Meningkatkan Kesulitan Serangan Enemy Dengan Menambahkan Influence Map Pada Metode A* Pada Procedural Generated Tower Defense Game

Michael Budiono, Liliana, Hans Juwiantho
Program Studi Informatika Fakultas Teknologi Industri Universitas Kristen Petra
Jl. Siwalankerto 121-131 Surabaya 60236
Telp. (031)-2983455, Fax. (031)-8417658
E-mail: C14170076@john.petra.ac.id, lilian@petra.ac.id, hans.juwiantho@petra.ac.id

ABSTRAK

Pada *tower defense game*, jika map dan penyerangan *enemy* tidak berubah maka strategi yang digunakan *player* akan tetap sama, hal ini membuat *game* tersebut memiliki *replay value* yang rendah dan membuat *player* berhenti memainkan *game* tersebut. Tower defense game yang sudah ada telah menggunakan *procedural generation* untuk membuat *level* yang berbeda pada setiap dimainkan, tetapi masih terdapat kekurangan dimana map terlihat polos dan memiliki pola yang sama setiap dimainkan, selain itu penyerangan *enemy* menggunakan A* memiliki pola yang simple dan tidak dapat mencari jalur yang menguntungkan untuk enemy sehingga enemy mudah dikalahkan dan game menjadi kurang menarik. Untuk mengatasi masalah tersebut digunakan *perlin noise* pada *procedural generation* agar map yang dibuat tidak terlihat polos dan tidak memiliki pola yang sama pada setiap dimainkan, selain itu digunakannya A* dengan ditambahkan dengan *influence map* pada penyerangan *enemy* agar penyerangan *enemy* dapat menjadi lebih menantang dan menarik untuk pemain.

Pada penelitian ini map dibuat menggunakan *perlin noise* untuk menentukan *terrain* pada *tile* yang ada pada map dan akan dilakukan pengecekan letak *start dan finish* menggunakan *floyd warshall algorithm* untuk menentukan apakah map perlu untuk dibuat ulang. Untuk penyerangan *enemy* digunakannya A* dengan ditambahkannya *influence map* untuk membuat *enemy* dapat memilih jalan yang lebih menguntungkan untuknya dengan berusaha menghindari jalan yang terhalang oleh *tower* dan jalan yang dapat diserang oleh *tower*.

Hasil pengujian menunjukan setelah dilakukannya *generate* secara berulang sebanyak 20 kali tidak ditemukannya map yang memiliki letak dan jarak antara *start* dan *finish* yang sama. Selain itu ditemukannya juga dengan pengimplementasian *procedural generation* dan *influence map* membuat game 25.7% lebih menantang jika dibandingkan dengan game yang tidak menggunakannya.

Kata Kunci: *Procedural Content Generation, Perlin Noise, Influence Map, Pathfinding, A**

ABSTRACT

In tower defense game, if the map dan enemy attack do not change, the strategy that will be used by the player will remain the same, this will make the game having a low replay value and will make the player stop playing the game. The existing tower defense game have used procedural generation to create different levels each time they are played, but there are still shortcoming where the map looks plain and has the same pattern every time it is played, other than that enemy attack using A* have a simple pattern and can't search for a profitable path for the enemy so the enemy become easy to defeat and the game become less interesting. To overcome this problem, perlin noise is used in procedural generation so that the map mode does not look plain and does not have the same pattern every time it is played, other than that enemy attack use A* by adding influence map so that enemy attack can be more challenging and interesting to the player.

In this thesis, the map was created using perlin noise to determine the terrain of the tile on the map and the location of start and finish will be checked using floyd warshall algorithm to determine if the map need to be remade. For enemy attack, A* is used with the addition of influence map to make the enemy can choose a path that is more profitable for it by avoiding roads blocked by tower and roads that can be attacked by towers.

The test results show that after the map is generated repeatedly for 20 times, no map has the same location and distance between start and finish. In addition, it is also found that the implementation of procedural generation and influence map made the game 25.7% more challenging when compared to games that did not use it.

Keywords: Procedural Content Generation, Perlin Noise, Influence Map, Pathfinding, A*

1. PENDAHULUAN

Pada *game tower defense* dengan map dan penyerangan *enemy* yang berbeda akan membuat *player* menggunakan strategi yang berbeda yang cocok untuk map dan penyerangan *enemy* tersebut. Jika map dan penyerangan *enemy* tidak berubah maka strategi yang akan digunakan oleh *player* akan tetap sama dan hal ini akan mengurangi *replay value* dan akan membuat *player* berhenti memainkan game tersebut. Tentunya jika penyerangan *enemy* sangat simple seperti jika *enemy* hanya berjalan lurus saja maka map tersebut akan menjadi membosankan dan kurang menarik karena kurangnya tantangan yang akan dihadapi *player* dimana peletakan *tower* akan menjadi tidak berpengaruh. Hal hal tersebut akan membuat *player* merasa enggan untuk memainkan *game* tersebut. Oleh karena itu *game tower defense* memerlukan desain map, dan *enemy* dengan penyerangan yang menarik dan beragam untuk dapat menarik perhatian *player* mereka. Pada penelitian sebelumnya sudah diterapkan *procedural content generation* untuk membuat map dan *enemy generation* pada game yang akan selalu berubah pada setiap dimainkan tetapi masih dapat mengatur kesulitan pada game dengan mengubah *variable* yang digunakan [7][4]. Penggunaan *procedural content generation* ini dilakukan agar strategi yang digunakan oleh pemain perlu untuk terus berubah pada setiap permainan untuk menyesuaikan dengan *level* yang akan dihadapi sehingga pemain tidak merasa bosan karena merasa *level*

yang dihadapi selalu sama. Tetapi *procedural content generation* untuk *tower defense game* masih memiliki penyerangan *enemy* yang simple yaitu hanya mengikuti *path* yang ada dan tidak dapat menyesuaikan dengan perubahan yang terjadi pada map sehingga penyerangan *enemy* masih terlihat simple dan kurang menarik karena kurangnya tantangan yang dihadapi oleh *player*.

Pada penelitian sebelumnya enemy akan menyerang *player* dengan cara mengikuti sebuah *path*, dan *path* penyerangan *enemy* dibuat dengan cara memilih titik secara acak didekat pusat map dan menambahkan *path* secara acak ke dua ujung map diarah yang berbeda, jika *path* penyerangan lebih dari satu maka mereka akan memilih titik acak di tengah *path penyerangan* awal yang telah dibuat pertama dan ditambahkan *path* penyerangan dari titik tersebut ke ujung map lain yang belum memiliki *path* penyerangan *enemy*. Sedangkan untuk *enemy generation*, mereka melakukan *spawn* tidak per *unit* tetapi per kelompok *enemy*, kelompok kelompok tersebut diambil dari *game* lain. Setelah mereka mengambil kelompok *enemy* tersebut *wave* penyerangan akan dibuat dengan menggunakan *genetic representation* dimana pada tiap *wave* akan dibuatnya *fixed length chromosome* yang berisikan kelompok *enemy* yang telah diambil yang akan dimasukan pada *fitness function* untuk tiap *wave* penyerangan [7]. Karena cara ini map dan *path* penyerangan yang mereka buat tidak terasa benar benar *random* dan akan terbuatnya pola dimana *path* penyerangan pada map akan berbentuk seperti tanda tambah, dan karena mereka menggunakan *path* untuk penyerangan *enemy* mereka, penyerangan *enemy* akan tetap sama jika berada dalam *level* yang sama dan tidak dapat beradaptasi dengan penarukan *tower* oleh *player*.

Pada penelitian berikutnya map *tower defense* dibuat dengan menggunakan *perlin noise* dan *enemy* akan menyerang dengan mengikuti *path* penyerangan *enemy* yang dibuat dengan cara mengambil satu titik *random* di map untuk menjadi *start* lalu *path* akan dibuat dimulai dari titik *start* tersebut, setelah itu akan dibuat *path* berikutnya pada ujung *path* terahkir, hal ini akan diulang sebanyak jumlah *path point count* yang telah ditentukan. Selain itu juga diberlakukannya beberapa *function* yang akan membantu pembuatan *path* seperti untuk menghindari *collision* dengan *path* lain yang telah dibikin. Setelah itu map yang dibuat akan dimasukan pada *fitness function* untuk mempertimbangkan apakah map perlu untuk di *generate* ulang. Untuk *enemy generation*, mereka membuat *wave enemy* dimana jumlah *enemy wave size* dan jumlah *enemy* merupakan angka *random* antara 1 dan 5, *wave* tersebut akan dispawn sejumlah X kali dimana X merupakan hasil perhitungan dari *path density* dan *difficulty* [4]. Pada penelitian tersebut, musuh hanya menyerang dengan mengikuti *path* saja sehingga kurang menarik dan *enemy generatio*n yang dibuat akan membuat setiap *wave* tidak bertambah susah dan tidak terlalu beda hanya jumlah *wave* tiap *level* yang berbeda

Pada penelitian berikutnya map dari *game tower defense* tidak dibuat secara otomatis menggunakan *procedural map generation*, tetapi dibuat manual oleh *user* setiap permainan akan dimainkan dengan *user* memilih titik *start*, *finish*, dan meletakan *obstacles* sehingga map yang digunakan untuk permainan akan berubah ubah. Setelah map selesai dibuat dan permainan dimulai maka enemy akan dispawn dan akan menggunakan A* *algortihm* untuk mencari *path* penyerangan terbaik untuk mencapai titik *finish* [1]. Pada penelitian tersebut, penyerangan enemy dapat menyesuaikan dengan map yang dibuat, tetapi penyerangan hanya akan menyesuaikan dengan map dan peletakan tower tidak akan mempengaruhi penyerangan enemy karena tidak adanya influence map sehingga penyerangan enemy kurang menarik.

Pada skripsi ini akan dibuatnya *game tower defense* dengan *procedural generation* dimana konten dari *game* seperti map dan *enemy wave* dibuat secara otomatis. Pembuatan *terrain* map akan mengimplementasikan *perlin noise* untuk membuat *terrain* yang terlihat unik dan natural. Sedangkan untuk pembuatan *wave* pada *enemy generation* akan dibuat secara random dengan diberikannya pembatasan pada jumlah enemy dan jumlah *total hit point* yang berbeda pada setiap wave, selain itu setiap macam dan ukuran enemy akan memiliki kemungkinan yang berbeda untuk muncul pada setiap wave. Sedangkan untuk penyerangan *enemy* akan diterapkannya A* *algorithm* dengan *influence map* dimana penyerangan *enemy* yang unik dan dapat menyesuaikan diri dengan map yang telah dibuat dan juga peletakan *tower* oleh *player* agar *enemy* dapat menjadi lebih menantang untuk player. Setelah itu akan dilihat keberhasilan dari *game* tersebut dengan melakukan percobaan memainkan *game* tersebut untuk melihat apakah setiap *level* yang digenerate akan terasa berbeda dan apakah *enemy* menggunakan A* dengan *influence map* dapat lebih menantang dari *enemy* yang menggunakan A* tanpa *influence map*.

2. LANDASAN TEORI

2.1 Tower Defense Game

Tower defense game adalah sebuah *subgenre* dari *game* strategi yang berfokus pada pengalokasian *resource* dan penempatan *unit (tower)*. Pada dasarnya *tower defense game* berisikan *player* membeli dan mengorganisasi *unit (tower)* yang akan menyerang musuh yang berada di sekitar *unit (tower)* tersebut untuk menghalangi musuh mencapai tujuan tertentu. *Player* akan menang jika sudah menghancurkan sejumlah musuh dan akan kalah jika sejumlah musuh sudah mencapai tujuannya. Pada *tower defense game*, map dapat dibedakan menjadi beberapa macam yaitu *linear path*, dan *branching path* dimana penyerangan *enemy* akan dilakukan dengan mengikuti *path* yang sudah ada, dan juga *free form path* dimana penyerangan *enemy* akan dilakukan dengan *enemy* mencari jalan ke titik *finish* sesuai dengan map yang sedang dimainkan [2].

2.2 Procedural Content Generation

Procedural Content Generator adalah cara pembuatan konten secara otomatis dengan menggunakan algoritma. Dengan menggunakan *procedural content generator* maka akan didapatkan beberapa kelebihan: mengurangi beban kerja dalam pembuatan game, mengurangi biaya yang dibutuhkan untuk membuat game, mengurangi storage space yang digunakan, dan memperbanyak keanekaragaman konten sehingga menaikan *replay value*[8].

2.3 Procedural Content Generation

Perlin noise adalah salah satu type dari *gradient noise* yang dikenalkan oleh Ken Perlin pada tahun 1985. *Gradient noise* adalah tipe *noise* dimana sebuah *sample* yang diambil akan menghasilkan nilai yang mirip dengan nilai nilai disekitarnya. Karena hal ini hasil dari *perlin noise* tidaklah benar benar *random* tapi *psuedorandom*. *Perlin noise* ini biasa digunakan untuk menghasilkan *surface* yang terlihat alami [4].

2.4 Floyd Warshall Algorithm

Floyd warshall algorithm adalah *algorithm* yang digunakan untuk mencari *path* terpendek dari antara semua *vertices* pada *graph*. *Algorithm* ini bekerja dengan cara mengecek apakah ada path dari kedua *vertices* melalui sebuah *vertices* lain yang dapat menghasilkan *cost* yang lebih kecil dari *cost* yang sekarang telah ada [6].

2.5 A* Algorithm

A* adalah sebuah generic search algorithm yang digunakan untuk menyelesaikan banyak masalah dimana salah satunya adalah pathfinding. Untuk mencari path, A* akan menggunakan evaluation function F(n) untuk memilih node yang akan diambil berikutnya .Struktur dari rumus F(n) adalah [3]:

$$F(n) = G(n) + H(n) \quad (1)$$

Dimana:
-G(n) adalah *cost* terendah untuk mencapai ke n dari *start*
-H(n) adalah estimasi *cost* dari n ke *goal*

2.6 Influence Map

Influence map merupakan teknik yang digunakan oleh sebuah *agents* untuk membantu dalam membuat keputusan. *Influence map* tidak memberikan instruksi tetapi hanya menyediakan informasi yang berguna untuk *agents* dalam membantu pengambilan keputusan [5].

3. DESAIN SISTEM

3.1 Analisa Permasalahan

Tower defense game masih memiliki beberapa masalah, salah satu dari masalah tersebut adalah pembuatan map menggunakan *procedural content generation* yang memiliki bentuk yang polos dimana map yang dibuat hanya berisikan sebuah jalur dan dataran yang menjadi background. Selain bentuk yang polos, juga terbentuknya sebuah pola yang sama pada map setiap kali map dibuat. Hal ini membuat map yang dibuat kurang terasa berbeda pada setiap dimainkan. Masalah lain yang dapat ditemukan adalah penyerangan *enemy* yang kurang menarik dimana *enemy* hanya akan mengikuti satu jalur yang telah ada pada map. Hal ini membuat kurangnya tantangan yang dihadapi pemain dan akan membuat game kurang menarik.

Untuk mengatasi masalah yang ada pada pembuatan map menggunakan *procedural content generation* maka akan digunakannya metode *perlin noise*. Dengan menggunakan *perlin noise*, akan terhindarnya terbentuknya pola pada map dan map dapat memiliki berbagai macam *terrain* dimana setiap *terrain* dapat memiliki kegunaan yang berbeda sehingga map tidak terlihat polos. Hal ini akan membuat map yang dibuat dapat terasa berbeda pada setiap dimainkan.

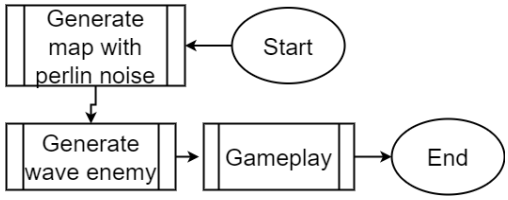
Untuk mengatasi masalah penyerangan *enemy* yang kurang menarik maka akan digunakannya A* dengan *influence map* dimana penyerangan *enemy* tidak akan hanya mengikuti sebuah jalur yang telah dibuat, tetapi musuh juga dapat mencari jalan yang lebih menguntungan bagi musuh. Hal ini akan membuat penyerangan *enemy* yang lebih menantang dan lebih menarik untuk pemain.

3.2 Desain Game

Game yang akan dibuat adalah sebuah *tower defense game*, dimana pemain memiliki tujuan untuk melindungi suatu daerah agar tidak dicapai oleh serangan musuh. Untuk melindungi daerah tersebut, pemain dapat membangun suatu bangunan yang akan menyerang musuh secara otomatis. Serangan musuh pada permainan ini akan dibagi menjadi 4 gelombang serangan dan pemain akan menang jika berhasil bertahan hingga serangan musuh pada gelombang keempat selesai. Pada game ini pemain akan memiliki 5 *life* yang akan berkurang sebanyak 1 untuk setiap musuh yang berhasil mencapai daerah yang dilindungi pemain dan pemain akan kalah jika *life* berkurang menjadi 0.

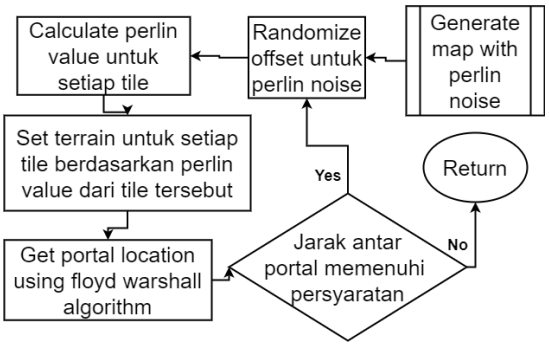
3.3 Desain Sistem

Dalam pembuatan game ini, garis besar sistem yang akan dibuat dapat dilihat pada Gambar 1. Aplikasi akan dimulai dengan proses pembuatan map menggunakan perlin noise, berikutnya dibuatnya wave enemy, dan pada akhirnya dimulainya gameplay.



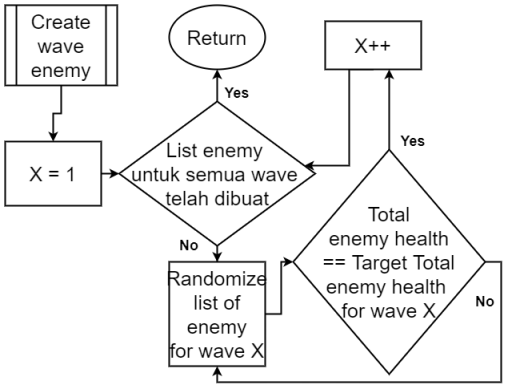
Gambar 1. Flowchart alur game

Pembuatan map dengan *perlin noise* ini dilakukan dengan menggunakan hasil *perlin noise* untuk menentukan *terrain* dari suatu *tile*. Setelah semua *tile* telah ditentukan terrainnya maka akan dilakukan pengecekan lokasi *portal* pada map menggunakan *floyd warshall algorithm* dengan cara mencari 2 *tile* terhubung yang memiliki jarak terjauh. Dari lokasi *portal* ini akan dilakukan pengecekan apakah jarak antar *portal* telah memenuhi persyaratan yang telah ditentukan. Jika map yang telah dibuat tidak memenuhi persyaratan jarak antar portal, maka map tersebut akan dibuat ulang. Alur jalannya proses ini dapat dilihat pada Gambar 2.



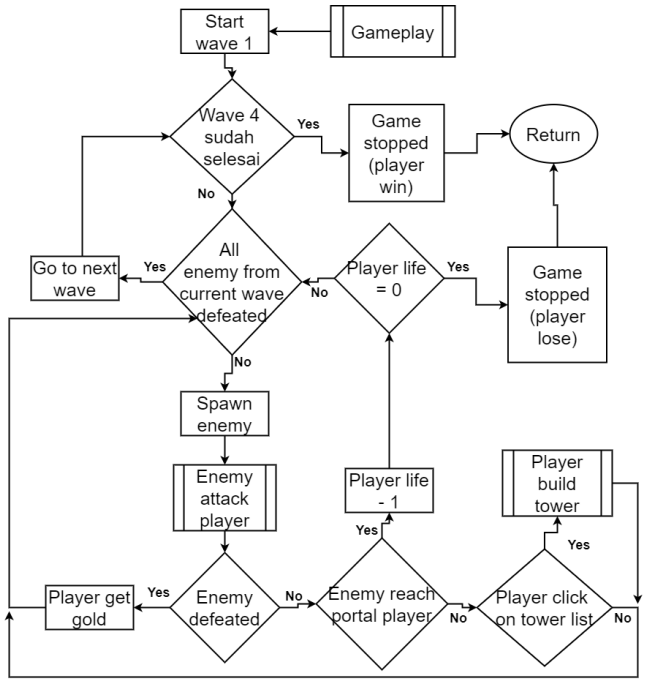
Gambar 2. Alur generate map with perlin noise

Pembuatan list wave enemy digunakan untuk menentukan enemy yang akan dilawan oleh pemain saat *gameplay*. List wave enemy akan dibuat dengan cara melakukan random pada enemy yang akan muncul. Setelah list wave enemy selesai dibuat maka akan dilakukan pengecekan terhadap total *health* dari enemy pada list tersebut dan jika total health tidak sesuai dengan total *health* yang telah ditentukan maka list wave enemy akan dibuat ulang. Setelah list wave enemy selesai dibuat maka akan dimulainya *gameplay*. Alur dari pembuatan wave enemy dapat dilihat pada Gambar 3.



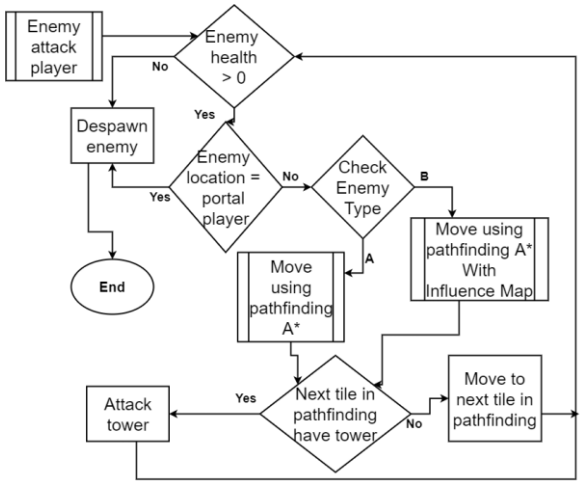
Gambar 3. Alur pembuatan wave enemy

Pada saat Gameplay berlangsung, musuh akan muncul pada lokasi *portal enemy* sesuai dengan list wave enemy. Jika semua musuh pada suatu gelombang telah dikalahkan maka permainan akan berlanjut ke gelombang berikutnya hingga gelombang serangan musuh keempat selesai. Proses permainan akan berakhir jika pemain berhasil bertahan hingga serangan musuh pada gelombang serangan keempat selesai atau *life* berkurang hingga menjadi 0. Alur jalannya gameplay dapat dilihat pada Gambar 4.



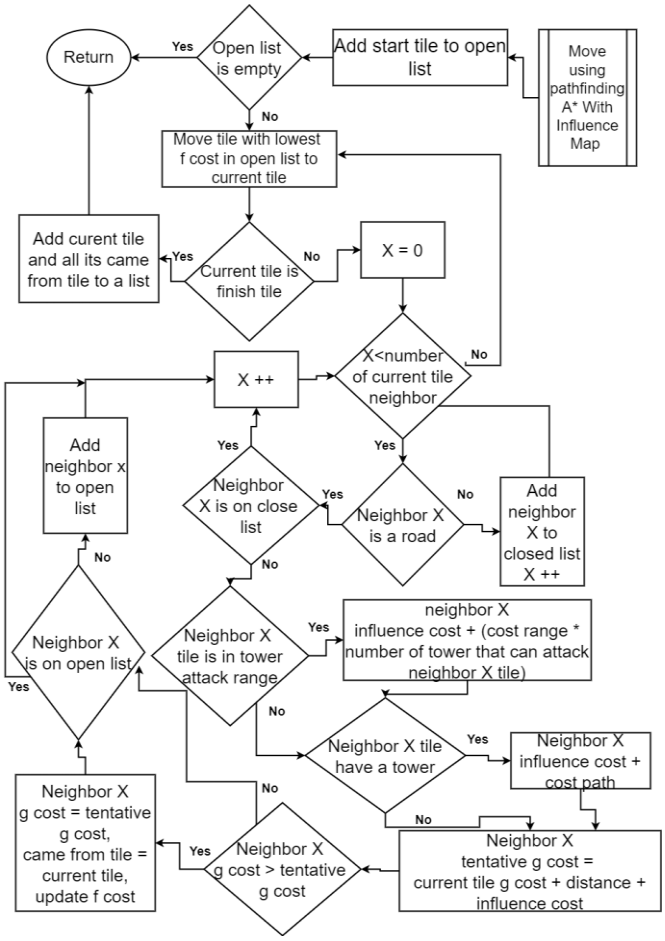
Gambar 4. Alur gameplay

Saat permainan berlangsung, musuh yang telah muncul akan berusaha mencapai *portal player* dan akan menyerang *tower* yang menghalangi jalannya. Untuk menentukan jalan yang akan diambil, musuh dengan tipe A akan mencari jalur terpendek menggunakan *pathfinding* A* sedangkan musuh dengan tipe B akan mencari jalur terpendek dengan memperhitungkan kondisi dari map menggunakan *pathfinding* A* dengan ditambahkannya *influence map*. Jika musuh telah mencapai *portal player* atau *health* dari musuh telah berkurang hingga 0 maka musuh akan menghilang. Jalannya proses penyerangan enemy dapat dilihat pada Gambar 5.



Gambar 5. Alur penyerangan enemy

Perbedaan A* tanpa *influence map* dan dengan influence map adalah pada A* tanpa *influence cost* saat menghitung *tentative g cost* hanya akan menggunakan *current tile g cost* ditambahkan dengan jarak antar *tile*, tetapi pada A* yang ditambahkan *influence map*, maka akan ditambahkan lagi *influence cost* dimana *influence cost* ini adalah *cost* tambahan yang bergantung pada kondisi permainan. Pada game ini *influence cost* yang digunakan adalah *cost range* (cost jumlah tower yang dapat menyerang tile yang akan dilewati), dan *cost path* (cost melewati *tile* dengan *tower)* dengan *cost path* memiliki prioritas yang lebih tinggi dari *cost range.* Dengan penambahan *influence cost,* musuh akan mencari jalur terpendek dengan menghindari tile yang terdapat tower atau dapat terkena serangan dari tower. Alur proses pathfinding A* dengan influence map dapat dilihat pada Gambar 6.



Gambar 6. Alur pathfinding A* dengan influence map

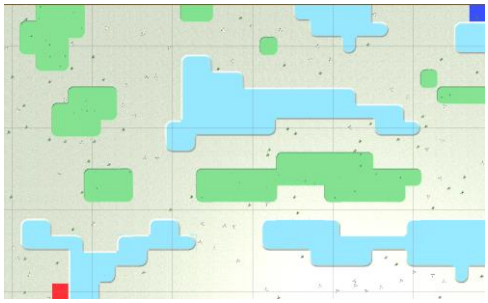
4. PENGUJIAN SISTEM

4.1 Pengujian Pembuatan Map

Pengujian ini dilakukan untuk memastikan map yang dibuat akan berbeda pada setiap *generate* dan telah memenuhi persyaratan yang telah ditetapkan. Pada pengecekan ini map akan dikatakan sama jika map memiliki lokasi *portal player*, lokasi *portal enemy*, jumlah *portal enemy* dan jarak antar *portal* yang sama dengan map lainnya. Pengujian ini dilakukan dengan cara dilakukannya *generate* map secara berulang sebanyak 20 kali. Map yang di *generate* menggunakan *perlin noise* dengan parameter *size* 30 x 18, dan *scale* 4, sedangkan untuk *offset* akan didapat secara *random*. Untuk persyaratan yang telah ditetapkan, map akan di *generate* ulang jika

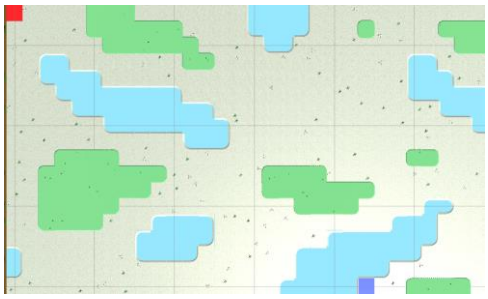
jarak dari *portal enemy* menuju ke *portal player* tidak mencapai jarak 45 *tile* atau jarak antar *portal enemy* tidak mencapai 25 *tile*.

Contoh map pada percobaan ke-1 dapat dilihat pada Gambar 7. Pada map tersebut terdapat 1 portal enemy dimana lokasi *portal player* pada grid adalah x=29, y=17, dan lokasi *portal enemy* pada grid adalah x=3, y=0. Dari lokasi portal tersebut didapat jarak *portal enemy* pertama dengan *portal player* adalah 51 *tile.*



Gambar 7. Contoh hasil map 1

Contoh map pada percobaan ke-2 dapat dilihat pada Gambar 8. Pada map tersebut terdapat 1 portal enemy dimana lokasi *portal player* pada grid adalah x=22, y=17, dan lokasi *portal enemy* pertama pada grid adalah x=3, y=0. Dari lokasi portal tersebut didapat jarak *portal enemy* pertama dengan *portal player* adalah 51 *tile.*



Gambar 8. Contoh hasil map 2

Dapat dilihat dari contoh diatas bahwa kedua map itu berbeda. Kedua map memiliki jarak antar portal enemy dan portal player yang sama, tetapi lokasi portal player dan portal enemy pada kedua map berbeda sehingga kedua map tersebut dikatakan berbeda.

Dari contoh pengecekan tersebut didapat 20 map yang di *generate* untuk pengujian ini tidak ada yang memiliki lokasi *portal player*, lokasi *portal enemy*, jumlah *portal enemy* dan jarak antar *portal* yang sama dengan map lainnya. Oleh karena itu semua map pada tabel tersebut adalah map yang berbeda.

4.2 Pengujian Pembuatan Wave Enemy

Pengujian hasil pembuatan daftar *wave enemy* dilakukan untuk memastikan daftar *wave enemy* yang dibuat telah berbeda pada setiap berjalannya permainan dan juga telah memenuhi persyaratan dari *total health* yang telah ditentukan. Pengujian dilakukan dengan menjalankan permainan dan melihat hasil dari daftar *wave enemy* yang telah dibuat untuk setiap *wave*. Daftar *wave enemy* ini menentukan jumlah *enemy*, dan *enemy* apa saja yang akan muncul. Contoh dari hasil pembuatan daftar *wave enemy* dapat dilihat pada Tabel 1.

Tabel 1. Hasil pembuatan wave enemy

Nomor percobaan	Wave	Jumlah total enemy	Jumlah enemy A Small	Jumlah enemy A Medium	Jumlah enemy A Big	Jumlah enemy B Small	Jumlah enemy B Medium	Total health enemy
1	1	50	25	25	0	0	0	7500
	2	92	32	33	0	27	0	12500
	3	166	34	48	0	48	36	25000
	4	242	53	50	55	41	43	50000
2	1	50	25	25	0	0	0	7500
	2	100	41	25	0	34	0	12500
	3	170	43	37	0	47	43	25000
	4	256	60	39	51	54	52	50000

4.3 Pengujian Gameplay

Pengujian *gameplay* dilakukan untuk memastikan permainan dapat berjalan dengan lanjar. Pengujian ini dilakukan dengan menjalankan permainan dan melihat apakah semua fitur dalam permainan dapat berjalan.

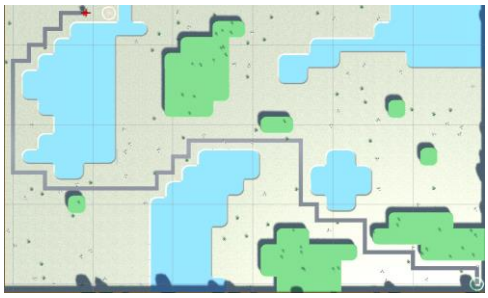
Pada saat permainan dimulai, pemain akan diperlihatkan *user interface* seperti pada Gambar 9. Pada *user interface* ini pemain dapat melihat informasi tentang jumlah *life* yang dimiliki pemain, gelombang musuh (*wave*) yang sedang dihadapi oleh pemain, jumlah *gold* yang dimiliki oleh pemain, dan jumlah *enemy* pada gelombang musuh (*wave*) yang sedang berlangsung pada sebelah kiri atas layar. Selain itu pemain juga dapat melihat pilihan *tower* yang dapat dibangun pada bagian kanan bawah layar.

Gambar 9. Tampilan gameplay

4.4 Pengujian Penyerangan Enemy

Pengujian penyerangan *enemy* dilakukan untuk memastikan proses penyerangan *enemy* dapat berjalan dengan baik. Pengujian ini dilakukan dengan mengamati pergerakan *enemy* pada saat *gameplay* berlangsung.

Saat *enemy* di *spawn* maka *enemy* akan mencari jalan terpendek untuk mencapai *portal player*. Untuk *enemy* yang memiliki tipe A, pencarian jalan ini akan dilakukan tanpa memperhitungkan lokasi dari *tower* sehingga walau jalan yang akan dilewati terhalang oleh *tower*, *enemy* dengan tipe A akan tetap memilih jalan terpendek untuk menuju *portal player*. Untuk *enemy* yang memiliki tipe B, pencarian jalan ini akan dilakukan dengan memperhitungkan lokasi dari *tower* sehingga *enemy* dengan tipe B akan memilih jalan yang tidak terhalangi oleh tower dan tidak terkena serangan dari tower jika masih dimungkinkan. Salah satu contoh dari jalan terpendek yang akan digunakan oleh *enemy* untuk mencapai *portal player* dapat dilihat pada Gambar 10.



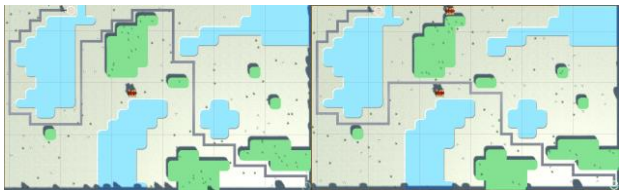
Gambar 10. Gambaran jalan yang diambil enemy

4.5 Pengujian A* Influence Map

Pengujian ini dilakukan untuk memastikan jalan yang diambil oleh *enemy* dengan tipe B sudah dapat memprioritaskan jalan yang tidak terkena serangan *tower* dan dihalangi *tower*. Contoh pergantian jalan dikarenakan *inlufence map* dapat dilihat pada Gambar 11. Pada gambar pertama *enemy* dengan tipe B akan memilih jalur terpendek dan pada gambar kedua *enemy* akan memilih jalur lain setelah jalur yang dapat terkena serangan tower. Hal ini terjadi karena setelah diletakannya tower pada tile yang seharusnya dilewati, GCost pada tile tersebut akan ditambahkan dengan *influence cost* sehingga FCost pada tile tersebut akan naik dan membuat A* memprioritaskan tile lain yang memiliki FCost lebih kecil.

Gambar 11. Perbandingan jalan pada enemy tipe B (1)

Pada contoh berikutnya dapat dilihat pada Gambar 12. Pada gambar pertama, enemy akan menggunakan jalan yang tidak terkena serangan tower karena jalan tersebut memiliki FCost yang lebih rendah dari jalan yang terkena serangan tower, tetapi pada gambar kedua diletakan tower pada jalan yang seharusnya diambil tersebut. Setelah diletakkannya tower, enemy akan memilih jalan yang dapat terkena serangan tower tetapi tidak jalannya tidak terhalang oleh tower, hal ini terjadi karena *Influence cost* yang ditambahkan pada tile yang terkena serangan tower lebih kecil dari *influence cost* yang ditambahkan pada tile yang terdapat tower. Karena tile yang terdapat tower ditambahkan *influence cost* yang lebih tinggi dari tile yang dapat terkena serangan tower, maka A* akan memprioritaskan tile yang terkena serangan tower daripada tile yang terhalang oleh tower.



Gambar 12. Perbandingan jalan pada enemy tipe B (2)

4.6 Pengujian Hasil Kuisioner

Pengujian kuisioner dilakukan untuk melihat pendapat dari pemain terhadap game yang telah dibuat. Untuk pengujian kuisioner akan dilakukan melalui pengambilan kuisioner dari 10 orang dimana semua dari mereka adalah laki-laki dan pernah bermain game dengan genre tower defense. Dari 10 orang tersebut 8 memiliki umur 19-20 tahun, dan 2 memiliki umur 22 dan keatas. Untuk mengisi kuisioner ini semua pemain akan diminta memainkan game yang telah dibuat dan juga game serupa yang tidak menggunakan *procedural content generation* dan tidak menggunakan *influence map* masing masing sebanyak 2 kali. Pertanyaan pada kuisioner ini akan ditanyakan untuk kedua game yang telah dimainkan, pertanyaan tersebut adalah sebagai berikut:

1. Berapa sisa life yang anda miliki saat permainan berakhir?
2. Seberapa manarik kah enemy pada game ini?
3. Seberapa susah untuk dikalahkan kan enemy pada game ini?
4. Jika dilihat dari sisi map, seberapa menarik kah game ini?

Dari kuisioner ini, pertanyaan pertama digunakan untuk melihat seberapa besar perbedaan kesulitan yang dihadapi pemain saat memainkan tower defense game yang menggunakan influence map dan procedural generation jika dibandingkan dengan tower defense game yang tidak menggunakan influence map dan procedural generation. Dengan semakin banyak berkurangnya life dari pemain maka dapat dilihat seberapa sulit permainan tersebut. Untuk pertanyaan kedua digunakan untuk melihat seberapa besar perbedaan kemenarikan enemy yang dihadapi pemain. Untuk pertanyaan ketiga digunakan untuk melihat seberapa besar perbedaan kesulitan enemy yang akan dihadapi oleh pemain. Sedangkan untuk pertanyaan keempat digunakan untuk melihat seberapa besar perbedaan kemenarikan map.

Dari jawaban dari pertanyaan pertama akan dilihat total sisa health dari percobaan pada tower defense game yang tidak menggunakan influence map dan procedural generation, maupun tower defense game yang menggunakan influence map dan procedural generation. Hasil dari total sisa health dapat dilihat pada Tabel 2.

Tabel 2. Total sisa life

Total health pada percobaan ke	Tower defense game yang tidak menggunakan influence map dan procedural generation	Tower defense game yang menggunakan influence map dan procedural generation
Pertama	34	23
Kedua	32	26

Dari hasil pada tabel 2 dapat dilihat bahwa tower defense game yang menggunakan influence map dan procedural generation memiliki total sisa life yang lebih sedikit, yaitu dengan total 49 life sedangkan tower defense game yang tidak menggunakan influence map dan procedural generation memiliki total 66 life. Dari hasil tersebut dapat disimpulkan bahwa tower defense game yang menggunakan influence map dan procedural generation akan lebih susah untuk dihadapi daripada tower defense game yang tidak menggunakan influence map dan procedural generation karena pemain kehilangan life lebih banyak sebesar 25.7% .

Hasil dari jawaban pertanyaan kedua akan diubah menjadi point. Untuk setiap responden yang menjawab sangat tidak menarik maka akan diberikan 1 point, untuk setiap responden yang menjawab tidak menarik maka akan diberikan 2 point, untuk setiap responden yang menjawab biasa saja maka akan diberikan 3 point, untuk setiap responden yang menjawab menarik maka akan diberikan 4 point, untuk setiap responden yang menjawab sangat menarik maka akan diberikan 5 point. Dari pengambilan point ini didapat tower defense game yang tidak menggunakan influence map dan procedural generation memiliki 31 point sedangkan tower defense

game yang menggunakan influence map dan procedural generation memiliki 41 point. Dari point tersebut dapat disimpulkan bahwa pemain berpendapat enemy lebih menarik sebesar 24.3% pada saat diterapkannya influence map pada enemy.

Hasil dari jawaban pertanyaan ketiga akan diubah menjadi point. Untuk setiap responden yang menjawab sangat mudah untuk dikalahkan maka akan diberikan 1 point, untuk setiap responden yang menjawab mudah untuk dikalahkan maka akan diberikan 2 point, untuk setiap responden yang menjawab biasa saja maka akan diberikan 3 point, untuk setiap responden yang menjawab susah untuk dikalahkan maka akan diberikan 4 point, untuk setiap responden yang menjawab sangat susah untuk dikalahkan maka akan diberikan 5 point. Dari pengambilan point ini didapat tower defense game yang tidak menggunakan influence map dan procedural generation memiliki 23 point sedangkan tower defense game yang menggunakan influence map dan procedural generation memiliki 39 point. Dari point tersebut dapat disimpulkan bahwa dengan mengimplementasikan influence map maka enemy akan menjadi lebih susah untuk dikalahkan oleh pemain sebesar 41% jika dibandingkan dengan enemy yang tidak mengimplementasikan influence map.

Hasil dari jawaban pertanyaan keempat akan diubah menjadi point. Untuk setiap responden yang menjawab sangat tidak menarik maka akan diberikan 1 point, untuk setiap responden yang menjawab tidak menarik maka akan diberikan 2 point, untuk setiap responden yang menjawab biasa saja maka akan diberikan 3 point, untuk setiap responden yang menjawab menarik maka akan diberikan 4 point, untuk setiap responden yang menjawab sangat menarik maka akan diberikan 5 point. Dari pengambilan point ini didapat tower defense game yang tidak menggunakan influence map dan procedural generation memiliki 33 point sedangkan tower defense game yang menggunakan influence map dan procedural generation memiliki 44 point. Dari point tersebut dapat disimpulkan bahwa map pada tower defense game menggunakan procedural generation lebih menarik sebesar 25% jika dibandingkan dengan tower defense game yang tidak menggunakan procedural generation.

5. KESIMPULAN

- Dari percobaan yang telah dilakukan dengan melakukan *generate* secara berulang sebanyak 20 kali dapat dilihat bahwa setiap map memiliki lokasi *portal player*, lokasi *portal enemy*, jumlah *portal enemy* dan jarak antar *portal* yang berbeda dari map lainnya dimana tidak adanya map yang memiliki semua lokasi dan jarak dari portal yang sama. Dari percobaan ini dapat disimpulkan bahwa map yang telah dibuat menggunakan *perlin noise* telah dapat menghasilkan map yang berbeda pada setiap *generate* nya.
- Tanpa digunakannya pengecekan lokasi *portal* menggunakan *floydwarshall algorithm* map akan memiliki lokasi *portal* yang berdekatan sehingga kualitas dari map yang dibikin akan menjadi lebih buruk.
- Dari kuisioner yang telah dilakukan dengan diambilnya total sisa life yang dimiliki pemain saat permainan berakhir, dapat disimpulkan bahwa dengan penambahan influence map dan procedural generation pada tower defense game akan membuat game menjadi susah untuk dihadapi oleh player dimana pemain kehilangan life lebih banyak sebesar 25.7% jika dibandingkan saat memainkan tower defense game yang tidak menggunakan influence map dan procedural generation.
- Dari kuisioner yang telah dilakukan, pemain merasa bahwa enemy yang menggunakan influence map lebih menarik sebesar 24.3% jika dibandingkan dengan enemy yang tidak menggunakan influence map, dan enemy yang menggunakan influence map lebih susah untuk dikalahkan sebesar 41% jika dibandingkan dengan enemy yang tidak menggunakan influence map.
- Dari kuisioner yang telah dilakukan, pemain merasa bahwa map yang dibuat menggunakan procedural generation lebih menarik sebesar 25% jika dibandingkan dengan map yang tidak menggunakan procedural generation.

6. REFERENSI

[1] Bogdan-Mihai Păduraru, A. I. 2017. Tower Defense with Augmented Reality. *Proceedings of the 14th Conference on Human Computer Interaction-RoCHI*, 113-118.

[2] Julian Togelius, E. A. 2011. Computational Intelligence and Tower Defence Games. *2011 IEEE Congress of Evolutionary Computation (CEC)*, 1-8. DOI= 10.1109/CEC.2011.5949738.

[3] Mathew, G. E. 2015. Direction Based Heuristic For Pathfinding In Video Games. *Procedia Computer Science*, *47*. DOI= https://doi.org/10.1016/j.procs.2015.03.206.

[4] Öhman, J. 2020. Procedural Generation of Tower Defense levels. (Bachelor Thesis, Linköping University, Department of Computer and Information Science). URI= http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1442180&dswid=-6945

[5] Rabin, S. 2020. *Game AI Pro 360: Guide to Tactics and Strategy*. CRC Press.

[6] Result, 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)*, *11*, 155-158. DOI= 10.1109/ICTS.2017.8265662.

[7] Simon Liu, L. C. 2019. Automatic generation of tower defense levels using PCG. *Proceedings of the 14th International Conference on the Foundations of Digital Games*, 1-9. DOI= https://doi.org/10.1145/3337722.3337723.

[8] Summerville, A., Snodgrass, S., & Guzdial, M. 2018. Procedural Content Generation via Machine Learning (PCGML). *IEEE Transactions on Games*, *10*, 257 - 270. DOI= 10.1109/TG.2018.284663.