilanggar. *Genotype* terbaik ditunjukkan dengan nilai *fitness* yang rendah. Dalam permainan, batasan peta permainan yang diinginkan akan dijelaskan di bawah ini.

1. Peta permainan harus memuat seluruh bioma yang tersedia

- Pengecekan suatu peta permainan telah memuat seluruh bioma atau tidak adalah dengan membuat *checklist* pada indeks bioma yang ada pada struktur data kedua.
- Poin penalti untuk pelanggaran batasan ini adalah 10.
- Pada Gambar 3.12, peta permainan yang terbentuk tidak melanggar batasan ini. Penambahan nilai *fitness* pada pengecekan batasan ini adalah 0.

2. Bioma yang sama tidak boleh terhubung langsung
   - Pengecekan dilakukan dengan menggunakan struktur data gabungan.
   - Poin penalti untuk pelanggaran ini adalah 3 untuk masing-masing bioma yang terhubung.
   - Pada Gambar 3.12, terdapat pelanggaran terhadap batasan ini, yaitu relasi antara *node* (0) dengan *node* (1), dimana kedua *node* tersebut mewakili bioma Desert. Penambahan nilai fitness pada pengecekan batasan ini adalah 3
3. Penjumlahan masing-masing *node* dengan tetangga-tetangga yang langsung terhubung tidak jauh dari angka nol
   - Batasan ini digunakan untuk menjaga peta permainan agar tidak terlalu sulit atau terlalu mudah di salah satu sisi peta saja. Pengecekan dilakukan dengan melihat struktur data gabungan, kemudian untuk masing-masing *node*, dilakukan pengecekan jumlah bobot dengan tetangga-tetangga yang terhubung langsung. Poin penalti untuk pelanggaran batasan ini adalah sebanyak nilai absolut dari jumlah bobot *node* terbesar.
   - Tabel 3.10 menjelaskan mengenai cara penghitungan nilai penalti untuk batasan ini. Dari tabel tersebut, akan diambil nilai *absolute* terbesar sebagai nilai penalti. Sehingga penambahan nilai *fitness* untuk batasan ini adalah sebanyak 5.

Dari ketiga pengujian fungsi *fitness,* didapatkan bahwa nilai fitness untuk *genotype* pada contoh tersebut adalah 0 + 3 + 5 = 8.

Tabel 3.10 Contoh Penghitungan Fungsi Fitness

Node ID	Relasi	Bobot Relasi	*Absolute*
0	Desert-Desert-Graveyard-Swamp	-5	5
1	Desert-Desert-Forest	-3	3
2	Swamp-Desert-Graveyard-Savana	-1	1
3	Forest-Desert	-1	1
4	Graveyard-Desert-Swamp-Greenland	-1	1
5	Greenland-Graveyard	1	1
6	Savana-Swamp	2	1

3.2.4.3 Implementasi Algoritma Genetika

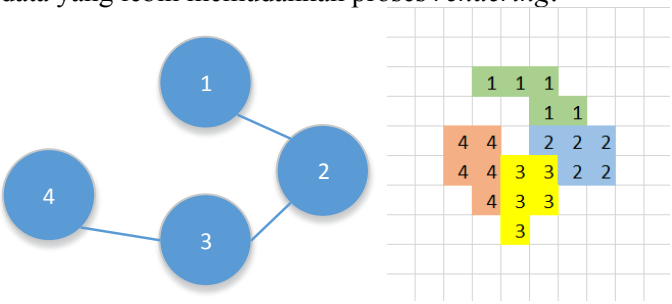
Algoritma Genetika dimulai dengan membuat populasi acak atas *genotype* yang sebelumnya sudah ditentukan. Panjang *list* yang dibuat sama dengan jumlah bioma yang ingin dibentuk. Algoritma Genetika akan terus melakukan evaluasi atas nilai *fitness* tiap *genotype* hingga didapat solusi dengan nilai *fitness* terbaik. Dua syarat algoritma berhenti adalah ketika sudah ditemukan solusi atau proses iterasi sudah dinyatakan maksimal.

Dalam setiap prosesnya, Algoritma Genetika akan melalui proses seleksi untuk menentukan calon induk, proses mutasi untuk membentuk *genotype* baru dengan mengubah satu indeks dalam *genotype* induk, dan juga proses *crossover* untuk membentuk individu baru dengan menyilangkan kedua induk.

3.2.4.4 *Graphical Realization*

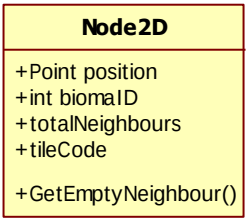
Graphical Realization dilakukan dengan mengubah bentuk struktur data ke dalam objek permainan *virtual*. *Graphical Realization* dimulai dengan memetakan struktur data ke dalam bentuk 2D *tiled-grid* untuk memudahkan proses *rendering*.

Gambar 3.15 menunjukkan contoh pemetaan *genotype* ke dalam bentuk data yang lebih memudahkan proses *rendering*.



Gambar 3.15 Contoh Pemetaan *Genotype* Dalam Bentuk Data yang Memudahkan Proses Rendering

Proses pemetaan struktur data ke dalam bentuk 2D *tiled-grid* dimulai dengan menyiapkan matriks kosong dua dimensi dengan tipe data *Node*2D. *Node*2D merupakan suatu kelas yang akan digunakan untuk mewakili satu *tile* di dalam *tiled-grid*. Struktur kelas dari *Node*2D ditunjukkan pada Gambar 3.16. Kelas *Node*2D digunakan untuk mewakili *tile* karena mampu menyimpan informasi posisi *tile*, jenis bioma, jumlah tetangga, dan mencari tempat kosong untuk diisi dengan tetangga selanjutnya.

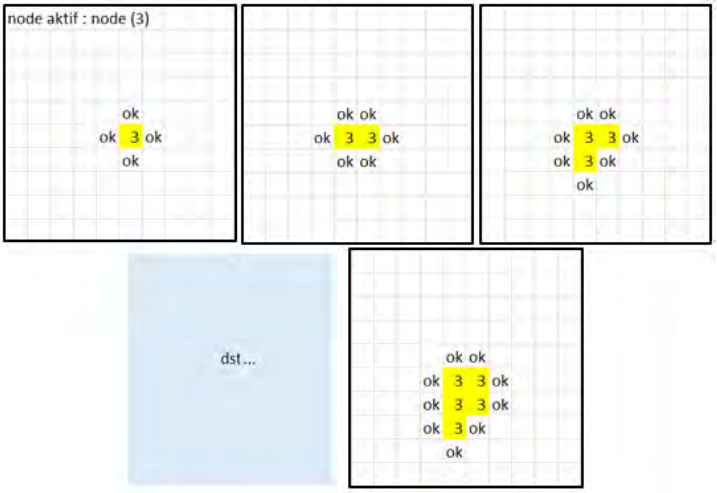


Gambar 3.16 Kelas *Node*2D

Selain menyiapkan matriks kosong, hal lain yang perlu disiapkan adalah deklarasi variabel mengenai jumlah *tile* yang akan dibentuk untuk mewakili satu bioma. Pada Gambar 3.15 dicontohkan jumlah *tile* sebanyak lima. Maka, setiap bioma akan dibentuk dengan menyusun lima *tile* yang berdekatan.

Pemetaan *genotype* ke dalam bentuk 2D *tiled-grid* dimulai dengan memilih salah satu *node* dalam *genotype*. Kemudian dilanjutkan dengan mengunjungi tetangga-tetangganya hingga semua *node* telah dikunjungi. Sebagai contoh *node* awal yang terpilih adalah *node* dengan nomor 3. Untuk masing-masing *node,* proses peletakan *tile* akan berulang sebanyak jumlah *tile* yang ditentukan. Peletakan pertama *tile* berada pada posisi tengah matriks, hal ini bertujuan agar peta permainan yang terbentuk tidak bertabrakan dengan batas matriks.

Setiap *tile* yang telah dipetakan akan menyimpan posisi dan informasi mengenai ketersediaan tetangga-tetangganya. Informasi ketersediaan tetangga-tetangga yang dimilikinya akan digunakan untuk peletakan *tile* selanjutnya.

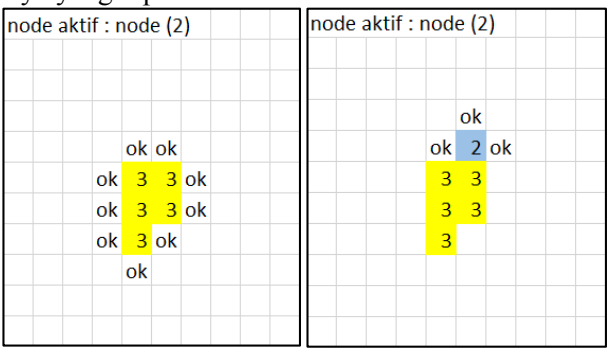


Gambar 3.17 Pemetaan *Node* Pertama pada 2D *tiled-grid*

Gambar 3.17 menunjukkan proses peletakan *tile* untuk *node* (3) dalam 2D *tiled-grid*. *Tile* pertama yang dipetakan ke dalam 2D *tiled-grid* akan ditempatkan di daerah tengah *grid*. Kemudian *tile* tersebut akan melakukan pengecekan terhadap *tile* kosong di sekitarnya untuk ditandai sebagai *tile* yang siap diisi oleh *tile* bernomor 3 selanjutnya. Peletakan *tile* baru pada 2D *tiled-grid*

diletakkan secara acak pada *tile* dengan tanda "ok". Peletakan tersebut akan membuat semua *tile* melakukan pengecekan ulang ketersediaan tetangga-tetangganya. Iterasi untuk bioma pertama akan berakhir setelah dilakukan lima kali peletakan *tile*.

Bioma kedua yang akan dipetakan adalah bioma yang bertetanggaan dengan *node* indeks 3 dan memiliki indeks bioma paling kecil. Pada Gambar 3.15 ditunjukkan bahwa tetangga *node* indeks 3 adalah *node* indeks 2 dan *node* indeks 4 sehingga *node* selanjutnya yang dipilih adalah *node* indeks 2.



Gambar 3.18 Pemetaan *Node* Kedua pada 2D *tiled-grid*

Peletakan *tile* dengan *node* indeks 2 akan mengutamakan ketersediaan tetangga yang bersebelahan secara langsung dengan *tile* indeks 2. Jika ketersediaan tersebut tidak dapat dipenuhi, maka *tile* dengan indeks 2 akan mencari ketersediaan *tile* indeks 3. *Tile* dengan indeks 3 dipilih karena *node* ini terhubung secara langsung dengan dengan *node* indeks 2 dan *node* indeks 3 juga sudah dipetakan ke dalam *tiled-grid*. Setelah *tile* pertama dari *node* indeks 2 diletakkan, maka hanya *tile* kosong yang bersebelahan langsung dengan *node* indeks 2 yang ditandai sebagai *node* "ok". Pemetaan *node* kedua ditunjukkan pada Gambar 3.18. Proses peletakan *tile* dilakukan secara berulang-ulang hingga semua *node* telah selesai dipetakan.

Proses peletakan *node* juga tetap memperhatikan relasi antar *node* dalam *genotype*. Sebagai contoh, jika *node* indeks 1 tidak terhubung dengan *node* indeks 4, peletakan berdekatan antara

node 1 dengan *node* 4 akan dilarang. Pada Gambar 3.19, tanda "x" menyatakan posisi yang terlarang untuk *tile* indeks 4.

Gambar 3.19 Pertimbangan Relasi Keterhubungan

3.2.5 Perancangan Skenario Keseimbangan Permainan

Keseimbangan permainan ditentukan oleh jumlah bobot hewan berbahaya, sumber daya yang tersedia, skor *inventory* pemain, dan status kesehatan pemain. Penentuan keseimbangan permainan dimulai dengan menentukan bobot untuk setiap objek yang mampu muncul (*spawn*) selama permainan. Pemberian bobot untuk setiap objek dalam permainan dijelaskan pada Tabel 3.11.

Keseimbangan permainan ditentukan oleh dua bobot utama yang mempengaruhi permainan. Kedua bobot ini dapat memicu sistem untuk memunculkan objek-objek dengan kriteria masing-masing.

Bobot pertama dinamakan bobot kesulitan. Bobot ini menentukan jenis kesulitan yang akan diberikan ke dalam permainan. Objek penentu kesulitan ditunjukkan dengan dengan bobot negatif pada Tabel 3.11. Sebagai contoh, jika permainan perlu ditambahkan bobot kesulitan sebanyak -2, maka ada kemungkinan bagi sistem untuk memasukkan dua SpiderHound ke dalam permainan.

Tabel 3.11 Bobot Objek Permainan

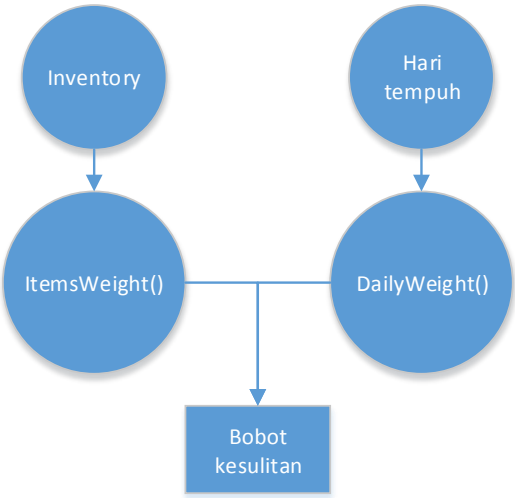
No	Nama	Type	Bobot	Kondisi Prioritas
1	BerryBush	Resources	2	*Health* pemain di bawah 50%
2	Pebble	Resources	1	-
3	Carrot	Resources	2	*Health* pemain di bawah 30%
4	SpiderDen	Animal	-1	Total SpiderDen di area permainan < 4
5	Hound	Animal	-3	*Health* pemain di atas 80%

Bobot kedua dinamakan bobot bantuan. Bobot ini menentukan jenis bantuan yang dapat dimasukkan ke dalam permainan. Objek bantuan ditunjukkan dengan bobot positif pada Tabel 3.11.

Di dalam permainan, pengecekan bobot permainan tidak dilakukan terus menerus. Fungsi Pengecekan bobot dinamakan Fungsi Regulasi. Fungsi Regulasi akan dijalankan sesuai dengan kriteria masing-masing bobot. Fungsi regulasi bobot kesulitan akan dicek setiap waktu menunjukkan sore hari, sedangkan fungsi regulasi bobot bantuan akan dijalankan setiap waktu menunjukkan malam hari.

3.2.5.1 Penghitungan Bobot Kesulitan

Nilai bobot kesulitan didasarkan kepada dua parameter. Parameter pertama adalah hari tempuh. Sedangkan parameter kedua adalah skor pemain yang dihitung dari bobot *item* di dalam *inventory* pemain dalam waktu tertentu. Bobot kesulitan didapat dari penjumlahan nilai parameter pertama dengan parameter kedua. Penghitungan bobot kesulitan ditunjukkan pada Gambar 3.20.



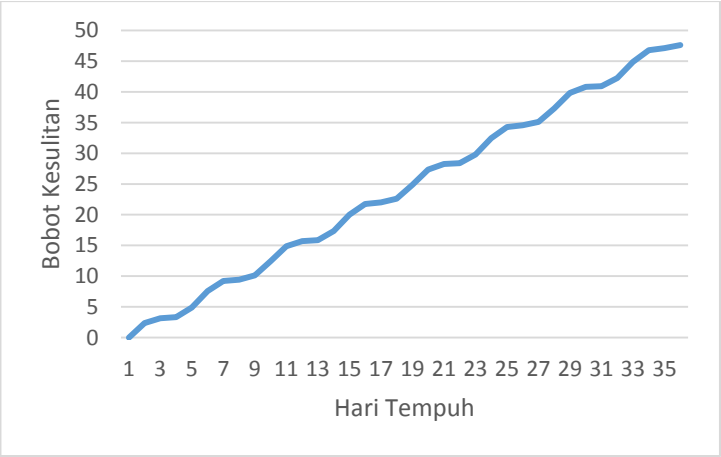
Gambar 3.20 Penghitungan Bobot Kesulitan

Parameter Hari Tempuh sebagai penentu bobot kesulitan akan membuat bobot permainan selalu naik. Hal ini dapat menjaga permainan agar memiliki kesulitan yang meningkat di setiap harinya.

Bobot kesulitan yang selalu meningkat akan membuat pemain kehilangan "ruang bernapas" karena pemain tersebut akan selalu dihadapkan dengan kesulitan yang bertubi-tubi di setiap harinya. Oleh karena itu akan digunakan fungsi sinus untuk perhitungan bobot kenaikan kesulitan. Gambar 3.21 menjelaskan mengenai kenaikan bobot kenaikan berdasarkan hari tempuh dengan menggunakan fungsi sinus pada persamaan 3.1. Fungsi sinus dipilih karena fungsi ini dapat memberikan nilai yang cenderung naik-turun. Konstanta merupakan bilangan positif yang menentukan kenaikan bobot tiap harinya.

(3.1)

$$\boldsymbol{Bobot} = \mathbf{sin}(\boldsymbol{hari}) + \boldsymbol{konstanta}$$



Gambar 3.21 Grafik Kenaikan Bobot Berdasarkan Hari

Parameter bobot *item* pemain bertujuan untuk membuat permainan agar dapat menyesuaikan kondisi pemain. Semakin lengkap *item* yang dimiliki pemain, semakin tinggi parameter bobot *item* pemain. Semakin sedikit *item* yang dimiliki pemain, semakin kecil parameter bobot *item* pemain.

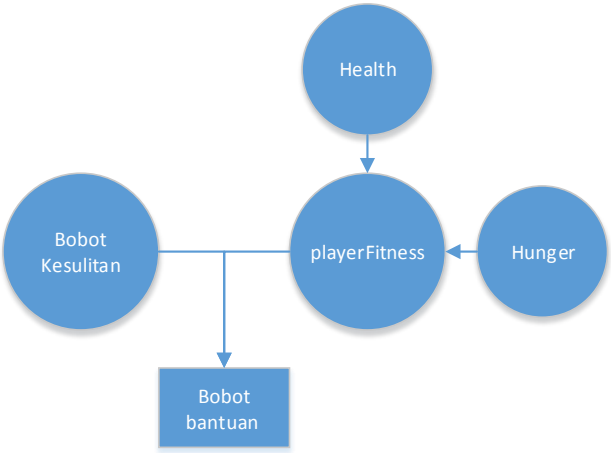
3.2.5.2 Penghitungan Bobot Bantuan

Bobot bantuan merupakan nilai yang dapat memicu sistem untuk mengurangi tingkat kesulitan permainan dengan cara menambahkan sumber daya ke dalam permainan. Objek sumber daya pada permainan ditunjukkan dengan bobot positif.

Nilai bobot bantuan didasarkan pada dua parameter. Parameter pertama adalah bobot kesulitan pada waktu itu. Sedangkan parameter kedua adalah status kesehatan pemain pada waktu itu.

Parameter bobot kesulitan sebagai penentu bantuan bertujuan agar kesulitan dalam permainan dapat ditutupi dengan sumber daya yang disediakan oleh sistem. Parameter status kesehatan pemain sebagai penentu bobot bantuan bertujuan agar sistem

dapat menyesuaikan tingkat bantuan permainan dengan kondisi kesehatan pemain.



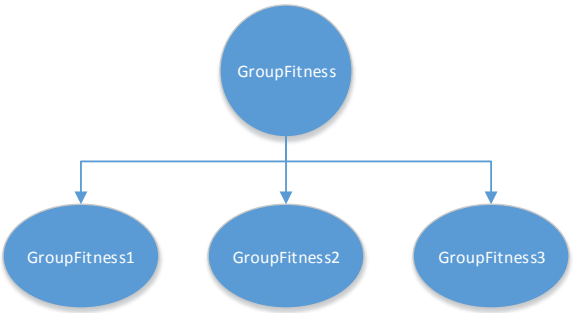
Gambar 3.22 Penghitungan Bobot Bantuan

3.2.5.3 PCG Sebagai Pengatur Keseimbangan

Procedural Content Generation dapat digunakan untuk menentukan objek yang harus ditambahkan ke dalam permainan. Pada tahap persiapan, sistem harus mengelompokkan objek permainan sesuai dengan bobot yang dimilikinya. Kelompok objek ini disebut dengan objek GroupFitness. GroupFitness1 berisi objek dengan bobot 1, GroupFitness2 berisi objek dengan bobot 2, dst. Penjelasan mengenai GroupFitness ditunjukkan pada Gambar 3.23. Masing-masing GroupFitness mampu mengeluarkan satu objek permainan dengan bobot yang sesuai GroupFitness tersebut dan memiliki prioritas paling tinggi untuk dikeluarkan.

Pengatur bobot dikendalikan oleh suatu GameObject. Ada dua jenis GameObject sesuai dengan jenis bobot yang ditangani. GameObject pengatur bobot kesulitan dinamakan MinusController. Sedangkan GameObject pengatur bobot bantuan dinamakan PlusController. Masing-masing GameObject

tersebut dapat mengeluarkan GroupFitness hingga bobot mencapai kebutuhan permainan.

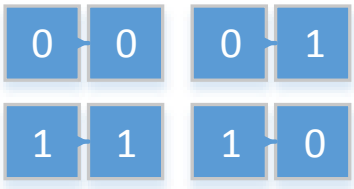


Gambar 3.23 Pengelompokan Bobot Objek

Berikut adalah penjelasan proses PCG sebagai pengatur keseimbangan permainan.

1. *Genotype Representation*

   Untuk mengatur keseimbangan, sistem hanya perlu melakukan *spawning* objek atau tidak. Oleh karena itu, *genotype representation* dari proses ini adalah sebuah *array* dengan isi 1 atau 0 saja. Panjang *array genotype* adalah sebanyak GroupFitness yang dimiliki oleh objek pengatur bobot. Jika pada suatu objek pengatur bobot memiliki GroupFitness1 dan GroupFitness2 saja, maka panjang *genotype* yang akan dibentuk adalah *array* dengan panjang 2. Kemungkinan *genotype* yang terbentuk ditunjukkan pada Gambar 3.24.



Gambar 3.24 Empat Kemungkinan Genotype PCG Keseimbangan dengan Panjang Dua

2. *Fitness Function*
   Pengaturan keseimbangan bobot dilakukan dengan melakukan penghitungan antara nilai *fitness* dengan bobot yang sekarang ingin dibentuk. Dimana untuk skenario kesulitan, bobot yang ingin dibentuk adalah bobot kesulitan. Sedangkan untuk skenario bantuan, bobot yang ingin dibentuk adalah bobot bantuan.
   Batasan fungsi *fitness* dalam permasalahan ini adalah penjumlahan bobot dari objek-objek yang terpilih harus sama dengan bobot yang ingin dicari. Penalti yang diberikan untuk pelanggaran batasan ini senilai selisih bobot yang ingin dicari dengan solusi bobot *genotype*.
3. *Implementasi Algoritma Genetika*
   Implementasi Algoritma Genetika sama dengan implementasi algortima genetika pada proses pembangkit peta dinamis. Algoritma Genetika akan mencari solusi hingga menemukan nilai terbaik atau telah mencapai batas iterasi yang diberikan.
4. *Graphical Realization*
   Solusi yang diberikan oleh Algoritma Genetika berupa genotype dengan struktur data array satu dimensi dengan isi 0 atau 1 saja. Solusi ini harus dipetakan terhadap daftar dari GroupFitness. Karena proses *spawning* objek tidak langsung dilakukan berdasarkan hasil yang muncul, namun ditentukan oleh masing-masing GroupFitness. Secara singkat, sistem hanya memberikan perintah untuk mengeluarkan objek dengan bobot tertentu, semisal 2. Selanjutnya FitnessGroup dengan bobot 2 akan mencari objek yang termasuk ke dalam bobot dua dan memiliki prioritas yang tinggi untuk dimunculkan ke dalam peta permainan.

BAB IV
IMPLEMENTASI

Pada bab ini dijelaskan mengenai implementasi dari perancangan perangkat lunak. Implementasi yang dijelaskan meliputi lingkungan pembangunan perangkat lunak, implementasi Algoritma Genetika implementasi pembangkit peta dinamis, dan implementasi skenario keseimbangan permainan.

4.1 Lingkungan Implementasi

Perangkat keras yang digunakan dalam pembangunan perangkat lunak ini berupa laptop dengan spesifikasi perangkat lunak dan perangkat lunak sebagai berikut:

Spesifikasi perangkat keras:

- Processor Intel® Core™ i3-3217U CPU @1.80GHz;
- Memori 8.00GB; dan
- Kartu *Graph*is NVIDIA GeForce 740M 2GB.

Spesifikasi perangkat lunak:

- Sistem Operasi Windows 8.1 Profesional 64-bit; dan
- Unity 3D v5.0.2f.

4.2 Implementasi Pembangkit Peta Dinamis

Penjelasan mengenai implementasi pembangkit peta dinamis akan dibagi ke dalam tiga bagian sesuai dengan langkah-langkah pembuatan peta dinamis yang telah dibahas di subbab 3.2.4**.**

4.2.1 Implementasi *Genotype Representation*

Genotype yang dibentuk dibagi ke dalam dua bagian, yaitu struktur data dua dimensi berupa *space tree* yang hanya menggambarkan keterhubungan bioma, tanpa ada informasi jenis bioma, dan struktur satu dimensi yang berisi informasi jenis bioma.

4.2.1.1 Pembentukan *Tree* Relasi *Node*

Tree untuk menggambarkan hubungan antar node dapat dibentuk menggunakan *adjacency graph*. Pada Kode Sumber 4.1 dijelaskan proses utama pembentukan *adjacency graph*.

```
public AdjacencyGraph CreateAdjacencyList(int length, int
maxneigh)
{
    graphLength = length;
    maxNeighbours = maxneigh;
    do {
         BuildadjListGraph ();
    } while (!IsAllConnected());

    return this;
}

void BuildadjListGraph()
{
    adjListGraph = new List<int>[graphLength];

    for (int i=0; i < graphLength; i++) {
        adjListGraph[i] = new List<int>();
    }

    for(int selected = 0; selected < graphLength; selected++) {
        if(adjListGraph[selected].Count < 3) {
            int neighbour;
            int loop = 0;
            do {
            neighbour = UnityEngine.Random.Range(0,
            graphLength);


       }      while      (neighbour      ==      selected      ||
       adjListGraph[neighbour].Count    >=    maxNeighbours    ||
       adjListGraph[neighbour].Contains(selected)              ||
       adjListGraph[selected].Contains(neighbour));

       adjListGraph[selected].Add(neighbour);
       adjListGraph[neighbour].Add(selected);
    }
}
```

Kode Sumber 4.1 Fungsi Pembuatan Adjacency Graph

Pengecekan semua *node* telah terhubung atau tidak di dalam satu *graph* yang sama dilakukan dengan menerapkan implementasi algortima traversal melalui fungsi IsAllConnected(). Algoritma traversal adalah algoritma di dalam *graph* yang digunakan untuk mengunjungi setiap *node* di dalam *graph*. Jika pada suatu pengecekan, jumlah yang dikunjungi sama dengan jumlah *node* total seharusnya, maka dapat disimpulkan bahwa semua *node* telah terhubung. Implementasi algoritma traversal ditunjukkan pada Kode Sumber 4.2.

```
bool IsAllConnected()
{
    if (adjListGraph == null)
        return false;

    int totalConnect = 0;

    Queue<int> q = new Queue<int> ();
    bool[] visited = new bool[graphLength];

    q.Enqueue (0);
    while (q.Count > 0) {
        int top = q.Dequeue();

        if(!visited[top]) {
            visited[top] = true;
            totalConnect++;
        }

        foreach(var node in adjListGraph[top]) {
        if(!visited[node]) {
            q.Enqueue(node);
        }
    }

    return totalConnect == graphLength ? true : false;
}
```

Kode Sumber 4.2 Implementasi Algoritma Traversal

4.2.1.2 Pembentukan *List* Informasi Bioma

List Informasi Bioma merupakan struktur data yang akan terus berubah untuk disesuaikan dengan *tree* relasi *node*. List Informasi bioma dibentuk menggunakan sebuah tipe data *string*,

di mana hanya berisi angka-angka yang menunjukkan indeks bioma.

Kode Sumber 4.3 menunjukkan proses pembentukan list. Tiga parameter penting dalam proses ini adalah allBiomes, Chromosome, dan chromosomeLength. allBiomes merupakan sebuah list yang berisi daftar mengenai semua bioma yang tersedia. ChromosomeLength menunjukkan banyaknya bioma yang diinginkan dalam peta permainan. Sedangakan Chromosome merupakan *list* yang berisi informasi bioma tersebut.

```
string Chromosome = “”;

public override void RandomIndividu()
{
    for (int i = 0; i < chromosomeLength; i++)
    {
        Chromosome += UnityEngine.Random.Range(0,
        allBiomes.Count).ToString();
     }
}
```

Kode Sumber 4.3 Pembentukan List Informasi Bioma

4.2.2 Implementasi Fungsi Fitness

Fungsi fitness merupakan sebuah fungsi yang digunakan untuk menilai suatu *genotype* dinyatakan baik atau tidak. Implementasi fungsi fitness ditunjukkan pada Kode Sumber 4.4. Pada fungsi tersebut sistem akan mengecek nilai batasan-batasan yang harus dipenuhi oleh suatu genotype. Batasan tersebut di antaranya dinyatakan dalam sameBiomaPenalty, totalSwamp, notusedBioma, dan juga result.

```
public override int CalculateFitness()
{
    List<int> weight = ChromosomeToList ();
    GameManager gameManager =
    GameObject.Find("GameManager").GetComponent<GameManager>();

    List<Transform> tilePrefabs = gameManager.tilePrefabs;
    int[] usedBioma = new int[varianChromosome];

    int result = 0;
    int sameBiomaPenalty = 0;
    int totalSwamp = 0;
    for(int node = 0; node < AdjacencyGraph.Instance.adjListGraph.Length; node++) {
```

```
        int sum = tilePrefabs[ weight[node] ].GetComponent<Tile>().weight;

        usedBioma[ weight[node] ] = 1;

         if(allBiomes[ weight[node] ].name == "Swamp")
        totalSwamp++;

        foreach(var neighbour in AdjacencyGraph.Instance.adjListGraph[node]) {
            sum += tilePrefabs[ weight[neighbour] ].GetComponent<Tile>().weight;
            usedBioma[ weight[neighbour] ] = 1;
        }

        if(Math.Abs(sum) > result) result = Math.Abs(sum);
    }
    int notusedBioma = 0;
    foreach (var i in usedBioma) {
        if(i == 0)
        notusedBioma += 3;

    int totalSwampPenalty = 0;
        if (totalSwamp > 1)
             totalSwampPenalty += 10;


    return Mathf.RoundToInt((result + notusedBioma + sameBiomaPenalty +
    totalSwampPenalty));
}
```

Kode Sumber 4.4 Fungsi Penghitungan Fitness

4.2.3 Implementasi Algoritma Genetika

Algortima Genetika merupakan algoritma yang digunakan untuk mencari *genotype* yang paling baik. Proses utama Algoritma Genetika ditunjukkan pada Kode Sumber 4.5.

Dalam Algoritma Genetika, proses pertama adalah pembentukan populasi, kemudian selama solusi belum ditemukan, proses seleksi, mutasi, dan crossover akan terus dilakukan untuk membentuk *genotype* yang baru. Pada Unity, fungsi Update merupakan fungsi yang akan dijalankan terus menerus, sehigga perlu adanya pemberhentian ketika solusi telah ditemukan.

```
void Update()
{
       if(population.Count == 0)
          InitPopulation();

       solution = population.Values [0];

       if ((solution.Fitness < fitnessThreshold ||
```

```
solution.Fitness > fitnessThreshold) && (!limitGeneration ||
generation < maxGeneration)) {

          Tuple<int, int> parents = BiasedRoulleteWheel ();

          Mutation (parents.Item1);
          CrossOver (parents);

          generation++;
          ready = false;

       } else {
          GetComponent<GeneticAlgorithm>().enabled = false;
          ready = true;
       }
}
```

Kode Sumber 4.5 Proses Utama Algoritma Genetika

4.2.3.1 Implementasi Roullette Wheel

Roulette Wheel merupakan fungsi seleksi dalam Algoritma Genetika di mana *genotype* dengan nilai fitness yang paling baik akan memiliki persentase terpilih yang lebih besar. Implementasi Biased Roulette Wheel ditunjukkan pada Kode Sumber 4.6.

```
Tuple<int, int> BiasedRoulleteWheel ()
{
    int index1;
    int index2;

    do {
       index1 = Random.Range (0, population.Count);
       index2 = Random.Range (0, index1);
    } while(index1 == index2);

    return new Tuple<int, int> (index1, index2);
}
```

Kode Sumber 4.6 Fungsi Seleksi Menggunakan Roulette Wheel

4.2.3.2 Implementasi Fungsi Mutasi

Fungsi mutasi merupakan fungsi untuk mengganti satu indeks dalam satu induks yang terseleksi untuk membentuk *genotype* baru. Implementasi fungsi mutasi ditunjukkan pada Kode Sumber 4.7.

```
void Mutation(int parentIndex)
{
    float randomNumber = Random.Range (0.0f, 1.0f);
    Individual parent = population.Values [parentIndex];

    if (randomNumber < mutationRate)
    {
        int position = Random.Range(0,
        population.Values[parentIndex].Chromosome.Length);

        Individual tmp = parent.MutateGen(position);

        if(Mathf.Abs(fitnessThreshold - tmp.Fitness) <
        Mathf.Abs(fitnessThreshold - parent.Fitness)) {
            population.RemoveAt(parentIndex);
            population.Add( Mathf.Abs(fitnessThreshold –
            tmp.Fitness), tmp);
        }
    }
}
```

Kode Sumber 4.7 Fungsi Mutasi pada Algoritma Genetika

Fungsi mutasi yang ada pada Algoritma Genetika hanya bersifat umum. Proses ini hanya menentukan di posisi mana indeks dari *genotype* tersebut akan diubah, kemudian dilakukan perbandingan antara *genotype* baru yang terbentuk dengan induknya untuk memilih mana di antara *genotype* tersebut yang dapat bertahan dalam populasi. Implementasi detail mengenai proses mutasi terdapat dalam induk, seperti pada Kode Sumber 4.8.

```
public override Individual MutateGen(int position)
{
   IndividualPlot tmp = IndividualPlot.DeepCopy (this);

   int newval = Math.Abs(System.Convert.ToInt32(tmp.Chromosome
   [position].ToString())-1);

   tmp.Chromosome = tmp.Chromosome.Remove(position,
   1).Insert(position, newval.ToString());

   return tmp;
}
```

Kode Sumber 4.8 Implementasi Detail mengenai Proses Mutasi

4.2.3.3 Fungsi Crossover

Fungsi *crossover* merupakan fungsi yang bertujuan untuk menghasilkan *genotype* baru dengan cara menyilangkan kedua parent. Implementasi *Crossover* pada Algoritma Gentika ditunjukkan pada Kode Sumber 4.9.

```
void CrossOver(Tuple<int, int> parentIndexes)
{
    float randomNumber = Random.Range (0.0f, 1.0f);
    Individual parent = population.Values
    [parentIndexes.Item1];

    if (randomNumber < crossoverRate)
    {
        int position = Random.Range(0,
        population.Values[parentIndexes.Item1].
        Chromosome.Length) + 1;

        Individual tmp =
population.Values[parentIndexes.Item1].SwapWith(population.Values[
parentIndexes.Item2], position);

        if(Mathf.Abs(fitnessThreshold - tmp.Fitness) <
Mathf.Abs(fitnessThreshold - parent.Fitness)) {

            population.RemoveAt(parentIndexes.Item1);
            population.Add( Mathf.Abs(fitnessThreshold -
tmp.Fitness) , tmp);
        }
    }
}
```

Kode Sumber 4.9 Crossover pada Algoritma Genetika

Fungsi Crossover pada Algoritma Genetika hanya bersifat umum, sedangkan implementasi detail mengenai fungsi mutasi ditunjukkan pada Kode Sumber 4.10 melalui fungsi SwapWith yang ada pada induk. Pada fungsi tersebut ditunjukkan bahwa fungsi *crossover* akan mengambil sebagian *genotype* dari satu induk untuk menganti sebagian *genotype* pada induk yang lain, sehingga dihasilkan individu baru.

```
public override Individual SwapWith(Individual other, int
```

```
lastposition)
{
    IndividualPlot tmp = IndividualPlot.DeepCopy (this);

    int startposition = UnityEngine.Random.Range (0,
    lastposition);

    string injection = other.Chromosome.Substring
    (startposition, lastposition-startposition);

    tmp.Chromosome = tmp.Chromosome.Remove (startposition,
    lastposition - startposition);
    tmp.Chromosome = tmp.Chromosome.Insert
    (UnityEngine.Random.Range(0, tmp.Chromosome.Length),
    injection);

    return tmp;
}
```

Kode Sumber 4.10 Implementasi Detail Fungsi Crossover

4.2.4 Implementasi *Graphical Realization*

```
public void GenerateExpanded2D(int biomaTotal, Point size, int mintile,
int maxtile)
{
    Width = size.x;
    Height = size.y;

    // BFS
    Queue<int> q = new Queue<int> ();
    bool[] visited = new bool[biomaTotal];

    Point position = new Point (width / 2, height / 2);

    q.Enqueue (Random.Range(0, biomaTotal));

     while (q.Count > 0) {
        int top = q.Dequeue();

        if (!visited[top]) {
            visited[top] = true;
            int n = Random.Range(mintile, maxtile);
            ExpandOneNode(top, top, position, n);
        }

        foreach (var neighbour in adjList[top])
        {
            if (!visited[neighbour])
            {
```

```
                visited[neighbour] = true;

                position = GetNextPosition(top, neighbour);
                int n = Random.Range(mintile, maxtile);
                ExpandOneNode(top, neighbour, position, n);
                visited[neighbour] = true;
                q.Enqueue(neighbour);
            }
        }
    }
}
```

Kode Sumber 4.11 Mapping Adjacency Graph ke Matriks Dua Dimensi

Pembentukan matriks dua dimensi memanfaatkan fungsi GenerateExpanded2D() yang ada dalam kelas GraphExpander. Fungsi tersebut akan mengunjungi setiap *node* dalam *space tree* untuk mencari penempatan *node* tersebut di dalam matriks dua dimensi. Setelah semua *node* telah d*item*patkan di dalam matriks dua dimensi, maka langkah selanjutnya adalah melakukan instansiasi objek *tile* permainan berdasarkan matriks tersebut. Implementasi instansiasi objek *tile* permaian tercantum pada Kode Sumber 4.12.

```
Void CreateMap() {
    Point anchor = GraphExpander.Instance.GetAnchorOneBioma(i);
    Node2D[,] mapToRender = GraphExpander.Instance.GetOneBiomaGraph(i);

    for (int x=0; x < mapToRender.GetLength(0); x++) {
      for (int y=0; y < mapToRender.GetLength(1); y++) {
         if(mapToRender[x,y] != null)
         {
            int index = (x+ (y*mapToRender.GetLength(0)));
            Vector3 pos = new Vector3((anchor.y *GameManager.Instance.
    tileSize) + (y*GameManager.Instance.tileSize), 0,
    (anchor.x*GameManager.Instance.tileSize) +
    (x*GameManager.Instance.tileSize));

    Transform   tile   =   (Transform)Instantiate(tileMapPrefab,   pos,
    Quaternion.identity);

     }
}
```

Kode Sumber 4.12 Instansiasi Objek *Tile*

4.3 Implementasi Skenario Keseimbangan Permainan

Implementasi skenario keseimbangan dibagi ke dalam dua pembahasan, yaitu skenario bobot kesulitan dan skenario bobot bantuan.

4.3.1 Skenario Bobot Kesulitan

Bobot kesulitan permainan ditentukan berdasarkan jumlah hari tempuh dan skor pemain dalam waktu tertentu. Implementasi penghitungan bobot kesulitan tercantum pada Kode Sumber 4.13.

```
int BadLevel ()
{
   int dayFitness =
   Mathf.CeilToInt(((float)GameManager.Instance.timeManager.CurrentDay /
   (float)GameManager.Instance.timeManager.TotalDay) * 50f);

   dayFitness = Mathf.CeilToInt(Mathf.Sin(dayFitness) + dayFitness);

   int playerFitness =
   (int)GameManager.Instance.player.GetComponent<Player>
   ().playerInvenPoint;

   int res = dayFitness + playerFitness;
   if (res > 100)
       res = 100;

   return res;
}
```

Kode Sumber 4.13 Penentuan Bobot Kesulitan

Fungsi BadLevel() mengambil nilai perhitungan hari dengan menggunakan kenaikan hari sebagai parameter pertama. Kemudian parameter kedua diambil dari skor *inventory* pemain.

Selanjutnya, hasil keluaran dari fungsi ini akan menjadi nilai evaluasi pada Algoritma Genetika untuk menentukan bobot kesulitan minimal permainan pada hari tersebut.

4.3.2 Skenario Bobot Bantuan

Bobot bantuan ditentukan berdasarkan bobot kesulitan dan status pemain dalam waktu tertentu. Implementasi penghitungan bobot bantuan tercantum pada Kode Sumber 4.14.

```
int GoodLevel ()
{
    int difficultyFitness =
Mathf.RoundToInt(GameManager.Instance.minusController.GetComponent<GAMinus>
().RealFitness);

    int playerFitness = 100 -
Mathf.RoundToInt(GameManager.Instance.player.GetComponent<Player>().playerH
ealthyPoint);

    return difficultyFitness + playerFitness;
}
```

Kode Sumber 4.14 Penentuan Bobot Bantuan

Fungsi GoodLevel() mengambil bobot kesulitan yang secara langsung ada dalam peta permainan melalui variabel RealFitness yang ada dalam kelas GAMinus. Sementara pengambilan status kesehatan pemain dilakukan dengan melihat variabel playerHealthyPoint yang telah menjadi atribut pemain.

4.3.3 Penentuan Skenario Keseimbangan dengan SBPCG

Proses pencarian solusi untuk mengatur keseimbangan melalui proses yang hampir sama ketika melakukan pembangkit peta dinamis. Pembeda antar SBPCG adalah bagaimana suatu *genotype* direpresentasikan, dievaluasi, dan direalisasikan dalam bentuk objek permainan nyata.

4.3.3.1 *Genotype Representation*

Dalam penentuan keseimbangan, *genotype* hanya merepresentasikan sebuah aksi terhadap indeks dalam *list* untuk ditampilkan atau tidak. Maka implementasi *genotype* dibentuk dengan mengunakan *list* satu dimensi dengan kemungkinan isi hnaya sebatas 0 atau 1. Implementasi *genotype representation* tersebut ditunjukkan pada Kode Sumber 4.15.

```
public override void RandomChromosome() {
    for (int i = 0; i < chromosomeLength; i++) {
       Chromosome += UnityEngine.Random.Range(0, 2).ToString();
    }
}
```

Kode Sumber 4.15 Fungsi Pembentukan *Genotype*

4.3.3.2 Implementasi Fungsi Fitness

Pada fungsi fitness, hal yang perlu dievaluasi adalah tingkat prioritas suatu objek untuk masuk ke dalam peta perminan. Implementasi fungsi fitness ditunjukkan pada Kode Sumber 4.16. Pada implementasi fungsi fitness, parameter yang perlu diketahui adalah daftar objek yang tersedia yang disimpan dalam objAvailable.

```
public override int CalculateFitness()
{
   List<FitnessGroup> objAvailable =
   GameManager.Instance.plusController.objAvailable;
   float result = 0;

   for (int i=0; i<chromosomeLength; i++) {
      int realIndex = i % objAvailable.Count;

      if(ChromosomeToList()[i] == 1) {
      FitnessGroup fg =
      objAvailable[realIndex].GetComponent<FitnessGroup>();

      if(fg.GetObjWithPriority(true) != null &&
      fg.GetObjWithPriority(true).GetComponent<HasPriorities>().enabled)
      {
          if(fg.GetObjWithPriority(true).GetComponent<HasPriorities>().
          CalculateAllPriorities() > 0) {

            result += objAvailable[realIndex].weight;
          }
          else  result += 100;
      }
      else result += 100;
      }
   }
   return Mathf.Abs(Mathf.RoundToInt(result));
}
```

Kode Sumber 4.16 Penghitungan Fungsi Fitness untuk Skenario Keseimbangan

ObjAvailable berisi grup-grup objek yang telah dikelompokkan berdasarkan bobot yang dimiliki objek tersebut. Implementasi inisialisasi ObjAvailable ditunjukkan pada Gambar 4.1.

Gambar 4.1 Inisialisasi ObjAvailable

4.3.3.3 Implementasi *Graphical Realization*

Graphical Realization mengambil solusi atas *genotype* terbaik yang dapat dibentuk. Dengan melakukan mapping solusi ke dalam daftar objAvailable, fungsi Spawn() mengeluarkan objek dengan solusi yang bernilai 1. Implementasi *spawning* objek ditunjukkan pada Kode Sumber 4.17.

```
if(GetComponent<GeneticAlgorithm>().solution != null &&
GetComponent<GeneticAlgorithm>().isReady)
{
    List<int> toSummon =
    GetComponent<GeneticAlgorithm>().solution.
    ChromosomeToList();

    /* Instantiate Monsters */
    for(int i = 0; i < toSummon.Count; i++)
    {
        if(toSummon[i] == 1)
        {
            int realIndex = i % objAvailable.Count;

            /* Di sini instansiasi FitnessGroup nya */

            Instantiate(objAvailable[realIndex], Vector3.zero,
            Quaternion.identity);

        }
    }
}
```

Kode Sumber 4.17 Fungsi Spawn Objek

BAB V
PENGUJIAN DAN EVALUASI

Pada bab ini akan dijelaskan mengenai rangkaian uji coba dan evaluasi yang dilakukan. Pengujian yang dilakukan adalah pengujian terhadap kebutuhan fungsionalitas. Proses pengujian dilakukan menggunakan metode *blackbox* berdasarkan skenario yang telah ditentukan.

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 adalah sebagai berikut:

Spesifikasi perangkat keras:

- Processor Intel® Core™ i3-3217U CPU @1.80GHz;
- Memori 8.00GB; dan
- Kartu *Graph*is NVIDIA GeForce 740M 2GB.

Spesifikasi perangkat lunak:

- Sistem Operasi Windows 8.1 Profesional 64-bit; dan
- Unity 3D versi 5.

5.2 Skenario Pengujian

Pada bagian ini akan dijelaskan tentang skenario pengujian yang dilakukan. Pengujian yang dilakukan adalah pengujian kebutuhan fungsionalitas. Pengujian fungsionalitas menggunakan metode (*black box*). Metode ini menekankan pada kesesuaian hasil keluaran sistem.

5.2.1 Pengujian Pembangkitan Peta Dinamis

Pengujian fitur pembangkitan peta dinamis bertujuan untuk melihat hasil keluaran peta dinamis yang dihasilkan oleh modul pembangkit peta dinamis. Pengujian peta dinamis akan dibagi ke dalam beberapa bahasan yaitu pengujian bentuk peta dinamis,

pengujian keseimbangan relasi peta, pengujian batasan bioma swamp, dan pengujian pemakaian semua bioma tersedia.

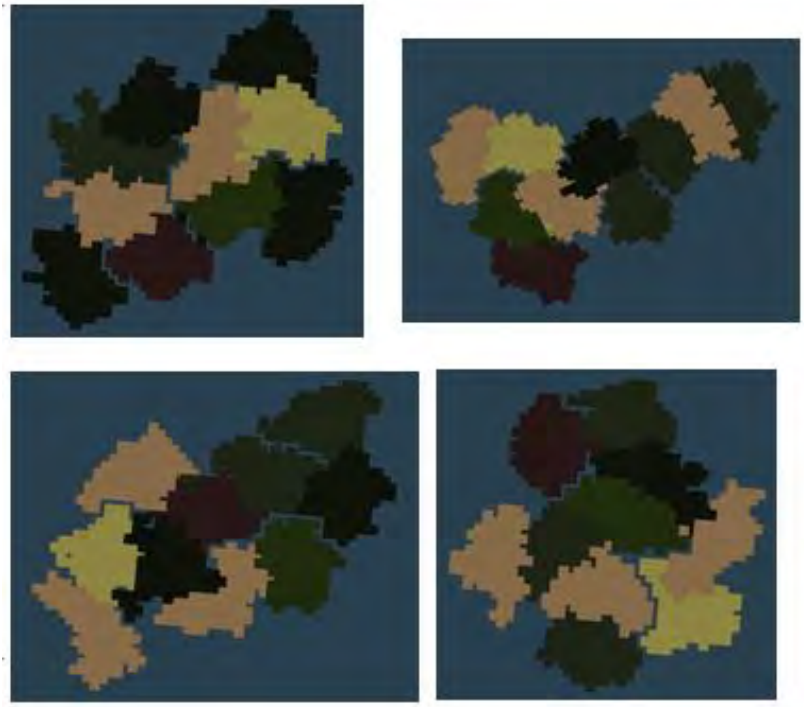
5.2.1.1 Pengujian Bentuk Peta Dinamis

Pengujian benuk peta dinamis bertujuan untuk melihat apakah peta yang dibangkitkan setiap kali permainan dimulai dari awal memiliki bentuk yang berbeda-beda. Skenario pengujian bentuk peta dinamis dijelaskan pada Tabel 5.1.

Tabel 5.1 Skenario Pengujian Pembangkit Peta Dinamis

Nama Skenario	Pengujian pembangkitan peta dinamis
Kode	UF-0001
Tujuan Pengujian	Mendeteksi apakah sistem mampu membangkitkan peta secara dinamis
Kondisi Awal	Pemain telah berada pada Menu Utama
Data Input	Daftar bioma tersedia, jumlah bioma yang ingin dibentuk, dan ukuran *tile*.
Prosedur Pengujian	1. Menekan tombol Play 2. Melihat bentuk peta dinamis yang terbentuk 3. Ulangi langkah (1) dan (2) minimal satu kali lagi 4. Bandingkan bentuk hasil peta dinamis yang telah terbentuk
Hasil yang Diharapkan	Sistem dapat membangkitkan peta permainan dengan bentuk yang bervariasi
Hasil Pengujian	Berhasil
Hasil yang Diperoleh	Sistem membangkitkan peta permainan dengan bentuk yang bervariasi

Gambar 5.1 menunjukkan keluaran yang dihasilkan dari pengujian pembangkit peta secara dinamis. Terlihat bahwa tidak ada peta yang memiliki bentuk yang sama. Hal ini menunjukkan bahwa sistem telah berhasil membangkitkan peta secara dinamis.



Gambar 5.1 Hasil Pembangkitan Peta dinamis

5.2.1.2 Pengujian Keseimbangan Relasi Peta

Pengujian keseimbangan relasi peta bertujuan untuk melihat apakah sistem telah mampu membangkitkan peta dengan relasi antar bioma yang seimbang. Yang dimaksud dengan seimbang adalah penjumlahan bobot dalam satu relasi index dalam *adjacency graph* sama dengan atau mendekati nol. Hasil yang diharapkan dari pengujian ini adalah bioma dengan bobot negatif akan cenderung bertetangga dengan bioma bobot positif. Skenario pengujian keseimbangan relasi peta ditunjukkan pada Tabel 5.2.

Tabel 5.2 Skenario Pengujian Keseimbangan Relasi Peta

Nama Skenario	Pengujian keseimbangan relasi peta
Kode	UF-0002
Tujuan Pengujian	Mendeteksi apakah sistem mampu membangkitkan peta dengan kombinasi relasi yang seimbang
Kondisi Awal	Pemain telah berada pada Menu Utama dalam mode *debugging* menggunakan Unity
Data Input	Daftar bioma tersedia, jumlah bioma yang ingin dibentuk, dan ukuran *tile*
Prosedur Pengujian	1. Menekan tombol Play 2. Melihat jendela *console* pada Unity 3. Mencatat hasil adjacency graph yang dihasilkan 4. Mencatat pembagian bioma 5. Menjumlahkan bobot tiap bioma dalam satu relasi (index) dan disimpan sebagai bobot relasi 6. Mengambil bobot relasi terbesar
Hasil yang Diharapkan	Sistem dapat membangkitkan peta dengan maksimal bobot relasi tidak jauh dari angka nol
Hasil Pengujian	Berhasil
Hasil yang Diperoleh	Sistem dapat membangkitkan peta dengan maksimal bobot relasi tidak jauh dari angka nol

Pengujian dimulai dengan mendefinisikan beberapa bioma dengan dengan bobot masing-masing. Spesifikasi bobot bioma yang akan dipakai dalam pengujian keseimbangan relasi peta tertera pada Tabel 5.3. Selanjutnya, hasil pengujian pada aplikasi dijelaskan melalui Tabel 5.4.

Tabel 5.3 Spesifikasi Bobot Bioma Pengujian

ID	Nama	Bobot
0	Bioma A	-2
1	Bioma B	1
2	Bioma C	-1
3	Bioma D	2
4	Bioma E	2
5	Bioma F	2

Tabel 5.4 Hasil Pengujian Keseimbangan Peta

No	Adjacency Graph			Relasi Bioma	Bobot Relasi	Abs(Maks)
	Index	Bioma	Tetangga			
1	0	D	3, 1	D+C+B	2	2
	1	B	0, 5	B+D+C	2	
	2	A	5, 3, 4	A+C+C+E	-2	
	3	C	0, 2, 6	C+D+A+F	1	
	4	E	2	E+A	0	
	5	C	2, 1	C+A+D	-1	
	6	F	3	F+C	1	
2	0	E	4, 2	E+C+A	-1	1
	1	B	2	B+A	-1	
	2	A	1, 0	A+B+E	1	
	3	A	5, 4, 6	A+F+C+D	1	
	4	C	0, 3, 5	C+E+A+F	1	
	5	F	3, 4	F+A+C	-1	
	6	D	3	D+A	0	
3	0	B	2, 3, 5	B+A+E+C	0	3

No	Adjacency Graph			Relasi Bioma	Bobot Relasi	Abs(Maks)
	Index	Bioma	Tetangga			
	1	F	5, 4	F+C+D	3	
	2	A	0, 4	A+B+D	1	
	3	E	0	E+B	3	
	4	D	2, 1	D+A+F	2	
	5	C	1, 0	C+F+B	2	
Rata-rata						2

Dari hasil pengujian keseimbangan peta yang telah dilakukan, didapatkan nilai dua. Hasil ini dirasa cukup baik karena hasil tidak jauh dari nilai nol.

5.2.1.3 Pengujian Pembatasan Jumlah Bioma Swamp

Pengujian pembatasan jumlah Bioma Swamp bertujuan untuk melihat apakah sistem mampu membangkitkan peta permainan dengan satu bioma swamp di dalam peta permainan. Skenario pengujian jumlah bioma swamp dijelaskan pada Tabel 5.5.

Tabel 5.5 Skenario Pengujian Pembatasan Jumlah Bioma Swamp

Nama Skenario	Pengujian pembatasan jumlah Bioma Swamp
Kode	UF-0003
Tujuan Pengujian	Mendeteksi apakah sistem dapat membangkitkan hanya satu Bioma Swamp dalam peta permainan
Kondisi Awal	Pemain telah berada pada Menu Utama dalam mode *debugging* menggunakan Unity
Data Input	Daftar bioma tersedia, jumlah bioma yang ingin dibentuk, dan ukuran *tile*

Prosedur Pengujian	1. Menekan tombol Play 2. Melihat hirarki pada Unity 3. Menghitung jumlah Swamp yang dihasilkan 4. Ulangi langkah (1) sampai (3) hingga beberapa kali percobaan
Hasil yang Diharapkan	Sistem hanya menghasilkan satu Bioma Swamp dalam peta permainan
Hasil Pengujian	Berhasil
Hasil yang Diperoleh	Sistem hanya membangkitkan satu Bioma Swamp dalam peta permainan

Tabel 5.6 Bioma Tersedia

ID	Nama Bioma
0	Desert
1	Forest
2	Graveyard
3	Greenland
4	Savana
5	Swamp

Gambar 5.2 Pengujian Pembatasan Jumlah Bioma

Tabel 5.7 Hasil Pengujian Jumlah Bioma

No Pengujian	Jumlah Bioma (ID Bioma)					
	0	**1**	**2**	**3**	**4**	**5**
1	3	3	2	1	1	1
2	3	1	2	1	2	1
3	3	2	1	1	2	1
4	2	3	2	1	1	1

Dalam empat kali percobaan pembangkitan peta dihasilkan empat peta permainan dengan jumlah Bioma Swamp hanya satu. Hal ini menunjukkan bahwa pengujian pembatasan jumlah bioma swamp telah berhasil.

5.2.1.4 Pengujian Pemakaian Semua Bioma Yang Tersedia

Pengujian pemakaian semua bioma yang tersedia bertujuan untuk mengecek apakah sistem dapat menghasilkan peta permainan dengan menggunakan semua bioma yang tersedia. Di dalam pengujian, terdapat enam macam bioma yang tersedia seperti tertera pada Tabel 5.6. Skenario pengujian pemakaian semua bioma yang tersedia dijelaskan pada Tabel 5.8.

Tabel 5.8 Spesifikasi Pengujian Pemakaian Semua Bioma yang Tersedia

Nama Skenario	Pengujian pemakaian semua bioma tersedia
Kode	UF-0004
Tujuan Pengujian	Mendeteksi apakah sistem mampu menghasilkan peta permainan dengan menggunakan semua bioma yang tersedia
Kondisi Awal	Pemain telah berada pada Menu Utama dalam mode *debugging* menggunakan Unity
Data Input	Daftar bioma tersedia, jumlah bioma yang ingin dibentuk, dan ukuran *tile*

Prosedur Pengujian	1. Menekan tombol Play 2. Menghitung jumlah Swamp yang dihasilkan 3. Ulangi langkah (1) sampai (3) hingga beberapa kali percobaan
Hasil yang Diharapkan	Sistem membangkitkan peta permainan dengan menggunakan semua bioma tersedia
Hasil Pengujian	Berhasil
Hasil yang Diperoleh	Sistem membangkitkan peta permainan dengan menggunakan semua bioma tersedia

Hasil pengujian pemakaian semua bioma tersedia diwakili oleh Tabel 5.7. Dari tabel tersebut dapat dilihat jika minimal jumlah bioma setiap percobaan adalah satu.

5.2.2 Pengujian Keseimbangan Permainan

Pengaturan bobot kesulitan hanya akan dilakukan setiap hari regulasi dimulai, yaitu pada hari genap permainan. Pengujian keseimbangan permainan akan dibagi ke dalam beberapa bahasan, yaitu pengujian bobot kesulitan, dan pengujian bobot bantuan.

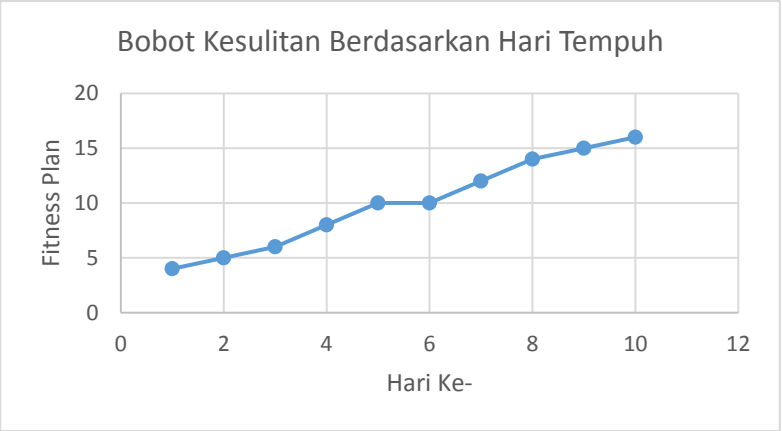
5.2.2.1 Pengujian Bobot Kesulitan Berdasarkan Hari

Pengujian bobot kesulitan berdasarkan hari bertujuan untuk mendeteksi apakah sistem dapat mengatur bobot kesulitan berdasarkan hari tempuh pemain. Skenario pengujian bobot kesulitan berdasarkan hari dijelaskan pada Tabel 5.9.

Tabel 5.9 Skenario Pengujian Bobot Kesulitan Berdasarkan Hari

Nama Skenario	Pengujian bobot kesulitan berdasarkan hari
Kode	UF-0005
Tujuan Pengujian	Mendeteksi apakah sistem mampu menentukan bobot kesulitan berdasarkan hari tempuh
Kondisi Awal	Berada pada posisi debugging Unity Editor

	dengan Scene Gameplay yang terbuka
Data Input	Hari tempuh
Prosedur Pengujian	1. Menekan tombol Play pada Unity Editor 2. Ubah parameter currentDay pada TimeManager 3. Tunggu hingga hari regulasi keseimbangan 4. Catat bobot kesulitan yang berhasil dihitung
Hasil yang Diharapkan	Sistem dapat mangatur bobot kesulitan berdasarkan perubahan hari
Hasil Pengujian	Berhasil
Hasil yang Diperoleh	Sistem dapat mangatur bobot kesulitan berdasarkan perubahan hari



Gambar 5.3 Grafik Pengujian Bobot Kesulitan Berdasarkan Hari

Pada Gambar 5.3 menunjukkan bobot kesulitan perhari akan selalu meningkat, pengujian dilakukan pada hari ke-1, hari ke-2, hari ke-3, hari ke-4 hingga hari ke-10.

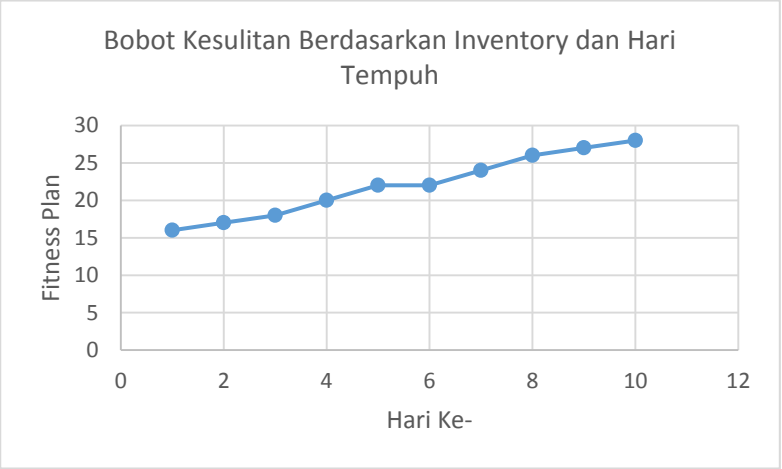
5.2.2.2 Pengujian Bobot Kesulitan Berdasarkan Skor *Inventory* Pemain

Pengujian bobot kesulitan berdasarkan skor bertujuan untuk mendeteksi apakah sistem dapat mengatur bobot kesulitan berdasarkan skor yang didapat pemain dalam waktu tertentu. Skenario pengujian bobot kesulitan berdasarkan hari dijelaskan pada Tabel 5.10. Pada Gambar 5.4 menunjukkan bobot kesulitan berdasarkan skor pemain akan selalu meningkat, pengujian dilakukan pada hari ke-1, hari ke-2, hari ke-3, hari ke-4 hingga hari ke-10.

Tabel 5.10 Skenario Pengujian Bobot Kesulitan Berdasarkan Skor Pemain

Nama Skenario	Pengujian bobot kesulitan berdasarkan skor pemain
Kode	UF-0006
Tujuan Pengujian	Mendeteksi apakah sistem mampu menentukan bobot kesulitan berdasarkan skor pemain
Kondisi Awal	Berada pada posisi debugging Unity Editor dengan Scene Gameplay yang terbuka
Data Input	Hari tempuh, experience
Prosedur Pengujian	1. Menekan tombol Play pada Unity Editor 2. Ubah parameter `currentDay` pada `TimaManager` agar sama dengan pengujian bobot kesulitan berdasarkan hari 3. Ubah parameter `experience` pada `Pemain` dengan nilai antara 0 sampai 1 4. Tunggu hingga hari regulasi keseimbangan 5. Bandingkan perbedaan bobot dengan

	pengujian berdasarkan hari tempuh
Hasil yang Diharapkan	Sistem dapat mangatur bobot kesulitan berdasarkan skor pemain
Hasil Pengujian	Berhasil
Hasil yang Diperoleh	Sistem dapat mangatur bobot kesulitan berdasarkan skor pemain



Gambar 5.4 Grafik Pengujian Bobot Kesulitan Berdasarkan Skor Pemain

5.2.2.3 Pengujian Bobot Bantuan Berdasarkan Bobot Kesulitan

Pengujian bobot bantuan berdasarkan bobot kesulitan bertujuan untuk mendeteksi apakah sistem dapat mengatur bobot bantuan berdasarkan bobot kesulitan pada waktu tertentu. Skenario pengujian bobot bantuan berdasarkan hari dijelaskan pada

Tabel 5.11.

Tabel 5.11 Skenario Pengujian Bobot Bantuan Berdasarkan Bobot Kesulitan

Nama Skenario	Pengujian bobot bantuan berdasarkan bobot kesulitan
Kode	UF-0007
Tujuan Pengujian	Mendeteksi apakah sistem mampu menentukan bobot bantuan berdasarkan bobot kesulitan
Kondisi Awal	Berada pada posisi debugging Unity Editor dengan Scene Gameplay yang terbuka
Data Input	Bobot kesulitan
Prosedur Pengujian	1. Menekan tombol Play pada Unity Editor 2. Ubah parameter `RealFitness` pada `GAMinus` 3. Tunggu hingga hari regulasi keseimbangan 4. Catat bobot bantuan yang berhasil dihitung
Hasil yang Diharapkan	Sistem dapat mangatur bobot kesulitan berdasarkan perubahan hari
Hasil Pengujian	Berhasil
Hasil yang Diperoleh	Sistem dapat mangatur bobot kesulitan berdasarkan perubahan hari

Tabel 5.12 Hasil Pengujian Bobot bantuan Berdasarkan Bobot Kesulitan

Hari ke -	Health	Hunger	Bobot Kesulitan	Bobot Bantuan
1	100	100	0	0
2	100	100	8	0

Hari ke -	Health	Hunger	Bobot Kesulitan	Bobot Bantuan
3	100	100	8	8
4	100	100	8	8
5	100	100	13	8
6	100	100	13.25	13
7	100	100	13.5	14
8	100	100	13.75	14
9	100	100	14	14
10	100	100	14.25	14

5.2.2.4 Pengujian Bobot Bantuan Berdasarkan Status Kesehatan Pemain

Pengujian bobot bantuan berdasarkan status kesehatan bertujuan untuk mendeteksi apakah sistem dapat mengatur bobot bantuan berdasarkan tingkat kesehatan pemain dalam waktu tertentu. Skenario pengujian bobot bantuan berdasarkan status kesehatan pemain dijelaskan pada Tabel 5.13.

Tabel 5.13 Skenario Pengujian Bobot Bantuan Berdasarkan Status Kesehatan Pemain

Nama Skenario	Pengujian bobot bantuan berdasarkan status kesehatan pemain
Kode	UF-0008
Tujuan Pengujian	Mendeteksi apakah sistem mampu menentukan bobot bantuan berdasarkan status kesehatan pemain
Kondisi Awal	Berada pada posisi debugging Unity Editor dengan Scene Gameplay yang terbuka
Data Input	Bobot kesulitan, health

Prosedur Pengujian	1. Menekan tombol Play pada Unity Editor 2. Ubah parameter RealFitness pada GAMinus agar sama dengan pengujian bobot kesulitan berdasarkan bobot kesulitan 3. Ubah parameter health pada Pemain dengan nilai antara 0 sampai 1 4. Tunggu hingga hari regulasi keseimbangan 5. Bandingkan perbedaan bobot dengan pengujian berdasarkan bobot kesulitan
Hasil yang Diharapkan	Sistem dapat mangatur bobot kesulitan berdasarkan skor pemain
Hasil Pengujian	Berhasil
Hasil yang Diperoleh	Sistem dapat mangatur bobot kesulitan berdasarkan skor pemain

Tabel 5.14 Hasil Uji Bobot Bantuan Berdasarkan Status Kesehatan Pemain

Hari ke -	Health	Hunger	Bobot Kesulitan	Bobot Bantuan
1	100	80	0	6
2	100	60	8	14
3	100	40	8	20
4	100	20	8	44
5	100	0	13	49
6	80	0	13.25	60
7	55	0	13.5	70
8	30	0	13.75	78
9	10	0	14	90
10	0	0	14.25	100

5.2.3 Pengujian Aturan Permainan

Pengujian aturan permainan bertujuan untuk mengecek apakah aplikasi dapat dimainkan hingga batas menang maupun kalah. Pembahasan pengujian aturan permainan akan dibagi ke dalam dua bagian, yaitu pengujian kemanangan pemain, dan pengujian kekalahan pemain.

5.2.3.1 Pengujian Kemenangan Pemain

Pengujian kemenangan pemain bertujuan untuk mengecek apakah permainan dapat dimenangkan oleh pemain. Skenario pengujian kemenangan pemain dijelaskan pada Tabel 5.15. Pada Gambar 5.5 menunjukkan apabila musuh terkuat telah kehabisan kekuatan, maka pemain akan memenangkan permainan.

Gambar 5.5 Tampilan Pengujian Kemenangan Pemain

Tabel 5.15 Skenario Pengujian Kemenangan Pemain

Nama Skenario	Pengujian kemenangan pemain
Kode	UF-0009
Tujuan Pengujian	Mendeteksi apakah permainan mampu dimenangkan pemain
Kondisi Awal	Berada pada Menu Utama permainan
Data Input	-
Prosedur Pengujian	1. Menekan tombol Play pada Menu Utama 2. Pemain memainkan permainan hingga hari terakhir dan mengalahkan lawan terkuat
Hasil yang Diharapkan	Sistem menampilkan informasi bahwa permainan telah dimenangkan
Hasil Pengujian	Berhasil
Hasil yang Diperoleh	Sistem menampilkan informasi permainan telah dimenangkan

5.2.3.2 Pengujian Kekalahan Pemain

Pengujian kekalahan pemain bertujuan untuk mengecek apakah permainan dapat membuat pemain kalah. Skenario pengujian kemenangan pemain dijelaskan pada Tabel 5.16.

Tabel 5.16 Skenario Pengujian Kekalahan Pemain

Nama Skenario	Pengujian kemenangan pemain
Kode	UF-0009
Tujuan Pengujian	Mendeteksi apakah permainan mampu membuat pemain kalah
Kondisi Awal	Berada pada Menu Utama permainan
Data Input	-

Prosedur Pengujian	1. Menekan tombol Play pada Menu Utama 2. nilai health pemain bernilai nol.
Hasil yang Diharapkan	Sistem memberikan tanda bahwa karakter utama tidak mampu bertahan hidup
Hasil Pengujian	Berhasil
Hasil yang Diperoleh	Sistem memberikan tanda bahwa karakter utama tidak mampu bertahan hidup

Pada Gambar 5.6 menunjukkan ketika nilai *health* karakter utama telah menunjukkan angka nol, maka tampilan halaman akan meredup dan menampilkan informasi bahwa permainan telah berakhir.

Gambar 5.6 Tampilan Pengujian Kekalahan Pemain

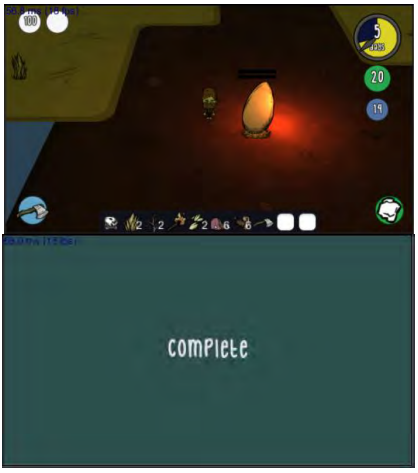
BAB VI
KESIMPULAN DAN SARAN

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. Aturan permainan dalam aplikasi ini telah diterapkan hingga pemain dapat menempuh status kalah maupun menang. Proses pemain menang dilakukan dengan cara membuat *health* batu bertuah yang menjadi musuh utama tersebut mencapai nilai nol seperti ditunjukkan pada Gambar 6.1.

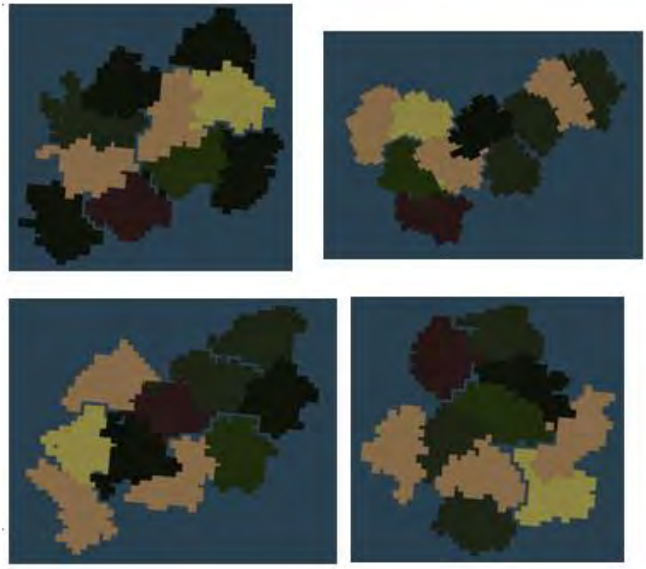


Gambar 6.1 Tampilan Pemain Memenangkan Permainan.

Proses pemain dinyatakan kalah jika pemain tidak dapat menjaga *health* yang dimilikinya hingga mencapai nilai nol seperti pada Gambar 6.2.

Gambar 6.2 Tampilan Pemain Dinyatakan Kalah

2. Peta permainan dapat dibangkitkan menggunakan *Procedural Content Generation*. Peta permainan yang dihasilkan berbentuk dinamis, di mana setiap permainan dimulai bentuk peta akan selalu berubah-ubah. Hasil dari pembangkitan peta dinamis ini ditunjukkan pada Gambar 6.3.
3. Skenario keseimbangan permainan dapat dibangkitkan menggunakan *Procedural Content Generation*. Ketika pemain merasa kesulitan, tingkat bantuan yang diberikan sistem juga meningkat. Hal ini dibuktikan dari tingkat kenaikan bobot bantuan yang meningkat dari angka 0 pada hari pertama ke angka 6 pada hari pertama. Ketika pemain memiliki perlengkapan yang cukup banyak, tingkat kesulitan permainan akan bertambah. Hal ini dibuktikan dengan kenaikan bobot kesulitan dari angka 4 pada hari pertama menjadi 16 pada hari pertama.



Gambar 6.3 Hasil Pembangkitan Peta Dinamis

6.2. Saran

Saran untuk pengembangan dan perbaikan sistem di masa yang akan datang adalah:

1. Pendefinisian bobot tiap objek lebih disesuaikan dengan fungsi objek tersebut.
2. Objek permainan bisa dikembangkan dengan menambahkan jenis baru dalam permainan.

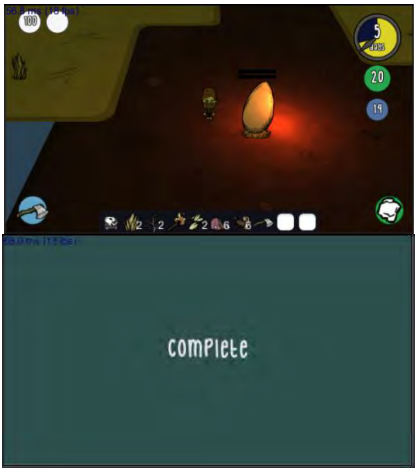
BAB VI
KESIMPULAN DAN SARAN

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. Aturan permainan dalam aplikasi ini telah diterapkan hingga pemain dapat menempuh status kalah maupun menang. Proses pemain menang dilakukan dengan cara membuat *health* batu bertuah yang menjadi musuh utama tersebut mencapai nilai nol seperti ditunjukkan pada Gambar 6.1.

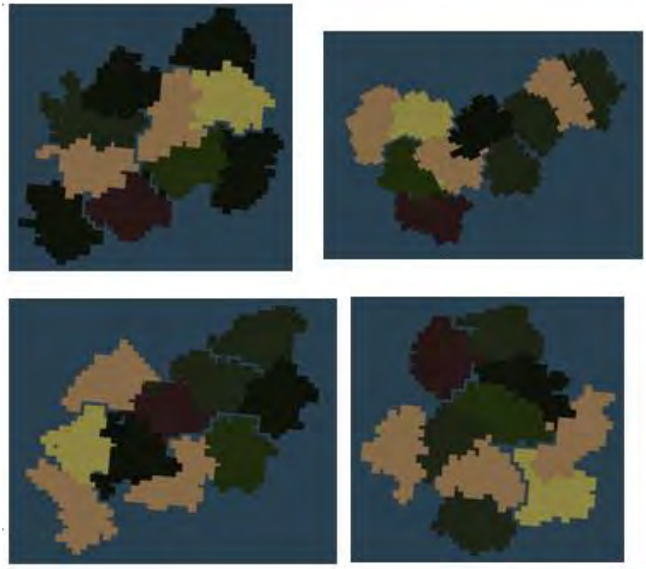


Gambar 6.1 Tampilan Pemain Memenangkan Permainan.

Proses pemain dinyatakan kalah jika pemain tidak dapat menjaga *health* yang dimilikinya hingga mencapai nilai nol seperti pada Gambar 6.2.

Gambar 6.2 Tampilan Pemain Dinyatakan Kalah

2. Peta permainan dapat dibangkitkan menggunakan *Procedural Content Generation*. Peta permainan yang dihasilkan berbentuk dinamis, di mana setiap permainan dimulai bentuk peta akan selalu berubah-ubah. Hasil dari pembangkitan peta dinamis ini ditunjukkan pada Gambar 6.3.
3. Skenario keseimbangan permainan dapat dibangkitkan menggunakan *Procedural Content Generation*. Ketika pemain merasa kesulitan, tingkat bantuan yang diberikan sistem juga meningkat. Hal ini dibuktikan dari tingkat kenaikan bobot bantuan yang meningkat dari angka 0 pada hari pertama ke angka 6 pada hari pertama. Ketika pemain memiliki perlengkapan yang cukup banyak, tingkat kesulitan permainan akan bertambah. Hal ini dibuktikan dengan kenaikan bobot kesulitan dari angka 4 pada hari pertama menjadi 16 pada hari pertama.



Gambar 6.3 Hasil Pembangkitan Peta Dinamis

6.2. Saran

Saran untuk pengembangan dan perbaikan sistem di masa yang akan datang adalah:

1. Pendefinisian bobot tiap objek lebih disesuaikan dengan fungsi objek tersebut.
2. Objek permainan bisa dikembangkan dengan menambahkan jenis baru dalam permainan.

DAFTAR PUSTAKA

[1] D. Whitley, "A Genetic Algorithm Tutorial," *Statistics and Computing,* vol. 4, pp. 65-85, 1994.

[2] J. Togelius, G. N. Yannakakis, K. O. Stanley dan C. Browne, "Search-Based Procedural Content Generation," dalam *Applications of Evolutionary Computation*, Springer Berlin Heidelberg, 2010, pp. 141-150.

[3] K. Hartsook, A. Zook, S. Das dan M. O. Riedl, "Toward Supporting Stories with Procedurally Generated Game Worlds," dalam *CIG*, Chicago, IEEE, 2011, pp. 297-304.

[4] GameRangkings, "Don't Starve for PC," [Online]. Available: http://www.gamerankings.com/pc/690316-dont-starve/index.html. [Diakses 6 Juli 2015].

[5] A. Wawro, "Klei Sold More than a Million Copies of Don't Starve in 2013," Gamasutra, 21 Januari 2014. [Online]. Available: http://www.gamasutra.com/view/news/208997/Klei_sold_more_than_a_million_copies_of_Dont_Starve_in_2013.php. [Diakses 6 Juli 2015].

[6] G. Andrade, R. Geber, A. S. Gomes dan V. Corruble, "Dynamic Game Balancing: An Evaluation of User Satisfaction," dalam *AIIDE*, The AAAI Press, 2006, pp. 3-8.

BIODATA PENULIS

Penulis, Andrie Prasetyo Utomo lahir di kota Surabaya pada tanggal 02 Oktober 1993. Penulis adalah anak keempat dari empat bersaudara.

Penulis telah menempuh pendidikan formal di SD Negeri Beringin 477 Surabaya (1999-2005), SMP Negeri 26 Surabaya (2005-2008), dan SMA Negeri 11 Surabaya (2008 – 2011). Pada tahun 2011, penulis memulai pendidikan S1 Jurusan Teknik Informatika, Fakultas Teknologi Informasi di Institut Teknologi Sepuluh Nopember Surabaya, Jawa Timur.

Di Jurusan Teknik Informatika, penulis mengambil bidang minat Interaksi, Grafika, dan Seni. Penulis memiliki ketertarikan di bidang pembuatan *game*. Selama menjadi mahasiswa, penulis pernah menjadi Asisten Software Perkantoran PIKTI (2012-2013), Asisten Basis Data S1 (2013-2014), Asisten Pemrograman Web S1 (2013-2014), Asisten Desain Web PIKTI (2013-2014), Asisten Pemrograman Web PIKTI (2013-2014), Asisten Multimedia Pendukung *Game* dan Animasi PIKTI (2014-2015), Asisten Teknologi Multimedia Terapan PIKTI (2014-2015), dan Asisten Teknologi Pembuatan *Game* PIKTI (2014-2015). Penulis dapat dihubungi melalui email: andriepu@gmail.com.