

**PERANCANGAN GAME “CAMPUS ADVENTURE”
MENGUNAKAN METODE *FUZZY LOGIC***

SKRIPSI

Oleh:

**FERNANDO STANLEY FELIX
NIM.2044020**



**JENJANG PENDIDIKAN STRATA-1
PROGRAM STUDI TEKNIK INFORMATIKA
STMIK TIME
MEDAN
2024**

LEMBAR PENGESAHAN

PERANCANGAN *GAME “CAMPUS ADVENTURE”*
MENGGUNAKAN METODE *FUZZY LOGIC*

SKRIPSI

Diajukan Untuk Melengkapi Persyaratan Guna
Mendapat Gelar Sarjana Strata Satu
Program Studi Teknik Informatika

Oleh:

FERNANDO STANLEY FELIX
NIM. 2044020

Disetujui Oleh:

Dosen Pembimbing I



Joni, S.Kom., M.M.

Dosen Pembimbing II



Feriani Astuti Tarigan, M.Kom., M.M

Medan, 25Juli2024

Diketahui dan Disahkan Oleh:

Ketua STMIK TIME

Ketua Program Studi



Dr. Edi Wijaya, S.Kom., M.Kom., M.M

Robet, M.Kom

Kepada
Yth Ketua STMIK TIME
Dr. Edi Wijaya, S.Kom., M.Kom., M.M
di Medan

Hal : **Surat Pernyataan Bebas Plagiat**

Saya yang bertanda tangan di bawah ini :

Nama	: Fernando Stanley Felix
NIM	: 2044020
Program Studi	: Teknik Informatika
Angkatan	: 2020
Judul Skripsi	: Perancangan <i>Game</i> “ <i>Campus Adventure</i> ” Menggunakan Metode <i>Fuzzy Logic</i>

Dengan ini menyatakan bahwa skripsi yang Saya tulis adalah karya ilmiah asli dari pikiran saya sendiri.

Apabila kemudian hari terbukti bahwa skripsi tersebut merupakan Plagiat, maka saya bersedia gelar dan ijazah yang saya peroleh ditinjau kembali.

Demikian surat pernyataan ini Saya perbuat dengan pikiran yang sehat, tanpa tekanan dari siapapun.

Diketahui oleh
Pembantu Ketua I



Hendri, M.Kom

Medan, 25 Juli 2024

Yang membuat pernyataan



Fernando Stanley Felix

ABSTRAK

Penelitian ini bertujuan untuk merancang dan mengembangkan *game* "*Campus Adventure*" yang menggabungkan simulasi kehidupan perkuliahan dengan elemen petualangan, menggunakan metode *Fuzzy Logic* untuk menciptakan musuh NPC yang cerdas dan adaptif. Latar belakang penelitian ini didasarkan pada fenomena calon mahasiswa yang merasa cemas dan kurang percaya diri dalam menghadapi tantangan perkuliahan. *Game* ini dirancang untuk memberikan pengalaman simulasi yang mendalam dan interaktif, di mana pemain dapat menghadapi berbagai tantangan seperti rasa malas, konflik dengan teman, dan manajemen waktu, yang disimulasikan melalui karakter musuh NPC. Penelitian ini berhasil menunjukkan bahwa metode *Fuzzy Logic* efektif dalam menciptakan perilaku NPC yang dinamis dan adaptif, memberikan tantangan yang realistis bagi pemain. *Game* "*Campus Adventure*" diharapkan dapat membantu calon mahasiswa mempersiapkan diri dengan lebih baik untuk kehidupan perkuliahan, mengembangkan keterampilan manajemen, dan meningkatkan rasa percaya diri mereka. Selain itu, penelitian ini juga menambah khasanah penelitian mengenai penerapan metode *Fuzzy Logic* dalam pengembangan *game*, yang dapat menjadi referensi bagi penelitian selanjutnya.

Kata Kunci: *Campus Adventure*, *Game*, Aplikasi *Mobile*, Simulasi Kehidupan, Petualangan

ABSTRACT

This research aims to design and develop the game "Campus Adventure," which combines a college life simulation with adventure elements, using Fuzzy Logic methods to create intelligent and adaptive NPC enemies. The background of this research is based on the phenomenon of prospective students feeling anxious and lacking confidence in facing college challenges. This game is designed to provide a deep and interactive simulation experience, where players can face various challenges such as laziness, conflicts with friends, and time management, simulated through NPC enemy characters. This research successfully demonstrates that the Fuzzy Logic method is effective in creating dynamic and adaptive NPC behavior, providing realistic challenges for players. The game "Campus Adventure" is expected to help prospective students better prepare for college life, develop management skills, and increase their confidence. Additionally, this research contributes to the body of research on the application of Fuzzy Logic methods in game development, which can serve as a reference for future research.

Keywords: Campus Adventure, Game, Mobile Application, Life Simulation, Adventure

KATA PENGANTAR

Ucapan puji dan syukur kepada hadirat Tuhan Yang Maha Esa atas segala berkat, hikmat, dan pertolongan-Nya sehingga penulis dapat menyelesaikan penyusunan skripsi ini tepat pada waktunya.

Tugas Akhir ini disusun untuk memenuhi salah satu syarat dalam menyelesaikan pendidikan Program Strata I pada STMIK TIME. Penulis berharap agar hasil dari penelitian yang dilakukan ini dapat membawa banyak manfaat bagi berbagai pihak.

Dalam menyelesaikan tugas akhir ini, penulis banyak mendapat bantuan dari berbagai pihak. Oleh karena itu, melalui kesempatan ini penulis ingin menyampaikan terima kasih kepada:

1. Bapak Joni, S.Kom., M.M., selaku Pembimbing I yang telah membimbing penulis selama mengerjakan skripsi ini.
2. Ibu Feriani Astuti Tarigan, M.Kom., M.M., selaku Pembimbing I yang telah membimbing penulis selama mengerjakan skripsi ini.
3. Bapak Simon Kangga Lee, selaku Ketua Yayasan STMIK TIME Medan.
4. Bapak Dr. Edi Wijaya, S.Kom., M.Kom., M.M., selaku Ketua STMIK TIME Medan yang telah membimbing penulis selama mengerjakan skripsi ini.
5. Bapak Hendri, M.Kom selaku Pembantu Ketua I STMIK TIME Medan yang telah membimbing penulis selama mengerjakan skripsi ini.
6. Bapak Robet, M.Kom, selaku Ketua Program Studi Teknik Informatika STMIK TIME Medan yang telah membimbing penulis selama mengerjakan skripsi ini.
7. Seluruh Dosen STMIK TIME Medan yang telah mendidik dan membimbing penulis selama masa perkuliahan.
8. Teristimewa kepada orang tua dan keluarga tercinta, yang telah memberikan dukungan baik moral maupun materi kepada penulis selama ini.
9. Seluruh teman yang telah banyak membantu penulis dan memberikan dorongan serta semangat dalam menyelesaikan skripsi ini

10. Semua yang telah membantu penulis dalam menyelesaikan skripsi ini yang tidak dapat disebutkan satu persatu

Penulis menyadari dalam proses penyelesaian skripsi yang berjudul “Perancangan *Game “Campus Adventure”* Menggunakan Metode *Fuzzy Logic*” masih jauh dari sempurna, mengingat keterbatasan kemampuan yang penulis miliki dan terbatasnya waktu yang digunakan. Untuk itu, penulis mengharapkan saran, kritikan, dan masukan yang membangun demi perkembangan dan kemajuan penelitian selanjutnya. Semoga penulisan skripsi ini dapat bermanfaat bagi pihak yang berkepentingan.

Akhir kata penulis berharap agar laporan penulisan Tugas Akhir ini dapat berguna dan bermanfaat bagi pembaca.

Medan, 25 Juli 2024

Penulis,

Fernando Stanley Felix

DAFTAR ISI

ABSTRAK	i
ABSTRACT	ii
KATA PENGANTAR.....	iii
DAFTAR ISI.....	v
DAFTAR GAMBAR.....	vii
DAFTAR TABEL	ix
DAFTAR LAMPIRAN	x
BAB I PENDAHULUAN	1
1.1 Latar Belakang.....	1
1.2 Rumusan Masalah.....	3
1.3 Batasan Masalah	3
1.4 Tujuan dan Manfaat Penelitian	3
1.5 Metodologi Penelitian.....	4
1.6 Analisis <i>Game</i>	6
1.7 Perancangan <i>Game</i>	6
1.8 Sistematika Penulisan	7
BAB II TINJAUAN PUSTAKA.....	8
2.1 <i>Game</i>	8
2.2 Aplikasi <i>Mobile</i>	10
2.3 <i>Artificial Intelligence</i> (AI)	11
2.4 <i>Non-player Character</i> (NPC)	15
2.5 Metode <i>Fuzzy Logic</i>	16
2.6 <i>Use Case Diagram</i>	18
2.7 Animasi.....	20
2.8 <i>Storyboard</i>	25
BAB III METODE PENELITIAN	27
3.1 Analisis Masalah.....	27
3.1.1 Analisis Proses Yang Berjalan Sesuai di Lapangan.....	27

3.1.2 Analisis Metode Yang Digunakan	29
3.1.3 Analisis Sistem Usulan.....	32
3.2 Perancangan	35
3.2.1 Perancangan Karakter.....	35
3.2.2 Perancangan <i>Storyboard</i>	36
BAB IV HASIL DAN PEMBAHASAN	41
4.1 Hasil	41
4.2 Pembahasan	52
BAB V KESIMPULAN DAN SARAN	54
5.1 Kesimpulan	54
5.2 Saran	54
DAFTAR PUSTAKA.....	56
LAMPIRAN	

DAFTAR GAMBAR

Gambar 2.1 Contoh <i>Stop-motion Animation</i> [16]	20
Gambar 2.2 <i>Traditional Animation</i> [16]	21
Gambar 2.3 <i>Computer Animation</i> [16].....	22
Gambar 2.4 Perbedaan Animasi 2D dan 3D [16]	23
Gambar 2.5 Contoh <i>Storyboard</i> [17]	26
Gambar 3.1 <i>Use Case Diagram Game “Campus Adventure”</i>	32
Gambar 3.2 Rancangan Karakter Utama	35
Gambar 3.3 Rancangan Karakter Musuh	36
Gambar 3.4 Rancangan Tampilan <i>Splash Screen</i>	36
Gambar 3.5 Rancangan Tampilan <i>Scene Awal Menu Game</i>	37
Gambar 3.6 Rancangan <i>Storyboard Prolog Game</i>	38
Gambar 3.7 Rancangan <i>Storyboard Lanjutan Prolog Game</i>	38
Gambar 3.8 Rancangan Tampilan <i>Scene Permainan Dimulai</i>	39
Gambar 3.9 Rancangan Tampilan <i>Scene Help</i>	40
Gambar 3.10 Rancangan Tampilan <i>SceneList Quest</i>	40
Gambar 4.1 Karakter Utama	41
Gambar 4.2 Karakter Musuh.....	42
Gambar 4.3 Tampilan <i>Splash Screen</i>	42
Gambar 4.4 Tampilan <i>Scene Awal Menu Game</i>	43
Gambar 4.5 Tampilan <i>Prolog Game</i>	43
Gambar 4.6 Tampilan <i>Lanjutan Prolog Game-1</i>	44
Gambar 4.7 Tampilan <i>Lanjutan Prolog Game-2</i>	44
Gambar 4.8 Tampilan <i>Lanjutan Prolog Game-3</i>	45
Gambar 4.9 Tampilan <i>Lanjutan Prolog Game-4</i>	45
Gambar 4.10 Tampilan <i>Lanjutan Prolog Game-5</i>	46
Gambar 4.11 Tampilan <i>Lanjutan Prolog Game-6</i>	46
Gambar 4.12 Tampilan <i>Lanjutan Prolog Game-7</i>	47
Gambar 4.13 Tampilan <i>Lanjutan Prolog Game-8</i>	47

Gambar 4.14 Tampilan Lanjutan Prolog <i>Game-9</i>	48
Gambar 4.15 Tampilan Lanjutan <i>Prolog Game-10</i>	48
Gambar 4.14 Tampilan Lanjutan Prolog <i>Game-11</i>	49
Gambar 4.17 Tampilan Lanjutan Prolog <i>Game-12</i>	49
Gambar 4.18 Tampilan <i>Scene</i> Permainan Dimulai	50
Gambar 4.19 Tampilan <i>Scene Help</i>	51
Gambar 4.20 Tampilan <i>SceneList Quest</i>	51

DAFTAR TABEL

Tabel 1.1 Jadwal Penelitian.....	5
Tabel 2.1 Simbol <i>Use Case Diagram</i> [15].....	19
Tabel 3.1 Skenario <i>Use Case Play Game</i>	33
Tabel 3.2 Skenario <i>Use Case Melihat Level Game</i>	33
Tabel 3.3 Skenario <i>Use Case Help</i>	34
Tabel 3.4 Skenario <i>Use Case Exit</i>	35

DAFTAR LAMPIRAN

Lampiran 1 Daftar Riwayat Hidup

Lampiran 2 Surat Keputusan Dosen Pembimbing Skripsi

Lampiran 3 Kartu Bimbingan Dosen I

Lampiran 4 Kartu Bimbingan Dosen II

Lampiran 5 *Listing* Program

BAB I

PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi yang pesat membuat adanya perubahan dalam gaya hidup masyarakat[1]. Salah satu contoh perubahan dalam gaya hidup tersebut adalah banyaknya orang yang bermain *game* sekarang. *Game* merupakan sebuah media yang biasa digunakan untuk meluangkan waktu kosong, dimana orang biasanya memainkan *game* dengan alasan untuk menghilangkan stress. Seiring berjalannya waktu, selain sebagai media hiburan, *game* juga banyak dimanfaatkan untuk berbagai kepentingan seperti untuk edukasi ataupun simulasi mengenai kehidupan perkuliahan[2]. Salah satu aspek menarik adalah permainan yang menggabungkan simulasi kehidupan kampus dengan petualangan *game*.

Fenomena permasalahan yang ditemukan di mana masih banyak calon mahasiswa yang merasa bahwa perkuliahan sangat sulit serta memiliki banyak tantangan sehingga terkadang mereka merasa cemas, kurang percaya diri, dan tidak siap menghadapi proses perkuliahan. Padahal perkuliahan tidaklah sulit sebagaimana yang dibayangkan asal calon mahasiswa mampu menghadapi tantangan-tantangan umum seperti malas belajar, kurangnya keterampilan manajemen waktu, kesulitan dalam mengatur prioritas, dan tantangan-tantangan lainnya dalam perkuliahan [3].

Oleh sebab itu, untuk memberikan motivasi kepada calon mahasiswa baru, maka dikembangkan *game* bernama "*Campus Adventure*" untuk menyediakan

pengalaman simulasi kehidupan petualangan perkuliahan yang mendalam. Alasan pemilihan media *game* dikarenakan *game* memiliki potensi untuk memberikan pengalaman belajar yang interaktif, menarik, dan lebih terlibat dibandingkan dengan metode lainnya. Dalam *game* ini, calon mahasiswa memerankan karakter mahasiswa dalam lingkungan virtual, yang disimulasikan menghadapi tantangan seperti rasa malas, konflik dengan teman, dan manajemen waktu ketika perkuliahan. Melalui pengalaman interaktif ini, calon mahasiswa dapat memahami lebih baik kehidupan perkuliahan, mengembangkan keterampilan manajemen, dan merasa lebih siap menghadapi proses perkuliahan.

Pada penelitian ini untuk membuat *game* “*Campus Adventure*” yang menarik maka diimplementasikan metode *Fuzzy Logic* untuk menciptakan musuh *Artificial Intelligence* (AI) yang pintar sehingga memberikan tantangan bagi pemain. Penelitian yang mengimplementasikan metode *Fuzzy Logic* dalam perancangan *game* sudah pernah dilakukan sebelumnya seperti penelitian yang membangun *game* Mlijo 2D dengan mengimplementasikan metode *Fuzzy Logic*. Hasil penelitian menunjukkan bahwa metode *Fuzzy Logic* cocok dalam menciptakan karakter NPC yang cerdas [4]. Penelitian berikutnya mengimplementasikan metode *Fuzzy Logic* dalam menciptakan perilaku NPC di mana hasil penelitian tersebut menyimpulkan bahwa metode *Fuzzy Logic* dapat menghasilkan *decision* yang adaptif berdasarkan kondisi yang ada[5].

Berdasarkan uraian pada latar belakang dan permasalahan tersebut maka penelitian ini akan mengambil judul “**Perancangan Game “*Campus Adventure*” Menggunakan Metode *Fuzzy Logic*”**”.

1.2 Rumusan Masalah

Berikut inirumusan masalah berdasarkan latar belakang masalah yang diuraikan yaitu:

1. Bagaimana merancang sebuah *game adventure* yang mampu mengenalkan petulangan dalam perkuliahan?
2. Bagaimana mengimplementasikan metode *Fuzzy Logic* dalam merancang *game adventure*?

1.3 Batasan Masalah

Berikut ini akan diuraikan batasan masalah pada penelitian ini agar penelitian dapat diselesaikan pada waktunya antara lain:

1. Musuh NPC yang diciptakan terdiri dari 3 karakter yaitu musuh malas dilambangkan dengan musuh *Spider*, konflik dengan teman dilambangkan dengan *Dinosaurus Hijau*, dan manajemen waktu dilambangkan dengan *Dinosaurus Merah*.
2. *Game* bersifat *single user*.
3. Level yang tersedia terdiri dari 3 level yang merepresentasikan proses perkuliahan dalam bentuk *list quest*.
4. Aplikasi yang dibangun dibatasi dapat dijalankan pada perangkat *mobile* dengan sistem operasi Android.

1.4 Tujuan dan Manfaat Penelitian

Tujuan yang diharapkan dalam penelitian ini adalah:

1. Merancang sebuah *game adventure* yang mampu mengenalkan petualangan dalam perkuliahan.
2. Mengimplementasikan dan mengevaluasi metode *Fuzzy Logic* dalam merancang *game adventure*.

Manfaat yang diharapkan dalam penelitian ini adalah:

- 1 Membantu dalam meningkatkan keterlibatan mahasiswa dalam memahami dan mengatasi tantangan yang mereka hadapi saat memulai perkuliahan. Ini akan membantu mereka merasa lebih siap dan percaya diri dalam menghadapi perubahan besar dalam kehidupan kampus.
- 2 Menambah khasanah penelitian terutama yang berkaitan dengan penerapan metode *Fuzzy Logic* serta diharapkan menjadi referensi atau acuan untuk penelitian-penelitian berikutnya yang memiliki metode pengembangan yang sama.
- 3 Membantu mahasiswa mengembangkan berbagai keterampilan, termasuk manajemen waktu, pengambilan keputusan, dan kemampuan sosial. Ini akan membantu mereka tidak hanya dalam perkuliahan tetapi juga dalam kehidupan sehari-hari mereka setelah lulus dari perguruan tinggi.

1.5 Metodologi Penelitian

Dalam penelitian ini, metode penelitian yang digunakan adalah sebagai berikut:

1. Jadwal Penelitian

Berikut ini akan diuraikan tabel jadwal penelitian dari awal penelitian hingga penelitian selesai yang dapat dilihat pada Tabel 1.1.

Tabel 1.1 Jadwal Penelitian

NO	Waktu Ket.	2023								2024															
		November				Desember				Januari				Februari				Maret				April			
		1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1	Rumusan Masalah																								
2	Pengumpulan Data																								
3	Analisis <i>Game</i>																								
4	Perancangan <i>Game</i>																								
5	Uji Coba <i>Game</i>																								
6	Penulisan Laporan																								

2. Bahan dan Alat

Berikut ini akan dijelaskan bahan dan alat yang digunakan yaitu berupa *hardware* dan *software* yang digunakan dalam proses penelitian.

Hardware yang digunakan yaitu sebuah laptop dengan spesifikasi:

- Processor Intel® Core™ i7-8700 @ 3.20GHz 3.19GHz
- RAM 16GB.
- Perangkat *keyboard*.
- Perangkat *touchpad*.
- Display* dengan resolusi 1920 x 1080 pixel.

Software yang digunakan yaitu:

- Sistem Operasi Windows 10 64 Bit.
- Android Studio.
- Corel Draw.
- Adobe Photoshop.

3. Metode Pengumpulan Data

Metode pengumpulan data yang digunakan pada penelitian ini antara lain sebagai berikut:

a. Metode Observasi

Penulis melakukan observasi secara langsung ke kampus STMIK TIME untuk mengumpulkan data-data terkait tantangan-tantangan mahasiswa dalam perkuliahan agar dapat diterjemahkan ke dalam karakter musuh NPC *game*.

b. Metode Studi Pustaka

Penulis melakukan pengumpulan data melalui internet maupun buku yang sesuai dengan topik yang dibahas.

1.6 Analisis Game

Analisis sistem pada penelitian ini terbagi menjadi beberapa tahapan proses yaitu analisis proses berjalan yang menguraikan proses berjalan yang digunakan dalam pengenalan proses perkuliahan kepada mahasiswa, analisis metode yang diusulkan menguraikan cara kerja dari metode *Fuzzy Logic*, dan analisis *game* usulan menguraikan fitur-fitur yang ada pada *game* “*Campus Adventure*” yang akan dikembangkan.

1.7 Perancangan Game

Perancangan *game* pada penelitian ini terdiri dari beberapa tahapan yaitu perancangan karakter *game* menggunakan Corel Draw dan Adobe Photoshop dan perancangan *storyboard game* menggunakan *software* Balsamiq Mockup 3.

1.8 Sistematika Penulisan

Secara garis besar Sistematika pembahasan pada skripsi ini dapat peneliti jelaskan sebagai berikut:

BAB I : PENDAHULUAN

Bab ini menguraikan latar belakang masalah, rumusan masalah, batasan masalah, tujuan dan manfaat, dan sistematika penulisan.

BAB II : TINJAUAN PUSTAKA

Bab ini membahas mengenai teori yang berkaitan dengan perancangan sistem dan teori-teori lainnya yang berhubungan dengan pembuatan *game* tersebut.

BAB III : METODE PENELITIAN

Bab ini menyajikan analisa masalah yang dihadapi dan perancangan *game*.

BAB IV : HASIL DAN PEMBAHASAN

Bab ini menguraikan rincian mengenai hasil dari perancangan *game* dan pembahasan terhadap sistem yang di kembangkan.

BAB V : KESIMPULAN DAN SARAN

Merupakan bab terakhir yang berisi kesimpulan dari uraian-uraian sebelumnya sistem serta saran yang berisi tentang gagasan-gagasan yang dapat dikembangkan selanjutnya.

BAB II

TINJAUAN PUSTAKA

2.1 *Game*

Game merupakan kata dalam bahasa inggris yang berarti permainan. Permainan adalah sesuatu yang dapat dimainkan dengan aturan tertentu sehingga ada yang menang dan ada yang kalah, biasanya dalam konteks tidak serius atau dengan tujuan *refreshing* [6]. Suatu cara belajar yang digunakan dalam menganalisa interaksi antara sejumlah pemain maupun perorangan yang menunjukkan strategi-strategi yang rasional. Terdapat beberapa jenis *game* yang dapat dikelompokkan sebagai berikut [7]:

1. Simulasi.

Contoh permainan yang termasuk dalam *game* simulasi adalah simulasi konstruksi dan manajemen, simulasi kendaraan seperti yang diterapkan pada permainan balapan, perang, luar angkasa, dan mecha.

2. Edukasi.

Contohnya adalah *edugames* yang dibuat dengan tujuan spesifik sebagai alat pendidikan, baik itu untuk belajar mengenal warna untuk balita, mengenal huruf dan angka, matematika, sampai belajar bahasa asing. *Developer* yang membuatnya, harus memperhitungkan berbagai hal agar *game* ini benar-benar dapat mendidik, menambah pengetahuan dan meningkatkan ketrampilan yang memainkannya.

3. *Entertainment.*

- a. Aksi seperti *shooting* (tembak-tembakkan atau hajar-hajaran bisa juga tusuk-tusukan, tergantung cerita dan tokoh di dalamnya). *Game* jenis ini sangat memerlukan kecepatan refleks, koordinasi mata-tangan, juga waktu. inti dari game jenis ini adalah tembak-tembakkan.
- b. *Fighting* (pertarungan), ada yang mengelompokan *game fighting* di bagian Aksi, namun penulis berpendapat berbeda, jenis ini memang memerlukan kecepatan refleks dan koordinasi mata-tangan, tetapi inti dari *game* ini adalah penguasaan jurus (hafal caranya dan lancar mengeksekusinya), pengenalan karakter dan waktu sangatlah penting. Dan berbeda seperti *game* Aksi pada umumnya yang umumnya hanya melawan *Artificial Intelligence* atau istilah umumnya melawan komputer saja, pemain jenis *fighting game* ini baru teruji kemampuan sesungguhnya dengan melawan pemain lainnya.
- c. Petualangan, *game* murni petualangan lebih menekankan pada jalan cerita dan kemampuan berpikir pemain dalam menganalisa tempat secara visual, memecahkan teka-teki maupun menyimpulkan rangkaian peristiwa dan percakapan karakter hingga penggunaan benda-benda tepat pada tempat yang tepat.
- d. *Role playing*, *game* jenis ini sesuai dengan terjemahannya, bermain peran, memiliki penekanan pada tokoh/peran perwakilan pemain di dalam permainan, yang biasanya adalah tokoh utamanya, dimana seiring kita memainkannya, karakter tersebut dapat berubah dan berkembang ke arah yang diinginkan pemain (biasanya menjadi

semakin hebat, semakin kuat, semakin berpengaruh, dll) dalam berbagai parameter yang biasanya ditentukan dengan naiknya *level*.

- e. *Casual games*, sesuai namanya, *game* yang casual itu tidak kompleks, mainnya rileks dan sangat mudah untuk dipelajari. Jenis ini biasanya memerlukan spesifikasi komputer yang standar pada jamannya dan ukurannya tidak lebih dari 100 MB karena biasanya dapat di *download* versi *demo*-nya di *website* resminya. *Genre* permainannya biasanya *puzzle* atau *action* sederhana dan umumnya dapat dimainkan hanya menggunakan.
- f. *Multiplayer online*, *game* yang dapat dimainkan secara bersamaan oleh lebih dari 2 orang (bahkan dapat mencapai puluhan ribu orang dalam satu waktu) membuat pemain dapat bermain bersama dalam satu dunia virtual dari sekedar chatting hingga membunuh naga bersama teman yang entah bermain di mana. Umumnya permainan tipe ini dimainkan di PC dan bertema RPG, walau ada juga yang bertema musik atau *action*.
- g. *Arcade game*, ciri khas dari *genre* ini adalah *gameplay* yang cepat, aksi yang intens, tingkat kesulitan yang meningkat secara bertahap, dan fokus pada skor tinggi atau pencapaian tertentu.

2.2 Aplikasi Mobile

Aplikasi *mobile* adalah perangkat lunak yang dibuat khusus untuk dijalankan di dalam tablet atau *smartphone*. Aplikasi *mobile* ini bisa diperoleh dari Play Store dari banyak Platform yang sudah tersedia saat ini seperti, iOS,

Android, Blackberry atau Windows Phone. Dengan menggunakan aplikasi *mobile* para pengguna merasakan kemudahan dalam menjalankan segala aktifitas seperti hiburan, berjualan, belajar, pekerjaan kantor, *browsing* dan masih banyak lagi [8]. Aplikasi *mobile* biasanya adalah aplikasi yang tersedia pada internet yang dapat diunduh oleh pengguna jika mereka ingin menggunakannya dengan tujuan tertentu untuk membantu pengguna sesuai dengan jenis aplikasi yang disediakan [9]. *Mobile* adalah kata sifat yang berarti dapat bergerak atau dapat digerakkan dengan bebas dan mudah [10]. Dengan menggunakan aplikasi *mobile*, maka dapat dengan mudah melakukan berbagai macam aktifitas mulai dari hiburan, berjualan, belajar, mengerjakan pekerjaan kantor, *browsing* dan lain sebagainya.

2.3 *Artificial Intelligence (AI)*

Kecerdasan buatan atau yang sering dikenal dengan *Artificial Intelligence* (AI) merupakan kemampuan mesin atau sistem komputer untuk meniru atau menunjukkan kecerdasan manusia. Definisi ini melibatkan kemampuan sistem untuk mengumpulkan informasi, memahami konteks, melakukan analisis, membuat keputusan, dan belajar dari pengalaman untuk menghadapi tugas-tugas yang kompleks. Tujuan utama AI adalah untuk mengembangkan sistem yang dapat melakukan tugas-tugas yang biasanya memerlukan kecerdasan manusia. Beberapa tujuan utama AI meliputi[11]:

1. Pengolahan Bahasa Alami (*Natural Language Processing*).

AI bertujuan untuk mengembangkan sistem yang mampu memahami, menghasilkan, dan berinteraksi dengan bahasa manusia secara alami. Hal

ini melibatkan kemampuan sistem untuk memahami konteks, sintaksis, semantik, dan pragmatik dalam bahasa manusia.

2. Pembelajaran Mesin (*Machine Learning*).

AI bertujuan untuk mengembangkan algoritma dan teknik yang memungkinkan mesin untuk belajar dari data dan pengalaman, tanpa diprogram secara eksplisit. Pembelajaran mesin melibatkan identifikasi pola, membuat prediksi, dan mengoptimalkan kinerja sistem berdasarkan data yang diberikan.

3. Pengenalan Pola (*Pattern Recognition*).

AI bertujuan untuk mengembangkan sistem yang dapat mengenali pola dan fitur dalam data, seperti pengenalan wajah, pengenalan suara, dan pengenalan tulisan tangan. Ini melibatkan penggunaan algoritma dan teknik untuk mengidentifikasi dan mengklasifikasikan pola-pola yang ada dalam data.

4. Penalaran dan Pengambilan Keputusan (*Reasoning and Decision Making*).

AI bertujuan untuk mengembangkan sistem yang dapat menganalisis informasi, melakukan penalaran, dan membuat keputusan berdasarkan pemahaman konteks dan aturan yang ditentukan. Ini melibatkan penggunaan logika formal, inferensi, dan pemodelan pengetahuan untuk memungkinkan sistem AI untuk mengambil keputusan yang masuk akal.

Berikut ini terdapat beberapa komponen-komponen utama dalam sistem

AI meliputi [11]:

1. Basis Pengetahuan (*Knowledge Base*).

Tempat penyimpanan informasi dan pengetahuan yang digunakan oleh sistem AI. Basis pengetahuan dapat berisi fakta, aturan, konsep, dan model yang relevan dengan domain atau tugas yang sedang dihadapi.

2. Mesin Inferensi (*Inference Engine*).

Mesin inferensi bertanggung jawab untuk melakukan proses logika dan inferensi berdasarkan informasi yang ada dalam basis pengetahuan. Ini melibatkan penerapan aturan, penalaran, dan manipulasi pengetahuan untuk mencapai kesimpulan atau membuat keputusan.

3. Antarmuka Pengguna (*User Interface*).

Antarmuka pengguna memungkinkan interaksi antara pengguna manusia dan sistem AI. Ini dapat berupa antarmuka grafis, perintah suara, atau interaksi berbasis teks yang memungkinkan pengguna untuk memberikan input, menerima output, dan berkomunikasi dengan sistem AI.

4. Algoritma dan Teknik Kecerdasan Buatan (*AI Algorithms and Techniques*).

Ada berbagai algoritma dan teknik yang digunakan dalam AI, termasuk pembelajaran mesin, jaringan saraf tiruan, logika proposisional, pemrosesan bahasa alami, dan banyak lagi. Algoritma ini memungkinkan sistem AI untuk memproses data, mengenali pola, belajar dari pengalaman, dan membuat keputusan.

Ada beberapa jenis AI yang digunakan dalam berbagai konteks dan aplikasi. Dalam hal ini, kita akan menjelajahi beberapa jenis utama AI, termasuk sistem berbasis aturan, sistem pakar, pembelajaran mesin, jaringan saraf tiruan,

dan AI yang terkait dengan robotika. Berikut adalah beberapa jenis AI yang umum[11]:

1. Sistem Berbasis Aturan.

Jenis AI ini menggunakan aturan-aturan yang ditentukan sebelumnya untuk menghasilkan output berdasarkan input yang diberikan. Sistem berbasis aturan terdiri dari himpunan aturan dan basis pengetahuan yang menggambarkan hubungan antara *input* dan *output*. Sistem ini bekerja dengan mencocokkan input dengan aturan yang sesuai untuk menghasilkan *output* yang diinginkan.

2. Sistem Pakar.

Sistem pakar adalah jenis AI yang menggabungkan pengetahuan ahli manusia dengan kemampuan komputasi untuk memecahkan masalah yang kompleks dalam domain tertentu. Sistem ini beroperasi dengan mengumpulkan pengetahuan dari para ahli manusia dan memanfaatkannya dalam membuat keputusan atau memberikan nasihat yang sesuai. Sistem pakar digunakan dalam bidang-bidang seperti kedokteran, keuangan, dan teknik.

3. Pembelajaran Mesin.

Pembelajaran mesin adalah cabang AI yang melibatkan pengembangan algoritma dan model statistik yang memungkinkan sistem untuk belajar dari data dan pengalaman. Dalam pembelajaran mesin, sistem menganalisis data untuk mengidentifikasi pola dan membuat prediksi atau keputusan berdasarkan informasi yang diberikan. Algoritma pembelajaran mesin dapat diklasifikasikan menjadi beberapa kategori, termasuk

pembelajaran terawasi, pembelajaran tak terawasi, dan pembelajaran penguatan.

4. Jaringan Saraf Tiruan.

Jaringan saraf tiruan (*artificial neural networks*) adalah jenis AI yang terinspirasi oleh struktur dan fungsi jaringan saraf dalam otak manusia. Jaringan saraf tiruan terdiri dari banyak unit pemrosesan sederhana yang disebut neuron buatan, yang saling terhubung dan berinteraksi untuk memproses informasi. Jaringan saraf tiruan dapat digunakan untuk tugas-tugas seperti pengenalan pola, pengenalan wajah, dan pemrosesan bahasa alami.

5. AI dalam Robotika.

AI juga digunakan dalam pengembangan robotika, di mana sistem AI dikombinasikan dengan fisik robot untuk menghasilkan perilaku yang cerdas dan adaptif. Robotika AI melibatkan penggunaan algoritma pengolahan sensorik, perencanaan gerakan, dan pembelajaran untuk memungkinkan robot berinteraksi dengan lingkungan sekitarnya dan melakukan tugas-tugas tertentu.

2.4 *Non-player Character (NPC)*

Istilah *Non-player Character (NPC)* menggambarkan semua karakter *game* yang tidak bisa dikendalikan oleh pemain *game*, dalam artian dikendalikan sepenuhnya oleh aplikasi. NPC berperan sebagai karakter yang membantu, membimbing, dan berkomunikasi dengan pemain untuk mencapai tujuan permainan. Dalam beberapa referensi, pembagian NPC dalam game meliputi [12]:

1. *Villain* (NPC Musuh)

Karakter musuh yang berada di dalam permainan, yang secara sistemik dapat menyerang karakter pemain.

2. *Services* NPC

NPC yang memberikan pelayanan kepada karakter utama, seperti melakukan jual beli *item* dengan pemain, membuatkan artifak tertentu, membantu memulihkan energi dan memberikan jasa pelayanan dengan timbal balik variabel tertentu seperti *gold* atau uang.

3. *Story/Quest* NPC

NPC yang akan menunjukkan informasi jalannya cerita. Memberikan latar belakang atau konsep utama yang ada dalam permainan, menunjukkan arah atau petunjuk tertentu, dan sering kali menjadi kunci untuk melanjutkan cerita dalam permainan.

4. *Ordinary* NPC

NPC yang keberadaannya hanya untuk membuat arena permainan tampak aktif. Karakter ini akan bergerak atau diam dan dapat diajak berkomunikasi, namun tidak memberikan informasi kunci yang signifikan untuk perkembangan cerita pemain.

2.5 Metode *Fuzzy Logic*

Metode *Fuzzy Logic* adalah suatu metode kecerdasan buatan yang dikenalkan pertama kali oleh Lothfi A. Zadeh pada tahun 1965. Inti dari *Fuzzy Logic* adalah memetakan ruang *input* ke ruang *output*. Mekanisme utama untuk melakukannya adalah pernyataan *if-then* yang disebut aturan. Semua aturan

dievaluasi secara paralel dan tidak memperhatikan urutannya. Aturan tersebut merujuk pada variabel dan kata sifat yang menggambarkan variabel-variabel tersebut. Sebelum membangun sistem dan menafsirkan aturannya, perlu ditentukan semua istilah yang akan direncanakan untuk digunakan dan kata sifat yang menggambarkan keadaannya. Sebagai contoh, untuk menentukan bahwa robot dapat berjalan “cepat”, harus ditentukan kisaran kecepatan robot yang bervariasi dan memenuhi maksud dengan kata “cepat” [13].

Metode *Fuzzy Logic* melibatkan beberapa tahapan dalam pemrosesan data yang mencakup [14]:

1. Pemodelan variabel.

Identifikasi variabel *input* dan *output* yang akan digunakan dalam sistem *fuzzy*. Variabel *input* mewakili masukan sistem, sedangkan variabel *output* mewakili keluaran yang diharapkan.

2. Fuzzyfikasi.

Proses mengubah data *input* yang bersifat krisis menjadi nilai *Fuzzy*. Hal ini melibatkan penentuan seberapa dekat setiap nilai input ke setiap himpunan *Fuzzy* yang ada dalam domain variabel *input*.

3. Penerapan aturan *Fuzzy*.

Menerapkan aturan-aturan *Fuzzy* yang telah ditetapkan sebelumnya untuk menghubungkan *inputFuzzy* dengan *outputFuzzy*. Aturan-aturan ini biasanya dinyatakan dalam bentuk "aturan-aturan *if-then*" yang menggambarkan logika *Fuzzy* yang digunakan untuk memproses *input*.

4. Inferensi *Fuzzy*.

Proses menghasilkan *outputFuzzy* berdasarkan aturan-aturan yang diterapkan pada nilai *inputFuzzy*. Hal ini melibatkan penggunaan operator logika *Fuzzy* seperti AND, OR, dan NOT untuk menggabungkan aturan-aturan yang relevan.

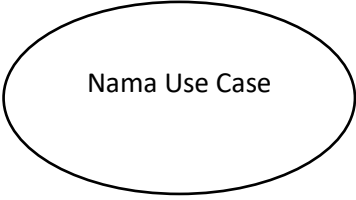
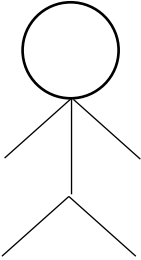
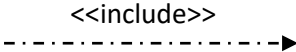
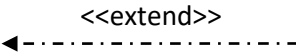
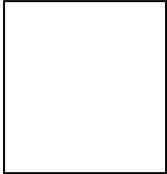
5. *Defuzzyfikasi*.

Mengubah nilai-nilai *Fuzzy output* menjadi nilai konkret atau *crisp* yang dapat digunakan dalam pengambilan keputusan atau tindakan selanjutnya. Proses ini melibatkan penggunaan metode tertentu, seperti centroid atau metode rata-rata tertimbang, untuk mengonversi *outputFuzzy* menjadi nilai yang dapat dipahami.

2.6 *Use Case Diagram*

Use CaseDiagram adalah salah satu alat pemodelan sistem yang merupakan komponen dari *Unified Modeling Language* (UML) yang digunakan untuk menggambarkan interaksi antara pengguna dan sistem. Penggunaan Diagram *Use Case* dapat memberikan gambaran yang efektif dalam menjelaskan konteks suatu sistem dengan jelas, sehingga memperlihatkan batasan sistem tersebut dengan jelas. Terdapat beberapa simbol-simbol dari *Use Case Diagram* seperti yang ditunjukkan pada Tabel 2.1 berikut [15].

Tabel 2.1 Simbol *Use Case Diagram*[15]

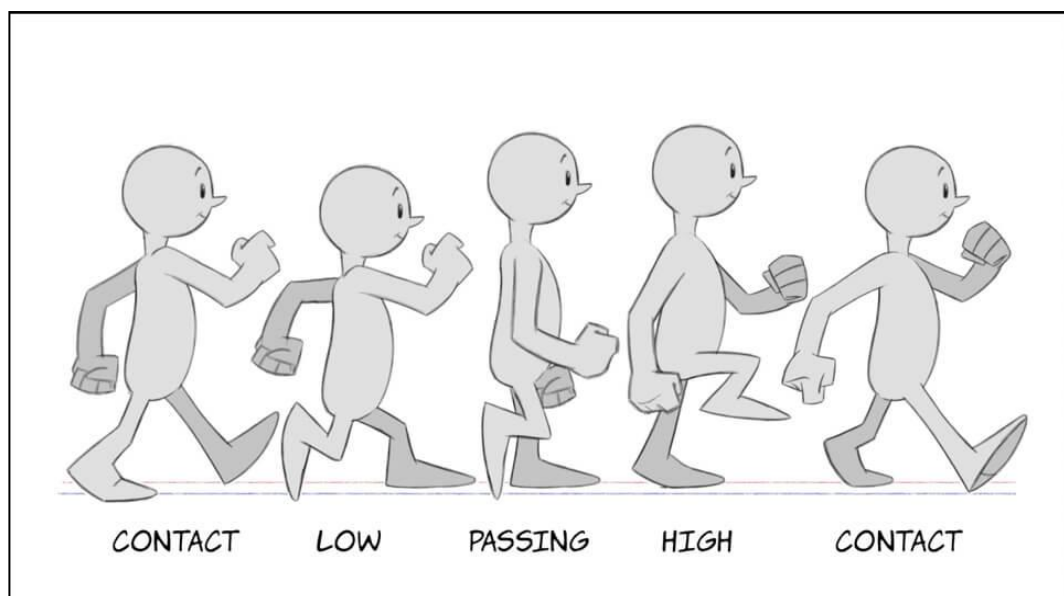
Nama	Simbol	Deskripsi
<i>Use Case</i>		Simbol yang menggambarkan fungsi dari sistem umumnya diekspresikan dalam bentuk kata kerja di awal setiap frasa.
Aktor		Simbol yang mewakili pengguna yang terlibat dalam <i>Use Case</i> biasanya menyatakan orang-orang yang berinteraksi dengan sistem.
Asosiasi		Simbol yang menghubungkan antara <i>Use Case</i> dan aktor menggambarkan komunikasi antara keduanya. Asosiasi digunakan untuk menggambarkan hubungan statis antara elemen-elemen tersebut.
<i>Include</i>		Simbol ini menggambarkan bahwa <i>Use Case</i> secara keseluruhan merupakan bagian dari fungsionalitas <i>Use Case</i> lainnya.
<i>Extend</i>		Simbol yang serupa dengan " <i>include</i> ", namun menunjukkan bahwa sebuah <i>Use Case</i> tambahan dapat berdiri sendiri tanpa membutuhkan <i>Use Case</i> yang ditambahkan.
Sistem		Simbol yang membatasi bahwa setiap interaksi dengan simbol lain harus berada di dalamnya.

2.7 Animasi

Animasi merupakan sarana visual atau audio visual yang membantu dalam mengilustrasikan konsep atau proses yang kompleks sehingga lebih mudah dipahami. Secara dasar, animasi terdiri dari serangkaian gambar yang disusun secara berurutan untuk menciptakan efek gerakan tertentu. Dalam pengkategorian umum, animasi dapat dibagi menjadi dua jenis[16]:

1. *Stop-motion Animation.*

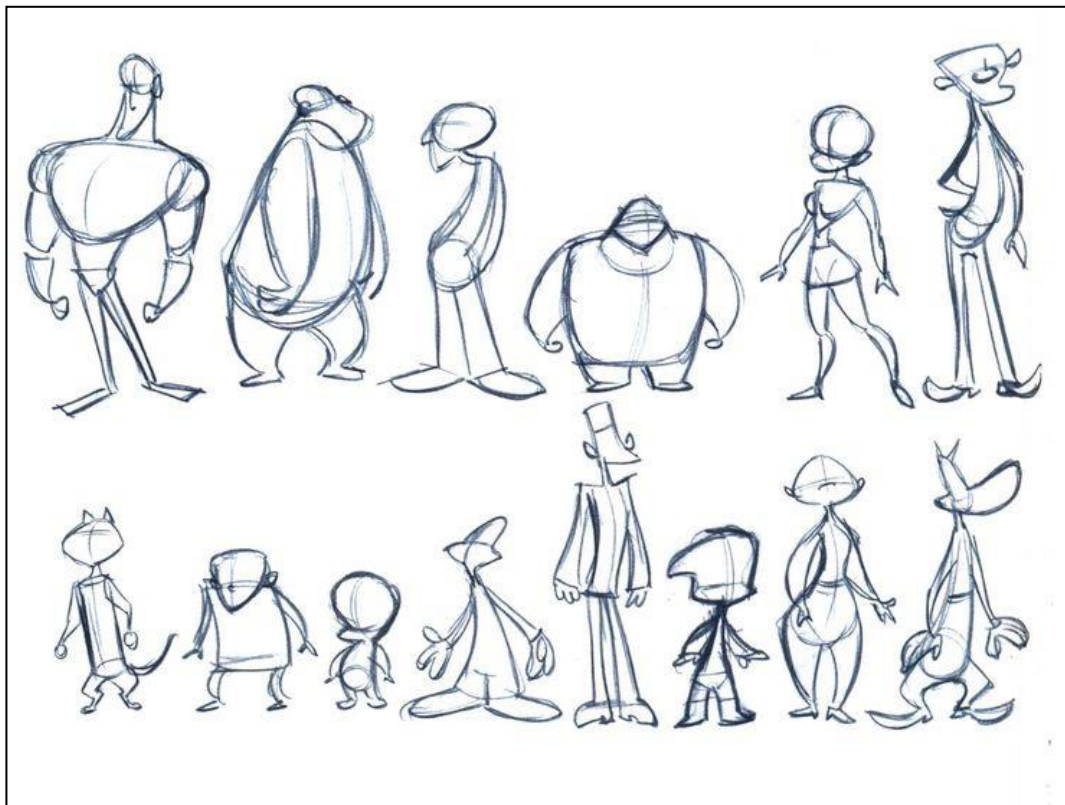
Bisa disebut juga dengan *claymation* karena jenis animasi ini sering menggunakan *clay* (tanah liat) sebagai objeknya. Animasi ini dihasilkan dari pengambilan gambar berupa boneka atau yang lainnya, yang digerakkan secara bertahap. Dalam pengerjaannya, teknik ini memiliki tingkat kesulitan dan memerlukan kesabaran yang tinggi. Salah satu contoh *stop-motion animation* adalah *Chicken Run* karya Nick Parks. Salah satu contoh pengaplikasian *stop-motion animation* ditunjukkan oleh Gambar 2.1 di bawah ini.



Gambar 2.1 Contoh *Stop-motion Animation* [16]

2. *Traditional Animation.*

Dinamakan tradisional karena teknik animasi inilah yang digunakan pada saat animasi pertama kali dikembangkan. Sering disebut juga dengan *cell animation* karena pengerjaannya dilakukan pada *celluloid transparent* yang sekilas mirip dengan transparansi OHP. Pada pembuatan animasi tradisional, setiap gerakan dari objek digambar satu persatu di atas *cell*. Salah satu contoh pengaplikasian *traditional animation* ditunjukkan oleh Gambar 2.2 di bawah ini.



Gambar 2.2*Traditional Animation* [16]

3. *Computer Animation.*

Secara keseluruhan animasi ini dikerjakan dengan bantuan komputer, mulai dari pembuatan karakter, pengaturan gerakan objek dan kamera, pemberian suara, dan *special effect*. Dengan animasi komputer, hal-hal

yang awalnya tidak mungkin digambarkan menjadi mungkin dan lebih mudah. Namun, animasi yang dihasilkan tergantung keahlian yang dimiliki dan *software* yang digunakan. Salah satu contoh pengaplikasian *computer animation* ditunjukkan oleh Gambar 2.3 di bawah ini.



Gambar 2.3 Computer Animation [16]

Selain itu, berdasarkan jenis dimensi, animasi dibagi menjadi 2 kelompok dimensi antara lain [16]:

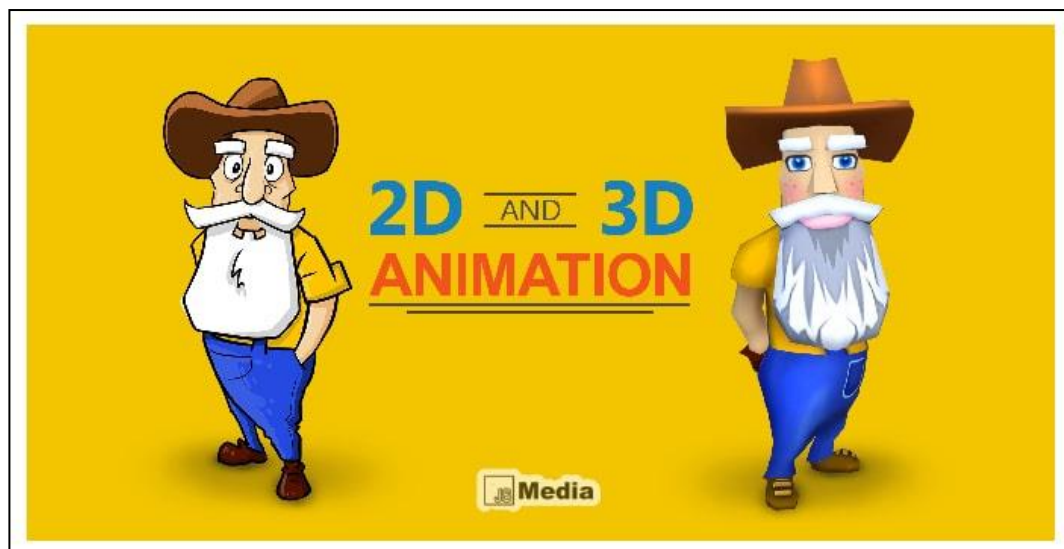
1. Animasi 2D.

Animasi 2D adalah jenis animasi dalam bentuk dua dimensi, artinya animator 2D membuat gambar dan karakter dalam format dua dimensi dan menghidupkannya dengan gerakan. Jenis animasi ini dianggap sebagai bentuk animasi tradisional dengan ciri karakter polos, tidak bervolume, dan hanya bergerak ke atas, bawah, kiri dan kanan. Contoh animasi 2D ditemukan pada serial kartun yang sering Anda tonton semasa kecil,

seperti kartun Mickey Mouse, Tom & Jerry, Doraemon, atau Spongebob Squarepants.

2. Animasi 3D.

Animasi 3D adalah seni untuk menciptakan gambar bergerak dalam ruang digital 3 dimensi. Melalui manipulasi objek atau model 3D dalam sebuah software untuk mengolah dan membuat animasi, *animator* mengurutkan gambar yang akan memberikan ilusi gerakan. Animasi 3D akan terlihat sangat realistis dan semua prosesnya dilakukan di komputer. Proses membuat animasi 3D umumnya dapat dibagi ke tiga tahap, yaitu *modelling*, *layout and animation*, dan *rendering*. *Modelling* adalah proses pembuatan objek 3D dalam suatu adegan di komputer. *Layout and animation* yaitu proses memposisikan objek dan membuat objek 3D bergerak. Kemudian proses selanjutnya adalah *rendering*, yaitu mengolah semua data di proses sebelumnya ke dalam suatu hasil akhir. Perbedaan dari animasi 2D dan 3D ditunjukkan oleh Gambar 2.4 di bawah ini.



Gambar 2.4 Perbedaan Animasi 2D dan 3D [16]

Terdapat beberapa konsep dasar dari sebuah animasi diuraikan sebagai berikut [16]:

1. Prinsip dasar animasi:
 - a. *Squash and Stretch*: Teknik yang menambahkan deformasi pada objek untuk memberikan kesan elastisitas dan berat.
 - b. *Anticipation*: Mengatur gerakan pra-aksi untuk memberi petunjuk kepada penonton tentang gerakan yang akan datang.
 - c. *Staging*: Menata adegan agar objek yang penting berada dalam fokus penonton.
 - d. *Straight Ahead Action and Pose to Pose*: Dua pendekatan umum dalam pembuatan animasi. "*Straight Ahead Action*" adalah membuat setiap frame secara berurutan, sementara "*Pose to Pose*" adalah membuat frame kunci dan kemudian mengisi frame di antara mereka.
 - e. *Follow Through and Overlapping Action*: Menggambarkan bagian dari objek yang terus bergerak bahkan setelah gerakan utama telah berhenti, serta bagian dari objek yang bergerak dengan kecepatan atau arah yang berbeda dari bagian lainnya.
 - f. *Arcs*: Gerakan alami sering mengikuti pola melingkar atau lengkungan.
 - g. *Slow In and Slow Out*: Objek cenderung memulai gerakan perlahan, mengambil kecepatan puncaknya, dan kemudian melambat saat berhenti.
 - h. *Exaggeration*: Meningkatkan atau mengurangi proporsi dan gerakan untuk menambah dramatis atau komedi.

2. Prinsip Fisika:

- a. *Gravity*: Menyertakan gaya gravitasi dalam gerakan objek untuk memberikan kesan keberatan.
- b. *Momentum*: Objek akan terus bergerak kecuali ada kekuatan yang menentang.
- c. *Friction*: Mengatur kecepatan atau perubahan arah gerakan berdasarkan gesekan dengan permukaan atau objek lainnya.

3. Aspek Teknis:

- a. *Frame rate*: Kecepatan dengan mana frame dijalankan per detik untuk menciptakan ilusi gerakan yang mulus.
- b. *Timing*: Pengaturan durasi dan jeda antara gerakan untuk mengkomunikasikan kecepatan dan ritme.

4. *Storyboarding* dan *Timing*:

- a. *Storyboard*: Serangkaian gambar yang menggambarkan urutan adegan untuk membantu mengatur plot dan komposisi.
- b. *Timing*: Mengatur durasi setiap adegan atau gerakan untuk mencapai efek yang diinginkan.

2.8 *Storyboard*

Storyboard adalah alat pra-visualisasi yang dirancang untuk memberikan serangkaian gambar berurutan bingkai demi bingkai, shot demi shot yang diadaptasi dari terjemahan naskah (skrip). *Storyboard* adalah konsep gambar yang menjelaskan dan menambah narasi naskah dan memungkinkan seluruh tim

produksi mengatur semua tindakan rumit yang diperlukan oleh naskah sebelum pembuatan film/animasi yang sebenarnya[17].

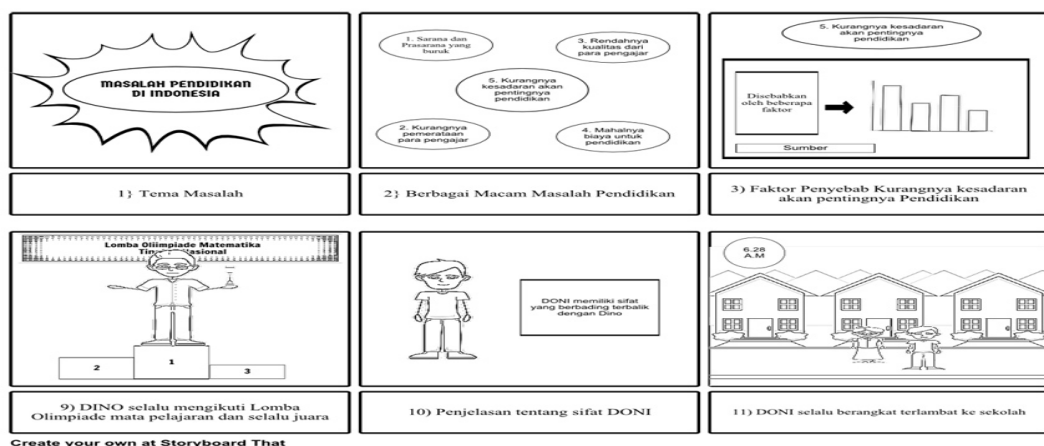
Terdapat elemen-elemen konstruktif pada *storyboard* yang berperan penting dalam pembuatan animasi, yaitu [17]:

1. Perspektif.
2. *Blocking dan Point of Interest*.
3. *Framing*.
4. *Direction*.
5. *Angle*.
6. Kreatif dan Komunikatif.

Terdapat 2 tahapan dalam membuat *storyboard* yaitu [17]:

1. Persiapan, mencakup diskusi, penentuan konsep atau ide, riset, membuat karakter berdasarkan cerita yang nantinya dibuat oleh *character development*, dan membuat naskah atau skenario.
2. Menggambar, yang di mana perancang *storyboard* mulai menuangkan isi naskah ke dalam bentuk gambar yang dibuat *per-frame*.

Berikut ini, Gambar 2.5 menunjukkan contoh dari *storyboard*.



Gambar 2.5 Contoh Storyboard[17]

BAB III

METODE PENELITIAN

3.1 Analisis Masalah

Pada sub bab ini, akan dijelajahi dengan cermat serta mendalam mengenai analisis masalah yang dihadapi dalam konteks yang relevan. Sebuah pemahaman yang mendalam tentang masalah merupakan fondasi yang penting dalam merancang solusi yang efektif dan berkelanjutan. Pada sub bab ini akan dibedah akar penyebab masalah, mengeksplorasi dampaknya secara menyeluruh, serta mengidentifikasi faktor-faktor yang terlibat. Analisis masalah pada penelitian ini dibagi menjadi 3 yaitu analisis proses yang berjalan sesuai di lapangan, analisis metode yang diusulkan, dan analisis sistem usulan.

3.1.1 Analisis Proses Yang Berjalan Sesuai di Lapangan

Analisis proses yang berjalan sesuai di lapangan adalah langkah penting dalam memahami efektivitas dan kesesuaian implementasi suatu proyek dengan kondisi nyata. Dengan menguraikan proses-proses yang terjadi di lapangan, kita dapat mengevaluasi sejauh mana pencapaian tujuan dan manfaat yang diharapkan dari sebuah proyek. Dalam konteks penelitian ini, analisis proses yang berjalan di lapangan akan memberikan wawasan yang berharga tentang bagaimana game "*Campus Adventure*" dan metode *Fuzzy Logic* diimplementasikan, serta bagaimana dampaknya terhadap target pengguna, yaitu calon mahasiswa.

1. Evaluasi Implementasi Game "Campus Adventure"
 - a. Pengamatan langsung terhadap penggunaan *game* "Campus Adventure" oleh target pengguna, yakni calon mahasiswa.
 - b. Analisis terhadap interaksi pengguna dengan berbagai fitur dan *level* yang disediakan dalam *game*.
 - c. Pengumpulan data mengenai tingkat keterlibatan dan kepuasan pengguna terhadap pengalaman bermain *game*.
2. Penilaian Efektivitas Metode *Fuzzy Logic*
 - a. Studi kasus terkait implementasi metode *Fuzzy Logic* dalam menciptakan musuh NPC dalam *game*.
 - b. Evaluasi kemampuan musuh NPC dalam memberikan tantangan yang menarik bagi pemain.
 - c. Pengamatan terhadap adaptabilitas keputusan yang dihasilkan oleh NPC berdasarkan kondisi yang berubah di dalam *game*.
3. Analisis Dampak Terhadap Pengguna
 - a. Survei atau wawancara dengan pengguna untuk mengetahui perubahan persepsi dan keterampilan yang terjadi setelah bermain *game* "Campus Adventure".
 - b. Evaluasi terhadap kemampuan pengguna dalam menghadapi tantangan-tantangan yang serupa di dunia nyata setelah menggunakan *game* ini.

Dengan memahami proses yang berjalan di lapangan, kita dapat mendapatkan pemahaman yang lebih mendalam tentang efektivitas dan relevansi *game* "Campus Adventure" serta metode *Fuzzy Logic* dalam memberikan solusi

terhadap tantangan-tantangan yang dihadapi oleh calon mahasiswa. Analisis ini akan membantu dalam pengembangan dan peningkatan lebih lanjut dalam pengembangan game “*Campus Adventure*” pada sub bab berikutnya.

Konsep game berupa sebuah game petualangan dimana terdapat seorang karakter berupa mahasiswa yang terjebak dalam dimensi zaman dahulu dimana mahasiswa tersebut harus menyelesaikan misi-misi tertentu dimana ketika menjalankan misinya terdapat beberapa musuh seperti malas, konflik dengan teman, dan manajemen waktu yang harus dihadapi dan dihindari. Ketika seluruh misi berhasil diselesaikan maka karakter utama akan kembali ke dunianya dan menjalani wisuda.

3.1.2 Analisis Metode Yang Digunakan

Pada penelitian ini untuk membuat game “*Campus Adventure*” yang menarik maka diimplementasikan metode *Fuzzy Logic* untuk menciptakan musuh *Artificial Intelligence* (AI) yang pintar sehingga memberikan tantangan bagi pemain. Berikut ini uraian cara kerja dari metode *Fuzzy Logic* dalam menciptakan karakter NPC cerdas diasumsikan pada karakter NPC malas antara lain:

1. Pendefinisian Variabel *Input* dan *Output*:

Variabel *Input*:

Jarak antara karakter *player* dan *enemy* Malas.

- a. Rentang jarak: $[0, \infty]$
- b. Fungsi keanggotaan: Dekat (D), Sedang (S), Jauh (J)

Variabel *Output*:

Aksi yang akan diambil oleh *enemy* Malas (*Chase, Approach, Fight, Patrol*).

- a. Rentang variabel *output*: [0, 1] (dimana 0 menunjukkan tidak melakukan aksi dan 1 menunjukkan melakukan aksi sepenuhnya).
- b. Fungsi keanggotaan: Kurang (K), Sedang (S), Banyak (B)

2. *Fuzzifikasi*:

Variabel *input* "Jarak" akan diubah menjadi nilai *Fuzzy* menggunakan fungsi keanggotaan untuk setiap label (D, S, J) berdasarkan jarak aktual antara Player dan Enemy Malas.

3. *Aturan Fuzzy*:

Aturan *Fuzzy* akan menentukan aksi yang harus diambil berdasarkan kondisi input. Berdasarkan deskripsi yang diberikan:

- a. Jika $\text{jarak} \leq 15$ pixel, maka label "Dekat" (D) akan diaktifkan.
- b. Jika $15 < \text{jarak} \leq 10$ pixel, maka label "Sedang" (S) akan diaktifkan.
- c. Jika $10 < \text{jarak} \leq 5$ pixel, maka label "Jauh" (J) akan diaktifkan.

4. *Inferensi Fuzzy*:

Setiap aturan *fuzzy* akan memberikan kontribusi pada nilai keanggotaan untuk setiap aksi (*Chase, Approach, Fight, Patrol*) berdasarkan tingkat keanggotaan dari label (D, S, J).

5. *Defuzzifikasi*:

- a. Nilai keanggotaan untuk setiap aksi akan digabungkan untuk mendapatkan nilai *crisp* yang mewakili aksi yang harus diambil.
- b. Metode defuzzifikasi, misalnya menggunakan metode *centroid*, akan digunakan untuk mengonversi nilai keanggotaan menjadi nilai *crisp*.

Contoh perhitungan:

Misalkan jarak antara *player* dan *enemy* malas adalah 12 *pixel*.

Variabel *input* "Jarak":

Fungsi keanggotaan:

1. D: $\mu(D) = 0$
2. S: $\mu(S) = (15 - 12) / (15 - 10) = 0.75$
3. J: $\mu(J) = 0$

Variabel *output* "Aksi":

1. Aksi "*Chase*":
 - a. $\mu(K) = 1 - 0.75 = 0.25$
 - b. $\mu(S) = 0.75$
 - c. $\mu(B) = 0$
2. Aksi "*Approach*":
 - a. $\mu(K) = 1 - 0.75 = 0.25$
 - b. $\mu(S) = 0.75$
 - c. $\mu(B) = 0$
3. Aksi "*Fight*":
 - a. $\mu(K) = 1 - 0.75 = 0.25$
 - b. $\mu(S) = 0.75$
 - c. $\mu(B) = 0$
4. Aksi "*Patrol*":
 - a. $\mu(K) = 1$
 - b. $\mu(S) = 0$
 - c. $\mu(B) = 0$

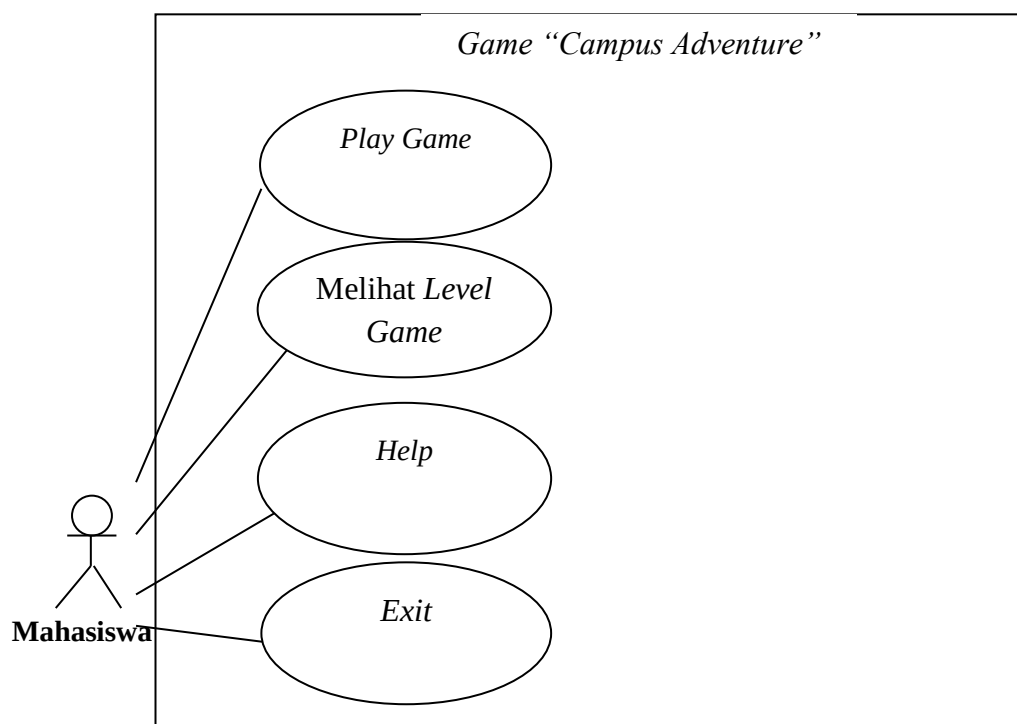
Defuzzifikasi:

Nilai *crisp* untuk setiap aksi akan dihitung menggunakan metode *centroid*, kemudian aksi dengan nilai tertinggi akan dipilih sebagai tindakan yang akan diambil oleh *enemy* Malas.

Dengan cara ini, *enemy malas* dapat secara adaptif menentukan tindakan yang harus diambil berdasarkan jarak aktual antara *player* dan dirinya sendiri, sesuai dengan *Fuzzy Logic* yang diterapkan.

3.1.3 Analisis Sistem Usulan

Pemodelan sistem usulan menggunakan *Use Case Diagram* seperti yang ditunjukkan pada Gambar 3.1.



Gambar 3.1 Use Case Diagram Game "Campus Adventure"

Selanjutnya akan dijabarkan secara detil skenario mengenai langkah-langkah dari setiap *Use Case* pada *game*. Berikut ini adalah seluruh skenario (*flow of event*) yang menggambarkan urutan interaksi aktor dengan *game* usulan dari *Use Case* tersebut, sebagaimana dijabarkan dari Tabel 3.1 hingga Tabel 3.4.

Tabel 3.1 Skenario *Use Case Play Game*

Identifikasi	
Nomor	1
Nama	<i>Play Game</i>
Tujuan	Memulai untuk bermain <i>game</i>
Aktor	Mahasiswa
Aksi Aktor	Reaksi Sistem
1. Mahasiswamengklik tombol “ <i>PLAY GAME</i> ”	
	2. Sistem akan menampilkan tampilan prolog cerita <i>scene game</i> pengenalan awal
3. Mahasiswa mengklik tombol “ <i>PLAY GAME</i> ”	
	4. Sistem akan melanjutkan prolog cerita <i>scene game</i> hingga menampilkan tampilan permainan dimulai

Tabel 3.2 Skenario *Use Case Melihat Level Game*

Identifikasi	
Nomor	2
Nama	Melihat <i>Level Game</i>
Tujuan	Melihat informasi <i>level game</i> yang harus diselesaikan untuk melanjutkan <i>adventure</i>

	selanjutnya
Aktor	Mahasiswa
Aksi Aktor	Reaksi Sistem
1. Mahasiswahas menekan tombol ikon berbentuk dokumen di sebelah sudut kanan <i>game</i>	
	2. Sistem akan menampilkan tampilan <i>level</i> <i>game</i> yang harus diselesaikan

Tabel 3.3 Skenario Use Case Help

Identifikasi	
Nomor	3
Nama	<i>Help</i>
Tujuan	Mempelajari cara untuk memainkan <i>game</i>
Aktor	Mahasiswa
Aksi Aktor	Reaksi Sistem
1. Mahasiswahas menekan tombol ikon berbentuk tanda tanya di sebelah sudut kiri <i>game</i>	
	2. Sistem akan menampilkan tampilancara memainkan <i>game</i>

Tabel 3.4 Skenario *Use CaseExit*

Identifikasi	
Nomor	4
Nama	<i>Exit</i>
Tujuan	Memberhentikan <i>game</i> atau keluar dari <i>game</i>
Aktor	Mahasiswa
Aksi Aktor	Reaksi Sistem
1. Mahasiswamenekan tombol “ <i>Exit</i> ”	
	2. Sistem akan memberhentikan <i>game</i> dan keluar dari sistem

3.2 Perancangan

3.2.1 Perancangan Karakter

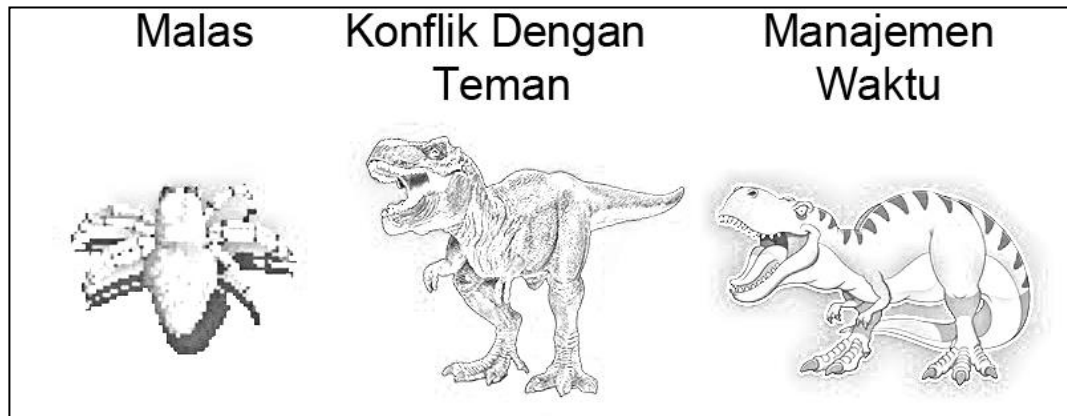
Berikut ini diuraikan perancangan karakter pada *game* yang dibangun pada penelitian ini.

1. Karakter Utama



Gambar 3.2 Rancangan Karakter Utama

2. Karakter Musuh



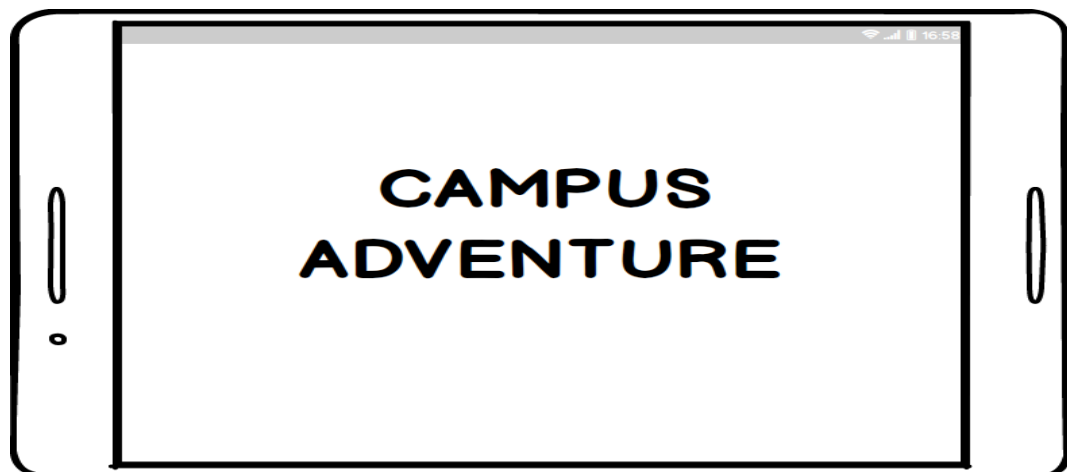
Gambar 3.3 Rancangan Karakter Musuh

3.2.2 Perancangan *Storyboard*

Pada bagian ini, akan digambarkan *storyboard* dari game usulan yang akan dibangun. Berikut ini adalah keseluruhan perancangan *storyboard* dari game “*Campus Adventure*” yang akan dibangun antara lain:

1. Rancangan Tampilan *Splash Screen*

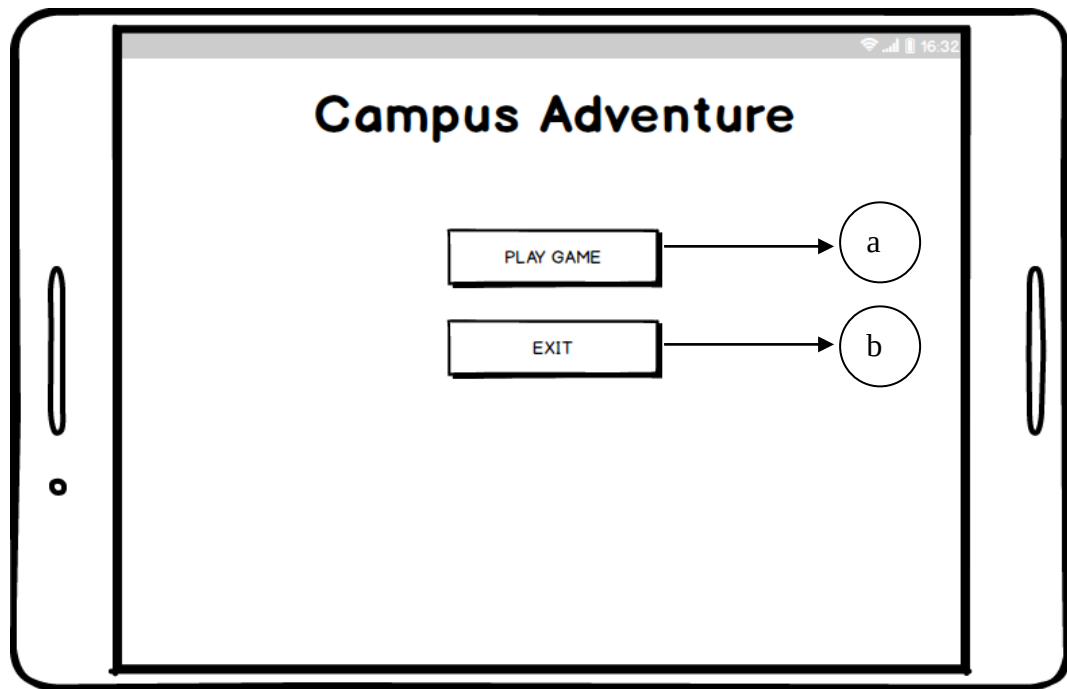
Tampilan *splash screen* berisi informasi pengenalan pembuat *game* berupa logo dari pembuat *game*, seperti tampak pada Gambar 3.4 berikut ini.



Gambar 3.4 Rancangan Tampilan *Splash Screen*

2. Rancangan Tampilan *Scene* Awal Menu *Game*

Tampilan berisikan *scene* pilihan yang berisikan menu-menu yang tersedia di *game*, seperti tampak pada Gambar 3.5 berikut ini.



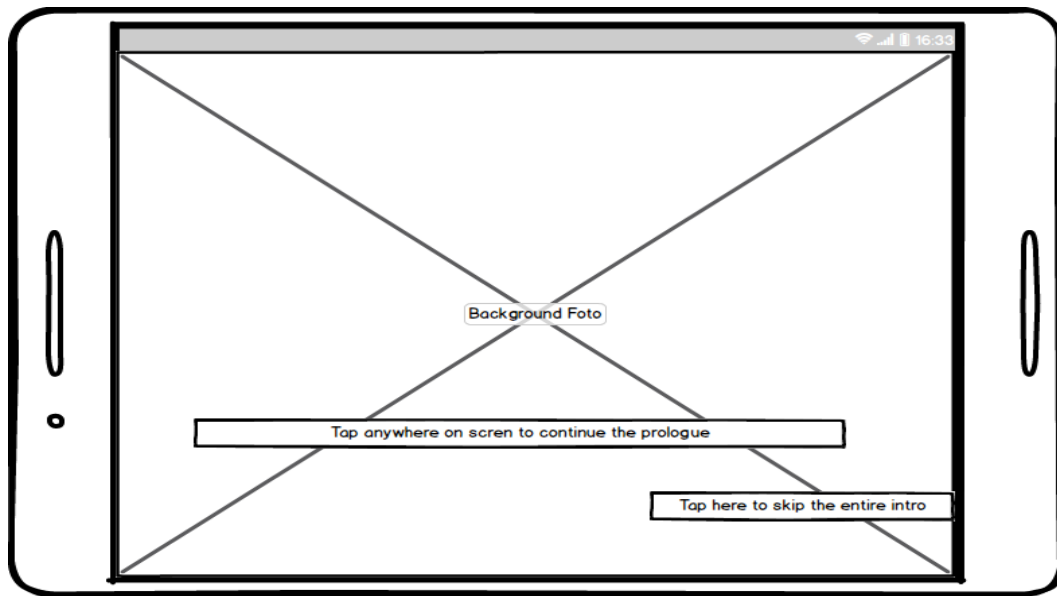
Gambar 3.5Rancangan Tampilan *Scene* Awal Menu *Game*

Keterangan:

- a. Tombol untuk memulai permainan.
- b. Tombol untuk keluar dari *game*.

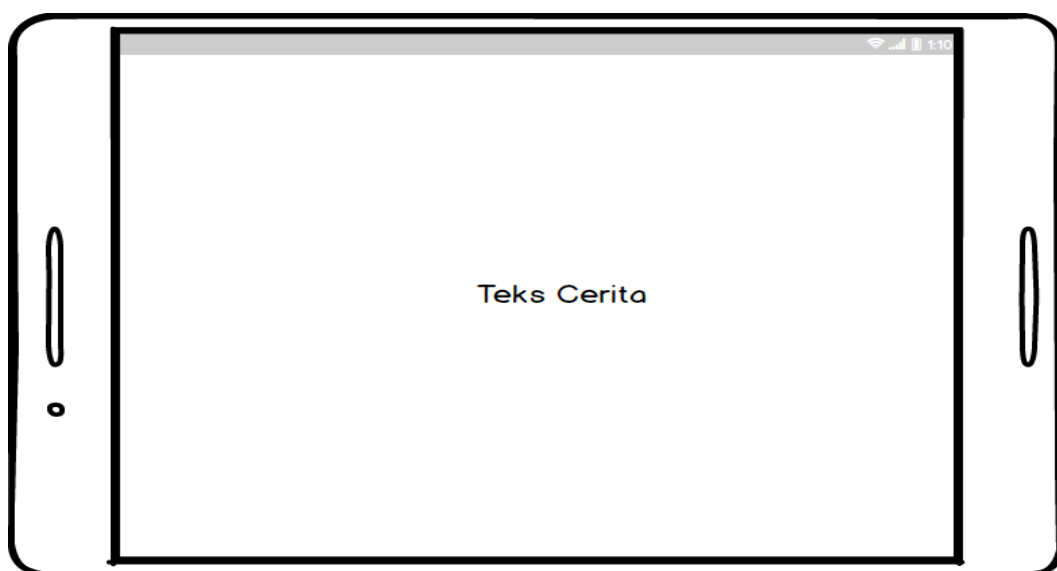
3. Rancangan *Storyboard Prolog Game*

Tampilan berisikan prolog pengenalan awal cerita *game* yang mengenalkan cerita tentang mahasiswa baru di kampus STMIK TIME, seperti tampak pada Gambar 3.6 berikut ini.



Gambar 3.6Rancangan *Storyboard Prolog Game*

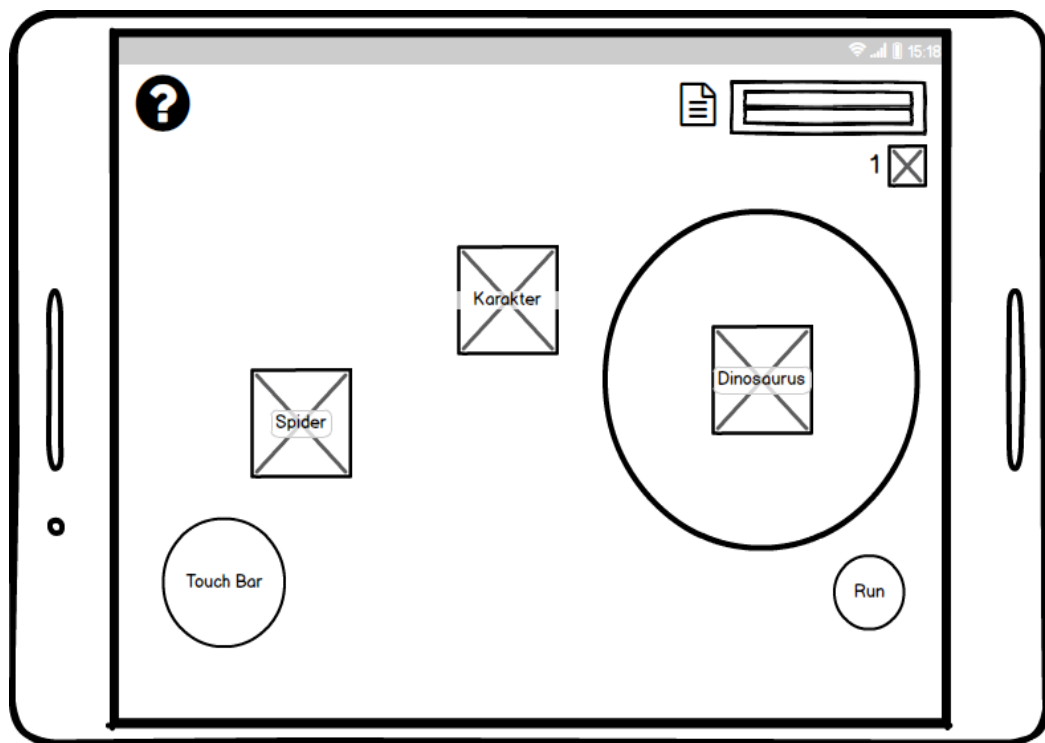
Apabila tombol *tap* ditekan maka akan lanjut ke cerita berikutnya seperti tampak pada Gambar 3.7 berikut ini.



Gambar 3.7 Rancangan *Storyboard Lanjutan Prolog Game*

4. Rancangan Tampilan *Scene* Permainan Dimulai

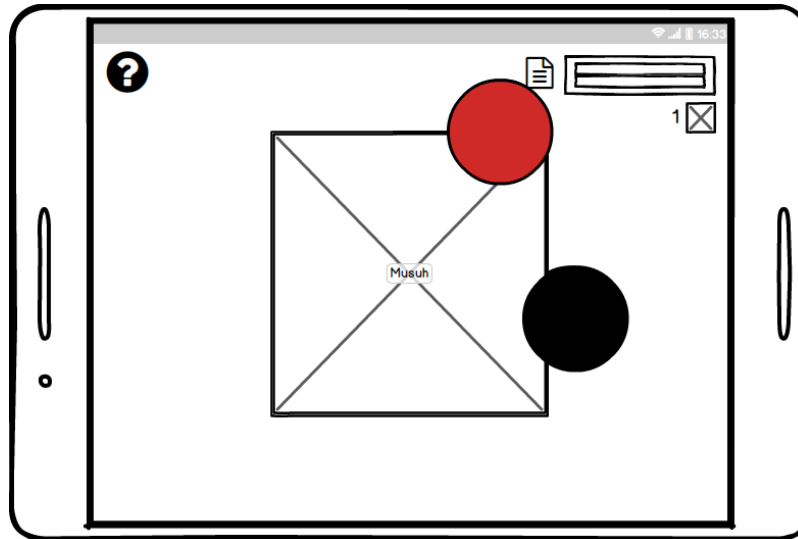
Tampilan *game* dimulai dimana mahasiswa dapat memainkan karakternya. Karakter musuh malas, konflik dengan teman, dan manajemen waktu akan mengganggu mahasiswa menyelesaikan *quest level* yang ada sehingga mahasiswa harus mampu membasi rasa malas, konflik dengan teman, dan manajemen waktu. Pada tampilan ini juga setiap perilaku dari karakter musuh NPC dimodelkan dengan metode *Fuzzy Logic* sehingga secara otomatis setiap gerakan dan tindakan yang terjadi berdasarkan hasil pengolahan dari metode *Fuzzy Logic*, seperti tampak pada Gambar 3.8 berikut ini.



Gambar 3.8 Rancangan Tampilan *Scene* Permainan Dimulai

5. Rancangan Tampilan *Scene Help*

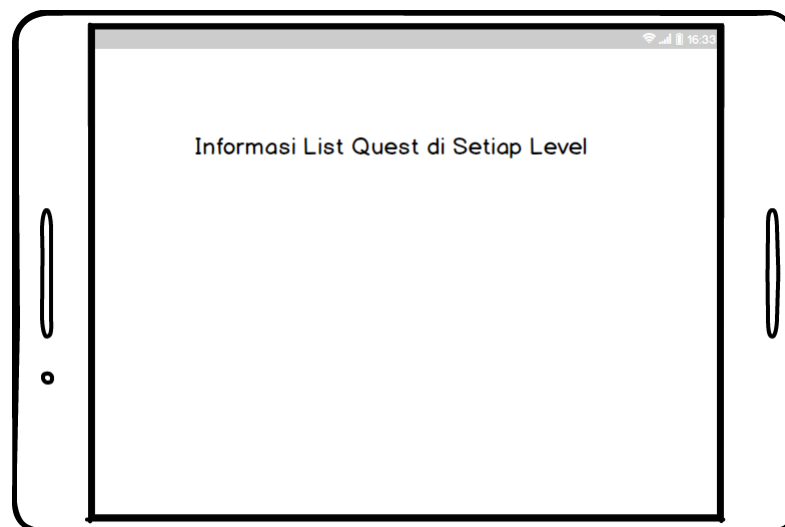
Tampilan yang berisikan bantuan cara memainkan *game*, seperti tampak pada Gambar 3.9 berikut ini.



Gambar 3.9Rancangan Tampilan *Scene Help*

6. Rancangan Tampilan *SceneLevel Game*

Tampilan yang berisikan *list level game* yang harus diselesaikan, seperti tampak pada Gambar 3.10 berikut ini.



Gambar 3.10Rancangan Tampilan *SceneList Quest*

BAB IV

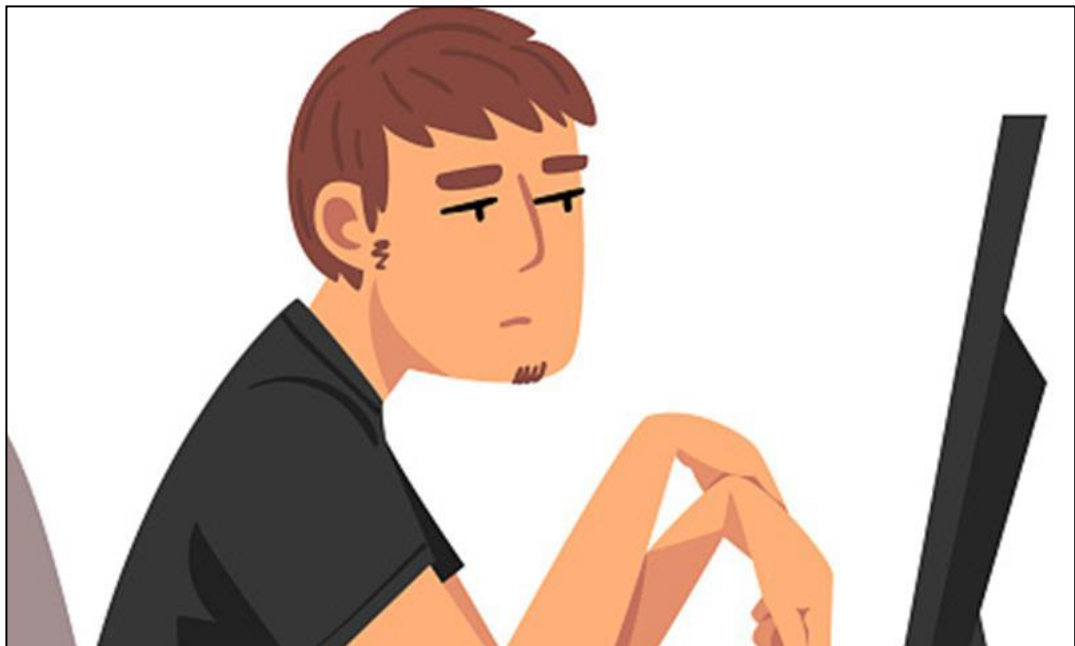
HASIL DAN PEMBAHASAN

4.1 Hasil

Hasil penelitian berupa dibangunnya sebuah *game* bernama “*Campus Adventure*” menggunakan metode *Fuzzy Logic*. Pertama-tama diuraikan seluruh hasil dari karakter *game* pada penelitian ini yang terlibat dalam *game* “*Campus Adventure*” yang dibangun antara lain:

1. Karakter Utama

Objek karakter utama dalam perancangan aplikasi permainan ini digambarkan seperti tampak pada Gambar 4.1 di bawah ini.



Gambar 4.1 Karakter Utama

2. Karakter Musuh

Objek karakter musuh dalam perancangan aplikasi permainan ini digambarkan seperti tampak pada Gambar 4.2 di bawah ini.



Gambar 4.2 Karakter Musuh

Selanjutnya diuraikan keseluruhan hasil tampilan dari *game* “*Campus Adventure*” yang akan dibangun antara lain:

1. Tampilan *Splash Screen*

Tampilan *splash screen* berisi informasi pengenalan pembuat *game* berupa logo dari pembuat *game*, seperti tampak pada Gambar 4.3 berikut ini.



Gambar 4.3 Tampilan *Splash Screen*

2. Tampilan Scene Awal Menu Game

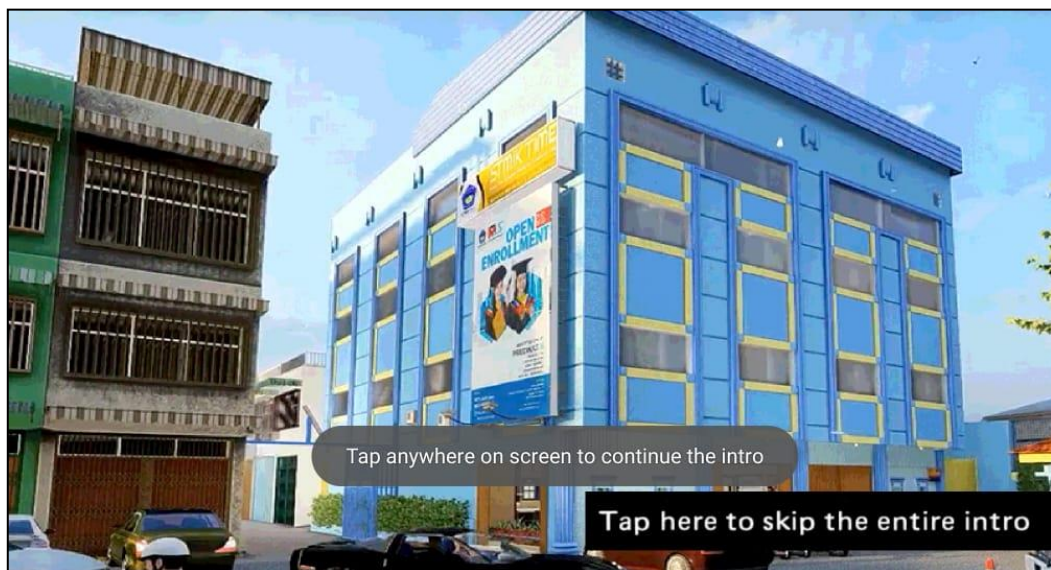
Tampilan berisikan *scene* pilihan yang berisikan menu-menu yang tersedia di *game*, seperti tampak pada Gambar 4.4 berikut ini.



Gambar 4.4Tampilan Scene Awal Menu Game

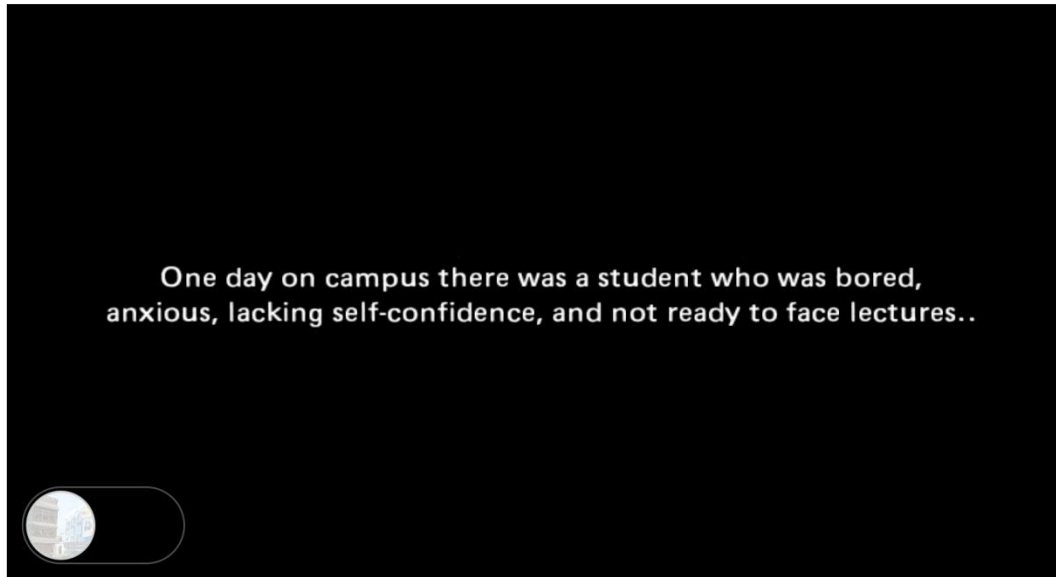
3. Storyboard Prolog Game

Tampilan berisikan prolog pengenalan awal cerita *game* yang mengenalkan cerita tentang mahasiswa baru di kampus STMIK TIME, seperti tampak pada Gambar 4.5 berikut ini.



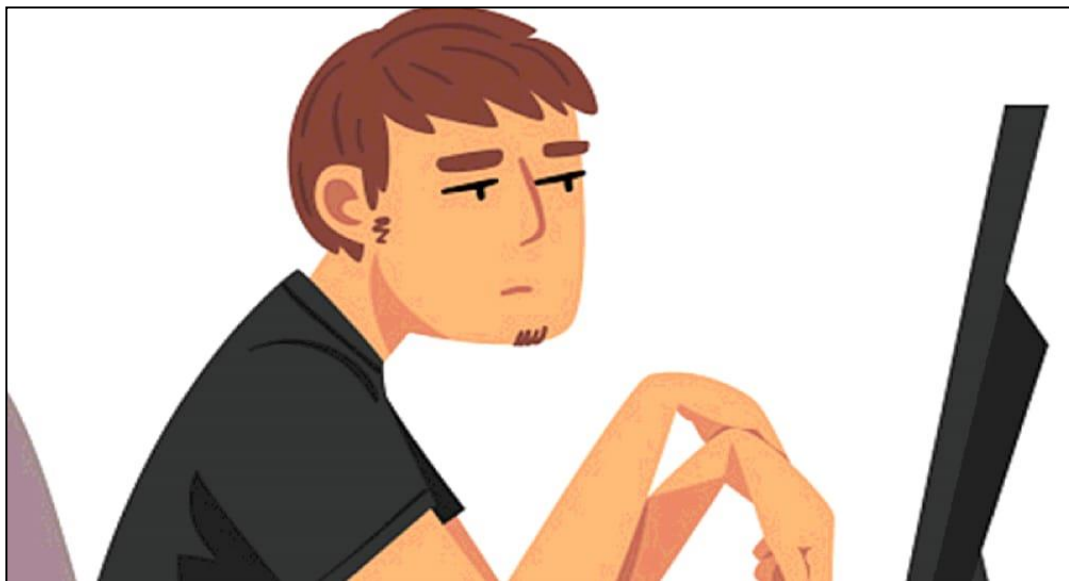
Gambar 4.5Tampilan Prolog Game

Apabila tombol *tap* ditekan maka akan lanjut ke cerita berikutnya seperti tampak pada Gambar 4.6 berikut ini.



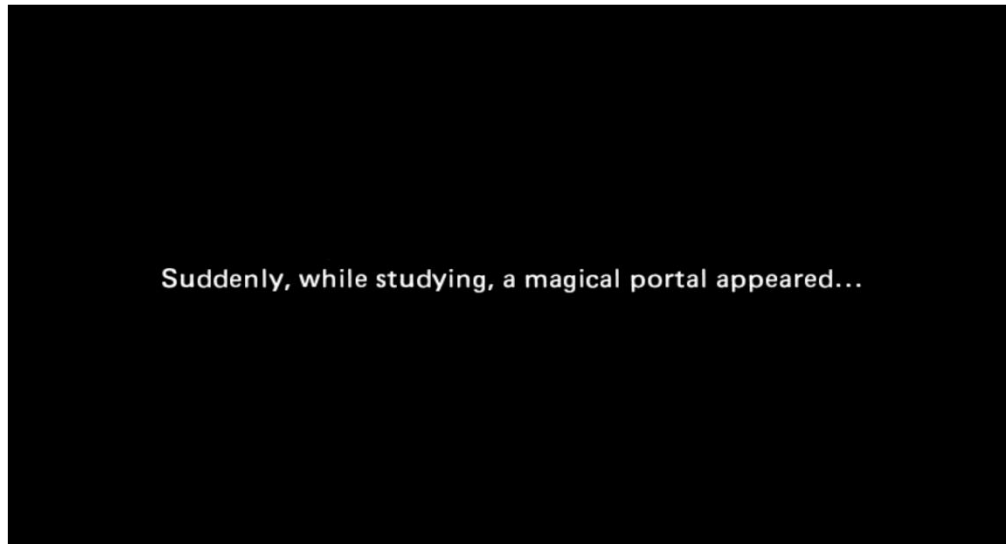
Gambar 4.6 Tampilan Lanjutan Prolog *Game-1*

Ketika layar ditekan menuju ke prolog berikutnya berupa gambar seorang mahasiswa yang sedang tidak siap menghadapi perkuliahan, seperti tampak pada Gambar 4.7 berikut ini.



Gambar 4.7 Tampilan Lanjutan Prolog *Game-2*

Ketika layar ditekan menuju ke prolog berikutnya berupa teks seorang mahasiswa yang sedang tidak siap menghadapi perkuliahan, seperti tampak pada Gambar 4.8 berikut ini.



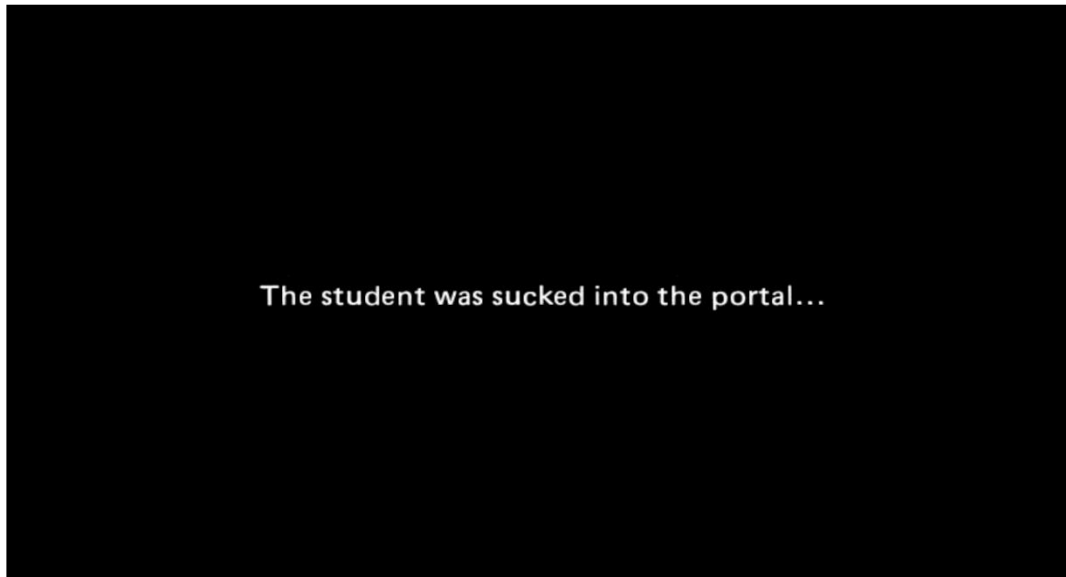
Gambar 4.8 Tampilan Lanjutan Prolog *Game-3*

Ketika layar ditekan menuju ke prolog berikutnya berupa gambar di lab komputer yang muncul sebuah portal ajaib, seperti tampak pada Gambar 4.9 berikut ini.



Gambar 4.9 Tampilan Lanjutan Prolog *Game-4*

Ketika layar ditekan menuju ke prolog berikutnya berupa teks bahwa mahasiswa tersebut tersedot oleh portal tersebut, seperti tampak pada Gambar 4.10 berikut ini.



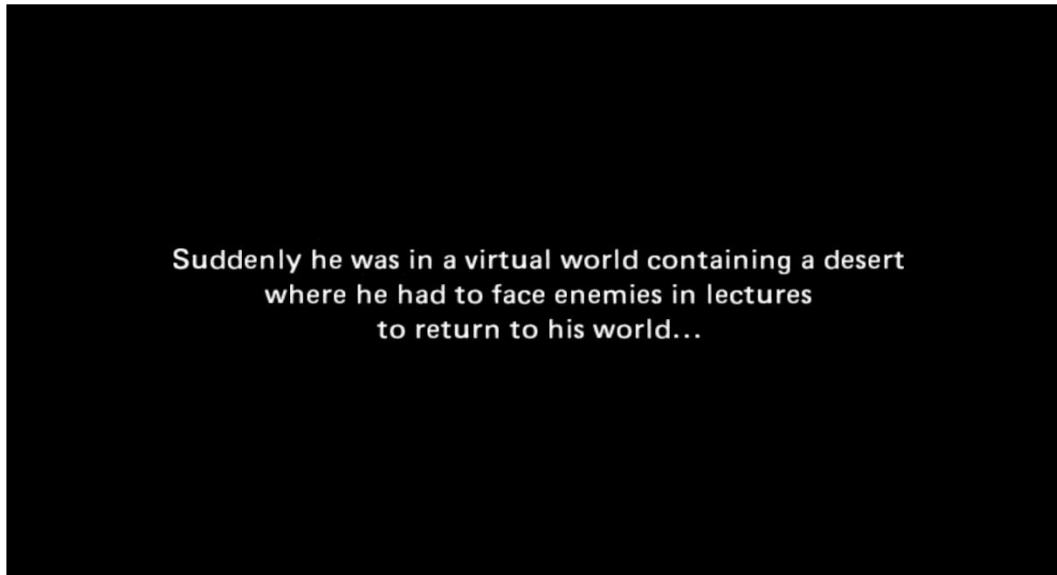
Gambar 4.10 Tampilan Lanjutan Prolog *Game-5*

Ketika layar ditekan menuju ke prolog berikutnya berupa gambar mahasiswa yang tersedot ke portal dan menuju dimensi lain, seperti tampak pada Gambar 4.11 berikut ini.



Gambar 4.11 Tampilan Lanjutan Prolog *Game-6*

Ketika layar ditekan menuju ke prolog berikutnya berupa teks bahwa mahasiswa tersebut tiba-tiba berada di sebuah dimensi virtual berupa padang pasir yang sangat luas, seperti tampak pada Gambar 4.12 berikut ini.



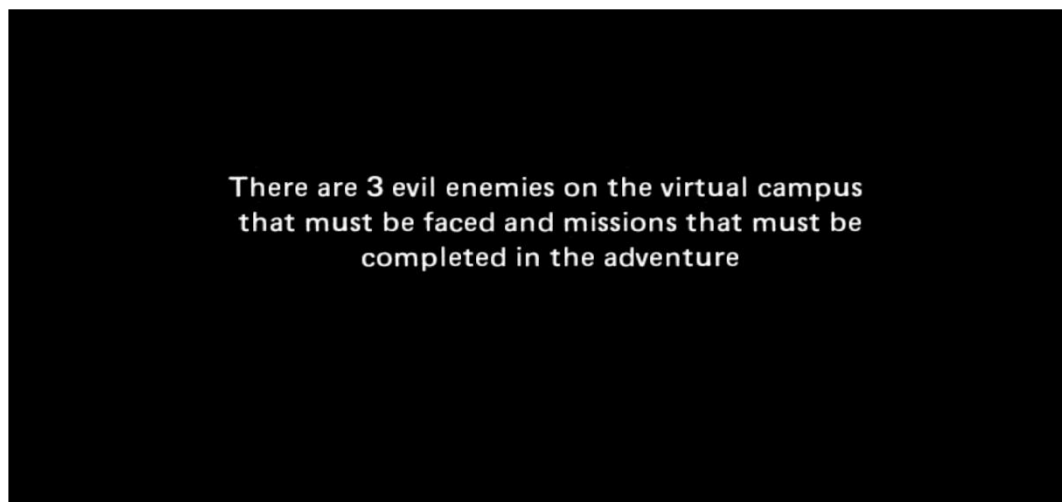
Gambar 4.12 Tampilan Lanjutan Prolog *Game-7*

Ketika layar ditekan menuju ke prolog berikutnya berupa gambar mahasiswa yang terjebak di dunia virtual dengan padang pasir, seperti tampak pada Gambar 4.13 berikut ini.



Gambar 4.13 Tampilan Lanjutan Prolog *Game-8*

Ketika layar ditekan menuju ke prolog berikutnya berupa teks bahwa terdapat 3 musuh perkuliahan yang harus dihadapi agar mahasiswa dapat kembali ke era dari mahasiswa tersebut, seperti tampak pada Gambar 4.14 berikut ini.



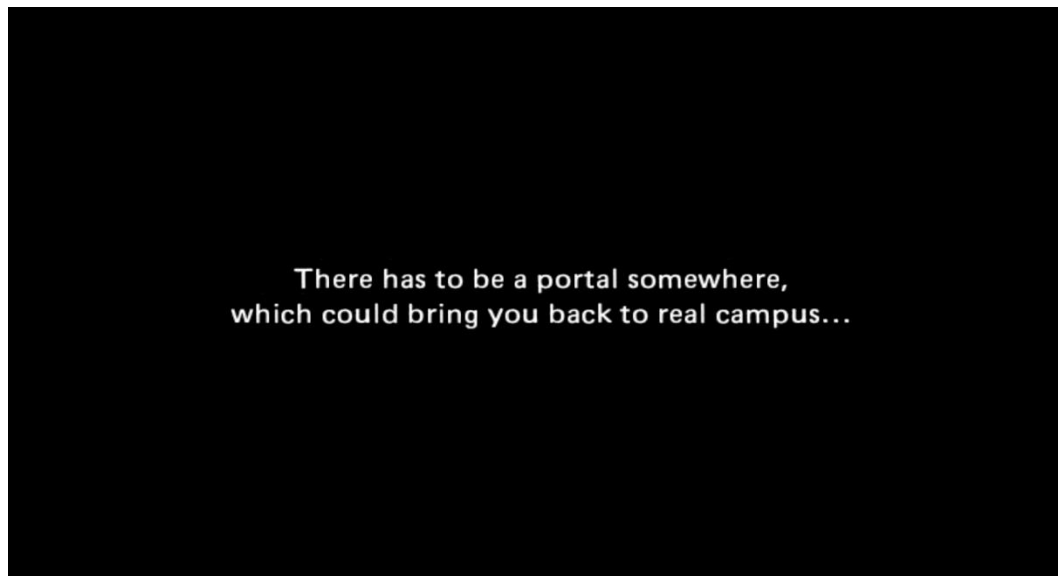
Gambar 4.14 Tampilan Lanjutan Prolog Game-9

Ketika layar ditekan menuju ke prolog berikutnya berupa gambar ketiga musuh yang dibangun dengan *Fuzzy Logic* pada penelitian ini, seperti tampak pada Gambar 4.15 berikut ini.

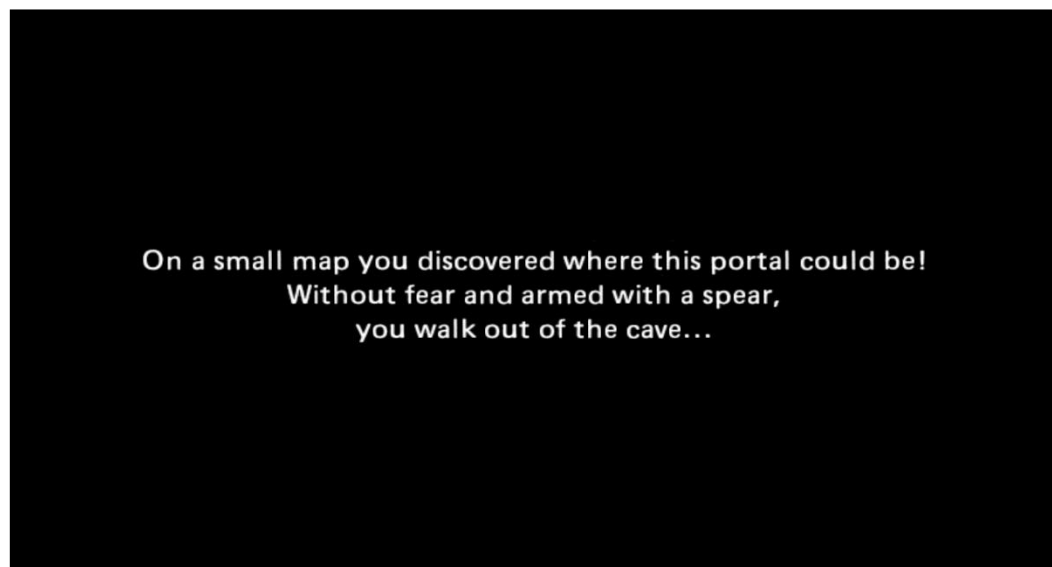


Gambar 4.15 Tampilan Lanjutan Prolog Game-10

Ketika layar ditekan menuju ke prolog berikutnya berupa berupa teks bahwa terdapat sebuah portal di padang pasir yang dapat membawa mahasiswa kembali ke zamannya dan ke kampus, seperti tampak pada Gambar 4.16 dan 4.17 berikut ini.



Gambar 4.16 Tampilan Lanjutan Prolog *Game-11*



Gambar 4.17 Tampilan Lanjutan Prolog *Game-12*

4. Tampilan *Scene* Permainan Dimulai

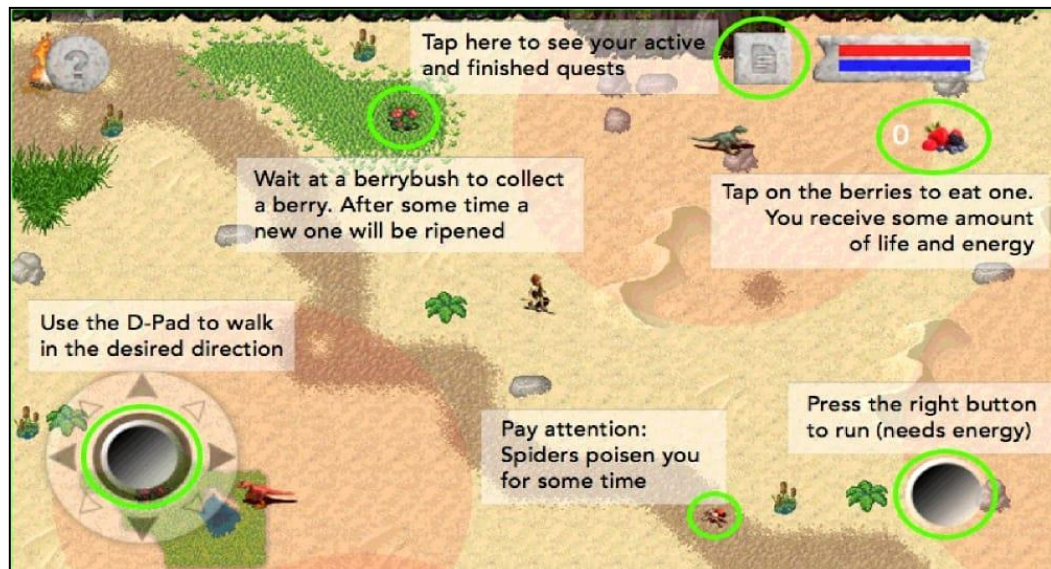
Tampilan *game* dimulai dimana mahasiswa dapat memainkan karakternya. Karakter musuh malas, konflik dengan teman, dan manajemen waktu akan mengganggu mahasiswa menyelesaikan *quest level* yang ada sehingga mahasiswa harus mampu membasi rasa malas, konflik dengan teman, dan manajemen waktu. Pada tampilan ini juga setiap perilaku dari karakter musuh NPC dimodelkan dengan metode *Fuzzy Logic* sehingga secara otomatis setiap gerakan dan tindakan yang terjadi berdasarkan hasil pengolahan dari metode *Fuzzy Logic*, seperti tampak pada Gambar 4.18 berikut ini.



Gambar 4.18Tampilan *Scene* Permainan Dimulai

5. Tampilan *Scene Help*

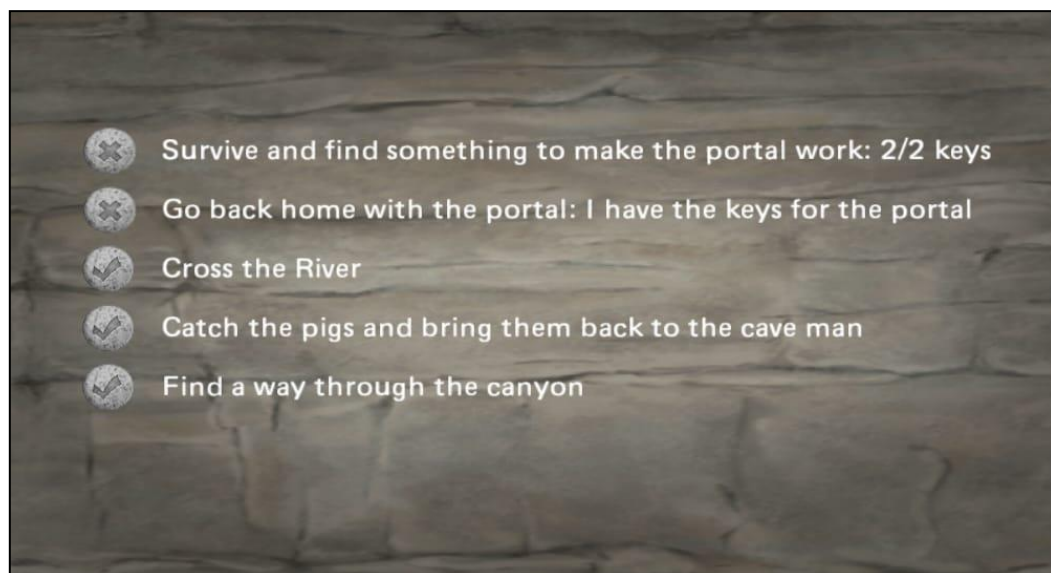
Tampilan yang berisikan bantuan cara memainkan *game*, seperti tampak pada Gambar 4.19 berikut ini.



Gambar 4.19Tampilan *Scene Help*

6. Tampilan *SceneLevel Game*

Tampilan yang berisikan *list level game* yang harus diselesaikan, seperti tampak pada Gambar 4.20 berikut ini.



Gambar 4.20Tampilan *SceneList Quest*

4.2 Pembahasan

Selanjutnya, dilakukan pembahasan mengenai kelebihan dan kekurangan dari game “*Campus Adventure*” yang dibangun pada penelitian ini. Kelebihan dari game yang dibangun yaitu:

1. Game “*Campus Adventure*” merupakan game berkonsep *adventure* sehingga mengenalkan petualangan dalam perkuliahan melalui simulasi yang mendalam. Pemain dapat merasakan tantangan-tantangan yang umum dihadapi oleh mahasiswa, seperti rasa malas, konflik dengan teman, dan manajemen waktu, yang semuanya dikemas dalam bentuk game petualangan yang interaktif dan menarik.
2. Game ini mengimplementasikan metode *Fuzzy Logic* untuk menciptakan musuh NPC (*Non-Playable Character*) yang pintar. Dengan menggunakan *Fuzzy Logic*, perilaku musuh dalam game menjadi lebih adaptif dan dinamis, sehingga memberikan tantangan yang lebih besar bagi pemain.

Kekurangan dari game yang dibangun yaitu:

1. Game ini hanya memiliki tiga jenis musuh NPC, yaitu musuh malas (*Spider*), konflik dengan teman (*Dinosaurus Hijau*), dan manajemen waktu (*Dinosaurus Merah*). Variasi musuh yang terbatas ini dapat membuat permainan terasa monoton setelah beberapa waktu.
2. Game ini hanya mendukung mode *single user*, sehingga tidak ada fitur *multiplayer* atau interaksi dengan pemain lain. Hal ini membatasi pengalaman bermain dan potensi kolaborasi atau kompetisi dengan teman.
3. Game ini hanya memiliki tiga *level* yang merepresentasikan proses perkuliahan. Jumlah level yang terbatas dapat membuat permainan cepat

selesai dan mengurangi *replayability* atau keinginan pemain untuk terus bermain.

4. Aplikasi ini hanya dapat dijalankan pada perangkat *mobile* dengan sistem operasi Android, sehingga pengguna iOS atau platform lain tidak dapat memainkan *game* ini.

BAB V

KESIMPULAN DAN SARAN

5.1 Kesimpulan

Adapun kesimpulan yang diuraikan berkaitan dengan hasil penelitian yang didapatkan yaitu:

1. Game "*Campus Adventure*" berhasil merancang sebuah game petualangan yang mampu mengenalkan petualangan dalam perkuliahan. Pemain dapat merasakan dan memahami berbagai tantangan yang akan mereka hadapi di lingkungan kampus melalui simulasi interaktif yang disediakan oleh game ini. Dengan demikian, calon mahasiswa dapat mempersiapkan diri dengan lebih baik untuk menghadapi kehidupan perkuliahan yang sebenarnya.
2. Game ini telah berhasil mengimplementasikan metode *Fuzzy Logic* untuk menciptakan musuh NPC yang cerdas dan adaptif. Dengan menggunakan *Fuzzy Logic*, perilaku musuh menjadi lebih dinamis dan menantang, sehingga meningkatkan kualitas dan kompleksitas permainan.

5.2 Saran

Adapun saran yang diuraikan berkaitan dengan penelitian yang didapatkan yaitu:

1. Untuk meningkatkan variasi dan kompleksitas permainan, disarankan agar pengembangan lebih lanjut mempertimbangkan penambahan jenis musuh NPC yang lebih banyak. Dengan lebih banyak jenis musuh, pemain akan

menghadapi berbagai tantangan yang berbeda, sehingga permainan tidak terasa monoton.

2. Untuk meningkatkan interaksi sosial dan kompetisi antar pemain, disarankan agar *game* ini dikembangkan untuk mendukung mode *multiplayer*. Mode ini memungkinkan pemain untuk berkolaborasi atau bersaing dengan teman-teman mereka, sehingga pengalaman bermain menjadi lebih kaya dan menarik.
3. Untuk memperpanjang durasi permainan dan meningkatkan *replayability*, disarankan agar lebih banyak *level* dan *quest* ditambahkan ke dalam *game*. Dengan lebih banyak level dan variasi *quest*, pemain akan terus tertantang dan termotivasi untuk menyelesaikan semua *level* yang ada.
4. Untuk memperluas jangkauan pengguna, disarankan agar *game* ini dikembangkan untuk dapat dijalankan pada berbagai platform selain *Android*, seperti iOS dan mungkin juga PC. Hal ini akan memungkinkan lebih banyak orang untuk mengakses dan menikmati *game* "*Campus Adventure*".

DAFTAR PUSTAKA

- [1] S. R. Cahyani and Y. Artanti, “Pengaruh online customer trust dan online store environment terhadap online impulse buying produk fashion melalui perceived enjoyment pada konsumen TikTok shop,” *J. Ilmu Manaj.*, vol. 11, no. 2, pp. 252–265, 2023.
- [2] F. Nugroho *et al.*, “2D Game ‘Omar’s Adventure’• design using the Finite State Machine Method,” *J. Informatics Telecommun. Eng.*, vol. 6, no. 1, pp. 18–26, 2022, doi: 10.31289/jite.v6i1.6327.
- [3] F. Novitria and R. N. Khoirunnisa, “Perbedaan Kecemasan Akademik Pada Mahasiswa Baru Jurusan Psikologi Ditinjau Dari Jenis Kelamin,” *J. Penelit. Psikol.*, vol. 9, no. 1, pp. 11–20, 2020.
- [4] C. Feriana, A. Rachman, and R. A. Kurniawan, “Pembuatan Animasi Pendek 2D Cerita Rakyat Timun Mas,” *J. Desain*, vol. 10, no. 1, p. 61, 2022, doi: 10.30998/jd.v10i1.13063.
- [5] L. E. Widodo, R. Intan, and H. Juwiantho, “Adaptive AI Pada Survival Horror Game Menggunakan Fuzzy,” *J. Infra*, vol. 8, no. 2, 2020, [Online]. Available: <http://publication.petra.ac.id/index.php/teknik-informatika/article/view/10507%0Ahttp://publication.petra.ac.id/index.php/teknik-informatika/article/download/10507/9350>.
- [6] A. Nathanufa and A. U. Zailani, “Perancangan Dan Implementasi Game Edukasi ‘ Math in Space ’ Sebagai Media Pembelajaran Operasi Hitung Bilangan Bulat Matematika Tingkat Sekolah Dasar Berbasis Android,” *Log. J. Ilmu Komput. dan Pendidik.*, vol. 1, no. 4, pp. 790–800, 2023.
- [7] R. K. Gea, I. H. . Manurung, A. F. . Sibero, and R. Sitanggang, “Perancangan Aplikasi Game 2D Kill Ghost Menggunakan Unity,” *J. Inform. Kaputama*, vol. 7, no. 1, pp. 100–112, 2023, doi: 10.59697/jik.v7i1.12.
- [8] A. F. S. Anggitta Dharmawan, “2019, Studi Komparatif User Experience

Desain Antar Muka Pengguna Aplikasi Mobile Berdasarkan Elemen Desain.”

- [9] S. D. Purnamasari and F. Panjaitan, “2020, Pengembangan Aplikasi E-Reporting Kerusakan Lampu Jalan Berbasis Mobile,” *Jusikom J. Sist. Komput. Musirawas*, vol. 5, no. 1, pp. 59–69.
- [10] Mamay Syani and Nindi Werstantia, “2018, Perancangan Aplikasi Pemesanan Catering Berbasis Mobile Android,” *J. Ilm. Ilmu dan Teknol. Rekayasa* |, vol. 1, no. 2, pp. 1–10.
- [11] Muttaqin *et al.*, *Implementasi Artificial Intelligence (AI) Dalam Kehidupan*. Medan: Yayasan Kita Menulis, 2023.
- [12] W. Wibawanto, *Game Edukasi RPG (Role Playing Game)*. Semarang: LPPM UNNES, 2020.
- [13] K. Arief, D. Purwanto, and H. Kusuma, “Algoritma Menghadang Bola dengan Metode Fuzzy Logic untuk Robot Penjaga Gawang Sepak Bola Beroda,” *J. Tek. ITS*, vol. 7, no. 2, pp. 349–354, 2018.
- [14] Asriyanik and K. Tarwati, “Metode Fuzzy Logic Untuk Penentuan Kelayakan Penerima Beasiswa Mahasiswa Di Universitas Muhammadiyah Sukabumi,” *JASISFO J. Sist. Inf.*, vol. 1, no. 2, p. 56, 2020.
- [15] R. A. Sukamto and M. Shalahuddin, “Rekayasa Perangkat Lunak Terstruktur dan Berorientasi Objek.” Informatika, Bandung, 2018.
- [16] Khairunnisa *et al.*, *Multimedia Teori dan Aplikasi Dalam Dunia Pendidikan*. Jambi: PT. Sonpedia Publishing Indonesia, 2023.
- [17] Hamdi, Z. Ramdhan, and A. Lionardi, “Perancangan Storyboard Untuk Animasi 2D Pengenalan Teknik Dasar Bermain Skateboard Untuk Mengurangi Tingkat Kecelakaan Permainan Skateboard,” *e-Proceeding Art Des.*, vol. 10, no. 2, pp. 2815–2840, 2023.

DAFTAR RIWAYAT HIDUP

Nama : Fernando Stanley Felix
Tempat/Tanggal Lahir : Lubuk Pakam, 24 September 2002
Jenis Kelamin : Laki - Laki
Agama : Buddha
Alamat : Jln. AR HAKIM GG PACAR NO 74
Email : fernandofelix8111@gmail.com

Pendidikan Formal :
2008 - 2014 : SD WR SUPRATMAN 1 MEDAN
2014 - 2017 : SMP WR SUPRATMAN 1 MEDAN
2017 - 2020 : SMA WR SUPRATMAN 1 MEDAN
2020 - 2024 : STMIK TIME Medan

Medan, 30 Mei 2022

Fernando Stanley Felix



SURAT KEPUTUSAN
NO. 060/SK/P1/STMIKTIME/XII/2023

Tentang
PENGANGKATAN DOSEN PEMBIMBING SKRIPSI
PROGRAM STUDI TEKNIK INFORMATIKA
STMIK TIME

1. **Membaca** : a. Hasil pemeriksaan berkas dan kelengkapan mahasiswa.
b. Presentasi Proposal Judul Skripsi mahasiswa tertanggal 09 Oktober – 18 November 2023.
2. **Mengingat** : a. Kurikulum Sekolah Tinggi Manajemen Informatika dan Komputer (STMIK)Time Program Studi Teknik Informatika.
b. Penyesuaian teknis pelaksanaan Skripsi Program Studi Teknik Informatika.
3. **Menimbang** : Bahwa skripsi mahasiswa merupakan wujud aplikasi kristalisasi sikap kemampuan dan ketrampilan mahasiswa/i dalam upaya menerapkan ilmu pengetahuan yang telah dipelajari dibangku kuliah, serta skripsi merupakan salah satu mata kuliah yang memiliki bobot 6 SKS, maka dipandang perlu untuk menetapkan Dosen Pembimbing I dan Dosen Pembimbing II Skripsi.

4. **Menetapkan** :
Pertama :

Mengangkat Saudara :

Pembimbing I	Joni, S.Kom., M.M
Pembimbing II	Feriani Astuti Tarigan, M.Kom., M.M

Sebagai Dosen Pembimbing Skripsi atas :

2044020	Fernando Stanley Felix	Teknik Informatika
---------	------------------------	--------------------

Dengan Judul Skripsi :

Perancangan Game "Campus Adventure" Menggunakan Metode Fuzzy Logic
--

Mahasiswa Program Strata Satu (S1) Sekolah Tinggi Manajemen Informatika dan Komputer (STMIK)Time.

- Kedua : Apabila terdapat kekeliruan di kemudian hari tentang penetapan Dosen Pembimbing Skripsi ini akan diperbaiki sebagaimana mestinya
- Ketiga : Surat Keputusan ini berlaku sejak tanggal ditetapkan 07 Desember 2023.

Ditetapkan di
Medan, 07 Desember 2023



Hendri, M.Kom
Pembantu Ketua I STMIK TIME



STMIK TIME

SEKOLAH TINGGI MANAJEMEN INFORMATIKA & KOMPUTER - TIME

FORMULIR BIMBINGAN SKRIPSI

NAMA : FERNANDO STANLEY FELIX
NIM : 2044020
JUDUL SKRIPSI : PERANCANGAN GAME "CAMPUS ADVENTURE"
MENGGUNAKAN METODE FUZZY LOGIC
NAMA DOSEN : JONI, S.Kom., M.M.
DOSEN PEMBIMBING : ☒ I (SATU) ☐ II (DUA)

TANGGAL BIMBINGAN	CATATAN BIMBINGAN	PARAF
5 Februari 2024	Perbaikan Bab 1 : ejaan kalimat, istilah-istilah asing, Rumusan Masalah, Tujuan Penelitian, dan Manfaat Penelitian	
16 Februari 2024	ACC Bab 1	
10 Maret 2024	Perbaikan Bab 2 : perataan penulisan paragraf, kalimat keterangan tabel dan gambar	
12 Maret 2024	ACC Bab 2	
15 April 2024	Perbaikan Bab 3 : kalimat keterangan tabel dan gambar	
25 April 2024	ACC Bab 3	
20 Juli 2024	Perbaikan Bab 4 : kalimat keterangan gambar, penamaan gambar	
25 Juli 2024	ACC Bab 4	
27 Juli 2024	ACC Bab 5	
29 Juli 2024	ACC Skripsi	

Ketentuan :

1. Bimbingan dengan dosen pembimbing minimal 8 (delapan) kali
2. Formulir Bimbingan wajib diisi dan ditunjukkan ke dosen pembimbing tiap bimbingan
3. Formulir Bimbingan Skripsi ini dianggap sah, apabila telah mendapatkan Pengesahan dari dosen pembimbing
4. Jika tidak memenuhi ketentuan di atas, skripsi dapat **DIBATALKAN**

Medan, Juli 2024
Dosen Pembimbing,



JONI, S.Kom., M.M.

Mahasiswa,

FERNANDO STANLEY FELIX



STMIK TIME

SEKOLAH TINGGI MANAJEMEN INFORMATIKA & KOMPUTER - TIME

FORMULIR BIMBINGAN SKRIPSI

NAMA : Fernando Stanley Felix
NIM : 2044020
JUDUL SKRIPSI : PERANCANGAN GAME "CAMPUS ADVENTURE"
MENGGUNAKAN METODE FUZZY LOGIC
NAMA DOSEN : Feriani Astuti Tarigan, M.Kom.,M.M
DOSEN PEMBIMBING : ☐ I (SATU) ☒ II (DUA)

TANGGAL BIMBINGAN	CATATAN BIMBINGAN	PARAF
22 April 2024	Revisi Bab 1 : Revisi Penomoran	
24 April 2024	Revisi Bab 2 : Tambahkan Teori	
25 April 2024	Revisi Bab 3 : ACC Sempro	
15 Juli 2024	Revisi Bab 3 : Perbaiki Kalimat dan Gambar	
18 Juli 2024	Revisi Bab 4 : Kalimat Keterangan	
22 Juli 2024	ACC Bab 4	
26 Juli 2024	ACC Bab 5	
27 Juli 2024	ACC SKRIPSI	

Ketentuan :

1. Bimbingan dengan dosen pembimbing minimal 8 (delapan) kali
2. Formulir Bimbingan wajib diisi dan ditunjukkan ke dosen pembimbing tiap bimbingan
3. Formulir Bimbingan Skripsi ini dianggap sah, apabila telah mendapatkan Pengesahan dari dosen pembimbing
4. Jika tidak memenuhi ketentuan di atas, skripsi dapat **DIBATALKAN**

Medan, Juli 2024
Dosen Pembimbing,

Mahasiswa,



Feriani Astuti Tarigan, M.Kom.,M.

FERNANDO STANLEY FELIX

LISTING PROGRAM

```

/*****
*****
*****
****
 * Copyright 2010 Mario
Zechner
(contact@badlogicgames.co
m)
 *
 * Licensed under the
Apache License, Version
2.0 (the "License"); you
may not use this file
except in compliance with
the
 * License. You may
obtain a copy of the
License at
 *
 *
http://www.apache.org/lic
enses/LICENSE-2.0
 *
 * Unless required by
applicable law or agreed
to in writing, software
distributed under the
License is distributed on
an "AS IS"
 * BASIS, WITHOUT
WARRANTIES OR CONDITIONS
OF ANY KIND, either
express or implied. See
the License for the
specific language
 * governing permissions
and limitations under the
License.

****
****
****
****/

package
com.badlogic.gdx.math;

/**
 * Encapsulates a 2D
vector. Allows chaining
methods by returning a
reference to itself
 * @author
badlogicgames@gmail.com
 *
 */
public final class
Vector2 {
    /** static temporary
vector */
    private final static
Vector2 tmp = new
Vector2();

    /** the x-component
of this vector */
    public float x;
    /** the y-component
of this vector */
    public float y;

    /**
 * Constructs a new
vector at (0,0)
 */
    public Vector2 () {

    }

    /**
 * Constructs a
vector with the given
components
 * @param x The x-
component
 * @param y The y-
component
 */
    public Vector2
(float x, float y) {
        this.x = x;
        this.y = y;
    }
}
```

```

    }

    /**
     * Constructs a
     vector from the given
     vector
     * @param v The
     vector
     */
    public Vector2
(Vector2 v) {
        set(v);
    }

    /**
     * @return a copy of
     this vector
     */
    public Vector2 cpy ()
    {
        return new
Vector2(this);
    }

    /**
     * @return The
     euclidian length
     */
    public float len ()
    {
        return
(float)Math.sqrt(x * x +
y * y);
    }

    /**
     * @return The
     squared euclidian length
     */
    public float len2 ()
    {
        return x * x +
y * y;
    }

    /**
     * Sets this vector
     from the given vector

```

```

     * @param v The
     vector
     * @return This
     vector for chaining
     */
    public Vector2 set
(Vector2 v) {
        x = v.x;
        y = v.y;
        return this;
    }

    /**
     * Sets the
     components of this vector
     * @param x The x-
     component
     * @param y The y-
     component
     * @return This
     vector for chaining
     */
    public Vector2 set
(float x, float y) {
        this.x = x;
        this.y = y;
        return this;
    }

    /**
     * Subtracts the
     given vector from this
     vector.
     * @param v The
     vector
     * @return This
     vector for chaining
     */
    public Vector2 sub
(Vector2 v) {
        x -= v.x;
        y -= v.y;
        return this;
    }

    /**
     * Normalizes this
     vector

```

```

        * @return This
        vector for chaining
        */
        public Vector2 nor ()
        {
            float len =
len();
            if (len != 0) {
                x /= len;
                y /= len;
            }
            return this;
        }

/**
 * Adds the given
vector to this vector
 * @param v The
vector
 * @return This
vector for chaining
 */
        public Vector2 add
(Vector2 v) {
            x += v.x;
            y += v.y;
            return this;
        }

/**
 * Adds the given
components to this vector
 * @param x The x-
component
 * @param y The y-
component
 * @return This
vector for chaining
 */
        public Vector2 add
(float x, float y) {
            this.x += x;
            this.y += y;
            return this;
        }

/**
 * @param v The
other vector

```

```

        * @return The dot
product between this and
the other vector
        */
        public float dot
(Vector2 v) {
            return x * v.x
+ y * v.y;
        }

/**
 * Multiplies this
vector by a scalar
 * @param scalar The
scalar
 * @return This
vector for chaining
 */
        public Vector2 mul
(float scalar) {
            x *= scalar;
            y *= scalar;
            return this;
        }

/**
 * @param v The
other vector
 * @return the
distance between this and
the other vector
 */
        public float dst
(Vector2 v) {
            float x_d = v.x
- x;
            float y_d = v.y
- y;
            return
(float)Math.sqrt(x_d *
x_d + y_d * y_d);
        }

/**
 * @param x The x-
component of the other
vector

```

```

        * @param y The y-
component of the other
vector
        * @return the
distance between this and
the other vector
        */
        public float dst
(float x, float y) {
            float x_d = x -
this.x;
            float y_d = y -
this.y;
            return
(float)Math.sqrt(x_d *
x_d + y_d * y_d);
        }

        /**
        * @param v The
other vector
        * @return the
squared distance between
this and the other vector
        */
        public float dst2
(Vector2 v) {
            float x_d = v.x
- x;
            float y_d = v.y
- y;
            return x_d *
x_d + y_d * y_d;
        }

        public String
toString () {
            return "[" + x
+ ":" + y + "]";
        }

        /**
        * Subtracts the
other vector from this
vector.
        * @param x The x-
component of the other
vector

```

```

        * @param y The y-
component of the other
vector
        * @return This
vector for chaining
        */
        public Vector2 sub
(float x, float y) {
            this.x -= x;
            this.y -= y;
            return this;
        }

        /**
        * @return a
temporary copy of this
vector. Use with care as
this is backed by a
single static Vector2
instance. v1.tmp().add(
        *           v2.tmp() )
will not work!
        */
        public Vector2 tmp ()
{
            return
tmp.set(this);
        }

        /**
        * @param v the
other vector
        * @return The cross
product between this and
the other vector
        */
        public float
cross(final Vector2 v) {
            return this.x *
v.y - v.x * this.y;
        }

        /**
        * @return The
manhattan length
        */
        public float
lenManhattan() {

```

```

        return
Math.abs(this.x) +
Math.abs(this.y);
    }
}
/*****
*****
*****
*****
* Copyright 2011 See
AUTHORS file.
*
* Licensed under the
Apache License, Version
2.0 (the "License");
* you may not use this
file except in compliance
with the License.
* You may obtain a copy
of the License at
*
*
http://www.apache.org/licenses/LICENSE-2.0
*
* Unless required by
applicable law or agreed
to in writing, software
* distributed under the
License is distributed on
an "AS IS" BASIS,
* WITHOUT WARRANTIES OR
CONDITIONS OF ANY KIND,
either express or implied.
* See the License for
the specific language
governing permissions and
* limitations under the
License.
*****
*****
*****
***/
package
com.badlogic.gdx.physics.
box2d;

```

```

import
java.util.ArrayList;

import
com.badlogic.gdx.math.Vec
tor2;
import
com.badlogic.gdx.physics.
box2d.BodyDef.BodyType;

/**
 * A rigid body. These
are created via
World.CreateBody.
 * @author mzechner
 *
 */
public class Body {
    /** the address of
the body */
    protected final long
addr;

    /** temporary float
array */
    private final float[]
tmp = new float[4];

    /** World */
    private final World
world;

    /** Fixtures of this
body */
    private
ArrayList<Fixture>
fixtures = new
ArrayList<Fixture>(2);

    /** Joints of this
body */
    protected
ArrayList<JointEdge>
joints = new
ArrayList<JointEdge>(2);

    /** user data */
    private Object
userData;

```

```

        /**
         * Constructs a new
body with the given
address
         * @param world the
world
         * @param addr the
address
         */
        protected Body
(World world, long addr)
{
            this.world =
world;
            this.addr =
addr;
        }

        /**
         * Creates a fixture
and attach it to this
body. Use this function
if you need to set some
fixture parameters, like
friction.
         * Otherwise you can
create the fixture
directly from a shape. If
the density is non-zero,
this function
automatically updates
         * the mass of the
body. Contacts are not
created until the next
time step.
         * @param def the
fixture definition.
         * @warning This
function is locked during
callbacks.
         */
        public Fixture
createFixture (FixtureDef
def) {
            Fixture fixture
= new Fixture(this,
jniCreateFixture(addr,
def.shape.addr,
def.friction,
def.restitution,
def.density,

            def.isSensor,
def.filter.categoryBits,
def.filter.maskBits,
def.filter.groupIndex));

            this.world.fixtures.
put(fixture.addr,
fixture);

            this.fixtures.add(fi
xture);
            return fixture;
        }

        private native long
jniCreateFixture (long
addr, long shapeAddr,
float friction, float
restitution, float
density,
                boolean
isSensor, short
filterCategoryBits, short
filterMaskBits, short
filterGroupIndex);

        /**
         * Creates a fixture
from a shape and attach
it to this body. This is
a convenience function.
Use b2FixtureDef if you
need to
         * set parameters
like friction,
restitution, user data,
or filtering. If the
density is non-zero, this
function automatically
         * updates the mass
of the body.
         * @param shape the
shape to be cloned.

```

```

        * @param density
the shape density (set to
zero for static bodies).
        * @warning This
function is locked during
callbacks.
        */
        public Fixture
createFixture (Shape
shape, float density) {
            Fixture fixture
= new Fixture(this,
jniCreateFixture(addr,
shape.addr, density));

            this.world.fixtures.
put(fixture.addr,
fixture);

            this.fixtures.add(fi
xture);
            return fixture;
        }

        private native long
jniCreateFixture (long
addr, long shapeAddr,
float density);

        /**
        * Destroy a fixture.
This removes the fixture
from the broad-phase and
destroys all contacts
associated with this
fixture.
        * This will
automatically adjust the
mass of the body if the
body is dynamic and the
fixture has positive
density. All
        * fixtures attached
to a body are implicitly
destroyed when the body
is destroyed.
        * @param fixture
the fixture to be removed.

```

```

        * @warning This
function is locked during
callbacks.
        */
        public void
destroyFixture (Fixture
fixture) {

            jniDestroyFixture(ad
dr, fixture.addr);

            this.world.fixtures.
remove(fixture.addr);

            this.fixtures.remove
(fixture);
        }

        private native void
jniDestroyFixture (long
addr, long fixtureAddr);

        /**
        * Set the position
of the body's origin and
rotation. This breaks any
contacts and wakes the
other bodies.
Manipulating a
        * body's transform
may cause non-physical
behavior.
        * @param position
the world position of the
body's local origin.
        * @param angle the
world rotation in radians.
        */
        public void
setTransform (Vector2
position, float angle) {

            jniSetTransform(addr,
position.x, position.y,
angle);
        }

        /**

```

```

        * Set the position
of the body's origin and
rotation. This breaks any
contacts and wakes the
other bodies.
Manipulating a
        * body's transform
may cause non-physical
behavior.
        * @param x the
world position on the x-
axis
        * @param y the
world position on the y-
axis
        * @param angle the
world rotation in radians.
        */
    public void
setTransform(float x,
float y, float angle) {

        jniSetTransform(addr,
x, y, angle);
    }

    private native void
jniSetTransform (long
addr, float positionX,
float positionY, float
angle);

    /**
     * Get the body
transform for the body's
origin. FIXME
     */
    private final
Transform transform = new
Transform();

    public Transform
getTransform () {

        jniGetTransform(addr,
transform.vals);
        return
transform;

```

```

    }

    private native void
jniGetTransform (long
addr, float[] vals);

    private final
Vector2 position = new
Vector2();

    /**
     * Get the world
body origin position.
     * @return the world
position of the body's
origin.
     */
    public Vector2
getPosition () {

        jniGetPosition(addr,
tmp);

        position.x =
tmp[0];
        position.y =
tmp[1];
        return position;
    }

    private native void
jniGetPosition (long addr,
float[] position);

    /**
     * Get the angle in
radians.
     * @return the
current world rotation
angle in radians.
     */
    public float
getAngle () {

        return
jniGetAngle(addr);
    }

    private native float
jniGetAngle (long addr);

```

```

        /**
         * Get the world
        position of the center of
        mass.
        */
        private final
        Vector2 worldCenter = new
        Vector2();

        public Vector2
        getWorldCenter () {

            jniGetWorldCenter(ad
            dr, tmp);

            worldCenter.x =
            tmp[0];
            worldCenter.y =
            tmp[1];

            return
            worldCenter;
        }

        private native void
        jniGetWorldCenter (long
        addr, float[]
        worldCenter);

        /**
         * Get the local
        position of the center of
        mass.
        */
        private final
        Vector2 localCenter = new
        Vector2();

        public Vector2
        getLocalCenter () {

            jniGetLocalCenter(ad
            dr, tmp);

            localCenter.x =
            tmp[0];
            localCenter.y =
            tmp[1];

            return
            localCenter;
        }

```

```

        private native void
        jniGetLocalCenter (long
        addr, float[]
        localCenter);

        /**
         * Set the linear
        velocity of the center of
        mass.
        */
        public void
        setLinearVelocity
        (Vector2 v) {

            jniSetLinearVelocity
            (addr, v.x, v.y);
        }

        /**
         * Set the linear
        velocity of the center of
        mass.
        */
        public void
        setLinearVelocity (float
        vX, float vY) {

            jniSetLinearVelocity
            (addr, vX, vY);
        }

        private native void
        jniSetLinearVelocity
        (long addr, float x,
        float y);

        /**
         * Get the linear
        velocity of the center of
        mass.
        */
        private final
        Vector2 linearVelocity =
        new Vector2();

        public Vector2
        getLinearVelocity () {

```

```

        jniGetLinearVelocity
(addr, tmp);

        linearVelocity.x =
tmp[0];

        linearVelocity.y =
tmp[1];

        return
linearVelocity;
    }

    private native void
jniGetLinearVelocity
(long addr, float[]
tmpLinearVelocity);

    /**
     * Set the angular
velocity.
     */
    public void
setAngularVelocity (float
omega) {

        jniSetAngularVelocit
y(addr, omega);
    }

    private native void
jniSetAngularVelocity
(long addr, float omega);

    /**
     * Get the angular
velocity.
     */
    public float
getAngularVelocity () {
        return
jniGetAngularVelocity(addr);
    }

    private native float
jniGetAngularVelocity
(long addr);

    /**
     * Apply a force at
a world point. If the
force is not applied at
the center of mass, it
will generate a torque
and affect the
        * angular velocity.
This wakes up the body.
        * @param force the
world force vector,
usually in Newtons (N).
        * @param point the
world position of the
point of application.
    */
    public void
applyForce (Vector2 force,
Vector2 point) {

        jniApplyForce(addr,
force.x, force.y, point.x,
point.y);
    }

    /**
     * Apply a force at
a world point. If the
force is not applied at
the center of mass, it
will generate a torque
and affect the
        * angular velocity.
This wakes up the body.
        * @param forceX the
world force vector on x,
usually in Newtons (N).
        * @param forceY the
world force vector on y,
usually in Newtons (N).
        * @param pointX the
world position of the
point of application on x.
        * @param pointY the
world position of the
point of application on y.
    */
    public void
applyForce (float forceX,

```

```

float forceY, float
pointX, float pointY) {

    jniApplyForce(addr,
forceX, forceY, pointX,
pointY);
    }

    private native void
jniApplyForce (long addr,
float forceX, float
forceY, float pointX,
float pointY);

    /**
     * Apply a torque.
     This affects the angular
     velocity without
     affecting the linear
     velocity of the center of
     mass. This wakes up
     * the body.
     * @param torque
     about the z-axis (out of
     the screen), usually in
     N-m.
     */
    public void
applyTorque (float torque)
{

    jniApplyTorque(addr,
torque);
    }

    private native void
jniApplyTorque (long addr,
float torque);

    /**
     * Apply an impulse
     at a point. This
     immediately modifies the
     velocity. It also
     modifies the angular
     velocity if the point of
     * application is
     not at the center of mass.
     This wakes up the body.

```

```

     * @param impulse
     the world impulse vector,
     usually in N-seconds or
     kg-m/s.
     * @param point the
     world position of the
     point of application.
     */
    public void
applyLinearImpulse
(Vector2 impulse, Vector2
point) {

    jniApplyLinearImpuls
e(addr, impulse.x,
impulse.y, point.x,
point.y);
    }

    /**
     * Apply an impulse
     at a point. This
     immediately modifies the
     velocity. It also
     modifies the angular
     velocity if the point of
     * application is
     not at the center of mass.
     This wakes up the body.
     * @param impulseX
     the world impulse vector
     on the x-axis, usually in
     N-seconds or kg-m/s.
     * @param impulseY
     the world impulse vector
     on the y-axis, usually in
     N-seconds or kg-m/s.
     * @param pointX the
     world position of the
     point of application on
     the x-axis.
     * @param pointY the
     world position of the
     point of application on
     the y-axis.
     */
    public void
applyLinearImpulse(float
impulseX, float impulseY,

```

```

float pointX, float
pointY) {

    jniApplyLinearImpuls
e(addr, impulseX,
impulseY, pointX, pointY);
}

private native void
jniApplyLinearImpulse
(long addr, float
impulseX, float impulseY,
float pointX, float
pointY);

/**
 * Apply an angular
impulse.
 * @param impulse
the angular impulse in
units of kg*m*m/s
 */
public void
applyAngularImpulse
(float impulse) {

    jniApplyAngularImpul
se(addr, impulse);
}

private native void
jniApplyAngularImpulse
(long addr, float
impulse);

/**
 * Get the total
mass of the body.
 * @return the mass,
usually in kilograms (kg).
 */
public float getMass
() {
    return
jniGetMass(addr);
}

private native float
jniGetMass (long addr);

/**
 * Get the
rotational inertia of the
body about the local
origin.
 * @return the
rotational inertia,
usually in kg-m^2.
 */
public float
getInertia () {
    return
jniGetInertia(addr);
}

private native float
jniGetInertia (long addr);

private final
MassData massData = new
MassData();

/**
 * Get the mass data
of the body.
 * @return a struct
containing the mass,
inertia and center of the
body.
 */
public MassData
getMassData () {

    jniGetMassData(addr,
tmp);

    massData.mass =
tmp[0];

    massData.center.x =
tmp[1];

    massData.center.y =
tmp[2];

    massData.I =
tmp[3];

    return massData;
}

```

```

        private native void
jniGetMassData (long addr,
float[] massData);

```

```

        /**
         * Set the mass
        properties to override
        the mass properties of
        the fixtures. Note that
        this changes the center
        of mass position.
         * Note that
        creating or destroying
        fixtures can also alter
        the mass. This function
        has no effect if the body
        isn't dynamic.
         * @param data the
        mass properties.
        */
        public void
setMassData (MassData
data) {

```

```

        jniSetMassData(addr,
data.mass, data.center.x,
data.center.y, data.I);
    }

```

```

        private native void
jniSetMassData (long addr,
float mass, float centerX,
float centerY, float I);

```

```

        /**
         * This resets the
        mass properties to the
        sum of the mass
        properties of the
        fixtures. This normally
        does not need to be
        called
         * unless you called
        SetMassData to override
        the mass and you later
        want to reset the mass.
        */
        public void
resetMassData () {

```

```

        jniResetMassData(addr
r);
    }

```

```

        private native void
jniResetMassData (long
addr);

```

```

        private final
Vector2 localPoint = new
Vector2();

```

```

        /**
         * Get the world
        coordinates of a point
        given the local
        coordinates.
         * @param localPoint
        a point on the body
        measured relative the the
        body's origin.
         * @return the same
        point expressed in world
        coordinates.
        */
        public Vector2
getWorldPoint (Vector2
localPoint) {

```

```

        jniGetWorldPoint(addr
r, localPoint.x,
localPoint.y, tmp);

```

```

        this.localPoint.x =
tmp[0];

        this.localPoint.y =
tmp[1];

        return
this.localPoint;
    }

```

```

        private native void
jniGetWorldPoint (long
addr, float localPointX,
float localPointY, float[]
worldPoint);

```

```

        private final
        Vector2 worldVector = new
        Vector2();

```

```

        /**
         * Get the world
         coordinates of a vector
         given the local
         coordinates.
         * @param
         localVector a vector
         fixed in the body.
         * @return the same
         vector expressed in world
         coordinates.

```

```

        */
        public Vector2
        getWorldVector (Vector2
        localVector) {

```

```

            jniGetWorldVector(addr, localVector.x,
            localVector.y, tmp);
            worldVector.x =
            tmp[0];
            worldVector.y =
            tmp[1];
            return
            worldVector;
        }

```

```

        private native void
        jniGetWorldVector (long
        addr, float localVectorX,
        float localVectorY,
        float[] worldVector);

```

```

        public final Vector2
        localPoint2 = new
        Vector2();

```

```

        /**
         * Gets a local
         point relative to the
         body's origin given a
         world point.
         * @param worldPoint
         a point in world
         coordinates.

```

```

        * @return the
        corresponding local point
        relative to the body's
        origin.

```

```

        */
        public Vector2
        getLocalPoint (Vector2
        worldPoint) {

```

```

            jniGetLocalPoint(addr, worldPoint.x,
            worldPoint.y, tmp);
            localPoint2.x =
            tmp[0];
            localPoint2.y =
            tmp[1];
            return
            localPoint2;
        }

```

```

        private native void
        jniGetLocalPoint (long
        addr, float worldPointX,
        float worldPointY, float[]
        localPoint);

```

```

        public final Vector2
        localVector = new
        Vector2();

```

```

        /**
         * Gets a local
         vector given a world
         vector.
         * @param
         worldVector a vector in
         world coordinates.
         * @return the
         corresponding local
         vector.

```

```

        */
        public Vector2
        getLocalVector (Vector2
        worldVector) {

```

```

            jniGetLocalVector(addr, worldVector.x,
            worldVector.y, tmp);

```

```

        localVector.x =
tmp[0];
        localVector.y =
tmp[1];
        return
localVector;
    }

    private native void
jniGetLocalVector (long
addr, float worldVectorX,
float worldVectorY,
float[] worldVector);

    public final Vector2
linVelWorld = new
Vector2();

    /**
     * Get the world
linear velocity of a
world point attached to
this body.
     * @param worldPoint
a point in world
coordinates.
     * @return the world
velocity of a point.
     */
    public Vector2
getLinearVelocityFromWorl
dPoint (Vector2
worldPoint) {

        jniGetLinearVelocity
FromWorldPoint(addr,
worldPoint.x,
worldPoint.y, tmp);
        linVelWorld.x =
tmp[0];
        linVelWorld.y =
tmp[1];
        return
linVelWorld;
    }

    private native void
jniGetLinearVelocityFromW
orldPoint (long addr,
float worldPointX, float
worldPointY, float[]
linVelWorld);

    /**
     * Get the world
velocity of a local point.
     * @param localPoint
a point in local
coordinates.
     * @return the world
velocity of a point.
     */
    public Vector2
getLinearVelocityFromLoca
lPoint (Vector2
localPoint) {

        jniGetLinearVelocity
FromLocalPoint(addr,
localPoint.x,
localPoint.y, tmp);
        linVelLoc.x =
tmp[0];
        linVelLoc.y =
tmp[1];
        return
linVelLoc;
    }

    private native void
jniGetLinearVelocityFromL
ocalPoint (long addr,
float localPointX, float
localPointY, float[]
linVelLoc);

    /**
     * Get the linear
damping of the body.
     */
    public float
getLinearDamping () {
        return
jniGetLinearDamping(addr);
    }

```

```

        private native float
jniGetLinearDamping (long
addr);

        /**
         * Set the linear
damping of the body.
         */
        public void
setLinearDamping (float
linearDamping) {

            jniSetLinearDamping(
addr, linearDamping);
        }

        private native void
jniSetLinearDamping (long
addr, float
linearDamping);

        /**
         * Get the angular
damping of the body.
         */
        public float
getAngularDamping () {
            return
jniGetAngularDamping(addr
);
        }

        private native float
jniGetAngularDamping
(long addr);

        /**
         * Set the angular
damping of the body.
         */
        public void
setAngularDamping (float
angularDamping) {

            jniSetAngularDamping
(addr, angularDamping);
        }

        private native void
jniSetAngularDamping
(long addr, float
angularDamping);

        /**
         * Set the type of
this body. This may alter
the mass and velocity.
         */
        public void setType
(BodyType type) {
            jniSetType(addr,
type.getValue());
        }

        private native void
jniSetType (long addr,
int type);

        /**
         * Get the type of
this body.
         */
        public BodyType
getType () {
            int type =
jniGetType(addr);
            if (type == 0)
return
BodyType.StaticBody;
            if (type == 1)
return
BodyType.KinematicBody;
            if (type == 2)
return
BodyType.DynamicBody;
            return
BodyType.StaticBody;
        }

        private native int
jniGetType (long addr);

        /**
         * Should this body
be treated like a bullet
for continuous collision
detection?

```

```

        */
        public void
setBullet (boolean flag)
{

        jniSetBullet(addr,
flag);
}

        private native void
jniSetBullet (long addr,
boolean flag);

        /**
         * Is this body
treated like a bullet for
continuous collision
detection?
         */
        public boolean
isBullet () {
                return
jniIsBullet(addr);
}

        private native
boolean jniIsBullet (long
addr);

        /**
         * You can disable
sleeping on this body. If
you disable sleeping, the
         */
        public void
setSleepingAllowed
(boolean flag) {

        jniSetSleepingAllowe
d(addr, flag);
}

        private native void
jniSetSleepingAllowed
(long addr, boolean flag);

        /**
         * Is this body
allowed to sleep

```

```

        */
        public boolean
isSleepingAllowed () {
                return
jniIsSleepingAllowed(addr
);
}

        private native
boolean
jniIsSleepingAllowed
(long addr);

        /**
         * Set the sleep
state of the body. A
sleeping body has very
low CPU cost.
         * @param flag set
to true to put body to
sleep, false to wake it.
         */
        public void setAwake
(boolean flag) {

        jniSetAwake(addr,
flag);
}

        private native void
jniSetAwake (long addr,
boolean flag);

        /**
         * Get the sleeping
state of this body.
         * @return true if
the body is sleeping.
         */
        public boolean
isAwake () {
                return
jniIsAwake(addr);
}

        private native
boolean jniIsAwake (long
addr);

```

```

    /**
     * Set the active
     state of the body. An
     inactive body is not
     simulated and cannot be
     collided with or woken up.
     If you pass a
         * flag of true, all
     fixtures will be added to
     the broad-phase. If you
     pass a flag of false, all
     fixtures will be removed
     from
         * the broad-phase
     and all contacts will be
     destroyed. Fixtures and
     joints are otherwise
     unaffected. You may
     continue to
         * create/destroy
     fixtures and joints on
     inactive bodies. Fixtures
     on an inactive body are
     implicitly inactive and
     will not
         * participate in
     collisions, ray-casts, or
     queries. Joints connected
     to an inactive body are
     implicitly inactive. An
     inactive
         * body is still
     owned by a b2World object
     and remains in the body
     list.
    */
    public void
    setActive (boolean flag)
    {
        jniSetActive(addr,
        flag);
    }

    private native void
    jniSetActive (long addr,
    boolean flag);

    /**

```

```

     * Get the active
     state of the body.
    */
    public boolean
    isActive () {
        return
        jniIsActive(addr);
    }

    private native
    boolean jniIsActive (long
    addr);

    /**
     * Set this body to
     have fixed rotation. This
     causes the mass to be
     reset.
    */
    public void
    setFixedRotation (boolean
    flag) {
        jniSetFixedRotation(
        addr, flag);
    }

    private native void
    jniSetFixedRotation (long
    addr, boolean flag);

    /**
     * Does this body
     have fixed rotation?
    */
    public boolean
    isFixedRotation () {
        return
        jniIsFixedRotation(addr);
    }

    private native
    boolean
    jniIsFixedRotation (long
    addr);

    /**
     * Get the list of
     all fixtures attached to

```

```

this body. Do not modify
the list!
    */
    public
    ArrayList<Fixture>
    getFixtureList () {
        return fixtures;
    }

    /**
     * Get the list of
    all joints attached to
    this body. Do not modify
    the list!
    */
    public
    ArrayList<JointEdge>
    getJointList () {
        return joints;
    }

    /**
     * Get the list of
    all contacts attached to
    this body.
     * @warning this
    list changes during the
    time step and you may
    miss some collisions if
    you don't use
    b2ContactListener. Do not
     *          modify
    the returned list!
    */
    // ArrayList<ContactEdge>
    getContactList()
    // {
    // return contacts;
    // }

    /**
     * Get the parent
    world of this body.
    */
    public World
    getWorld () {
        return world;
    }

```

```

/**
 * Get the user data
 */
    public Object
    getUserData () {
        return userData;
    }

    /**
     * Set the user data
     */
    public void
    setUserData (Object
    userData) {
        this.userData =
    userData;
    }
}
/*****
*****
*****
*****
****
 * Copyright 2011 See
AUTHORS file.
 *
 * Licensed under the
Apache License, Version
2.0 (the "License");
 * you may not use this
file except in compliance
with the License.
 * You may obtain a copy
of the License at
 *
 *
http://www.apache.org/lic
enses/LICENSE-2.0
 *
 * Unless required by
applicable law or agreed
to in writing, software
 * distributed under the
License is distributed on
an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR
CONDITIONS OF ANY KIND,
either express or implied.

```

```

    * See the License for
    the specific language
    governing permissions and
    * limitations under the
    License.

```

```

*****
*****
*****
***

```

```

package
com.badlogic.gdx.utils;

```

```

import
org.andengine.util.math.M
athUtils;

```

```

import java.util.Iterator;
import
java.util.NoSuchElementEx
ception;

```

```

/**

```

```

    * An unordered map that
    uses long keys. This
    implementation is a
    cuckoo hash map using 3
    hashes, random walking,
    and a small

```

```

    * stash for problematic
    keys. Null values are
    allowed. No allocation is
    done except when growing
    the table size. <br>

```

```

    * <br>
    * This map performs very
    fast get, containsKey,
    and remove (typically
    O(1), worst case
    O(log(n))). Put may be a
    bit slower,

```

```

    * depending on hash
    collisions. Load factors
    greater than 0.91 greatly
    increase the chances the
    map will have to rehash
    to the

```

```

    * next higher POT size.
    * @author Nathan Sweet

```

```

*/
public class LongMap<V> {
    private static final
int PRIME1 = 0xbelf14b1;
    private static final
int PRIME2 = 0xb4b82e39;
    private static final
int PRIME3 = 0xcd1c241;
    private static final
int EMPTY = 0;

```

```

        public int size;

        long[] keyTable;
        V[] valueTable;
        int capacity,
        stashSize;
        V zeroValue;
        boolean hasZeroValue;

```

```

        private float
loadFactor;
        private int
hashShift, mask,
threshold;
        private int
stashCapacity;
        private int
pushIterations;

```

```

        private Entries
entries;
        private Values
values;
        private Keys keys;

```

```

/**
    * Creates a new map
    with an initial capacity
    of 32 and a load factor
    of 0.8. This map will
    hold 25 items before
    growing the
    * backing table.

```

```

    */
    public LongMap () {
        this(32, 0.8f);
    }

```

```

        /**
         * Creates a new map
         with a load factor of 0.8.
         This map will hold
         initialCapacity * 0.8
         items before growing the
         backing
         * table.
         */
        public LongMap (int
        initialCapacity) {

            this(initialCapacity,
            0.8f);
        }

        /**
         * Creates a new map
         with the specified
         initial capacity and load
         factor. This map will
         hold initialCapacity *
         loadFactor items
         * before growing
         the backing table.
         */
        public LongMap (int
        initialCapacity, float
        loadFactor) {
            if
            (initialCapacity < 0)
            throw new
            IllegalArgumentException(
            "initialCapacity must
            be >= 0: " +
            initialCapacity);
            if (capacity >
            1 << 30) throw new
            IllegalArgumentException(
            "initialCapacity is too
            large: " +
            initialCapacity);
            capacity =
            MathUtils.nextPowerOfTwo(
            initialCapacity);

            if (loadFactor
            <= 0) throw new
            IllegalArgumentException(

```

```

            "loadFactor must be > 0:
            " + loadFactor);
            this.loadFactor
            = loadFactor;

            threshold =
            (int)(capacity *
            loadFactor);
            mask = capacity
            - 1;
            hashShift = 31
            -
            Integer.numberOfTrailingZ
            eros(capacity);
            stashCapacity =
            Math.max(3,
            (int)Math.ceil(Math.log(c
            apacity)) + 1);
            pushIterations
            =
            Math.max(Math.min(capacit
            y, 32),
            (int)Math.sqrt(capacity)
            / 4);

            keyTable = new
            long[capacity +
            stashCapacity];
            valueTable =
            (V[])new
            Object[keyTable.length];
            }

            public V put (long
            key, V value) {
                if (key == 0) {
                    V oldValue
                    = zeroValue;
                    zeroValue
                    = value;

                    hasZeroValue = true;
                    size++;
                    return
                    oldValue;
                }

                // Check for
                existing keys.

```

```

        int index1 =
(int)(key & mask);
        long key1 =
keyTable[index1];
        if (key1 == key)
        {
            V oldValue
= valueTable[index1];

            valueTable[index1] =
value;

            return
oldValue;
        }

        int index2 =
hash2(key);
        long key2 =
keyTable[index2];
        if (key2 == key)
        {
            V oldValue
= valueTable[index2];

            valueTable[index2] =
value;

            return
oldValue;
        }

        int index3 =
hash3(key);
        long key3 =
keyTable[index3];
        if (key3 == key)
        {
            V oldValue
= valueTable[index3];

            valueTable[index3] =
value;

            return
oldValue;
        }

        // Check for
empty buckets.
        if (key1 ==
EMPTY) {

            keyTable[index1] =
key;

            valueTable[index1] =
value;

            if
(size++ >= threshold)
resize(capacity << 1);
            return
null;
        }

        if (key2 ==
EMPTY) {

            keyTable[index2] =
key;

            valueTable[index2] =
value;

            if
(size++ >= threshold)
resize(capacity << 1);
            return
null;
        }

        if (key3 ==
EMPTY) {

            keyTable[index3] =
key;

            valueTable[index3] =
value;

            if
(size++ >= threshold)
resize(capacity << 1);
            return
null;
        }

        push(key, value,
index1, key1, index2,
key2, index3, key3);
        return null;
    }
}

```

```

        public void putAll
(LongMap<V> map) {
            for (Entry<V>
entry : map.entries())

                put(entry.key,
entry.value);
            }

            /**
             * Skips checks for
existing keys.
             */
            private void
putResize (long key, V
value) {
                if (key == 0) {
                    zeroValue
= value;

                    hasZeroValue = true;
                    return;
                }

                // Check for
empty buckets.
                int index1 =
(int)(key & mask);
                long key1 =
keyTable[index1];
                if (key1 ==
EMPTY) {

                    keyTable[index1] =
key;

                    valueTable[index1] =
value;

                    if
(size++ >= threshold)
resize(capacity << 1);
                    return;
                }

                int index2 =
hash2(key);
                long key2 =
keyTable[index2];

```

```

                if (key2 ==
EMPTY) {

                    keyTable[index2] =
key;

                    valueTable[index2] =
value;

                    if
(size++ >= threshold)
resize(capacity << 1);
                    return;
                }

                int index3 =
hash3(key);
                long key3 =
keyTable[index3];
                if (key3 ==
EMPTY) {

                    keyTable[index3] =
key;

                    valueTable[index3] =
value;

                    if
(size++ >= threshold)
resize(capacity << 1);
                    return;
                }

                push(key, value,
index1, key1, index2,
key2, index3, key3);
            }

            private void push
(long insertKey, V
insertValue, int index1,
long key1, int index2,
long key2, int index3,
long key3) {
                long[] keyTable
= this.keyTable;
                V[] valueTable
= this.valueTable;
                int mask =
this.mask;

```

```

        // Push keys
until an empty bucket is
found.
        long evictedKey;
        V evictedValue;
        int i = 0,
pushIterations =
this.pushIterations;
        do {
            // Replace
the key and value for one
of the hashes.
            switch
(MathUtils.random(0, 2))
{
                case 0:

                    evictedKey = key1;

                    evictedValue =
valueTable[index1];

                    keyTable[index1] =
insertKey;

                    valueTable[index1] =
insertValue;
                        break;
                case 1:

                    evictedKey = key2;

                    evictedValue =
valueTable[index2];

                    keyTable[index2] =
insertKey;

                    valueTable[index2] =
insertValue;
                        break;
                default:

                    evictedKey = key3;

                    evictedValue =
valueTable[index3];

```

```

        keyTable[index3] =
insertKey;

        valueTable[index3] =
insertValue;
                        break;
                    }

                    // If the
evicted key hashes to an
empty bucket, put it
there and stop.
                        index1 =
(int)(evictedKey & mask);
                        key1 =
keyTable[index1];
                        if (key1
== EMPTY) {

                            keyTable[index1] =
evictedKey;

                            valueTable[index1] =
evictedValue;
                                if
(size++ >= threshold)
resize(capacity << 1);

                                    return;
                                }

                                    index2 =
hash2(evictedKey);
                                    key2 =
keyTable[index2];
                                    if (key2
== EMPTY) {

                                        keyTable[index2] =
evictedKey;

                                        valueTable[index2] =
evictedValue;
                                            if
(size++ >= threshold)
resize(capacity << 1);

                                                return;
                                            }

```

```

        }

        index3 =
hash3(evictedKey);
        key3 =
keyTable[index3];
        if (key3
== EMPTY) {

            keyTable[index3] =
evictedKey;

            valueTable[index3] =
evictedValue;

            if
(size++ >= threshold)
resize(capacity << 1);

            return;
        }

        if (++i ==
pushIterations) break;

        insertKey
= evictedKey;

        insertValue =
evictedValue;
        } while (true);

        putStash(evictedKey,
evictedValue);
    }

    private void
putStash (long key, V
value) {
        if (stashSize
== stashCapacity) {
            // Too
many pushes occurred and
the stash is full,
increase the table size.

            resize(capacity <<
1);

```

```

        put (key,
value);

        return;
    }
    // Update key
in the stash.
    long[] keyTable
= this.keyTable;
    for (int i =
capacity, n = i +
stashSize; i < n; i++) {
        if
(keyTable[i] == key) {

            valueTable[i] =
value;

            return;
        }
    }
    // Store key in
the stash.
    int index =
capacity + stashSize;
    keyTable[index]
= key;

    valueTable[index] =
value;
    stashSize++;
}

    public V get (long
key) {
        if (key == 0)
return zeroValue;
        int index =
(int)(key & mask);
        if
(keyTable[index] != key)
{
            index =
hash2(key);

            if
(keyTable[index] != key)
{
                index
= hash3(key);

```

```

        if
        (keyTable[index] != key)
        return getStash(key);
    }
    }
    return
    valueTable[index];
}

    private V getStash
    (long key) {
        long[] keyTable
    = this.keyTable;
        for (int i =
    capacity, n = i +
    stashSize; i < n; i++)
            if
            (keyTable[i] == key)
            return valueTable[i];
            return null;
    }

    public V remove
    (long key) {
        if (key == 0) {
            if
            (!hasZeroValue) return
            null;
            V oldValue
            = zeroValue;
            zeroValue
            = null;

            hasZeroValue = false;
            size--;
            return
            oldValue;
        }

        int index =
        (int)(key & mask);
        if
        (keyTable[index] == key)
        {

            keyTable[index] =
            EMPTY;
            V oldValue
            = valueTable[index];

```

```

        valueTable[index] =
        null;
            size--;
            return
            oldValue;
        }

        index =
        hash2(key);
        if
        (keyTable[index] == key)
        {

            keyTable[index] =
            EMPTY;
            V oldValue
            = valueTable[index];

            valueTable[index] =
            null;
            size--;
            return
            oldValue;
        }

        index =
        hash3(key);
        if
        (keyTable[index] == key)
        {

            keyTable[index] =
            EMPTY;
            V oldValue
            = valueTable[index];

            valueTable[index] =
            null;
            size--;
            return
            oldValue;
        }

        return
        removeStash(key);
    }

```

```

        V removeStash (long
key) {
            long[] keyTable
= this.keyTable;
            for (int i =
capacity, n = i +
stashSize; i < n; i++) {
                if
(keyTable[i] == key) {
                    V
oldValue = valueTable[i];

                    removeStashIndex(i);
                    size-
-;

                    return oldValue;
                }
                return null;
            }
}

```

```

        void
removeStashIndex (int
index) {
            // If the
removed location was not
last, move the last tuple
to the removed location.
            stashSize--;
            int lastIndex =
capacity + stashSize;
            if (index <
lastIndex) {

                keyTable[index] =
keyTable[lastIndex];

                valueTable[index] =
valueTable[lastIndex];

                valueTable[lastIndex]
= null;
            } else

                valueTable[index] =
null;
        }
}

```

```

        public void clear ()
{
            long[] keyTable
= this.keyTable;
            V[] valueTable
= this.valueTable;
            for (int i =
capacity + stashSize; i--
> 0;) {
                keyTable[i]
= EMPTY;

                valueTable[i] = null;
            }
            size = 0;
            stashSize = 0;
            zeroValue =
null;
            hasZeroValue =
false;
        }

        /**
         * Returns true if
the specified value is in
the map. Note this
traverses the entire map
and compares every value,
which may be
         * an expensive
operation.
         */
        public boolean
containsValue (Object
value, boolean identity)
{
            V[] valueTable
= this.valueTable;
            if (value ==
null) {
                if
(hasZeroValue &&
zeroValue == null) return
true;

                long[]
keyTable = this.keyTable;
                for (int i
= capacity + stashSize;
i-- > 0;)

```

```

        if
(keyTable[i] != EMPTY &&
valueTable[i] == null)
return true;
    } else if
(identity) {
        if (value
== zeroValue) return true;
        for (int i
= capacity + stashSize;
i-- > 0;)
            if
(valueTable[i] == value)
return true;
        } else {
            if
(hasZeroValue &&
value.equals(zeroValue))
return true;
            for (int i
= capacity + stashSize;
i-- > 0;)
                if
(value.equals(valueTable[
i])) return true;
            }
            return false;
        }

    public boolean
containsKey (long key) {
        if (key == 0)
return hasZeroValue;
        int index =
(int)(key & mask);
        if
(keyTable[index] != key)
{
            index =
hash2(key);
            if
(keyTable[index] != key)
{
                index
= hash3(key);
                if
(keyTable[index] != key)
return
containsKeyStash(key);

```

```

    }
    }
    return true;
}

    private boolean
containsKeyStash (long
key) {
        long[] keyTable
= this.keyTable;
        for (int i =
capacity, n = i +
stashSize; i < n; i++)
            if
(keyTable[i] == key)
return true;
        return false;
    }

    /**
     * Increases the
size of the backing array
to accomodate the
specified number of
additional items. Useful
before adding many
     * items to avoid
multiple backing array
resizes.
     */
    public void
ensureCapacity (int
additionalCapacity) {
        int sizeNeeded
= size +
additionalCapacity;
        if
(sizeNeeded >= threshold)
resize(MathUtils.nextPowe
rOfTwo((int)(sizeNeeded /
loadFactor)));
    }

    private void resize
(int newSize) {
        int oldEndIndex
= capacity + stashSize;

```

```

        capacity =
newSize;
        threshold =
(int)(newSize *
loadFactor);
        mask = newSize
- 1;
        hashShift = 31
-
Integer.numberOfTrailingZ
eros(newSize);
        stashCapacity =
Math.max(3,
(int)Math.ceil(Math.log(n
ewSize)));
        pushIterations
=
Math.max(Math.min(capacit
y, 32),
(int)Math.sqrt(capacity)
/ 4);

        long[]
oldKeyTable = keyTable;
        V[]
oldValueTable =
valueTable;

        keyTable = new
long[newSize +
stashCapacity];
        valueTable =
(V[])new Object[newSize +
stashCapacity];

        size =
hasZeroValue ? 1 : 0;
        stashSize = 0;
        for (int i = 0;
i < oldEndIndex; i++) {
            long key =
oldKeyTable[i];
            if (key !=
EMPTY) putResize(key,
oldValueTable[i]);
        }
    }

```

```

        private int hash2
(long h) {
            h *= PRIME2;
            return (int)((h
^ h >>> hashShift) &
mask);
        }

        private int hash3
(long h) {
            h *= PRIME3;
            return (int)((h
^ h >>> hashShift) &
mask);
        }

        public String
toString () {
            if (size == 0)
return "[]";
            StringBuilder
buffer = new
StringBuilder(32);

            buffer.append('[');
            long[] keyTable
= this.keyTable;
            V[] valueTable
= this.valueTable;
            int i =
keyTable.length;
            while (i-- > 0)
            {
                long key =
keyTable[i];
                if (key ==
EMPTY) continue;

                buffer.append(key);

                buffer.append('=');

                buffer.append(valueT
able[i]);

                break;
            }
            while (i-- > 0)
            {

```

```

        long key =
keyTable[i];
        if (key ==
EMPTY) continue;

        buffer.append(", ");

        buffer.append(key);

        buffer.append('=');

        buffer.append(valueT
able[i]);
    }

    buffer.append(']');
    return
buffer.toString();
}

/**
 * Returns an
iterator for the entries
in the map. Remove is
supported. Note that the
same iterator instance is
returned each
    * time this method
is called. Use the {@link
Entries} constructor for
nested or multithreaded
iteration.
 */
    public Entries<V>
entries () {
        if (entries ==
null)
            entries =
new Entries(this);
        else

        entries.reset();
        return entries;
    }

/**
 * Returns an
iterator for the values
in the map. Remove is

```

```

supported. Note that the
same iterator instance is
returned each
    * time this method
is called. Use the {@link
Entries} constructor for
nested or multithreaded
iteration.
 */
    public Values<V>
values () {
        if (values ==
null)
            values =
new Values(this);
        else

        values.reset();
        return values;
    }

/**
 * Returns an
iterator for the keys in
the map. Remove is
supported. Note that the
same iterator instance is
returned each time
    * this method is
called. Use the {@link
Entries} constructor for
nested or multithreaded
iteration.
 */
    public Keys keys ()
    {
        if (keys ==
null)
            keys = new
Keys(this);
        else

        keys.reset();
        return keys;
    }

    static public class
Entry<V> {
        public long key;

```



```

        map.valueTable[currentIndex] = null;
    }

```

```

        currentIndex =
INDEX_ILLEGAL;
        map.size--;
    }
}

```

```

    static public class
Entries<V> extends
MapIterator<V> implements
Iterable<Entry<V>>,
Iterator<Entry<V>> {
    private Entry<V>
entry = new Entry();

```

```

        public Entries
(LongMap map) {
            super(map);
        }

```

```

        /**
         * Note the
same entry instance is
returned each time this
method is called.
         */

```

```

        public Entry<V>
next () {
            if
(!hasNext) throw new
NoSuchElementException();
            long[]
keyTable = map.keyTable;
            if
(nextIndex == INDEX_ZERO)
{

```

```

                entry.key = 0;

```

```

                entry.value =
map.zeroValue;
            } else {

```

```

                entry.key =
keyTable[nextIndex];

```

```

                entry.value =
map.valueTable[nextIndex];
            }

```

```

        currentIndex =
nextIndex;

        findNextIndex();
        return

```

```

entry;
    }

```

```

        public boolean
hasNext () {
            return
hasNext;
        }

```

```

        public
Iterator<Entry<V>>
iterator () {
            return
this;
        }
    }

```

```

    static public class
Values<V> extends
MapIterator<V> implements
Iterable<V>, Iterator<V>
{

```

```

        public Values
(LongMap<V> map) {
            super(map);
        }

```

```

        public boolean
hasNext () {
            return
hasNext;
        }

```

```

        public V next ()
{
            V value;
            if
(nextIndex == INDEX_ZERO)

```

```

        value
    = map.zeroValue;
        else
            value
    =
    map.valueTable[nextIndex];

        currentIndex =
    nextIndex;

        findNextIndex();
        return
    value;
    }

    public
    Iterator<V> iterator () {
        return
    this;
    }

    }

    static public class
    Keys extends MapIterator
    {
        public Keys
    (LongMap map) {
            super(map);
        }

        public long
    next () {
        long key =
    nextIndex == INDEX_ZERO ?
    0 :
    map.keyTable[nextIndex];

        currentIndex =
    nextIndex;

        findNextIndex();
        return key;
    }

    }
}

```