PEMBUATAN SISTEM *INFINITE WAVE* DALAM PENGEMBANGAN *VIDEO GAME* "*FROM THE DOWNTOWN*"

SKRIPSI

MUHAMMAD SAID AGUNG NAUFAL NASUTION

201401055

PROGRAM STUDI S-1 ILMU KOMPUTER

FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI

UNIVERSITAS SUMATERA UTARA

MEDAN

2024

PEMBUATAN SISTEM *INFINITE WAVE* DALAM PENGEMBANGAN *VIDEO GAME* "*FROM THE DOWNTOWN*"

SKRIPSI

Diajukan untuk melengkapi tugas dan memenuhi syarat memperoleh ijazah Sarjana Ilmu Komputer

MUHAMMAD SAID AGUNG NAUFAL NASUTION
201401055

PROGRAM STUDI S-1 ILMU KOMPUTER
FAKULTAS ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVERSITAS SUMATERA UTARA
MEDAN
2024

PERSETUJUAN

Judul	: PEMBUATAN SISTEM *INFINITE WAVE* DALAM PENGEMBANGAN *VIDEO GAME* "*FROM THE DOWNTOWN*"
Kategori	: SKRIPSI
Nama	: MUHAMMAD SAID AGUNG NAUFAL NASUTION
Nomor Induk Mahasiswa	: 201401055
Program Studi	: SARJANA (S-1) ILMU KOMPUTER
Fakultas	: ILMU KOMPUTER DAN TEKNOLOGI INFORMASI UNIVERSITAS SUMATERA UTARA
Komisi Pembimbing	:

Pembimbing 2	Pembimbing 1
Dr. Jos Timanta Tarigan S.Kom., M.Sc	Dewi Sartika Br Ginting S.Kom., M.Kom
NIP. 198501262015041001	NIP. 199005042019032023

Diketahui/Disetujui Oleh
Program Studi S-1 Ilmu Komputer
Ketua,

Dr. Amalia ST., M.T.
NIP. 197812212014042001

PERNYATAAN

PEMBUATAN SISTEM *INFINITE WAVE* DALAM PENGEMBANGAN *VIDEO GAME "FROM THE DOWNTOWN"*

SKRIPSI

Saya mengakui bahwa skripsi ini adalah hasil karya saya sendiri, kecuali beberapa kutipan dan ringkasan yang masing-masing telah disebutkan sumbernya.

Medan, 9 Juli 2024

Muhammad Said Agung Naufal Nasution
201401055

PENGHARGAAN

Bismillahirrahmanirrahim, puji dan syukur dipanjatkan kepada Allah *Subhanahu Wa Ta'ala* berkat limpahan rahmat-Nya sehingga penulis dapat berada pada tahap penyusunan skripsi ini sebagai syarat untuk mendapatkan gelar Sarjana Komputer di Program Studi S-1 Ilmu Komputer, Fakultas Ilmu Komputer dan Teknologi Informasi, Universitas Sumatera Utara. Shalawat dan salam dicurahkan kepada Rasulullah *Shalallaahu 'Alayhi Wasallam* yang telah membimbing umat manusia dari zaman kegelapan menuju zaman terang benderang saat ini.

Dengan penuh rasa hormat pada kesempatan ini penulis mengucapkan terima kasih sebesar-besarnya kepada Mama tecinta, Nurcahaya Lubis atas segala bentuk kasih sayang serta doa-doa yang dipanjatkan untuk penulis. Dan terima kasih kepada Ayah, Abdul Hadi Nasution atas dukungan dan kasih sayang yang membersamai di setiap langkah penulis. Terima kasih untuk setiap dukungan yang telah diberikan hingga penulis dapat berada di titik ini.

Penyusunan skripsi ini tidak terlepas dari bantuan, dukungan, dan bimbingan dari banyak pihak. Oleh karena itu, penulis mengucapkan banyak terima kasih kepada:

1. Bapak Prof. Dr. Muryanto Amin S.Sos., M.Si. selaku Rektor Universitas Sumatera Utara.
2. Ibu Dr. Maya Silvi Lydia B.Sc., M.Sc. selaku Dekan Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.
3. Bapak Dr. Mohammad Andri Budiman S.T., M.Comp.Sc., M.E.M. selaku Wakil Dekan I Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.
4. Ibu Dr. Amalia, S.T., M.T. selaku Ketua Program Studi S-1 Ilmu Komputer Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.
5. Ibu Dewi Sartika Br. Ginting S.Kom., M.Kom. selaku dosen Pembimbing I yang telah memberikan banyak bimbingan dan masukan yang berharga kepada penulis selama proses penyusunan skripsi ini.
6. Bapak Dr. Jos Timanta Tarigan S.Kom., M.Sc. selaku dosen Pembimbing II yang telah memberikan bimbingan serta masukannya kepada penulis yang memudahkan penulis selama proses penyusunan skripsi ini.

7. Bapak Herriyance S.T., M.Kom. selaku Dosen Pembimbing Akademik yang telah memberikan saran, motivasi dan banyak dukungan kepada penulis selama perkuliahan.
8. Seluruh Bapak dan Ibu Dosen Program Studi S-1 Ilmu Komputer, yang telah membimbing penulis selama masa perkuliahan hingga akhir masa studi.
9. Teristimewa kepada Orang Tua penulis Abdul Hadi Nasution dan Nurcahaya Lubis yang telah memberikan penulis kasih sayang yang tiada henti, ilmu yang bermanfaat, dan berbagai doa bagi penulis sehingga penulis dapat menjalani perkuliahan dengan baik hingga penyusunan skripsi ini.
10. Saudara tercinta Nazla Atikah Hikmatias Nasution dan Ahmad Saiful Rakha Nasution yang selalu mendukung penulis dalam menjalani kehidupan masa kuliah hingga sampai menyelesaikan tugas akhir.
11. Keluarga Besar dari pihak Mama dan Ayah yang selalu percaya, menyemangati dan mendoakan penulis dalam setiap langkah yang diambil untuk mendapatkan gelar S-1.
12. Sahabat *Discord* Ragam Ekesekusi yang selalu sedia pada malam hari untuk menemani penulis selama perkuliahan hingga penyusunan skripsi ini.
13. Teman seperjuangan Kaum Mujahirudin yang telah memberikan banyak semangat dan kata-kata mutiara kepada penulis untuk menyelesaikan skripsi ini.
14. Teman-teman Gamedev dan IOT yang membantu serta bersama-sama penulis dalam menyusun skripsi ini hingga selesai.
15. Fachriza Adrian dan keluarga yang telah memberikan tempat untuk belajar bersama dalam penyelesaian skripsi.
16. Jefta Steven Filemon Sinaga dan keluarga yang selalu memberikan tempat belajar yang nyaman di luar kampus yang membantu penulis untuk menyelesaikan skripsi ini.
17. Dewi Ragil yang menemani dan mendukung penulis dalam menyusun skripsi ini hingga selesai.
18. Stambuk 2020 terkhusus kom B yang telah menemani penulis selama masa perkuliahan dan memberikan pengalaman belajar yang berharga kepada penulis.
19. Abang-kakak senior terkhusus stambuk 2018 yang telah memberikan masukan, motivasi serta doa yang baik kepada penulis selama masa perkuliahan hingga penulisan skripsi ini.

Dan seluruh pihak yang telah memberi dukungan serta doa baik yang tidak dapat penulis sebutkan satu per-satu. Semoga Allah *Subhanahu Wa Ta'ala* senantiasa melimpahkan keberkahan serta kemudahan atas semua dukungan yang telah diberikan kepada penulis dan semoga hasil penelitian ini dapat memberi manfaat maupun inspirasi di masa yang akan datang.

Medan, 9 Juli 2024

Penulis,

Muhammad Said Agung Naufal Nasution

ABSTRAK

Penelitian ini bertujuan untuk mengembangkan dan mengimplementasikan sistem infinite wave ke dalam game "From The Downtown." Sistem ini dirancang untuk menghasilkan gelombang musuh yang tidak terbatas dengan tingkat kesulitan dinamis, yang disesuaikan dengan performa pemain. Dalam penelitian ini, berbagai jenis musuh diperkenalkan, mulai dari musuh normal hingga musuh boss, dengan pembobotan yang dirancang secara spesifik untuk setiap jenis musuh. Metode yang digunakan dalam penelitian ini mencakup perancangan algoritma Dynamic Difficulty Adjustment (DDA) yang memungkinkan tingkat kesulitan musuh meningkat sesuai dengan peningkatan status pemain. Hasil implementasi menunjukkan bahwa sistem infinite wave ini mampu memunculkan berbagai jenis musuh, yaitu empat jenis musuh normal (*melee*, *chaser*, *range*, dan *charger*) dan dua jenis musuh *boss* (melee *boss* dan range *boss*). Musuh normal memiliki pembobotan masing-masing: *melee* 35%, *chaser* 25%, *range* 20%, dan *charger* 20%. Musuh *melee* dan *chaser* mulai muncul pada *wave* 1, musuh *range* pada *wave* 6, dan musuh *charger* pada *wave* 11. Musuh boss muncul setiap 5 *wave* sekali secara bergantian. Jumlah musuh yang dimunculkan bertambah 5 setiap *wave*, dimulai dari 10 musuh dan maksimal 50 musuh per *wave*. Setiap 15 *wave*, jumlah musuh kembali menjadi 10 dengan status yang meningkat. Pada pergantian *wave*, panel upgrade muncul untuk meningkatkan status pemain. Penelitian ini menegaskan bahwa sistem *infinite wave* dapat meningkatkan pengalaman bermain dengan menghadirkan tingkat kesulitan yang selalu menantang dan seimbang, serta memberikan kontribusi terhadap pengembangan sistem *infinite wave* dalam game.

Kata Kunci: *Game Development, Infinite Wave, Dynamic Difficulty Adjustment (DDA)*

CREATION OF INFINITE WAVE SYSTEM IN THE DEVELOPMENT OF VIDEO GAME “FROM THE DOWNTOWN”

ABSTRACT

This research aims to develop and implement an infinite wave system into the game "From The Downtown." This system is designed to generate infinite waves of enemies with dynamic difficulty levels, which are adjusted according to the player's performance. In this research, various types of enemies are introduced, ranging from normal enemies to boss enemies, with weightings specifically designed for each type of enemy. The method used in this research includes the design of a Dynamic Difficulty Adjustment (DDA) algorithm that allows the enemy difficulty to increase according to the player's status improvement. The implementation results show that the infinite wave system is capable of generating various types of enemies, namely four types of normal enemies (melee, chaser, range, and charger) and two types of boss enemies (melee boss and range boss). Normal enemies have their respective weightings: melee 35%, chaser 25%, range 20%, and charger 20%. Melee and chaser enemies start appearing in wave 1, range enemies in wave 6, and charger enemies in wave 11. Boss enemies appear once every 5 waves in turn. The number of enemies spawned increases by 5 every wave, starting from 10 enemies and a maximum of 50 enemies per wave. Every 15 waves, the number of enemies returns to 10 with increased status. At the turn of the wave, an upgrade panel appears to increase the player's status. This research confirms that the infinite wave system can improve the game experience by providing a challenging and balanced difficulty level, and contributes to the development of the infinite wave system in games.

Keywords: Game Development, Infinite Wave, Dynamic Difficulty Adjustment (DDA)

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1

PENDAHULUAN

1.1 Latar Belakang

Pengembangan *video game* telah menjadi sebuah arena kreativitas yang berkembang pesat, menawarkan pengalaman hiburan yang menantang dan mengasyikkan bagi para pemainnya. Salah satu tren yang semakin populer dalam dunia pengembangan *game* adalah konsep *infinite level* atau *endless level*, di mana tujuan utamanya adalah untuk bertahan sepanjang waktu dengan menghadapi berbagai rintangan, mengalahkan musuh, atau mengumpulkan skor sebanyak mungkin (Padrones, 2022).

Variasi menarik dari konsep *infinite level* yang patut diperhatikan adalah *game* dengan sistem *infinite wave*, di mana pemain dihadapkan pada gelombang musuh yang terus meningkat tanpa batas. Sistem *infinite wave* ini menuntut pemain untuk bertahan sekaligus menyerang musuh yang terus muncul. Namun, salah satu tantangan krusial dalam mengimplementasikan sistem ini adalah menjaga tingkat kesulitan agar tetap sesuai dengan kemampuan pemain. Tingkat kesulitan yang terlalu mudah dapat membuat permainan membosankan, sementara tingkat kesulitan yang terlalu tinggi dapat menimbulkan frustrasi pada pemain (Zohaib, 2018).

Solusi untuk tantangan ini terletak pada penerapan konsep kesulitan yang dinamis, dimana tingkat kesulitan permainan menyesuaikan secara otomatis berdasarkan kemampuan pemain. Untuk menciptakan pengalaman bermain yang optimal dan menantang, perlu diadopsi konsep *flow channel* (Zohaib, 2018), di mana tingkat kesulitan dan keterampilan pemain seimbang dengan baik, sehingga dengan demikian akan menciptakan kondisi psikologis dimana pemain akan merasa sepenuhnya terlibat dan tetap fokus ke dalam permainan. Dan pada akhirnya akan berdampak baik pada pengalaman bermain yang menantang dan mengasyikkan.

Terdapat penelitian terdahulu yang mendekati topik ini yaitu penelitian yang dilakukan oleh Daniel Hind dan Carlo Harvey yang berjudul "*A NEAT Approach to Wave Generation in Tower Defense Games*" menjelaskan dalam penelitian tersebut penggunaan jaringan saraf tiruan sebagai pengelola gelombang musuh dalam *game tower defense*. Jaringan saraf tiruan tersebut akan mengatur jumlah musuh dan jarak waktu kemunculannya dalam bentuk gelombang-gelombang (Hind & Harvey, 2022).

Dari fokus permasalahan yang telah dijelaskan di atas maka pada penelitian ini akan dikembanganken sebuah *game* bernama "*From The Downtown*," yaitu sebuah *game* dengan tema *zombie apocalypse* yang dibuat dengan *genre top-down shooter* yaitu *game* tembak-menembak dengan sudut pandang dari atas karakter pemain (Adams, 2010). Musuh utama dalam *game* ini berupa *zombie* dengan berbagai jenis dan kemampuannya yang beragam. Tujuan utama dalam *game* ini adalah bertahan hidup sebaik mungkin sambil menghadapi serbuan *zombie* yang semakin intens. *Game* ini hanya memiliki satu *level* yang *infinite* atau tidak terbatas. Ketika *game* ini dimulai akan memunculkan karakter pemain yang berjauhan dari tempat munculnya para *zombie*, kemudian para *zombie* akan mendatangi pemain dan menyerang pemain dan pemain harus mengalahkan para *zombie* yang muncul sebanyak-banyaknya. Contoh dari *game top-down shooter* dengan sistem *infinite wave* seperti *Geometry Wars* (Activision, 2007), *Vampire Survivors* (poncle, 2022), *Crimsonland* (10tons Ltd, 2014) dan *Tesla vs Lovecraft* (10tons Ltd, 2018).

Fokus utama pada penelitian ini adalah membuat sistem *infinite wave* yang akan mengatur kemunculan musuh atau *enemy spawn* dengan berbagai varian serangan, ketahanan, dan kekuatannya. Sistem ini juga akan menyesuaikan tingkat kesulitan musuh yang muncul pada gelombang yang tak terbatas tersebut secara dinamis berdasarkan status pemain. Melalui penelitian ini, diharapkan dapat dihasilkan sebuah *game* yang tidak hanya menarik namun juga memberikan tantangan yang optimal bagi para pemain.

1.2 Rumusan Masalah

Tantangan terbesar untuk mengimplementasikan sistem *infinite wave* ke dalam *video game* yaitu menjaga tingkat kesulitan agar tetap sesuai dengan kemampuan

pemain dalam artian tidak terlalu sulit dan tidak terlalu mudah. Sehingga pada penelitian ini akan dibuat sistem *infinite wave* yang akan mengatur kemunculan musuh dengan berbagai varian serangan, ketahanan, dan kekuatannya. Sistem ini juga akan menyesuaikan tingkat kesulitan musuh yang muncul pada gelombang yang tak terbatas tersebut secara dinamis berdasarkan status pemain.

1.3 Batasan Masalah

Batasan masalah dalam penelitian ini adalah sebagai berikut:

1. Penelitian ini akan mengimplementasikan sistem *infinite wave* ke dalam *video game* "*From The Downtown*".
2. *Game* akan dibangun dengan *genre top-down shooter*.
3. Sistem *infinite wave* hanya memunculkan jenis-jenis musuh yang terdapat di dalam *game*.
4. Sistem *infinite wave* tersebut muncul dalam bentuk gelombang musuh yang tidak berujung atau *infinite*.
5. Sistem *infinite wave* akan dibangun dengan menyesuaikan tingkat kesulitannya secara dinamis berdasarkan statistic pemain.
6. Pembuatan sistem *infinite wave* pada penelitian ini akan dilakukan dengan menggunakan platform *game* engine Unity dan dengan menggunakan bahasa pemrograman C#.

1.4 Tujuan Penelitian

Membuat sistem *infinite wave* yang dapat mengatur kemunculan musuh dengan berbagai macam jenis mulai dari bentuk serangan, ketahanan dan kekuatannya, serta menyesuaikan tingkat kesulitannya secara dinamis berdasarkan status pemain ke dalam *game* "*From The Downtown*".

1.5 Manfaat Penelitian

Manfaat dari dilakukannya penelitian ini adalah sebagai berikut:

1. Menjadikan *game* yang lebih dinamis dan menarik untuk dimainkan berkali-kali.
2. Inovasi dalam *game* yang menggunakan sistem *infinite wave* dapat memberikan daya tarik *game* dalam jangka panjang. Memanfaatkan sistem *infinite wave* dengan tingkat kesulitan yang dinamis akan meningkatkan

kemampuan pemain dalam mengembangkan st*rate*gi yang lebih baik untuk dapat bertahan dari gelombang musuh yang terus muncul tak terbatas.

3. Mampu memberikan pengalaman bermain yang menantang bagi para pemain.
4. Dapat menjadi referensi bagi pengembangan sistem *infinite wave* di bidang *video game* maupun di bidang-bidang yang lain untuk penelitian yang akan datang.

1.6 Metodologi Penelitian

Beberapa metode yang diterapkan dalam penelitian ini adalah sebagai berikut:

1. Studi Pustaka

   Pada tahap ini penelitian dimulai dengan mencari referensi dari berbagai sumber terpercaya dan melakukan peninjauan pustaka melalui buku-buku, jurnal, *e-book,* artikel ilmiah, makalah ataupun situs internet yang berhubungan dengan *Game Development*, *Infinite Wave*, *Infinite Level*, *Dynamic Difficulty*, *Top Down Shooter* dan pengimplementasiannya ke dalam *Unity*.

2. Analisa dan Perancangan

   Pada tahap ini dilakukan sebuah perancangan dari *video game* yang akan dibuat sesuai dengan permasalahan dan perancangan *video game*. Perancangan *video game* ini akan dirancang dengan pembuatan *flowchart*.

3. Implementasi

   Pada tahap ini, membuat sebuah *game* dengan menggunakan *game engine Unity* dan menggunakan bahasa pemrograman C# sesuai dengan *flowchart* yang telah dirancang.

4. Pengujian

   Pada tahap ini, dilakukan uji coba pada *video game* yang telah dibuat berhasil dijalankan, serta mampu menjalankan sistem *infinite wave* pada *video game* tersebut. Akan diuji bagaimana sistem *infinite wave* mampu mengatur kemunculan musuh dengan berbagai macam jenis mulai dari bentuk serangan, ketahanan dan kekuatannya, serta menyesuaikan tingkat kesulitannya secara dinamis berdasarkan kemampuan atau keterampilan pemain.

5. Dokumentasi

Pada tahap ini, penelitian yang telah dilakukan, didokumentasikan mulai dari tahap analisa sampai kepada pengujian dalam bentuk skripsi.

1.7 Penelitian Relevan

Beberapa penelitian terdahulu yang relevan dengan penelitian ini, antara lain:

1. Penelitian yang dilakukan oleh Daniel Hind dan Carlo Harvey yang berjudul "A NEAT Approach to Wave Generation in Tower Defense *Game*s" menjelaskan dalam penelitian tersebut penggunaan jaringan saraf tiruan sebagai pengelola gelombang musuh dalam *game tower defense*. Jaringan saraf tiruan tersebut akan mengatur jumlah musuh dan jarak waktu kemunculannya dalam bentuk gelombang-gelombang (Hind & Harvey, 2022).
2. Penelitian yang dilakukan oleh Victor Christian Tania, Jeanny Pragantha dan Darius Andana Haris yang berjudul "Pembuatan *Game* Shoot 'Em Up "Stop The *Alien*s" Dengan Fitur Accelerometer Menggunakan Unity Berbasis Android" menjelaskan *game* yang dikembangkan pada penelitian tersebut memiliki desain *level* berupa *endless level*, yang mana dalam *game* tersebut mengharuskan pemain untuk bertahan hidup selama mungkin dengan mengalahkan musuh yang dapat menembakkan peluru serta mampu bergerak lebih cepat, yang akan meningkat seiring berjalannya waktu, mendorong pemain untuk terus berusaha bertahan hidup dan mencetak skor tertinggi. (Tania et al., 2022).
3. Pada penelitian yang berujudul "Applying Hidden Markov Model for Dynamic *Game* Balancing" menjelaskan bahwa penerapan machine learning menggunakan metode Hidden Markov Model (HMM), terbukti efektif dalam meningkatkan keseimbangan dan tingkat kesulitan permainan video secara dinamis. Melalui penelitian ini, sebuah *game* tembak-menembak telah dibuat dan diujikan dengan partisipasi sukarelawan. Pada penelitian tersebut menunjukkan bahwa permainan yang menggunakan pendekatan HMM memberikan tantangan yang lebih signifikan bagi pemain.

1.8 Sistematika Penulisan

Sistematika penulisan skripsi yang digunakan dalam penelitian ini adalah sebagai berikut:

BAB 1 PENDAHULUAN

Bab ini mencakup penjelasan mengenai latar belakang pemilihan judul, rumusan dan batasan masalah, tujuan, manfaat, dan metodologi penelitian, penelitian relevan, dan sistematika penulisan skripsi.

BAB 2 LANDASAN TEORI

Bab ini menjelaskan beberapa teori yang berkaitan dengan penelitian, yaitu *infinite wave*, *Dynamic Difficulty Adjustment* (DDA) dan *Top-Down Shooter Game.*

BAB 3 ANALISIS DAN PERANCANGAN

Bab ini menjelaskan mengenai analisis dan dilakukan perancangan diagram yang diperlukan, seperti arsitektur sistem.

BAB 4 IMPLEMENTASI DAN PENGUJIAN

Bab ini berisi penjelasan mengenai implementasi sistem ke dalam *game* yang kemudian akan diuji dengan memainkan langsung game tersebut.

BAB 5 KESIMPULAN DAN SARAN

Bab ini berisi kesimpulan yang dapat diperoleh berdasarkan pemaparan pada setiap bab serta saran yang diberikan peneliti sebagai masukan untuk penelitian selanjutnya.

BAB 2

LANDASAN TEORI

2.1 *Infinite Wave*

Pada dasarnya sistem *infinite wave* merupakan pengembangan dari sistem *enemy wave* atau gelombang musuh. *Enemy wave* merupakan serangkaian atau gelombang musuh yang muncul secara bersamaan atau berturut-turut selama permainan. *Enemy wave* dirancang untuk memberikan tingkat kesulitan yang bertingkat dalam permainan. Seiring berjalannya waktu, gelombang musuh menjadi lebih sulit dan kompleks yang akan mendorong pemain untuk meningkatkan strategi permainan.

Konsep dasar pada *enemy wave* adalah gelombang musuh di dalam *game* yang akan diatur waktu kemunculan dan jumlah musuh yang akan dimunculkan. Misalnya dalam detik ke-1 sampai dengan detik ke-5 gelombang pertama akan muncul 2 musuh per detiknya, sehingga total musuh yang muncul adalah 10 pada 5 detik pertama, kemudian di detik 25 akan muncul lagi gelombang musuh selanjutnya dan begitulah seterusnya. Dari konsep dasar ini kemudian dapat dikembangkan menjadi gelombang musuh yang tidak terbatas atau *infinite wave* (Borromeo, 2021).

Gelombang musuh pada sistem *infinite wave* merupakan kondisi dimana gelombang musuh akan muncul secara terus menerus tanpa batas. Dengan demikian dalam pengembangannya sistem *infinite wave* dapat menjadi lebih dinamis daripada *enemy wave* pada umumnya. Dengan adanya sistem *infinite wave* ini akan memberikan kesempatan lebih banyak kepada pemain untuk mengembangkan kemampuan serta st*rate*gi untuk bertahan sekaligus menyerang di dalam permainannya. Contoh *game* dengan penggunaan sistem *infinite wave* adalah *Geometry Wars* (Activision, 2007).

Pada *game infinite wave* ini menentukan *spawn point* akan sangat berpengaruh dalam permainan. *Spawn point* merupakan titik lokasi munculnya musuh di dalam *game*. *Spawn point* tersebut akan disebar ke beberapa titik atau lokasi st*rate*gis di dalam peta permainan dan diletakkan berjauhan dari lokasi karakter pemain pertama kali. Kemudian dari titik-titik itu akan muncul musuh

dengan waktu yang random, sehingga musuh akan dapat muncul dalam waktu yang bersamaan dari titik yang berbeda. Dan dengan *spawn point* ini tingkat kesulitan musuh yang keluar dapat disesuaikan secara dinamis (Pereira et al., 2021).

2.2 *Dynamic Difficulty Adjustment* (DDA)

Penyesuaian tingkat kesulitan dinamis (*Dynamic Difficulty Adjustment*/DDA) adalah suatu proses secara otomatis mengubah tingkat kesulitan dalam sebuah permainan untuk mengoptimalkan pengalaman pemain. Kesulitan permainan seharusnya sesuai agar seorang pemain tidak merasa bosan ketika permainan terlalu mudah, dan tidak merasa frustrasi ketika permainan terlalu sulit. DDA umumnya diterapkan untuk setiap pemain berdasarkan kemampuan, keterampilan, dan tindakan yang diamati oleh pemain tersebut (Zohaib, 2018).

Kesulitan permainan untuk menyediakan tingkat kesulitan yang tepat untuk semua orang dianggap sebagai salah satu alasan utama ketidakpuasan pemain. Seorang pemain memerlukan tantangan yang terus-menerus untuk tetap terlibat dalam permainan (Or et al., 2021). Ketika kemampuan pemain dan tantangan dalam permainan seimbang dengan baik maka pemain memasuki *flow*-state, yaitu pemain akan merasakan kepuasan yang mendalam. Oleh karena itu, sebagian besar pada *single player games* atau *game* dengan pemain tunggal akan secara terus-menerus meningkatkan tingkat kesulitan selama permainan berlangsung (Huber et al., 2021).

2.3 *Top-Down Shooter Game*

Top-Down Shooter Game merupakan salah satu *genre* yang ada dalam *video game*. Penjelasan *Top-Down Shooter Game* dapat dibagi menjadi dua bagian, yaitu *Top-down* dan *Shooter Game*.

2.3.1 Top-down

Top-down merupakan perspektif dari atas ke bawah yang menunjukkan dunia permainan dari atas secara langsung dengan kamera mengarah lurus ke bawah. Biasanya *game top-down* dibuat pada *game* bertema 2D atau dua dimensi, namun seiring perkembangan teknologi *top-down* dapat diimplementasikan pada *game* bertema 3D atau tiga dimensi (Adams, 2010).

2.3.2 Shooter game

Shooter game merupakan *game* dengan mekanisme permain menembak dan mengalahkan musuh. Dalam *shooter game* sederhana, permainan dimulai dengan beberapa sumber daya seperti musuh yang sudah ada di dunia permainan, namun lebih banyak musuh dapat muncul di *spawn point*. Pada *shooter game* musuh merupakan bagian dari ekonomi permainan, dengan mengalahkan musuh pemain mendapatkan keuntungan seperti koin, item atau *exp*erience *point* yang dapat ditukarkan untuk meningkatkan kemampuan pemain (Adams, 2010).

Contoh *game* dengan *genre top-down shooter* adalah *game* berjudul *Alien Shooter* (Sigma Team Inc., 2003) yang bisa dilihat pada Gambar 2. 1.

Gambar 2. 1 *Alien Shooter* (Sigma Team Inc., 2003)

BAB 3

ANALISIS DAN PERANCANGAN

3.1 Analisis

Analisis merupakan tahap awal sebelum melakukan perancangan dan pengembangan sistem. Tahap analisis memberikan pemahaman pada peneliti tentang masalah yang dihadapi dalam pengembangan sistem. Hal ini dilakukan agar sistem yang dikembangkan dapat memenuhi kebutuhan yang ada dan dapat mencapai tujuan akhirnya.

3.1.1 Analisis masalah

Dalam pembuatan sistem *infinite wave* ke dalam *game* terdapat beberapa masalah utama yang muncul. Pertama adalah variasi musuh, dalam pembuatan sistem *infinite wave* harus sangat memperhatikan jenis-jenis musuh yang akan dimunculkan mulai dari bentuk serangannya serta pengaruhnya dalam kesulitan *game* ketika dimainkan. Kedua adalah menyesuaikan kesulitan yang tetap sesuai dengan pemain sehingga pemain tetap mendapatkan pengalaman yang menarik tanpa perlu merasa frustasi karena *game* terlalu sulit dan tidak merasa bosan karena *game* terlalu mudah.

3.1.2 Analisis kebutuhan

Analisis kebutuhan merupakan langkah selanjutnya yang bertujuan untuk mengidentifikasi kebutuhan apa saja yang diperlukan agar sistem yang dirancang dapat mencapai tujuan yang ditetapkan. Proses ini terbagi menjadi dua aspek utama, yaitu fungsional dan non-fungsional.

1. Kebutuhan fungsional

   Kebutuhan fungsional adalah kumpulan fungsi yang harus ada dalam sistem untuk memenuhi kebutuhan pengguna yang dalam hal ini merupakan pemain. Kebutuhan ini mencakup spesifikasi fungsionalitas yang harus ada pada sistem untuk mencapai tujuan utama. Penelitian ini memiliki kebutuhan fungsional utama, yaitu:

a. Melakukan penyesuaian kemunculan variasi musuh yang ada dengan menangani urutan kemunculannya untuk memberikan tantangan yang berkelanjutan selama permainan.
b. Mengatur gelombang musuh dan menentukan jumlah musuh yang akan muncul setiap gelombangnya.
c. Menyesuaikan pembobotan jenis-jenis musuh berdasarkan bentuk serangannya dan pengaruhnya pada kesulitan permainan.
d. Melakukan proses penyesuaian kesulitan yang dinamis (*Dynamic Difficulty Adjustment*) dengan status pemain sebagai parameter utamanya.
e. Status pemain ditingkatkan dengan menggunakan sistem *upgrade* penambahan nilai.

2. Kebutuhan non-fungsional

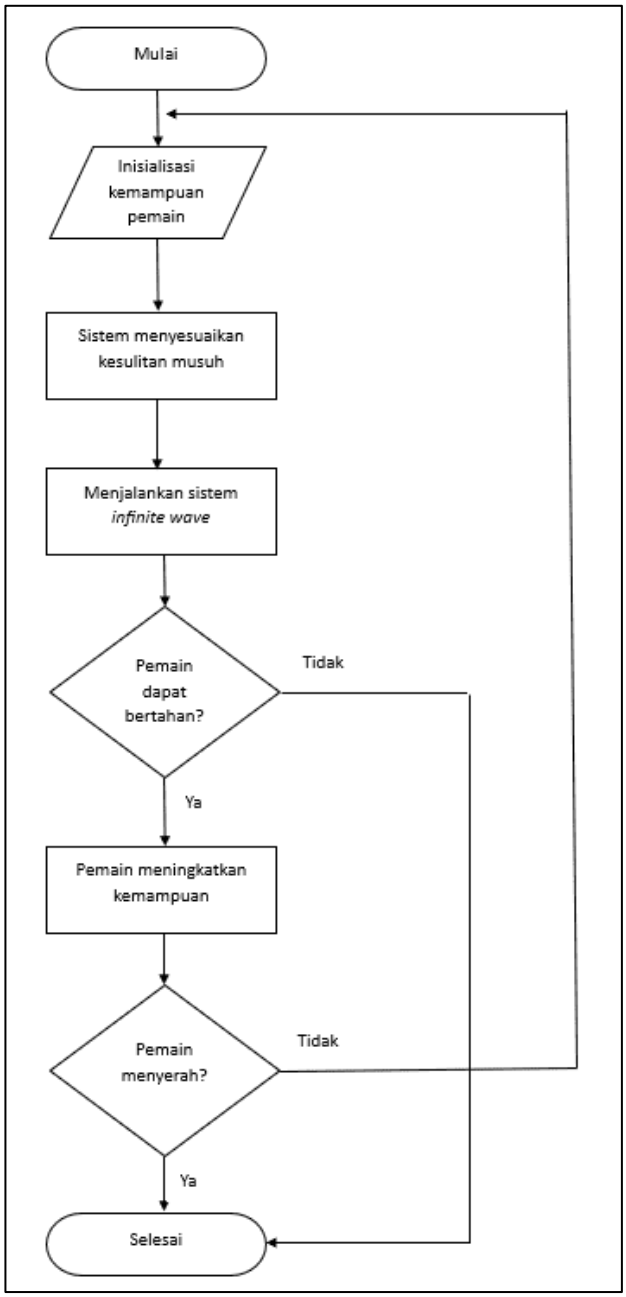
   Kebutuhan non-fungsional adalah spesifikasi tambahan yang mendukung sistem, dapat berupa kinerja maupun batasan dari sistem. Penelitian ini memiliki beberapa kebutuhan non-fungsional, yaitu:

   a. *Game*play mudah dipahami.
   b. Visual, efek, serta mekanika permainan yang menarik.
   c. Sistem persenjataan yang variatif.
   d. Sistem dapat melakukan drop item yang jatuh dari musuh yang dikalahkan dan dapat ditukarkan dengan item yang diperlukan selama permainan berlangsung.

3.2 Arsitektur Umum Sistem

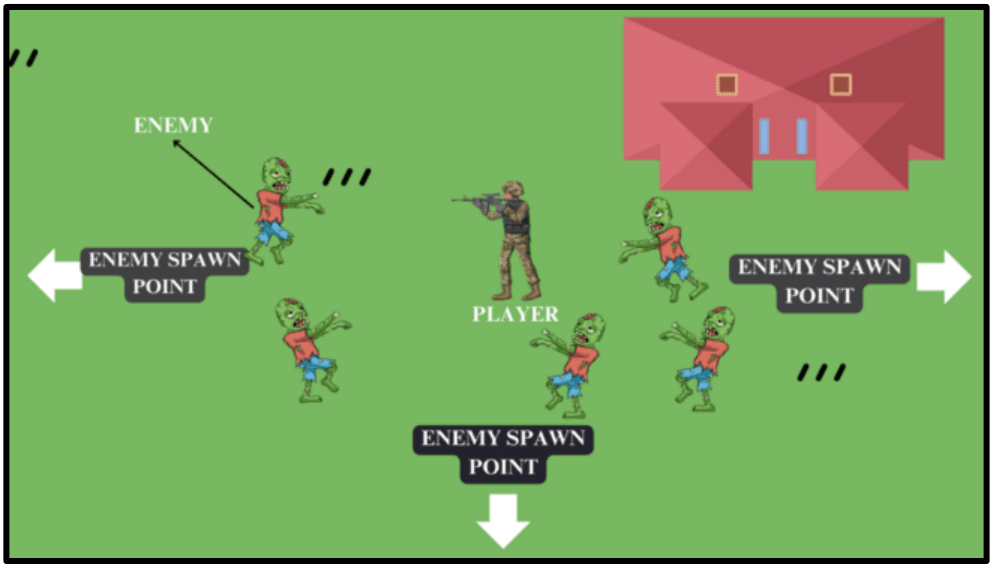
Rangkaian alur sistem infinite wave dalam permainan dimulai dengan menginisialisasi elemen penting dan status pemain. Sistem kemudian menyesuaikan kesulitan berdasarkan nomor gelombang saat ini. Setelah itu, sistem menjalankan gelombang musuh tak terbatas. Permainan terus berlangsung dengan sistem yang memeriksa apakah semua musuh telah dikalahkan dalam gelombang saat ini. Jika pemain dapat mengalahkan semua musuh, maka pemain diberi kesempatan untuk meningkatkan kemampuannya melalui panel upgrade yang muncul. Setelah pemain meningkatkan kemampuannya, sistem akan mengecek

apakah pemain ingin melanjutkan permainan atau tidak. Jika pemain memilih untuk melanjutkan permainan, sistem akan melanjutkan ke gelombang berikutnya dengan menyesuaikan bobot *spawn* musuh dan menjalankan gelombang baru. Namun, jika pemain memilih untuk menyerah atau keluar permainan, maka permainan selesai. Siklus ini akan terus berulang, membentuk alur permainan yang tak terbatas. Untuk lebih jelasnya dapat dilihat pada Gambar 3. 1.



Gambar 3. 1 Arsitektur Umum Sistem

Rancangan awal pada sistem *infinite wave* adalah sebagai berikut: Pemain atau player akan diposisikan jauh dari titik munculnya musuh atau *enemy spawn point*. Musuh berupa *zombie* akan muncul secara berkala dari titik-titik *spawn* ini dan akan langsung bergerak untuk menyerang pemain. Pemain harus bertahan dari serangan *zombie* dan mengalahkan setiap *zombie* yang datang. Setiap *zombie* yang dikalahkan akan memberikan pemain *experience points* (*exp*), yang dapat digunakan untuk meningkatkan status pemain melalui panel *upgrade* yang tersedia setelah setiap gelombang. Selain itu, *zombie* yang dikalahkan juga akan menjatuhkan *item-item* yang dapat dikumpulkan oleh pemain. *Item-item* ini bisa digunakan atau ditukarkan dengan barang-barang yang diperlukan selama permainan. Setelah mengalahkan semua musuh dalam satu gelombang, pemain akan diberi waktu untuk menggunakan panel upgrade guna meningkatkan kemampuan mereka sebelum melanjutkan ke gelombang berikutnya. Siklus ini akan terus berulang, dengan pemain menghadapi gelombang musuh yang semakin sulit, mengumpulkan *exp* dan item, serta melakukan *upgrade* untuk bertahan selama mungkin dalam permainan yang tak terbatas ini. Gambaran rancangan awal tersebut dapat dilihat pada Gambar 3. 2.



Gambar 3. 2 Arsitektur Umum Sistem

3.3 Perancangan Sistem *Infinite Wave*

Menentukan jenis musuh yang akan muncul dalam *game* adalah langkah pertama dalam merancang sistem *infinite wave*. Setiap musuh memiliki tingkat kesehatan, kekuatan serangan dan jenis serangan yang berbeda. Selanjutnya, kalkulasi *Adjustment Dynamic Difficulty* (DDA) dilakukan untuk mengubah tingkat kesulitan permainan secara otomatis sesuai dengan performa pemain dan tingkat kesulitan yang diinginkan.

Ketika di-*spawn*, juga perlu dibuat pembobotan jenis musuh, yang berguna untuk menentukan persentase dimunculkannya jenis musuh dalam satu *wave* atau gelombang musuh. Terakhir, sistem harus dirancang untuk memiliki siklus gelombang yang tidak terbatas, dengan setiap gelombang baru dimulai setelah gelombang sebelumnya selesai. Gelombang baru mungkin memiliki lebih banyak musuh atau dapat memunculkan jenis musuh yang baru untuk menyesuaikan tantangan dengan tingkat kesulitan gelombang sebelumnya.

3.3.1 Jenis musuh

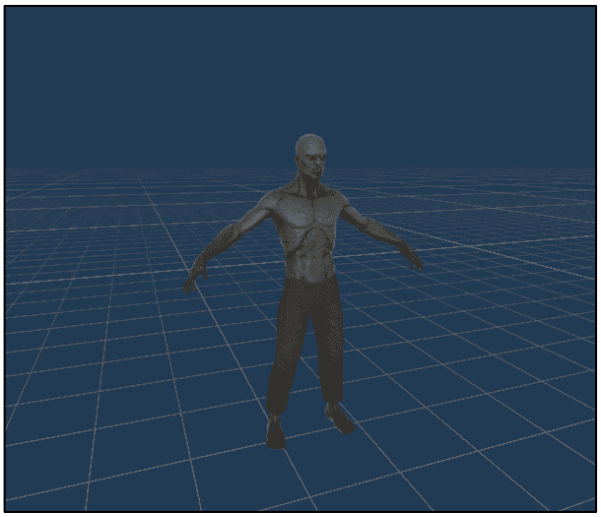
Jenis musuh yang akan dimunculkan pada sistem *infinite wave* terbagi menjadi dua, yaitu sebagai berikut:

1. Musuh Normal

   Musuh normal merupakan jenis musuh yang akan dimunculkan selama permainan berlangsung. Pada *game* ini terdapat 4 musuh normal yaitu *Melee*, *Chaser*, *Range* dan *Charger*.

   a. *Melee*

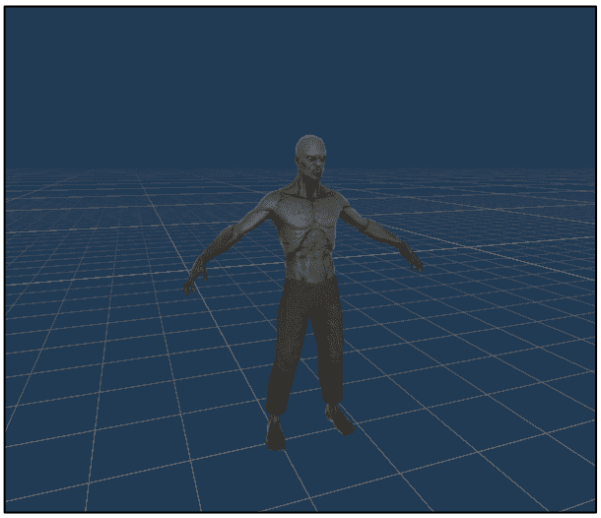
      Melee merupakan jenis musuh yang akan menyerang dengan jarak yang dekat. *Melee* akan berjalan menuju ke arah pemain dan pada jarak yang dekat akan langsung menyerang pemain dengan mencakar pemain. Tampilan *melee* bisa dilihat pada Gambar 3. 3.



Gambar 3. 3 Musuh *Melee*

b. *Chaser*

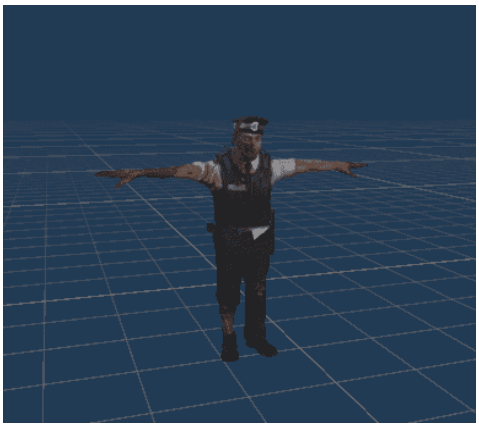
Chaser merupakan musuh yang akan menyerang dengan jarak dekat. *Chaser* akan berlari dengan sangat cepat menuju ke arah pemain dan pada jarak yang dekat akan langsung menyerang pemain dengan mencakar pemain. Tampilan *chaser* memiliki tampilan yang sama dengan *melee* dan bisa dilihat pada pada Gambar 3. 4.



Gambar 3. 4 Musuh *Chaser*

c. *Range*

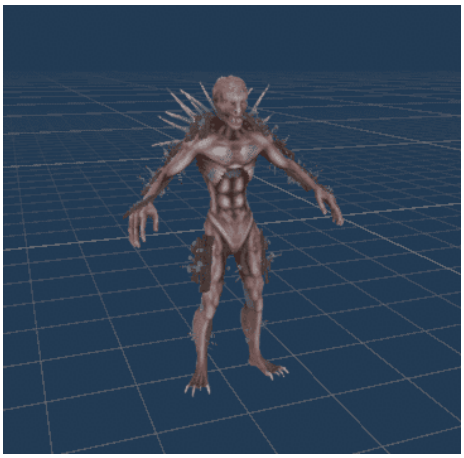
Range merupakan musuh yang akan menyerang pemain pada jarak yang jauh. *Range* akan mengeluarkan cairan beracun ke arah pemain dan ketika pemain terkena racun tersebut akan mendapatkan efek *poison* atau beracun dimana *health point* akan berkurang secara perlahan. Tampilan *range* bisa dilihat pada pada Gambar 3. 5.



Gambar 3. 5 Musuh *Range*

d. *Charger*

Charger merupakan musuh yang akan menyerang pemain pada jarak yang sedang. *Charger* berlari ke arah pemain dan pada jarak yang pas akan mengambil ancang-ancang dan langsung melompat ke arah pemain. Tampilan *charger* dapat dilihat pada Gambar 3. 6.



Gambar 3. 6 Musuh *Charger*

Musuh normal dapat dibagi tingkat kesulitannya berdasarkan variasi serangannya menjadi 3 tingkatan. Tingkatan kesulitan tersebut dapat dilihat pada Tabel 3. 1.

Tabel 3. 1 Tabel Tingkatan Kesulitan Musuh

Jenis Musuh	Nilai Pembobotan Musuh
Melee	Easy
Chaser	Normal
Range	Hard
Charger	Hard

Bentuk dan variasi serangan musuh normal dipengaruhi oleh jarak antara musuh dan pemain. Adapun bentuk variasi dan jenis serangan dari masing-masing musuh normal dapat dilihat pada table 3. 2.

Tabel 3. 2 Tabel Bentuk Variasi Serangan Musuh

Jenis Musuh	Jarak Serangan	Bentuk Serangan (*Enemy Behaviour*)
Melee	Dekat (1 meter)	- Saat dimulai musuh *melee* akan mendekati pemain dengan cara berjalan ke arah pemain. - Jika pemain memasuki jarak serang dari musuh *melee*, maka musuh *melee* akan menyerang pemain dengan cara mencakar pemain.

		- Jika pemain terkena serangan tersebut, maka pemain akan mendapatkan *damage* atau kerusakan yang ringan sehingga dapat menyebabkan darah pemain berkurang. - Jika pemain berada di luar jarak serang musuh *melee*, maka musuh *melee* akan mendekati pemain dengan cara berjalan ke arah pemain.
Chaser	Dekat (1 meter)	- Saat dimulai musuh *chaser* akan mendekati pemain dengan cara berlari ke arah pemain. - Jika pemain memasuki jarak serang dari musuh *chaser*, maka musuh *chaser* akan menyerang pemain dengan cara mencakar pemain. - Jika pemain terkena serangan tersebut, maka pemain akan mendapatkan *damage* atau kerusakan yang ringan dan dapat menyebabkan darah pemain berkurang.

		- Jika pemain berada di luar jarak serang musuh *chaser*, maka musuh *chaser* akan mendekati pemain dengan cara berlari ke arah pemain.
Range	Jauh (7 meter)	- Saat dimulai musuh *range* akan mendekati pemain dengan cara berlari ke arah pemain. - Jika pemain memasuki jarak serang dari musuh *range*, maka musuh *range* akan menyerang pemain dengan cara menyemburkan cairan racun ke arah pemain sebanyak tiga kali. - Jika pemain terkena cairan racun tersebut maka pemain akan terkena efek *poison* atau efek beracun yang mengakibatkan darah pemain akan terus berkurang selama beberapa detik. - Jika pemain berada di luar jarak serang musuh *range*, maka musuh *range* akan mendekati pemain dengan cara

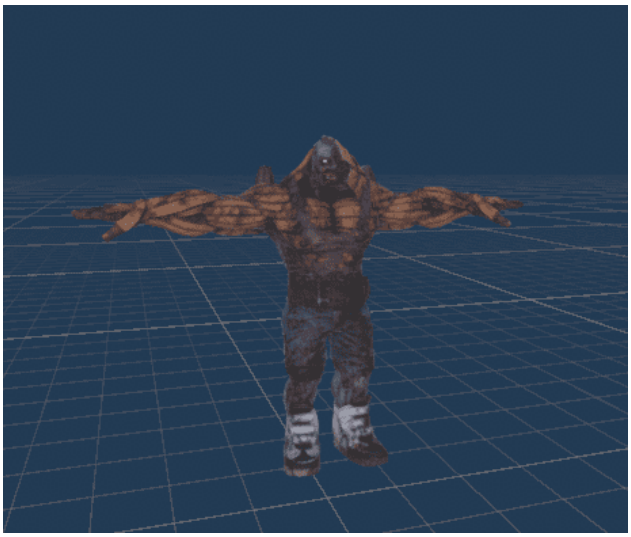
		berjalan cepat ke arah pemain.
Charger	Sedang (5 meter)	- Saat dimulai musuh *charger* akan mendekati pemain dengan cara berlari ke arah pemain. - Jika pemain memasuki jarak serang dari musuh *charger*, maka musuh *charger* akan menyerang pemain dengan cara melompat secara langsung ke arah pemain. - Jika pemain terkena serangan tersebut, maka pemain akan mendapatkan *damage* atau kerusakan yang sangat besar dan dapat menyebabkan darah pemain berkurang drastis. - Jika pemain berada di luar jarak serang musuh *melee*, maka musuh *melee* akan mendekati pemain dengan cara berjalan ke arah pemain.

2. Musuh *Boss*

Musuh boss muncul ketika setiap pemain berhasil melewati 10 *wave* atau gelombang musuh. Terdapat 2 musuh *boss*, yaitu *Melee Boss* dan *Range Boss*. Setiap *Boss* memiliki 2 mode yaitu mode normal dan mode *rage* dan bentuk serangannya akan berbeda pada setiap mode. Mode normal merupakan mode pada awal musuh *boss* muncul dan mode *rage* merupakan mode ketika darah atau *health* musuh *boss* tersisa sedikit.

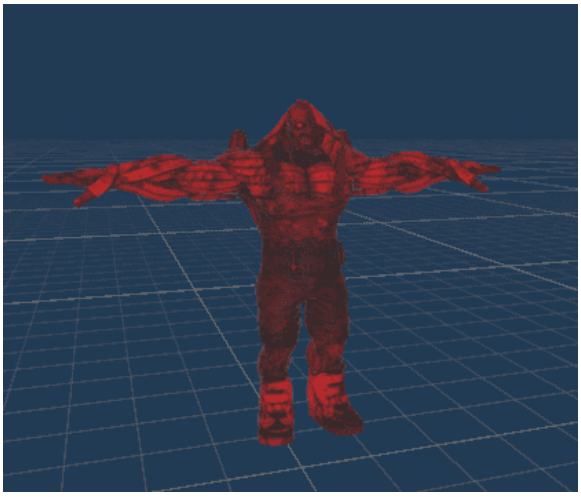
a. *Melee Boss*

Melee Boss akan menyerang musuh dari jarak dekat dan sedang. Pada mode normal, *Melee Boss* memiliki dua serangan yaitu jump *attack* untuk jarak sedang dan combo *attack* untuk jarak dekat. Dan ketika mode *rage Melee Boss* akan memilki tambahan speed ketika melakukan penyerangan dan berjalan. Bentuk serangannya juga menjadi berbeda, yaitu swing *attack* untuk jarak sedang dan *attack* combo untuk serangan jarak dekat.



Gambar 3. 7 *Melee Boss* Mode Normal

Gambar 3. 7 merupakan tampilan *melee boss* dalam mode normal.

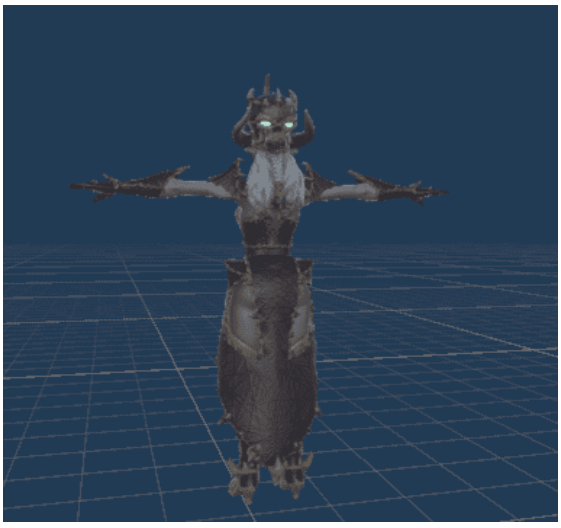


Gambar 3. 8 *Melee Boss* Mode *Rage*

Gambar 3. 8 merupakan tampilan *melee boss* dalam mode *rage*.

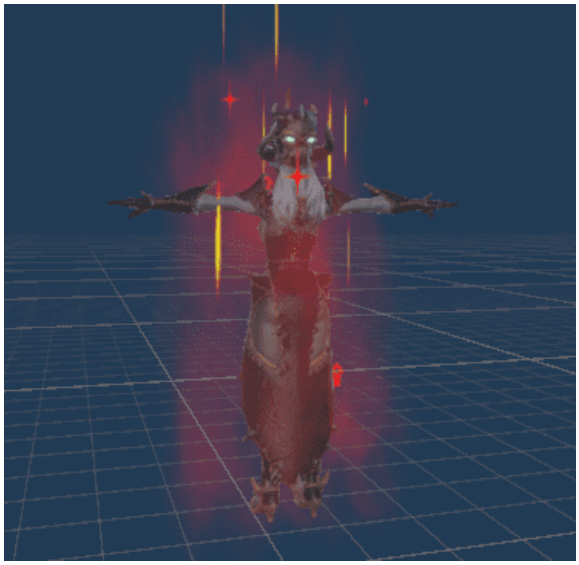
b. *Range Boss*

Range Boss akan menyerang musuh dari jarak jauh dan sedang. Pada mode normal, *Melee Boss* memiliki dua serangan yaitu fire ball untuk jarak sedang dan *exp*losive untuk jarak jauh. Dan ketika mode *rage* *Range Boss* memiliki serangan yang berbeda, yaitu fire ball untuk jarak sedang dan big shoot untuk serangan jarak jauh.



Gambar 3. 9 *Range Boss* Mode Normal

Gambar 3. 9 merupakan tampilan *range boss* dalam mode normal.



Gambar 3. 10 *Range Boss* Mode Normal

Gambar 3. 10 merupakan tampilan *range boss* dalam mode normal.

3.3.2 Kalkulasi dynamic difficulty adjustment

Difficulty atau kesulitan akan diatur bedasarkan *upgrade* status pemain. Kesulitan ini akan mempengaruhi status musuh.

1. *Upgrade* Status (*Attack* dan *Health*)

   Ketika pemain meningkatkan status maka sistem akan secara otomatis menyesuaikan status musuh. Jika pemain meningkatkan serangannya maka sistem juga akan meningkatkan serangan musuh, dan jika pemain meningkatkan *health* atau darahnya maka sistem akan meningkatkan *health* musuh. Untuk memenuhi kebutuhan tersebut maka diperlukan perhitungan yang sesuai, berikut merupakan perhitungan penyesuaian status:

   a. Peningkatan *Attack*

      Untuk menyesuaikan peningkatan *attack* musuh, yang pertama kali dilakukan adalah mencari faktor peningkatan *attack* musuh yang didapat dengan mencari selisih dari total nilai serangan yang pemain miliki saat ini dan nilai serangan dasar pemain yaitu nilai serangan yang dimiliki pemain ketika permainan dimulai, kemudian selisih tersebut dibagi dengan nilai serangan dasar pemain.

Langkah berikutnya adalah menentukan total nilai serangan musuh. Total nilai serangan musuh ini dapat dihitung dengan mengambil nilai serangan dasar pemain dan mengalikannya dengan hasil penjumlahan dari 1 ditambah setengah dari faktor peningkatan *attack* musuh.

Penyesuaian peningkatan *attack* musuh dapat dilihat lebih jelas dengan rumus pada persamaan 1 dan persamaan 2:

$$FPSM = \frac{TSP - SDP}{SDP} \quad \text{.....(1)}$$

$$Attack\ Musuh = SDP\ (1 + 0{,}5\ \times FPSM) \quad \text{.....(2)}$$

Keterangan:

FPSM = Faktor peningkatan serangan musuh

TSP = Total serangan pemain yang merupakan total serangan pemain setelah melakukan peningkatan

SDP = Serangan dasar pemain yang merupakan total serangan pemain pertama kali ketika permainan dimulai

Attack Musuh = Total serangan yang dimiliki musuh

b. Peningkatan *Health*

Untuk menyesuaikan peningkatan *health* musuh menggunakan cara yang sama dengan menyesuaikan peningkatan *attack* musuh, hal yang pertama kali dilakukan adalah mencari faktor peningkatan *health* musuh yang didapat dengan mencari selisih dari total nilai *max health* (nilai maksimum *health*) yang pemain miliki saat ini dan nilai *max health* dasar pemain yaitu nilai *max health* yang dimiliki pemain ketika permainan dimulai, kemudian selisih tersebut dibagi dengan nilai *max health* pemain.

Langkah berikutnya adalah menentukan total nilai serangan musuh. Total nilai *health* musuh ini dapat dihitung dengan mengambil nilai *max health* pemain dan mengalikannya dengan hasil penjumlahan dari 1 ditambah setengah dari faktor peningkatan *health* musuh

Penyesuaian peningkatan *health* musuh dapat dilihat lebih jelas dengan rumus persamaan 3 dan persamaan 4:

$$FPHM = \frac{THP - HDP}{HDP} \quad \text{.....(3)}$$

$$Health\ Musuh = HDP\ (1 + 0{,}5\ \times FPHM) \quad \text{.....(4)}$$

Keterangan:

FPHM = Faktor peningkatan *health* musuh

THP = Total *health* pemain yang merupakan total *max health* pemain setelah melakukan peningkatan

HDP = *Health* dasar pemain yang merupakan total *max health* pemain pertama kali ketika permainan dimulai

Health Musuh = Total *health* yang dimiliki oleh musuh

3.3.3 Pembobotan jenis musuh

Pembobotan jenis musuh berguna untuk menentukan persentase kemunculan musuh normal yang ada di dalam *game*. Semakin besar persentasenya maka akan semakin sering pula musuh tersebut akan muncul di dalam *game*. Pembobotan musuh dibuat berdasarkan jenis musuh normal dan pengaruh bentuk serangannya pada jalannya permainan. Bentuk serangan dari masing-masing musuh normal dapat dilihat pada Tabel 3. 2. Musuh dengan pengaruh serangan yang paling menyulitkan pemain akan memiliki bobot paling rendah. Dengan demikian berdasarkan jenis-jenis musuh dan bentuk serangannya dibuatlah pembobotannya seperti pada Tabel 3. 3.

Tabel 3. 3 Tabel Nilai Pembobotan Musuh

Jenis Musuh	Nilai Pembobotan Musuh
Melee	35 %
Chaser	25%
Range	20%
Charger	20%

Namun ketika permainan dimulai sistem tidak akan langsung memunculkan semua jenis musuh tersebut. Pada awal permainan sistem akan memunculkan atau mengaktifkan 2 jenis musuh, yaitu *Melee* dan *Chaser*. Ketika pemain berhasil mencapai 6 *wave*, maka jenis musuh ketiga ikut diaktifkan, yaitu *Range*. Dan ketika pemain berhasil berhasil mencapai 11 *wave*, maka jenis musuh keempat ikut diaktifkan, yaitu *Charger*. Dikarenakan musuh tidak muncul bersamaan tetapi *sequential*, maka pembobotannya akan berubah sampai semua musuh muncul. Untuk mengatur pembobotan jenis musuh tersebut dapat dihitung dengan menggunakan rumus:

$$BPM\ (x) = \frac{NPM\ (x)}{TBPM}$$

Keterangan:

x = Jenis musuh di dalam game

BPM = Bobot persen musuh x yang akan dimunculkan dalam *wave* yang dijalani oleh pemain

NPM = Nilai pembobotan musuh yang telah ditetapkan pada Tabel 3. 3

TBPM = Total bobot persen musuh yang sedang aktif pada *wave* yang sedang dijalani pemain

Rumus tersebut digunakan untuk menentukan bobot persen musuh yang akan muncul dalam *wave* yang sedang dihadapi oleh pemain. Dalam rumus tersebut, x merupakan masing-masing jenis musuh yang sedang aktif di dalam *wave*. BPM atau bobot persen musuh x mengindikasikan persentase kemunculan musuh jenis x dalam *wave* tersebut. NPM atau nilai pembobotan musuh adalah nilai bobot yang telah ditetapkan untuk masing-masing jenis musuh sesuai dengan Tabel 3.3. Terakhir, TBPM atau total bobot persen musuh adalah total nilai pembobotan dari semua jenis musuh yang aktif pada gelombang yang sedang dijalani pemain. Dengan menggunakan rumus ini, persentase kemunculan setiap jenis musuh dapat dihitung secara proporsional berdasarkan bobot yang telah ditentukan, sehingga dapat menyeimbangkan kesulitan dan variasi musuh yang dihadapi oleh pemain dalam setiap *wave*.

3.3.4 Siklus wave yang tidak terbatas

Siklus *wave* yang tidak terbatas akan dimulai dari *wave* 1 hingga *wave* seterusnya. Untuk berpindah dari satu *wave* ke *wave* selanjutnya pemain harus mengalahkan semua musuh yang muncul pada *wave* tersebut. Untuk setiap perpindahan *wave* terdapat peningkatan jumlah musuh yang muncul sebanyak 5 musuh. Jumlah musuh yang muncul dibatasi berjumlah maksimal 50 musuh. Setiap pemain berhasil menyelesaikan 15 wave akan mereset kembali jumlah musuh yang muncul kembali menjadi seperti di awal permainan, dan musuh akan mendapatkan peningkatan *attack power* dan *health point*.

Dengan demikian siklus *wave* yang tidak terbatas tersebut dapat dirumuskan menjadi:

$$JumlahMusuh = JMD + ((NomorWave\ mod\ 15) - 1) * 5$$

Keterangan:

JumlahMusuh	=	Jumlah musuh yang akan dimunculkan dalam satu gelombang musuh.
JMD	=	Jumlah musuh dasar yang dimunculkan di awal permainan
NomorWave	=	*Wave* yang sedang dijalani pemain
Angka 5	=	Nilai peningkatan jumlah musuh pada setiap *wave* nya

Rumus tersebut digunakan untuk menghitung jumlah musuh yang akan dimunculkan pada tiap *wave*. Jumlah musuh yang muncul akan meningkat sebanyak 5 musuh untuk setiap *wave*, dengan mekanisme reset setiap 15 *wave*.

3.4 Rencana Pengujian

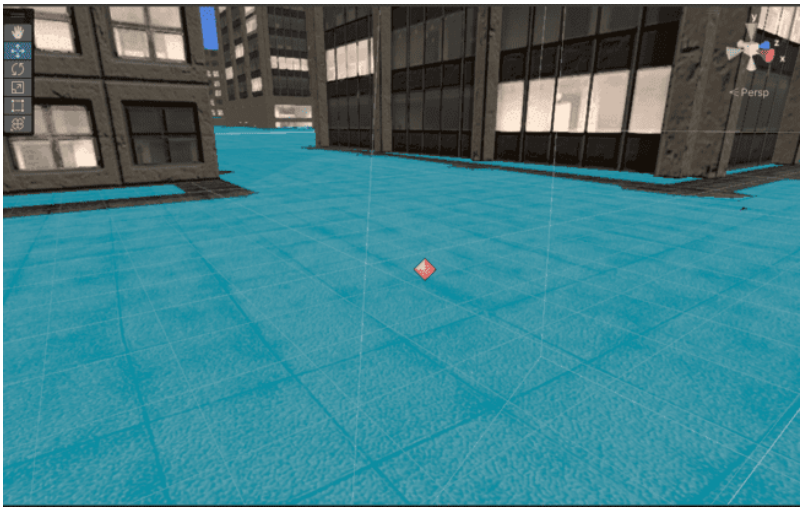
Untuk memastikan sistem *infinite wave* berjalan sesuai dengan perancangan dan tidak ada *bug* atau masalah yang mengganggu pengalaman bermain, rencana pengujian sistem *infinite wave* akan dilakukan secara langsung saat bermain *game*. Pengujian ini akan menilai kemampuan sistem untuk mengeluarkan berbagai jenis musuh dan meningkatkan kesulitan permainan secara dinamis.

BAB 4

IMPLEMENTASI DAN PENGUJIAN

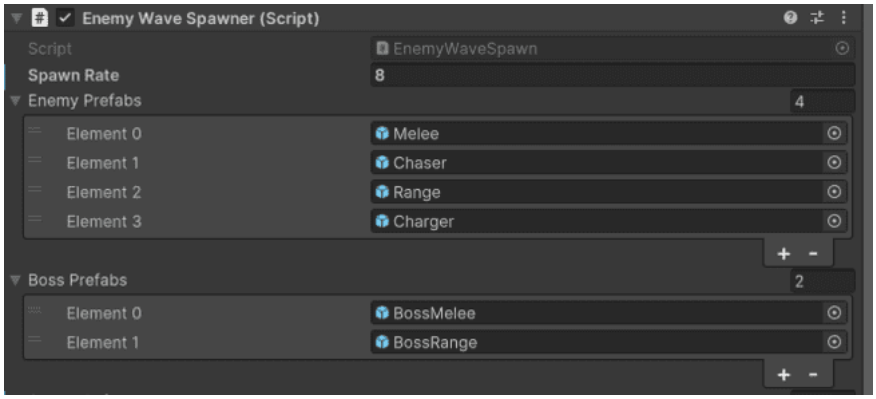
4.1 Implementasi Sistem *Infinite Wave*

Setelah tahap analisis dan perancangan, tahap selanjutnya adalah mengimplementasikan sistem *infinite wave* ke dalam *game* "*From The Downtown*". Untuk mengimplementasikan sistem *infinite wave*, harus dibuat beberapa objek kosong sebagai *spawn point* gelombang musuh. *Spawn point* tersebut akan disebar di beberapa titik yang berjauhan dari *spawn* pemain.



Gambar 4. 1 *Spawn Point Enemy*

Gambar 4. 1 menunjukkan salah satu *spawn point enemy* yang ada pada map dan diletakkan berjauhan dari *spawn* pemain. Kemudian dari *spawn point* tersebut akan diambil jenis-jenis musuh yang ada pada *game* mulai dari musuh normal hingga musuh *boss*.



Gambar 4. 2 *Inspector* Jenis-Jenis Musuh

Gambar 4. 2 menjelaskan jenis-jenis musuh yang dimasukkan ke dalam game mulai dari musuh normal hingga musuh *boss* melalui *inspector* objek *spawn point*. Selanjutnya dimasukkan juga ke dalam *spawn point* tersebut perhitungan nilai pembobotan musuh yang akan dimunculkan melalui kode yang ditampilkan pada Gambar 4. 3.

```
private void UpdateSpawnWeights()
{
    float[] baseWeights = { 35f, 25f, 20f, 20f };
    int activeCount = (waveNumber > 20) ? 4 : (waveNumber > 10) ? 3 : 2;

    float totalActiveWeight = 0f;
    for (int i = 0; i < activeCount; i++)
    {
        totalActiveWeight += baseWeights[i];
    }

    for (int i = 0; i < spawnWeights.Length; i++)
    {
        spawnWeights[i] = (i < activeCount) ? (baseWeights[i] * 100f / totalActiveWeight) : 0f;
    }
}
```

Gambar 4. 3 Kode Perhitungan Nilai Pembobotan Musuh

Kemudian untuk mengatur *spawn* musuh *boss*, sistem akan mendeteksi jumlah *wave* yang telah dilewati oleh pemain. Musuh *boss* akan dimunculkan setiap kali pemain melewati 5 *wave* dan musuh *boss* yang dimunculkan akan selalu bergantian antara *melee boss* dan *range boss*. Pengimplementasian di dalam kode bisa dilihat melalui Gambar 4. 4.

```
private void SpawnBoss()
{
    int spawnPointIndex = Random.Range(0, spawnPoints.Length);
    currentBoss = Instantiate(bossPrefabs[bossIndex % bossPrefabs.Length], spawnPoints[spawnPointIndex].position, Quaternion.identity);
    bossIndex++;
    bossIsActive = true;

    if (attackPowerEnemyBoss != null)
    {
        attackPowerEnemyBoss.IsDead = false;
    }

    Health bossHealth = currentBoss.GetComponent<Health>();
    if (bossHealth != null)
    {
        bossHealth.OnDeath += BossDefeated;
    }
}
```

Gambar 4. 4 Kode *Spawn* Musuh *Boss*

Selanjutnya untuk peningkatan kemampuan musuh *boss*, setiap kali musuh *boss* dikalahkan maka untuk musuh *boss* selanjutnya akan mendapatkan *health* tambahan sebanyak 100 *health* dan *attack* tambahan sebanyak 20 *attack power*.

```
1 reference
void UpdateBossHealth(float healthChange)
{
    foreach (Health health in enemyBossHealthList)
    {
        health.InitialHealth += healthChange;
        health.MaximumHealth += healthChange;
    }
}
```

Gambar 4. 5 Kode *Upgrade Health* Musuh *Boss*

Gambar 4. 5 merupakan kode untuk meningkatkan atau *upgrade health* musuh *boss*. Nilai dari *healthChange* adalah 100.

```
1 reference
void UpdateMeleeWeaponDamage(float powerChange)
{
    foreach (MeleeWeapon weapon in meleeWeaponDamages)
    {
        weapon.MinDamageCaused += powerChange;
        weapon.MaxDamageCaused += powerChange;
    }
}

1 reference
void UpdateProjectileDamage(float powerChange)
{
    foreach (DamageOnTouch projectile in projectileDamages)
    {
        projectile.MinDamageCaused += powerChange;
        projectile.MaxDamageCaused += powerChange;
    }
}
```

Gambar 4. 6 Kode *Upgrade Attack* Musuh *Boss*

Gambar 4. 6 merupakan kode untuk *upgrade attack* musuh *boss.* Nilai dari *powerChange* adalah 20 *attack power* dan akan langsung ditambahkan dengan jumlah *damage* pada senjata musuh *boss*. *Melee weapon* merupakan senjata atau serangan yang dimiliki oleh *melee boss*. *Projectile* merupakan serangan atau senjata yang dimilki oleh *range boss*.

4.2 Implementasi *Upgrade Health* dan *Attack*

Upgrade health dan *attack* merupakan sistem yang mengatur peningkatan status dari pemain. Peningkatan status ini dilakukan dalam panel *upgrade* yang dibuat di dalam *game*. Panel *upgrade* akan muncul setiap kali pemain berhasil melewati 1 gelombang musuh. Pada panel *upgrade* tersebut akan menyediakan tampilan yang akan mengatur *upgrade attack* dan *upgrade health* seperti yang ditunjukkan pada Gambar 4. 7.

Gambar 4. 7 Panel *Upgrade* Status

Kemudian pada panel tersebut akan dimasukkan logika yang akan menyesuaikan peningkatan status pemain dengan peningkatan status musuh, yaitu penyesuaian pada *attack* dan *health*. Hal ini diimplementasikan pada kode yang ditampilkan pada Gambar 4. 8.

```
2 references
private void AdjustEnemyStats()
{
    GameObject player = GameObject.FindGameObjectWithTag("Player");
    if (player != null)
    {
        Health playerHealth = player.GetComponent<Health>();
        AttackPower playerAttack = player.GetComponent<AttackPower>();

        // Menghitung faktor peningkatan kesehatan musuh berdasarkan kesehatan pemain
        float healthIncreaseFactor = (playerHealth.MaximumHealth - basePlayerHealth) / basePlayerHealth;
        float attackIncreaseFactor = (playerAttack.attackPower - basePlayerAttackPower) / basePlayerAttackPower;

        foreach (Health enemyHealth in enemyHealthList)
        {
            // Menyesuaikan kesehatan musuh berdasarkan faktor peningkatan kesehatan pemain
            enemyHealth.InitialHealth = baseEnemyHealth * (1 + 0.5f * healthIncreaseFactor);
            enemyHealth.MaximumHealth = enemyHealth.InitialHealth;
            Debug.Log("Enemy health adjusted to: " + enemyHealth.InitialHealth);
        }

        // Menyesuaikan kekuatan serangan musuh berdasarkan kekuatan serangan pemain
        if (enemyAttackPower != null)
        {
            enemyAttackPower.attackPower = (int)(baseEnemyAttackPower * (1 + 0.5f * attackIncreaseFactor));
            Debug.Log("Enemy attack power adjusted to: " + enemyAttackPower.attackPower);
        }
    }
}
```

Gambar 4. 8 Kode *Upgrade* Status

Gambar 4. 8 menjelaskan kode untuk penyesuaian kesulitan musuh dengan perubahan peningkatan status yang dilakukan pemain.

4.3 Hasil Pengujian

4.3.1 Pengujian sistem infinite wave

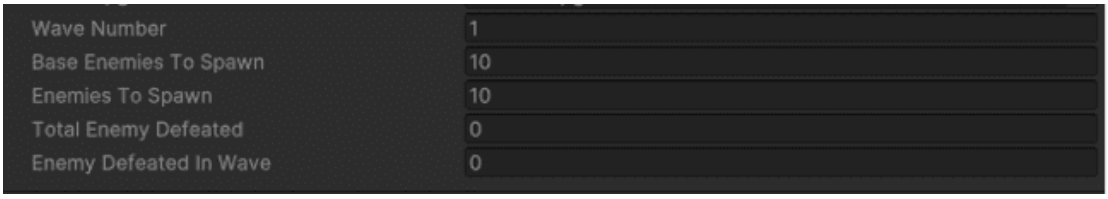
Pengujian sistem *infinite wave* diuji secara langsung saat *game* dimainkan. Dan berikut adalah hasil dari pengujian yang dilakukan. *Game From The Downtown* akan diawali dengan tampilan main menu yang bisa dilihat pada Gambar 4. 9.

Gambar 4. 9 Tampilan *Main Menu*

Ketika dimulai, *game* akan langsung menjalankan sistem *infinite wave*. Pemain akan memulai *wave* atau gelombang musuh pertama dan pemain akan diletakkan di posisi yang jauh dengan *spawn point* musuh seperti yang ditampilkan pada Gambar 4. 10.

Gambar 4. 10 *Spawn* Pemain

Pemain harus mengalahkan jumlah musuh yang ditentukan dalam satu *wave* yang bisa dilihat pada Gambar 4. 11.

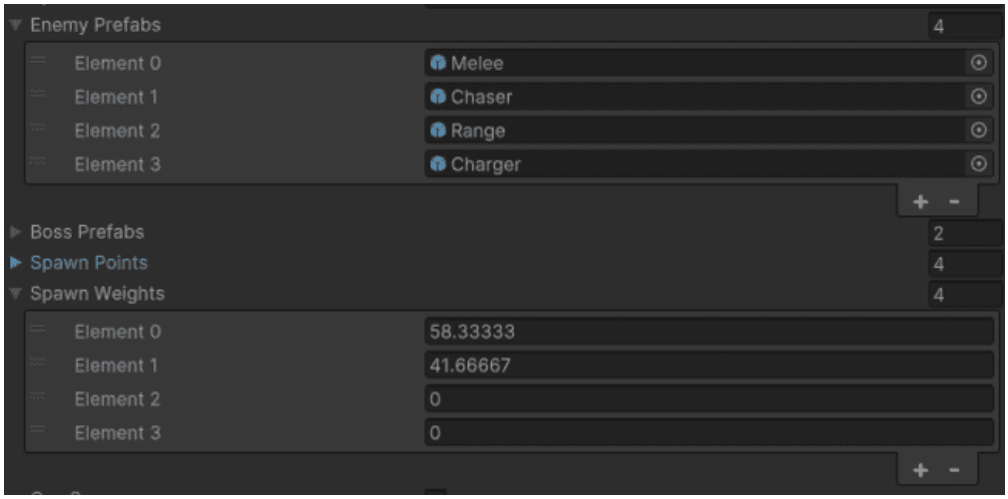


Gambar 4. 11 Jumlah Musuh Dalam Satu Gelombang

Pada Gambar 4. 11 terdapat parameter ”*Wave Number*” yang menunjukkan *wave* yang sedang dijalani oleh pemain. Parameter ”*Base Enemy To Spawn*” menunjukkan jumlah musuh yang muncul pada awal *game* dimulai. Parameter ”*Enemies To Spawn*” menjelaskan jumlah musuh yang dimunculkan pada *wave* yang sedang dijalani oleh pemain. Parameter ”*Total Enemy Defeated”* menjelaskan tentang total kesuluruhan musuh yang sudah dikalahkan oleh pemain selama permainan berlangsung. Parameter ”*Enemy Defeated In Wave*” berisi

jumlah musuh yang sudah dikalahkan oleh pemain di dalam *wave* yang sedang dijalankan.

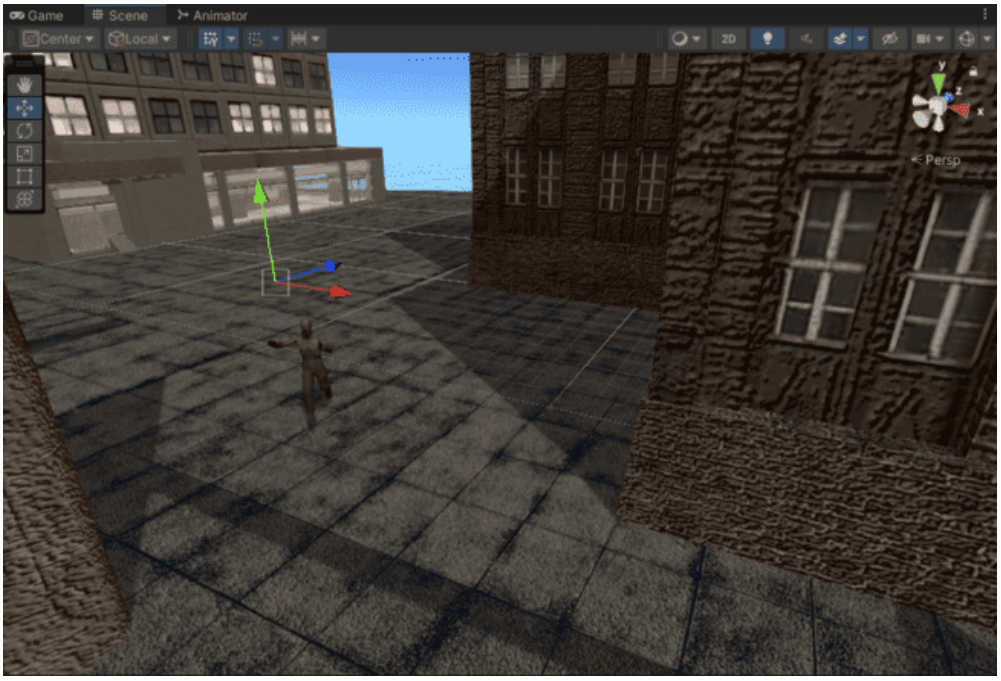
Di awal *game* musuh yang muncul hanya 2 jenis musuh yaitu *Melee* dan *Chaser* dengan nilai pembobotan musuh yang telah dihitung oleh sistem berdasarkan rumus perhitungan nilai pembobotan jenis musuh. Musuh yang muncul serta hasil perhitungan nilai pembobotan jenis musuh bisa dilihat pada Gambar 4. 12.



Gambar 4. 12 Pembobotan Nilai Musuh *Melee* dan *Chaser* di Awal *Game*

Pada Gambar 4. 12 terdapat list dengan nama *Enemy Prefabs* berisi tentang jenis-jenis musuh normal yang akan dimunculkan di dalam permainan. Jenis-jeins musuh tersebut diurutkan mulai dari *Element* 0 dengan jenis musuh *Melee* hingga yang terakhir *Element* 3 dengan jenis musuh *Charger*. List dengan nama *Spawn Weights* berisi nilai pembobotan musuh yang akan dimunculkan dalam satu *wave*. Pembobotan musuh tersebut akan menunjukkan persentase kemunculan musuh dalam satu *wave*. Ketika permainan dimulai musuh *melee* memiliki bobot 58,3 % sementara musuh *chaser* memiliki bobot 41,7 %.

Musuh *melee* dan *chaser* muncul dari *spawn point* yang telah diatur di beberapa titik pada peta permainan. Musuh yang muncul dari *spawn point* tersebut dapat dilihat dari Gambar 4. 13.



Gambar 4. 13 Musuh *Melee* dan *Chaser* Muncul dari Spawn Point

Pemain bisa mengambil senjata pada *item picker* yang terletak di *ground* pada *game*, dapat dilihat dari gambar Gambar 4. 14 dan menggunakannya untuk mengalahkan musuh yang muncul.

Gambar 4. 14 *Item Picker* Senjata

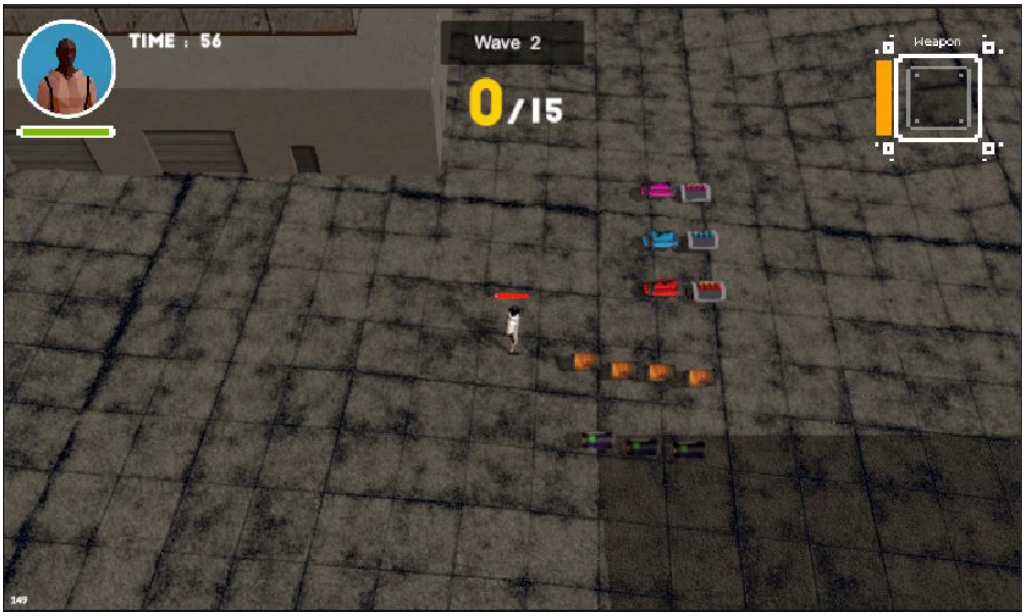
Setiap kali pemain berhasil mengalahkan musuh maka pemain akan mendapatkan *exp* dari jenis musuh yang dikalahkan. Dengan *exp* tersebut pemain bisa melakukan *upgrade* status di panel *upgrade*. Panel *upgrade* akan terbuka

setiap pemain berhasil mengalahkan semua musuh yang ada dalam satu gelombang. Dan hal ini dapat dilihat pada Gambar 4. 15.

Gambar 4. 15 Panel *Upgrade* Status Pemain

Gambar 4. 15 menunjukkan panel tempat pemain dapat melakukan *upgrade* status. Pada panel tersebut memperlihatkan jumlah *exp* yang diperoleh oleh pemain serta *upgrade cost* atau biaya yang digunakan untuk melakukan *upgrade* status. *Game* akan dilanjutkan setelah pemain selesai melakukan *upgrade* dan menekan tombol “Selesai”.

Setelah pemain berhasil melewati gelombang musuh atau *wave* dan selesai melakukan peningkatan, maka pada *wave* selanjutnya akan menambah jumlah musuh yang akan muncul sebanyak 5 musuh. Jumlah musuh yang bertambah tersebut bisa dilihat dari Gambar 4. 16.

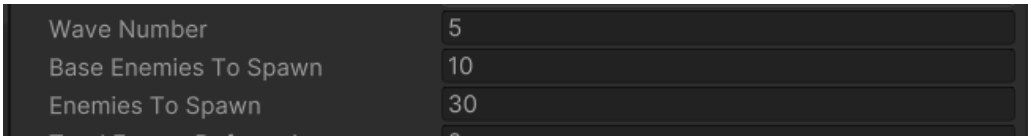


Gambar 4. 16 Peningkatan Jumlah Musuh

Selanjutnya setiap kali pemain berhasil melewati 5 gelombang musuh, maka musuh *boss* akan muncul secara bergantian. Musuh *boss* yang pertama muncul adalah *Melee Boss*, yang terlihat melalui Gambar 4. 17.

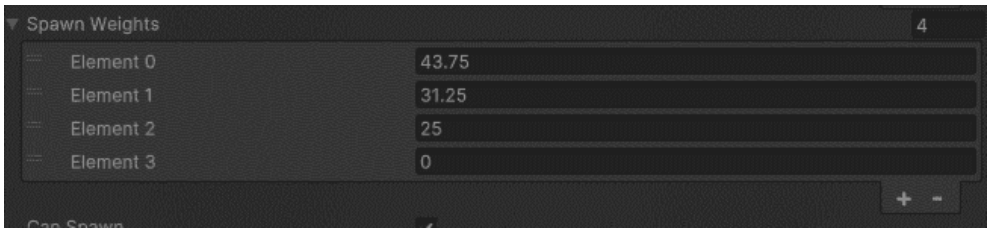
Gambar 4. 17 *Melee Boss* Muncul dan Menyerang Pemain

Melee boss pada Gambar muncul ketika *wave number* bernilai 5 yang berarti pemain berhasil melewati 5 gelombang, hal ini dapat dilihat pada Gambar 4. 18. Pada Gambar 4. 18 juga dapat dilihat musuh yang akan dimunculkan pada *wave* 5 meningkat menjadi 30 musuh.



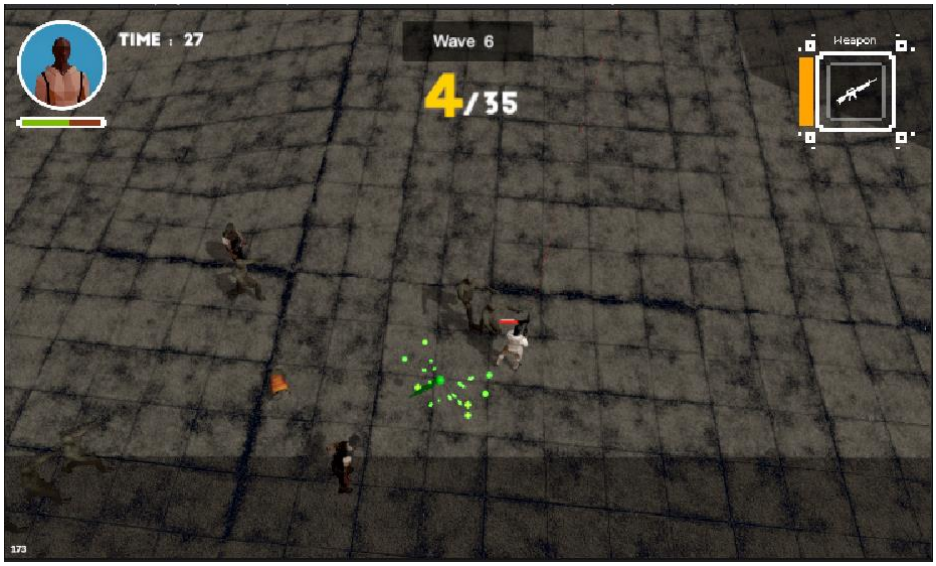
Gambar 4. 18 *Wave Number*

Jika pemain berhasil mengalahkan musuh *boss* maka pada *wave* ke 6 akan memunculkan jenis musuh yang ketiga yaitu *range*. Musuh *range* muncul ketika pemain berhasil menyelesaikan atau menghalahkan musuh di *wave* yang ke 5. Karena musuh ketiga sudah muncul maka akan dilakukan kembali perhitungan nilai pembobotan musuh yang akan muncul di dalam *game*. Pembobotan tersebut dapat dilihat pada Gambar 4. 19.



Gambar 4. 19 Nilai Pembobotan Musuh diperbarui

Gambar 4.19 menjelaskan nilai pembobotan musuh yang diperbarui, sehingga musuh *melee* memiliki bobot 43,75 %, musuh *chaser* memiliki bobot 31,25 % dan musuh ketiga yaitu *range* memiliki bobot kemunculan senilai 25 %. Selanjutnya musuh *range* dimunculkan di dalam *wave*, musuh *range* dapat aktif dan menyerang pemain seperti yang ditunjukkan Gambar 4. 20.

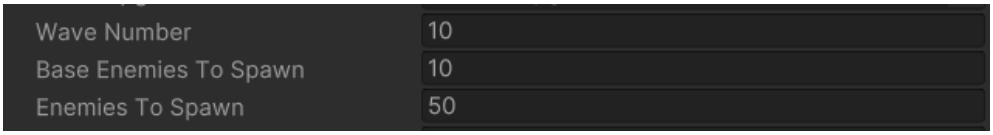


Gambar 4. 20 Musuh *Range* Menyerang Pemain

Ketika pemain berhasil melewati tambahan 5 *wave* selanjutnya maka akan memunculkan boss selanjutnya yaitu *range boss*. *Range boss* muncul ke dalam *game* dan dapat menyerang pemain seperti yang terlihat pada Gambar 4. 21.

Gambar 4. 21 *Range Boss* Muncul dan Menyerang Pemain

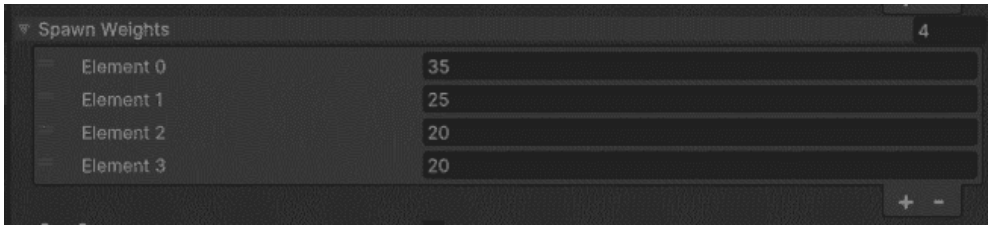
Range boss pada Gambar 4. 21 muncul ketika *wave number* bernilai 10 yang berarti pemain berhasil melewati 10 gelombang, hal ini dapat dilihat pada Gambar 4. 22.



Gambar 4. 22 *Wave Number*

Gambar 4. 22 juga menjelaskan total musuh yang dimunculkan di dalam *wave* 10 berjumlah 50. Hal ini juga membuktikan bahwa musuh yang dimunculkan berjumlah maksimal 50 musuh dan tidak bisa lebih dari 50.

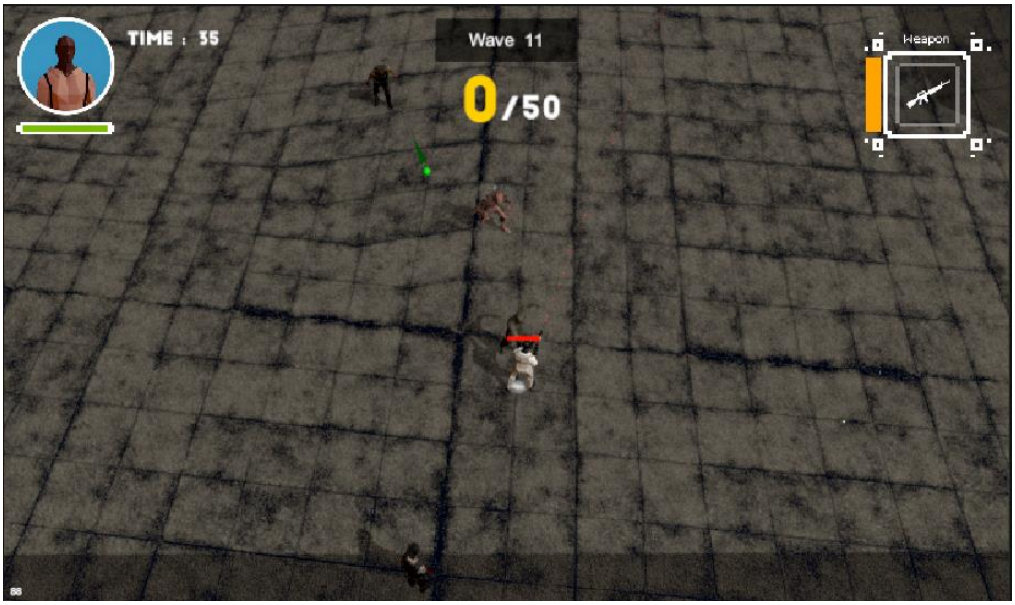
Selanjutnya ketika musuh *boss* kembali berhasil dikalahkan, maka pada *wave* 11 akan memunculkan jenis musuh yang keempat yaitu *range*. Musuh *range* muncul ketika pemain berhasil menyelesaikan atau menghalahkan musuh di *wave* 10. Dikarenakan musuh terakhir sudah muncul yaitu *charger*, maka nilai pembobotan jenis musuh akan diperbarui seperti yang terlihat pada Gambar 4. 23.



Gambar 4. 23 Nilai Pembobotan Musuh Diperbarui

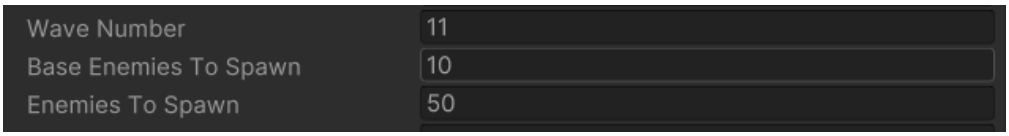
Gambar 4. 23 menjelaskan nilai pembobotan musuh yang baru ketika semua musuh sudah muncul. Musuh *melee* memiliki bobot senilai 35%, musuh *chaser* memiliki bobot 25%, musuh *range* memiliki bobot 20% dan musuh *charger* memiliki bobot 20%. Dengan demikian juga membuktikan bahwa nilai pembobotan musuh ketika semua musuh muncul sesuai dengan nilai pembobotan musuh yang telah dirancang sebelumnya.

Selanjutnya musuh normal terakhir yaitu *charger* muncul ke dalam *game* dan dapat menyerang pemain seperti yang terlihat pada Gambar 4. 24.



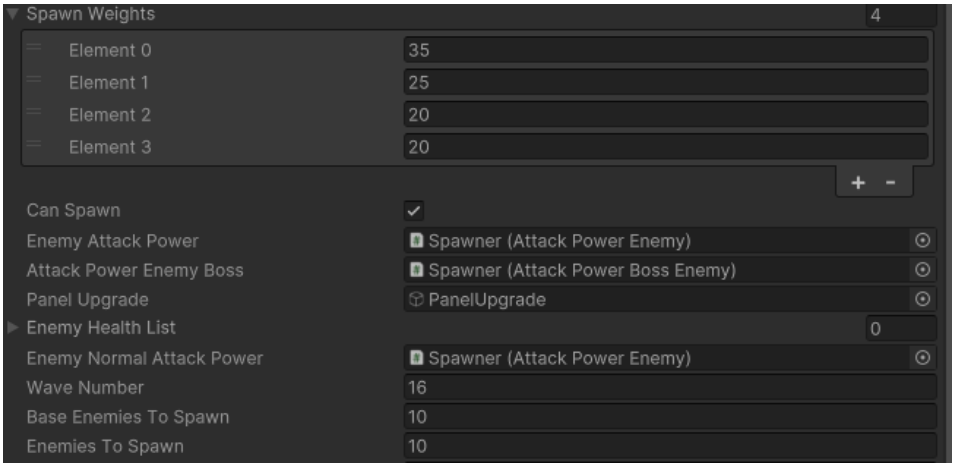
Gambar 4. 24 *Charger* Muncul dan Menyerang Pemain

Musuh *Charger* pada Gambar 4. 24 muncul ketika *wave number* bernilai 11, hal ini dapat dilihat pada Gambar 4. 25.



Gambar 4. 25 *Wave Number*

Selanjutnya ketika pemain berhasil menyelesaikan *wave* 15 maka pada *wave* 16 jumlah musuh yang dikeluarkan akan direset kembali jumlahnya lalu akan dimulai kembali dari jumlah 10 musuh, dan musuh akan tetap bertambah pada *wave* selanjutnya. Ketika jumlah musuh yang muncul tersebut direset, maka musuh akan mendapatkan status tambahan yaitu berupa peningkatan *attack* dan *health*. Hal ini berlaku setiap kali pemain berhasil melewati 15 *wave*. Untuk lebih jelas dapat dilihat pada Gambar 4. 26.



Gambar 4. 26 Jumlah Musuh pada *Wave* 16

Pada Gambar 4. 26 menjelaskan pada *wave number* 16 maka jumlah musuh yang dimunculkan adalah berjumlah 10, yang dapat dilihat pada parameter ”*Enemies To Spawn*”. Namun semua jenis musuh normal akan terus lanjut dimunculkan karena sudah diaktifkan pada *wave* sebelumnya dan nilai pembobotan jenis-jenis musuh tersebut tetap sesuai seperti yang ditentukan.

Begitu seterusnya dengan siklus yang tidak terbatas. Sistem akan terus mengatur gelombang musuh yang tidak terbatas dan akan menyesuaikan peningkatan status pemain dengan status musuh hingga pemain bisa dikalahkan atau permainan berhenti.

4.3.2 Pengujian dynamic difficulty adjustment

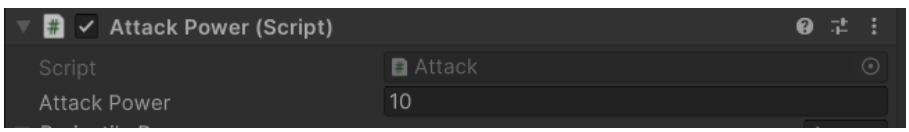
Kesulitan dinamis di dalam *game* ”*From The Downtown*” dipengaruhi oleh peningkatan status yang dilakukan oleh pemain. Peningkatan status pada pemain berupa peningkatan *attack* dan *max health*. *Attack* merupakan nilai kualitas serangan yang dikeluarkan pemain untuk mengalahkan musuh, sementara *max health* merupakan total maksimal *health* yang dimiliki pemain sehingga jika semakin banyak *health* yang dimiliki pemain akan semakin lama pula pemain dapat bertahan dalam permainan. Ketika pemain meingkatkan statusnya maka musuh juga akan menyesuaikan peningkatan status yang dilakukan oleh pemain, jika pemain melakukan peningkatan pada *attack* maka musuh juga akan menyesuaikan *attack* nya dan jika pemain melakukan peningkatan pada *max health* maka musuh juga akan menyesuaikan *max health* miliknya.

1. Peningkatan *Attack*

   Kesulitan dinamis pada peningkatan *attack* dimulai ketika pemain melakukan peningkatan status berupa *attack*. Peningkatan tersebut dilakukan pada panel *upgrade* yang terbuka pada saat pemain berhasil mengalahkan semua musuh pada setiap *wave* yang sedang dijalani oleh pemain. Panel *upgrade* tersebut dapat dilihat pada Gambar 4. 27.

Gambar 4. 27 Tampilan Panel *Upgrade*

Gambar 4. 27 menunjukkan tampilan dari panel *upgrade*. Pada panel upgrade tersebut menampilkan dua atribut yang dapat ditingkatkan oleh pemain yaitu *attack* dan *health*. Pada awal permainan pemain memiliki *attack* dengan nilai 10. Hal ini juga dapat dilihat pada parameter ”*Attack Power*” pemain yang ditunjukkan pada Gambar 4. 28.



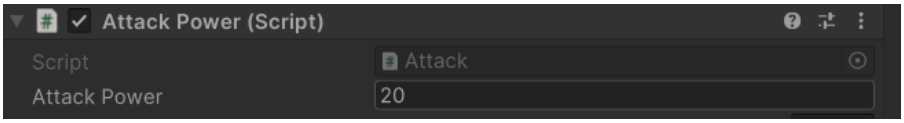
Gambar 4. 28 Jumlah *Attack* Pemain

Gambar 4. 28 menunjukkan parameter ”*Attack Power*” yang merupakan nilai *attack* yang dimiliki pemain. Ketika permainan dimulai milai *attack* tersebut berjumlah 10. Selanjutnya pemain dapat melakukan peningkatan *attack* selama pemain memiliki *exp* yang cukup untuk melakukan peningkatan. *Exp* tersebut diperoleh dari membunuh musuh yang ada dalam

permainan. Pada panel *upgrade* pemain dapat melihat jumlah *exp* yang dimilikinya pada atribut atau informasi tambahan dengan tulisan ”*Player exp*” yang ada pada bagian kanan bawah panel *upgrade* tersebut. Hal ini dapat dilihat lebih jelas pada Gambar 4. 29.

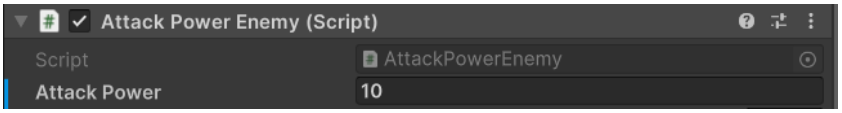
Gambar 4. 29 Peningkatan *Attack* Pemain

Gambar 4. 29 memperlihatkan keadaan panel *upgrade* ketika pemain selesai melakukan peningkatan *attack*. Dalam panel *upgrade* tersebut dapat dilihat bahwa nilai *attack* yang dimiliki pemain setelah melakukan peningkatan *attack* adalah sejumlah 20, yang berarti terdapat penambahan 10 *attack* dari sebelumnya di awal permainan. Untuk membuktikan peningkatan tersebut berjalan dengan baik dapat dilihat pada Gambar 4. 30.



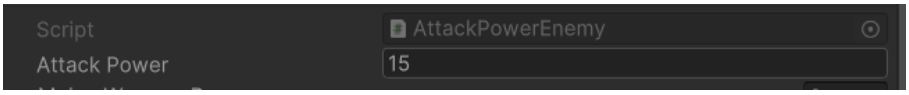
Gambar 4. 30 Jumlah *Attack* Pemain Setelah Peningkatan

Gambar 4. 30 memperlihatkan perubahan nilai *attack* pemain pada parmeter "*Attack Power*” yang sebelumnya berjumlah 10 sekarang bertambah menjadi 20. Selanjutnya dengan adanya peningkatan tersebut akan mempengaruhi nilai *attack* yang dimiliki musuh. Pada awalnya musuh juga memiliki jumlah *attack* yang sama dengan pemain ketika permainan dimulai yaitu berjumlah 10. Hal ini dapat dilihat pada Gambar 4. 31.



Gambar 4. 31 *Attack* Musuh pada Awal Permainan

Gambar 4. 31 menunjukkan parameter ”*Attack Power*” pada komponen *Attack Power Enemy* yang merupakan nilai *attack* yang dimiliki oleh musuh. Pada awal permainan nilai *attack* yang dimiliki oleh musuh adalah 10. Dan ketika pemain melakukan peningkatan *attack* pemain maka musuh akan menyesuaikan nilai *attack* yang dimilikinya menyesuaikan dengan peningkatan yang dilakukan oleh pemain. Hal ini lebih jelasnya dapat dilihat pada Gambar 4. 32.



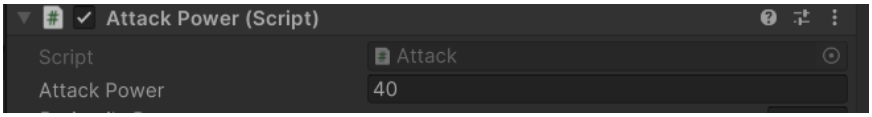
Gambar 4. 32 Peningkatan *Attack* Musuh

Gambar 4. 32 menunjukkan nilai *attack* musuh setelah pemain melakukan peningkatan *attack* yaitu berjumlah 15. Nilai *attack* musuh tersebut merupakan hasil penyesuaian *attack* musuh setelah pemain melakukan peningkatan sejumlah 10 *attack* dari nilai *attack* pemain ketika pertama kali *game* dimulai, yang berarti pemain memiliki nilai *attack* berjumlah 20. Penyesuaian tersebut menggunakan rumus peningkatan *attack* yang telah dirancang sebelumnya.

Selanjutnya akan diuji ketika pemain melakukan peningkatan yang lebih banyak dari sebelumnya. Seperti yang diperlihatkan pada Gambar 4. 33 dimana pemain melakukan peningkatan *attack* hingga nilai *attack* pemain berjumlah 40.

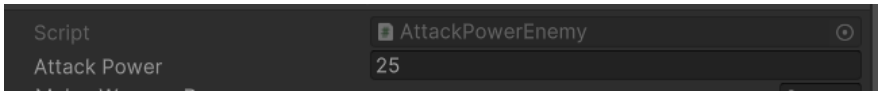
Gambar 4. 33 Peningkatan *Attack* pada Panel Upgrade

Pada Gambar 4. 33 dapat dilihat nilai *attack* pemain ditingkatkan sampai berjumlah 40 dan bisa dilihat pada Gambar 4. 34 menunjukkan perubahan nilai *attack* pemain menjadi 40.



Gambar 4. 34 Jumlah *Attack* Pemain Diperbarui

Setelah nilai *attack* pemain berubah seperti pada Gambar 4. 34 maka selanjutnya musuh akan menyesuaikan nilai *attack* miliknya seperti yang diperlihatkan pada Gambar 4. 35.



Gambar 4. 35 Peningkatan *Attack* Musuh

Gambar 4. 35 menunjukkan nilai attack musuh berjumlah 25. Nilai attack musuh tersebut merupakan hasil penyesuaian attack musuh setelah pemain melakukan peningkatan sejumlah 30 attack dari nilai attack pemain ketika pertama kali game dimulai, yang berarti pemain memiliki nilai attack berjumlah 40. Sama seperti sebelumnya penyesuaian tersebut menggunakan rumus peningkatan attack yang telah dirancang. Penyesuaian kesulitan yang

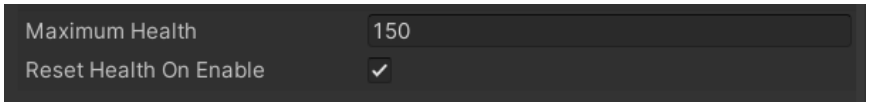
dinamis ini akan terus-menerus berlanjut hingga pemain berhasil dikalahkan atau permainan dihentikan.

2. Peningkatan *Health*

   Sama halnya dengan peningktan *attack*, kesulitan dinamis pada peningkatan health dimulai ketika pemain melakukan peningkatan status pada health. Peningkatan tersebut dilakukan pada panel *upgrade* yang terbuka pada saat pemain berhasil mengalahkan semua musuh pada setiap *wave* yang sedang dijalani oleh pemain. Panel *upgrade* tersebut dapat dilihat pada Gambar 4. 36.

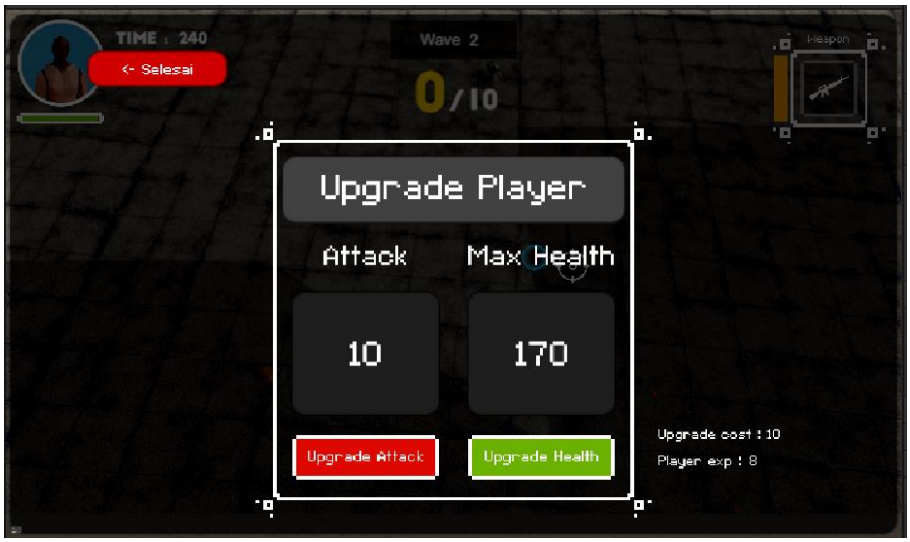
Gambar 4. 36 Tampilan Panel *Upgrade*

Gambar 4. 36 menunjukkan tampilan dari panel *upgrade*. Pada panel upgrade tersebut menampilkan dua atribut yang dapat ditingkatkan oleh pemain yaitu *attack* dan *health*. Pada awal permainan pemain memiliki *max health* dengan nilai 150. *Max health* merupakan jumlah maksimal *health* yang dapat dimiliki oleh pemain, semakin banyak *health* yang dimiliki pemain maka akan semakin lama juga pemain dapat bertahan dari serangan musuh yang muncul. Hal ini juga dapat dilihat pada parameter ”*Max Health*” pemain yang ditunjukkan pada Gambar 4. 37.



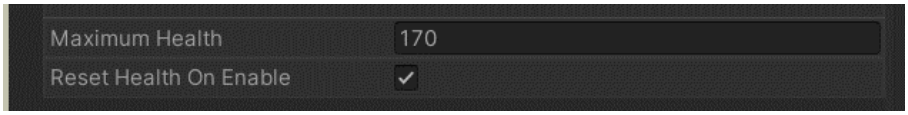
Gambar 4. 37 Jumlah *Max Health* Pemain

Gambar 4. 38 menunjukkan parameter ”*Maximum health*” yang merupakan nilai *max health* yang dimiliki pemain. Ketika permainan dimulai milai *max health* tersebut berjumlah 150. Selanjutnya pemain dapat melakukan peningkatan *max health* selama pemain memiliki *exp* yang cukup untuk melakukan peningkatan. *Exp* tersebut diperoleh dari membunuh musuh yang ada dalam permainan. Pada panel *upgrade* pemain dapat melihat jumlah *exp* yang dimilikinya pada atribut atau informasi tambahan dengan tulisan ”*Player exp*” yang ada pada bagian kanan bawah panel *upgrade* tersebut. Hal ini dapat dilihat lebih jelas pada Gambar 4. 38.



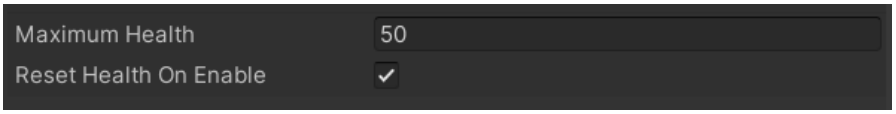
Gambar 4. 38 Peningkatan *Max Health* Pemain

Gambar 4. 38 memperlihatkan keadaan panel *upgrade* ketika pemain selesai melakukan peningkatan *max health*. Dalam panel *upgrade* tersebut dapat dilihat bahwa nilai *max health* yang dimiliki pemain setelah melakukan *max health* adalah sejumlah 170, yang berarti terdapat penambahan 20 *health* dari sebelumnya di awal permainan. Untuk membuktikan peningkatan tersebut berjalan dengan baik dapat dilihat pada Gambar 4. 39.



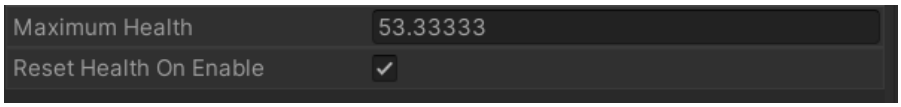
Gambar 4. 39 Jumlah *Max Health* Pemain Setelah Peningkatan

Gambar 4. 39 memperlihatkan perubahan nilai *max health* pemain pada parameter "*Maximum Health*" yang sebelumnya berjumlah 150 sekarang bertambah menjadi 170. Selanjutnya dengan adanya peningkatan tersebut akan mempengaruhi nilai *health* yang dimiliki musuh. Pada awalnya musuh memiliki jumlah *health* berjumlah 50. Hal ini dapat dilihat pada Gambar 4. 40.



Gambar 4. 40 *Max Health* Musuh pada Awal Permainan

Gambar 4. 40 menunjukkan parameter "*Maximum Health*" yang merupakan nilai *max health* yang dimiliki oleh musuh. Pada awal permainan nilai *max health* yang dimiliki oleh musuh adalah 50. Dan ketika pemain melakukan peningkatan *max health* pemain maka musuh akan menyesuaikan nilai *max health* yang dimilikinya menyesuaikan dengan peningkatan yang dilakukan oleh pemain. Hal ini lebih jelasnya dapat dilihat pada Gambar 4. 41.



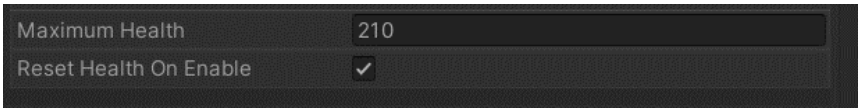
Gambar 4. 41 Peningkatan *Max Health* Musuh

Gambar 4. 41 menunjukkan nilai *max health* musuh setelah pemain melakukan peningkatan *max health* yaitu berjumlah 53,3. Nilai *max health* musuh tersebut merupakan hasil penyesuaian setelah pemain melakukan peningkatan sejumlah 20 *health* dari nilai *max health* pemain ketika pertama kali *game* dimulai, yang berarti pemain memiliki nilai *max health* berjumlah 170. Penyesuaian tersebut menggunakan rumus peningkatan *attack* yang telah dirancang sebelumnya.

Selanjutnya akan diuji ketika pemain melakukan peningkatan yang lebih banyak dari sebelumnya. Seperti yang diperlihatkan pada Gambar 4. 42 dimana pemain melakukan peningkatan *max health* hingga berjumlah 210.

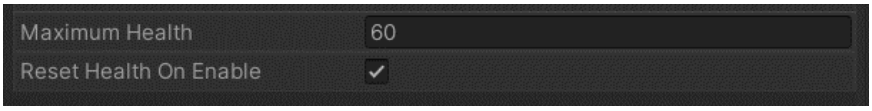
Gambar 4. 42 Peningkatan *Max Health* pada Panel Upgrade

Pada Gambar 4. 42 dapat dilihat nilai *max health* pemain ditingkatkan sampai berjumlah 210 dan bisa dilihat pada Gambar 4. 43 menunjukkan perubahan nilai *max health* pemain menjadi 210.



Gambar 4. 43 Jumlah *Max Health* Pemain Diperbarui

Setelah nilai *max health* pemain berubah seperti pada Gambar 4. 43. maka selanjutnya musuh akan menyesuaikan nilai *health* miliknya seperti yang diperlihatkan pada Gambar 4. 44.



Gambar 4. 44 Peningkatan *Max Health* Musuh

Gambar 4. 44 menunjukkan nilai *max health* musuh berjumlah 60. Nilai *max health* musuh tersebut merupakan hasil penyesuaian *max health* musuh

setelah pemain melakukan peningkatan sejumlah 60 *health* dari nilai *max health* pemain ketika pertama kali *game* dimulai, yang berarti pemain memiliki nilai *max health* berjumlah 210. Sama seperti sebelumnya penyesuaian tersebut menggunakan rumus peningkatan *attack* yang telah dirancang. Penyesuaian kesulitan yang dinamis ini akan terus-menerus berlanjut hingga pemain berhasil dikalahkan atau permainan dihentikan.

BAB 5

PENUTUP

5.1 Kesimpulan

Berdasarkan hasil implementasi dan pengujian sistem *infinite wave* ke dalam *game* "*From The Downtown*", diperoleh kesimpulan sebagai berikut:

1. Sistem *infinite wave* dapat diimplementasikan ke dalam *game* "*From The Downtown*" dan dapat memunculkan beragam jenis musuh mulai dari musuh normal hingga musuh *boss*.
2. Jenis musuh normal yang dimunculkan ada 4, yaitu *melee*, *chaser*, *range* dan *charger* dan jenis musuh *boss* yang dimunculkan ada 2, yaitu *melee boss* dan *range boss*.
3. Jenis musuh normal dimunculkan dengan nilai pembobotannya masing-masing, yaitu *melee* bernilai 35%, *chaser* bernilai 25%, *range* bernilai 20% dan *charger* bernilai 20%.
4. Musuh *melee* dan *chaser* mulai dimunculkan pada *wave* 1, musuh *range* mulai muncul pada wave 6 dan musuh *charger* mulai muncul pada *wave* 11.
5. Jenis musuh *boss* dimunculkan setiap 5 *wave* (gelombang musuh) sekali dan dimunculkan sebanyak 1 musuh *boss* secara bergantian.
6. Jumlah musuh yang dimunculkan pada tiap *wave* akan bertambah 5 dengan jumlah awal musuh pada saat permainan dimulai yaitu 10 dan jumlah maksimal musuh yang dapat dimunculkan dalam satu *wave* adalah 50 musuh.
7. Setiap 15 *wave* jumlah musuh yang dimunculkan kembali menjadi 10 musuh namun dengan status musuh yang meningkat.
8. Pada pergantian *wave* akan muncul *panel upgrade* yang berguna untuk meningkatkan status pemain.
9. *Dynamic Difficulty Adjustment* (DDA) diimplementasikan dengan status pemain sebagai parameternya. Setiap kali pemain meningkatkan status, maka musuh juga akan meningkatkan statusnya menyesuaikan status pemain.

5.2 Saran

Saran yang diberikan untuk penelitian selanjutnya adalah sebagai berikut:

1. Menambahkan jenis musuh dengan bentuk dan variasi serangan yang lebih beragam.
2. Membuat sistem skor dan leaderboard. Hal ini dapat memotivasi pemain untuk terus bersaing dengan pemain lainnya.
3. Memberikan efek suara dan visual yang lebih menarik pada setiap *event* yang ada pada *game*.
4. Memberikan tambahan fitur *timer* untuk memberikan pengalaman bermain yang lebih menarik dan menegangkan.

DAFTAR PUSTAKA

10tons Ltd (2014). *Crimsonland*.

10tons Ltd (2018). *Tesla vs Lovecraft*;

Activision (2007). *Geometry Wars*.

Adams, E (2010). Fundamentals Of *Game* Design, Second Edition. New Riders.

Goandy, H., Young, J. C., & Hansun, S. (2020). No Escape: A 2D *Top-Down* Shooting Roguelike *Game* Embedded with Drunkard Walk Algorithm. International Journal of Advanced Trends in Computer Science and Engineering, 9(2), 1045-1049.

Hind, D., & Harvey, C. (2022). A NEAT Approach to Wave Generation in Tower Defense *Game*s. 2022 International Conference on Interactive Media, Smart Systems and Emerging Technologies (IMET), 1-8.

Huber, T., Mertes, S., *Range*lova, S., Flutura, S., & André, E. (2021). *Dynamic Difficulty Adjustment* in Virtual Reality Exer*game*s through Experience-Driven Procedural Content Generation. 2021 IEEE Symposium Series on Computational Intelligence (SSCI), 1-8.

Melhart, D., Liapis, A., & Yannakakis, G. N. (2022). The arousal *video game* annotation (AGAIN) dataset. IEEE Transactions on Affective Computing, 13(4), 2171-2184.

Meireles, P. S., Silva-Calpa, G. F. M., & Raposo, A. B. (2020). Exploring Direct User-Interaction Techniques in A Virtual Reality *Game* for People with Hand Impairments. SBC-Proceedings of SB*Game*s.

Or, D. B., Kolomenkin, M., & Shabat, G. (2021). DL-DDA-Deep Learning Based *Dynamic Difficulty Adjustment* With Ux And *Game*play Constraints. 2021 IEEE Conference on *Game*s (CoG), 1-7.

Pereira, L. T., Viana, B. M., & Toledo, C. F. (2021). Procedural *Enemy* Generation through Parallel Evolutionary Algorithm. 2021 20th Brazilian Symposium on Computer *Game*s and Digital Entertainment (SB*Game*s). 126-135.

Padrones, F. V. (2022). Development of a Virtual Reality First Person *Shooter Game* with Procedural Map Generation.

Poncle (2022). *Vampire Survivors*.

Sigma Team Inc. (2003). *Alien Shooter*.

Tania, V. C., Pragantha, J., & Haris, D. A. (2021). Pembuatan *Game* Shoot 'Em Up "Stop The *Alien*s" Dengan Fitur Accelerometer Menggunakan Unity Berbasis Android. Jurnal Ilmu Komputer dan Sistem informatika, 9(1), 202-208.

Zamith, M., da Silva Junior, J. R., Clua, E. W., & Joselli, M. (2020). Applying Hidden Markov Model for Dynamic *Game* Balancing. 2020 19th Brazilian Symposium on Computer *Game*s and Digital Entertainment (SB*Game*s). 38-46.

Zohaib, M. (2018). *Dynamic Difficulty Adjustment* (DDA) in Computer *Game*s: A Review. Hindawi Advances in Human-Computer Interaction.