

**IMPLEMENTASI *PROCEDURAL CONTENT GENERATION* DALAM *TERRAIN
MODELLING GAME ASTRO SHOOTER***

SKRIPSI

OCTORICO HAPOSAN MARVIN

181401055



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

**IMPLEMENTASI *PROCEDURAL CONTENT GENERATION* DALAM *TERRAIN
MODELLING GAME ASTRO SHOOTER***

SKRIPSI

**Diajukan untuk melengkapi tugas akhir dan memenuhi syarat untuk memperoleh gelar
Sarjana Ilmu Komputer**

OCTORICO HAPOSAN MARVIN

181401055



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

PERSETUJUAN

Judul : IMPLEMENTASI *PROCEDURAL CONTENT*
GENERATION DALAM *TERRAIN MODELLING*
GAME ASTRO SHOOTER


Kategori : SKRIPSI

Nama : OCTORICO HAPOSAN MARVIN
Nomor Induk Mahasiswa :
Program Studi : SARJANA (S-1) ILMU KOMPUTER


Fakultas : ILMU KOMPUTER DAN TEKNOLOGI INFORMASI
UNIVRSITAS SUMATERA UTARA

Telah diuji dan dinyatakan lulus di Medan, 15 Oktober 2024


Dosen Pembimbing II


Anandhini Medianty Nababan, S.Kom, M.T.
NIP 199304132021022001

Dosen Pembimbing I


Fauzan Nurahmadi S.Kom., M.Cs
NIP 198512292018051001

Diketahui/Disetujui Oleh
Ketua Program Studi S1 Ilmu Komputer


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

PERNYATAAN**IMPLEMENTASI *PROCEDURAL CONTENT GENERATION* DALAM *TERRAIN
MODELLING GAME ASTRO SHOOTER*****SKRIPSI**

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

Medan, 15 Oktober 2024



Octorico Haposan Marvin
NIM, 181401055

UCAPAN TERIMA KASIH

Puji dan syukur dipanjatkan kepada Tuhan Yang Maha Esa, karena atas rahmat dan karunia-Nya peneliti dapat menyelesaikan penyusunan skripsi ini sebagai syarat untuk memperoleh gelar Sarjana Komputer dari Program Studi S-1 Ilmu Komputer Universitas Sumatera Utara.

Pada kesempatan ini, peneliti ingin mengucapkan banyak terimakasih kepada pihak-pihak yang sudah membantu dalam pengerjaan skripsi ini. Dengan segala hormat dan kerendahan hati peneliti mengucapkan terimakasih kepada:

1. Bapak Dr. Muryanto Amin, S.Sos., M.Si. selaku Rektor Universitas Sumatera Utara.
2. Ibu Dr. Maya Silvi Lydia, B.Sc., M.Sc. selaku Dekan Fakultas Ilmu Komputer dan Teknologi Informasi Universitas Sumatera Utara.
3. Ibu Dr. Amalia, S.T., M.T. selaku Ketua Program Studi S-1 Ilmu Komputer Universitas Sumatera Utara.
4. Bapak Fauzan Nurahmadi S.Kom., M.Cs Selaku Dosen Pembimbing I yang telah mendorong, membimbing, serta memberikan arahan kepada peneliti dalam penyusunan skripsi ini.
5. Ibu Anandhini Medianty Nababan, S.Kom., M.T. Selaku Dosen Pembimbing II yang juga selalu memberikan bimbingan, saran, serta motivasi kepada peneliti dalam penyusunan skripsi ini.
6. Bapak Jos Timanta Tarigan, S.Kom., M.Sc. Selaku Dosen Pembimbing I yang memberikan saran, kritikan, dan juga masukan kepada peneliti dalam penyusunan skripsi ini.
7. Ibu Sri Melvani Hardi, S.Kom., M.Kom. Selaku Dosen Pembimbing II yang memberikan saran dan masukan kepada peneliti dalam penyusunan skripsi ini.
8. Seluruh Dosen dan seluruh Staf Akademik Fakultas Ilmu Komputer dan Teknologi Universitas Sumatera Utara yang sudah memberikan pengetahuan dan memfasilitasi seluruh aspek perkuliahan dari awal semester hingga akhir.

9. Orang tua peneliti, Ibu Marta Lusiana Sianipar dan saudara kandung peneliti, Christoper Ramos Marvin, S.E., dan Anastasia Sarah Elisabeth Marvinina yang senantiasa memberikan doa, dukungan, dan motivasi kepada penulis dalam menyelesaikan skripsi ini.
10. Sahabat-sahabat peneliti dari awal hingga akhir perkuliahan, Mhd Iqbal Maulana, S.Kom., Alvin Adam Nasution, S.Kom., Angga Maulana Putra, Fikih Firmansyah, M. Daffa Abigail, Khairul Ariandi Rida, Hafizh Amrullah, Istikanah Wulandari, dan Muhammad Reza Fahlevi, S.Kom., yang memberikan dukungan dalam menyelesaikan skripsi ini.
11. Teman-teman dan rekan-rekan sesama mahasiswa stambuk 2018 yang telah terlibat secara langsung maupun tidak langsung yang tidak dapat disebutkan satu per satu dalam memberikan dukungan kepada peneliti dalam menyelesaikan skripsi ini.
12. Teman-teman serta pihak-pihak lain yang tidak dapat peneliti sebutkan satu per satu yang telah memberikan dukungan yang berarti dalam penelitian ini.

Akhir kata, semoga penelitian ini dapat memberi manfaat bagi para pembaca. Bila terdapat kesalahan pada penelitian ini, dengan ini peneliti memohon maaf yang sebesar-besarnya, serta memohon kritikan dan saran yang membangun demi perbaikan penelitian dan penulisan skripsi ini ke depannya. Terima kasih.

Medan, 15 Oktober 2024

Peneliti,

Octorico Haposan Marvin

ABSTRAK

Pengembangan *game* sebagai industri hiburan interaktif terus berkembang pesat, dengan tuntutan untuk menyajikan pengalaman bermain yang inovatif dan menarik. Salah satu elemen utama yang memengaruhi pengalaman bermain adalah lingkungan permainan, terutama dalam konteks *terrain modelling*. Proses pengembangan manual *terrain* sering kali memakan waktu dan sumber daya yang signifikan, serta dapat membatasi variasi dan dinamika lingkungan *game*.

Penelitian ini bertujuan untuk mengatasi tantangan tersebut dengan mengimplementasikan *Procedural Content Generation* (PCG) dalam *terrain modelling game Astro Shooter* untuk menghasilkan peta medan *game*. PCG menjadi pendekatan yang menjanjikan untuk menciptakan lingkungan yang dinamis dan unik secara otomatis, meningkatkan variasi, dan memberikan pengalaman bermain yang lebih menarik bagi pemain.

Algoritma PCG yang diimplementasikan dalam penelitian ini adalah algoritma *Diamond-Square*, dipilih karena kemampuannya dalam menghasilkan *terrain* yang realistis dengan pola elevasi yang alami.

Hasil pengujian menunjukkan bahwa implementasi PCG dapat menghasilkan medan permainan yang variatif, tidak monoton, dan tampak realistis, memberikan kontribusi positif terhadap pengalaman bermain *Astro Shooter*.

Kata Kunci: *Procedural Content Generation, Terrain Modelling, Astro Shooter, Diamond-Square Algorithm, Game Development*

*IMPLEMENTING PROCEDURAL CONTENT GENERATION IN
GAME ASTRO SHOOTER'S TERRAIN MODELLING*

ABSTRACT

The development of game as an interactive entertainment industry continues on rapidly with demands to create gaming experience that is innovative and interesting. One of the main elements that impacts gaming experience is the gaming environment, especially in the context of terrain modelling. Manually developing terrain is usually a very costly and time-consuming process, it could also limit game's environment's dynamic and variation.

This research aims to resolve these challenges by implementing Procedural Content Generation (PCG) in game Astro Shooter's terrain modelling to create the game's terrain map. PCG is a promising approach to create unique and dynamic environment automatically, increase variation, and giving a more interesting gaming experience to players.

The PCG algorithm implemented in this research is the Diamond-Square algorithm. This algorithm is chosen for its ability in creating realistic terrain with natural elevation patterns.

Testing results shows that PCG implementation can produce a game field that's variative, not monotonous, and looks realistic, giving positive contribution to the playing experience of Astro Shooter.

Keywords: *Procedural Content Generation, Terrain Modelling, Astro Shooter, Diamond-Square Algorithm, Game Development*

DAFTAR ISI

PERNYATAAN	iii
UCAPAN TERIMA KASIH	iv
ABSTRAK	vi
ABSTRACT	vii
DAFTAR ISI	viii
DAFTAR GAMBAR	x
DAFTAR TABEL	xi
BAB 1 PENDAHULUAN	1
1.1. Latar Belakang	1
1.2. Rumusan Masalah	3
1.3. Batasan Masalah	3
1.4. Tujuan Penelitian	3
1.5. Manfaat Penelitian	3
1.6. Metodologi Penelitian	4
1.7. Sistematika Penulisan	5
BAB 2 LANDASAN TEORI	7
2.1. <i>Procedural Content Generation (PCG) dalam Game Development</i>	7
2.1.1. <i>Definisi dan Konsep Dasar PCG</i>	7
2.1.2. <i>Manfaat PCG dalam Pengembangan Game</i>	9
2.1.3. <i>Tantangan dan Keterbatasan PCG</i>	11
2.1.4. <i>Penerapan PCG dalam Genre Game</i>	13
2.2. Permodelan Terrain dalam Game First Person Shooter	16
2.2.1. <i>Teknik Manual dalam Permodelan Terrain</i>	16
2.2.2. <i>Pendekatan Procedural dalam Permodelan Terrain</i>	16
2.2.3. <i>Karakteristik Terrain dalam Game FPS</i>	17
2.2.4. <i>Tantangan dalam Permodelan Terrain untuk Game FPS</i>	18

2.3.	<i>Genre Game First Person Shooter (FPS)</i>	18
2.4.	Penelitian yang Relevan	19
BAB 3	ANALISIS DAN PERANCANGAN SISTEM	20
3.1.	Analisis Kebutuhan Sistem	20
3.1.1.	<i>Kebutuhan Fungsional</i>	20
3.1.2.	<i>Kebutuhan Non-fungsional</i>	20
3.2.	Perancangan Sistem.....	21
3.2.1.	<i>Arsitektur Sistem</i>	21
3.2.2.	<i>Flowchart code ADiamond-Square</i>	23
BAB 4	IMPLEMENTASI DAN PENGUJIAN	27
4.1.	Implementasi Sistem PCG	27
4.1.1.	<i>Spesifikasi Perangkat Keras dan Perangkat Lunak</i>	27
4.1.2.	<i>Implementasi Algoritma Diamond-Square</i>	27
4.2.	Pengujian	34
4.2.1.	<i>Pengujian Fungsional</i>	34
4.2.2.	<i>Pengujian Kinerja</i>	34
4.2.3.	<i>Hasil Pengujian</i>	35
BAB 5	KESIMPULAN DAN SARAN	39
5.1.	Kesimpulan.....	39
5.2.	Saran	39
	DAFTAR PUSTAKA	41

DAFTAR GAMBAR

Gambar 3.1 Diagram Arsitektur Sistem	22
Gambar 3.2 Diagram <i>Code</i> Implementasi Algoritma <i>Diamond-Square</i>	23
Gambar 3.3 Diagram Fungsi <i>Initialize Height Map</i>	24
Gambar 3.4 <i>Flowchart</i> Algoritma <i>Diamond-Square</i>	25
Gambar 3.5 Diagram Fungsi <i>Create Terrain Mesh</i>	26
Gambar 4.1 Insialisasi <i>TerrainMesh</i> pada <i>ADiamondSquare</i>	28
Gambar 4.2 <i>Method BeginPlay</i>	29
Gambar 4.3 Inisialisasi <i>HeightMap</i>	30
Gambar 4.4 <i>Diamond Step</i>	31
Gambar 4.5 <i>Square Step</i>	31
Gambar 4.6 Pembagian Rekursif	32
Gambar 4.7 Membuat Mesh Terrain	34
Gambar 4.8 Hasil Rendering <i>Terrain</i> Iterasi Pertama	37
Gambar 4.9 Hasil Rendering <i>Terrain</i> Iterasi Kedua	37
Gambar 4.10 Hasil Rendering <i>Terrain</i> Iterasi Ketiga	38

DAFTAR TABEL

Tabel 4.1 Hasil Pengujian Algoritma <i>Diamond-Square</i>	35
Tabel 4.2 Rentang Waktu Pembangkitan & Framerate	38

BAB 1

PENDAHULUAN

1.1. Latar Belakang

Pembuatan aset adalah bagian yang paling memakan waktu dan biaya dalam pengembangan *game*. Hal ini dikarenakan aset *game* meliputi beragam hal, mulai dari suara dan musik sampai pemandangan dan benda-benda dalam *game*. Diantaranya, asset seni adalah yang paling lama dan paling vital untuk dibuat.

Sebuah *game* dapat dimainkan (*playable*) apabila *game* tersebut memiliki konten yang baik. Konten *game* atau isi *game* (*game content*) yang dimaksud disini adalah level permainan, peta, *quest* / misi, tekstur, karakter-karakter, tumbuh-tumbuhan, aturan, dinamika, dan struktur dalam *game* (Togelius et al., 1998). Dapat dilihat bahwasanya sebagian besar konten *game* itu adalah asset seni (tekstur, peta, karakter-karakter, tumbuh-tumbuhan dan desain struktur permainan). Diantara asset-asset itu, peta merupakan yang paling harus ada. Hal ini karena semua aset dan konten *game* akan diletakkan di atas peta (*map*) tersebut.

Di zaman ini, kebanyakan *game* menawarkan peta yang luas untuk dijelajahi, pembuatan peta permainan yang baik menjadi satu tolok ukur kualitas sebuah *game*. Tentu saja luas sebuah peta tidak menjadi satu-satunya penentu baik tidaknya suatu peta *game*. Kemenarikan peta juga menjadi salah satu poin penting dalam membuat peta/dunia *game*. Untuk itu, desain medan peta menjadi vital, tidak hanya untuk membuat *game* yang menarik dan menantang, tetapi juga untuk menjual keunikan grafis *game*.

Peta permainan yang juga peta medan suatu *game* (biasa disebut *terrain map*) dapat muncul dalam berbagai bentuk, baik sebagai dunia lautan atau bawah laut (*Raft*, *Subnautica*), dunia dengan beragam bioma (*Minecraft*, *Skyrim*), sampai alam semesta luar angkasa (*No Man's Sky*, *Eve*, *Elite*). Pada peta-peta ini ada banyak elemen yang menjadikan pembuatan *terrain map*

bagian paling lama dan paling memusingkan dalam pengembangan *game* setelah pembuatan cerita (*storyboarding*).

Untuk mengatasi hal ini, ada beberapa cara yang biasanya dipilih, seperti menggunakan asset lama, mengurangi volume konten *terrain*, dan *procedural generation* / pembangkitan prosedural dengan komputer. Diantara beragam cara tersebut, pembangkitan prosedural (PCG) adalah yang paling sering dipandang sebelah mata. Hal ini dikarenakan, berbeda dengan cara-cara lain yang dilakukan secara manual, PCG secara fundamental membuat *terrain map* dengan komputer dari nol dengan campur tangan manusia seminimal mungkin. Karena hal ini, muncul prasangka bahwa hasil dari PCG monoton dan tidak memiliki jiwa karena sedikitnya atau bahkan tidak adanya campur tangan manusia di dalamnya, dan kurang menarik karena menggunakan pola yang dapat ditebak oleh pemain (Short et al., 2017).

Maka dari itu, penelitian ini bertujuan menghasilkan suatu sistem yang dapat menghasilkan medan yang variative dan realistis dalam *game* dengan menggunakan metode *midpoint displacement* pada sebuah medan datar. Salah satu alasan peneliti mengangkat tema ini, meskipun sudah ada penelitian mengenai pembangkitan medan dengan menggunakan *noise* pada medan datar seperti penelitian membuat *metode random midpoint displacement* yang ditingkatkan untuk simulasi medan natural (Huang et al., 2010) dan penggunaan *midpoint displacement* 3 dimensi untuk menghasilkan media *fractal porous* (Jilesen et al., 2012), belum ada penelitian yang mengimplementasikan metode *midpoint displacement* untuk menghasilkan medan yang digunakan selain sebagai diagram/alat peraga, biasanya dalam bentuk simulasi. Ini juga berarti penelitian-penelitian tersebut hanya menggunakan satu iterasi yang dibangkitkan, tanpa menyinggung masalah monotonitas penggunaan PCG.

Game Astro Shooter merupakan *game* buatan penulis yang dibuat untuk mengimplementasikan metode ini dalam sebuah *game*, dalam hal ini, *game first-person shooter* (FPS). Penulis merasa *game* FPS, yang biasanya berorientasi level, dengan *map* yang unik di setiap level, akan lebih mudah menunjukkan penggunaan metode *midpoint displacement* dalam menghasilkan medan / *terrain* secara prosedural.

1.2. Rumusan Masalah

Procedural Content Generation sering dikritik memberikan hasil yang terkesan repetitif dan terkadang aneh / janggal (tidak realistis). Maka dari itu, peneliti mencoba menggunakan metode *midpoint displacement (diamond-square algorithm)* untuk menghasilkan medan permainan yang realistis dan mengimplementasikan *Procedural Content Generation* dalam pemodelan *terrain Astro Shooter* untuk meningkatkan tingkat keberagaman dan keunikan *terrain* permainan.

1.3. Batasan Masalah

Berdasarkan latar belakang dan rumusan masalah di atas, peneliti perlu membatasi ruang lingkup permasalahan agar hasil penelitian dapat mencapai tujuan. Batasan-batasan masalah yang ditetapkan antara lain:

1. PCG (*Procedural Content Generation*) digunakan hanya untuk membuat *terrain*
2. *Game* tidak memiliki sistem *level*
3. Menggunakan metode *midpoint displacement (diamond-square algorithm)*
4. Program yang dihasilkan berupa *prototype game first-person shooter* yang dapat menunjukkan hasil *terrain generation*
5. *Game* yang dihasilkan dikembangkan dengan menggunakan *Unreal Engine 5.3*

1.4. Tujuan Penelitian

Tujuan penelitian ini adalah untuk membangkitkan peta medan *game Astro Shooter* menggunakan *Procedural Content Generation* dengan metode *midpoint displacement*.

1.5. Manfaat Penelitian

Beberapa manfaat yang dapat diperoleh dari penelitian ini antara lain:

1. Memberikan kontribusi pada pemahaman penggunaan PCG dalam konteks pemodelan *terrain game*, memberikan alternatif desain yang lebih dinamis dan menarik. Manfaat ini juga dapat dirasakan oleh *game developer* yang ingin meningkatkan tingkat ketegangan dan keseruan dalam karya-karya mereka.

2. Memberikan kontribusi pada pengembangan industri *game*, khususnya dalam aspek penggunaan *Procedural Content Generation* dalam pemodelan *terrain*. Selain itu, penelitian ini dapat memberikan wawasan dan panduan praktis bagi pengembang *game* yang tertarik memanfaatkan teknologi ini.
3. Memberikan panduan praktis bagi pengembang *game* untuk memanfaatkan teknologi PCG dalam merancang permainan yang menarik dan inovatif.
4. Membantu pengembangan *game*, khususnya dalam hal meningkatkan keunikan dan keberagaman lingkungan permainan. Dengan memahami potensi PCG dalam pemodelan *terrain*, pengembang *game* dapat lebih efektif merancang lingkungan permainan yang dapat mempertahankan minat pemain.

1.6. Metodologi Penelitian

Metodologi penelitian yang diimplementasikan pada penelitian ini adalah:

1. Studi Pustaka

Pada tahap ini penelitian dimulai dengan mencari literatur dari beragam sumber yang kredibel dan melakukan peninjauan pustaka melalui buku-buku, jurnal, *e-book*, dan artikel ilmiah yang berhubungan dengan *Procedural Content Generation*, *Midpoint Displacement (Diamond-Square Algorithm)*, *Terrain Modelling*, dan *Game Development*.

2. Analisa dan Perancangan

Berdasarkan ruang lingkup penelitian, peneliti melakukan analisis terhadap masalah serta tahapan penelitian yang ada beserta studi pustaka yang telah dihimpun. Kemudian, peneliti melakukan perancangan *game prototype*.

3. Implementasi

Pada tahap ini, peneliti membangun *game prototype* yang akan menjadi implementasi *diamond-square algorithm* dengan menggunakan *Unreal Engine* dan *Visual Studio*.

4. Pengujian

Pada tahap ini, dilakukan pengujian pada *game* yang telah selesai untuk melihat bagaimana implementasi PCG dalam *game Astro Shooter*.

5. Dokumentasi

Pada tahap ini, penelitian yang telah dilakukan akan didokumentasikan mulai dari tahap analisis, perancangan, implementasi, hingga pengujian ke dalam penulisan skripsi.

1.7. Sistematika Penulisan

BAB 1: PENDAHULUAN

Bab 1 akan membahas latar belakang masalah, rumusan masalah, batasan masalah, tujuan penelitian, manfaat penelitian, metode yang digunakan untuk penelitian dan sistematika penulisan yang akan digunakan dalam penulisan skripsi.

BAB 2: LANDASAN TEORI

Bab 2 akan membahas mengenai tinjauan teoritis tentang *Procedural Content Generation*, *Midpoint Displacement*, dan *Terrain Modelling*.

BAB 3: ANALISIS DAN PERANCANGAN

Bab 3 akan membahas tentang analisis masalah, analisis kebutuhan, dan perancangan sistem yang akan dibangun yang digambarkan dalam berbagai diagram berdasarkan rumusan dan batasan masalah yang akan diteliti.

BAB 4: IMPLEMENTASI DAN PENGUJIAN

Bab 4 akan membahas tentang implementasi PCG dalam *game prototype Astro Shooter* untuk menghasilkan *terrain* menggunakan bahasa pemrograman C++ yang kemudian diaplikasikan dalam tahapan pengujian.

BAB 5: KESIMPULAN DAN SARAN

Bab 5 akan membahas kesimpulan hasil penelitian yang telah dilakukan disertai dengan saran yang diberikan untuk penelitian yang mendatang.

BAB 2

LANDASAN TEORI

2.1. Procedural Content Generation (PCG) dalam Game Development

Procedural Content Generation (PCG) merujuk pada teknik yang digunakan untuk menciptakan konten secara otomatis dalam permainan video. Pendekatan ini telah menjadi fokus penelitian dalam pengembangan permainan, karena potensi untuk meningkatkan *kereplay-an*, memperluas variasi konten, dan mengurangi kerja manual dalam pembuatan *game*.

2.1.1. Definisi dan Konsep Dasar PCG

Procedural Content Generation (PCG) adalah metode pembuatan konten dalam game yang dilakukan secara otomatis menggunakan algoritma dan teknik komputasi tertentu, tanpa intervensi manual yang signifikan dari desainer *game*. Konten yang dihasilkan dapat mencakup berbagai elemen dalam permainan, seperti *level*, *terrain*, karakter, objek, misi, dan cerita. PCG memungkinkan pengembang untuk menghasilkan konten yang kaya dan bervariasi dengan efisiensi yang tinggi.

1. Definisi PCG

Procedural Content Generation (PCG) atau biasa disebut juga *Procedural Generation* adalah suatu rangkaian kata yang rumit untuk menggambarkan sesuatu yang sederhana: pembuatan data oleh computer. PCG biasanya digunakan untuk menciptakan konten *video game* atau film animasi, seperti *landscape*, objek 3D, desain karakter, animasi, atau dialog NPC (*non-player character*). (*Massachusetts Institute of Technology*). PCG adalah proses menggunakan algoritma untuk menciptakan data dan konten digital secara otomatis dalam permainan video. Proses ini melibatkan penggunaan teknik matematika, algoritma komputasi, dan logika pemrograman untuk menghasilkan elemen-elemen yang membentuk dunia *game*, seperti *terrain*, *level*, karakter, dan objek.

2. Konsep Dasar PCG

Konsep dasar PCG mencakup beberapa elemen kunci yang memungkinkan pembuatan konten secara otomatis:

- **Algoritma dan Teknik Komputasi:** PCG menggunakan berbagai algoritma untuk menghasilkan konten. Algoritma ini bisa berupa algoritma deterministik atau algoritma berbasis probabilitas yang menghasilkan konten berdasarkan aturan dan parameter tertentu.
- **Randomisasi:** Salah satu elemen kunci dalam PCG adalah penggunaan elemen acak atau randomisasi untuk menciptakan variasi. Elemen acak ini memungkinkan konten yang dihasilkan berbeda setiap kali proses dijalankan, memberikan pengalaman unik bagi pemain.
- **Parameterisasi:** Konten yang dihasilkan secara *procedural* sering kali bergantung pada sejumlah parameter yang dapat diatur oleh desainer *game*. Parameter ini bisa mencakup ukuran dunia, tingkat kesulitan, jenis *terrain*, dan lain-lain. Dengan mengubah parameter ini, desainer dapat mengontrol hasil akhir dari konten yang dihasilkan.
- **Iterasi dan Refinement:** PCG sering kali melibatkan proses iteratif di mana konten dihasilkan melalui beberapa tahap dan kemudian disempurnakan. Misalnya, *terrain* mungkin dihasilkan dengan algoritma dasar terlebih dahulu, kemudian ditingkatkan dengan detail tambahan dan fitur yang lebih kompleks.

3. Contoh Teknik PCG

Berikut ini contoh-contoh penerapan teknik *Procedural Content Generation*:

- **Noise Functions:** Fungsi *noise*, seperti *Perlin noise* dan *Simplex noise*, digunakan untuk menghasilkan *terrain* dan tekstur yang realistis. Fungsi ini menghasilkan pola acak

yang terlihat alami dan digunakan dalam banyak aplikasi PCG.

- ***L-System***: Digunakan untuk menghasilkan struktur seperti tanaman dan pohon, *L-System* adalah sistem penulisan berbasis aturan yang dapat menghasilkan bentuk yang kompleks dari aturan sederhana.
- ***Tiling dan Dungeon Generation***: Teknik ini sering digunakan dalam permainan roguelike untuk menghasilkan tata letak *dungeon* secara acak. Algoritma seperti *cellular automata* dan *random walk* sering digunakan untuk tujuan ini.
- ***Fractals***: Fraktal digunakan untuk menghasilkan *terrain* yang realistis. Algoritma seperti *Diamond-Square* dan *Midpoint Displacement* adalah contoh dari teknik ini.

2.1.2. Manfaat PCG dalam Pengembangan Game

Procedural Content Generation (PCG) menawarkan berbagai manfaat signifikan dalam pengembangan *game*. Penggunaan PCG tidak hanya mempercepat proses pembuatan konten tetapi juga membuka kemungkinan baru dalam hal variasi dan kompleksitas dunia *game*. Berikut adalah beberapa manfaat utama PCG dalam pengembangan *game*:

1. Variasi Tak Terbatas

PCG memungkinkan pengembang *game* untuk menghasilkan konten yang berbeda setiap kali permainan dimulai, memberikan pengalaman unik bagi setiap pemain. Hal ini sangat penting dalam *genre game* yang mengandalkan eksplorasi dan kejutan, seperti *roguelike* dan *sandbox games*. Variasi ini dicapai dengan memanfaatkan algoritma yang menghasilkan konten berdasarkan parameter dan aturan yang telah ditentukan, sehingga meskipun dasar-dasarnya tetap sama, hasil akhirnya bisa sangat berbeda.

2. Pengurangan Waktu dan Biaya

Dengan PCG, pengembang tidak perlu membuat setiap elemen konten secara manual. Sebaliknya, mereka dapat menciptakan algoritma yang secara otomatis menghasilkan

konten berdasarkan aturan yang ditentukan. Ini menghemat waktu dan sumber daya yang signifikan, memungkinkan tim pengembang untuk fokus pada aspek-aspek lain dari *game development*, seperti desain gameplay dan narasi.

3. Pengembangan Dunia yang Luas

PCG memungkinkan pembuatan dunia game yang sangat besar dan kompleks yang sulit dicapai melalui metode manual. Dengan menggunakan algoritma yang tepat, pengembang dapat menciptakan peta dunia yang luas dengan berbagai elemen geografis, struktur, dan fitur unik tanpa perlu merancang setiap detail secara individual.

4. Peningkatan *Replayability*

Game yang menggunakan PCG seringkali memiliki *replayability* yang tinggi karena konten yang selalu berubah-ubah. Pemain dapat menikmati pengalaman bermain yang berbeda setiap kali mereka memainkan *game*, yang membuat mereka terus kembali untuk mencoba hal-hal baru. Ini sangat penting untuk menjaga ketertarikan pemain dalam jangka panjang.

5. Kreativitas dan Eksperimen

PCG memungkinkan pengembang untuk bereksperimen dengan ide-ide baru tanpa perlu menciptakan setiap elemen dari awal. Ini membuka ruang untuk kreativitas yang lebih besar dan memungkinkan pengembang untuk mencoba berbagai konsep dengan cepat dan efisien.

6. Personalisasi dan Penyesuaian

PCG memungkinkan penyesuaian konten berdasarkan preferensi pemain. Dengan menerapkan parameter yang berbeda, *game* dapat menghasilkan konten yang sesuai dengan gaya bermain dan preferensi individu pemain, sehingga meningkatkan pengalaman bermain secara keseluruhan. Biasanya ditemukan pada *game* RPG (*Role-playing game*) untuk menghasilkan *quest* dan karakter yang sesuai dengan pilihan dan tindakan pemain.

7. Adaptabilitas dan Skalabilitas

PCG sangat berguna untuk *game* yang memerlukan adaptabilitas dan skalabilitas. Konten yang dihasilkan secara *procedural* dapat dengan mudah disesuaikan untuk berbagai platform dan perangkat, serta dapat ditingkatkan atau diperluas sesuai kebutuhan tanpa memerlukan banyak perubahan manual.

2.1.3. Tantangan dan Keterbatasan PCG

Meskipun *Procedural Content Generation* (PCG) menawarkan banyak keuntungan, implementasinya dalam pengembangan game juga menghadapi berbagai tantangan dan keterbatasan yang perlu diatasi oleh para pengembang. Berikut ini adalah beberapa tantangan dan keterbatasan utama yang terkait dengan PCG:

1. Kontrol Desainer

Salah satu tantangan utama dalam penggunaan PCG adalah bagaimana menjaga kontrol desainer terhadap konten yang dihasilkan. Konten yang dihasilkan secara prosedural dapat menghasilkan hasil yang tidak diharapkan atau tidak sesuai dengan visi desainer. Untuk mengatasi hal ini, pengembang perlu:

- **Menentukan Parameter yang Tepat:** Desainer harus menentukan parameter dan batasan yang jelas untuk algoritma PCG, sehingga konten yang dihasilkan tetap sesuai dengan gaya dan tujuan permainan.
- **Menggunakan Teknik *Hybrid*:** Menggabungkan PCG dengan elemen-elemen yang didesain secara manual untuk memastikan kualitas dan konsistensi konten.

2. Keseimbangan dan Kesulitan

Mengelola keseimbangan dan tingkat kesulitan dalam permainan yang menggunakan PCG merupakan tantangan signifikan. Konten yang dihasilkan harus menyediakan tantangan yang tepat tanpa membuat permainan terlalu mudah atau terlalu sulit. Beberapa masalah yang terkait dengan ini meliputi:

- **Randomness vs. Struktur:** Konten *procedural* sering kali didasarkan pada elemen acak, yang bisa menghasilkan variasi yang luas dalam tingkat kesulitan. Menemukan keseimbangan antara *randomness* dan struktur yang terarah adalah penting.
- **Pacing:** *Pacing* atau ritme permainan harus dikelola dengan baik. Konten *procedural* bisa mengganggu *pacing* jika level atau tantangan tidak diatur dengan baik.
- **Playtesting:** Menguji permainan dengan konten yang dihasilkan secara prosedural bisa menjadi sulit dan memakan waktu, karena variasi yang luas membutuhkan pengujian ekstensif untuk memastikan bahwa semua kemungkinan hasil tetap seimbang dan dapat dimainkan.

3. Pengujian dan Debugging

Konten *procedural* memerlukan strategi pengujian dan *debugging* yang lebih komprehensif dibandingkan dengan konten yang dibuat secara manual. Beberapa tantangan dalam aspek ini meliputi:

- **Variasi yang Luas:** Jumlah variasi dalam konten yang dihasilkan membuatnya sulit untuk menguji semua kemungkinan skenario. Ini dapat mengakibatkan *bug* atau masalah keseimbangan yang tidak terdeteksi.
- **Automated Testing:** Mengembangkan alat dan metodologi pengujian otomatis untuk menangani variasi konten yang luas adalah penting, tetapi ini juga merupakan tantangan teknis yang signifikan.
- **Consistency and Reliability:** Menjamin bahwa setiap elemen yang dihasilkan tetap memenuhi standar kualitas yang diinginkan dan tidak menyebabkan *crash* atau masalah kinerja dalam permainan.

2.1.4. Penerapan PCG dalam Genre Game

Procedural Content Generation (PCG) telah diterapkan dalam berbagai *genre game* untuk meningkatkan variasi konten dan pengalaman bermain. Berikut ini adalah beberapa *genre game* utama dan cara PCG diterapkan dalam masing-masing genre tersebut:

1. Roguelike

Roguelike adalah *genre* yang dikenal dengan *gameplay* berbasis *dungeon-crawling* yang sulit dan *permadeath* (karakter mati permanen). Dalam *genre* ini, PCG biasanya digunakan untuk membuat *dungeon* atau *level* yang dihasilkan secara acak setiap kali permainan dimulai. Contoh penerapan PCG dalam *roguelike* meliputi:

- **"Rogue"**: *Game* yang menjadi dasar nama *genre* ini menggunakan PCG untuk menghasilkan *layout dungeon* yang berbeda setiap kali pemain bermain.
- **"The Binding of Isaac"**: *Game* ini memanfaatkan PCG untuk menciptakan ruang bawah tanah, *item*, dan musuh yang berbeda di setiap *run*, memastikan bahwa setiap permainan adalah unik.

Penerapan PCG dalam *roguelike* memberikan *replayability* yang tinggi karena pemain menghadapi tantangan baru setiap kali bermain, sehingga meningkatkan ketegangan permainan.

2. Sandbox

Sandbox games menawarkan dunia terbuka di mana pemain memiliki kebebasan untuk mengeksplorasi, membangun, dan berinteraksi dengan lingkungan. PCG digunakan untuk menciptakan dunia yang luas dan beragam secara otomatis. Contoh penerapan PCG dalam *sandbox* meliputi:

- **"Minecraft"**: Salah satu contoh paling terkenal, Minecraft menggunakan PCG untuk menghasilkan *terrain*, *biomes*, gua, dan struktur lainnya secara acak, memungkinkan pemain untuk mengeksplorasi dan membangun dalam dunia yang selalu berbeda.
- **"Terraria"**: *Game* ini juga menggunakan PCG untuk menciptakan dunia, musuh, dan *loot* yang beragam, memberikan pengalaman eksplorasi yang tak terbatas.

PCG dalam *sandbox* games memungkinkan penciptaan dunia yang sangat luas tanpa batasan manual, memberikan kebebasan dan variasi yang hampir tak terbatas bagi pemain.

3. Role-Playing Games (RPG)

Dalam *genre* RPG, PCG sering digunakan untuk menghasilkan dunia *game*, *quest*, *item*, dan karakter yang beragam. Penerapan PCG dalam RPG memungkinkan pemain untuk mengalami cerita dan petualangan yang berbeda setiap kali bermain. Contoh penerapan PCG dalam RPG meliputi:

- **"No Man's Sky"**: *Game* ini menggunakan algoritma PCG untuk menciptakan seluruh alam semesta yang terdiri dari triliunan planet, masing-masing dengan ekosistem, flora, dan fauna yang unik.
- **"Diablo" series**: *Game* ini menggunakan PCG untuk menghasilkan *dungeon*, *loot*, dan *quest* yang berbeda setiap kali pemain memulai permainan, memastikan pengalaman yang selalu baru.

PCG dalam RPG meningkatkan *replayability* dan eksplorasi, membuat dunia *game* terasa hidup dan dinamis, serta memberikan kejutan dan variasi dalam setiap sesi permainan.

4. First Person Shooter (FPS)

Meskipun kurang umum dibandingkan *genre* lain, PCG juga diterapkan dalam beberapa game FPS untuk menciptakan variasi dalam elemen-elemen tertentu seperti level design dan senjata. Contoh penerapan PCG dalam FPS meliputi:

- **"Borderlands" series:** *Game* ini terkenal dengan penggunaan PCG untuk menciptakan variasi senjata yang luar biasa. Setiap senjata memiliki kombinasi unik dari berbagai komponen, menghasilkan jutaan kemungkinan senjata yang berbeda.
- **"DayZ":** Meskipun lebih dikenal sebagai *game survival*, DayZ menggunakan PCG untuk penempatan *loot* dan *item* di dalam dunia *game*, memastikan pengalaman yang berbeda setiap kali bermain.

Dalam FPS, PCG dapat meningkatkan variasi dan strategi dalam permainan, memberikan pemain pengalaman yang unik dan menantang setiap kali bermain.

5. Platformer

Platformer games, yang menekankan pada navigasi dan traversal melalui level dengan berbagai rintangan, juga menggunakan PCG untuk menciptakan level yang beragam. Contoh penerapan PCG dalam platformer meliputi:

- **"Spelunky":** *Game* ini menggunakan PCG untuk menghasilkan level-level yang berbeda setiap kali pemain bermain, membuat setiap run unik dan menantang.
- **"Dead Cells":** *Game roguelike metroidvania* ini menggunakan PCG untuk menciptakan layout level yang berubah-ubah, memberikan variasi dalam setiap permainan.

Penerapan PCG dalam platformer memungkinkan variasi tantangan dan rintangan, membuat game tetap menarik dan menantang meski telah dimainkan berulang kali.

2.2. Permodelan Terrain dalam Game First Person Shooter

Permodelan terrain adalah elemen krusial dalam pengembangan game, terutama untuk genre First Person Shooter (FPS). Dalam game FPS, pemain merasakan dan berinteraksi dengan lingkungan dari sudut pandang orang pertama, sehingga kualitas dan detail dari terrain sangat mempengaruhi pengalaman bermain. Terrain yang realistis dan dinamis tidak hanya meningkatkan estetika permainan tetapi juga memberikan tantangan dan variasi gameplay yang penting untuk menjaga minat pemain.

2.2.1. Teknik Manual dalam Permodelan Terrain

Teknik manual dalam permodelan terrain mencakup proses-proses seperti *sculpting* dan *painting*. Dalam teknik *sculpting*, desainer game menggunakan alat-alat digital untuk memahat terrain, menciptakan fitur-fitur geografis seperti gunung, lembah, dan bukit secara detail. *Painting*, di sisi lain, digunakan untuk menambahkan tekstur dan warna pada terrain yang sudah dipahat, memberikan tampilan yang lebih realistis.

Teknik ini memungkinkan kontrol penuh atas setiap aspek terrain, tetapi sering kali memerlukan waktu dan tenaga yang signifikan. Dalam game FPS, dimana detail dan kualitas lingkungan sangat penting, metode manual sering digunakan untuk elemen-elemen kunci yang membutuhkan perhatian khusus.

2.2.2. Pendekatan Procedural dalam Permodelan Terrain

Pendekatan *procedural* dalam permodelan terrain menggunakan algoritma untuk menghasilkan terrain secara otomatis. Teknik ini dapat mengurangi waktu dan tenaga yang dibutuhkan dalam proses pembuatan terrain secara manual. Salah satu keunggulan utama dari pendekatan procedural adalah kemampuannya untuk menciptakan variasi yang luas dalam terrain, yang dapat meningkatkan replayability dari sebuah game.

Dalam game FPS, *terrain procedural* memungkinkan lingkungan yang selalu baru dan berbeda setiap kali pemain memulai permainan. Hal ini penting untuk menjaga minat pemain dan memberikan pengalaman bermain yang dinamis. Pendekatan *procedural* juga memfasilitasi pembuatan terrain yang besar dan kompleks, yang mana akan sulit dicapai

dengan metode manual. Beberapa metode yang sering digunakan dalam pendekatan procedural adalah:

1. ***Perlin Noise dan Simplex Noise***:

Algoritma ini menghasilkan peta ketinggian yang dapat digunakan untuk menciptakan terrain yang tampak alami dengan variasi bukit, lembah, dan dataran. *Perlin Noise* menghasilkan pola yang lebih halus, sementara *Simplex Noise* lebih cepat dan mengurangi artefak visual.

2. **Algoritma *Fractal* (seperti *Diamond-Square*)**:

Algoritma ini menghasilkan detail *terrain* dengan menggunakan konsep fractal untuk menciptakan variasi yang realistis dan skala yang berbeda. *Diamond-Square* adalah algoritma populer yang digunakan untuk menciptakan terrain procedural dengan mendetailkan pola dasar melalui pembagian berulang. Model berbasis *fractal* dianggap banyak orang sebagai metode yang efisien untuk menghasilkan *terrain* yang realistis (Fournier et al., 1982).

2.2.3. Karakteristik Terrain dalam Game FPS

Terrain dalam game FPS memiliki beberapa karakteristik khusus yang membedakannya dari genre game lainnya. Beberapa karakteristik tersebut antara lain:

- **Realisme:** *Terrain* dalam game FPS harus realistis untuk meningkatkan imersi pemain. Elemen seperti tekstur tanah, vegetasi, dan fitur geografis harus dibuat sedetail mungkin.
- **Navigasi:** *Terrain* harus dirancang sedemikian rupa sehingga memungkinkan navigasi yang mudah dan intuitif untuk pemain. Jalan setapak, jembatan, dan titik-titik orientasi harus diperhatikan dalam desain *terrain*.
- **Strategi:** *Terrain* dalam game FPS sering digunakan untuk mengatur strategi permainan. Tempat persembunyian, jalur serangan, dan posisi strategis harus dipertimbangkan dalam pembuatan *terrain*.

- **Interaktivitas:** *Terrain* dalam game FPS sering kali interaktif. Pemain dapat berinteraksi dengan elemen-elemen *terrain*, seperti menghancurkan objek, membuka jalan baru, atau menggunakan lingkungan untuk keuntungan taktis.

2.2.4. Tantangan dalam Permodelan *Terrain* untuk Game FPS

Permodelan *terrain* untuk game FPS menghadapi beberapa tantangan, antara lain:

- **Keseimbangan antara Realisme dan Performansi:** Mencapai keseimbangan antara realisme dan performa *game* adalah tantangan utama. *Terrain* yang sangat detail dapat membebani sistem dan mengurangi performa *game*.
- **Variasi dan Repetisi:** Menghindari repetisi dan menciptakan variasi dalam *terrain procedural* adalah tantangan lain. *Terrain* yang berulang dapat mengurangi daya tarik visual dan *gameplay* dari *game*.
- **Optimasi:** *Terrain* yang besar dan kompleks harus dioptimalkan agar dapat berjalan lancar di berbagai perangkat. Teknik seperti *Level of Detail* (LOD) dan *culling* sering digunakan untuk mengatasi masalah ini.

2.3. Genre Game First Person Shooter (FPS)

Genre game First Person Shooter (FPS) adalah salah satu *genre* yang paling dikenal dan populer dalam industri permainan video. Dalam game FPS, pemain mengalami lingkungan game dari sudut pandang karakter utama, yang sering kali disebut sebagai "sudut pandang orang pertama". Dalam sudut pandang ini, pemain melihat dunia permainan melalui mata karakter yang mereka kendalikan, memberikan pengalaman yang lebih langsung dan imersif.

Karakteristik Utama:

- **Sudut Pandang Orang Pertama:** Sudut pandang ini memungkinkan pemain untuk melihat dunia game seperti melalui mata karakter yang mereka kendalikan. Hal ini menciptakan rasa kehadiran yang kuat dan memungkinkan pengalaman yang sangat imersif.

- **Pertempuran Berbasis Aksi:** FPS sering kali fokus pada pertempuran yang cepat dan intens, di mana pemain harus merespons dengan cepat terhadap ancaman yang datang. Ini memerlukan keterampilan motorik yang baik dan pengambilan keputusan yang cepat.
- **Arsenal Senjata yang Luas:** Salah satu ciri khas dari FPS adalah keberagaman senjata yang tersedia untuk pemain. Mulai dari senapan mesin, senapan sniper, hingga senjata ledakan, pemain memiliki banyak pilihan untuk menyesuaikan gaya bermain mereka.
- **Level Desain yang Kompleks:** Level dalam game FPS sering kali dirancang dengan teliti untuk memberikan pengalaman bermain yang menarik. Ini bisa termasuk berbagai jenis lingkungan seperti kota-kota, kompleks industri, atau lokasi militer, dengan banyak jalan rahasia, tempat persembunyian, dan ruang terbuka untuk eksplorasi.

2.4. Penelitian yang Relevan

Butir-butir di bawah ini merupakan beberapa penelitian yang relevan dengan penelitian yang dilakukan pada skripsi ini, yaitu penelitian yang mengimplementasikan suatu metode *procedural generation*.

1. Penelitian menggunakan metode *midpoint displacement* tiga dimensi untuk menghasilkan media geometri *fractal porous* tiga dimensi (Jilesen et al., 2012) membuat metode berdasarkan algoritma *Diamond-Square* untuk menghasilkan bentuk – bentuk bebatuan yang berpori secara *procedural*
2. Penelitian *procedural generation* untuk *terrain modelling* menggunakan *reverse midpoint-displacement* (Fikrie., 2017) berhasil membangkitkan *heightmap* yang memiliki jalur perbukitan dan sungai secara *procedural*
3. Penelitian untuk menerapkan PCG pada pembangkitan *level game maze* heksagonal dalam Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya (Putri et al., 2019) menggunakan algoritma BTA (*binary tree algorithm*) untuk membangkitkan *level game* berupa labirin dengan *grid* heksagonal

BAB 3

ANALISIS DAN PERANCANGAN SISTEM

3.1. Analisis Kebutuhan Sistem

Analisis kebutuhan sistem dilakukan untuk menjelaskan secara rinci kebutuhan fungsional dan non-fungsional yang harus dipenuhi oleh sistem. Analisis kebutuhan sistem adalah langkah penting dalam pengembangan permainan. Ini membantu kita memahami secara jelas apa yang diperlukan oleh sistem PCG untuk memastikan bahwa permainan memiliki lanskap yang dinamis dan menarik setiap kali dimainkan. Analisis kebutuhan ini akan membentuk dasar bagi perancangan sistem yang efektif dan efisien.

3.1.1. Kebutuhan Fungsional

Kebutuhan fungsional menggambarkan fitur-fitur utama yang harus dimiliki oleh sistem PCG agar dapat menghasilkan lanskap yang sesuai dengan kebutuhan permainan *Astro Shooter*. Hal ini meliputi:

- Generasi *landscape* secara dinamis berdasarkan parameter yang dapat disesuaikan.
- Kemampuan untuk menghasilkan variasi lanskap yang berbeda untuk setiap sesi permainan.
- Integrasi dengan sistem permainan untuk memastikan keseimbangan dan kesesuaian tingkat kesulitan.
- Kemampuan untuk mempertahankan kinerja permainan yang stabil bahkan saat menghasilkan lanskap secara dinamis.

3.1.2. Kebutuhan Non-fungsional

Kebutuhan non-fungsional mencakup aspek-aspek seperti kinerja, skalabilitas, dan keamanan sistem PCG. Beberapa kebutuhan non-fungsional yang penting dalam konteks ini meliputi:

- **Kinerja:** Sistem harus mampu menghasilkan lanskap secara cepat untuk memastikan pengalaman bermain yang lancar.
- **Skalabilitas:** Sistem harus dapat menangani berbagai tingkat kompleksitas lanskap tanpa mengorbankan kinerja.
- **Kualitas:** Lanskap yang dihasilkan harus memiliki kualitas visual yang memadai dan tidak terlalu repetitif.

3.2. Perancangan Sistem

Bagian ini akan membahas rancangan sistem yang akan diimplementasikan untuk memenuhi kebutuhan yang telah dianalisis sebelumnya. Ini mencakup arsitektur sistem, algoritma PCG yang akan digunakan, dan integrasi dengan sistem *game Astro Shooter*.

3.2.1. Arsitektur Sistem

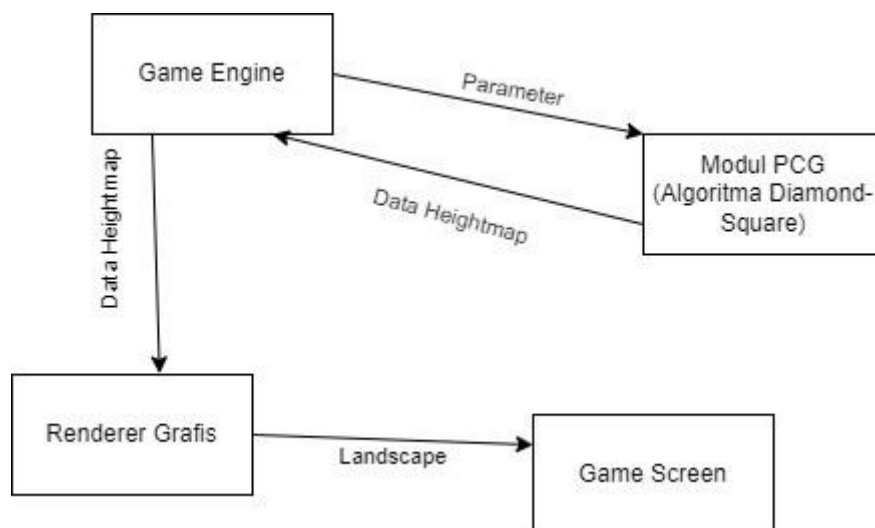
Arsitektur sistem menggambarkan struktur keseluruhan dari sistem PCG, termasuk komponen-komponen utama dan hubungan antar mereka. Arsitektur sistem untuk implementasi *Procedural Content Generation* (PCG) dalam permainan *Astro Shooter* menggunakan algoritma *Diamond-Square* dalam pengembangan game *Astro Shooter* terdiri dari beberapa komponen utama:

1. ***Game Engine Astro Shooter***: Komponen utama yang mengendalikan logika permainan, tampilan grafis, dan interaksi pemain. Komponen ini juga bertanggung jawab atas pemrosesan *input*, pengaturan logika permainan, dan *rendering* grafis. *Game Engine* yang dipakai disini adalah Unreal Engine.
2. ***Modul Procedural Content Generation***: Komponen yang bertanggung jawab atas generasi terrain secara dinamis menggunakan algoritma *Diamond-Square*. Modul ini terhubung dengan *game engine* untuk menerima parameter dan menghasilkan lanskap sesuai kebutuhan permainan. Dalam hal ini, modul berupa *Actor* dalam Unreal Engine.
3. ***Algoritma Diamond-Square***: Algoritma yang digunakan untuk menghasilkan detail lanskap dengan cara iteratif berdasarkan koordinat titik-titik dalam peta. Terdiri dari langkah-langkah pembagian segitiga dan penyempurnaan nilai ketinggian berdasarkan nilai rata-rata dari titik sudut segitiga.
4. ***Pengaturan Parameter***: Modul atau sistem yang memungkinkan pengaturan parameter untuk generasi lanskap, seperti ukuran peta, tingkat detail, dan variasi.

5. **Renderer Grafis:** Komponen yang bertanggung jawab atas visualisasi lanskap yang dihasilkan oleh modul PCG ke dalam tampilan grafis yang dapat ditampilkan oleh permainan.

Tiap komponen ini berinteraksi dalam satu sistem:

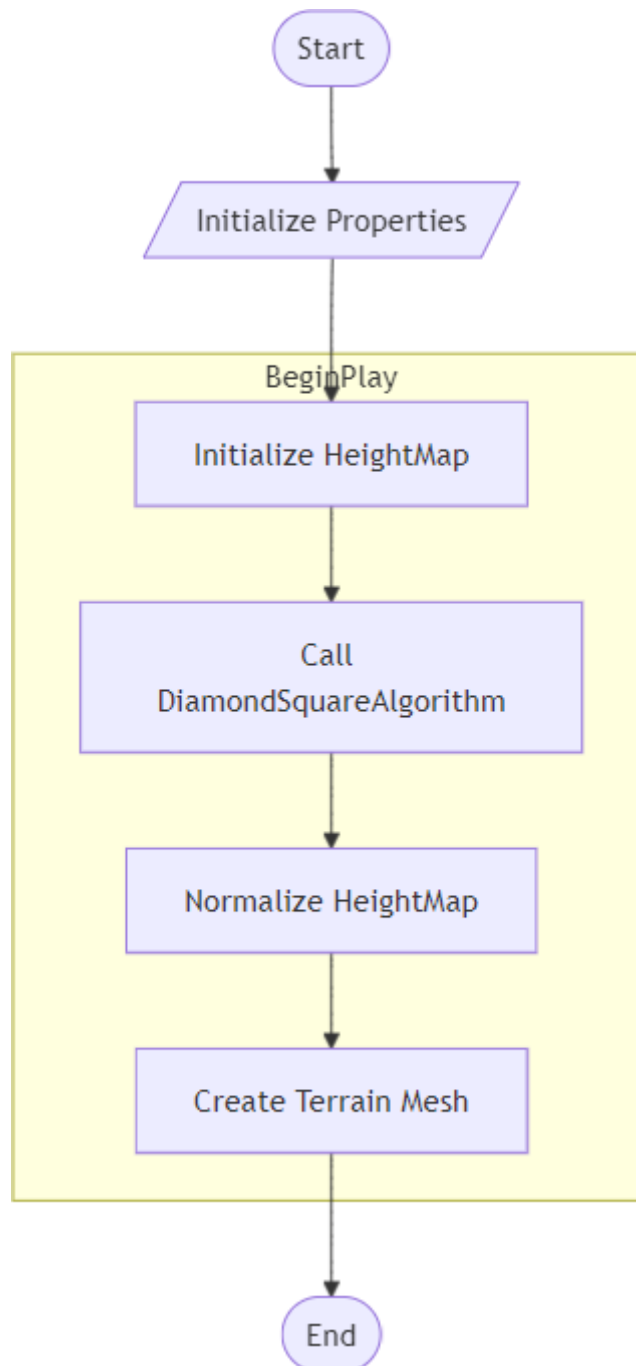
- **Game Engine Astro Shooter mengirimkan parameter ke Modul PCG:** *Game engine* memberikan parameter seperti ukuran peta dan tingkat detail *landscape* yang diinginkan ke modul PCG.
- **Modul PCG menggunakan Algoritma *Diamond-Square* untuk menghasilkan *Landscape*:** Berdasarkan parameter yang diterima, modul PCG menggunakan algoritma *Diamond-Square* untuk menghasilkan detail lanskap secara dinamis.
- **Lanskap yang dihasilkan diintegrasikan kembali ke *Game Engine*:** Setelah lanskap selesai di-*generate*, data ketinggian (*heightmap*) atau representasi lanskap lainnya dikirimkan kembali ke game engine untuk integrasi dengan logika permainan dan *rendering* grafis.
- **Renderer Grafis menampilkan Lanskap ke Layar:** *Renderer* grafis menerima data landscape yang dihasilkan dan menampilkan visualisasi yang sesuai ke layar permainan.



Gambar 3.1 Diagram Arsitektur Sistem

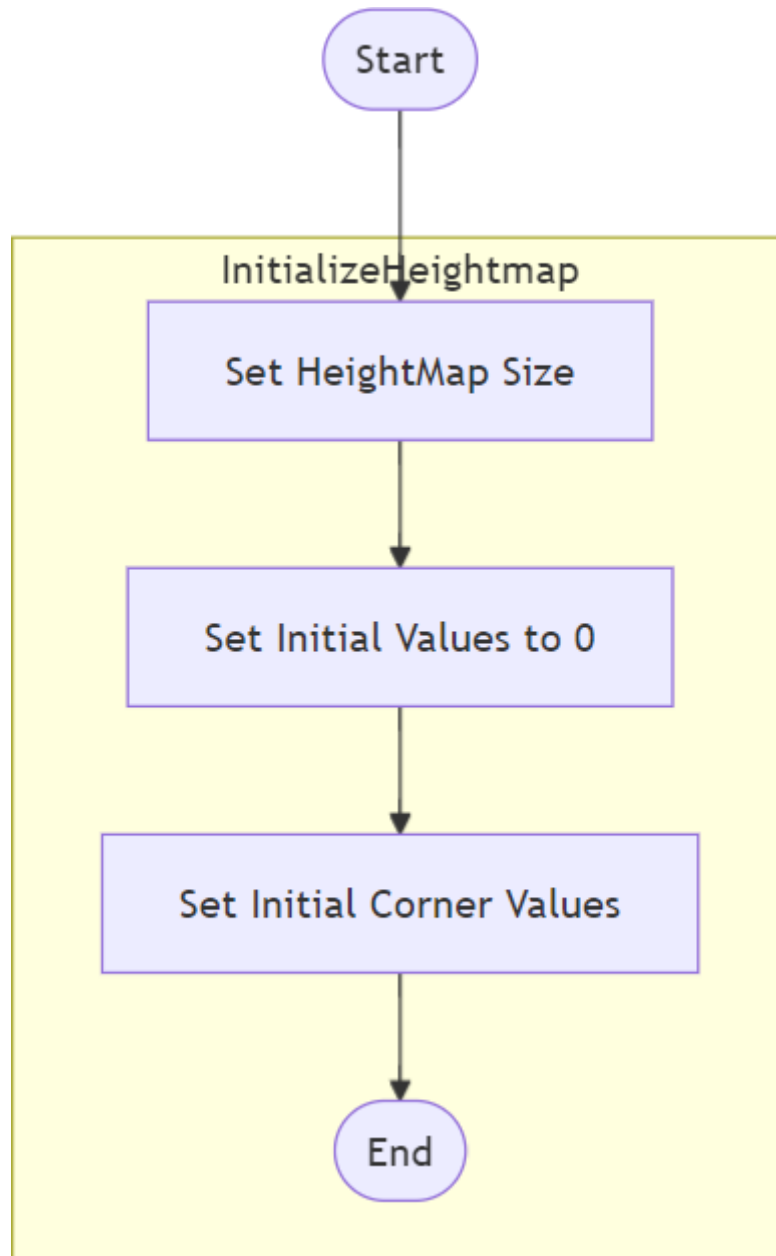
3.2.2. Flowchart code ADiamond-Square

Code untuk mengimplementasikan PCG pada game *Astro Shooter* (*ADiamondSquare*) direpresentasikan dengan *flowchart* gambar 3.2.



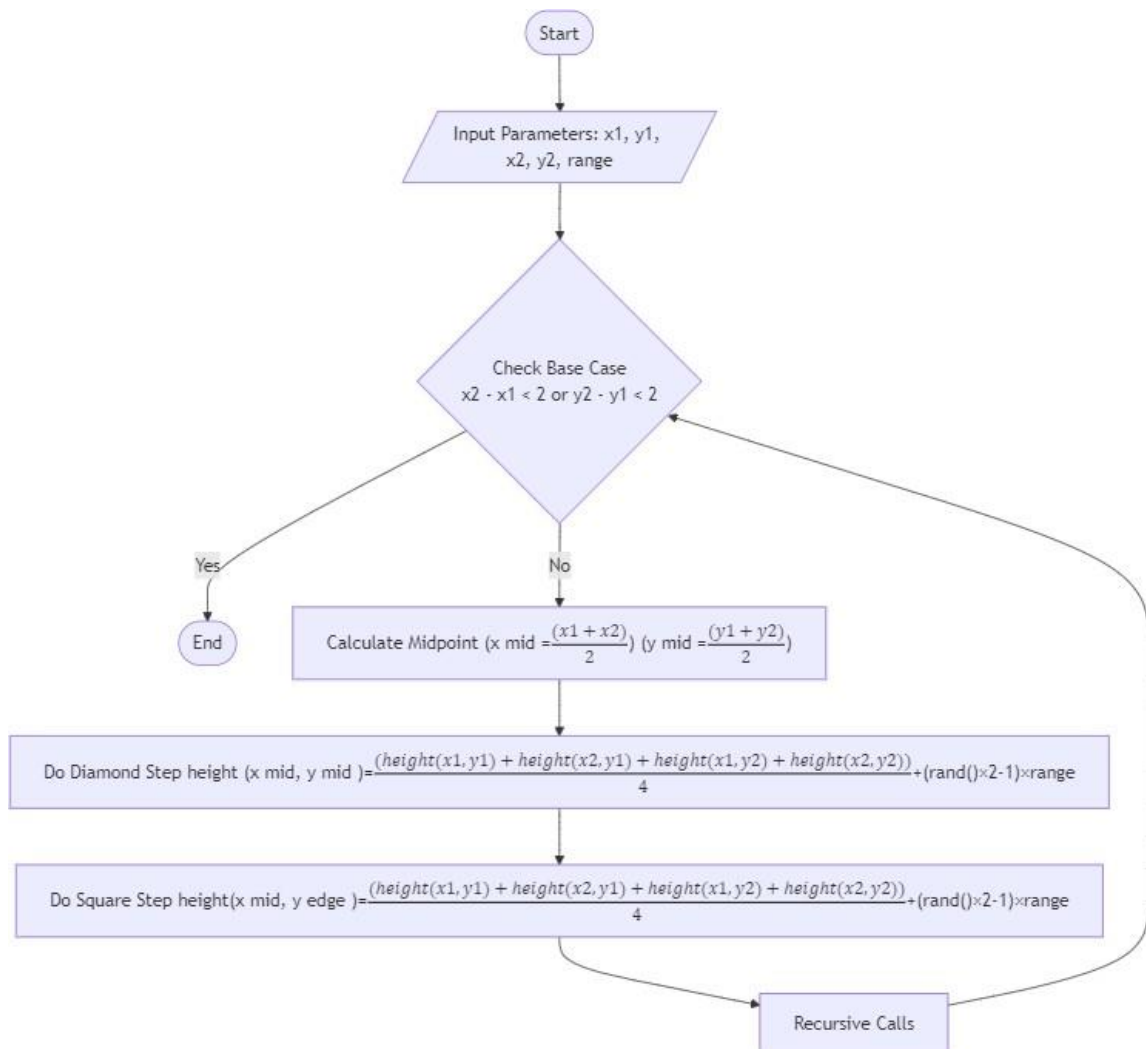
Gambar 3.2 Diagram *code* implementasi algoritma *Diamond-Square*

Flowchart Gambar 3.3 menunjukkan fungsi *Initialize HeightMap* pada flowchart *ADiamondSquare* yang sebelumnya.



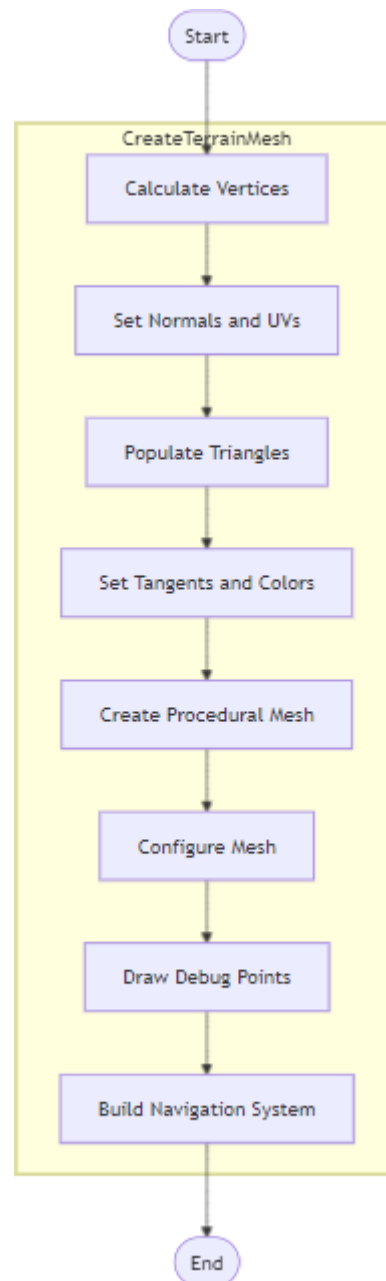
Gambar 3.3 Diagram fungsi *Initialize Height Map*

Algoritma *Diamond-Square* pada code *ADiamondSquare* direpresentasikan dengan *flowchart* berikut ini, dengan formula perhitungannya.



Gambar 3.4 Flowchart Algoritma *Diamond-Square*

Terakhir, *code* akan menghasilkan *mesh terrain* yang telah dibeikan kontur dengan algoritma *Diamond-Square*, seperti yang terlihat pada *flowchart* gambar 3.5.



Gambar 3.5 Diagram fungsi *Create Terrain Mesh*

BAB 4

IMPLEMENTASI DAN PENGUJIAN

4.1. Implementasi Sistem PCG

Pada bagian ini, akan dijelaskan langkah-langkah implementasi sistem PCG menggunakan algoritma *Diamond-Square* dalam konteks pengembangan permainan *Astro Shooter*. Implementasi sistem PCG adalah tahap di mana konsep-konsep yang telah dirancang akan diubah menjadi kode yang berfungsi dalam lingkungan pengembangan permainan *Astro Shooter*. Implementasi ini terfokus pada menghasilkan lanskap secara dinamis saat permainan dimulai. Penjelasan ini mencakup aspek-aspek seperti integrasi dengan *game engine*, pengaturan parameter, dan proses generasi *landscape*.

4.1.1. Spesifikasi Perangkat Keras dan Perangkat Lunak

Spesifikasi perangkat keras dan perangkat lunak yang digunakan dalam penelitian ini adalah sebagai berikut:

1. Prosesor Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz
2. *Memory* 16.00 GB RAM SODIMM
3. *Storage* HDD 932GB SSD 477GB
4. Sistem Operasi *Windows 11 Home Single Language* 23H2 64-bit
5. *Unreal Engine* 5.3.2
6. *Microsoft Visual Studio Community* 2022 17.9.6

4.1.2. Implementasi Algoritma *Diamond-Square*

Algoritma *Diamond-Square* digunakan untuk menghasilkan *terrain* secara procedural dengan memanfaatkan sebuah *grid* titik-titik (*grid heightmap*) dan melakukan iterasi untuk mengisi nilai ketinggian pada setiap titik berdasarkan pola *Diamond-Square* dengan membagi *grid* menjadi *sub-grid* yang lebih kecil secara berulang, sambil menambahkan variasi nilai ketinggian untuk menciptakan *terrain* yang realistis. Proses ini dilakukan melalui dua langkah utama: langkah *Diamond* dan langkah *Square*. Dalam implementasi algoritma *Diamond-Square*, langkah-langkah yang diperlukan melibatkan detail teknis tentang bagaimana

algoritma ini diintegrasikan ke dalam sistem PCG dan diterapkan untuk menghasilkan lanskap dalam permainan *Astro Shooter*.

1. Struktur Kelas *ADiamondSquare*

Kelas *ADiamondSquare* adalah kelas utama yang bertanggung jawab atas pembuatan *terrain* menggunakan algoritma *Diamond-Square*. Dalam konstruktor kelas ini, komponen *TerrainMesh* dibuat sebagai bagian dari objek ini dan digunakan sebagai *root component*. Ini memungkinkan objek *ADiamondSquare* untuk memiliki representasi visual dalam permainan. Selain itu, parameter seperti ukuran *terrain* (*TerrainSize*), kekasaran (*Roughness*), dan skala tinggi (*HeightScaleFactor*) diinisialisasi di sini. Ukuran *terrain* menentukan dimensi dari *grid* tinggi-tinggi yang akan dibangkitkan, sedangkan kekasaran mengontrol seberapa banyak variasi ketinggian yang diizinkan di setiap iterasi algoritma. Skala tinggi mengontrol ketinggian maksimum yang akan dicapai oleh ketinggian *terrain*.

```
#include "DiamondSquare.h"
#include "KismetProceduralMeshLibrary.h"
#include "NavigationSystem.h"

// Sets default values
ADiamondSquare::ADiamondSquare()
{
    PrimaryActorTick.bCanEverTick = true;

    TerrainMesh =
CreateDefaultSubobject<UProceduralMeshComponent>(TEXT("TerrainMesh"));
    RootComponent = TerrainMesh;

    TerrainSize = 384;
    Roughness = 0.6;
    HeightScaleFactor = 0.1f; // Adjust this value to control terrain height
}
```

Gambar 4.1 Inisialisasi *TerrainMesh* pada *ADiamondSquare*

2. Metode BeginPlay

Metode *BeginPlay* dipanggil saat permainan dimulai atau objek *ADiamondSquare* di-spawn di dalam permainan. Pada tahap ini, *HeightMap* diinisialisasi sebagai *array 2D* yang merepresentasikan ketinggian setiap titik dalam terrain. Metode *InitializeHeightMap* kemudian dipanggil untuk mengisi *HeightMap* dengan nilai-nilai awal, dan kemudian algoritma *Diamond-Square* dijalankan menggunakan metode *DiamondSquareAlgorithm* untuk menghasilkan ketinggian yang realistis untuk terrain. Setelah ketinggian ditentukan, *mesh terrain* dibuat menggunakan metode *CreateTerrainMesh*.

```
void ADiamondSquare::BeginPlay()
{
    Super::BeginPlay();

    HeightMap.Init(TArray<float>(), TerrainSize);

    InitializeHeightMap();

    DiamondSquareAlgorithm(0, 0, TerrainSize - 1, TerrainSize - 1, TerrainSize *
Roughness);

    CreateTerrainMesh();
}
```

Gambar 4.2 Method BeginPlay

3. Inisialisasi Height Map

Metode *InitializeHeightMap* bertanggung jawab untuk menginisialisasi *HeightMap*. Ini dilakukan dengan mengatur ukuran *array* sesuai dengan *TerrainSize* dan mengisi nilai-nilai awal untuk titik-titik sudut terrain dengan nilai acak antara 0 dan 1. Inisialisasi ini memberikan fondasi untuk algoritma *Diamond-Square* memanipulasi nilai-nilai ini untuk menciptakan variasi ketinggian dan membuat *mesh terrain* menggunakan data yang dihasilkan.

```

void ADiamondSquare::InitializeHeightMap()
{
    HeightMap.SetNum(TerrainSize);

    for (int32 i = 0; i < TerrainSize; i++)
    {
        HeightMap[i].SetNum(TerrainSize);

        for (int32 j = 0; j < TerrainSize; j++)
        {
            HeightMap[i][j] = 0.0f;
        }
    }

    HeightMap[0][0] = FMath::FRand();
    HeightMap[0][TerrainSize - 1] = FMath::FRand();
    HeightMap[TerrainSize - 1][0] = FMath::FRand();
    HeightMap[TerrainSize - 1][TerrainSize - 1] = FMath::FRand();
}

```

Gambar 4.3 Inisialisasi *HeightMap*

4. Algoritma Diamond-Square

Algoritma *Diamond-Square* adalah algoritma pembangkitan *terrain* yang menghasilkan peta ketinggian dengan variasi ketinggian yang realistis melalui pembagian dan interpolasi berulang. Algoritma ini menggunakan dua langkah utama: langkah *diamond* dan langkah *square*. Berikut penjelasan detail mengenai implementasinya:

- **Langkah *Diamond***

Pada langkah *Diamond*, titik tengah dari setiap *diamond* (segiempat yang dibentuk oleh empat titik) dihitung sebagai rata-rata ketinggian keempat titik sudut ditambah nilai acak yang dikalikan dengan faktor kekasaran (*roughness*).

```

float ADiamondSquare::GetMidpoint(int32 x1, int32 y1, int32 x2, int32 y2)
{
    return (HeightMap[x1][y1] + HeightMap[x2][y1] + HeightMap[x1][y2] +
    HeightMap[x2][y2]) / 4.0;
}

void ADiamondSquare::DiamondSquareAlgorithm(int32 x1, int32 y1, int32 x2, int32
y2, float range)
{
    if ((x2 - x1) < 2 || (y2 - y1) < 2)
        return;

    int32 xMid = (x1 + x2) / 2;
    int32 yMid = (y1 + y2) / 2;

    // Diamond step
    HeightMap[xMid][yMid] = GetMidpoint(x1, y1, x2, y2) + (FMath::FRand() * 2 -
1) * range * HeightScaleFactor;

```

Gambar 4.4 *Diamond step*

- **Langkah Square**

Langkah *square* dilakukan dengan menghitung ketinggian untuk titik-titik yang berada di tengah-tengah setiap sisi dari kuadrat yang telah diproses di langkah *diamond*. Ketinggian titik-titik ini dihitung sebagai rata-rata ketinggian dari empat titik yang membentuk diamond dengan nilai acak yang dikalikan dengan faktor kekasaran.

```

// Square step
HeightMap[x1][yMid] = GetMidpoint(x1, y1, x2, y2) + (FMath::FRand() * 2 - 1)
* range * HeightScaleFactor;
HeightMap[xMid][y1] = GetMidpoint(x1, y1, x2, y2) + (FMath::FRand() * 2 - 1)
* range * HeightScaleFactor;
HeightMap[x2][yMid] = GetMidpoint(x1, y1, x2, y2) + (FMath::FRand() * 2 - 1)
* range * HeightScaleFactor;
HeightMap[xMid][y2] = GetMidpoint(x1, y1, x2, y2) + (FMath::FRand() * 2 - 1)
* range * HeightScaleFactor;

```

Gambar 4.5 *Square step*

- **Pembagian Rekursif**

Algoritma ini memanggil dirinya sendiri secara rekursif untuk setiap kuadran yang terbentuk dari langkah-langkah *diamond* dan *square*. Pada setiap iterasi rekursif, ukuran kuadran berkurang setengahnya dan nilai range juga dikurangi dengan mengalikan dengan faktor kekasaran (*roughness*).

Ini menghasilkan variasi ketinggian yang semakin halus seiring dengan pembagian yang lebih kecil.

```
// Recursive calls for each quadrant
DiamondSquareAlgorithm(x1, y1, xMid, yMid, range * Roughness);
DiamondSquareAlgorithm(x1, yMid, xMid, y2, range * Roughness);
DiamondSquareAlgorithm(xMid, y1, x2, yMid, range * Roughness);
DiamondSquareAlgorithm(xMid, yMid, x2, y2, range * Roughness);
}
```

Gambar 4.6 Pembagian Rekursif

5. Membuat Mesh Terrain

Metode *CreateTerrainMesh* bertanggung jawab untuk menghasilkan mesh terrain berdasarkan *HeightMap*. Ini dilakukan dengan menciptakan serangkaian verteks berdasarkan ketinggian dari *HeightMap*, kemudian triangulasi *grid* untuk membentuk permukaan terrain. *Mesh* ini kemudian diatur dengan normal dan koordinat tekstur untuk visualisasi yang akurat. Selain itu, properti kolisi dan navigasi diberikan agar objek-objek dalam permainan dapat berinteraksi dengan *terrain* dengan benar.

```
void ADiamondSquare::CreateTerrainMesh()
{
    UE_LOG(LogTemp, Warning, TEXT("CreateTerrainMesh called!"));

    if (TerrainMesh)
    {
        TArray<FVector> Vertices;
        TArray<int32> Triangles;
        TArray<FVector> Normals;
        TArray<FVector2D> UV0;
        TArray<FLinearColor> VertexColors;
        TArray<FProcMeshTangent> Tangents;

        float ScaleFactor = 10.0f;

        for (int32 y = 0; y < TerrainSize; y++)
        {
            for (int32 x = 0; x < TerrainSize; x++)
            {
                FVector VertexLocation = FVector(x * ScaleFactor, y *
                ScaleFactor, HeightMap[x][y] * ScaleFactor);
                Vertices.Add(VertexLocation);
            }
        }
    }
}
```

```

        Normals.Add(FVector(0.0f, 0.0f, 1.0f));

        float U = static_cast<float>(x) /
static_cast<float>(TerrainSize - 1);
        float V = static_cast<float>(y) /
static_cast<float>(TerrainSize - 1);
        UV0.Add(FVector2D(U, V));
    }
}

for (int32 y = 0; y < TerrainSize - 1; y++)
{
    for (int32 x = 0; x < TerrainSize - 1; x++)
    {
        int32 VertexIndex0 = x + y * TerrainSize;
        int32 VertexIndex1 = x + (y + 1) * TerrainSize;
        int32 VertexIndex2 = (x + 1) + y * TerrainSize;
        int32 VertexIndex3 = (x + 1) + (y + 1) * TerrainSize;

        Triangles.Add(VertexIndex0);
        Triangles.Add(VertexIndex1);
        Triangles.Add(VertexIndex2);

        Triangles.Add(VertexIndex2);
        Triangles.Add(VertexIndex1);
        Triangles.Add(VertexIndex3);
    }
}

Tangents.Init(FProcMeshTangent(0.0f, 0.0f, 1.0f),
Vertices.Num());
VertexColors.Init(FLinearColor(1.0f, 1.0f, 1.0f, 1.0f),
Vertices.Num());

TerrainMesh->CreateMeshSection_LinearColor(0, Vertices,
Triangles, Normals, UV0, VertexColors, Tangents, true);
TerrainMesh->SetMaterial(0, TerrainMaterial);
TerrainMesh-
>SetCollisionEnabled(ECollisionEnabled::QueryAndPhysics);
TerrainMesh-
>SetCollisionObjectType(ECollisionChannel::ECC_PhysicsBody);
TerrainMesh-
>SetCollisionResponseToAllChannels(ECollisionResponse::ECR_Block);
TerrainMesh->SetGenerateOverlapEvents(true);
TerrainMesh->SetCanEverAffectNavigation(true);
}

UWorld* World = GetWorld();

```

```

if (World)
{
    UNavigationSystemV1* NavSystem =
FNavigationSystem::GetCurrent<UNavigationSystemV1>(World);
    if (NavSystem)
    {
        NavSystem->Build();
    }
}

if (!TerrainMesh)
{
    UE_LOG(LogTemp, Warning, TEXT("TerrainMesh is not valid!"));
    return;
}
}

```

Gambar 4.7 Membuat *Mesh Terrain*

4.2. Pengujian

Setelah implementasi selesai, langkah berikutnya adalah pengujian untuk memastikan bahwa *terrain* yang dihasilkan sesuai dengan harapan dan dapat digunakan dalam konteks permainan *Astro Shooter*.

4.2.1. Pengujian Fungsional

Pengujian fungsional dilakukan untuk memastikan bahwa setiap komponen bekerja dengan benar:

- **Validasi Terrain:** Memastikan *terrain* yang dihasilkan memiliki ketinggian dan kontur yang sesuai dengan parameter yang ditentukan.
- **Visualisasi Mesh:** Memastikan *mesh terrain* terlihat dengan benar dalam permainan, tanpa artefak visual atau kesalahan rendering.
- **Kolisi dan Navigasi:** Memastikan bahwa objek-objek dalam permainan dapat berinteraksi dengan *terrain* dengan benar, termasuk kolisi dan navigasi AI.

4.2.2. Pengujian Kinerja

Pengujian kinerja bertujuan untuk memastikan bahwa *terrain* dapat dihasilkan dan di-render dalam waktu yang wajar tanpa mempengaruhi kinerja permainan secara signifikan:

- **Waktu Pembangkitan:** Mengukur waktu yang dibutuhkan untuk menghasilkan *terrain* menggunakan algoritma *Diamond-Square*.
- **Framerate:** Memastikan permainan tetap berjalan dengan framerate yang stabil meskipun *terrain* kompleks.

4.2.3. Hasil Pengujian

Hasil dari pengujian-pengujian tersebut akan dirangkum dalam tabel 4.1.

Tabel 4.1 Hasil Pengujian Algoritma *Diamond-Square*

Jenis Pengujian	Hasil	Catatan
Validasi <i>Terrain</i>	Berhasil	<i>Terrain</i> sesuai dengan parameter yang ditentukan
Visualisasi <i>Mesh</i>	Berhasil	<i>Mesh</i> terlihat baik tanpa artefak visual
Kolisi	Berhasil	Karakter tidak menembus <i>terrain</i>
Navigasi	Gagal	<i>Terrain</i> tidak dapat dinavigasi AI
Waktu Pembangkitan	886 - 724 ms	Waktu pembangkitan cukup cepat
<i>Framerate</i>	Stabil (~ 26 FPS)	Tidak ada penurunan <i>framerate</i> yang signifikan
Monotonitas	Berhasil	<i>Terrain</i> menunjukkan variasi ketinggian yang wajar dan realistis

Sumber: Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer

Beberapa parameter pada tabel 4.1 di atas (Validasi *Terrain*, Visualisasi *Mesh*, Kolisi, dan Waktu Pembangkitan) diambil dari Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer Universitas Brawijaya. Penulis juga memasukkan tiga parameter lainnya, (Navigasi, *Framerate*, dan Monotonitas) yang dianggap relevan.

Analisis Hasil Pengujian

1. Validasi *Terrain*:

- **Hasil:** Berhasil.
- **Catatan:** *Terrain* yang dihasilkan sesuai dengan parameter *Roughness* dan *HeightScaleFactor* yang ditentukan, menunjukkan variasi ketinggian yang sesuai dengan harapan.

2. **Visualisasi *Mesh*:**

- **Hasil:** Berhasil.
- **Catatan:** *Mesh terrain* dirender dengan baik dalam *game* tanpa adanya artefak visual seperti lubang atau distorsi, menunjukkan bahwa koordinat UV dan tekstur diterapkan dengan benar.

3. **Kolisi:**

- **Hasil:** Berhasil.
- **Catatan:** Karakter dan objek lain dalam *game* tidak menembus *terrain*, menunjukkan bahwa sistem kolisi bekerja dengan baik.

4. **Navigasi:**

- **Hasil:** Gagal.
- **Catatan:** AI tidak dapat menavigasi *terrain*, menunjukkan bahwa sistem navigasi AI memerlukan penyesuaian agar dapat mengenali *terrain* sebagai area yang dapat dilalui.

5. **Waktu Pembangkitan:**

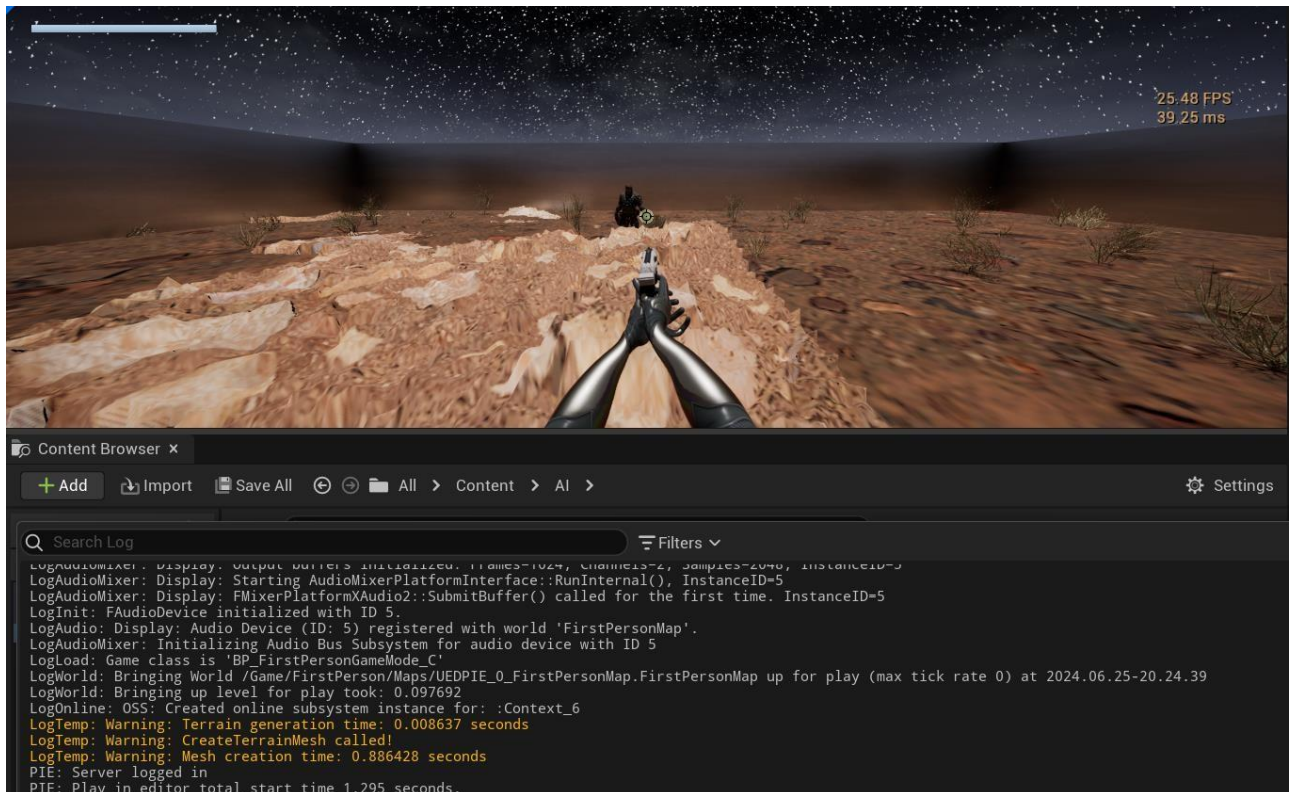
- **Hasil:** 724 - 886 ms.
- **Catatan:** Waktu pembangkitan *terrain* berada dalam kisaran yang cukup cepat, menunjukkan efisiensi algoritma Diamond-Square untuk ukuran *terrain* 384x384 dengan *Roughness* 0.6 dan *HeightScaleFactor* 0.1f.

6. **Framerate:**

- **Hasil:** Stabil (~ 26 FPS).
- **Catatan:** Framerate stabil sekitar 26 FPS setelah *terrain* dihasilkan dan dirender, meskipun ini lebih rendah dari standar 30 FPS. Hal ini mungkin memerlukan optimasi lebih lanjut untuk meningkatkan kinerja.

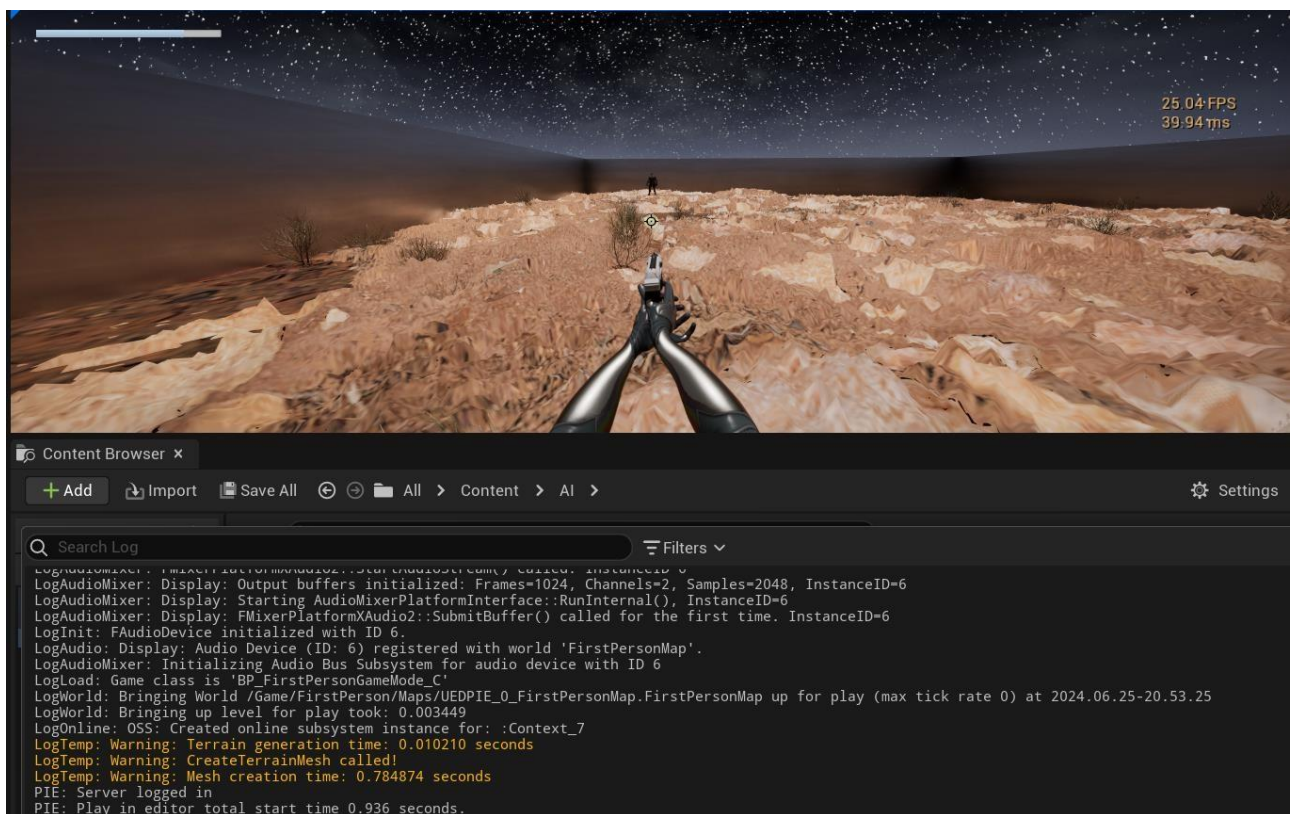
7. **Monotonitas:**

- **Hasil:** Berhasil.
- **Catatan:** *Terrain* menunjukkan variasi ketinggian yang wajar dan realistis, tidak monoton atau terlalu seragam, sesuai dengan parameter yang ditentukan



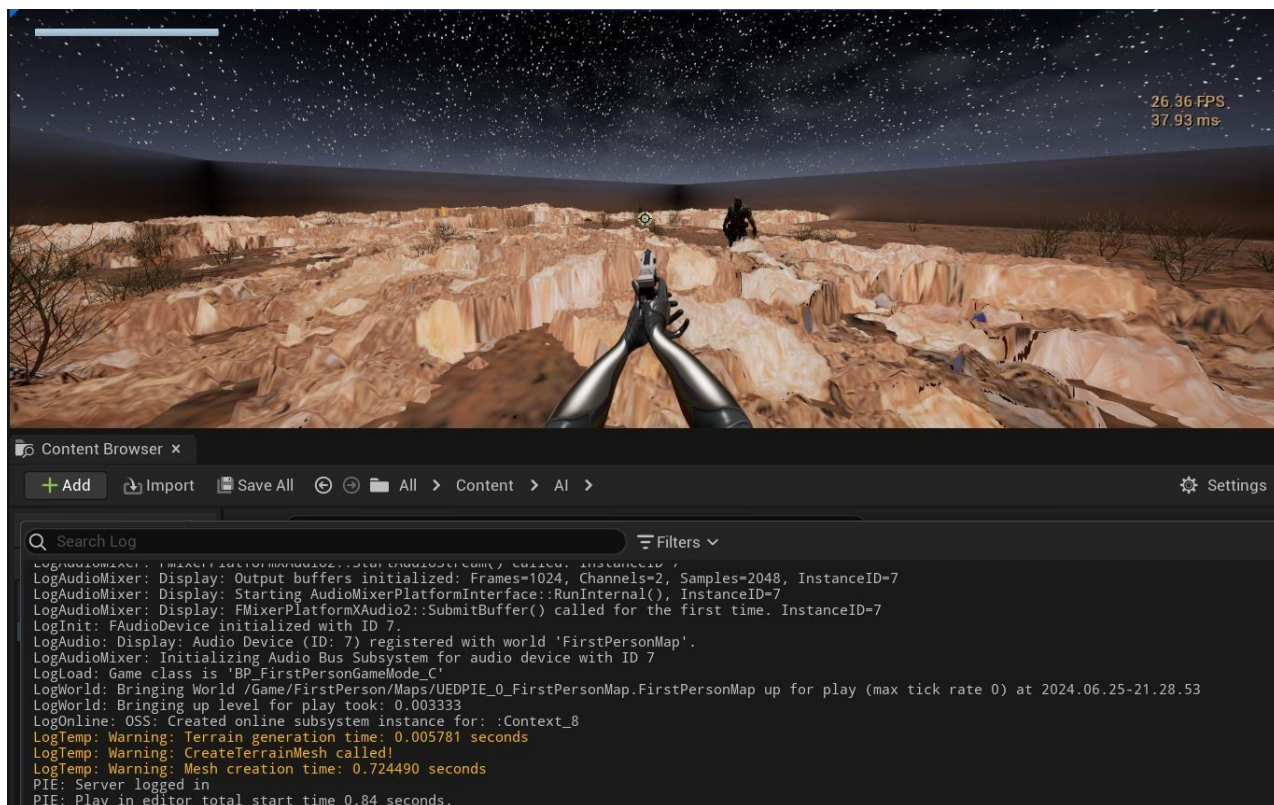
Gambar 4.8 Hasil *rendering terrain* iterasi pertama

Pada gambar 4.8 dan gambar 4.9 dapat dilihat perbedaan kenampakan *landscape* pada kedua iterasi



Gambar 4.9 Hasil *rendering terrain* iterasi kedua

Pada iterasi ketiga di bawah ini, dapat dilihat perbedaan kenampakan dengan kedua iterasi sebelumnya dan kolisi bekerja dengan baik, ditampilkan dengan karakter *player* dan *AI* tidak dapat menebus *terrain mesh* yang dihasilkan (kenampakan yang berwarna terang).



Gambar 4.10 Hasil *rendering terrain* iterasi ketiga

Melalui visualisasi *heightmap* yang terlihat pada gambar - gambar hasil *rendering* di atas, dapat dilihat bahwa *terrain* yang dihasilkan sesuai dengan parameter dan tidak ada kenampakan artefak visual (kejanggalan pada *terrain*), juga dapat dilihat setiap iterasi baru menghasilkan bentuk *terrain* yang berbeda satu sama lainnya. Dapat juga dilihat pada ketiga gambar di atas kecepatan pembuatan *terrain* (*mesh creation time*) dan juga *FPS counter* pada sisi kanan gambar.

Tabel 4.2 Rentang Waktu Pembangkitan & *Framerate*

Iterasi	Waktu Pembangkitan	<i>Framerate</i>
Iterasi 1	886 ms	25.48fps
Iterasi 2	784 ms	25.04fps
Iterasi 3	724 ms	26.36fps

BAB 5

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Penelitian ini telah berhasil mengimplementasikan algoritma *Diamond-Square* untuk generasi *terrain* secara prosedural dalam game *Astro Shooter*. Berikut adalah beberapa kesimpulan utama yang dapat diambil dari hasil implementasi dan pengujian:

- **Implementasi Algoritma *Diamond-Square*:** Algoritma *Diamond-Square* berhasil diterapkan untuk menghasilkan *terrain* dengan variasi ketinggian yang realistis dan sesuai dengan parameter ***TerrainSize***, ***Roughness***, dan ***HeightScaleFactor*** yang telah ditentukan.
- **Monotonitas *Terrain*:** *Terrain* yang dihasilkan menunjukkan variasi ketinggian yang wajar dan realistis tanpa menjadi terlalu monoton atau terlalu acak, menunjukkan keberhasilan algoritma dalam menghasilkan *terrain* yang baik.

Secara keseluruhan, penelitian ini menunjukkan bahwa algoritma *Diamond-Square* efektif untuk generasi *terrain* prosedural dalam game, namun ada aspek yang memerlukan perbaikan dan optimasi lebih lanjut.

5.2. Saran

Berdasarkan hasil implementasi dan pengujian, serta *feedback* dari teman-teman yang mencoba game ini, berikut adalah beberapa saran yang mereka berikan untuk pengembangan lebih lanjut:

- **Perbaikan Sistem Navigasi AI:** Meskipun karakter AI sudah dapat menavigasi medan permainan dengan penambahan bidang tanah datar, masih diperlukan penyesuaian lebih lanjut pada sistem navigasi AI agar dapat mengenali dan menavigasi *terrain* yang dihasilkan oleh algoritma *Diamond-Square*. Ini dapat melibatkan penyesuaian pada *setting* navigasi atau penggunaan algoritma tambahan untuk meningkatkan kemampuan navigasi AI.

- **Optimasi Kinerja:** Saat pengujian, *framerate* permainan tetap stabil di sekitar 26 FPS. Hal ini sudah terbilang bagus akan tetapi, optimasi lebih lanjut diperlukan untuk mencapai *framerate* yang lebih tinggi dan memastikan pengalaman bermain yang lebih lancar. Ini bisa melibatkan optimasi *rendering*, penggunaan teknik LOD (*Level of Detail*), atau peningkatan efisiensi algoritma.
- **Pengembangan Fitur Tambahan:** Salah satu *feedback* dari teman – teman yang mencoba *game* ini adalah penambahan fitur-fitur seperti penyesuaian parameter *terrain* secara *real-time*, variasi biome, atau integrasi dengan sistem cuaca dinamis yang dapat meningkatkan kompleksitas dan keindahan *terrain* yang dihasilkan dan memberikan pengalaman bermain yang lebih kaya dan menarik.

DAFTAR PUSTAKA

- Archer, T. 2011. *Procedurally Generating Terrain*, Midwest Instruction and Computing Symposium. Morningside College.
- Belhadj, F., & Audibert, P. 2005. Modeling landscapes with ridges and rivers. *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pp. 255-258. <https://doi.org/10.1145/1101616.1101648>
- De Lima, E. S., Feijó, B., & Furtado, A. L. 2019. Procedural Generation of Quests for Games Using Genetic Algorithms and Automated Planning. *Proceedings of 2019 18th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pp. 495-504.
- Fikrie, I. 2017. Procedural generation untuk terrain modelling menggunakan reverse midpoint displacement. Institut Pertanian Bogor.
- Fournier, A., Fussell, D., Carpenter, L. 1982. Computer rendering of stochastic models. *Communications of the ACM* 25(6): 371-384. <https://doi.org/10.1145/358523.358553>
- Huang, S., & Li, X.-X. 2010, January. Improved random midpoint-displacement method for natural terrain simulation. *IEEE 2010 Third International Conference on Information and Computing Science (ICIC)*, pp. 255-258. <https://doi.org/10.1109/icic.2010.71>.
- Jilesen, J., Kuo, J., & Lien, F. 2011, December. Three-dimensional midpoint displacement algorithm for the generation of fractal porous media. *Computers & Geosciences* 46: 164-173. <https://doi.org/10.1016/j.cageo.2011.12.002>.
- Kiting, I. H. 2020. Implementasi procedural level generation pada aplikasi game pyramid exploration. Universitas Atma Jaya Yogyakarta.
- Putri, C.S., Jonemaro, E.M., & Akbar, M.A. 2019. Penerapan *procedural content generation* pada pembangkitan *level* gim maze heksagonal. *Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer* 3(9): 8563-8571.
- Short, T. X., & Adams, T. 2017. *Procedural Generation in Game Design*. 2nd Edition. Taylor & Francis, CRC Press: Boca Raton. <https://doi.org/10.1201/9781315156378>.
- Togelius, J., Champandard, A.J., Lanzi, P.L., Mateas, M., Paiva, A., Preuss, M & Stanley, E.S. 2013. Procedural Content Generation: Goals, Challenges and Actionable Steps. *Artificial and Computational Intelligence in Games* 6: 61-75. <http://dx.doi.org/10.4230/DFU.Vol6.12191.61>.
- Utomo, A.P. 2015. Rancang bangun modul *procedural content generation* pada *procedural content* permainan friend and foe. Institut Teknologi Sepuluh Nopember.