REPOSITOR, Vol. 2, No. 7, Juli 2020, Pp. 965-976
ISSN : 2714-7975
E-ISSN : 2716-1382

Island Generator pada Game Open world Menggunakan Algoritma Perlin noise

M. Naufal Azzmi H.[*1], Lailatul Husniah[2], Ali Sofyan Kholimi[3]
[1,2,3]Teknik Informatika/Universitas Muhammadiyah Malang
naufalazzmi@webmail.umm.ac.id[1], huniah@umm.ac.id[2], kholimi@umm.ac.id[3]

Abstrak

Perkembangan *game* pada saat ini berkembang dengan sangat cepat, dalam perkermbangan game topik AI adalah topik yang paling banyak diteliti oleh beberapa peneliti khususnya pada pembuatan suatu konten *game* menggunakan metode PCG (*procedural content generation*). Pada pembuatan sebuah *game world* menggunakan metode PCG sudah banyak *developer game* yang sukses dengan mengimplementasikan metode ini, metode ini banyak digunkan pada geme dengan genre *RPG, Rouglikes, Platformer, SandBox, Simulation* dan lain sebagainya, Pada penelitian ini berfokus pada pengembangan sebuah *game world generator* untuk *game* berjenis *open world* yang berupa sebuah kepulauan dengan metode PCG dengan menggunakan algoritma perlin noise sebagai algoritma pembentuk textur utama pulau yang dimana pada penelitian ini memanfaatkan beberapa variable *noise* seperti *octave*, *presistance* dan *lacunarity* guna untuk menambah kontrol dari hasil textur yang dihasilkan serta algoritma penempatan pulau untuk membuat sebuah *game world* yang menyerupai sebuah kepulauan. Dari hasil uji generator terkait degan pengujian *playability* dan performa dapat disimpulkan bahwa generator yang dikembangkan *playable* serta performa yang dianaliasa menggunakan notasi Big O menunjukkan $O(n)$ (linear).

Kata Kunci: *Perlin Noise, Procedural Content Generation, Game World, Playability, Video Game*

Abstract

Game development is currently growing very fast, game development AI is the most discussed topic by most researchers especially in the developing of game content using the PCG (procedural content generation) method. In making a game world using the PCG method, many game developers have succeeded by implementing this method, this method is widely used on RPGs, Rouglikes, Platformers, SandBox, Simulations and ect, This study focuses on developing a game world generator game for open world type games in the form of an archipelago using the PCG method using the noise perlin algorithm as the island's main texturizing algorithm which in this study utilizes several noise variables such as octave, presistance and use for add control of the texture results as well as the island placement algorithm's to create a game world that resembles an archipelago form. From the generator test results related to the playability and performance testing, it shows that map are being generated by the generators are playable and performance that are analyzed using Big O notation show O (n) (linear).

Keywords: *Perlin Noise, Procedural Content Generation, Game World, Playability, Video Game*

1. Pendahuluan

Untuk mendesain beberapa konten dari *game world* tersebut, saat ini para pengembang *game* industri digital menggunakan metode *Prosedural Content Generation* (PCG) [1]. Hal tersebut di aplikasikan karena mengurangi beban developer dalam membangun konten – konten *game* mereka dan pada sisi lain, beberapa *game* yang mengimplementasikan PCG akan membuat suatu *game* tersebut memberikan pengalaman yang unik dan membuat konten *game* dalam jumlah yang besar tanpa di simpan ke dalam memori, Sebagai contoh, *Dungeons* dalam *game* klasik *Rogue* (oleh Michael Toy, Glenn Wichman, Ken Arnold, 1980) di buat secara dinamis setiap *game* tersebut di mulai [2].

Dalam implementasi *Prosedural Content Generation* (PCG) untuk konten *game* yang akan di bangun, memerlukan algoritma yang berbeda beda setiap kontennya, untuk pembuatan sebuah *game world* tentunya memiliki beberapa konten pembangun, contoh dalam pembuatan *game world* berjenis kepulauan, beberapa konten seperti lautan, daratan, langit, pepohonan,

makhluk hidup dan konten–konten lainnya memiliki algoritma yang berbeda beda dalam pembuatannya, pada penelitian ini, peneliti berfokus dalam pembuatan konten secara prosedural untuk *terrain* (medan) sebuah kepulauan, daratan ini membuat sebuah pulau-pulau kecil dan beberapa pulau besar. Dalam pembuatan daratan secara prosedural, ada beberapa algoritma yang bisa di terapkan seperti *Diamon-Square algorithm, Genetic algoritm, Cellular automata, Simplex noise, Worley noise*, dan *perlin noise* [3][4][5]. Namun karena kasus dalam penelitian ini adalah pembuatan sebuah daratan pada kepulauan, peneliti memilih algoritma *perlin noise*.

Perlin noise adalah algoritma yang ditemukan oleh Ken Perlin pada tahun 1985 sebagai teknik penghalusan pada *noise* yang saat itu terlalu kasar digunakan pada proses penciptaan *terrain* (medan) [6] , *Perlin noise* juga dikenal sebagai *gradient noise* dimana titik-titik gradien diatur secara *pseudo-random* pada sebuah ruang dan terjadi proses interpolasi dan penghalusan di antara titik-titik tersebut [7]. Penelitian asli yang menjelaskan penggunaan *noise* dalam generasi tekstur dirilis pada 1985. Sejak itu perlin menerbitkan makalah lain dari penelitian sebelumnya berjudul "Improving *noise*" [8] yang menggambarkan suatu algoritma *noise* yang memiliki tujuan memperbaiki algoritma yang telah ada sebelumnya.

Penelitian yang dilakukan oleh Olsen [9] meneliti tentang optimasi dari pembuatan *terrain* secara prosedural menggunakan beberpa metode seperti *mid-point disPlacement* dan *cellular automoata* untuk meminimalisir hasil dari kecuraman sebuah *terrain* yang telah di buat secara prosedural, hasil penelitian ini di manfaatkan *ke dalam game* berjenis *realtime strategy* (RTS) dan menunjukkan hasil bahwa metode yang di evaluasi dan di modifikasi oleh Olsen dapat digunkan untuk menyempurnakan medan yang dibuat secara prosedural dan dapat digunakan pada contoh jenis *game* yang serupa.

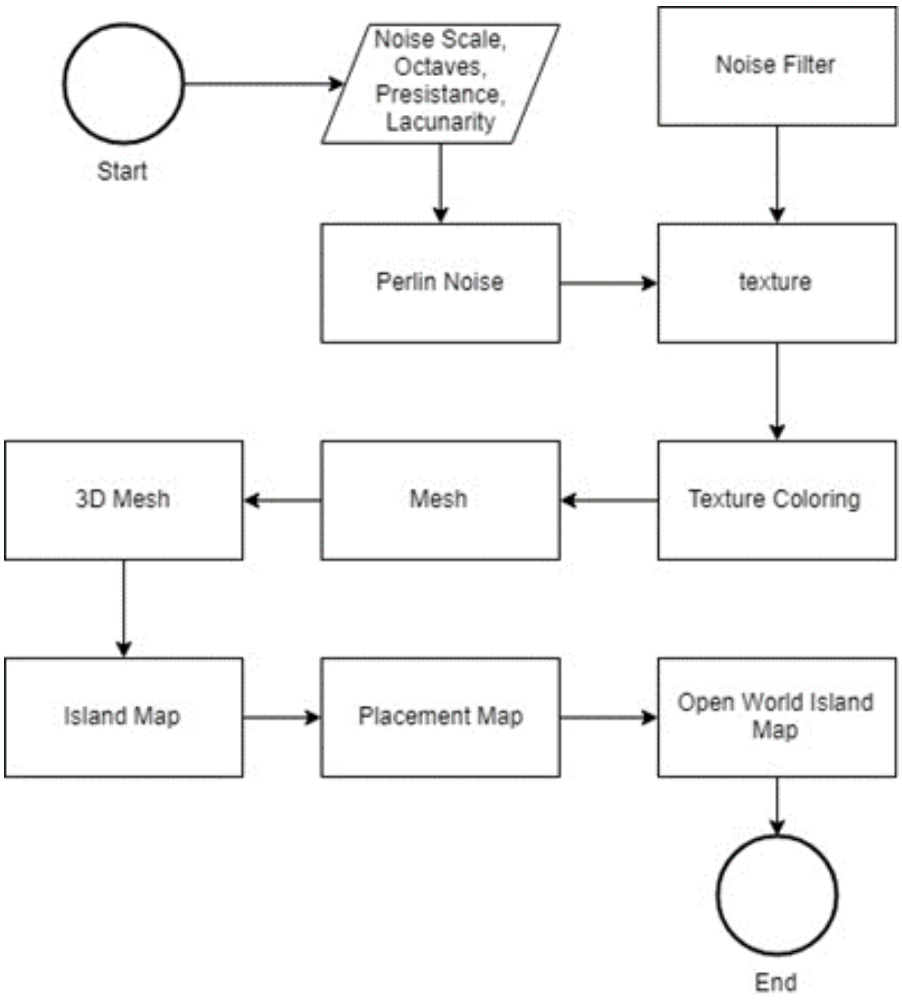
Penerapan algoritma *perlin noise* pada *Unity Engine* telah dilakukan sebelumnya pada penelitian yang dilakukan Vuontisjärvi [3] yaitu membuat sebuah planet secara prosedural. Dalam penelitian tersebut algoritma perin *noise* digunakan untuk membuat data permukaan planet yang akan di bangun bada sebuah bidang lingkaran. Pada penelitian yang di lakukan oleh Andersson [10] mengimplementasikan *perlin noise* dan beberapa algoritma *noise* lainnya untuk membuat lingkungan 3D berdasarkan jalanan, Sedangkan penelitian oleh Hyttinen [5] *perlin noise* digunakan sebagai metode untuk membuat permukaan sintesis. Penelitian yang dilakukan oleh Paraberry [11] meneliti tentang pembuatan permukaan secara prosedural berdasarkan data dari suatu daerah menggunakan beberapa algoritma *noise*, salah satunya *perlin noise*.

Penelitian sebelumnya sudah banyak menggunakan *perlin noise* sebagai algoritma utama pada *terrain generator* andalan dengan hasil yang bagus, *perlin noise* juga dapat merepresentasikan *terrain* lebih bagus sesuai dengan kriteria *terrain* yang diinginkan. Dalam penerapan *perlin noise* sebagai algoritma pembuatan *terrain* secara prosedural peneliti ingin mengembangkan pembuatan *terrain* untuk *game open world* menggunakan algoritma *perlin noise*. Sebagai implementasinya peneliti menggunakan *game ocean conqueror*, *game Ocean Conqueror* menggambarkan *game world* Samudra dengan genre *game Open world* diperlukan sebuah kepulauan yang di mana terdapat pulau – pulau dengan jenis permukaan yang berbeda–beda setiap kali dimainkan, dibutuhkan sebuah algoritma yang tepat dalam membuat *map* secara prosedural. Sedangkan pada penelitian sebelumnya belum pernah dilakukan penelitian untuk *terrain generator* pada *game open world*, yang dimana pada umumnya *map game open world* dikembangkan dengan teknik manual, karena *game Ocean Conqueror* berupa *Open world* sebuah kepulauan dimana *map* ini pada umumnya berbeda pada jenis *game* yang lain. Oleh karena itu penulis mendapat ide untuk mengimplementasikan algoritma *perlin noise* pada jenis *game open world*, dan menulis skripsi yang berjudul "*Island Generator* pada *Game Open world* Menggunakan Algoritma *Perlin noise*". Hal yang ingin diteliti adalah apakah *Perlin noise* bisa di implementasikan pada *map game open world* dan berhasil *membuat map* secara prosedural.

2. Metode Penelitian

Pada rancangan aplikasi Gambar 1, langkah pertama yang akan di selesaikan adalah menentukan variabel pembentuk *perlin noise* seperti skala *noise*, *octave*, *persistence* dan *lacunarity*. Kemudian hasil dari variabel tersebut akan membentuk sebuah *perlin noise* yang dimana hasilnya nantinya akan dijadikan sebuah *texture* utama untuk *map* yang akan di bangun. Untuk menjadikannya *map* berbentuk 3D dan terbentuk seperti kepulauan, diperlukan sebuah *mesh* dan filter untuk memfilter hasil dari *noise* yang sudah di generasi. Filter yang digunakan pada penelitian ini ada 2 yaitu radial dan *box*. Setelah data *texture* sudah di tambahkan *mesh* dan filter barulah data *terrain* 3D terbentuk dan kemudian melalui tahap *rendering* dan

membentuk sebuah 1 *map* pulau terbentuk, langkah tersebut di ulang sesuai banyak pulau yang akan di generasi untuk membentuk *map Open world*. Untuk membentuk sebuah *map open world*, pulau - pulau yang dibuat akan ditaruh sesuai dengan *size map* yang diberikan dan terbentuklah sebuah *map open world* dengan berisi kepulauan yang dibuat menggunakan algoritma *perlin noise*.



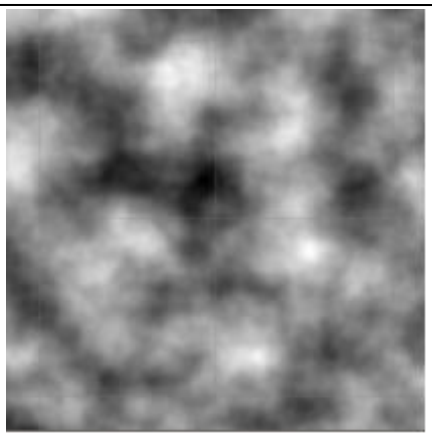
Gambar 1. Rancangan dari Generator

2.1 Penerapan Algoritma Perlin noise

Untuk mengimplementasikan *perlin noise* pada *game engine* Unity, peneliti menggunakan fungsi yang telah disediakan oleh Unity yaitu Mathf.Perlin*Noise*() namun karena pada pengujian ini menggunakan variabel *scale*, *octave*, *persistence*, dan *lacunarity* alur pembuatan *perlin noise* perlu sedikit perubahan.

Dalam mengimplementasikan *perlin noise* sebagai *map generator* pada *game engine* Unity. Diperlukan beberapa *class* dan *script* untuk mengimplementasikan hasil berupa sebuah *gameobject* pada Unity, *Script* yang paling utama adalah *script* pembentuk *noise*. pada implementasi *perlin noise* menggunakan fungsi *perlin noise* Unity, perlu ditambahkan variabel lain seperti variabel pembentuk *coherent noise* pada *script* yaitu *octave*, *persistence* dan *lacunarity.*

Pada penelitian ini, untuk mendekati hasil *map* yang berupa kepulauan, nilai variabel yang digunakan adalah *Scale* 50, *Octave* 4, *Persistence* = 0.426, *Lacunarity* = 2. Hasil *noise* yang dibuat dengan nilai variabel tersebut dapat dilihat pada Gambar 2.



Gambar 2. Hasil Perlin Noise

2.2 Generating texture

Data dari hasil *noise* tersebut kemudian di aplikasikan ke dalam sebuah *game object* bertipe *texture* pada *game engine* Unity. Pada setiap *pixel* dari *texture* tersebut diisi dengan data hasil dari *noise* yang telah di generasi pada proses sebelumnya sesuai dengan luas dimensi yang di inginkan, jika hasil nilai dari *noise* tidak sesuai dengan kebutuhan *texture* yang di inginkan yang dimana pada penelitian ini berjenis *texture* sebuah pulau, maka langkah perhitungan variabel pembentuk *noise* kembali di atur hingga sesuai dengan kebutuhan dari *texture* yang sesuai dengan kepulauan tersebut.

2.3 Noise Filter

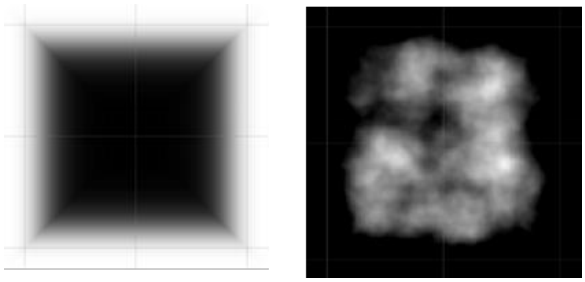
Setelah *membuat* sebuah *noise*, untuk mendekati *texture* dan hasil kepulauan dengan mengatur variabel *perlin noise* tidak akan cukup untuk menghasilkan sebuah kepulauan oleh karena itu diperlukannya sebuah filter untuk *membuat* bentuk sebuah pulau dari *perlin noise*.

Cara kerja dari filter ini adalah mengurangi nilai dari *perlin noise* dengan filter yang generasi. Karena *perlin noise* adalah sebuah *noise* yang berisi nilai 0 hingga 1 oleh karena itu filter ini nantinya mengubah nilai tersebut sesuai area sehingga membentuk *noise* baru setelah difilter. Untuk membuat *noise* agar membentuk sebuah pulau yaitu dengan menggunakan filter, dimana fungsi filter ini adalah membuat nilai pada sekeliling *texture noise* menjadi nilai 0 yang nantinya digunakan sebagai perairan. Berikut Persamaan 1 rumus filter yang digunakan dalam implementasi.

$$f(x) = \frac{x^a}{x^a + (b - bx)^a} \quad (1)$$

Dengan:
a = titik pertama (minimum) dalam fungsi gradasi
b = titik kedua (maximum) dalam fungsi gradasi

Nilai *noise* setalah dikurangi dengan filter akan membentuk pola sesuai dengan nilai filter yang telah dibuat sebelumnya, warna putih menandakan bahwa nilai *noise* adalah 1 dan hitam adalah 0. Hasil dari *noise* setelah difilter dapat dilihat pada Gambar 3.

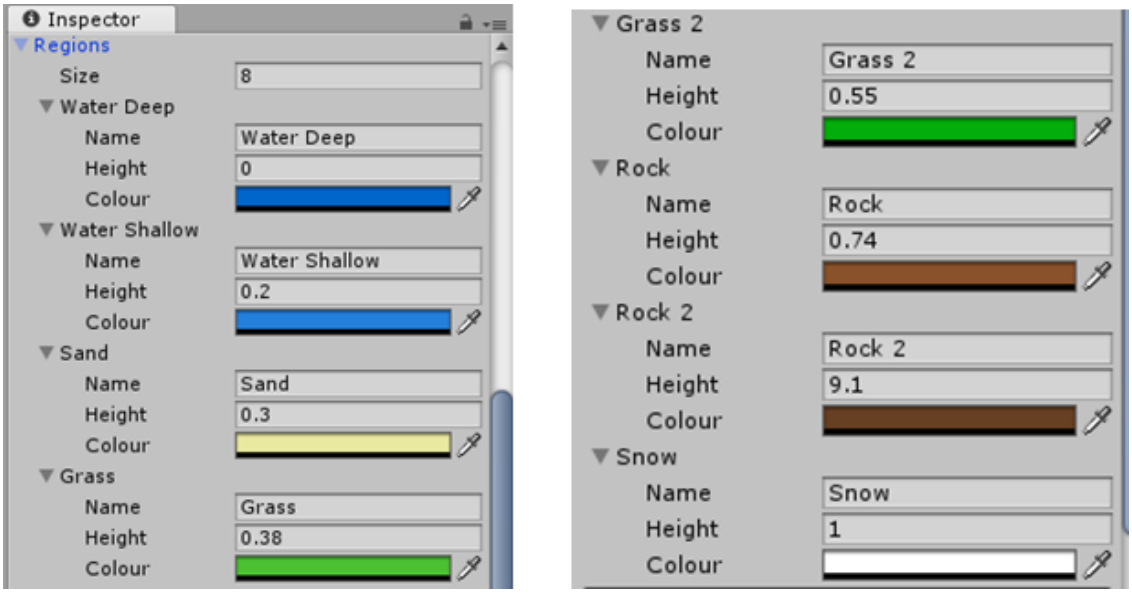


Gambar 3. Filter dan hasil filter

2.4 Texture Coloring

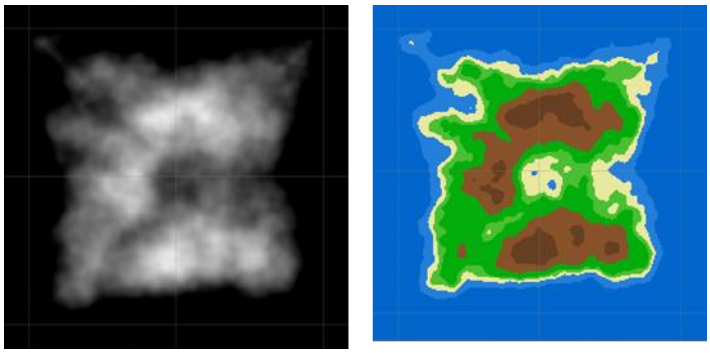
Pada tahap *rendering*, nilai *noise* yang sudah dikalkulasi dengan filter akan mempunyai 2 jenis warna gradasi yaitu hitam dengan nilai 0 hingga putih dengan nilai maksimum 1. Dalam mewarnai *texture* dari *noise* yang dibentuk, pewarnaan dilakukan berdasarkan dari *range* nilai *noise* yang telah di generasi.

Dalam penelitian ini, pewarnaan dilakukan berdasarkan pembagian wilayah daratan pulau yang akan di generasi. Wilayah dibagi menjadi 8 bagian yaitu air dalam, air dangkal, pasir, dataran rendah, dataran tinggi, pegunungan rendah, pegunungan tinggi, dan puncak. Dapat dilihat pada Gambar 4 pembagian wilayah dilakukan berdasarkan masing–masing nilai *noise* pada setiap wilayah.



Gambar 4. Inspector pewarnaan noise

Pada Gambar 5 *noise* sebelum diberi warna (kiri) mempunyai nilai *noise* dari 0 hingga 1, rentang nilai ini digunakan untuk membagi wilayah dari *noise* menjadi 8 bagian seperti pada gambar (kanan).

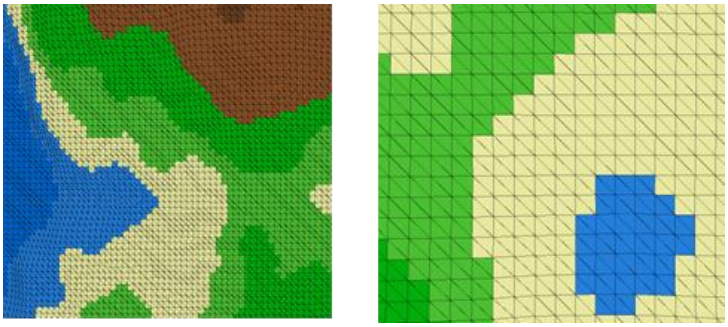


Gambar 5. Noise setelah diwarnai

2.5 Terrain Mesh

Mesh merupakan sebuah koordinat titik-titik pembentuk *terrain* (medan) dalam *game world*. Dalam pembuatan *mesh* pada Unity, di perlukan sebuah *texture* yang sebelumnya sudah di buat secara prosedural menggunakan algoritma *perlin noise*, data - data *texture* tersebut di ambil kemudian di representasikan ke dalam sebuah *pixel* yang sesuai dengan dimensi lebar dan tinggi dari *terrain* yang akan di buat.

Data dari *mesh* ini yang kemudian dimasukkan *ke dalam texture* yang dibuat sesuai dengan hasil *noise* terakhir yang setelah difilter. Hasil setelah *texture* ditambahkan *mesh* dapat dilihat pada Gambar 6.



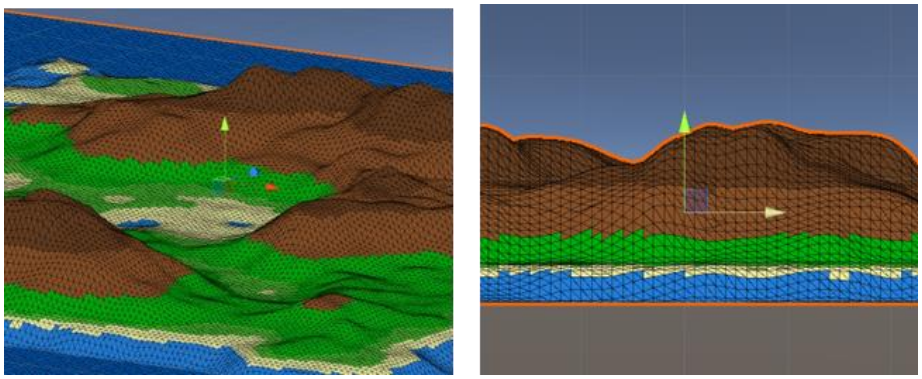
Gambar 6. Penambahan Mesh

2.6 3D Terrain

Untuk membuat *terrain* 3D pada Unity dibutuhkan *texture* yang sebelumnya sudah di tambahkan *mesh* pada komponennya. Pada setiap *Vertex* dari *mesh* mempunyai komponen posisi x, y dan z agar *mesh* yang buat menjadi 3D sesuai ketinggian nilai dari *noise* diperlukan posisi y yang bari dari setiap *vertex* pada *map.*

Untuk membuat *terrain* 3D dibutuhkan *texture* yang sebelumnya sudah di tambahkan *mesh* pada komponennya. Pada setiap *Vertex* dari *mesh* mempunyai komponen posisi x, y dan z agar *mesh* yang buat menjadi 3D sesuai ketinggian nilai dari *noise* diperlukan posisi y yang bari dari setiap *vertex* pada *map*. Berikut *pseudocode* untuk membuat posisi y baru dari *vertex*.

Dalam algoritma sederhana ini, posisi sumbu y dari *vertex* akan dikalikan dengan variabel "Mesh*Height*Multiplier" yang dimana variabel ini berfungsi sebagai variabel perkalian dari nilai *noise map* yang sudah dibuat sebelumnya sehingga hasil dari posisi *vertex* sumbu y akan berubah seperti pada Gambar 7.



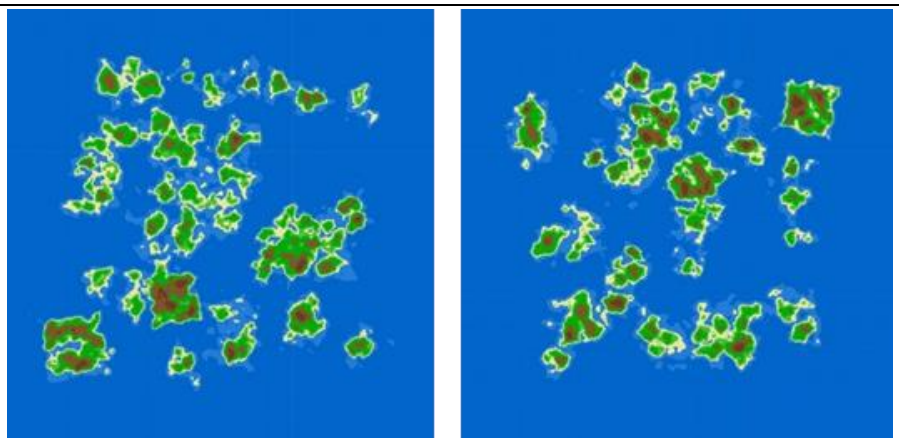
Gambar 7. Bentuk 3D Map

2.7 Placement Island

Pada penelitian ini berfokus untuk menciptakan suatu *map* pada *game open world*, seperti yang dibahas sebelumnya *Open world* adalah genre *game* yang dimana pemain dapat bebas bereksplorasi bebas pada *game world* yang disediakan. Namun untuk membuat sebuah *game world* berjenis kepulauan yang dimana setiap pulaunya di buat menggunakan algoritma *perlin noise*, namun untuk membuat sebuah *game world* kepulauan, dibutuhkan cara untuk menempatkan beberapa pulau yang sudah dibuat sebelumnya ke dalam *game world*.

Untuk menempatkan pulau-pulau yang telah dibuat sebelumnya, penelitian ini menggunakan 2 metode peletakan pulau. Peletakan ini menggunakan algoritma *random* sederhana, jenis yang pertama adalah peletakan secara *random* dalam posisi, rotasi dan jenis pulau, jenis peletakan kedua hampir sama dengan jenis pertama, namun peletakan posisi pulau tidak akan terjadi tabrakan dengan pulau lain.

Untuk membuat *open world*, *map* dari sebuah pulau tersebut dibuat dengan jumlah pulau yang diinginkan dan diletakkan dengan menggunakan algoritma *random* untuk menghasilkan sebuah *map open world*.



Gambar 8. Hasil Open World Map

Pada Gambar 8, *Map* dihasilkan dengan menaruh skala, rotasi dan posisi masing-masing pulau tersebut secara acak dari batas yang telah ditentukan dan membentuk sebuah *map open world* berjenis kepulauan.

3. Hasil Penelitian dan Pembahasan

Pengujian dari penelitian ini dibagi menjadi 2 pengujian, pengujian *playability* yaitu pengujian untuk mengetahui apakah *map* yang dibuat dapat dimainkan pada sebuah *game* atau tidak, dan pengujian performa dari pembentuk *gameworld* yaitu untuk mengetahui performa dari aplikasi yang dibangun dalam *membuat* sebuah *map* untuk *game open world* berjenis kepulauan.

3.1 Pengujian Playability

Berdasarkan penelitian yang dilakukan Olsen [9] menyebutkan bahwa *playablity* ditest berdasarkan dari jenis permainan yang dibuat. Pada penelitian ini *membuat* sebuah *gameworld* yang dimana jenis dari *game*nya adalah *open world* yang dimana pada *game* dengan jenis ini memungkinkan pemain untuk bereksplorasi sebebas pemain dengan Batasan dari aturan *game* yang telah dibuat.

Pada pengujian ini *map generator* akan diuji pada *game Ocean Conqueror*. *Game open world* ini mengambil latar belakang pertempuran bajak laut dalam menguasai Samudra yang mengambil seting pada zaman kolonial, *game world* yang dibutuhkan oleh *game* ini berupa sebuah *map* berjenis kepulauan.

3.1.1 Karakteristik Game

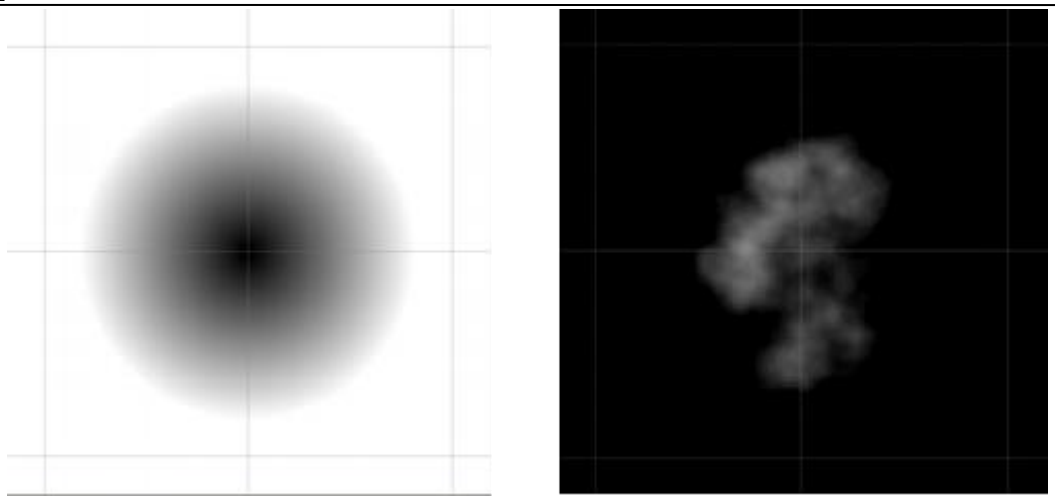
Dalam pengujian *playablity*, terdapat beberapa karakteristik *game world* yang harus dipenuhi oleh *island generator* ini, yaitu *Map* dalam *game* ini berjenis kepulauan, Pemain hanya dapat bergerak pada perairan, Masing-masing pulau dapat memiliki atribut, Terdapat pulau pulau kecil untuk *map* dengan mode *game tag-team fight.*

Untuk mengikuti aturan dari *game* dengan *map generator* yang telah dibuat, perlu adanya beberapa penyesuaian, karena *game* ini di bangun atas *engine* Unity setiap pulau perlu dimasukkan *ke dalam game object* dan mempunyai atribut *collider* masing–masing pada setiap pulaunya agar memiliki Batasan antara pulau dan lautan sesuai kriteria dari *game* tersebut. Dalam *game* Ocean Conqueror, diperlukan jenis pulau dan ukuran yang berbeda beda, untuk membuat ukuran dan jenis pulau yang berada, perlu sedikit penyesuaian pada filter pembentuk *texture noise*, filter yang digunakan berbentuk filter radial (lingkaran) untuk menghasilkan pulau dengan skala yang kecil dan berbeda.

Pada radial filter, konsep yang diterapkan dengan *box* filter sama, hanya saja untuk radial filter ini menghasilkan sebuah filter yang berbentuk lingkaran (radial).

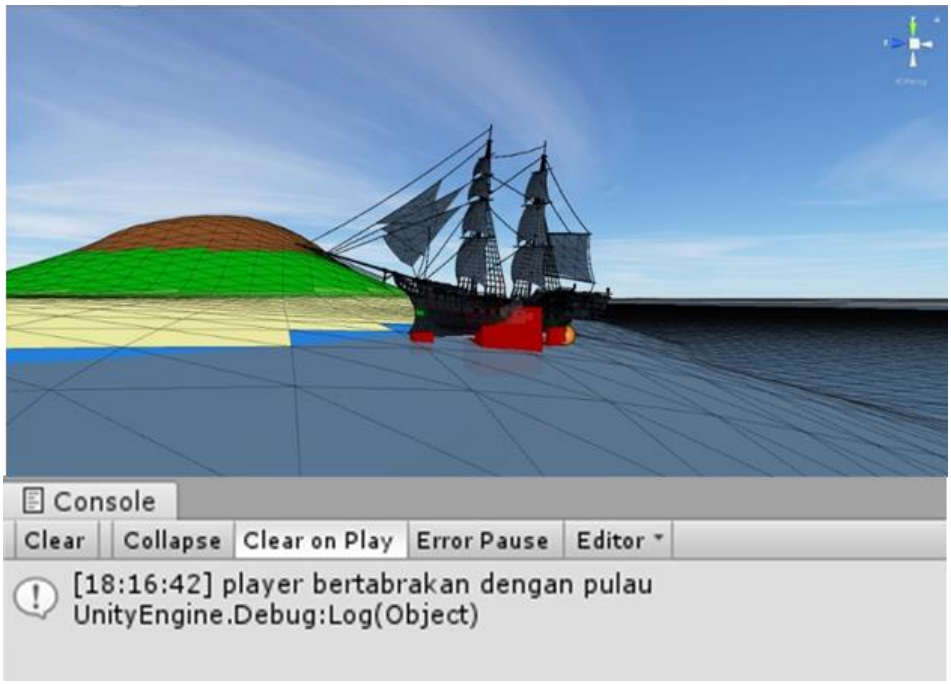
Hasil dari *noise* setelah difilter menggunakan Radial filter yang telah dibuat akan terlihat lebih kecil dan berpola melingkar, sama seperti radial filter namun pola *noise* yang dihasilkan sedikit berbeda dengan *box* filter, seperti pada Gambar 9.

ISSN: 2714-7975; E-ISSN: 2716-1382



Gambar 9. Radial Mask

Terdapat dua jenis filter yang digunakan yaitu *box* filter dan radial filter untuk menghasilkan jenis pulau yang berbeda beda. Untuk membuat elemen pulau tersebut berbeda-beda, pada Unity, masing-masing pulau perlu dimasukkan *ke dalam game object* dan memiliki komponen *collider* pada masing-masing pulau. Kemudian pemain diuji dengan menabrakkan kapal ke permukaan pulau untuk mengetahui apakah *map* yang dibuat memenuhi kriteria dari *game* tersebut.



Gambar 10. Hasil Debug

Saat *collider* dari masing–masing *object* pemain dan pulau bersentuhan, sistem memberi peringatan bahwa *collider* masing–masing *object* tersebut bertabrakan dan pemain tidak dapat bergerak pada permukaan pulau tersebut, ini menunjukkan bahwa Hasil yang didapat adalah seperti pada Gambar 10, pemain dapat berinteraksi dengan pulau yang dibuat oleh metode PCG dan sekaligus membuktikan bahwa *map generator* yang dibuat *playable*.

3.2 Pengujian Performa Generator

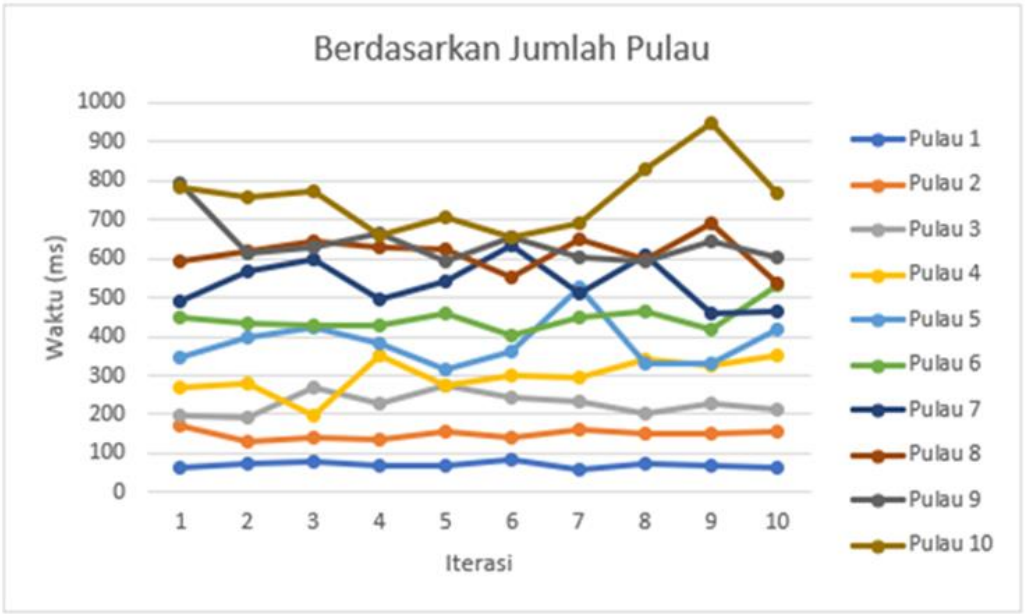
Dalam pengujian performa *generator*, pengujian dilakukan dengan mengukur kecepatan proses generasi dari sebuah pulau, dari proses pembentukan *texture*, pembentukan 3D *mesh*

hingga pembentukan peta *open world* yang telah berisi pulau–pulau. Adapun spesifikasi perangkat keras dan perangkat lunak yang digunakan sebagai berikut Intel Core i7 6th Gen 6500U (2.50 GHz), 8 GB RAM, NVIDIA GeForce 940M; Perangkat lunak Windows 10 Home, Unity 3D 2018.3.11f1, Visual Studio 2017

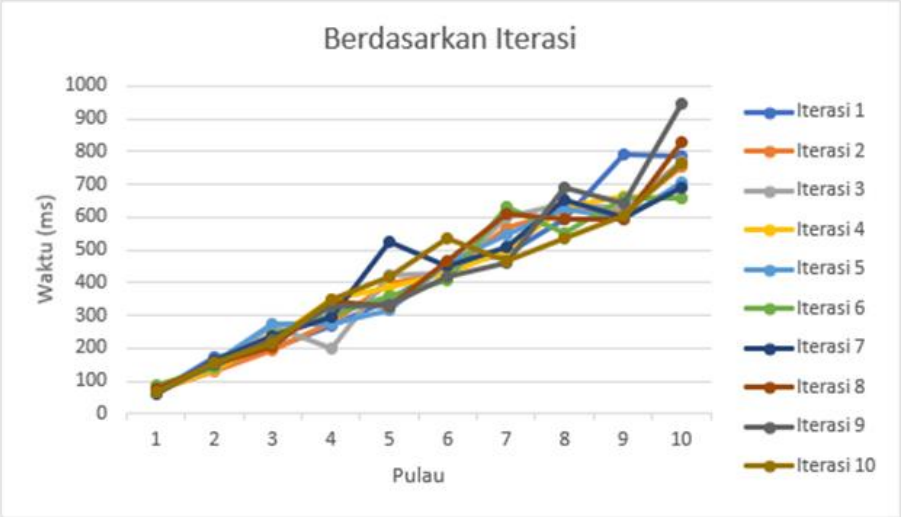
Tabel 1. Waktu Generasi Per-Pulau Dalam Millisecond (ms)

Iterasi	1 pulau	2 pulau	3 pulau	4 pulau	5 pulau
1	65	173	199	269	347
2	72	130	192	281	397
3	78	142	270	197	422
4	69	133	228	350	384
5	70	158	275	273	317
6	85	142	244	300	360
7	59	163	235	297	525
8	76	152	203	342	329
9	68	153	227	325	332
10	63	158	213	350	418

Iterasi	6 pulau	7 Pulau	8 Pulau	9 pulau	10 pulau
1	451	489	593	792	785
2	433	569	620	613	756
3	429	598	643	627	772
4	429	496	631	664	659
5	459	544	626	595	706
6	405	634	552	657	656
7	448	511	652	602	689
8	466	610	596	595	831
9	418	460	691	643	948
10	533	467	535	606	766



Gambar 11. Grafik Perbandingan Berdasarkan Jumlah Pulau



Gambar 12 Grafik Perbandingan Berdasarkan Iterasi

Pada Tabel 1 Pengujian dilakukan dengan *membuat* 1 pulau sebanyak 10 kali kemudian 2 pulau sebanyak 10 kali hingga 10 pulau, dan total proses hingga 100 proses pengujian. Dapat dilihat dalam Gambar 11 dapat disimpulkan bahwa semakin banyak jumlah pulau yang dibuat akan mempengaruhi waktu generasi pulau tersebut, dan pada Gambar 12 jika diperhatikan berdasarkan iterasinya, jumlah iterasi setiap pulau mempunyai waktu generasi yang hampir sama namun yang membedakan adalah jumlah dari pulau yang dibuat.

Berdasarkan tabel tersebut jika di analisa menggunakan notasi *Big O* algoritma yang telah dibangun merupakan notasi $O(n)$ atau notasi linear, yang bearti bahwa semakin banyak pulau yang dibuat oleh *generator* maka semakin banyak memakan waktu untuk *membuat* pulau tersebut.

4. Kesimpulan

Dari penelitian yang dilakukan algoritma *perlin noise* dapat membentuk *texture* utama dalam pembuatan *height map* atau *terrain* pada sebuah *game*. Variabel–variabel pembentuk *coherent noise* seperti *octave*, *persistance* dan *lacunarity* dapat memberi dampak dalam pembuatan *texture noise*, Sedangkan untuk membuat *map* berjenis pulau diperlukan sebuah filter untuk membuat suatu daratan yang dikelilingi lautan, serta jenis filter juga dapat mempengaruhi hasil dari *map* yang dibuat.

Berdasarkan pengujian *playability map generator* yang dibangun dapat dimainkan dan cocok untuk segala jenis *game* khususnya untuk membangun *map game open world* berjenis kepulauan karena setiap elemen pulau dapat dimodifikasi dan tentunya lebih efisien waktu jika membutuhkan desain *map* yang membutuhkan berbagai jenis desain pulau karena menggunakan konsep PCG (*Prosedural Content Generation*).

Dari hasil pengujian performa menggunakan notasi *Big O*, performa algoritma *island generator* ini menunjukkan notasi $O(n)$ atau notsi linear yang dimana kecepatan *generator* berpengaruh dengan jumlah pulau yang dibuat, semakin banyak pulau yang dibuat, maka semakin banyak waktu yang dibutuhkan.

Daftar Notasi

n : Jumlah data
a : Titik pertama (minimum) dalam fungsi gradasi
b : Titik kedua (maximum) dalam fungsi gradasi
O : Notasi Oh.

Referensi

[1] E. Adams, *Fundamentals of game design*. Pearson Education, 2014.
[2] L. Ferreira and C. Toledo, “Generating levels for physics-based puzzle games with estimation of distribution algorithms,” no. November, pp. 1–6, 2015.

[3] H. Vuontisjärvi, “Procedural planet generation in game development,” 2014.
[4] R. Smelik and K. De Kraker, “A survey of procedural methods for terrain modelling,” in *Proceedings of the …*, 2009, pp. 25–34.
[5] T. Hyttinen, “Terrain synthesis using noise,” no. May, 2017.
[6] W. Wijaya and A. Rahman, “Analisis Perbandingan Perlin Noise Dan Simplex Noise Untuk Penciptaan Permukaan Daratan Pada Pembuatan Game,” *Konf. Nas. Sist. Inf. 2018*, 2018.
[7] S. Gustavson, “Simplex noise demystified,” *Linköping Univ. Linköping, Sweden, Res. Rep.*, 2005.
[8] K. Perlin, “Improving noise,” in *ACM Transactions on Graphics (TOG)*, 2002, vol. 21, no. 3, pp. 681–682.
[9] J. Olsen, “Realtime procedural terrain generation,” 2004.
[10] M. Andersson, K. Berger, F. B. Bengtsson, B. Gelotte, J. G. Sagdahl, and S. Kvarnström, “Procedural Generation of a 3D Terrain Model Based on a Predefined Road Mesh.”
[11] I. Parberry, “Designer worlds: Procedural generation of infinite terrain from real-world elevation data,” *J. Comput. Graph. Tech.*, vol. 3, no. 1, 2014.