Procedural Content Generation pada *Game* Tower Defense menggunakan *Perlin Noise* dan Algoritma *Floyd Warshall*

[1]Hans Juwiantho, [2]Liliana Liliana, [3]Michael Budiono
[123]Program Studi Informatika, Fakultas Teknologi Industri, Universitas Kristen Petra
[1]hans.juwiantho@petra.ac.id, [2]lilian@petra.ac.id, [3]c14170076@john.petra.ac.id

Abstrak

Game tower defense membutuhkan desain map, rute perjalanan musuh, dan desain wave musuh yang digunakan untuk membedakan setiap tingkat. Desain yang dirancang secara manual membutuhkan banyak tenaga dan waktu. Untuk mengatasi masalah tersebut digunakan procedural content generation untuk membuat map secara otomatis. Tidak semua konten bisa dibuat secara otomatis, beberapa konten yang didesain secara manual pada penelitian ini adalah desain portal, desain tower, desain musuh, dan desain ubin. Desain map dibuat secara otomatis menggunakan *Perlin Noise* untuk menentukan jenis ubin pada map dan dilakukan pengecekan letak portal pemain dan portal musuh menggunakan algoritma *Floyd-Warshall* untuk menentukan apakah map sudah sesuai persyaratan jarak antar portal. Hasil pengujian menunjukkan setelah dilakukan sebanyak 100 kali, 25 kali percobaan perlu diulang karena jarak antar portal tidak sesuai aturan. Rata-rata waktu pembuatan map hanya 0,99 detik. *Wave* musuh pada setiap map juga berbeda-beda dari jumlah setiap jenis dan urutan musuh keluar.

Kata kunci: Pembuatan Konten Prosedural, Perlin Noise, Algoritma Floyd Warshall, Tower Defense

Procedural Content Generation in Tower Defense Game using Perlin Noise and Floyd Warshall Algorithm

Abstract

To distinguish each level, tower defense games require map design, enemy travel routes, and enemy wave designs. Manually designed designs require a lot of effort and time. To overcome this problem, procedural content generation is used to create maps automatically. Not all content can be created automatically, some of the content designed manually in this research is portal design, tower design, enemy design, and tile design. The map design is created automatically using Perlin Noise to determine the type of tiles on the map and using Floyd-Warshall algorithm to check the location of player and enemy portals whether the map meets the distance requirements. The test results show that 25 out of 100 experiment needs to be repeated. The average map creation time is only 0.99 seconds. The enemy waves on each map also vary from the number of each type and the order in which the enemies come out.

Keywords: Procedural Content Generation, Perlin Noise, Floyd Warshall Algorithm, Tower Defense

Pendahuluan

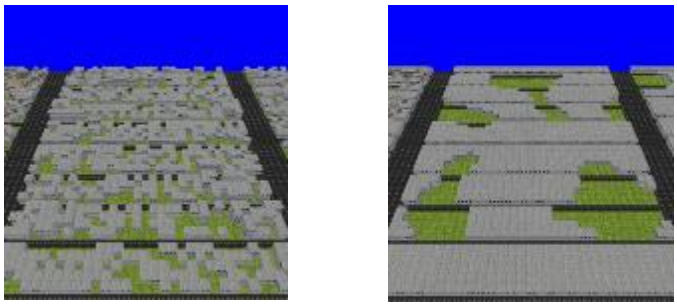
Pembuatan *game tower defense* memerlukan desainer *game* untuk merancang setiap map permainan. Suatu *game tower defense* membutuhkan beberapa desain map yang

digunakan untuk membedakan setiap tingkat pada *game*. Jika desain map pada *game tower defense* dirancang secara manual untuk setiap levelnya, membutuhkan banyak tenaga dan waktu. Dalam mendesain map untuk *game tower defense* harus ada 3 elemen standar, yaitu map yang dibuat harus memiliki rute perjalanan musuh dari titik awal menuju titik tujuan, memiliki tantangan yang cukup dari urutan monster yang di *spawn*, dan dari jenis *tower* yang dapat dibangun (Du et al., 2019). Proses yang perlu dilakukan saat pembuatan map untuk game tower defense yaitu proses mendesain map itu sendiri, mendesain lokasi setiap *tower*, mendesain gambar yang digunakan, mendesain rute perjalanan monster, dan melakukan uji coba terhadap *map* apakah dapat dimainkan atau tidak (Liu et al., 2019).

Selain pembuatan map pada *tower defense* yang membutuhkan waktu yang lama, penggunaan map yang sama pada beberapa level juga akan membuat pemain bosan karena *replayability* yang rendah. Map yang berbeda-beda dapat membuat pemain tertarik untuk memainkan *game* kembali karena ada hal baru yang menantang dan menarik setiap *game* dimainkan. Untuk mempercepat proses pembuatan map pada *game tower defense* dan menghasilkan map berbeda-beda, diperlukan algoritma yang dapat membuat desain map secara otomatis. Dengan membuat desain map secara otomatis, *game tower defense* dapat memiliki banyak variasi map berbeda yang dapat dimainkan. *Procedural content generation* (PCG) dapat digunakan untuk menghasilkan konten pada *game* secara otomatis. Konten dalam *game* bisa berupa level, map, aturan permainan, *story*, item, *quest*, dan karakter (Yannakakis & Togelius, 2018). Penggunaan PCG dapat menggantikan desainer dalam melakukan desain, khususnya desain map yang membutuhkan cukup banyak waktu dan proses dalam pembuatannya (Öhman, 2020). Hasil dari PCG sangat mempengaruhi *replayability* dalam *game* karena memberikan pengguna petualangan baru setiap *game* dijalankan ulang (Risi & Preuss, 2020).

Generator map dibagi menjadi 3 kategori, yaitu *Random* Level Generator, *Constructive* Level Generator, dan *Search-based* Level Generator (Stangl, 2017). Teknik generator yang digunakan pada penelitian ini menggabungkan *random* level generator dan *constructive* level generator. *Random* level generator yang digunakan tidak hanya sekedar *random*, tetapi sudah ditingkatkan hasilnya dengan *Perlin Noise* yang ditemukan oleh Ken Perlin (Perlin, 1985). *Perlin Noise* menghasilkan tekstur dan medan secara prosedural tanpa dibuat manual. Fungsi yang digunakan adalah *pseudo-random* yang

memberikan nilai hasil *random* yang selalu menghasilkan nilai yang sama jika memberikan *input seed* yang sama. Dari hasil *random* kemudian diterapkan *Noise* untuk menghasilkan transisi yang halus dari satu area ke area lain dan teksturnya tidak terlihat acak di area tertentu. Contoh perbedaan hasil *generate* map dengan *random* dan dengan *Perlin Noise* dapat dilihat pada Gambar 1.



Gambar 1. Hasil *random* (kiri) dan hasil *Perlin Noise* (kanan) (Frank & Olsson, 2017)

Setiap map perlu memiliki rute penyerangan dari monster yang berbeda sehingga strategi yang diperlukan untuk menyelesaikan setiap map juga berbeda. Penyerangan monster yang terlalu sederhana seperti hanya berjalan lurus dari satu titik ke titik lain juga membuat map tersebut menjadi membosankan dan kurang menarik karena kurangnya tantangan yang dihadapi pemain serta peletakan *tower* untuk setiap level pasti selalu sama jika terjadi kondisi tersebut. Oleh karena itu, selain desain map yang berbeda beda, *game tower defense* juga memerlukan desain rute monster yang berbeda beda juga untuk dapat dimainkan berulang kali.

Rute monster ditentukan dengan cara menentukan 2 titik sebagai posisi awal dan tujuan dari monster. Setiap monster muncul di posisi awal dan berjalan ke posisi tujuan. Untuk melihat apakah rute yang dihasilkan bisa dilewati monster dari posisi awal ke posisi tujuan, perlu di lakukan pengecekan rute. Algoritma yang dapat digunakan untuk melakukan pengecekan rute serta mencari rute terpendek contohnya *dijkstra*, A* dan *Floyd-Warshall* (Umar et al., 2021). Algoritma *dijkstra* dan A* mencari rute terpendek berdasarkan rute yang sedang ditelusuri, sehingga bisa menghasilkan rute pendek dengan cepat tetapi belum tentu yang terbaik dari semua kemungkinan rute yang ada (Azis et al., 2018). Sebaliknya untuk Algoritma *Floyd-Warshall*, proses yang dilakukan sedikit lebih lama karena mengecek semua kemungkinan rute yang ada dan menghasilkan rute yang terbaik dari semua kemungkinan rute tersebut (Risald et al., 2017). Pada penelitian ini akan digunakan algoritma *Floyd-Warshall* untuk mengecek seberapa panjang rute yang telah *digenerate* dan apakah rute sudah sesuai dengan ketentuan minimum panjang rute

yang diinginkan, sehingga hasil rute tidak terlalu berdekatan antara titik awal dan titik akhir.

Metode

Pada bagian ini akan dibahas mengenai *Procedural content generation* dengan metode *Perlin Noise* dalam pembuatan map dan algoritma *Floyd-Warshall* dalam mencari rute terbaik dari dua titik beserta rumus-rumus yang digunakan.

A. *Perlin Noise*

Perlin Noise merupakan salah satu tipe dari *gradient noise* yang menghasilkan nilai yang mirip pada suatu posisi dan area di sekitar posisi yang terkena *Noise* (Öhman, 2020). *Perlin Noise* sendiri dikenalkan oleh Ken Perlin pada tahun 1985 (Perlin, 1985). Tipe lain dari *gradient noise* selain *perlin noise* meliputi *simplex noise* dan *simulation noise. Simplex noise* memiliki waktu proses yang lebih cepat dari *perlin noise*, tetapi *perlin noise* memiliki hasil bentuk map dua dimensi daratan lebih baik dibandingkan dengan *simplex noise* untuk digunakan pada game (Wijaya & Rahman, 2018).

Pembuatan map pada penelitian ini mengimplementasikan *perlin noise* untuk membuat *terrain* yang dihasilkan terlihat unik dan natural. Hasil dari map yang dikenakan *perlin noise* memiliki transisi antar ubin yang halus dari satu area ke area lain dan teksturnya tidak terlihat acak di area yang terkena *perlin noise* yang dibutuhkan dalam game ini untuk rute perjalanan musuh dari posisi awal menuju posisi akhir. Selain menggunakan *noise,* pembuatan map juga bisa menggunakan fungsi *random,* tetapi hasil map yang diperoleh kurang natural dan kurang baik untuk digunakan dalam game tower defense karena menghasilkan transisi yang terlalu kasar dari satu area ke area lain dan teksturnya sangat acak, sehingga tidak layak digunakan dalam game yang dibuat.

Cara kerja dari *Perlin Noise* dengan mengambil nilai pada suatu posisi sebagai *input* dan mengembalikan nilai dalam *range* tertentu. *Range* bisa antara -1.0 hingga +1.0 atau akar dari dimensi yang digunakan dibagi dengan 2. Parameter *input* dari *Perlin Noise* yang digunakan sesuai dengan dimensinya. Parameter yang digunakan pada 2 dimensi merupakan posisi x dan y sebagai *input*. Pada pembuatan map, setiap posisi yang ada dikenakan *Perlin Noise* untuk setiap koordinatnya dan menghasilkan nilai baru sebagai penentu lokasi tersebut harus diisi dengan bidang apa.

B. Algoritma *Floyd-Warshall*

Algorima *Path Finding* dibutuhkan untuk mencari jarak terpendek dari *vertices* ke *vertices* lainnya pada suatu *graph*. Macam-macam algoritma *path finding* contohnya *dijkstra*, A* dan *Floyd-Warshall*. Algoritma *dijkstra* dan A* mencari rute terpendek dari satu *vertices* menuju satu lokasi *vertices* lainnya, sehingga menghasilkan rute pendek dengan cepat untuk satu *vertices* (Azis et al., 2018), tetapi akan lama jika mencari semua rute terpendek untuk semua *vertices.* Algoritma *Floyd-Warshall* menghasilkan semua rute terpendek untuk semua *vertices,* waktu proses algoritma *Floyd-Warshall* lebih lama jika dibandingkan dengan algoritma *djikstra* atau A* yang hanya mencari satu rute terpendek untuk satu *vertices* (Risald et al., 2017). Pada penelitian ini digunakan algoritma *Floyd-Warshall* untuk mengecek seberapa panjang maksimal rute yang telah *digenerate* untuk semua *vertices* yang ada dan apakah rute yang dihasilkan sudah sesuai dengan ketentuan minimum panjang rute yang ditentukan, sehingga diperoleh map yang memiliki standar yang sama dengan minimum rute terpanjang berdasarkan angka yang diinputkan.

Algoritma ini bekerja dengan cara mengecek apakah ada jalur dari dua titik melalui titik lain yang dapat menghasilkan rute perjalanan yang lebih dekat dari rute yang telah ada (Risald et al., 2017). Saat melakukan pengecekan pada suatu titik X sebagai perantara antara titik A menuju titik C, jika jarak dari A ke X ditambahkan dengan jarak X ke C lebih pendek dari pada jarak A ke C yang telah dihitung sebelumnya, maka hasil yang digunakan untuk menghitung jarak dari A ke C adalah jarak A ke X ditambah X ke C. Tidak terjadi perubahan jika jarak A ke C sudah lebih pendek.

Pembahasan

Hasil yang telah dilakukan pada penelitian ini meliputi desain *game tower defense*, desain beberapa macam ubin untuk pembuatan map, desain *tower*, desain musuh, desain pembuatan map, dan pengujian *Perlin Noise* terhadap pembuatan map yang menggunakan algoritma *Floyd-Warshall* pada rute musuh dari titik awal hingga titik tujuan.

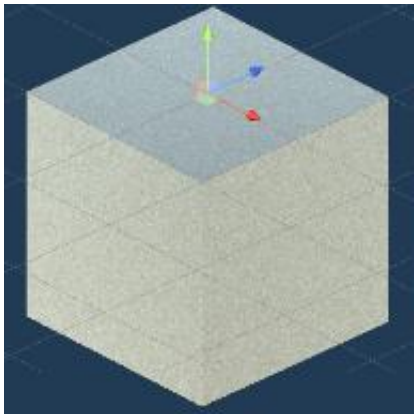
Pemain memainkan *game* dengan tujuan untuk melindungi suatu daerah agar tidak dicapai oleh musuh. Pemain dapat membangun suatu bangunan yang dapat menyerang

musuh secara otomatis. Serangan musuh yang didesain memiliki 3 gelombang serangan monster. Pemain menang jika berhasil mempertahankan lokasi yang sudah ditentukan dari 3 gelombang serangan monster dan kalah jika 5 musuh berhasil mencapai lokasi tersebut.

A. Desain Ubin

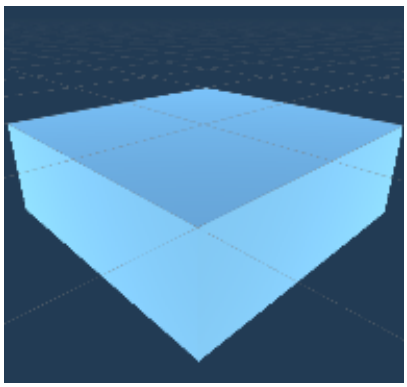
Map yang didesain terbentuk dari kumpulan ubin dengan jumlah sebanyak 540 yang digabungkan dengan panjang 30 ubin dan lebar 18 ubin. Pembuatan map menggunakan *procedural content generation* dengan *Perlin Noise*. Tiap ubin dibedakan menjadi tiga macam jenis berdasarkan kegunaan pada ubin tersebut yang setiap ubin memiliki kegunaan yang berbeda. Berikut adalah pembagian ubin berdasarkan jenis yang ada pada *game*:

1. Ubin dengan jenis jalan: Merupakan ubin yang dapat dilewati oleh musuh. Ubin ini dapat ditempati oleh satu *tower*. *Asset* dari ubin jalan dapat dilihat pada Gambar 2.



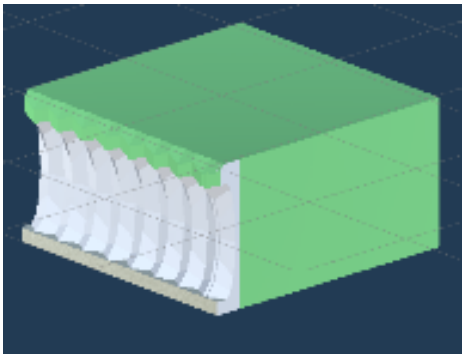
Gambar 2. *Asset* ubin dengan jenis jalan

2. Ubin dengan jenis perairan: Ubin ini tidak dapat dilewati oleh musuh dan tidak dapat ditempati oleh bangunan. *Asset* dari jenis perairan dapat dilihat pada Gambar 3.



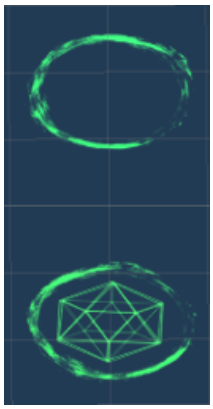
Gambar 3. *Asset* ubin dengan jenis perairan

3. Ubin dengan jenis dataran tinggi: Ubin ini tidak dapat dilewati oleh musuh, tetapi ubin ini dapat ditempati oleh satu bangunan. *Asset* dari jenis dataran tinggi dapat dilihat pada Gambar 4.



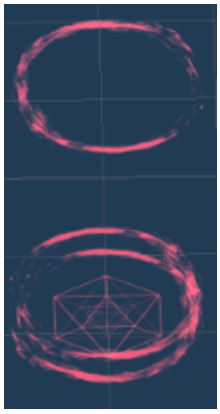
Gambar 4. *Asset ubin* dengan jenis dataran tinggi

Desain untuk lokasi yang harus dijaga pemain berbentuk lingkaran hijau dengan animasi. Pada lokasi ini tidak dapat dibangun *tower* oleh pemain dan dapat dilihat sejak *game* dijalankan. Monster berjalan dari posisi awal dan menuju ke posisi ini. Lokasi yang perlu dijaga oleh pemain dari awal hingga akhir hanya satu. *Asset* yang menunjukkan posisi pemain dapat dilihat pada Gambar 5.



Gambar 5. *Asset* lokasi yang harus dijaga

Asset yang digunakan sebagai tanda lokasi awal monster untuk hidup mirip dengan *asset* lokasi yang harus dijaga pemain tetapi memiliki warna merah. Tower tidak dapat dibangun pada posisi ini. *Asset* yang digunakan untuk lokasi awal monster hidup dapat dilihat pada Gambar 6.



Gambar 6. *Asset portal* musuh

B. Desain Tower

Tower merupakan sebuah objek bangunan yang dapat dibangun oleh pemain untuk menahan serangan musuh. Tower yang telah dibangun dapat menyerang musuh yang ada di sekitarnya secara otomatis sesuai dengan radius jarak tembaknya. Untuk membangun *tower*, pemain memerlukan uang dalam *game* berupa *gold*. Tower yang menghalangi jalan dari monster dapat diserang oleh monster. Tower yang diserang oleh musuh hingga *health tower* menjadi 0 maka *tower* tersebut hancur. Pada game ini jenis tower dibedakan menjadi 2 macam *tower* yaitu *tower* tipe tank dan tipe *damage*. Tipe *damage* memiliki *attribute attack power* (serangan yang dapat diberikan ke musuh saat *tower* menyerang) lebih tinggi dari tipe tank dan tipe tank (ketahanan *tower* saat diserang oleh musuh) memiliki *attribute health* lebih tinggi dari tipe *damage*. Berikut adalah desain *tower* yang ada pada *game* ini:

1. Tower tipe *damage*: Tower ini memiliki *attack power* yang tinggi dan *health* yang rendah. *Asset* dari *tower* tipe *damage* dapat dilihat pada Gambar 7.

Gambar 7. *Asset* tower tipe *damage*

2. Tower tipe tank: Tower ini memiliki *health* yang tinggi dan *attack power* yang rendah. *Asset* dari *tower* tipe tank dapat dilihat pada Gambar 8.

Gambar 8. *Asset tower* tipe tank

Atribut dari desain *tower* yang ada pada *game* ini dapat dilihat pada Tabel 1. Dapat dilihat *attack power tower* tipe *damage* lebih besar dari pada *tower* tipe tank dan sebaliknya untuk *health* pada tipe tank.

Tabel 1. Desain *attribute tower*

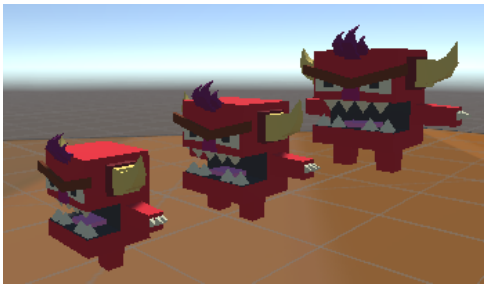
Tower	***Attack power***	***Health***	***Cost***
Tower tipe *damage*	70	400	1000 *Gold*
Tower tipe tank	35	800	1000 *Gold*

C. Desain Musuh

Musuh adalah karakter yang dihadapi oleh pemain. Musuh pada *game* ini dibedakan menjadi beberapa macam berdasarkan ukuran. Ukuran musuh membedakan atribut seperti *attack power* dan *health*. Musuh memiliki warna merah dengan 3 macam ukuran yaitu kecil, sedang, dan besar. *Asset* dari musuh dapat dilihat pada Gambar 9.

Gambar 9. *Asset* musuh A

Perbandingan musuh berdasarkan ukuran yang ada pada *game* ini dapat dilihat pada Gambar 10 musuh yang berada di depan memiliki ukuran kecil, musuh yang berada di tengah memiliki ukuran sedang, dan musuh yang ada di belakang memiliki ukuran besar. Perbandingan ukuran dari kecil, sedang, dan besar adalah 0.75 untuk kecil, 1 untuk sedang, dan 1.5 untuk besar.



Gambar 10. Macam-macam ukuran musuh

Atribut lengkap yang dimiliki oleh musuh berdasarkan ukuran yang ada pada *game* ini dapat dilihat pada Tabel 2. Setiap musuh memiliki *attack power*, *health*, dan *gold* yang diberikan setelah mati berbeda-beda. Semakin besar ukuran musuh, semakin banyak *gold* yang didapatkan.

Tabel 2. Atribut musuh berdasarkan ukuran

Ukuran	***Attack power***	***Health***	***Gold gain***
Kecil	25	100	25
Sedang	50	200	50
Besar	100	400	100

Sebelum musuh dikeluarkan dari posisi awal, *game* membuat daftar musuh yang dikeluarkan secara *random* dengan memilih antara 3 ukuran. Setiap ronde serangan memiliki batas dari total *health* dari seluruh musuh. Pada *game* ini dibatasi dengan 3 ronde serangan yang berbeda, pada ronde serangan pertama, musuh hanya berukuran kecil dan sedang dengan target total *health* adalah 8000. Saat ronde serangan selanjutnya, musuh memiliki segala bentuk ukuran dari berukuran kecil, sedang, hingga besar dengan target total *health* yang lebih banyak yaitu 12000. Sedangkan pada ronde terakhir, musuh ukuran kecil tidak digunakan, musuh yang digunakan adalah musuh ukuran sedang dan besar serta terdapat peningkatan target total *health* hingga 20000.

D. Desain Pembuatan Map

Proses pembuatan map dimulai dengan melakukan *random* pada variabel *offset* untuk menghasilkan map yang berbeda setiap program atau *game* dijalankan. Untuk setiap ubin pada map akan dihitung nilai *Perlin* dari ubin tersebut berdasarkan lokasi posisi dari ubin, skala *Perlin*, dan nilai *offset*. Hasil dari perhitungan nilai *Perlin* digunakan untuk menentukan jenis ubin dengan mencocokkan hasil perhitungan batasan sesuai dengan batas yang telah ditentukan. Ubin perairan dihasilkan jika hasil dari nilai *Perlin* lebih kecil dari 0.33, ubin yang bisa dilalui atau ubin jalan diperoleh dari nilai

Perlin antara 0.33 hingga 0.66, dan ubin daratan tinggi tempat menaruh tower yang tidak bisa diserang jika nilai *Perlin* lebih besar dari 0.66.

Nilai *Perlin* diatur untuk memiliki hasil antara 0.0 hingga 1.0 sehingga map memiliki ubin dengan jenis perairan, jenis jalan, dan daratan tinggi sama banyaknya dan tidak terlalu sedikit pada salah satu jenis ubin. Setelah nilai *Perlin* untuk setiap ubin dihitung maka selanjutnya menentukan lokasi portal pemain dan portal musuh menggunakan algoritma *Floyd-Warshall*. Setelah itu dilakukan pengecekan apakah map perlu untuk dibuat ulang atau tidak.

Pengecekan dilakukan dengan mengecek jarak antar semua ubin dengan jenis jalan yang ada pada map menggunakan algoritma *Floyd-Warshall*. Dari hasil pengecekan jarak, dipilih 2 ubin jalan yang memiliki hasil perhitungan jarak terbesar dengan menaruh portal pemain pada ubin pertama dan portal musuh diletakan pada ubin kedua. Jika jarak antara portal musuh dengan portal pemain tidak mencapai 45 ubin maka map dibuat ulang dengan proses yang sama hingga ditemukan map yang memenuhi aturan tersebut.

E. Desain *Game* Tower *Defense*

Pada saat permainan berlangsung, musuh muncul pada lokasi portal musuh sesuai dengan ronde serangan musuh yang telah ditentukan. Jika semua musuh yang ada pada suatu ronde telah dikalahkan maka permainan berlanjut ke ronde serangan musuh berikutnya hingga ronde serangan musuh selesai. Setiap musuh yang berhasil dibunuh oleh pemain memberikan *gold* pada pemain dengan jumlah yang sesuai dengan ukuran musuh yang telah dikalahkan. Pemain dapat membangun *tower* dengan cara melakukan klik pada *list tower* yang ada di layar dan klik lagi pada map untuk meletakkan *tower* tersebut. Proses permainan berakhir jika pemain berhasil bertahan hingga serangan musuh pada ronde serangan selesai atau nyawa berkurang hingga menjadi 0.

F. Pengujian *Perlin Noise* Terhadap Pembuatan Map

Pengujian dilakukan untuk memastikan hasil map yang didapatkan berbeda-beda setiap program atau *game* dijalankan serta untuk mengetahui berapa lama proses pembuatan map tersebut. Jika map yang dihasilkan tidak memiliki kriteria yang ditentukan, map kemudian dibuat ulang dengan proses yang sama. Map dikatakan sama jika map memiliki lokasi portal pemain, lokasi portal musuh, dan jarak antar portal yang sama dengan map lainnya. Pengujian dilakukan dengan cara melakukan pembuatan map secara berulang sebanyak 100 kali. Map dibuat menggunakan *Perlin Noise* dengan

parameter ukuran panjang 30 ubin dan lebar 18 ubin, skala *Perlin* 4 dan nilai *offset* yang didapat secara *random*. Pemilihan lokasi portal pemain dan portal musuh menggunakan algoritma *Floyd-Warshall*, lokasi dipilih berdasarkan pemilihan antara 2 titik yang paling jauh dari map yang telah dibuat dengan *Perlin Noise*. Contoh beberapa hasil dari map yang telah dibuat dan telah memenuhi persyaratan dapat dilihat pada Tabel 3.

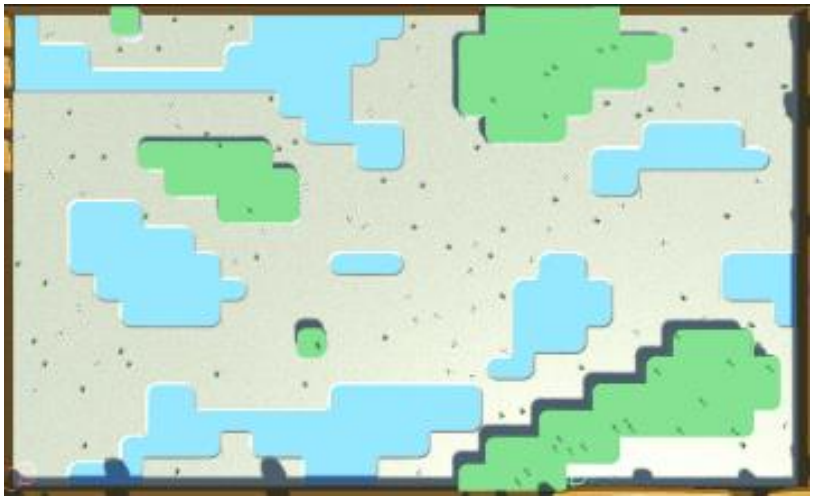
Tabel 3. Hasil pengujian pembuatan map

Percobaan ke	Lokasi portal pemain	Lokasi portal musuh	Jarak antar Portal	Waktu Pembuatan
1	x=21, y=0	x=0, y=0	55	1,1923 Detik
2	x=29, y=13	x=0, y=6	46	0,8516 Detik
3	x=29, y=0	x=28, y=17	56	0,8790 Detik
4	x=26, y=16	x=0, y=17	53	0,8576 Detik
5	x=29, y=0	x=0, y=0	47	0,8987 Detik
6	x=25, y=17	x=0, y=0	52	0,8514 Detik
7	x=0, y=17	x=0, y=3	48	0,8200 Detik
8	x=29, y=0	x=0, y=17	46	1,3027 Detik
9	x=16, y=17	x=0, y=0	45	0,8322 Detik
10	x=29, y=12	x=29, y=0	50	0,8857 Detik
11	x=0, y=17	x=0, y=5	66	1,2051 Detik
12	x=27, y=0	x=0, y=15	46	1,3951 Detik
13	x=28, y=17	x=9, y=17	51	0,8872 Detik
14	x=0, y=15	x=0, y=1	50	0,8356 Detik
15	x=29, y=5	x=0, y=14	48	0,8981 Detik
16	x=0, y=17	x=0, y=9	68	0,8803 Detik
17	x=29, y=17	x=0, y=0	46	0,8552 Detik
18	x=27, y=0	x=0, y=14	45	0,8993 Detik
19	x=29, y=17	x=0, y=0	46	1,7590 Detik
20	x=0, y=10	x=0, y=1	71	0,8296 Detik
21	x=29, y=0	x=0, y=17	46	0,8583 Detik
22	x=29, y=14	x=29, y=0	66	1,2465 Detik
23	x=29, y=9	x=5, y=17	54	1,2849 Detik
24	x=29, y=17	x=0, y=1	47	0,9090 Detik
25	x=29, y=16	x=0, y=0	47	1,3692 Detik
26	x=14, y=0	x=0, y=3	69	0,7731 Detik
27	x=29, y=11	x=1, y=14	47	0,8759 Detik
28	x=29, y=1	x=0, y=17	47	0,8809 Detik
29	x=29, y=0	x=7, y=17	51	0,8737 Detik
30	x=29, y=17	x=3, y=0	49	0,8360 Detik
31	x=29, y=17	x=0, y=1	47	0,8966 Detik
32	x=29, y=17	x=9, y=0	55	0,8714 Detik
33	x=29, y=0	x=9, y=0	50	0,8544 Detik

34	x=29, y=0	x=0, y=17	46	0,9396 Detik
35	x=12, y=1	x=0, y=8	51	0,9001 Detik
36	x=29, y=12	x=14, y=17	58	0,8682 Detik
37	x=29, y=17	x=0, y=5	51	0,7735 Detik
38	x=29, y=10	x=0, y=0	51	1,4061 Detik
39	x=29, y=0	x=0, y=1	60	0,8245 Detik
40	x=8, y=6	x=1, y=13	60	1,3058 Detik
41	x=29, y=0	x=1, y=17	47	0,9561 Detik
42	x=29, y=5	x=10, y=0	56	0,8859 Detik
43	x=29, y=0	x=0, y=17	46	0,8715 Detik
44	x=29, y=17	x=1, y=0	47	1,2983 Detik
45	x=1, y=17	x=0, y=0	46	0,8418 Detik
46	x=20, y=0	x=16, y=0	74	0,8188 Detik
47	x=29, y=17	x=29, y=0	53	0,9280 Detik
48	x=29, y=0	x=3, y=0	48	1,3062 Detik
49	x=28, y=17	x=18, y=0	63	1,6195 Detik
50	x=29, y=8	x=29, y=1	49	1,2485 Detik
51	x=29, y=0	x=0, y=9	46	0,8822 Detik
52	x=29, y=0	x=8, y=0	63	0,8581 Detik
53	x=29, y=0	x=3, y=17	49	0,9048 Detik
54	x=12, y=2	x=0, y=0	54	0,8724 Detik
55	x=29, y=0	x=6, y=17	50	1,8246 Detik
56	x=29, y=6	x=7, y=17	55	0,8612 Detik
57	x=29, y=17	x=0, y=0	46	0,8498 Detik
58	x=1, y=17	x=0, y=0	70	0,9244 Detik
59	x=29, y=14	x=16, y=8	47	0,8896 Detik
60	x=29, y=17	x=0, y=0	46	0,9369 Detik
61	x=29, y=0	x=21, y=17	51	0,8710 Detik
62	x=29, y=17	x=1, y=0	47	0,9011 Detik
63	x=29, y=14	x=0, y=8	47	0,8495 Detik
64	x=24, y=0	x=0, y=17	49	1,2472 Detik
65	x=29, y=4	x=1, y=17	47	0,8255 Detik
66	x=29, y=17	x=0, y=17	49	0,8755 Detik
67	x=29, y=0	x=0, y=17	46	0,8955 Detik
68	x=29, y=9	x=0, y=0	52	0,7941 Detik
69	x=29, y=0	x=0, y=17	46	0,9005 Detik
70	x=3, y=11	x=0, y=13	51	1,2708 Detik
71	x=19, y=13	x=0, y=16	58	0,8235 Detik
72	x=14, y=0	x=10, y=0	54	1,2615 Detik
73	x=29, y=0	x=6, y=17	46	0,8393 Detik
74	x=29, y=15	x=29, y=4	63	0,8194 Detik
75	x=25, y=17	x=0, y=17	63	0,8044 Detik
76	x=29, y=17	x=1, y=0	47	0,8972 Detik

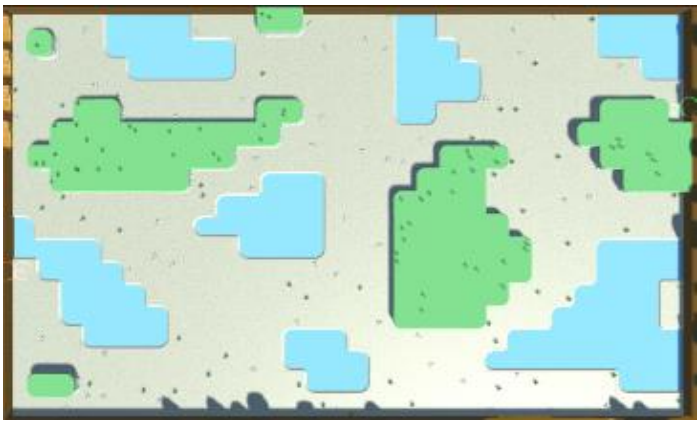
77	x=29, y=1	x=0, y=0	48	1,2525 Detik
78	x=29, y=0	x=0, y=17	48	0,8624 Detik
79	x=29, y=17	x=0, y=9	49	0,9034 Detik
80	x=29, y=17	x=0, y=0	46	0,8899 Detik
81	x=29, y=17	x=3, y=9	52	0,9274 Detik
82	x=27, y=9	x=5, y=17	54	1,2936 Detik
83	x=10, y=4	x=3, y=9	76	0,8190 Detik
84	x=29, y=0	x=0, y=17	46	0,8680 Detik
85	x=29, y=17	x=0, y=0	46	1,8028 Detik
86	x=28, y=17	x=0, y=0	47	0,8736 Detik
87	x=29, y=17	x=0, y=0	46	0,8524 Detik
88	x=22, y=17	x=0, y=0	47	1,3454 Detik
89	x=14, y=10	x=0, y=0	46	0,8863 Detik
90	x=25, y=0	x=0, y=17	48	0,9480 Detik
91	x=0, y=17	x=0, y=0	67	0,8397 Detik
92	x=29, y=5	x=0, y=17	45	0,8317 Detik
93	x=28, y=4	x=0, y=13	49	0,8781 Detik
94	x=29, y=0	x=0, y=12	45	0,8733 Detik
95	x=29, y=17	x=9, y=0	51	0,8758 Detik
96	x=29, y=17	x=0, y=0	46	0,9182 Detik
97	x=29, y=1	x=1, y=17	48	1,3096 Detik
98	x=29, y=0	x=0, y=17	46	0,9048 Detik
99	x=29, y=14	x=1, y=0	48	1,3520 Detik
100	x=29, y=1	x=0, y=17	49	1,2374 Detik

Contoh map pada percobaan pertama dari Tabel 3 dapat dilihat pada Gambar 11. Pada map tersebut lokasi portal pemain pada posisi x=21 y=0 dan lokasi portal musuh pada x=0 y=0. Dari lokasi portal tersebut didapat jarak portal musuh dengan portal pemain adalah 55 ubin karena terdapat ubin perairan dan dataran tinggi yang membuat perjalanan musuh memutar.



Gambar 11. Hasil map percobaan pertama

Contoh map pada percobaan kedua dari Tabel 3 dapat dilihat pada Gambar 12. Pada map tersebut lokasi portal pemain pada *grid* adalah x=29 y=13 dan lokasi portal musuh pada adalah x=0 y=6. Dari lokasi portal tersebut maka akan didapat jarak portal musuh pertama dengan portal pemain adalah 46 ubin.



Gambar 12. Hasil map percobaan kedua

Dapa dilihat pada Gambar 11 dan Gambar 12 menghasilkan map yang berbeda. Map pertama memiliki lokasi portal dan jarak portal yang berbeda dengan map kedua. Jarak antar portal musuh dan portal pemain juga berbeda sehingga kedua map tersebut dapat dikatakan berbeda. Dari hasil percobaan yang dilakukan sebanyak 100 kali, beberapa posisi portal pemain, portal musuh, dan jarak portal memiliki hasil yang sama, tetapi map yang dihasilkan berbeda serta rute yang dilalui oleh musuh berbeda juga seperti percobaan 17 dan 19 serta percobaan 57 dan 60. Selain map yang berbeda, komposisi musuh yang keluar juga berbeda, pada percobaan 17 wave 1 mengeluarkan 26 musuh kecil dan 27 musuh sedang sedangkan percobaan 19 mengeluarkan 30 musuh kecil dan 25 musuh sedang.

Pada proses pembuatan map, jika hasil pembuatan map tidak memenuhi persyaratan jarak antar portal, maka map tersebut dibuat ulang. Beberapa contoh dari map yang tidak memenuhi persyaratan dan akan dibuat ulang dapat dilihat pada Tabel 4.

Tabel 4. Sampel data pembuatan map ulang

Percobaan ke	Lokasi portal pemain	Lokasi portal musuh	Jarak antar Portal	Waktu Pembuatan
1	x=16, y=0	x=0, y=0	32	
1	x=21, y=0	x=0, y=0	55	1,1923 Detik
8	x=29, y=3	x=0, y=0	44	
8	x=29, y=0	x=0, y=17	46	1,3027 Detik
11	x=27, y=17	x=6, y=0	42	
11	x=0, y=17	x=0, y=5	66	1,2051 Detik
12	x=29, y=0	x=0, y=15	44	
12	x=27, y=0	x=0, y=15	46	1,3951 Detik
19	x=29, y=15	x=5, y=1	38	
19	x=16, y=0	x=0, y=13	33	
19	x=29, y=17	x=0, y=0	46	1,7590 Detik
22	x=24, y=0	x=0, y=13	41	
22	x=29, y=14	x=29, y=0	66	1,2465 Detik
23	x=14, y=0	x=0, y=17	33	
23	x=29, y=9	x=5, y=17	54	1,2849 Detik

Dari contoh Tabel 4 pada percobaan pertama, jarak portal musuh pertama dengan portal pemain adalah 32 ubin. Map ini tidak memenuhi persyaratan jarak antara portal musuh dengan portal pemain yaitu 45 ubin atau lebih. Dari 100 kali percobaan, terdapat 25 kali yang mendapatkan jarak antar portal yang tidak sesuai dengan syarat. Rata-rata waktu yang dibutuhkan untuk pembuatan map yang memenuhi persyaratan dari 75 kali pembuatan map adalah 0,869 detik. Sedangkan rata-rata waktu yang dibutuhkan jika map yang dihasilkan tidak sesuai dengan persyaratan untuk 25 map adalah 1,365 detik. Waktu pembuatan map awal membutuhkan waktu sekitar 0,8 detik dan proses untuk mengulang hanya membutuhkan waktu antara 0,3 hingga 0,6 detik. Total rata-rata waktu dari 100 percobaan adalah 0,99 detik.

G. Pengujian *Spawn* Musuh

Selain melakukan pengujian map, musuh yang *dispawn* untuk setiap ronde *game* juga diuji. Pada Tabel 5 merupakan sampel dari *wave* musuh yang dikeluarkan pada *game* tersebut. Pada contoh percobaan ke 1 dan ke 2, total monster kecil yang di keluarkan berbeda, pada *wave* 1, 2 dan 3 berbeda semua. Hasil ini diperoleh dari pembuatan *list* monster yang telat didesain sesuai dengan jumlah *health* setiap *wave*. Dengan mengikuti

aturan wave 1 memiliki total 8000 *health*, *wave* 2 memiliki total 12000 *health*, dan *wave* 3 sebanyak 20000 *health*. Urutan monster yang dikeluarkan dapat berbeda-beda sesuai dengan nilai *random* yang telah dimasukkan kedalam *list*, bisa saja muncul urut kecil semua, bergantian kecil-sedang-besar atau besar dahulu kemudian kecil.

Tabel 5. *Sample* percobaan *wave* musuh

Percobaan ke	Wave1 Kecil	Wave1 Sedang	Wave2 Kecil	Wave2 Sedang	Wave2 Besar	Wave3 Sedang	Wave3 Besar
1	30	25	10	17	19	36	32
2	28	26	24	14	17	34	33
3	26	27	8	16	20	32	34
4	26	27	14	13	20	34	33
5	38	21	10	19	18	38	31
6	26	27	14	17	18	26	37

Kesimpulan

Dari hasil desain dan pengujian, dapat diperoleh kesimpulan bahwa dari percobaan yang telah dilakukan dengan melakukan *generate* secara berulang sebanyak 100 kali dapat dilihat bahwa 25 kali map perlu dibuat ulang karena tidak memenuhi kriteria persyaratan map, tetapi waktu yang dibutuhkan untuk membuat ulang map hanya sebanyak 0,3 detik hingga 0,6 detik saja dengan rata-rata dari 100 kali percobaan adalah 0,99 detik. Hal ini dapat meningkatkan *replayability* pada *game tower defense* yang dapat menyediakan map dengan cepat dan berbeda-beda saat game dijalankan. Beberapa map memiliki lokasi portal pemain, lokasi portal musuh, dan jarak antar portal yang sama, tetapi dari tampilan map yang dihasilkan serta rute yang perlu dilalui oleh musuh berbeda-beda. Musuh yang dikeluarkan juga berbeda-beda sesuai dengan hasil *random* yang sudah dimasukkan ke dalam *list*. Dari percobaan ini dapat disimpulkan bahwa map yang telah dibuat menggunakan *Perlin Noise* telah dapat menghasilkan map yang berbeda pada setiap pembuatannya, dan urutan serta musuh yang di *spawn* juga berbeda untuk setiap map.

Referensi

Azis, H., dg. Mallongi, R., Lantara, D., & Salim, Y. (2018). Comparison of Floyd-Warshall Algorithm and Greedy Algorithm in Determining the Shortest Route. *2018 2nd East Indonesia Conference on Computer and Information Technology (EIConCIT)*, 294–298.

Du, Y., Li, J., Hou, X., Lu, H., Liu, S. C., Guo, X., Yang, K., & Tang, Q. (2019). *Automatic level Generation for Tower Defense Games*. IEEE.

Frank, E., & Olsson, N. (2017). *Procedural city generation using Perlin noise*. http://urn.kb.se/resolve?urn=urn:nbn:se:bth-14855

Liu, S., Chaoran, L., Yue, L., Heng, M., Xiao, H., Yiming, S., Licong, W., Ze, C., Xianghao, G., Hengtong, L., Yu, D., & Qinting, T. (2019, August 26). Automatic generation of tower defense levels using PCG. *ACM International Conference Proceeding Series*. https://doi.org/10.1145/3337722.3337723

Öhman, J. (2020). *Procedural Generation of Tower Defense levels*. http://www.ep.liu.se/.

Perlin, K. (1985). An Image Synthesizer. *SIGGRAPH Comput. Graph.*, *19*(3), 287–296. https://doi.org/10.1145/325165.325247

Risald, Mirino, A. E., & Suyoto. (2017). Best routes selection using Dijkstra and Floyd-Warshall algorithm. *2017 11th International Conference on Information & Communication Technology and System (ICTS)*, 155–158.

Risi, S., & Preuss, M. (2020). From Chess and Atari to StarCraft and Beyond: How Game AI is Driving the World of AI. *KI - Künstliche Intelligenz*, *34*(1), 7–17. https://doi.org/10.1007/s13218-020-00647-w

Stangl, R. (2017). *Procedural Content Generation: Techniques and Applications*.

Umar, R., Yudhana, A., & Prayudi, A. (2021). *ANALISIS PERBANDINGAN ALGORITMA DJIKSTRA, A-STAR, DAN FLOYD WARSHALL DALAM PENCARIAN RUTE TERDEKAT PADA OBJEK WISATA KABUPATEN DOMPU*. *8*(2), 227–234. https://doi.org/10.25126/jtiik.202182866

Wijaya, W., & Rahman, A. (2018). *Analisis Perbandingan Perlin Noise dan Simplex Noise Untuk Penciptaan Permukaan Daratan Pada Pembuatan Game*.

Yannakakis, G. N., & Togelius, J. (2018). Generating Content. In *Artificial Intelligence and Games* (pp. 151–202). Springer International Publishing. https://doi.org/10.1007/978-3-319-63519-4_4