BAB I

PENDAHULUAN

A. Latar Belakang

Perkembangan teknologi digital yang pesat telah membawa transformasi signifikan dalam berbagai sektor, termasuk industri hiburan dan permainan digital (*game*). Industri *game* kini menjadi salah satu sektor kreatif dengan pertumbuhan tercepat secara global. *Game* tidak hanya menjadi media hiburan, tetapi juga sarana edukasi, ekspresi seni, hingga simulasi sosial. Perkembangan ini didorong oleh kemajuan teknologi grafis, kecerdasan buatan, serta meningkatnya daya komputasi perangkat keras. Salah satu genre yang menonjol dalam beberapa tahun terakhir adalah *game* aksi *survival*. Genre ini menawarkan pengalaman interaktif di mana pemain dituntut untuk bertahan hidup dalam dunia terbuka (*open world*) yang dinamis, berbahaya, dan penuh tantangan. Elemen seperti eksplorasi, perolehan sumber daya, pembangunan tempat tinggal, hingga interaksi dengan lingkungan dan karakter lain, menjadikan genre ini sangat diminati oleh berbagai kalangan.

Namun, kenyataannya, pengembangan dunia permainan yang luas dan dinamis bukanlah hal yang mudah dilakukan. Dalam praktiknya, pembuatan lingkungan secara manual memerlukan banyak waktu, tenaga, dan sumber daya manusia, yang pada skala besar bisa menjadi tidak efisien. Selain itu, keterbatasan waktu dan biaya sering mengakibatkan elemen dunia yang dibuat menjadi berulang dan kurang bervariasi. Sementara itu, penggunaan pendekatan acak atau *random* dalam pembuatan konten sering kali menghasilkan dunia yang tidak terstruktur

dengan baik dan kurang sesuai dengan konteks desain permainan. Kondisi ini menciptakan kesenjangan antara harapan akan dunia permainan yang menarik dan kenyataan pengembangannya di lapangan.

Melihat permasalahan tersebut, diperlukan pendekatan alternatif yang dapat membantu pengembang dalam membangun dunia permainan yang luas tanpa sepenuhnya mengandalkan desain manual. Salah satu pendekatan yang banyak digunakan dalam pengembangan konten digital adalah *Procedural Generation* (PG). PG adalah teknik yang memungkinkan pembuatan data atau konten secara algoritmik dengan parameter tertentu, sehingga dapat mengurangi ketergantungan terhadap aset yang dibuat satu per satu. Dengan demikian, metode ini memberikan kemungkinan untuk menghadirkan variasi konten yang lebih banyak tanpa menambah beban kerja secara proporsional.

Penggunaan PG telah diadopsi dalam sejumlah *game* komersial seperti Minecraft dan No Man's Sky untuk membangun dunia permainan yang luas dan bervariasi. Namun, dalam *genre* aksi *survival*, penerapannya masih menghadirkan tantangan, terutama dalam menjaga struktur dunia agar tetap logis, terarah, dan mendukung jalannya permainan. Selain itu, dibutuhkan keseimbangan antara keragaman elemen yang dihasilkan dan konsistensi desain agar tidak mengganggu pengalaman bermain.

Penelitian ini menerapkan PG pada empat elemen utama dalam dunia *game*, yaitu peta (*map*), permukaan medan (*terrain*), penempatan objek (*object spawner*), dan pemunculan musuh (*enemy spawner*), dengan memanfaatkan algoritma *Perlin Noise*. Algoritma ini dipilih karena kemampuannya menghasilkan pola distribusi

yang lebih alami dan berkesinambungan. Melalui pendekatan ini, penelitian berfokus pada studi kasus pengembangan dunia *game* aksi *survival* High on Survival.

B. Rumusan Masalah

Berdasarkan latar belakang di atas, penelitian ini berusaha menjawab permasalahan utama: Bagaimana cara mengimplementasikan *Procedural Generation* dalam pengembangan *game* aksi *survival* High on Survival?

C. Tujuan Penelitian

Penelitian ini bertujuan untuk mengimplementasikan *Procedural Generation* menggunakan metode *Perlin Noise* dalam pengembangan *game* aksi *survival* High on Survival.

D. Batasan Masalah

Agar penelitian ini tetap terarah dan fokus, terdapat beberapa batasan yang penulis tetapkan sebagai berikut:

1. Algoritma *Perlin Noise* merupakan satu-satunya metode *Procedural Generation* yang digunakan dalam penelitian ini untuk menghasilkan elemen-elemen dunia dalam *game* aksi *survival* High on Survival. Metode lain seperti *Cellular Automata* maupun *Wave Function Collapse* tidak termasuk dalam ruang lingkup pembahasan.
2. Fokus penelitian terbatas pada penerapan *Procedural Generation* untuk peta (*map*), permukaan medan (*terrain*), penempatan objek (*object spawner*), dan pemunculan musuh (*enemy spaw*ner) serta sistem chunk streamline. Aspek lain dalam pengembangan *game* seperti kecerdasan buatan (AI),

mekanik permainan, jaringan multiplayer, optimasi grafis, desain karakter, dan sistem fisika tidak menjadi topik kajian.

3. Sistem penyimpanan data yang digunakan tidak mendukung sinkronisasi antar perangkat atau pemindahan data lintas platform, sehingga fitur tersebut tidak termasuk dalam ruang lingkup penelitian ini.
4. Aset-aset yang digunakan dalam pengembangan *game* sebagian besar berasal dari *Unity Asset Store* dan hanya dimodifikasi seperlunya untuk keperluan penelitian, sehingga tidak dilakukan pembuatan aset orisinal secara keseluruhan.
5. Pengujian dilakukan secara terbatas dengan metode *Blackbox* dan *Whitebox*, yang meliputi pengujian dampak variasi *map* permainan berdasarkan *seed* yang berbeda pada *game* High on Survival.

E. Manfaat Penelitian

Adapun manfaat yang diharapkan dari penelitian ini adalah sebagai berikut:

1. Memberikan gambaran teknis awal bagi pengembang *game* dalam menerapkan *Procedural Generation* (PG) menggunakan algoritma *Perlin Noise* pada *game* aksi *survival*, khususnya dalam aspek pembuatan peta, *terrain*, penempatan objek, dan musuh.
2. Menambah wawasan mengenai pemanfaatan algoritma PG dalam mendukung proses perancangan dunia permainan yang lebih variatif, yang dapat menjadi pertimbangan dalam waktu dan sumber daya pengembangan.

BAB II

TINJAUAN PUSTAKA

A. *Game* Aksi *Survival*

Game aksi *survival* adalah salah satu genre yang menuntut ketahanan, strategi, dan adaptasi dari pemain dalam menghadapi berbagai tantangan di dalam dunia permainan. Dalam genre ini, pemain biasanya d*item*patkan dalam lingkungan yang keras, tidak bersahabat, atau penuh bahaya, di mana mereka harus bertahan hidup dengan memanfaatkan sumber daya yang tersedia. Elemen utama yang sering d*item*ui dalam *game* aksi *survival* meliputi eksplorasi, manajemen sumber daya, pertarungan, serta interaksi dinamis dengan lingkungan.



Gambar 2. 1 Poster No Man's Sky**.**
(*Sumber: https://www.playstation.com/en-id/games/no-mans-sky/*)

Salah satu aspek kunci dalam *game* aksi *survival* adalah dunia permainan yang tidak dapat diprediksi dan selalu menawarkan tantangan baru. Oleh karena itu, banyak pengembang *game survival* mengadopsi *Procedural Generation* (PG) untuk

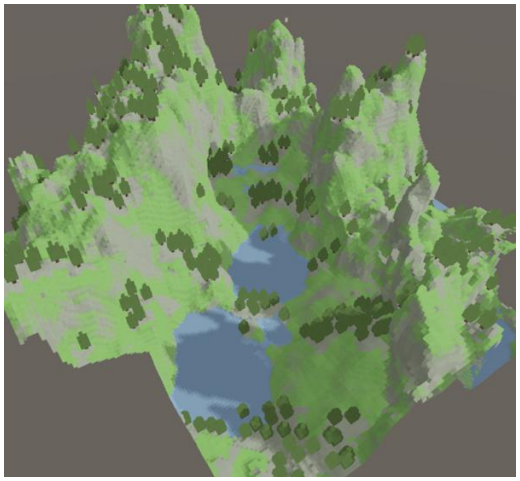
menciptakan lingkungan yang berubah-ubah dan tidak monoton, sehingga meningkatkan pengalaman bermain dan menjaga keterlibatan pemain dalam jangka panjang. Dengan PG, dunia permainan dapat dibangun secara otomatis menggunakan algoritma tertentu, memastikan bahwa setiap kali pemain memulai permainan, mereka akan mendapatkan pengalaman yang berbeda.

Pada penelitian ini dalam konteks *game* High on Survival, penerapan PG menggunakan metode *Perlin Noise* digunakan untuk:

1. Menentukan peta (*map*) *biome* dan lingkungan – Dunia dalam *game* memiliki berbagai *biome* misalnya hutan, padang pasir, atau daerah bersalju.
2. Membentuk permukaan medan (*terrain*) – *Terrain* dalam *game* dihasilkan secara *procedural* berdasarkan nilai *map*.
3. Menentukan lokasi distribusi sumber daya dan objek – Penempatan objek seperti makanan, air, dan tempat perlindungan ditentukan berdasarkan informasi dari *map biome* dan aturan objek masing-masing.
4. Menentukan posisi dan pola kemunculan musuh – Lokasi kemunculan musuh disesuaikan dengan nilai *biome* dan karakteristik lingkungan yang terbentuk pada nilai *map* dan aturan musuh masing-masing.

Studi oleh Hendrix, dkk. (2013) menunjukkan bahwa penerapan *Procedural Generation* (PG) dalam *game* aksi *survival* berpotensi menambah variasi dan meningkatkan pengalaman bermain melalui dunia yang terus berubah. Elemen seperti keunikan lingkungan, tantangan yang tidak berulang, dan suasana eksplorasi menjadi aspek yang dapat dipengaruhi oleh pendekatan ini.

B. *Procedural Generation*



Gambar 2. 2 *Terrain Procedural Generation*.
(*Sumber: https://heredragonsabound.blogspot.com/2019/02/perlin-noise-procedural-content.html*)

Procedural Generation (PG) atau Pembangkitan Prosedural adalah pendekatan teknologi dalam menciptakan konten secara otomatis menggunakan algoritma tertentu. Konsep ini sangat berguna dalam pengembangan berbagai aplikasi teknologi, terutama di bidang hiburan digital seperti *game*. Dalam PG, konten dihasilkan secara dinamis berdasarkan parameter yang telah ditentukan, sehingga dapat menciptakan variasi tanpa batas dengan efisiensi waktu dan biaya yang tinggi. Hal ini dilakukan melalui algoritma yang mampu menciptakan elemen seperti bentuk, pola, atau struktur dengan tingkat akurasi dan keunikan yang tinggi.

Keunggulan utama PG adalah kemampuannya dalam menciptakan dunia yang dinamis, imersif, dan menarik tanpa harus mengandalkan proses manual. Dengan teknik ini, pengembang dapat menghasilkan konten seperti peta, karakter, atau mekanisme permainan dengan lebih fleksibel dan sesuai dengan kebutuhan

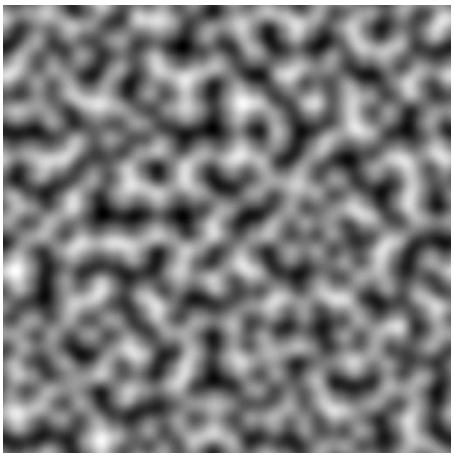
desain tertentu. Teknologi PG juga berperan penting dalam mengurangi repetisi dalam desain konten, yang sering menjadi kelemahan dalam metode konvensional.

Menurut Hendrix, dkk (2013) PG untuk *game* merupakan pemanfaatan komputer untuk menghasilkan konten permainan, membedakan variasi konten yang dihasilkan, serta memilih yang dinilai paling menarik bagi pemain. Hal ini tidak digunakan hanya dalam konsep *game* saja. PG sebelumnya juga digunakan dalam berbagai bidang lainnya seperti Visualisasi dan Grafika Komputer, Komputasi Multimedia, Visualisasi dan Animasi Komputer, dll. Salah satu *game* yang terkenal yaitu Path of Exile, merupakan *game* yang mengimplementasikan metode PG untuk memproduksi universe yang sudah teracak dan playable. Kemudian Borderlands, dimana PG digunakan untuk menghasilkan sejumlah kelompok senjata yang bermacam-macam dengan detail yang berbeda - beda. Pembuatan konten secara acak tersebut akan menambahkan nilai replayability pada *game* tersebut dikarenakan setiap permainan baru dari awal akan berbeda dan memiliki *seed value* berbeda.

C. *Perlin Noise* dalam *Procedural Generation*

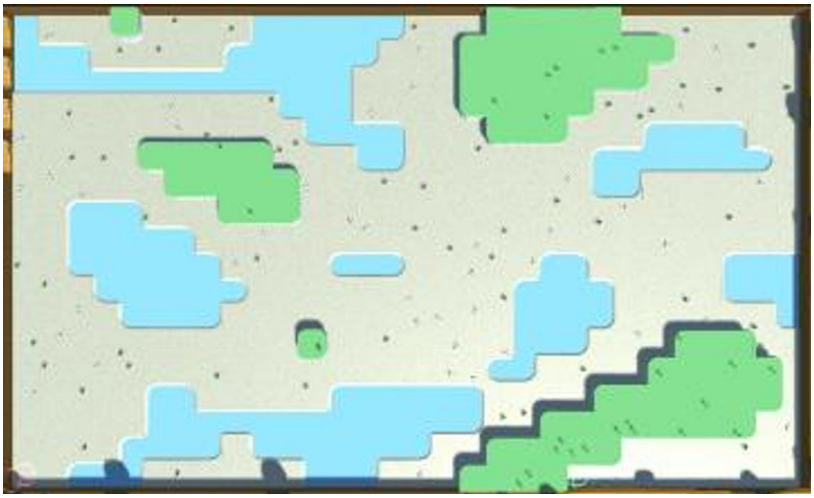
Perlin Noise merupakan salah satu tipe dari *gradient noise* yang menghasilkan nilai yang mirip pada suatu posisi dan area di sekitar posisi yang terkena *Noise* (Juwiantho, H., dkk, 2023). *Perlin Noise* sendiri dikenalkan oleh Ken Perlin pada tahun 1985. Tipe lain dari *gradient noise* selain *Perlin Noise* meliputi *simplex noise* dan *simulation noise*. *Simplex noise* memiliki waktu proses yang lebih cepat dari *Perlin Noise*, tetapi *Perlin Noise* memiliki hasil bentuk *map* dua

dimensi daratan lebih baik dibandingkan dengan *simplex noise* untuk digunakan pada *game* (Juwiantho, H., dkk, 2023).



Gambar 2. 3 *Perlin Noise*.
(*Sumber: https://saifjouda.github.io/SaifJouda/procedural*)

Pembuatan *map* pada penelitian yang dilakukan oleh Juwiantho, dkk (2023) mengimplementasikan *Perlin Noise* untuk membuat *mesh terrain* yang dihasilkan terlihat unik dan natural. Implementasi teknik ini dalam Unity dapat dilakukan dengan memanfaatkan fungsi bawaan seperti *Mathf.PerlinNoise* untuk menghitung nilai *noise* dan membentuk *mesh* berdasarkan nilai tersebut. Hasil dari *map* yang dikenakan *Perlin Noise* memiliki transisi antar ubin yang halus dari satu area ke area lain dan teksturnya tidak terlihat acak di area yang terkena *Perlin Noise* yang dibutuhkan dalam *game* ini untuk rute perjalanan musuh dari posisi awal menuju posisi akhir. Selain menggunakan *noise*, pembuatan *map* juga bisa menggunakan fungsi *random*, tetapi hasil *map* yang diperoleh kurang natural dan kurang baik untuk digunakan dalam *game* tower defense karena menghasilkan transisi yang terlalu kasar dari satu area ke area lain dan teksturnya sangat acak, sehingga tidak layak digunakan dalam *game* yang dibuat.



Gambar 2. 4 Hasil *map* percobaan pertama.
(Sumber: Juwiantho, H., Liliana, L., & Budiono, M., 2023)

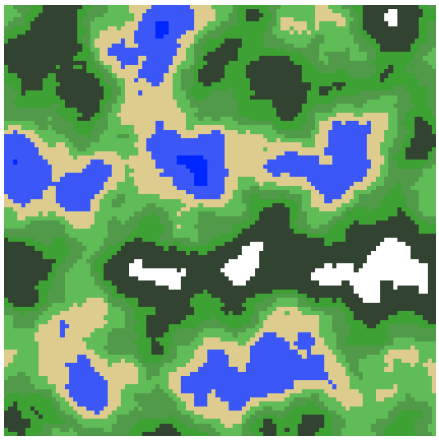
Cara kerja dari *Perlin Noise* dengan mengambil nilai pada suatu posisi sebagai input dan mengembalikan nilai dalam *range* tertentu. *Range* bisa antara -1.0 hingga +1.0 atau akar dari dimensi yang digunakan dibagi dengan 2. Parameter input dari *Perlin Noise* yang digunakan sesuai dengan dimensinya.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

1. *Biome Map*

Penerapan *Perlin Noise* pada pembangkitan *terrain* dengan memanfaatkan nilai *noise* dalam rentang tertentu, pengembang dapat mengelompokkan area ke dalam berbagai kategori lingkungan, seperti gurun, hutan, pegunungan, atau padang rumput.

Dalam pengembangannya, *multi-layer Perlin Noise* dapat digunakan untuk menciptakan transisi yang lebih halus antara berbagai jenis *biome*. Pendekatan ini

memungkinkan lingkungan dalam dunia *game* terlihat lebih alami dan tidak memiliki batasan yang terlalu kaku. Alih-alih hanya menggunakan satu lapisan *noise*, beberapa lapisan dengan frekuensi dan amplitudo berbeda dapat digabungkan untuk menghasilkan variasi yang lebih kompleks dan realistis.



Gambar 2. 5 Hasil generasi *map biome*.
(*Sumber: https://code2d.wordpress.com/2020/07/21/perlin-noise/)*

Peran *Multi-Layer Perlin Noise* dalam Pemetaan *Biome*:

a. Menciptakan Transisi Lingkungan yang Halus

Transisi antarwilayah, seperti antara daratan dan lautan, dibentuk dengan mengatur nilai ambang (*threshold*) dan rentang transisi dalam lapisan benua (*ApplyContinentsLayer*). Pendekatan ini memungkinkan terciptanya perubahan elevasi yang bertahap, menghasilkan zona pesisir atau tepian pulau yang tidak terkesan tiba-tiba, serta mendukung pembentukan wilayah peralihan seperti pantai secara lebih alami.

b. Menyesuaikan *Biome* dengan Elevasi dan Struktur *Terrain*

Pemetaan *biome* dapat dikaitkan dengan ketinggian terrain yang dihasilkan oleh *Perlin Noise*. Daerah dengan elevasi tinggi dapat diklasifikasikan sebagai pegunungan, sementara dataran rendah dapat menjadi hutan atau pantai. Dengan

cara ini, distribusi *biome* menjadi lebih realistis dan sesuai dengan bentuk geografis dunia *game*.

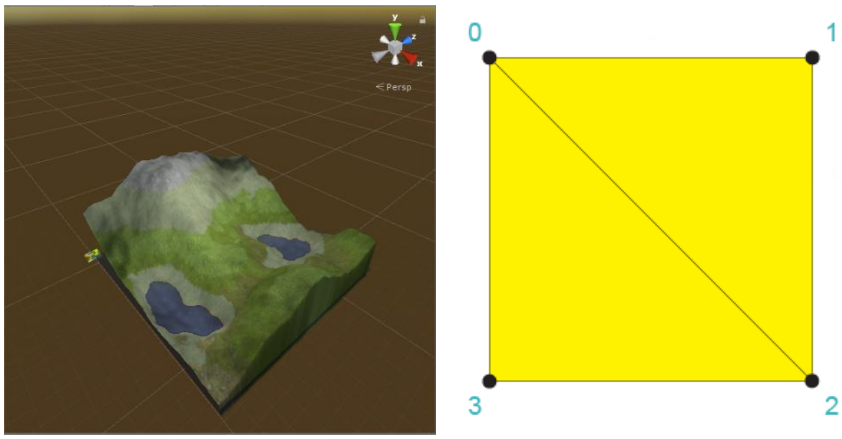
c. Menambahkan Detail dengan Simulasi Erosi

Setelah pembentukan dasar kontur dari lapisan *terrain*, dilakukan pengurangan nilai elevasi melalui lapisan erosi (*ApplyErosionLayer*) menggunakan *noise* tersendiri. Proses ini tidak hanya memperhalus permukaan, tetapi juga menciptakan lekukan dan variasi ketinggian tambahan yang dapat menyerupai lembah atau aliran air, sehingga medan permainan tampak lebih alami dan kompleks secara visual.

2. Generasi *Mesh Terrain*

Generasi *mesh terrain* secara prosedural dapat menghasilkan pola tekstur yang lebih alami dibandingkan dengan penggunaan *noise* acak biasa (Wijaya, dkk, 2018). Teknik ini memungkinkan pengembang *game* untuk menciptakan terrain dengan elevasi yang realistis, di mana nilai *noise* berfungsi sebagai parameter untuk menentukan ketinggian setiap titik pada *grid* terrain.

Dalam mesh *terrain*, permukaan dibangun dari kumpulan verteks (*vertices*) yang disusun dalam bentuk *grid*. Setiap titik pada *grid* memiliki koordinat (x, y, z), di mana: x dan y mewakili posisi horizontal pada bidang, z (tinggi) ditentukan menggunakan nilai *Perlin Noise* pada titik tersebut. Setelah verteks terbentuk, mereka dihubungkan menjadi segitiga (*triangles*) untuk membentuk *mesh*. Biasanya digunakan *quad-based grid*, di mana setiap 4 verteks membentuk 2 segitiga.



Gambar 2. 6 Hasil generasi *terrain*.
(*Sumber: https://github.com/pecarprimoz/procedural-gen-dipl*)

Menurut Wijaya, dkk (2018), "*Perlin Noise* memiliki karakteristik interpolasi yang halus, sehingga dapat menghasilkan *terrain* yang lebih alami dibandingkan dengan metode *noise* acak." Dengan menyesuaikan nilai skala dan frekuensi, terrain dapat divariasikan untuk membentuk berbagai fitur geografis seperti perbukitan, lembah, hingga pegunungan.

3. Distribusi Objek

Dalam *game survival*, objek seperti pohon, Batu, dan air harus didistribusikan secara alami agar lingkungan terasa lebih hidup. Teknik procedural object placement sering menggunakan *Perlin Noise* untuk menentukan kepadatan dan distribusi objek. Menurut Red Blob *Game*s (2013), distribusi objek dapat dikontrol dengan cara berikut:

a. Nilai *noise* rendah = sedikit objek (misalnya, padang rumput terbuka).
b. Nilai *noise* menengah = area dengan vegetasi sedang (misalnya, semak-semak).
c. Nilai *noise* tinggi = area hutan lebat dengan banyak pohon dan beBatuan.

Dalam studinya, Red Blob *Game*s (2013) menyatakan bahwa pemanfaatan *Perlin Noise* dalam distribusi objek memberikan variasi lingkungan yang lebih alami tanpa pola pengulangan yang terlihat jelas.

4. Pemunculan Musuh

Dalam *game survival*, pemunculan musuh (*enemy spawning*) juga dapat dikontrol menggunakan *Perlin Noise*. Dengan cara ini, musuh tidak muncul secara acak, tetapi mengikuti pola yang lebih sistematis berdasarkan lingkungan sekitar.

Sebagai contoh, area dengan nilai *Perlin Noise* yang lebih tinggi dapat diatur sebagai zona dengan tingkat kesulitan lebih besar, di mana musuh yang lebih kuat atau dalam jumlah lebih banyak akan muncul. Sebaliknya, area dengan nilai *noise* lebih rendah dapat dianggap sebagai zona yang lebih aman dengan sedikit atau tanpa musuh, seperti:

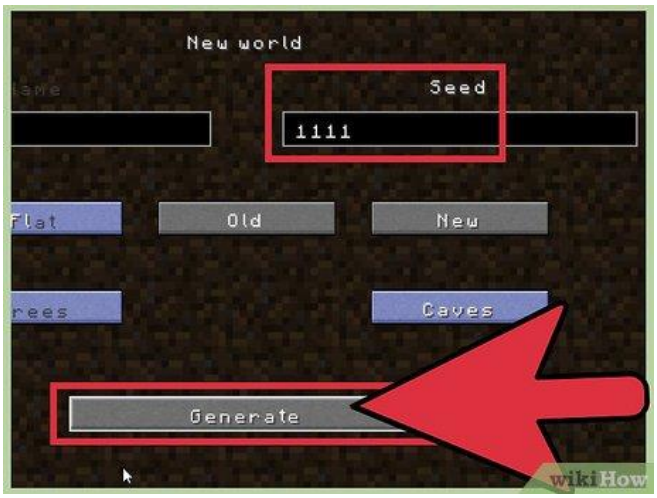
a. Nilai *noise* rendah = Area aman, hanya muncul musuh kecil.
b. Nilai *noise* menengah = Kemunculan musuh standar.
c. Nilai *noise* tinggi = Zona berbahaya dengan musuh yang lebih kuat.

D. *Seed*

Seed adalah nilai dalam bentuk bilangan bulat (*integer*) yang digunakan sebagai parameter dalam menentukan hasil dari suatu proses generasi (Sampaio, dkk, 2017). Dalam penerapannya di dalam permainan video (video *game*), *seed* diproses menggunakan algoritma yang dapat mengubah suatu kata atau frasa menjadi bilangan bulat, sehingga lebih mudah untuk diingat.

Sebagai contoh, sistem *seed* diterapkan dalam permainan Minecraft, khususnya pada edisi Bedrock. Dalam versi ini, *seed* yang berupa kata atau frasa

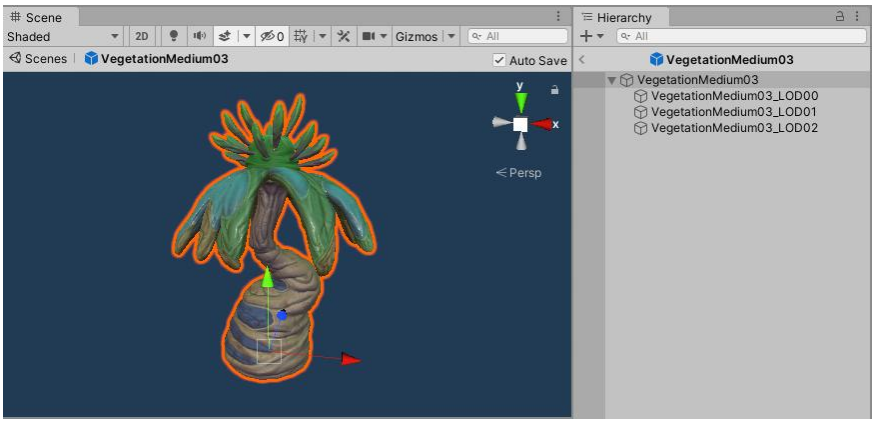
diproses menggunakan algoritma hashing, dengan memanfaatkan fungsi "*String.hashCode*" dalam bahasa pemrograman Java (Mojang, 2011).



Gambar 2. 7 Pengaturan *seed* Minecraft.
(*Sumber: https://www.wikihow.com/Use-Seeds-in-Minecraft)*

E. *Prefabs*

Prefab merupakan sebuah *game object* yang telah dikonversi menjadi *template*. Dalam *game engine* Unity, sistem *prefab* digunakan untuk menyimpan konfigurasi suatu *game object*, termasuk komponen, nilai properti, serta semua *child object* yang ada (Unity Technologies, 2018).



Gambar 2. 8 *Prefab* pohon.
(*Sumber: Unity Technologies, 2018*)

Dalam konteks *procedural generation*, *prefab* memainkan peran penting. Dengan menggunakan *prefab*, pengembang dapat menghasilkan dunia permainan secara otomatis dengan mengatur aturan tertentu dalam algoritma *generation*. Misalnya, dalam permainan yang memiliki peta yang dihasilkan secara prosedural, berbagai variasi bangunan, pohon, atau objek lingkungan lainnya dapat dibuat menggunakan *prefab* sebagai dasar.

Penggunaan *prefab* memungkinkan optimalisasi performa dalam *procedural generation*. Dibandingkan dengan membuat objek secara individual setiap kali permainan dijalankan, *prefab* memungkinkan penggunaan kembali elemen yang telah dibuat sebelumnya, sehingga mengurangi beban pemrosesan. Selain itu, Unity mendukung fitur *nested prefab*, di mana sebuah *prefab* dapat berisi *prefab* lain di dalamnya, memungkinkan struktur hierarki yang lebih kompleks dan fleksibel dalam *procedural generation*.

F. *Chunk Streamline*

Chunk Streamline adalah teknik dalam pengembangan *game* yang bertujuan untuk mengoptimalkan proses *Procedural Generation* (PG) dengan membagi dunia permainan menjadi bagian-bagian kecil yang disebut "*chunks*". Pendekatan ini memungkinkan sistem untuk memuat dan memproses hanya bagian dunia yang relevan dengan posisi pemain, sehingga meningkatkan kinerja dan efisiensi sumber daya.

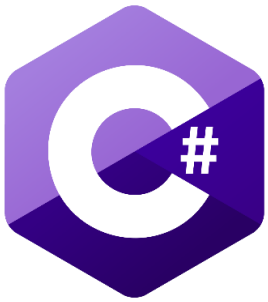
Teknik *chunk-based loading* pertama kali diperkenalkan dalam *game* Minecraft, yang menggunakan sistem ini untuk memuat hanya sebagian dunia di

sekitar pemain, sementara area yang jauh akan dihapus dari memori hingga dibutuhkan kembali (Mojang, 2011).

Menurut dokumentasi pengembangan Minecraft, pendekatan ini memungkinkan permainan untuk mengatasi tantangan skala dunia besar dalam PG tanpa mengorbankan pengalaman bermain (Mojang, 2011). Teknik ini dikenal dengan istilah *load-on-demand*, di mana dunia *game* hanya dimuat sesuai dengan kebutuhan saat pemain bergerak ke area baru.

Dalam dunia *game* berbasis *Procedural Generation*, seperti yang digunakan dalam Minecraft dan *game* lainnya seperti No Man's Sky (Hello *Game*s, 2016), *Chunk Streamline* sangat efektif dalam menjaga kinerja permainan. Dunia yang sangat luas, yang dihasilkan secara acak atau prosedural, membutuhkan cara untuk mengoptimalkan pengelolaan data. Dengan *chunking*, pengembang dapat memastikan bahwa hanya bagian dunia yang perlu diakses yang dimuat ke dalam memori pada waktu tertentu.

G. Bahasa Pemrograman C#



Gambar 2. 9 Logo bahasa C#.
(*Sumber: https://commons.wikimedia.org/wiki/File:C-Sharp_Logo.svg*)

C# adalah bahasa pemrograman modern yang dikembangkan oleh Microsoft dan dirancang untuk berjalan di platform .NET. Bahasa ini berbasis pemrograman

berorientasi objek (OOP), yang berarti mendukung konsep seperti *class, inheritance, polymorphism, dan encapsulation*. Menurut Saptarini dkk. (2020), bahasa pemrograman telah menjadi bidang yang semakin diminati. Berbagai jenis bahasa pemrograman dikategorikan berdasarkan tingkat abstraksi dan cara kerjanya. Dalam pengembangannya, C# dibuat agar sederhana, fleksibel, dan dapat digunakan untuk berbagai jenis aplikasi, termasuk aplikasi *desktop*, *web*, *mobile*, *game*, serta layanan *cloud*. Namun, menurut Syakir, A.A. (2015), ada kritik yang menyebutkan bahwa C# banyak mengambil inspirasi dari bahasa pemrograman lain.

Salah satu keunggulan utama C# adalah sistem tipe data yang kuat serta manajemen memori otomatis melalui *garbage collector*, yang membantu mengurangi potensi kesalahan umum pada bahasa pemrograman lainnya. Dengan adanya .NET Core, aplikasi berbasis C# kini dapat dijalankan di berbagai sistem operasi, seperti Windows, macOS, dan Linux, menjadikannya lebih fleksibel dan portabel.

Sintaks C# yang bersih dan mudah dipahami membuatnya ramah bagi pemula. Selain itu, bahasa ini sangat populer dalam pengembangan *game* karena digunakan oleh Unity, salah satu mesin *game* terkemuka di dunia. C# juga banyak digunakan dalam pengembangan aplikasi web melalui framework ASP.NET serta aplikasi *mobile* menggunakan Xamarin. Dengan dukungan penuh dari ekosistem Microsoft yang luas, C# terus berkembang menjadi pilihan utama bagi pengembang yang ingin membangun aplikasi modern dan terintegrasi.

H. Unity *Game* Engine

Game engine merupakan sebuah *framework* yang dirancang untuk mendukung proses pengembangan video *game* dengan menyediakan berbagai fungsi serta integrasi penting dengan perangkat lunak lainnya. Contohnya, *game engine* seperti Unity dapat terhubung dengan perangkat lunak pembuat aset 3D, seperti Maya atau Blender, sehingga mempermudah proses pembuatan *game* (Unity Technologies, 2018).

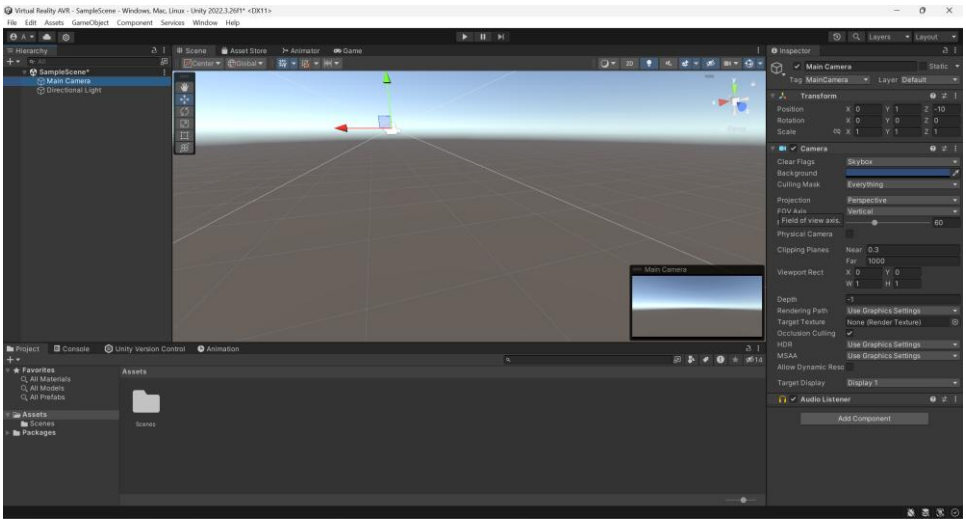
Salah satu pendiri Unity, David Helgason mendeskripsikan Unity sebagai *game engine* dan *Integrated Development Env*Besi*ment* (IDE) untuk membuat media interaktif seperti video *game*. Unity dapat digunakan untuk membuat *game* tiga dimensi (3D) dan dua dimensi (2D), serta simulasi interaktif dan media lainnya. Unity telah diadopsi oleh industri selain *game*, seperti film, otomotif, dan arsitektur. Seiring dengan kemajuan teknologi, penggunaan *game engine* telah menjadi standar dalam industri, baik oleh studio pengembang AAA maupun studio *indie*. Berikut ini adalah beberapa fungsi utama yang umumnya dimiliki oleh sebuah *game engine*:

1. Dukungan terhadap bahasa pemrograman, di mana *game engine* setidaknya terintegrasi dengan satu bahasa pemrograman tertentu. Contohnya, Unity menggunakan C#, sedangkan Unreal Engine menggunakan C++.
2. *Rendering engine*, yang berperan dalam mengelola tampilan aset 2D dan 3D, atau kombinasi keduanya. Biasanya, fitur ini juga mencakup animasi untuk meningkatkan visual dalam permainan.

3. *Physics engine*, yang bertugas mengimplementasikan prinsip fisika dalam lingkungan video *game*, sehingga memberikan efek yang lebih realistis.
4. *Audio engine*, yang digunakan untuk menangani pemrosesan dan pemutaran suara dalam permainan.

Unity juga memiliki banyak keunggulan jika di jadiakan sebagai *game engine* karena menurut Setiawan, dkk (2020) unity adalah aplikasi multiplatform yang dapat membuat aplikasi dengan format *file* exe, apk dan lain-lain yang dapat digunakan diberbagai sistem operasi. yang membuat unity menjadi *game engine* fleksibel dalam pembuatan *game*. Pada penelitian ini penulis akan menggunakan Unity 6 6000.0.42f1 dengan antarmuka Unity Hub 3.11. Ada beberapa *tools-tools* yang digunakan dan perlu diketahui yaitu:

1. Unity Editor



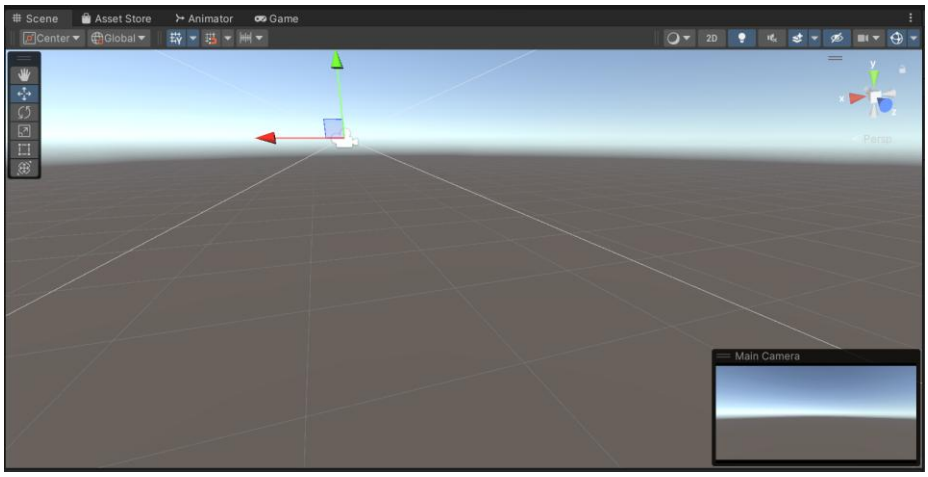
Gambar 2. 10 Unity editor.

Proyek baru secara *default* akan memiliki beberapa objek/*GameObject* seperti *Main Camera* dan Pencahayaan (*Directional Light*) pada panel *hierarchy*. Sebelum kita mulai menambahkan atau mengubah objek di dalam proyek, sangat

penting untuk memahami beberapa panel yang tersedia di Unity Editor terlebih dahulu. Panel-panel ini akan membantu kita dalam mengatur, melihat, dan memanipulasi berbagai elemen dalam proyek kita dengan lebih efektif, panel-panelnya seperti berikut.

2. Panel *Scene*

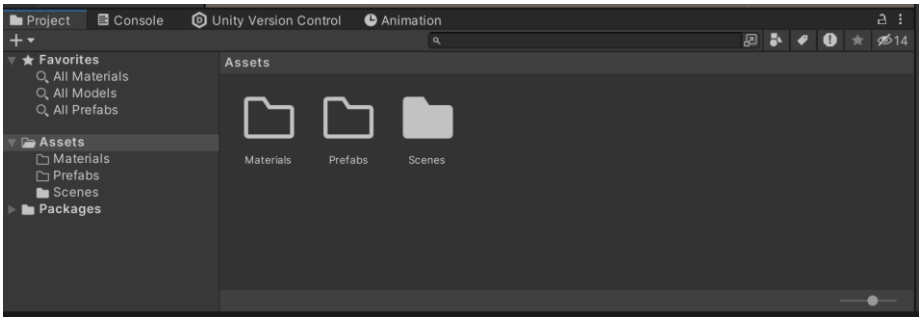
Panel *scene* digunakan untuk melihat secara *visual game* yang kita bangun. Dengan *transform tools* yang berada pada *toolbar* Unity, kita bisa melakukan modifikasi kepada *Game Object* yang kita masukan ke dalam *scene*.



Gambar 2. 11 Panel *scene.*

3. Panel *Project*

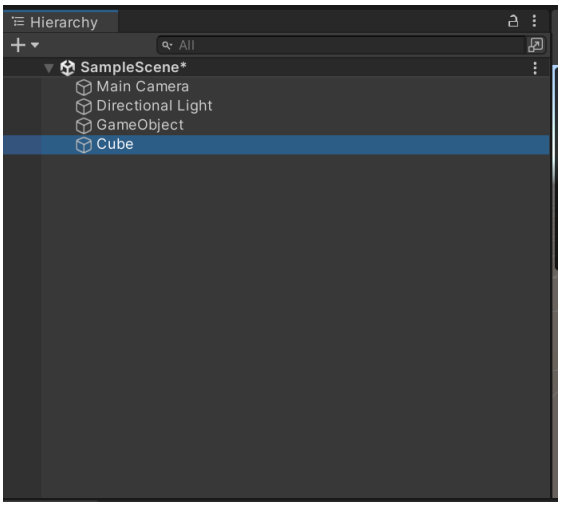
Panel *project* digunakan untuk mengorganisir aset yang kita gunakan. Struktur *file* yang kita buat di dalam *project window* akan disimpan dalam struktur yang sama juga di dalam harddisk kita. Di dalam project window, kita bisa memasukan aset yang akan kita gunakan.



Gambar 2. 12 Panel *project*

4. Panel *Hierarchy*

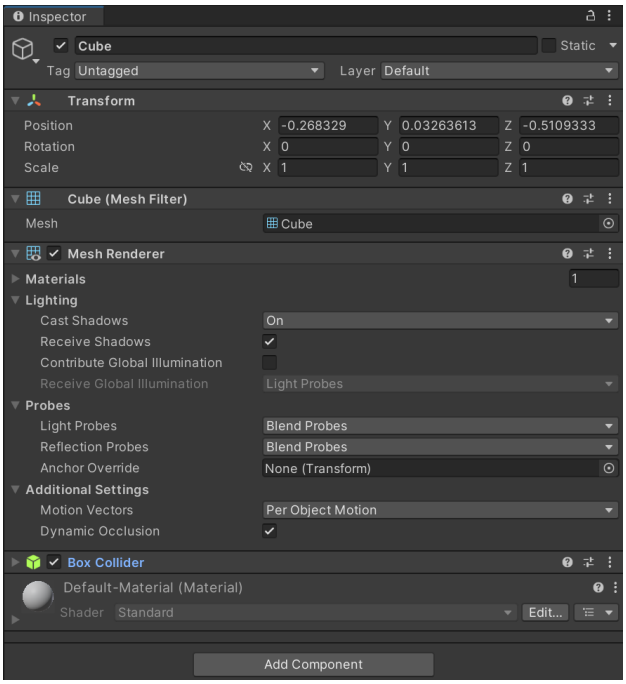
Panel *hierarchy* berisikan seluruh *game object* yang ada di dalam di dalam *game scene*. Jika di dalam *project* window adalah aset yang ada di dalam harddisk kita, *hiearchy* berisikan aset yang kita gunakan di dalam *scene*. Kita bisa memasukan aset dari panel *project* ke *hierarchy* dengan cara menyeret *file* atau kita juga bisa *create aset* menggunakan *drop down create* yang ada di *toolbar hierarchy*.



Gambar 2. 13 Panel *hierarchy*.

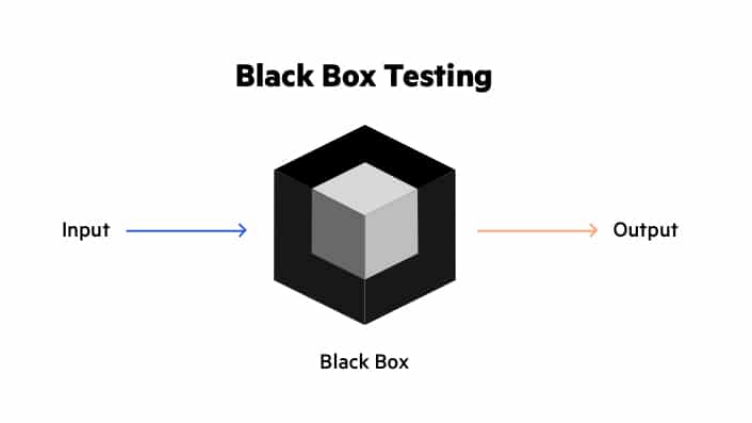
5. Panel *Inspector*

Panel *inspector* adalah tempat untuk mengubah nilai dari properti dan setting. panel *inspector* bisa berbeda tampilan dan fungsinya untuk tiap komponen yang berbeda. Inspector bisa menampilkan informasi dari player *setting, asset, game object, scene, prefab*, dan lain-lain. Jika kita sedang melihat sebuah *game object*, maka yang ditampilkan oleh *inspector* adalah komponen yang ada di dalamnya. Pada umumnya bagian atas akan berisikan nama *game object* yang sedang dilihat beserta beberapa informasi seperti tag dan layer, lalu di bagian bawahnya adalah komponen yang terdapat dalam *game object* tersebut. Satu hal yang pasti, setiap *game object* pasti memiliki header dan komponen *transform*.



Gambar 2. 14 Panel *inspector.*

F. *Blackbox Testing*



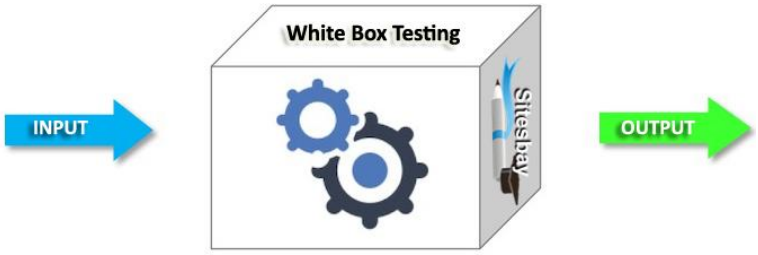
Gambar 2. 15 *Blackbox testing*.
(*Sumber: Fedianto, dkk, 2024*)

Metode pengujian *Blackbox* adalah teknik pengujian yang berfokus pada spesifikasi fungsional perangkat lunak. Penguji dapat menentukan kumpulan kondisi input dan melakukan tes pada spesifikasi fungsional program. Uji *Blackbox* ini lebih fokus pada fungsionalitas sistem. Implementasi memiliki kelebihan dan kekurangan pengujian *Blackbox*. Salah satu keuntungannya adalah membantu Anda menemukan aspek-aspek yang tidak terpenuhi dari spesifikasi kebutuhan pengembangan perangkat lunak. Untuk kekurangan *Blackbox*, tes ini tidak sepenuhnya dilakukan dengan sejumlah terbatas penguji sehubungan dengan perangkat lunak yang diuji. (Fedianto, dkk, 2024).

Pengujian *Blackbox* adalah metode pengujian perangkat lunak atau aplikasi yang berfokus pada fungsionalitas perspektif pengguna tanpa memeriksa struktur internal atau desain sistem. Sederhananya, pengujian *Blackbox* menilai apakah sistem dapat membuat edisi yang sesuai berdasarkan input yang ditentukan. Jika output cocok dengan ekspektasi, sistem dipertimbangkan dengan tepat. Namun, jika

langkah -langkah dilakukan, mereka perlu ditingkatkan. Metode ini juga dikenal sebagai tes perilaku, tes kotak tertutup, atau tes yang ditentukan.

G. *Whitebox* Testing



Gambar 2. 16 *Whitebox* testing.
(*Sumber: Wintana & Nurhadi, 2022)*

Whitebox testing adalah teknik pengujian perangkat lunak yang menitikberatkan pada komponen internal, seperti desain, struktur kode, dan cara kerja perangkat lunak dari awal hingga akhir. Pengujian ini bertujuan untuk menganalisis alur data dari input ke output, menilai fungsionalitas, serta memastikan keamanan perangkat lunak. Dalam *white box* testing, penguji memiliki akses penuh untuk melihat dan menganalisis kode yang digunakan dalam pengembangan perangkat lunak. Oleh karena itu, metode ini juga dikenal dengan istilah *clear box* testing*, open box* testing*, code-based testing*, atau *structural* testing.

Whitebox testing dilakukan karena fokusnya pada logika internal dan struktur kode perangkat lunak. Uji kasus dirancang untuk menguji fungsi internal perangkat lunak dari sudut pandang pengembang. Teknik ini dilakukan ketika pengembang memiliki pemahaman menyeluruh tentang struktur program. Dengan menggunakan pendekatan ini, setiap cabang dan keputusan dalam program dapat diuj (Wintana & Nurhadi, 2022).

Whitebox testing memiliki beberapa kelebihan, di antaranya adalah kemampuannya untuk mengidentifikasi dan menghapus kode tersembunyi atau yang tidak diperlukan. Metode ini memungkinkan pengujian secara menyeluruh karena setiap bagian dari struktur atau logika program dieksplorasi secara detail. Selain itu, *Whitebox* testing membantu mengoptimalkan kode dan dapat dilakukan bahkan saat antarmuka pengguna grafis (GUI) masih dalam tahap pengembangan (Fedianto, dkk, 2024).

H. Penelitian Terdahulu

Untuk mendukung penelitian ini, diperlukan berbagai studi sebelumnya yang relevan. Penelitian-penelitian terdahulu yang berkaitan dengan topik yang dibahas akan memberikan informasi penting yang dibutuhkan oleh peneliti. Dalam konteks ini, referensi yang berhubungan dengan tantangan dalam teknologi informasi dari penelitian sebelumnya digunakan sebagai acuan. Oleh karena itu, penelitian ini dilakukan secara daring dengan memanfaatkan berbagai temuan yang dipublikasikan dalam makalah, tesis, atau artikel ilmiah.

Selviana dkk. (2023) dalam penelitiannya menyimpulkan bahwa implementasi *Procedural Content Generation* (PCG) menggunakan bahasa pemrograman C# dan metode *Perlin Noise* di Unity dapat meningkatkan kualitas *gameplay* serta memberikan pengalaman bermain yang lebih mendalam bagi pengguna. Teknik generasi *map* dan pemunculan objek secara acak menciptakan variasi serta kejutan dalam *gameplay*, sehingga membuat pemain lebih tertarik. Selain itu, PCG juga mempercepat proses pengembangan *game* karena konten dihasilkan secara otomatis. Hasil penelitian ini berkontribusi pada pengembangan

game di Unity, khususnya dalam teknik generasi map dan pemunculan objek secara acak pada *game* 3D. Oleh karena itu, penggunaan PCG diharapkan dapat terus dikembangkan untuk meningkatkan kualitas *game* dan memberikan pengalaman bermain yang lebih baik. Penelitian selanjutnya dapat mengeksplorasi metode lain dalam PCG atau mengoptimalkan penggunaan *Perlin Noise* untuk hasil yang lebih variatif.

Arifudin dkk. (2024) membuktikan bahwa implementasi Algoritma *Random Walk* dalam *Procedural Generation* dapat menciptakan level yang otomatis dan dinamis dalam *game* Dungeon Crawler 2D. Berdasarkan pengujian *Blackbox*, algoritma ini mampu menghasilkan level yang selalu diacak setiap kali permainan dimulai atau pemain memasuki portal ke level berikutnya, sehingga meningkatkan *replayability* dan memberikan pengalaman bermain yang berbeda di setiap sesi.

Namun, beberapa aspek masih dapat ditingkatkan dalam penelitian selanjutnya, seperti optimasi algoritma untuk meningkatkan efisiensi dan mengurangi waktu pemrosesan, serta penyesuaian tingkat kesulitan level agar sesuai dengan progresi pemain. Selain itu, menambahkan elemen naratif, diversifikasi lingkungan dan musuh, serta melakukan studi lebih lanjut tentang pengalaman pengguna dapat semakin memperkaya *gameplay*. Dengan pengembangan lebih lanjut, penelitian ini dapat memberikan kontribusi lebih besar dalam bidang *Procedural Generation*, khususnya dalam genre *Dungeon Crawler* 2D.

Juwiantho dkk. (2023) juga membahas penerapan *Procedural Content Generation* (PCG) dalam *game* Tower Defense menggunakan *Perlin Noise* untuk

pembuatan *map* dan *Floyd-Warshall* untuk memastikan jarak portal pemain dan musuh memenuhi syarat. Hasil pengujian menunjukkan bahwa dari 100 percobaan, 25% map perlu dibuat ulang, tetapi rata-rata waktu pembuatan *map* tetap cepat, yaitu 0,99 detik. Selain itu, variasi musuh yang berbeda di setiap map meningkatkan *replayability game*. Metode ini terbukti efisien dan efektif dalam menciptakan *map* yang unik dan menantang, meskipun ada kelemahan dalam beberapa *map* yang harus diulang. Secara keseluruhan, pendekatan ini dapat diterapkan pada *game* lain yang memerlukan konten prosedural. Hasil penelitian ini menunjukkan efisiensi PG mampu menciptakan *map* yang bervariasi, menantang, dan dapat dibuat dalam waktu yang sangat singkat. Keunggulan metode ini terletak pada efisiensi waktu, keberagaman *map*, dan dinamika rute musuh, yang memaksa pemain untuk selalu menggunakan strategi berbeda dalam setiap permainan. Namun, ada kelemahan di mana beberapa *map* perlu dibuat ulang karena tidak memenuhi kriteria jarak antar portal, meskipun waktu tambahan yang dibutuhkan masih dalam batas wajar. Secara keseluruhan, metode ini terbukti efektif dalam meningkatkan variasi *map* pada *game* Tower Defense dan dapat diterapkan dalam pengembangan *game* lain yang memerlukan konten yang dihasilkan secara prosedural.

Penelitian oleh Ramadhan dkk. (2022) membahas penggunaan *Perlin Noise* dalam PG pada *game* eksplorasi *sandbox* untuk menciptakan bentuk dunia yang lebih natural dan dinamis. Penelitian ini mengungkapkan bahwa *Perlin Noise* dapat menghasilkan bentuk bukit dan gua yang terlihat natural, serta memberikan wawasan lebih lanjut tentang cara algoritma *noise* dapat diterapkan dalam penciptaan terrain untuk eksplorasi dunia yang lebih imersif.

I. Kerangka Berpikir

Untuk memahami kerangka piker studi kasus ini dapat dilihat pada Gambar 2.17.

Rumusan Masalah

Bagaimana cara mengimplementasikan *Procedural Generation* (PG) dalam pengembangan *game* aksi *survival High on Survival*.

⬇

Solusi

Menggunakan algoritma *Perlin Noise* sebagai metode *Procedural Generation* untuk membentuk empat elemen utama dunia *game*: peta (*biome map*), permukaan medan (*terrain*), penempatan objek (*object spawner*), dan pemunculan musuh (*enemy spawner*) secara otomatis.

⬇

Tahapan

1. Analisis kebutuhan dan desain struktur PG.
2. Implementasi *Perlin Noise* untuk pembangkitan peta dan terrain.
3. Penempatan objek dan musuh secara prosedural berdasarkan nilai peta.
4. Integrasi sistem chunk *streamline*.
5. Pengujian sistem dengan metode *Blackbox* dan White *Box*.

⬇

Hasil yang Diharapkan

Fungsi sistem yang diterapkan berjalan sesuai dengan yang dirancang, yakni pemetaan dunia, pembentukan *terrain*, penempatan objek, dan kemunculan musuh yang bervariasi dan logis, serta dapat diuji dan dijalankan dalam *game*.

Gambar 2. 17 Kerangka berpikir.

BAB III

METODE PENELITIAN

A. Jenis Penilitian

Penelitian ini merupakan penelitian eksperimental yang berfokus pada pengembangan *procedural generation* berbasis *Perlin Noise* dalam *game* aksi *survival* High on Survival. Tujuan penelitian ini adalah mengimplementasikan dan menguji algoritma Perlin Noise dalam pembuatan *map*, *terrain*, penempatan objek, dan pemunculan musuh, serta melihat dampaknya terhadap variasi *map* permainan berdasarkan *seed* yang berbeda.

B. Lokasi dan Waktu Penelitian

Lokasi penelitian dilakukan di Jl. Cempaka No.74, Kab. Pinrang dan waktu yang digunakan dalam pelaksanaan penelitian ini adalah 4 (empat) bulan dimulai pada bulan Maret sampai dengan bulan Juni 2025.

Tabel 3. 1 Jadwal Pelaksanaan Penelitian.

No	Uraian Kegiatan	Bulan			
		Maret	April	Mei	Juni
1	Studi Literatur				
2	Analisis kebutuhan				
3	Perancangan sistem PG				
4	Implementasi PG				
5	Pengujian				

C. Alat dan Bahan

Dalam pelaksanaan peneltian ini digunakan beberapa alat dan bahan yang terdiri dari perangkat keras (*hardware*) dan perangkat lunak (*software*).

1. Alat

Alat yang akan digunakan dalam bentuk perangkat keras (*hardware*) dan perangkat lunak (*software*). Spesifikasi *hardware* dan *software* yang digunakan dapat dilihat pada Tabel 3. 2 dan Tabel 3. 3.

Tabel 3. 2 Spesifikasi *hardware*.

Spesifikasi *Hardware*		
No	**Nama**	**Keterangan**
1	Merk Laptop	Lenovo Loq 15APH8
2	*Processor*	AMD Ryzen 5 7640HS
3	Kartu Grafis	NVIDIA GeForce RTX 4050 6GB
4	RAM	16 GB
5	Penyimpanan	512 GB

Tabel 3. 3 Spesifikasi *software*.

Spesifikasi *Software*		
No	**Nama**	**Keterangan**
1	Sistem Operasi	Windows 11 23H2
2	Editor	Visual Studio 2022
3	Bahasa Pemrograman	C# (C Sharp)
4	*Game Engine*	Unity 6 6000.0.42f1

2. Bahan

Bahan yang digunakan dalam pengimplementasian aplikasi tertera pada Tabel 3. 4.

Tabel 3. 4 Bahan-bahan penelitian.

Bahan		
No	**Nama**	**Keterangan**
1	Algoritma	*Procedural Generation*, *Perlin Noise*
2	*Prefabs*	Template objek seperti pohon, Batu, musuh, dan *items*.

D. Teknik Pengumpulan Data

1. Studi Literatur

Mengumpulkan referensi terkait *procedural generation*, metode *Perlin Noise*, *biome generation*, dan teknik pengembangan *game*.

2. Perancangan Sistem

a. Merancang sistem pembangkitan peta (*map*) secara prosedural menggunakan algoritma *Perlin Noise* untuk menghasilkan distribusi *biome* secara alami.

b. Membentuk permukaan medan (*terrain*) berdasarkan nilai elevasi dari *Perlin Noise*, disesuaikan dengan peta *biome*.

c. Menyusun aturan penempatan objek (*object spawner*) seperti pohon, Batu, dan sumber daya berdasarkan klasifikasi *biome* dan nilai *noise*.

d. Merancang logika pemunculan musuh (*enemy spawner*) yang dipengaruhi oleh jenis *biome* dan nilai ambang tertentu dari *Perlin Noise*.

e. Mengembangkan sistem *chunk streamline* untuk membagi dunia permainan menjadi bagian-bagian kecil (*chunk*) agar proses pemuatan dunia menjadi lebih ringan dan tidak membebani performa permainan.

3. Implementasi *Procedural Generation* pada *Game* High On Survival

Implementasi dilakukan dengan menggunakan algoritma *Perlin Noise* di Unity untuk menghasilkan beberapa elemen dunia dalam *game* High on Survival. Elemen-elemen tersebut meliputi:

a. Peta (*Map*) *Biome*, Peta *biome* dibentuk secara prosedural berdasarkan nilai *Perlin Noise*, yang dikategorikan ke dalam jenis *biome* tertentu.
b. Permukaan Medan (*Terrain*), Nilai *noise* digunakan untuk menentukan elevasi pada setiap titik dalam *terrain*, sehingga membentuk permukaan.
c. Penempatan Objek (*Object Spawner*), Objek seperti pohon dan Batu d*item*patkan secara otomatis sesuai dengan nilai *noise* dan jenis *biome* pada titik tersebut.
d. Pemunculan Musuh (*Enemy Spawner*), Musuh muncul berdasarkan zona yang ditentukan oleh nilai *Perlin Noise* dan kondisi *biome*, untuk menciptakan variasi lokasi kemunculan.
e. *Chunk Streamline*, Dunia permainan dibagi menjadi bagian-bagian kecil (*chunk*) yang dimuat berdasarkan posisi pemain, untuk membantu mengelola area permainan yang luas.

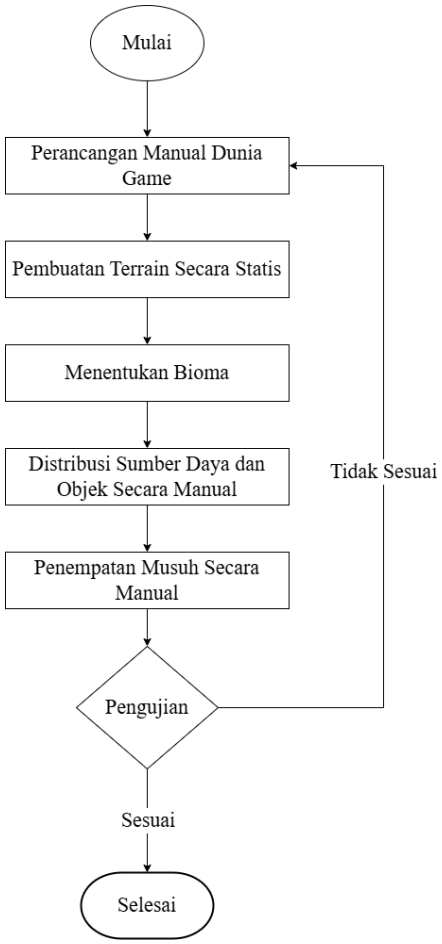
4. Pengujian

Pengujian ini bertujuan untuk memastikan bahwa setiap fitur dalam sistem berfungsi sesuai dengan yang dirancang. Pengujian mencakup seluruh aspek fungsional, mulai dari awal permainan, proses *procedural generation*, jalannya permainan (*gameplay*), hingga kondisi permainan berakhir (*game over*). Selain itu,

pengujian juga dilakukan untuk melihat dampak variasi peta yang dihasilkan menggunakan *seed* yang berbeda.

E. Rancangan Sistem

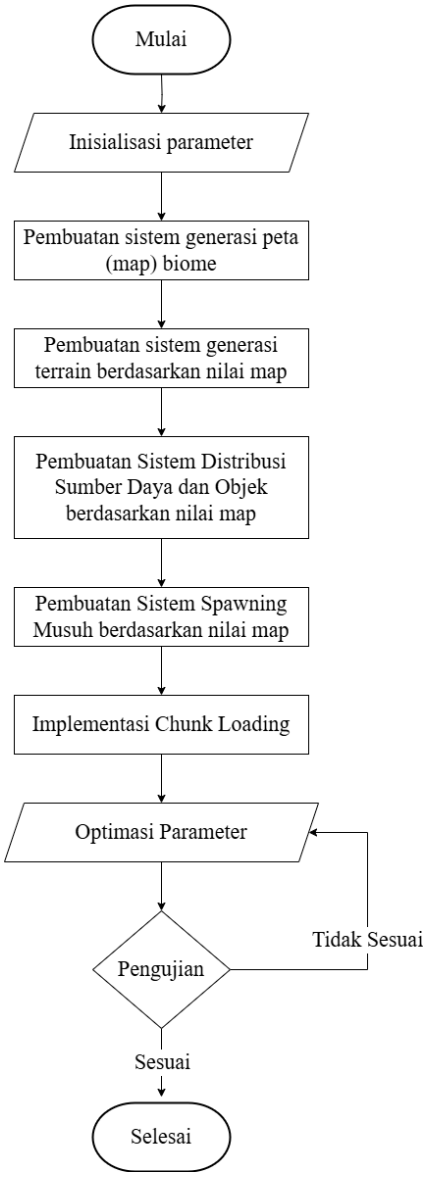
1. Sistem yang Berjalan



Gambar 3. 1 Sistem yang berjalan.

Pada sistem yang berjalan saat ini, membangun dunia permainan secara manual, dengan permukaan medan, peta *biome*, sumber daya, dan musuh d*item*patkan statis oleh pengembang. Metode ini membuat pengalaman bermain kurang dinamis dan repetitif, serta memerlukan banyak waktu dan sumber daya dalam pembuatannya.

2. Sistem yang Diusulkan



Gambar 3. 2 Sistem yang diusulkan.

Sistem yang diusulkan menggunakan *Procedural Generation* berbasis *Perlin Noise* untuk menciptakan dunia permainan yang dinamis dan bervariasi. Algoritma ini membentuk medan alami, menentukan *biome* secara adaptif, serta mengatur distribusi sumber daya, objek, dan musuh, sehingga setiap sesi permainan terasa unik dan tidak terprediksi.

BAB IV

HASIL DAN PERANCANGAN SISTEM

Pada bab ini, penulis menjelaskan hasil dokumentasi pengembangan *game*, rancangan sistem yang digunakan, proses implementasinya ke dalam *game* High on Survival, serta pengujian terhadap sistem yang telah dibuat.

A. Dokumentasi

Dokumentasi ini bertujuan menjelaskan fitur-fitur dalam *game* High on Survival, baik dari segi tampilan, fungsi, hingga interaksi antar komponen.

1. Gambaran Umum *Game*

a. Menu Utama



Gambar 4. 1 Gambaran menu utama

Menu utama merupakan antarmuka awal yang ditampilkan ketika *game* dijalankan. Fungsinya adalah memberikan akses awal kepada pemain untuk

memulai atau melanjutkan permainan, menyesuaikan pengaturan, atau keluar dari aplikasi.

b. Tampilan *Heads-Up Display* (HUD)



Gambar 4. 2 Gambaran tampilan *Heads-Up Display*

Heads-Up Display (HUD) berfungsi sebagai tampilwan informasi penting yang terus muncul di layar selama permainan berlangsung. Tujuannya adalah memberikan pemain informasi status karakter, tujuan misi, dan Skor permainan secara *real-time* tanpa mengganggu jalannya permainan. HUD ini mencakup:

1) Status karakter (HP, lapar, haus, tidur).
2) Skor permainan (penambahan berdasarkan aksi).
3) Informasi misi aktif saat ini.

c. Tujuan *game*

Tujuan *game* High on Survival adalah untuk bertahan hidup dan menyelesaikan seluruh misi yang telah ditetapkan. Pemain harus memenuhi berbagai jenis misi seperti mengumpulkan *item*, membunuh musuh, membangun

struktur, dan membuat *item* melalui *crafting*, sambil mengelola kebutuhan hidup seperti kesehatan, stamina, dan energi.

Pemain akan dinyatakan menang jika berhasil menyelesaikan seluruh misi hingga misi terakhir dalam batas waktu yang ditentukan. Setelah misi terakhir selesai, akan muncul tampilan kemenangan (Victory Screen)

2. Menu Utama



Gambar 4. 3 Menu utama

Menyediakan navigasi awal bagi pemain untuk memulai permainan, memuat permainan yang telah disimpan, atau keluar dari aplikasi. Dalam tampilan utama ini, terdapat beberapa tombol yang memiliki fungsi sebagai berikut:

a. Main

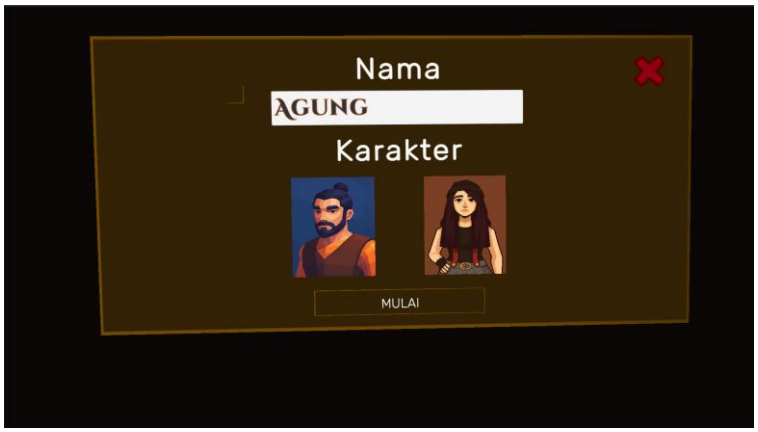
Tombol Main berfungsi sebagai pemicu tampilan submenu awal permainan. Ketika tombol ini ditekan, akan dilakukan pengecekan terhadap status panel submenu, dan jika belum aktif, maka akan ditampilkan dua tombol tambahan:



Gambar 4. 4 Tombol-tombol Main

1) *Mulai Baru*

Saat tombol "*Mulai Baru*" ditekan, pemain akan diarahkan ke tampilan untuk memasukkan nama dan memilih karakter..



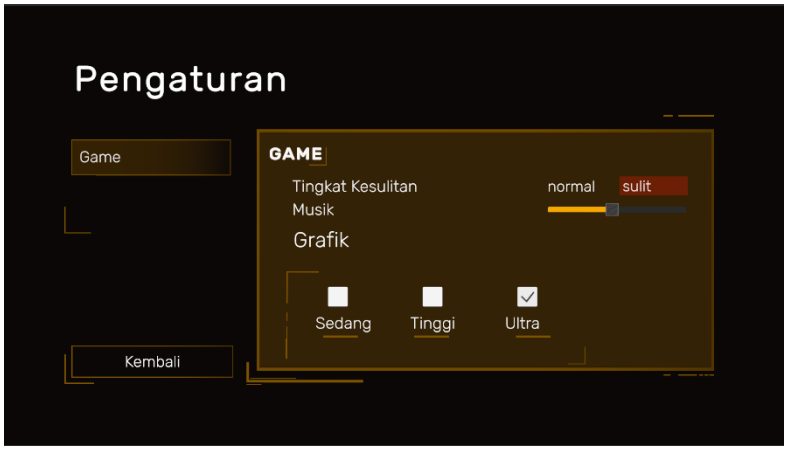
Gambar 4. 5 UI input nama dan karakter

Setelah nama diisi dan karakter dipilih, pemain dapat menekan tombol Mulai untuk memuat dan memasuki *scene game*.

2) *Lanjutkan*

Untuk memuat permainan terakhir dari data yang tersimpan menggunakan *save system* dan memuat *scene game*.

b. Pengaturan



Gambar 4. 6 Tampilan Pengaturan

Ketika tombol ini ditekan, tampilan akan berpindah dari menu utama ke panel pengaturan. Panel pengaturan ini biasanya mencakup beberapa komponen yakni:

1) **Tingkat kesulitan**



Gambar 4. 7 Pengaturan tingkat kesulitan

Menentukan frekuensi kemunculan musuh: "*Normal*" = 1x, "*Hardcore*" = 1.5x. Disimpan di *PlayerPrefs* dengan key "*Difficulty*" dan digunakan saat spawn musuh.

```
private float GetSpawnMultiplierByDay()
{
    int difficulty = PlayerPrefs.GetInt("Difficulty");
    float difficultyMultiplier = (difficulty == 1) ? 1.5f : 1f;
    return difficultyMultiplier;
}
```

2) **Musik**

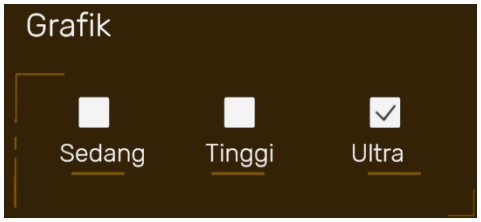


Gambar 4. 8 Pengaturan musik

Slider untuk mengatur volume musik. Nilainya disimpan di *PlayerPrefs* dengan *key* "*MusicVolume*" dan diterapkan pada sistem audio.

```
public void MusicSlider (float value){
    AudioManager.instance.SetBGMVolume(value);
    PlayerPrefs.SetFloat("MusicVolume",
musicSlider.GetComponent<Slider>().value);
}
```

3) Grafik



Gambar 4. 9 Pengaturan grafik

Toggle untuk memilih kualitas: "*Medium*", "*High*", "*Ultra*". Mempengaruhi efek visual seperti tekstur, bayangan, pencahayaan. Disimpan di *PlayerPrefs* dengan *key* "*GraphicsQuality*".

```
public void SetGraphicMedium()
 {
     PlayerPrefs.SetInt("GraphicsQuality", 0);
     QualitySettings.SetQualityLevel(0, true);
     UpdateGraphicQualityUI(0);
 }
```

c. Keluar



Gambar 4. 10 Tombol keluar

Ketika di tekan akan muncul panel yang menampilkan konfirmasi keluar *game* dengan dua tombol yaitu Ya dan Tidak, tekan Ya untuk keluar *game*.



Gambar 4. 11 Konfirmasi keluar *game*

3. Pemain

a. Karakter

Terdapat dua karakter yang dapat dimainkan, yaitu karakter laki-laki dan perempuan. Keduanya memiliki status dan kemampuan dasar yang sama, sehingga tidak memengaruhi jalannya permainan

1) Karakter Laki-laki



Gambar 4. 12 Karakter laki-laki

2) Karakter Perempuan



Gambar 4. 13 Karakter perempuan

b. Status



Gambar 4. 14 Status pemain

Pemain memiliki empat status utama yang ditampilkan di HUD sebagai indikator kebutuhan hidup, yaitu:

1) Nyawa (HP)



Gambar 4. 15 Balok nyawa (HP)

Menunjukkan jumlah nyawa (HP) pemain, yang ditampilkan dengan balok berwarna merah. Jika balok ini habis, pemain dianggap mati dan permainan berakhir. Nilai HP dapat dipulihkan secara bertahap seiring waktu atau dengan mengonsumsi *item* makanan.

2) Lapar



Gambar 4. 16 Balok lapar

Menunjukkan tingkat Kelaparan pemain, yang ditampilkan dengan balok berwarna oranye. Jika balok ini habis, pemain akan mulai kehilangan HP secara perlahan. Nilai Kelaparan dapat dipulihkan dengan mengonsumsi *item* makanan.

3) Haus



Gambar 4. 17 Balok haus

Menunjukkan tingkat kehausan pemain, yang ditampilkan dengan balok berwarna biru. Jika balok ini habis, pemain akan kehilangan HP secara bertahap. Nilai ini dapat dipulihkan dengan mengonsumsi *item* minuman atau makanan.

4) Tidur



Gambar 4. 18 Balok tidur

Menunjukkan tingkat kelelahan pemain, yang ditampilkan dengan balok berwarna ungu. Jika tingkat balok tidur terlalu rendah, pemain tidak dapat berlari dan regenerasi HP akan berhenti. Tidur ini hanya dapat dipulihkan dengan tidur menggunakan ranjang.

```
private bool AnyNeedAtZero()
{
    return hunger.currentValue == 0f || thirst.currentValue ==
0f;
}
private IEnumerator ContinuousHealthDecay()
{
    float damagePerSecond = 5f;
    float damageInterval = 1.8f;
    float damageTimer = 0f;
```

```
    while (AnyNeedAtZero() && health.currentValue > 0f)
    {
        health.Subtrack(damagePerSecond * Time.deltaTime);
        damageTimer += Time.deltaTime;

if (damageTimer >= damageInterval)
        {
            onTakeDamage?.Invoke();
            damageTimer = 0f;
        }
        yield return null;
    }
    healthDecayCoroutine = null;
```

c. Serangan



Pemain menerima serangan dari musuh yang berada dalam jangkauan. Saat serangan terjadi, fungsi TakePhysicDamage(int *damage*) dipanggil untuk mengurangi HP. Jika HP ≤ 0, maka pemain mati dan UI *game over* ditampilkan. Ketika pemain terkena serangan atau HP berkurang, sistem akan men-trigger *event onTakeDamage*. Setiap kali *event* ini aktif, tampilan *Flash Damage* akan muncul di layar untuk memberi efek visual bahwa pemain sedang diserang. Efek ini berlangsung singkat dan otomatis tersembunyi setelah 0.75 detik.

```
public void TakePhysicDamage(int amount)
{
    health.Subtrack(amount);
    onTakeDamage?.Invoke();
}
public void Subtrack(float amount)
{
    float oldValue = currentValue;
    currentValue = Mathf.Max(currentValue - amount, 0);
}
public void Die()
{
    if (isGameOver) return;
    isGameOver = true;
    Cursor.visible = true;
    Cursor.lockState = CursorLockMode.None;
    gameOverPanel.SetActive(true);
    SkorText.text = SkorManager.Skor.ToString();
    daysText.text = SkorManager.Days.ToString();
    Time.timeScale = 0f;
}
```



Gambar 4. 19 Tampilan *flash damage*

Saat pemain menyerang dan mengenai musuh, sistem memeriksa komponen `*IDamagable*`. Jika ada, fungsi *TakePhysicDamage*() dipanggil dan HP musuh berkurang sesuai damage senjata..

```
IDamagable target = other.GetComponent<IDamagable>();
if (target != null)
{
    target.TakePhysicDamage(Mathf.CeilToInt(damage));
}
```

d. Input Pemain

Input pemain diatur menggunakan Unity Input System dengan skema Actions seperti pada tabel dibawah. Setiap aksi memiliki binding untuk keyboard dan *game*pad.

Tabel 4. 1 Daftar input dan fungsinya

Aksi	Tombol Keyboard/Mouse	Tombol *Gamepad*	Fungsi
Move	Wasd	*Stick* kiri	Gerakkan karakter ke depan/belakang/samping
Cursor	-	D-Pad	Arahkan kursor UI saat menggunakan *gamepad*
Click	-	B	Pilih/klik UI saat menggunakan *gamepad*
Action	Klik kiri	B, *Trigger* kanan	Serang, gunakan alat seperti kapak/pedang
AltAction	Klik kanan	Trigger kiri	Gunakan fungsi alternatif alat
Jump	Space	A	Melompat
Inventory	Tab	Y	Membuka/menutup UI inventori
Interact	E	X	Mengambil *item*, berinteraksi dengan objek
Look	Mouse delta	*Stick* kanan	Menggerakkan arah pandang kamera
Sprint	Shift kiri	Bumper Kiri	Berlari
Map	M	Select	Menampilkan peta
Pause	Esc	Start	Membuka menu jeda

e. *Animator*

Animator mengontrol transisi animasi karakter berdasarkan parameter *boolean*, *float*, dan *trigger* yang dipicu oleh sistem input. Sistem ini tertanam langsung dalam skrip PlayerController, yang memproses input dan status pemain, kemudian meneruskannya ke *Animator* melalui parameter yang telah didefinisikan.

1) *Move*



Gambar 4. 20 Animasi *move*

Saat pemain bergerak, *animator* menerima nilai MoveX dan MoveY untuk mengontrol blending animasi.

2) *Action*



Gambar 4. 21 Animasi *action*

Saat input *Action* ditekan, animasi aksi akan dipicu, dan animation *event* akan secara sinkron mengaktifkan *hitbzox* senjata. Jenis animasi yang dimainkan akan menyesuaikan dengan *item* yang sedang digunakan oleh pemain.

3) *AltAction*



Gambar 4. 22 Animasi *altaction* busur

Saat pemain menggunakan busur dan menekan input *AltAction*, parameter *isDrawing* diaktifkan. *Animator* akan transisi ke animasi menarik busur, dan kekuatan tarikan ditampilkan melalui UI berbentuk balok. Semakin lama input ditekan, semakin besar kekuatan serangan dan jarak panah saat dilepaskan. Untuk menembakkan panah tekan input *Action*.



Gambar 4. 23 Animasi *altaction* perisai

Jika pemain menggunakan perisai dan menekan input *AltAction*, parameter *isBlocking* diaktifkan. *Animator* akan memainkan animasi mengangkat perisai. Saat dalam keadaan *blocking*, serangan musuh tetap menghasilkan *damage*, tetapi hanya sebesar 20% dari *damage* sebenarnya tetapi kecepatan pemain berkurang.

4) *Jump*



Gambar 4. 24 Animasi *jump*

Saat input *Jump* ditekan dan pemain menyentuh tanah, *animator* memicu *Jump trigger* untuk memainkan animasi lompat.

f. Peralatan

Pemain dapat menggunakan berbagai jenis peralatan yang dapat di-*equip* melalui inventaris ke tangan kanan atau kiri. Setiap peralatan memiliki fungsi spesifik yang mempengaruhi interaksi pemain dengan dunia permainan. Peralatan ini juga terhubung langsung dengan sistem animasi, efek, dan interaksi objek di dunia *game*. Berikut adalah daftar peralatan dan fungsinya:

1) Kapak



Gambar 4. 25 Peralatan kapak

Saat input *Action* ditekan, pemain akan mengayunkan kapak untuk menebang pohon. Setiap ayunan mengurangi jumlah *item* pada pohon dan menambahkannya ke dalam inventaris. Jika jumlah *item* habis, pohon akan tumbang. Syarat penggunaan: Memerlukan status lapar dan haus di atas ambang minimum.



Gambar 4. 26 Pohon tumbang

2) Beliung



Gambar 4. 27 Peralatan Beliung

Saat input *Action* ditekan, pemain akan memukul Batu atau bijih Besi menggunakan Beliung. Setiap pukulan mengurangi jumlah *item* pada objek tersebut dan langsung menambahkannya ke dalam inventaris. Jika jumlah *item* habis, Batu atau bijih Besi akan hancur. Syarat penggunaan: Memerlukan status lapar dan haus di atas ambang minimum.

3) Alat Bangunan



Gambar 4. 28 Peralatan Alat Bangunan

Saat input *AltAction* ditekan, memasuki mode membangun, memungkinkan pemain memilih struktur seperti alat, pondasi, atau furniture jika memiliki bahan yang sesuai ditandai dengan UI-nya menjadi hijau.



Gambar 4. 29 Membangun api unggun

Saat kit ini aktif, sistem akan menampilkan *ghost preview* bangunan yang bisa diletakkan jika area valid. Tekan input *Action* untuk menempatkan bangunan lalu bahan yang diperlukan akan dihapus dari inventaris.

4) Pedang



Gambar 4. 30 Peralatan pedang

Senjata jarak dekat utama. Saat input *Action* ditekan, pemain melakukan ayunan pedang. Animasi ayunan diputar, dan *animation event* akan mengaktifkan *hitbox* serangan secara sinkron. Saat mengenai musuh, *damage* langsung diberikan. Syarat penggunaan: Memerlukan status lapar dan haus di atas ambang minimum.

5) Busur



Gambar 4. 31 Peralatan busur

Senjata jarak jauh. Saat input *AltAction* ditekan dan ditahan, pemain mulai menarik busur. Semakin lama tarikan, semakin besar *damage* dan jarak panah. Ketika input *Action* ditekan selama menarik busur, panah dilepaskan dan bisa mengenai musuh dari jarak jauh. Syarat penggunaan: Memerlukan status lapar dan tidur di atas ambang minimum.

6) Perisai



Gambar 4. 32 Peralatan perisai

Digunakan untuk bertahan dari serangan musuh. Saat input *AltAction* ditekan dan ditahan, pemain masuk ke mode bertahan dan animasi *block* diputar. Selama bertahan, *damage* dari serangan musuh dikurangi menjadi hanya 20% dari *damage* tetapi kecepatan pemain berkurang. **S**yarat penggunaan: Memerlukan status lapar dan tidur di atas ambang minimum.

f. Interaksi

Pemain dapat berinteraksi dengan berbagai objek di dalam *game* dengan mengarahkan crosshair ke objek tersebut dan menekan input Interact. Jenis interaksi tergantung pada objek yang dituju, antara lain:

1) Mengambil *Item*

Saat pemain mengarahkan crosshair ke *item* seperti kayu, Batu, makanan atau bahan lain, jika muncul indikasi seperti “Tekan [E] Ambil *Item*”.



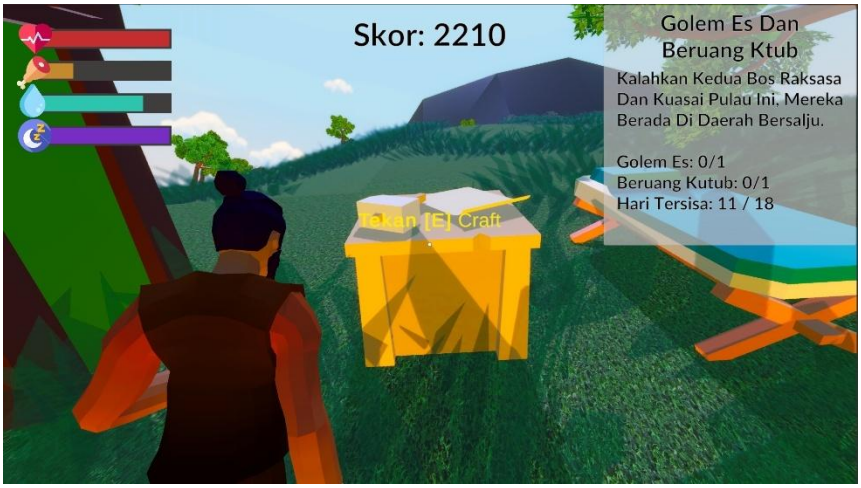
Gambar 4. 33 Interaksi mengambil *item*

Menekan input Interact, *item* akan diambil dan langsung masuk ke dalam inventaris. Setelah itu, *item* akan hilang dari dunia *game*.



Gambar 4. 34 Interaksi *item* diambil

2) Meja *Crafting*



Gambar 4. 35 Interaksi meja *crafting*

Saat pemain mengarahkan crosshair ke Meja *Crafting* dan menekan input *Interact*, UI *Crafting* akan muncul. Resep yang memenuhi syarat akan berwarna hijau dan bisa diklik.



Gambar 4. 36 UI resep *cracfting*



Gambar 4. 37 Inventaris sebelum *crafting*

Saat diklik, *crafting* akan dilakukan: bahan dihapus dari inventaris, suara *crafting* diputar, dan *item* hasil langsung ditambahkan ke inventaris.



Gambar 4. 38 Inventaris setelah *crafting*

3) Api Unggun



Gambar 4. 39 Interaksi api unggun

Saat mengarahkan *crosshair* ke Api Unggun dan menekan input Interact, api akan dinyalakan jika belum menyala. Api unggun bisa digunakan sebagai penerangan di malam hari.



Gambar 4. 40 Interaksi api unggun menyala

Tekan input Interact lagi untuk mematikannya.

4) Ranjang



Gambar 4. 41 Interaksi ranjang

Pemain dapat tidur saat malam dengan mengarahkan crosshair ke Ranjang dan menekan tombol Interact. Saat tidur, waktu akan dipercepat hingga pagi hari,

dan status pemain seperti energi akan dipulihkan. Ranjang hanya dapat digunakan saat malam tiba. Siklus waktu dalam *game* berlangsung selama 24 jam dalam satu hari permainan.



Gambar 4. 42 Interaksi ranjang siang hari

g. Inventaris

Sistem inventaris memungkinkan pemain menyimpan, mengelola, dan menggunakan berbagai *item* yang diperoleh selama permainan. Berikut penjelasan komponennya:

1) UI *Slots*



Gambar 4. 43 UI *slots* Inventaris

Inventaris terdiri dari 16 *slot* yang dapat menampung *item*. Setiap *slot* hanya dapat diisi oleh satu jenis *item*, dengan jumlah maksimum tergantung pada tipe *item*. *Slot* akan terisi secara otomatis dari urutan paling kiri ke kanan.

2) *Items*

Item memiliki tiga tipe yaitu:

a) *Consumable*

Item yang dapat digunakan sekali pakai, seperti makanan yang memberikan efek seperti memulihkan HP, kehausan, atau Kelaparan. *Item* ini habis setelah digunakan.

b) *Equipable*

Item yang dapat dipasang atau dipakai oleh pemain, seperti senjata kapak, pedang, busur atau perisai, yang dapat meningkatkan kemampuan pemain saat bertarung atau melakukan aktivitas tertentu.

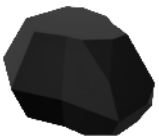
c) *Resource*

Item yang digunakan sebagai bahan dasar untuk *crafting* atau pembangunan, seperti Kayu, Batu, Besi, yang bisa diproses menjadi *item* lainnya.

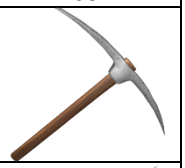
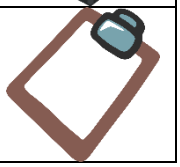


Gambar 4. 44 Inventaris *items*

Tabel 4. 2 Daftar *items* dalam game

No	Nama	Tipe	Fungsi	Stack	Icon
1	Apel	*Consumable*	Memberi: HP: 25 Kehausan: 20 Kelaparan: 30	5	
2	Pisang	*Consumable*	Memberi: HP: 30 Kehausan: 10 Kelaparan: 25	5	
3	Jeruk	*Consumable*	Memberi: HP: 15 Kehausan: 35 Kelaparan: 15	5	
4	Daging	*Consumable*	Memberi: HP: 30 Kelaparan: 40	3	
5	Kelapa	*Consumable*	Memberi: HP:15 Kehausan:40 Kelaparan: 15	5	
6	Batu	*Resource*	Digunakan untuk membuat *item*.	15	
7	Besi	*Resource*	Digunakan untuk membuat *item*.	10	
8	Kayu	*Resource*	Digunakan untuk membuat *item*.	20	
9	Panah	*Resource*	Amunisi untuk busur.	5	
10	Kapak	*Equipable*	Alat yang digunakan untuk menebang Kayu.	1	

Lanjutan Tabel 4. 2

No	Nama	Tipe	Fungsi	Stack	Icon
11	Beliung	*Equipable*	Alat yang digunakan untuk mengumpulkan Batu dan Besi.	1	
12	Pedang	*Equipable*	Digunakan untuk bertarung jarak dekat.	1	
13	Busur	*Equipable*	Digunakan untuk Bertarung jarak jauh.	1	
14	Perisai	*Equipable*	Digunakan untuk menangkis serangan musuh. (tangan kiri)	1	
15	Alat Bangunan	*Equipable*	Digunakan untuk membangun bangunan.	1	

3) Add *Item*s

Item akan ditambahkan ke *slot* kosong pertama yang tersedia. Jika *item* sudah ada dan belum mencapai jumlah maksimal, maka akan di-*stack*. Jika tidak ada *slot* kosong, *item* tidak bisa ditambahkan.

4) Deskripsi dan Fungsi Tombol

Setiap *slot* memiliki tombol interaktif. Ketika *slot* diklik, akan muncul nama *item*, deskripsi *item*, serta tombol yang berperan sesuai dengan tipe *item* di *slot* tersebut. Berikut contoh masing-masing tipe *item*:

a) *Consumable*



Gambar 4. 45 Deskripsi *item consumable*

b) *Resource*



Gambar 4. 46 Deskripsi *item resource*

c) *Equipable*



Gambar 4. 47 Deskripsi *item equipable*

Penjelasan tombol:

a) Gunakan



Gambar 4. 48 Tombol gunakan

Menggunakan *item* Consumable untuk memberikan efek langsung memulihkan HP, Kelaparan, kehausan.

b) Pasang



Gambar 4. 49 Tombol pasang

Memasang *item* Equipable (kapak, pedang, busur, perisai) ke tangan kiri atau kanan pemain.

c) Buang



Gambar 4. 50 Tombol buang

Membuang *item* ke dunia, *item* akan muncul di depan pemain.

h. Jeda

Ketika pemain menekan input Jeda, permainan akan dijeda dan tampilan menu Jeda akan muncul di layar.



Gambar 4. 51 Tampilan UI jeda

1) *Slider* Musik



Gambar 4. 52 *Slider* musik

Mengatur volume musik latar dalam *game*. Pemain bisa menyeret *slider* ke kiri untuk mengecilkan atau ke kanan untuk membesarkan volume musik.

2) *Slider* Sensitivitas



Gambar 4. 53 *Slider* sensitivitas

Mengatur sensitivitas kontrol kamera atau input *Look*.

3) Tombol Simpan



Gambar 4. 54 Tombol simpan

Untuk menyimpan progres permainan saat ini. Menyimpan *seed*, posisi pemain, status pemain, inventaris, Skor, hari dan progress misi saat ini.

4) Tombol Keluar



Gambar 4. 55 Tombol keluar

Untuk keluar dari permainan dan kembali ke menu utama.

i. Perhitungan Skor



Gambar 4. 56 Skor pemain

Sistem perhitungan skor menggunakan tipe data int sebagai representasi nilai skor yang diperoleh pemain. Variabel skor dikelola melalui kelas *ScoreManager*, yang menyimpan skor total dalam satu sesi permainan. Setiap kali pemain menyelesaikan suatu misi, sistem akan memeriksa kondisi penyelesaian melalui flag *isCompleted* pada objek misi. Jika bernilai *true*, maka skor yang terkait (*scoreValue*) akan ditambahkan ke skor utama menggunakan operasi penjumlahan langsung:

```
ScoreManager.Score += currentMission.scoreValue;
```

Skor ini kemudian ditampilkan melalui UI HUD dengan mengonversinya menjadi teks (*string*) agar dapat dibaca oleh pemain:

```
scoreText.text = ScoreManager.Score.ToString();
```

Sistem ini bekerja secara *event-based*, artinya skor hanya bertambah ketika misi selesai atau aksi tertentu yang bernilai (seperti membunuh musuh) terjadi. Selain itu, skor ini juga disimpan saat pemain melakukan *save game*, sehingga dapat dipulihkan kembali saat permainan dimuat.

4. Musuh

Musuh dalam *game* berperan sebagai tantangan utama yang harus dihindari atau dikalahkan oleh pemain. Musuh akan mendeteksi keberadaan pemain, mengejar, dan menyerang jika pemain berada dalam jarak serang. Setiap musuh memiliki jumlah nyawa (HP) dan besaran *damage* yang dihasilkan, HP musuh dapat dikurangi oleh serangan pemain. Jika HP musuh habis, maka musuh akan mati, memainkan animasi kematian, dan dapat menjatuhkan *item*.

a. Deteksi Pemain & Pengejaran

Musuh menggunakan *Physics.CheckSphere* untuk mendeteksi apakah pemain berada dalam radius penglihatan dan radius serang.

```
playerInSightRange = Physics.CheckSphere(transform.position,
sightRange, whatIsPlayer);
playerInAttackRange = Vector3.Distance(transform.position,
player.position) <= attackRange;

if (playerInSightRange && !playerInAttackRange)
    ChasingPlayer();
```

Jika pemain terdeteksi namun belum masuk radius serang, maka fungsi *ChasingPlayer()* akan membuat musuh berlari mendekati pemain.

b. Serangan dan Penerimaan *Damage*



Gambar 4. 57 *Damage* serangan

Musuh dapat menerima *damage* dari serangan pemain. Saat menerima *damage*, musuh akan bereaksi dengan memutar animasi terkena serang. Jika HP musuh habis (≤ 0), maka akan dipicu animasi kematian. Setelah itu, musuh akan dihapus dari dunia dan menjatuhkan *item* sebagai *loot* untuk pemain.



Gambar 4. 58 Menerima *damage*

Musuh menyerang jika pemain berada dalam jarak serang. Serangan dipicu lewat animasi, dan *damage* diberikan saat animasi pada animation *event* tertentu

menggunakan metode *DealsDamage().* Serangan memiliki jeda agar tidak terlalu cepat.

```
public void TakePhysicDamage(int damage)
{
    currentHP -= damage;
    animator.SetTrigger("TakeHit");
    if (currentHP <= 0)
Die();
}
void Die()
{
    animator.SetTrigger("Die");
    agent.isStopped = true;
    this.enabled = false;
    Instantiate(dropOnDeath[x].dropPrefab, transform.position,
Quaternion.identity);
    Destroy(gameObject, 5f);
}
protected virtual void DealsDamage()
{
    foreach (Collider col in hits)
    {
        if (col.CompareTag("Player"))
        {
            if (player != null)
            {

player.GetComponent<IDamagable>().TakePhysicDamage(damage);
                playerHit = true;
                break;
            }
        }
    }
}
```

c. *Animator* Musuh

Animator musuh digunakan untuk transisi antara status musuh, seperti diam, berjalan, menyerang, kena serang, dan mati. *Trigger* dan *boolean* dari *Animator* dikontrol oleh skrip.

1) Diam



Gambar 4. 59 Animasi musuh diam

Musuh akan berada dalam kondisi diam ketika tidak ada pemain dalam jarak deteksi dan tidak sedang dalam mode patroli. Animasi diam aktif saat parameter *IsRunning = false* dan *IsWalking = false*.

2) Berjalan



Gambar 4. 60 Animasi musuh berjalan

Jika musuh berada dalam mode patroli, mereka akan berjalan mengikuti titik-titik tertentu. Animasi jalan akan diaktifkan saat musuh berpindah dari satu titik patroli ke titik lain. Animasi berjalan aktif saat parameter *IsRunning = false dan IsWalking = true.*

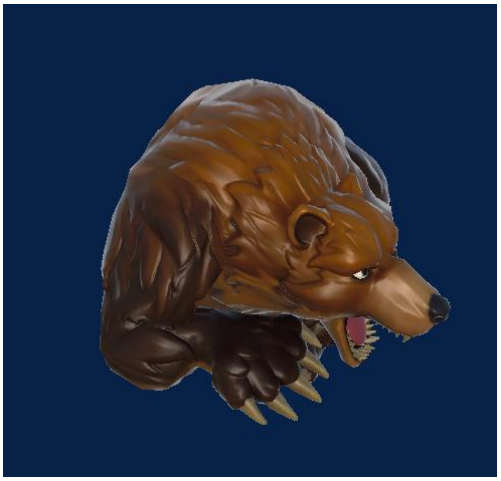
3) Berlari



Gambar 4. 61 Animasi musuh berlari

Saat pemain masuk dalam jarak deteksi, musuh berhenti patroli dan mulai mengejar. Animasi beralih ke lari. Animasi lari aktif saat parameter *IsRunning* = *true.*

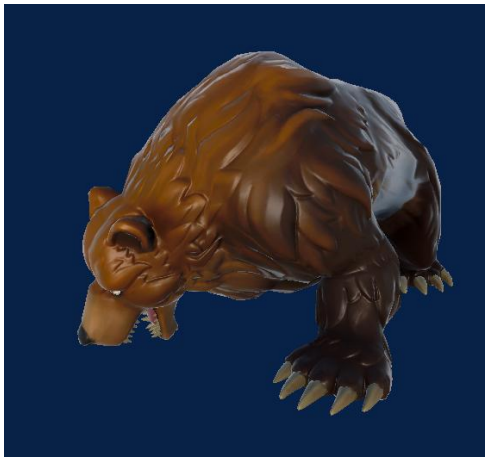
4) Menyerang



Gambar 4. 62 Animasi musuh menyerang

Jika pemain berada dalam jarak serang, musuh akan berhenti dan menyerang. Animasi serangan diaktifkan menggunakan parameter *Attack trigger*.

5) Menerima serangan



Gambar 4. 63 Animasi musuh menerima serangan

Animasi ini diaktifkan saat musuh menerima serangan dari pemain. Animasi menerima serangan akan dipicu menggunakan parameter *trigger TakeHit* ketika HP musuh turun melewati ambang tertentu, memberikan efek seperti terpukul atau terpental.

6) Mati



Gambar 4. 64 Animasi musuh mati

Jika HP musuh <= 0, animasi kematian akan aktif menggunakan parameter *trigger Die*. Musuh tidak akan bergerak atau menyerang lagi.

d. Musuh Dalam *Game*

Dalam *game* ini, terdapat berbagai jenis musuh yang tersebar di berbagai lokasi *biome* yang berbeda. Setiap musuh memiliki nilai nyawa (HP), kekuatan serangan (*damage*), lokasi kemunculan tertentu, dan kemungkinan menjatuhkan *item* saat dikalahkan. Berikut ini adalah daftar beberapa musuh yang terdapat dalam permainan:

1) Kalajengking



Gambar 4. 65 Kalajengking

HP 100, *Damage* 10, Lokasi: Hutan & Padang Rumput, *Drop*: Apel, Jeruk.

2) Bandit



Gambar 4. 66 Bandit

HP 100, *Damage* 14, Lokasi: Hutan & Padang Rumput, *Drop*: Apel, Besi.

3) Beruang



Gambar 4. 67 Beruang

HP 200, *Damage* 15, Lokasi: Hutan & Padang Rumput, *Drop*: Daging.

4) Beruang Kutub



Gambar 4. 68 Beruang kutub

HP 300, *Damage* 20, Lokasi: Salju, *Drop*: Daging.

5) Makhluk Rawa Merah



Gambar 4. 69 Makhluk Rawa Merah

HP 150, *Damage* 15, Lokasi: Hutan, *Drop*: Batu, Besi.

6) Makhluk Rawa Hijau



Gambar 4. 70 Makhluk Rawa Hijau

HP 100, *Damage* 10, Lokasi: Hutan, *Drop*: Batu.

7) Croakie



Gambar 4. 71 Croakie

HP 100, *Damage* 30, Lokasi: Pantai, *Drop*: -.

8) *Golem Pasir*



Gambar 4. 72 *Golem pasir*

HP 250, *Damage* 18, Lokasi: Pantai, *Drop*: Batu.

9) *Golem Es*



Gambar 4. 73 Golem es

HP 300, *Damage* 22, Lokasi: Salju, *Drop*: Batu, Besi.

10) Kadal Mutan Merah



Gambar 4. 74 Kadal mutan merah

HP 150, *Damage* 15, Lokasi: Pegunungan, *Drop*: Daging

11) Kadal Mutan Hijau



Gambar 4. 75 Kadal mutan hijau

HP 120, *Damage* 13, Lokasi: Pegunungan, *Drop*: Daging, Kayu.

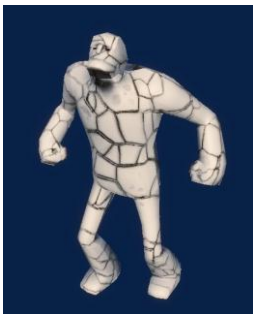
12) *Bengkorak*



Gambar 4. 76 *Bengkorak*

HP 120, *Damage* 12, Lokasi: Hutan, *Drop*: Kelapa.

13) *Mini Golem*



Gambar 4. 77 *Mini golem*

HP 130, *Damage* 9, Lokasi: Pantai, *Drop*: Kayu.

14) *Serigala*



Gambar 4. 78 *Serigala*

HP 200, *Damage* 15, Lokasi: Padang Rumput & Pegunungan, *Drop*: Daging.

5. Misi perjalanan

Dalam *game* ini terdapat total 19 misi yang terbagi ke dalam 4 jenis misi, masing-masing memiliki cara penyelesaian yang berbeda berdasarkan aksi pemain. Semua misi dijalankan secara bertahap dari awal permainan hingga misi terakhir.

a. Jenis-jenis misi

1) *CollectObjective*

Pemain harus mengumpulkan sejumlah *item* tertentu. Sistem akan memantau jumlah *item* yang dikumpulkan dan memeriksa apakah sudah memenuhi syarat.

```
if (collected<i>Item</i>ID.ToLower().Contains(req.<i>item</i>ID.ToLower()))
{
    req.currentCount++;
    CheckCompletion();
}
```

2) *KillObjective*

Pemain diminta untuk membunuh sejumlah musuh berdasarkan tipe. Setiap kali musuh yang relevan terbunuh, progres akan bertambah.

```
If(killedEnemyType.ToLower().Contains(req.enemyType.ToLower()))
{
    req.currentKills++;
    CheckCompletion();
}
```

3) *BuildObjective*

Pemain perlu membangun struktur tertentu, seperti rumah atau api unggun. Sistem akan mencatat jika bangunan yang dibangun sesuai dengan yang diminta.

```
if (buildingID.ToLower().Contains(req.buildingID.ToLower()))
{
    req.currentCount++;
    CheckCompletion();
}
```

4) *CraftObjective*

Pemain harus membuat *item* tertentu melalui sistem *crafting*. Progres meningkat saat *item* yang sesuai berhasil dibuat.

```
if (crafted<i>Item</i>ID.ToLower().Contains(req.<i>item</i>ID.ToLower()))
{
    req.currentCount++;
    CheckCompletion();
}
```

b. Kriteria penyelesaian misi

Setiap misi memiliki parameter pencapaian tertentu yang berbeda tergantung pada jenis misinya. Namun secara umum, misi dinyatakan selesai jika memenuhi dua syarat utama yaitu semua sub-tugas telah selesai berupa *array* atau *list* yang harus dipenuhi dan tidak melewati batas waktu hari yang ditentukan. Berikut kriteria penyelesaian misi berdasar jenis misi:

1) *CollectObjective*

Misi dinyatakan selesai apabila pemain telah mengumpulkan semua *item* yang dibutuhkan ke dalam inventaris, sesuai dengan ID *item* dan jumlah minimum yang ditentukan oleh misi. Sistem akan memeriksa total *item* yang dikumpulkan pemain selama misi berlangsung dan mencocokkannya dengan target jumlah dari tiap *item*.

```
return objectives.All(obj =>
inventaris.Get<i>Item</i>Amount(obj.<i>item</i>ID) >= obj.requiredAmount);
```

2) *KillObjective*

Misi dianggap selesai ketika pemain berhasil membunuh musuh-musuh sesuai jenis dan jumlah yang diminta. Setiap kali musuh dikalahkan, progres akan bertambah, dan misi akan selesai setelah seluruh target jumlah musuh terpenuhi.

```
return requiredEnemies.All(req => req.currentKills >=
req.requiredKills);
```

3) *BuildObjective*

Untuk misi membangun, pemain harus menyelesaikan pembangunan terhadap objek-objek tertentu. Misi akan dinyatakan selesai setelah jumlah bangunan yang dibutuhkan telah benar-benar selesai dibangun oleh pemain.

```
return buildTargets.All(target => target.builtCount >=
target.requiredCount);
```

4) *CraftObjective*

Pada misi *crafting*, pemain diminta untuk membuat *item* tertentu melalui meja *crafting*. Progres akan bertambah saat pemain berhasil membuat *item* dengan ID yang sesuai, dan misi dianggap selesai setelah semua target jumlah *item* yang harus dicraft terpenuhi.

```
return objectives.All(obj => obj.currentCount >=
obj.requiredCount);
```

5) Batas Hari



Gambar 4. 79 Tampilan misi gagal

Setiap misi memiliki batas waktu penyelesaian. Jika pemain menyelesaikan misi melewati batas hari tersebut, maka misi dianggap gagal dan permainan berakhir (*Game Over*):

```
timeSpent = SkorManager.Days;
if (timeSpent == timeLimit)
{
    ShowGameOver();
}
```



Gambar 4. 80 Tampilan misi selesai

Sebaliknya, jika misi berhasil diselesaikan sebelum batas waktu, maka Skor pemain akan ditambahkan sesuai nilai Skor misi dan tampilan bahwa misi telah selesai akan ditampilkan.

c. Daftar misi dalam *game*

Setiap misi terdiri dari informasi penting berupa judul, deskripsi tujuan, Skor yang diberikan, batas waktu penyelesaian, serta syarat-syarat misi yang harus dipenuhi oleh pemain

1) Misi 1



Gambar 4. 81 Misi ke-1

Judul: Memulai Survival.

Deskripsi: {nama} Ambil Perlengkapanmu untuk Memulai Bertahan hidup.

Skor: 50.

Batas waktu: Hari ke-2.

Syarat: Mengumpulkan 1 Alat Bangunan, 1 Kapak, 1 Beliung

2) Misi 2



Gambar 4. 82 Misi ke-2

Judul: Jejak Pertama {nama} di Hutan Terasing.

Deskripsi: Gunakan Kapak. Kumpulkan makanan alami dari hutan, pepohonan dan pantai.

Skor: 50.

Batas waktu: Hari ke-2.

Syarat: Mengumpulkan 3 Apel, 3 Jeruk, 3 Kelapa.

3) Misi 3



Gambar 4. 83 Misi ke-3

Judul: Sumber Daya Keras.

Deskripsi: {nama} bangun dan buat peralatan lebih lanjut, kamu butuh bahan keras. Gunakan Beliung untuk menambang Besi dan mengumpulkan Batu.

Skor: 100.

Batas waktu: Hari ke-3.

Syarat: Mengumpulkan 12 Batu, 9 Besi.

4) Misi 4



Gambar 4. 84 Misi ke-4

Judul: Meja *Crafting*.

Deskripsi: Kini saatnya membuat alat untuk survival lanjutan. {nama} Gunakan bahan mentah dan bangun Meja Craft agar kamu bisa menciptakan senjata dan perlengkapan lain.

Skor: 120.

Batas waktu: Hari ke-4.

Syarat: Membangun 1 Meja *Crafting*.

5) Misi 5



Gambar 4. 85 Misi ke-5

Judul: Persenjataan Pertama {nama}.

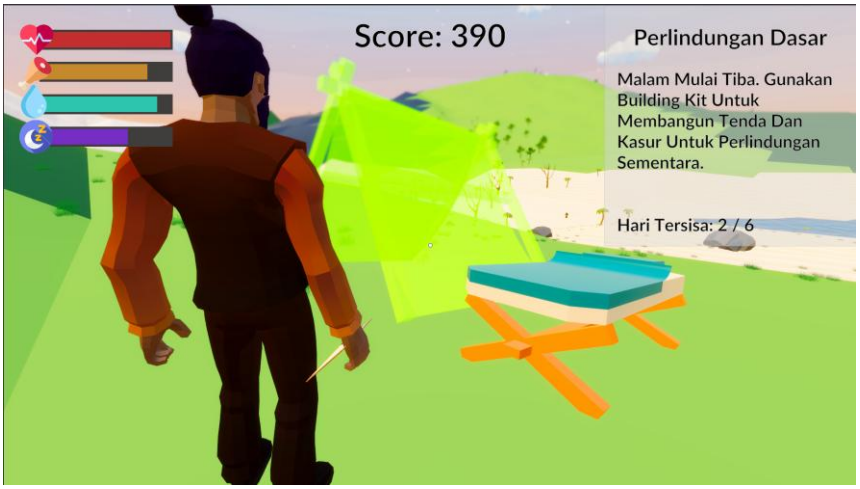
Deskripsi: Gunakan Meja *Crafting*, Kamu tidak tahu apa yang mengintai di balik semak. Buat Pedang untuk perlindungan diri.

Skor: 70.

Batas waktu: Hari ke-5.

Syarat: Membuat 1 Pedang.

6) Misi 6



Gambar 4. 86 Misi ke-6

Judul: Perlindungan Dasar.

Deskripsi: Malam mulai tiba. Gunakan Alat Bangunan untuk membangun Tenda dan Ranjang untuk perlindungan sementara.

Skor: 160.

Batas waktu: Hari ke-6.

Syarat: Membangun 1 Tenda.

7) Misi 7



Gambar 4. 87 Misi ke-7

Judul: Pembuktian Diri sejati {nama}.

Deskripsi: Makhluk asing muncul di area hutan. Mereka menyerang saat kamu mendekat. Saatnya menguji senjata pertama mu.

Skor: 150.

Batas waktu: Hari ke-7.

Syarat: Membunuh 2 Musuh.

8) Misi 8



Gambar 4. 88 Misi ke-8

Judul: Sumber Kehangatan.

Deskripsi: Tanpa api, malam akan sangat dingin. Gunakan Kayu dan Batu untuk membangun Api Unggun.

Skor: 100.

Batas waktu: Hari ke-8.

Syarat: Membangun 1 Api Unggun.

9) Misi 9



Gambar 4. 89 Misi ke-9

Judul: Serangan Tak Terduga.

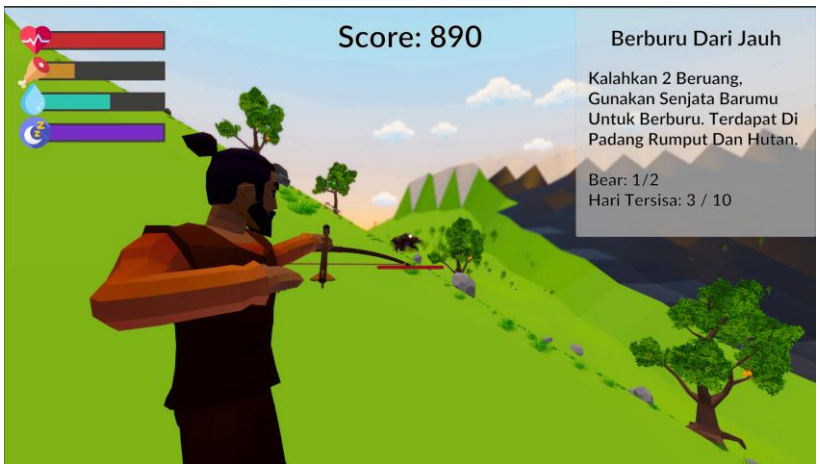
Deskripsi: {nama} Buat Busur dan Panah untuk meneyerang dari jarak jauh.

Skor: 100.

Batas waktu: Hari ke-9.

Syarat: Membuat 1 Busur dan 10 Panah.

10) Misi 10



Gambar 4. 90 Misi ke-10

Judul: Berburu dari Jauh.

Deskripsi: Kalahkan 2 Beruang, Gunakan senjata barumu untuk berburu. terdapat di padang rumput dan hutan.

Skor: 120.

Batas waktu: Hari ke-10.

Syarat: Membunuh 2 Beruang.

11) Misi 11



Gambar 4. 91 Misi ke-11

Judul: Bandit Penyerang !!.

Deskripsi: Bandit akan mengincarmu !. Kalahkan 3 Bandit yang berada di hutan dan padang rumput.

Skor: 120.

Batas waktu: Hari ke-11.

Syarat: Membunuh 3 Bandit.

12) Misi 12



Gambar 4. 92 Misi ke-12

Judul: Pertahan dalam Pertarungan.

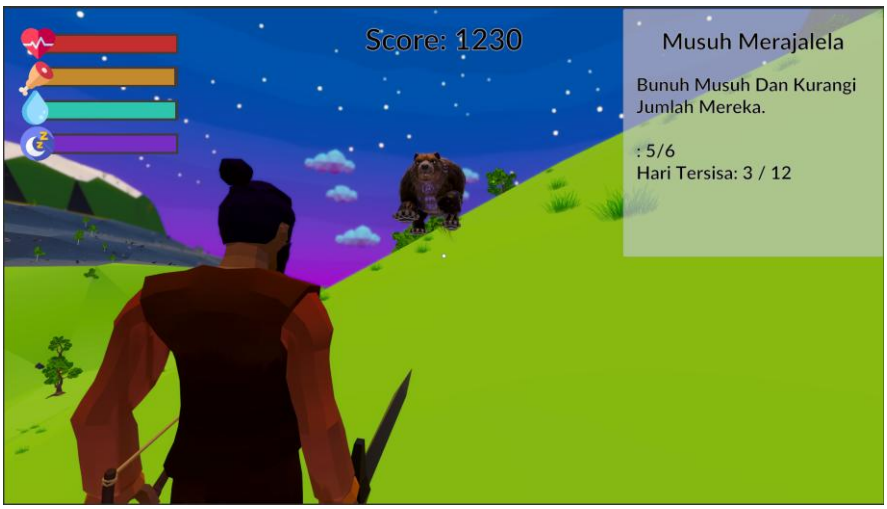
Deskripsi: {nama} Gunakan Meja *Crafting* untuk membuat Perisai.

Skor: 100.

Batas waktu: Hari ke-11.

Syarat: Membuat 1 Perisai.

13) Misi 13



Gambar 4. 93 Misi ke-13

Judul: Musuh Merajalela.

Deskripsi: Bunuh Musuh dan kurangi jumlah mereka.

Skor: 200.

Batas waktu: Hari ke-12.

Syarat: Membunuh 6 Musuh.

14) Misi 14



Gambar 4. 94 Misi ke-14

Judul: Musuh Rawa pemarah.

Deskripsi: Kalahkan makhluk rawa berbentuk katak raksasa di sekitar hutan.

Skor: 160.

Batas waktu: Hari ke-13.

Syarat: Membunuh 3 Makhluk Rawa.

15) Misi 15



Gambar 4. 95 Misi ke-15

Judul: Waspada Kadal Mutan.

Deskripsi: Kadal mutan baru baru ini terlihat dan ingin menyerang, kalahkan mereka. terdapat di pegunungan.

Skor: 160.

Batas waktu: Hari ke-14.

Syarat: Membunuh 4 Kadal Mutan.

16) Misi 16



Gambar 4. 96 Misi ke-16

Judul: Markas Baru {nama} !.

Deskripsi: Kumpulkan Sumber daya. Gunakan Alat Bangunan untuk membangun Gubuk.

Skor: 125.

Batas waktu: Hari ke-15.

Syarat: Membangun 1 Gubuk.

17) Misi 17



Gambar 4. 97 Misi ke-17

Judul: Makhluk Bengkorak.

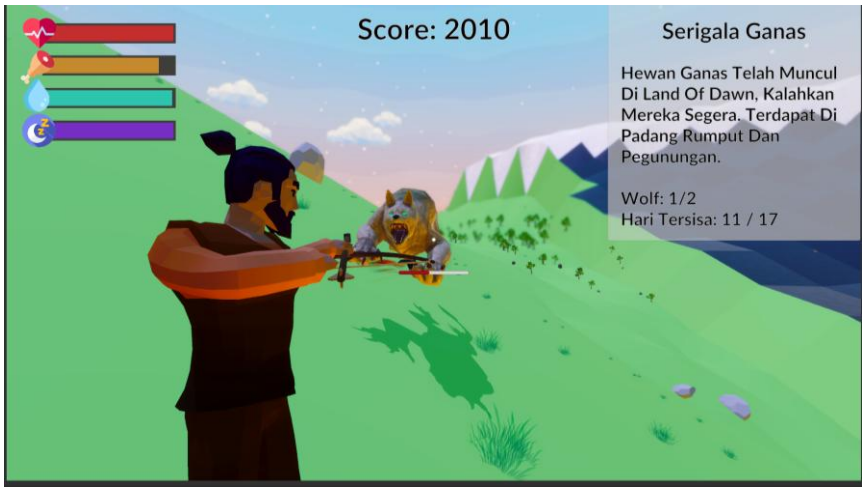
Deskripsi: Merupakan Mayat yang hidup kembali dan akan menyerang terdapat di hutan !.

Skor: 135.

Batas waktu: Hari ke-16.

Syarat: Membunuh 4 Skeleton.

18) Misi 18



Gambar 4. 98 Misi ke-18

Judul: Serigala Ganas.

Deskripsi: Hewan ganas telah muncul di Land of Dawn, Kalahkan mereka segera. terdapat di padang rumput dan pegunungan.

Skor: 200.

Batas waktu: Hari ke-17.

Syarat: Membunuh 2 Wolf.

19) Misi 19



Gambar 4. 99 Misi ke-19

Judul: Beruang kutub dan Golem es.

Deskripsi: Kalahkan Kedua Bos Raksasa dan kuasai pulau ini, mereka berada di daerah bersalju.

Skor: 600.

Batas waktu: Hari ke-18.

Syarat: Membunuh 1 Beruang kutub dan 1 Golem es.

d. Kondisi Menang

Jika misi ke-19 berhasil diselesaikan oleh pemain sebelum batas waktu maka permainan akan dianggap selesai dengan kemenangan. Sistem akan secara otomatis menampilkan UI kemenangan yang menampilkan gambar pemain serta Skornya.



Gambar 4. 100 Tampilan UI menang laki-laki



Gambar 4. 101 Tampilan UI menang perempuan

B. Perancangan Sistem

1. Sistem Generasi Peta (*Map Generator*)

Layer atau lapisan pertama pada sistem ini yaitu membuat sebuah peta (*map*) yang akan menjadi fondasi utama sistem ini. *Map* ini merupakan representasi data 2D yang akan digunakan untuk membentuk dunia 3D. Berikut adalah tahapan perancangannya:

a. Langkah pertama yaitu menentukan resolusi ukuran *map,* ukuran peta dalam bentuk grid, misalnya 512 (tinggi) x 512 (lebar) titik. Setiap titik mewakili satu area di dunia 3D dan disimpan dalam variabel *array* dua dimensi. Contoh: float[,] *heightMap* = new float[512, 512];.

b. Menghitung nilai *noise* untuk setiap koordinat titik (x, z), Setiap koordinat (x, z) dihitung nilai *noise*nya dengan fungsi *Mathf.PerlinNoise(x + seed, z + seed)*, yang menghasilkan nilai antara 0 hingga 1. Proses ini dilakukan secara iteratif menggunakan *nested loop*. *Seed* digunakan untuk menjamin hasil *noise* deterministik (selalu sama untuk *seed* yang sama).

c. Menerapkan lapisan-lapisan pembentuk elevasi (ketinggian), Ketinggian tiap titik dibentuk dari beberapa lapisan:

   1) Lapisan kontur benua: membentuk area daratan dan lautan berdasarkan *noise* dasar.

   2) Lapisan detail elevasi: *noise* skala kecil ditambahkan untuk membuat kontur bukit dan lembah.

   3) Lapisan erosi: area dengan *noise* dan elevasi tertentu dikurangi nilainya untuk mensimulasikan aliran air.

4) Lapisan sungai: jalur sungai dibentuk di area rendah berdasarkan nilai *noise* khusus.

Nilai akhir disimpan dalam *heightMap*[,].

d. Membuat array nilai *noise* suhu dan kelembaban, untuk menentukan *biome* yang lebih kompleks, ditambahkan dua *array* tambahan:
   1) *temperatureMap*[,]: menyimpan nilai suhu (0 - 1), dihasilkan dari *noise* dengan skala tertentu.
   2) humidityMap[,]: menyimpan nilai kelembaban (0 - 1), menggunakan *noise* skala berbeda untuk menciptakan variasi lembab-kering.
e. Mengklasifikasikan *biome* berdasarkan ambang nilai (*threshold*), setiap Tiap titik (x, z) diklasifikasikan ke dalam *biome* berdasarkan nilai elevasi, suhu (*temperatureMap*[x,z]), dan kelembaban (humidityMap[x,z]), dengan aturan:
   1) *Noise* < 0.1 = Laut.
   2) *Noise* > 0.15 = Pantai.
   3) *Noise* antara 0.15 – 0.6, *Temp* antara 0.4 – 0.8, *Humid* > 0.3 = Hutan.

Skrip MapGenerator:

```
using UnityEngine;
using UnityEngine.UI;
public class MapGenerator : MonoBehaviour
{
    [Header("World Settings")]
    public int width = 512;
    public int height = 512;
    public float heightMultiplier = 10f;
    public int seed = 0;
    [Header("Continents Layer")]
    public float continentScale = 100f;
    public float continentThreshold = 0.3f;
    [Header("Terrain Layer")]
    public float terrainScale = 25f;
    public float terrainMultiplier = 1f;
    [Header("Erosion Layer")]
    public float erosionScale = 50f;
    [Header("Biome Settings")]
    public float temperatureScale = 50f;
    public float humidityScale = 60f;
    public float biomeBlend = 0.05f;
    [Header("Visualization")]
    public RawImage biomeMapRawImage;
    private float[,] heightMap;
    private float[,] temperatureMap;
    private float[,] humidityMap;
    private Biome[,] biomeMap;
    public GameObject player;
    public enum Biome : int
    {
        Ocean = 0,
        Beach = 1,
        Grassland = 2,
        Forest = 3,
        Mountain = 4,
        Snow = 5
    }
    public float[,] GetHeightMap() => heightMap;
    public Biome[,] GetBiomeMap() => biomeMap;
    public float GetHeightAt(int x, int z) => heightMap[x, z];
    public Biome GetBiome(int x, int z) => biomeMap[x, z];
```

```
    public int GetMapWidth() => width;
    public int GetMapHeight() => height;
    public static MapGenerator instance;
    private void Awake()
    {
        if (instance == null)
        {
            instance = this;
            DontDestroyOnLoad(gameObject);
        }
        else Destroy(gameObject);
    }
    void Start()
    {InitializeMaps();  GenerateWorld();  }
    void Update()
    {VisualizeBiomeMap(); }
    public void GenerateWorld()
    {
        InitializeMaps();
        ApplyContinentsLayer();
        ApplyTerrainLayer();
        ApplyErosionLayer();
        ApplyRiverLayer();
        GenerateBiomeMaps();
        GenerateBiomeMap();
        VisualizeBiomeMap();
    }
    public void InitializeMaps()
    {
        heightMap = new float[width, height];
        temperatureMap = new float[width, height];
        humidityMap = new float[width, height];
        biomeMap = new Biome[width, height];
    }
    private float GetPerlinNoise(float x, float y)
    {
        float newX = x + seed;
        float newY = y + seed;
        return Mathf.PerlinNoise(newX, newY);
    }
    void ApplyContinentsLayer()
    {
        float transitionRange = 0.2f;
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                float xCoord = x / continentScale;
                float yCoord = y / continentScale;
                float sample = GetPerlinNoise(xCoord, yCoord);
                if (sample < continentThreshold)
                {
                    float t =
Mathf.InverseLerp(continentThreshold - transitionRange,
continentThreshold, sample);
```

```
                heightMap[x, y] = Mathf.SmoothStep(0f, 0.05f,
t);
            }
            else heightMap[x, y] = sample - continentThreshold;
        }
    }
}
void ApplyTerrainLayer()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            if (heightMap[x, y] <= 0) continue;

            float xCoord = x / terrainScale;
            float yCoord = y / terrainScale;
            heightMap[x, y] += GetPerlinNoise(xCoord, yCoord) *
terrainMultiplier;
        }
    }
}
void ApplyErosionLayer()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            if (heightMap[x, y] <= 0) continue;

            float xCoord = x / erosionScale;
            float yCoord = y / erosionScale;
            heightMap[x, y] -= GetPerlinNoise(xCoord, yCoord) *
erosionIntensity;
            heightMap[x, y] = Mathf.Clamp01(heightMap[x, y]);
        }
    }
}
void ApplyRiverLayer()
{
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            float xCoord = x / riverScale;
            float yCoord = y / riverScale;

            float riverNoise =
Mathf.Abs(Mathf.PerlinNoise(xCoord + seed, yCoord + seed) -
0.5f);

            if (riverNoise < riverThreshold && heightMap[x, y]
> 0.1f)
            {
                heightMap[x, y] -= 0.05f;
                biomeMap[x, y] = Biome.Ocean;
```

```
                }
            }
        }
    }
    void GenerateBiomeMaps()
    {
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                temperatureMap[x, y] = GetPerlinNoise(x /
temperatureScale, y / temperatureScale);
                humidityMap[x, y] = GetPerlinNoise(x /
humidityScale, y / humidityScale);
            }
        }
    }
    void GenerateBiomeMap()
    {
        for (int x = 0; x < width; x++)
        {
            for (int y = 0; y < height; y++)
            {
                float elevation = heightMap[x, y];
                float temp = temperatureMap[x, y];
                float humidity = humidityMap[x, y];

                if (elevation < 0.1f)
                {biomeMap[x, y] = Biome.Ocean;}
                else if (elevation < 0.15f)
                {biomeMap[x, y] = Biome.Beach;}
                else if (elevation > 0.85f)
                {biomeMap[x, y] = temp < 0.5f ? Biome.Snow :
Biome.Mountain;}
                else
                {biomeMap[x, y] = GetContinentalBiome(temp,
humidity);}
            }
        }
    }
    Biome GetContinentalBiome(float temp, float humidity)
{
    if (temp < 0.2f) return Biome.Snow;
    else if (temp < 0.4f) return humidity < 0.3f ?
Biome.Mountain : Biome.Forest;
    else if (temp < 0.6f) return humidity < 0.4f ?
Biome.Grassland : Biome.Forest;
    else if (temp < 0.8f) return humidity < 0.3f ?
Biome.Grassland : Biome.Forest;
    else return humidity < 0.4f ? Biome.Beach : Biome.Forest;
}
    void VisualizeBiomeMap()
 {
     if (PlayerController.instance.IsMapOpen)
     {
         Texture2D texture = new Texture2D(width, height);
```

```
            Vector3 playerPosition = player.transform.position;
            float worldUnitsPerPixel = 45f;
            int playerX = Mathf.FloorToInt(playerPosition.x /
worldUnitsPerPixel);
            int playerZ = Mathf.FloorToInt(playerPosition.z /
worldUnitsPerPixel);
            int markerSize = 2;
            for (int x = 0; x < width; x++)
            {
                for (int y = 0; y < height; y++)
                {
                    Color color;

                    if (biomeMap[x, y] == Biome.Ocean)
                    {
                        if (heightMap[x, y] < 0.05f) color =
new Color(0f, 0f, 0.3f);
                        else color = new Color(0.1f, 0.1f,
0.8f);
                    }
                    else color = GetBiomeColor(biomeMap[x, y]);
                    if (Mathf.Abs(x - playerX) <= markerSize &&
Mathf.Abs(y - playerZ) <= markerSize)
                    {
                        color = Color.red;
                    }
                    texture.SetPixel(x, y, color);
                }
            }
            texture.Apply();
            biomeMapRawImage.texture = texture;
        }
    }
    public Color GetBiomeColor(Biome biome)
    {
        switch (biome)
        {
            case Biome.Ocean: return new Color(0.0f, 0.4f,
0.7f);
            case Biome.Beach: return new Color(0.98f, 0.9f,
0.6f);
            case Biome.Grassland: return new Color(0.44f,
0.75f, 0.3f);
            case Biome.Forest: return new Color(0.13f, 0.55f,
0.13f);
            case Biome.Mountain: return new Color(0.5f, 0.5f,
0.5f);
            case Biome.Snow: return new Color(0.95f, 0.95f,
1.0f);
            default: return Color.magenta;
        }
    }
    public float[,] GetChunkHeightMap(int chunkX, int chunkZ,
int chunkSize)
    {
        float[,] chunkMap = new float[chunkSize, chunkSize];
```

```
        for (int x = 0; x < chunkSize; x++)
        {
            for (int z = 0; z < chunkSize; z++)
            {
                int worldX = chunkX * chunkSize + x;
                int worldZ = chunkZ * chunkSize + z;

                if (worldX < width && worldZ < height)
                    chunkMap[x, z] = heightMap[worldX, worldZ];
                else
                    chunkMap[x, z] = 0f;
            }
        }
        return chunkMap;
    }
}
```

2. Sistem Generasi Permukaan Medan (*Terrain*) dan *Chunk Streamline*

Sistem ini membentuk medan 3D berdasarkan data *heightMap*[,] dan *biomeMap*[,] yang dihasilkan oleh sistem *map generator*. Untuk menjaga penggunaan memori dalam dunia berukuran besar, digunakan sistem *Chunk Streamline*, yaitu pemrosesan dunia berdasarkan unit-unit kecil bernama chunk.

a. Pembagian dunia ke dalam *chunk*, dunia dibagi ke dalam chunk berukuran tetap, misalnya 64 x 64 unit. Setiap *chunk* akan digenerasi, di-*render*, dan dimuat secara terpisah agar lebih hemat memori.

b. Pengambilan data peta, setiap *chunk* mengambil data *heightMap*[,] dan *biomeMap*[,] dari *class* MapGenerator, yang berisi informasi elevasi dan tipe *biome* untuk wilayah tersebut.

c. Pembuatan *vertex grid, vertex grid* dibentuk dari titik-titik (x, y, z) berdasarkan elevasi (y) dari *heightMap*[,]. Setiap empat titik membentuk dua segitiga (*triangle*) untuk menciptakan permukaan *terrain*.

d. Pembentukan *mesh terrain*, *mesh* untuk setiap *chunk* dibentuk berdasarkan data elevasi. Nilai elevasi dapat dimodifikasi dengan parameter seperti

stepHeight dan *stairSmoothness* untuk menciptakan bentuk bertingkat namun tetap halus.

e. Submesh berdasarkan *biome*, setiap *biome* pada *chunk* direpresentasikan sebagai *submesh* terpisah, sehingga memungkinkan penggunaan beberapa material *biome* (hutan, salju, gurun, dll.) dalam satu *mesh terrain*.

f. Penambahan *collider* dan *renderer*. setelah *mesh* terbentuk:

   1) *MeshCollider* ditambahkan agar medan dapat berinteraksi fisik dengan pemain dan objek lainnya.

   2) *MeshRenderer* digunakan untuk menampilkan visual *terrain* dengan material sesuai *biome*.

g. Pemrosesan dinamis berdasarkan posisi pemain, hanya *chunk* yang berada dalam radius tertentu (*viewDistance*) dari posisi pemain yang dimuat dan ditampilkan. *Chunk* di luar jangkauan akan dilepas dari memori. Posisi pemain dipantau secara berkala untuk memperbarui chunk yang aktif secara dinamis.

Skrip ChunkManager:

```
using UnityEngine;
using System.Collections.Generic;
using static MapGenerator;
using System.Collections;
using System;
public class ChunkManager : MonoBehaviour
{
    public static ChunkManager instance;
    [Header("References")]
    public MapGenerator mapGen;
    public Transform player;
    [Header("Chunk Settings")]
    public int chunkSize = 64;
    public int viewDistance = 2;
    [Header("Spawning")]
    public ObjectSpawner objectSpawner;
    public EnemySpawner enemySpawner;
```

```
[Header("Water")]
public GameObject waterPlanePrefab;
private GameObject currentWaterPlane;
public float waterHeight = 60f;
[Header("Terrain Settings")]
public float scale = 50f;
public float stepHeight = 30f;
public Material[] biomeMaterials;
[Range(0f, 1f)]
public float stairSmoothness = 0.5f;
private Dictionary<Vector2Int, GameObject> loadedChunks = new
Dictionary<Vector2Int, GameObject>();
private Vector2Int currentPlayerChunk = Vector2Int.zero;
private Texture2D biomeTexture;
public System.Action OnTerrainReady;
private void Awake()
{
    if (!instance) instance = this;
    else Destroy(instance);
}
private float[,] heightMap;
private Biome[,] biomeMap;
private float chunkCheckTimer = 0f;
public float checkInterval = 15f;
void Update()
{
    chunkCheckTimer += Time.deltaTime;
    if (chunkCheckTimer >= checkInterval)
    {
        chunkCheckTimer = 0f;
        Vector2Int newChunk = GetChunkCoord(player.position);
        if (newChunk != currentPlayerChunk)
        {
            currentPlayerChunk = newChunk;
            UpdateChunks();
        }
    }
}
public void Start()
{
    mapGen.GenerateWorld();
    heightMap = mapGen.GetHeightMap();
    biomeMap = mapGen.GetBiomeMap();
    objectSpawner.Initialize();
    enemySpawner.Initialize();
    currentPlayerChunk = GetChunkCoord(player.position);
    UpdateChunks();
}
public void ResetChunks()
{
    foreach (var chunk in loadedChunks.Values) Destroy(chunk);
    loadedChunks.Clear();
    if (currentWaterPlane != null)
    {
        Destroy(currentWaterPlane);
        currentWaterPlane = null;}
```

```
    currentPlayerChunk = GetChunkCoord(player.position);
    mapGen.GenerateWorld();
    heightMap = mapGen.GetHeightMap();
    biomeMap = mapGen.GetBiomeMap();
    objectSpawner.Initialize();
    enemySpawner.Initialize();
    UpdateChunks();
}
void GenerateBiomeTexture()
{
    int width = mapGen.GetMapWidth();
    int height = mapGen.GetMapHeight();
    biomeTexture = new Texture2D(width, height);
    var biomeMap = mapGen.GetBiomeMap();
    for (int x = 0; x < width; x++)
    {
        for (int y = 0; y < height; y++)
        {
            Color color = mapGen.GetBiomeColor(biomeMap[x, y]);
            biomeTexture.SetPixel(x, y, color);
        }
    }
    biomeTexture.Apply();
}
Vector2Int GetChunkCoord(Vector3 worldPos)
{
    int x = Mathf.FloorToInt(worldPos.x / (chunkSize * scale));
    int z = Mathf.FloorToInt(worldPos.z / (chunkSize * scale));
    return new Vector2Int(x, z);
}
private readonly HashSet<Vector2Int> wantedChunks = new
HashSet<Vector2Int>();
private readonly List<Vector2Int> toRemoveChunks = new
List<Vector2Int>();
void UpdateChunks()
{
    wantedChunks.Clear();
    for (int dx = -viewDistance; dx <= viewDistance; dx++)
        for (int dz = -viewDistance; dz <= viewDistance; dz++)
        {
            var coord = new Vector2Int(currentPlayerChunk.x +
dx, currentPlayerChunk.y + dz);
            wantedChunks.Add(coord);
            if (!loadedChunks.ContainsKey(coord))
                LoadChunk(coord);
        }
    toRemoveChunks.Clear();
    foreach (var kv in loadedChunks)
        if (!wantedChunks.Contains(kv.Key))
            toRemoveChunks.Add(kv.Key);
    foreach (var coord in toRemoveChunks)
        UnloadChunk(coord);
    UpdateWaterPlane();
}
void LoadChunk(Vector2Int coord)
{
```

```
    var go = new GameObject($"Chunk_{coord.x}_{coord.y}");
    go.transform.parent = transform;
    go.layer = LayerMask.NameToLayer("Ground");
    var mf = go.AddComponent<MeshFilter>();
    var mr = go.AddComponent<MeshRenderer>();
    mr.materials = biomeMaterials;
    var mc = go.AddComponent<MeshCollider>();
    Mesh mesh = GenerateChunkMesh(coord);
    mf.mesh = mesh;
    mc.sharedMesh = mesh;
    loadedChunks[coord] = go;
    StartCoroutine(SpawnDelayed(coord, go));
}
IEnumerator SpawnDelayed(Vector2Int coord, GameObject go)
{
    yield return null;
    objectSpawner.SpawnObjectsInChunk(coord, go, chunkSize);
    enemySpawner.SpawnEnemiesInChunk(coord, go, chunkSize);
    for (int i = 0; i < 5; i++)
        yield return null;
    GameStart gameStart = FindFirstObjectByType<GameStart>();
    if (gameStart != null && !gameStart.gameStarted)
    {
        gameStart.PlacePlayerOnTerrain();
    }
}
void UnloadChunk(Vector2Int coord)
{
    Destroy(loadedChunks[coord]);
    loadedChunks.Remove(coord);
}
Mesh GenerateChunkMesh(Vector2Int coord)
{
    int worldW = mapGen.GetMapWidth();
    int worldH = mapGen.GetMapHeight();
    List<Vector3> vertices = new List<Vector3>();
    List<Vector2> uvs = new List<Vector2>();
    List<int> triangles = new List<int>();
    List<int> subMeshTriangles = new List<int>();
    List<List<int>> subMeshes = new List<List<int>>();
    for (int i = 0; i < biomeMaterials.Length; i++)
    { subMeshes.Add(new List<int>()); }
    for (int z = 0; z < chunkSize; z++)
    {
        for (int x = 0; x < chunkSize; x++)
        {
            int worldX = coord.x * chunkSize + x;
            int worldZ = coord.y * chunkSize + z;
            float h00;
            float h10;
            float h01;
            float h11;
            float h00_raw = GetHeightSafe(heightMap, worldX,
worldZ, worldW, worldH);
            float h10_raw = GetHeightSafe(heightMap, worldX +
1, worldZ, worldW, worldH);
```

```
                float h01_raw = GetHeightSafe(heightMap,
worldX, worldZ + 1, worldW, worldH);
                float h11_raw = GetHeightSafe(heightMap, worldX
+ 1, worldZ + 1, worldW, worldH);
                h00 = Mathf.Lerp(h00_raw, Mathf.Round(h00_raw /
stepHeight) * stepHeight, stairSmoothness);
                h10 = Mathf.Lerp(h10_raw, Mathf.Round(h10_raw /
stepHeight) * stepHeight, stairSmoothness);
                h01 = Mathf.Lerp(h01_raw, Mathf.Round(h01_raw /
stepHeight) * stepHeight, stairSmoothness);
                h11 = Mathf.Lerp(h11_raw, Mathf.Round(h11_raw /
stepHeight) * stepHeight, stairSmoothness);
                Biome biome = GetBiomeSafe(biomeMap, worldX,
worldZ, worldW, worldH);
                int vertIndex = vertices.Count;
                vertices.Add(new Vector3(worldX * scale, h00,
worldZ * scale));
                vertices.Add(new Vector3((worldX + 1) * scale,
h10, worldZ * scale));
                vertices.Add(new Vector3(worldX * scale, h01,
(worldZ + 1) * scale));
                vertices.Add(new Vector3((worldX + 1) * scale,
h11, (worldZ + 1) * scale));
                uvs.Add(new Vector2(0, 0));
                uvs.Add(new Vector2(1, 0));
                uvs.Add(new Vector2(0, 1));
                uvs.Add(new Vector2(1, 1));
                subMeshes[(int)biome].Add(vertIndex + 0);
                subMeshes[(int)biome].Add(vertIndex + 2);
                subMeshes[(int)biome].Add(vertIndex + 1);
                subMeshes[(int)biome].Add(vertIndex + 1);
                subMeshes[(int)biome].Add(vertIndex + 2);
                subMeshes[(int)biome].Add(vertIndex + 3);
            }
        }
        Mesh m = new Mesh();
        m.vertices = vertices.ToArray();
        m.uv = uvs.ToArray();
        m.subMeshCount = biomeMaterials.Length;
        for (int i = 0; i < biomeMaterials.Length; i++)
        { m.SetTriangles(subMeshes[i].ToArray(), i); }
        m.RecalculateNormals();
        return m;
    }
    public float GetHeightSafe(float[,] heightMap, int x, int
z, int w, int h)
    {
        if (x >= 0 && x < w && z >= 0 && z < h)
        {
            float baseHeight = heightMap[x, z] *
mapGen.heightMultiplier;
            Biome biome = GetBiomeSafe(biomeMap, x, z, w, h);
```

```
            switch (biome)
            {
                case Biome.Snow: return baseHeight =
heightMap[x, z] * (1.2f * mapGen.heightMultiplier);
                case Biome.Mountain: return baseHeight =
heightMap[x, z] * (1.15f * mapGen.heightMultiplier);
                case Biome.Ocean: return baseHeight =
heightMap[x, z] * (0.4f * mapGen.heightMultiplier);
                case Biome.Beach: return baseHeight =
heightMap[x, z] * (1.1f * mapGen.heightMultiplier);
                default:  return baseHeight;
            }
        }
        return 0f;
    }
    Biome GetBiomeSafe(Biome[,] biomeMap, int x, int z, int w,
int h)
    {
        if (x >= 0 && x < w && z >= 0 && z < h)  return
biomeMap[x, z];
        return Biome.Ocean;
    }
    void UpdateWaterPlane()
    {
        if (currentWaterPlane != null)
            Destroy(currentWaterPlane);
        float worldX = (currentPlayerChunk.x + 0.5f) *
chunkSize * scale;
        float worldZ = (currentPlayerChunk.y + 0.5f) *
chunkSize * scale;
        Vector3 waterCenter = new Vector3(worldX, waterHeight,
worldZ);
        currentWaterPlane = Instantiate(waterPlanePrefab,
waterCenter, Quaternion.identity);
        currentWaterPlane.name = "WaterPlane";
        currentWaterPlane.transform.localScale = new
Vector3(7f, 1f, 7f);
    }
}
```

3. Sistem Distribusi Objek

Sistem ini bertanggung jawab menempatkan objek-objek lingkungan seperti pohon, Batu, dan sumber daya ke dalam dunia permainan secara prosedural. Distribusi dilakukan berdasarkan data *biome*, elevasi, dan posisi *chunk*.

a. Pengelompokan objek berdasarkan *biome*, setiap jenis objek memiliki aturan spawn (*ObjectSpawnRule*) yang mencakup *prefab*, peluang muncul,

pengelompokan (klaster), dan daftar *biome* tempat objek boleh muncul. Aturan ini dikelompokkan dalam *dictionary biomeRules*, yang memetakan setiap *biome* ke daftar objek yang relevan.

b. Pemrosesan *chunk* secara lokal, objek hanya didistribusikan saat sebuah *chunk* dimuat. Fungsi *SpawnObjectsInChunk()* menangani penempatan objek dalam *chunk* aktif, berdasarkan koordinat *chunk* dan ukuran *chunk* (*chunkSize*).

c. Penentuan titik dan pola klaster, untuk setiap titik dalam *chunk*:

   1) Sistem menghasilkan nilai klaster *noise* menggunakan *Mathf.PerlinNoise*.

   2) Jika nilai *noise* melewati ambang (*clusterThreshold*) dan acakan (*spawnChance*) lolos, maka titik dianggap sebagai pusat klaster.

   3) Objek kemudian d*item*patkan di area sekeliling pusat klaster dalam radius tertentu (misalnya 9x9 unit).

d. Validasi Lingkungan dan *raycast*, sebelum menempatkan objek:

   1) Dicek apakah *biome* di titik tersebut cocok.

   2) Posisi X dan Z diberi *jitter* agar objek tidak terlalu simetris.

   3) Dilakukan *raycast* dari atas untuk memastikan permukaan valid (bukan air atau area tak boleh d*item*pati).

   4) Posisi juga dicek apakah sudah digunakan (*takenObjectPositions*) agar tidak tumpang tindih.

e. Penyesuaian tinggi dan skala objek, beberapa penyesuaian dilakukan seperti:

1) Ketinggian interpolasi dihitung dari *heightMap*[,] untuk menempatkan objek sejajar permukaan *terrain*.
2) Objek di-*instantiate* dengan rotasi acak (penuh atau hanya Y-axis).
3) Skala objek ditentukan secara deterministik berdasarkan posisi agar hasil tetap konsisten antar sesi permainan.

f. Sinkronisasi dengan sistem *chunk*, distribusi objek hanya terjadi di *chunk* yang aktif dalam radius *viewDistance* dari pemain. Saat *chunk* dilepas, objek juga akan dihapus atau disembunyikan. Ini memastikan efisiensi memori dan performa.

Skrip ObjectSpawner:

```
using UnityEngine;
using System.Collections.Generic;
using static MapGenerator;
[System.Serializable]
public class ObjectSpawnRule
{
    public GameObject objectPrefab;
    public float spawnChance;
    public float clusterFrequency;
    public float clusterThreshold;
    public List<Biome> biomes;
}
public class ObjectSpawner : MonoBehaviour
{
    public static ObjectSpawner instance;
    [Header("Object Spawning")]
    public List<ObjectSpawnRule> objectSpawnRules;
    private Dictionary<Biome, List<ObjectSpawnRule>>
biomeRules;
    private HashSet<Vector3> takenObjectPositions = new
HashSet<Vector3>();
    [Header("Terrain Settings")]
    public float scale = 50f;
    public float stepHeight = 30f;
    [Header("Raycast Settings")]
    public LayerMask terrainLayerMask;
    public LayerMask dontPlaceLayerMask;
    public MapGenerator mapGen;
    private void Awake()
    {
        if (instance == null) instance = this;
```

```
    else Destroy(gameObject);
    terrainLayerMask = LayerMask.GetMask("Ground", "Terrain");
    dontPlaceLayerMask = LayerMask.GetMask("Water",
"Interactable", "Resource");
    scale = ChunkManager.instance.scale;
}
public void Initialize()
{
    biomeRules = new Dictionary<Biome,
List<ObjectSpawnRule>>();
    if (objectSpawnRules == null)
    {
        Debug.logError("objectSpawnRules is not initialized!");
        return;
    }
    foreach (var rule in objectSpawnRules)
    {
        if (rule == null || rule.biomes == null) continue;
        foreach (var biome in rule.biomes)
        {
            if (!biomeRules.ContainsKey(biome))
                biomeRules[biome] = new
List<ObjectSpawnRule>();
            biomeRules[biome].Add(rule);
        }
    }
    if (mapGen == null) return;
}
public void SpawnObjectsInChunk(Vector2Int chunkCoord,
GameObject parent, int chunkSize)
{
    if (mapGen == null || biomeRules == null)  return;
    if (parent == null)  return;
    Random.InitState(chunkCoord.x * 73856093 ^ chunkCoord.y *
19349663);
    int startX = chunkCoord.x * chunkSize;
    int startZ = chunkCoord.y * chunkSize;
    int mapWidth = mapGen.GetMapWidth();
    int mapHeight = mapGen.GetMapHeight();
    var heightMap = mapGen.GetHeightMap();
    var biomeMap = mapGen.GetBiomeMap();
    if (heightMap == null || biomeMap == null) return;
    for (int x = 0; x < chunkSize; x++)
    {
        for (int z = 0; z < chunkSize; z++)
        {
            int worldX = startX + x;
            int worldZ = startZ + z;
            if (worldX < 0 || worldX >= mapWidth || worldZ < 0
|| worldZ >= mapHeight) continue;
            Biome biome = biomeMap[worldX, worldZ];
            if (!biomeRules.ContainsKey(biome)) continue;
            foreach (var rule in biomeRules[biome])
            {
                if (rule == null || rule.objectPrefab == null)
continue;
```

```
                float clusterNoise = Mathf.PerlinNoise(
                    worldX * rule.clusterFrequency + 1000f,
                    worldZ * rule.clusterFrequency + 1000f
                );
                if (clusterNoise >= rule.clusterThreshold &&
Random.value <= rule.spawnChance)
                {
                    int clusterRadius = 1;
                    for (int dx = -clusterRadius; dx <=
clusterRadius; dx++)
                    {
                        for (int dz = -clusterRadius; dz <=
clusterRadius; dz++)
                        {
                            int cx = worldX + dx;
                            int cz = worldZ + dz;
                            if (cx < 0 || cx >= mapWidth || cz
< 0 || cz >= mapHeight)  continue;
                            if (biomeMap[cx, cz] != biome)
continue;
                            float jitterRange = scale * 0.4f;
                            float offsetX = Random.Range(-
jitterRange, jitterRange);
                            float offsetZ = Random.Range(-
jitterRange, jitterRange);
                            float posX = cx * scale + offsetX;
                            float posZ = cz * scale + offsetZ;
                            Vector3 rayOrigin = new
Vector3(posX, 1000f, posZ);
                            float rawHeight =
GetInterpolatedHeight(heightMap, posX, posZ);
                            float steppedHeight =
Mathf.Round(rawHeight / stepHeight) * stepHeight;
                            float adjustedHeight =
Mathf.Lerp(rawHeight, steppedHeight,
ChunkManager.instance.stairSmoothness);
                            LayerMask combinedMask =
terrainLayerMask | dontPlaceLayerMask;
                            Ray ray = new Ray(rayOrigin,
Vector3.down);
                            if (Physics.Raycast(ray, out
RaycastHit hitInfo, 2000f, combinedMask))
                            {
                                if (((1 <<
hitInfo.collider.gameObject.layer) & dontPlaceLayerMask) != 0)
continue;
                                Vector3 pos = hitInfo.point;
                                if
(takenObjectPositions.Contains(pos)) continue;
                                Quaternion rotation;
                                if
(rule.objectPrefab.GetComponent<ResourceBatu>() != null)
rotation = Random.rotation;
                                else rotation =
Quaternion.Euler(0, Random.Range(0f, 360f), 0);
```

```
                                    GameObject obj =
Instantiate(rule.objectPrefab, pos, rotation,
parent.transform);
                                    int seed = cx * 73856093 ^
cz * 19349663 ^ rule.objectPrefab.name.GetHashCode();
                                    System.Random prng = new
System.Random(seed);
                                    float scaleFactor =
Mathf.Lerp(1f, 2f, (float)prng.NextDouble());
                                    obj.transform.localScale =
rule.objectPrefab.transform.localScale * scaleFactor;
                                    if (obj == null) continue;
                                    obj.name =
rule.objectPrefab.name;

takenObjectPositions.Add(pos);
                                }
                            }
                        }
                    }
                }
            }
        }
    }
    float GetInterpolatedHeight(float[,] heightMap, float
worldX, float worldZ)
    {
        if (mapGen == null || heightMap == null) return 0f;
        int mapWidth = mapGen.GetMapWidth();
        int mapHeight = mapGen.GetMapHeight();
        float scaledX = worldX / scale;
        float scaledZ = worldZ / scale;
        int x0 = Mathf.FloorToInt(scaledX);
        int x1 = x0 + 1;
        int z0 = Mathf.FloorToInt(scaledZ);
        int z1 = z0 + 1;
        x0 = Mathf.Clamp(x0, 0, mapWidth - 1);
        x1 = Mathf.Clamp(x1, 0, mapWidth - 1);
        z0 = Mathf.Clamp(z0, 0, mapHeight - 1);
        z1 = Mathf.Clamp(z1, 0, mapHeight - 1);
        float h00 = heightMap[x0, z0];
        float h10 = heightMap[x1, z0];
        float h01 = heightMap[x0, z1];
        float h11 = heightMap[x1, z1];
        float tx = scaledX - x0;
        float tz = scaledZ - z0;
        float h0 = Mathf.Lerp(h00, h10, tx);
        float h1 = Mathf.Lerp(h01, h11, tx);
        float height = Mathf.Lerp(h0, h1, tz);
        return height * mapGen.heightMultiplier;
    }
    public void MarkObjectAsTaken(Vector3 objectPosition)
    {
        try
        {
            takenObjectPositions.Add(objectPosition);
```

```
            takenObjectPositions.Add(objectPosition);
        }
        catch (System.Exception e)
        {
            Debug.logError($"Error in
ObjectSpawner.MarkObjectAsTaken(): {e.Message}");
        }
    }
}
```

4. Sistem Distribusi Musuh

Sistem ini bertugas menempatkan musuh secara prosedural ke dalam dunia permainan. Distribusi dilakukan berdasarkan *biome*, elevasi, serta skala kesulitan yang meningkat seiring waktu (hari permainan), dan hanya aktif di chunk yang dimuat.

a. Pengelompokan musuh berdasarkan *biome*, setiap musuh memiliki aturan muncul (*EnemySpawnRule*) yang menentukan *prefab*, peluang muncul, frekuensi dan ambang klaster, *biome* yang relevan, serta jumlah maksimum musuh per klaster (*maxPerCluster*). Aturan ini dikelompokkan ke dalam *biomeRules* untuk efisiensi pencarian.

b. Pemrosesan *chunk* secara dinamis, musuh hanya didistribusikan di dalam *chunk* yang aktif. Fungsi *SpawnEnemiesInChunk()* memproses penempatan musuh berdasarkan koordinat *chunk* dan ukuran *chunk* (*chunkSize*).

c. Penentuan titik dan pola klaster, untuk setiap titik dalam *chunk*:

    1) Setiap titik dalam *chunk* dievaluasi menggunakan *Mathf.PerlinNoise* untuk menentukan apakah menjadi pusat klaster musuh.

    2) Jika lolos ambang *clusterThreshold* dan *spawnChance*, maka sistem mulai menyebar musuh di sekeliling titik tersebut dalam radius klaster (misal 9x9 unit).

3) Jumlah musuh dalam klaster dibatasi oleh *maxPerCluster*, dan skala kesulitan meningkat seiring hari melalui *GetSpawnMultiplierByDay()*.

d. Validasi posisi dan elevasi, sebelum menempatkan objek:
   1) Posisi musuh diberi acakan (*jitter*) dan divalidasi dengan *raycast* dari atas untuk memastikan permukaan valid (tidak di atas air, Batu, atau objek lain).
   2) Elevasi dihitung menggunakan *heightMap*[,] dan disesuaikan dengan *stepHeight* serta *stairSmoothness* agar musuh d*item*patkan di permukaan yang benar.
   3) Posisi yang sudah digunakan disimpan dalam *takenEnemyPositions* agar tidak tumpang tindih.

e. Identifikasi dan Persistensi Musuh
   1) Setiap musuh diberi nama unik (*enemyIdentifier*) berdasarkan *prefab* dan koordinat. Nama ini digunakan untuk mencatat musuh yang sudah mati (*deadEnemies*) agar tidak di-munculkan ulang.
   2) Data kematian musuh disimpan dan dapat diambil/diperbarui agar progres dunia tetap konsisten antar sesi permainan.

f. Integrasi dengan sistem *chunk*, musuh hanya muncul dan aktif di dalam *chunk* yang sedang dimuat berdasarkan jarak pandang (*viewDistance*) pemain. Ketika *chunk* dilepas, musuh juga tidak di-munculkan ulang jika sudah mati.

Skrip EnemySpawner:

```
using UnityEngine;
using System.Collections.Generic;
using static MapGenerator;
```

```
[System.Serializable]
public class EnemySpawnRule
{
    public GameObject enemyPrefab;
    public float spawnChance;
    public float clusterFrequency;
    public float clusterThreshold;
    public List<Biome> biomes;
    public int maxPerCluster = 3;
}
public class EnemySpawner : MonoBehaviour
{
    public static EnemySpawner instance;
    [Header("Enemy Spawning")]
    public List<EnemySpawnRule> enemySpawnRules;
    private Dictionary<Biome, List<EnemySpawnRule>> biomeRules;
    private HashSet<Vector3> takenEnemyPositions = new
HashSet<Vector3>();
    [Header("Terrain Settings")]
    public float scale = 50f;
    public float stepHeight = 30f;
    [Header("Raycast Settings")]
    public LayerMask terrainLayerMask;
    public LayerMask dontPlaceLayerMask;
    public MapGenerator mapGen;
    private HashSet<string> deadEnemies = new
HashSet<string>();
    private void Awake()
    {
        if (instance == null) instance = this;
        else Destroy(gameObject);
        terrainLayerMask = LayerMask.GetMask("Ground",
"Terrain");
        dontPlaceLayerMask = LayerMask.GetMask("Water",
"Interactable", "Resource");
    }
    public void Initialize()
    {
        biomeRules = new Dictionary<Biome,
List<EnemySpawnRule>>();
        foreach (var rule in enemySpawnRules)
        {
            if (rule == null || rule.biomes == null) continue;
            foreach (var biome in rule.biomes)
            {
                if (!biomeRules.ContainsKey(biome))
biomeRules[biome] = new List<EnemySpawnRule>();
                biomeRules[biome].Add(rule);
            }
        }
    }
   public void SpawnEnemiesInChunk(Vector2Int chunkCoord,
GameObject parent, int chunkSize)
   {
       if (mapGen == null || biomeRules == null || parent ==
null) return;
```

```
Random.InitState(chunkCoord.x * 73856093 ^ chunkCoord.y *
19349663);
int startX = chunkCoord.x * chunkSize;
int startZ = chunkCoord.y * chunkSize;
int mapWidth = mapGen.GetMapWidth();
int mapHeight = mapGen.GetMapHeight();
var heightMap = mapGen.GetHeightMap();
var biomeMap = mapGen.GetBiomeMap();
for (int x = 0; x < chunkSize; x++)
{
    for (int z = 0; z < chunkSize; z++)
    {
        int worldX = startX + x;
        int worldZ = startZ + z;
        if (worldX < 0 || worldX >= mapWidth || worldZ < 0 ||
worldZ >= mapHeight) continue;
        Biome biome = biomeMap[worldX, worldZ];
        if (!biomeRules.ContainsKey(biome)) continue;
        foreach (var rule in biomeRules[biome])
        {
            float clusterNoise = Mathf.PerlinNoise(worldX *
rule.clusterFrequency + 1000f, worldZ * rule.clusterFrequency +
1000f);
            if (clusterNoise < rule.clusterThreshold ||
Random.value > rule.spawnChance) continue;
            int clusterRadius = 1;
            float multiplier = GetSpawnMultiplierByDay();
            int baseMax = rule.maxPerCluster;
            int scaledMax = Mathf.CeilToInt(baseMax *
multiplier);
            int targetSpawnCount = Random.Range(1, Mathf.Max(2,
scaledMax + 1));
            int spawnedCount = 0;
            for (int dx = -clusterRadius; dx <= clusterRadius;
dx++)
            {
                for (int dz = -clusterRadius; dz <=
clusterRadius; dz++)
                {
                    if (spawnedCount >= targetSpawnCount)
break;
                    int cx = worldX + dx;
                    int cz = worldZ + dz;
                    if (cx < 0 || cx >= mapWidth || cz < 0 ||
cz >= mapHeight) continue;
                    if (biomeMap[cx, cz] != biome) continue;
                    string enemyIdentifier =
rule.enemyPrefab.name + "_" + cx + "_" + cz;
                    if (deadEnemies.Contains(enemyIdentifier))
continue;
                    float jitterRange = scale * 0.4f;
                    float offsetX = Random.Range(-jitterRange,
jitterRange);
                    float offsetZ = Random.Range(-jitterRange,
jitterRange);
                    float posX = cx * scale + offsetX;
```

```
                        float posZ = cz * scale + offsetZ;
                        Vector3 rayOrigin = new Vector3(posX,
1000f, posZ);
                        float rawHeight =
GetInterpolatedHeight(heightMap, posX, posZ);
                        float steppedHeight =
Mathf.Round(rawHeight / stepHeight) * stepHeight;
                        float adjustedHeight =
Mathf.Lerp(rawHeight, steppedHeight,
ChunkManager.instance.stairSmoothness);
                        LayerMask combinedMask =
terrainLayerMask | dontPlaceLayerMask;
                        Ray ray = new Ray(rayOrigin,
Vector3.down);
                        if (Physics.Raycast(ray, out RaycastHit
hitInfo, 2000f, combinedMask))
                        {
                            if (((1 <<
hitInfo.collider.gameObject.layer) & dontPlaceLayerMask) != 0)
continue;
                            Vector3 pos = hitInfo.point;
                            if
(takenEnemyPositions.Contains(pos)) continue;
                            Quaternion randomRot =
Quaternion.Euler(0, Random.Range(0f, 360f), 0);
                            GameObject enemy =
Instantiate(rule.enemyPrefab, pos, randomRot,
parent.transform);
                            if (enemy != null)
                            {
                                enemy.name = enemyIdentifier;
                                takenEnemyPositions.Add(pos);
                                spawnedCount++;
                            }
                        }
                    }
                    if (spawnedCount >= rule.maxPerCluster)
break;
                }

            }
        }
    }
}
public void MarkEnemyAsDead(string enemyIdentifier)
{
    if (!string.IsNullOrEmpty(enemyIdentifier))
deadEnemies.Add(enemyIdentifier);
}
public void SetKilledEnemy(HashSet<string> killedEnemies)
{
    deadEnemies = new HashSet<string>(killedEnemies);
}
public HashSet<string> GetKilledEnemy()
{ return new HashSet<string>(deadEnemies);
}
```

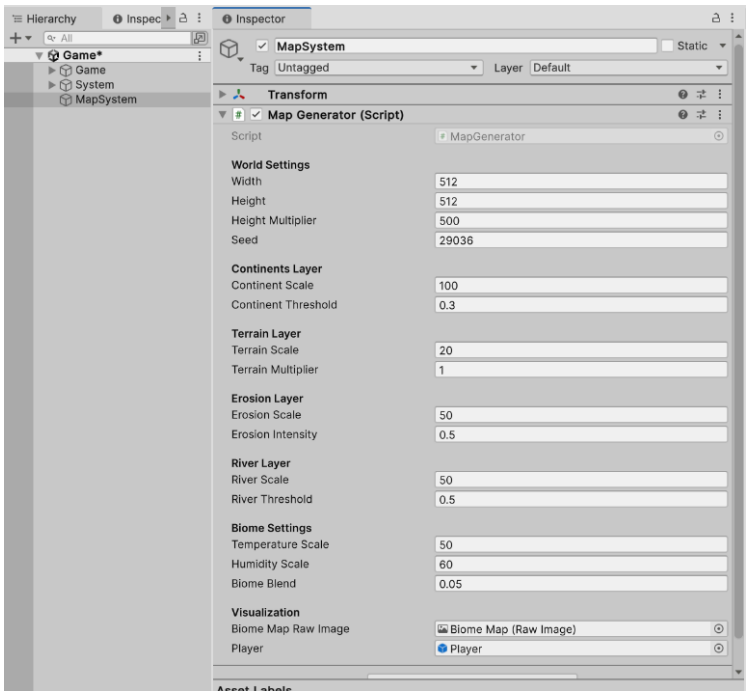
```
    float GetInterpolatedHeight(float[,] heightMap, float
worldX, float worldZ)
    {
        float scaledX = worldX / scale;
        float scaledZ = worldZ / scale;
        int mapWidth = heightMap.GetLength(0);
        int mapHeight = heightMap.GetLength(1);
        int x0 = Mathf.FloorToInt(scaledX);
        int z0 = Mathf.FloorToInt(scaledZ);
        int x1 = Mathf.Clamp(x0 + 1, 0, mapWidth - 1);
        int z1 = Mathf.Clamp(z0 + 1, 0, mapHeight - 1);
        if (x0 < 0 || z0 < 0 || x0 >= mapWidth || z0 >=
mapHeight) return 0f;
        float h00 = heightMap[x0, z0];
        float h10 = heightMap[x1, z0];
        float h01 = heightMap[x0, z1];
        float h11 = heightMap[x1, z1];
        float tx = scaledX - x0;
        float tz = scaledZ - z0;
        float h0 = Mathf.Lerp(h00, h10, tx);
        float h1 = Mathf.Lerp(h01, h11, tx);
        float height = Mathf.Lerp(h0, h1, tz);
        return height * mapGen.heightMultiplier;
    }
    private float GetSpawnMultiplierByDay()
    {
        int difficulty = PlayerPrefs.GetInt("Difficulty");
        int currentDay = Mathf.Max(SkorManager.Days, 1);
        float dayMultiplier;
        if (currentDay <= 5) dayMultiplier = Mathf.Lerp(0.25f,
1f, (currentDay - 1) / 4f);
        else if (currentDay <= 10) dayMultiplier =
Mathf.Lerp(1f, 1.5f, (currentDay - 5) / 5f);
        else dayMultiplier = 1.5f;
        float difficultyMultiplier = (difficulty == 1) ? 1.5f :
1f;
        return dayMultiplier * difficultyMultiplier;
    }
}
```

C. Implementasi Sistem

Tahap implementasi merupakan proses penerjemahan rancangan sistem ke dalam bentuk kode program yang dapat dijalankan di dalam *game engine* Unity menggunakan bahasa pemrograman C#. Berikut adalah implementasi dari masing-masing sistem:

1. Implementasi Sistem Generasi Peta (*Map Generator*)

Pertama, buat skrip MapGenerator.cs berdasarkan rancangan sistem sebelumnya untuk menghasilkan *array heightMap*[,], *temperatureMap*[,], dan *biomeMap*[,]. Lalu, di panel *hierarchy*, buat *GameObject* kosong dan beri nama MapSystem. Terakhir, tambahkan skrip tersebut ke MapSystem dengan menyeretnya ke panel *inspector* atau melalui menu *Add Component*.



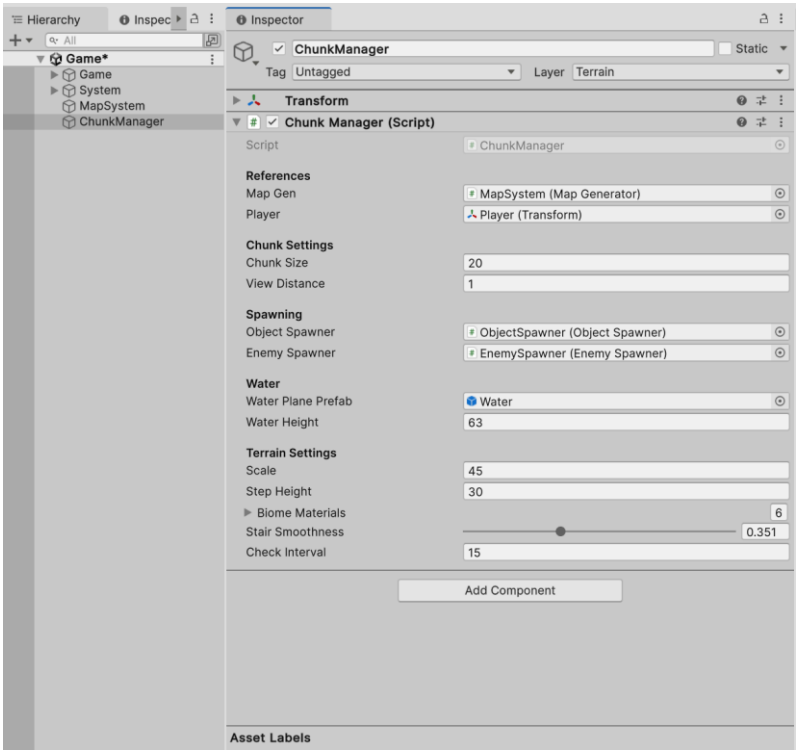
Gambar 4. 102 Implementasi sistem *map generator*.

Terdapat beberapa parameter yang dapat diubah seperti *Width*, *Height*, dan *Height Multiplier* untuk mengatur ukuran dan tinggi peta. *Seed* digunakan untuk

variasi acak. Skala dan *threshold* pada *layer Continent, Terrain, Erosion,* dan *River* mengatur bentuk dan detail permukaan. Sedangkan *Temperature Scale, Humidity Scale,* dan *Biome Blend* memengaruhi pembagian *biome*.

2. Implementasi Sistem *Terrain* dan *Chunk Streamline*

Buat skrip ChunkManager.cs berdasarkan rancangan sistem untuk mengelola pembentukan *terrain* berdasarkan data dari *heightMap*[,] dan *biomeMap*[,], serta menangani pemrosesan *chunk* secara dinamis sesuai posisi pemain. Setelah itu, di panel *hierarchy*, buat *GameObject* kosong dan beri nama ChunkManager. Terakhir, tambahkan skrip ChunkManager.cs ke *GameObject* tersebut melalui panel *inspector* atau menu *Add Component*.



Gambar 4. 103 Implementasi sistem *terrain* dan *chunk streamline.*

Terdapat beberapa parameter yang bisa diatur sesuai kebutuhan, Namun adapun parameter yang perlu diisi di panel *inspector* sebagai berikut:

a. Map Gen: Masukkan *GameObject* MapSystem ke sini.

b. *Player*: Masukkan *GameObject* player (seret *Transform*-nya).

c. *Object Spawner*: *GameObject* ObjectSpawner akan masuk kesini.

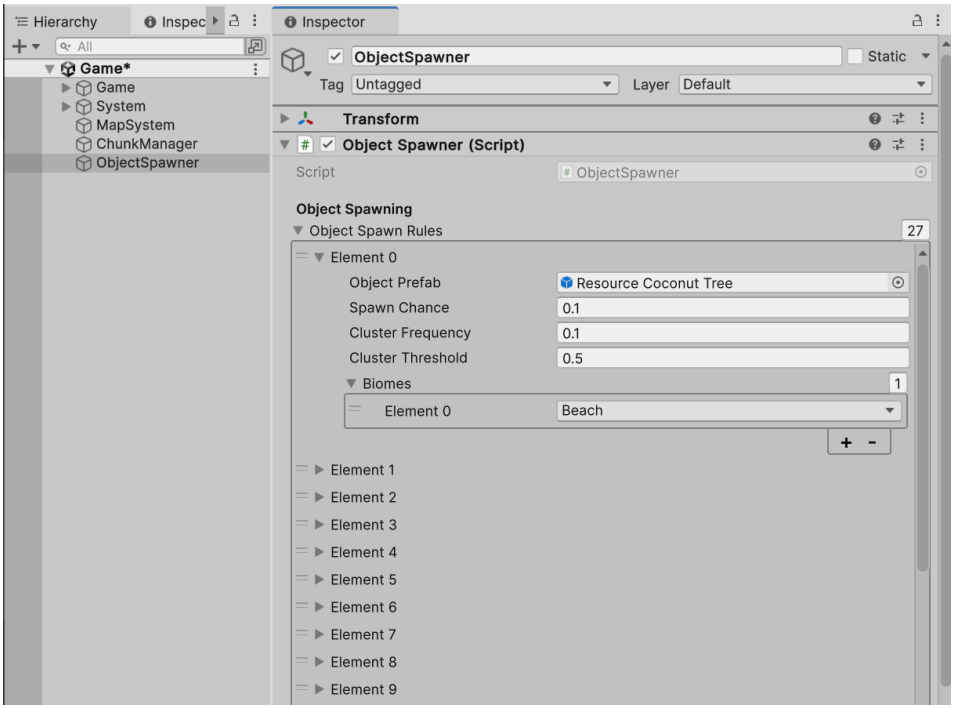
d. *Enemy Spawner*: *GameObject* EnemySpawner akan masuk kesini.

e. *Water Plane Prefabs*: Masukkan *Prefabs Water Plane* ke sini.

f. *Biome Materials*: Material untuk masing-masing *biome*.

3. Implementasi Sistem Distribusi Objek

Buat skrip ObjectSpawner.cs berdasarkan rancangan sistem sebelumnya. Tambahkan script ini ke *GameObject* baru bernama ObjectSpawner di panel *hierarchy*. Pastikan skrip ini terhubung ke ChunkManager.cs.



Gambar 4. 104 Implementasi sistem distribusi objek.

Di panel *inspector*, akan muncul bagian *Object Spawning array* objek yang akan dimunculkan. Tambahkan kan *element* dengan mengklik tanda + kemudian isi parameternya:

a. *Object Prefabs*: *Prefab* objek yang akan di-munculkan.

b. *Spawn Chance*: Peluang munculnya objek (0.1 = 10%).

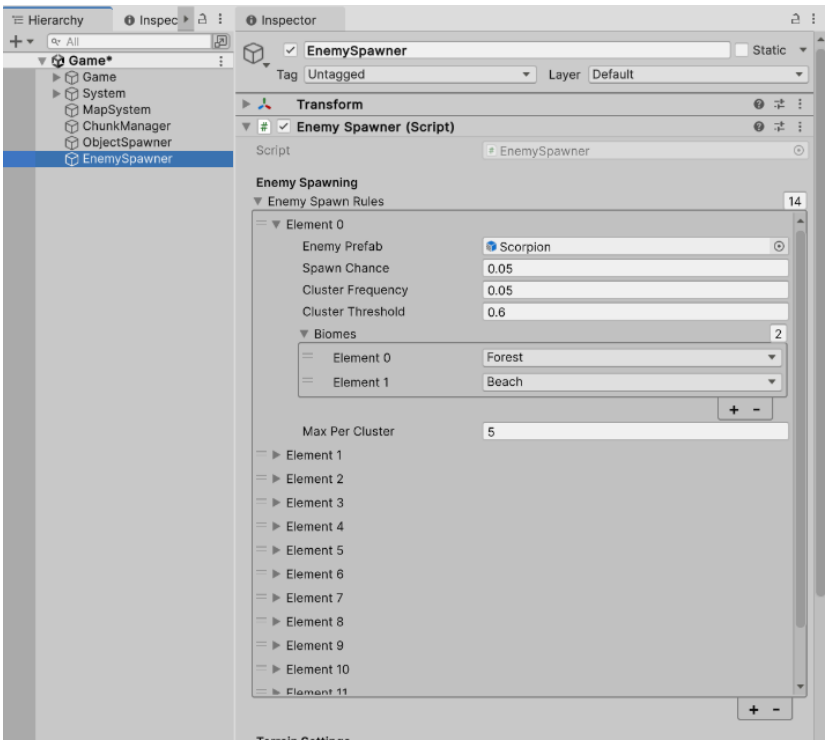
c. *Cluster Frequency*: Jarak antar klaster objek.

d. *Cluster Threshold*: Ambang batas terbentuknya klaster.

e. *Biomes*: *Biome* tempat objek bisa muncul.

4. Implementasi Sistem Distribusi Musuh

Buat skrip EnemySpawner.cs berdasarkan rancangan sistem sebelumnya. Tambahkan script ini ke *GameObject* baru bernama EnemySpawner di panel *hierarchy*. Pastikan skrip ini terhubung ke ChunkManager.cs.



Gambar 4. 105 Implementasi sistem distribusi musuh.

Di panel *inspector*, akan muncul bagian *Enemy Spawning array* objek yang akan dimunculkan. Tambahkan kan *element* dengan mengklik tanda + kemudian isi parameternya:

a. *Enemy Prefabs*: *Prefab* musuh yang akan di-munculkan.
b. *Spawn Chance*: Peluang munculnya objek (0.1 = 10%).
c. *Cluster Frequency*: Jarak antar klaster objek.
d. *Cluster Threshold*: Ambang batas terbentuknya klaster.
e. *Biomes*: *Biome* tempat musuh bisa muncul.

D. Pengujian

1. Perangkat Pengujian Spesifikasi Minimum

Selama pengujian untuk memastikan kecocokan dengan spesifikasi minimum, perangkat dikondisikan agar hanya menjalankan *game* High on Survival tanpa ada aplikasi lain yang aktif di latar belakang. Berikut adalah spesifikasi *hardware* dan *software* perangkat yang digunakan.

Tabel 4. 3 Spesifikasi *hardware* perangkat pengujian.

Spesifikasi *Hardware*		
No	**Nama**	**Keterangan**
1	*Processor*	2 *Core*
2	*RAM*	8 GB
3	*Video RAM*	128 MB
4	Penyimpanan	512 GB

Tabel 4. 4 Spesifikasi *software* perangkat pengujian.

Spesifikasi *Software*		
No	**Nama**	**Keterangan**
1	Sistem Operasi	Windows 10

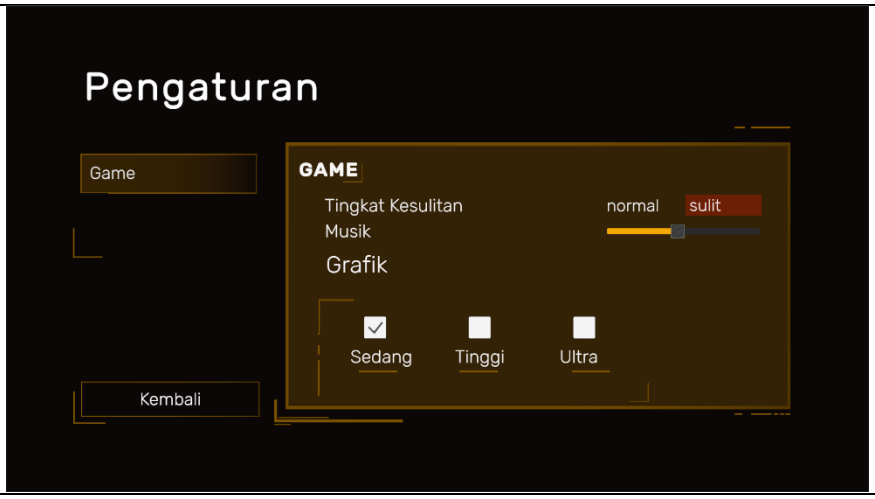
2. *Blackbox*

Pengujian *Blackbox* dapat diperhatikan pada tabel-tabel di bawah ini:

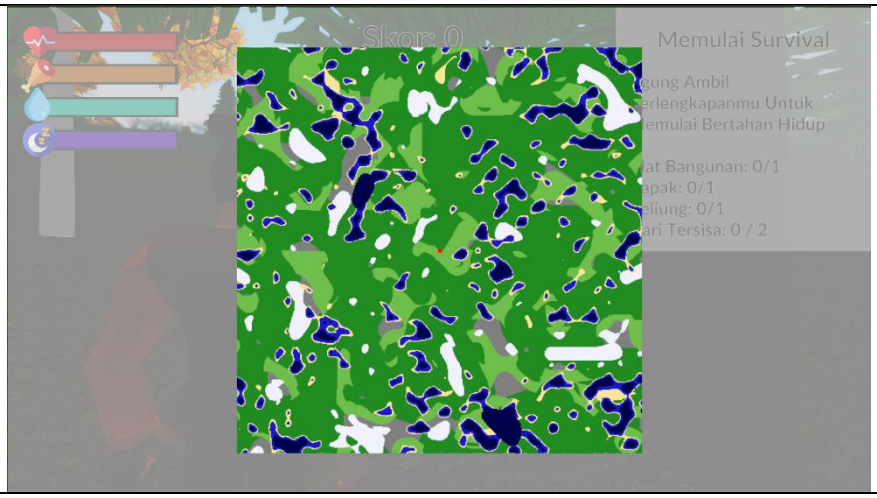
Tabel 4. 5 Pengujian *Blackbox* menu utama.

Nama	Hasil	Keterangan
Menu utama atau tampilan awal *game*	✓	Berhasil.
Screenshot		
		

Tabel 4. 6 Pengujian *Blackbox* pengaturan.

Nama	Hasil	Keterangan
Pengaturan *game*	✓	Berhasil.
Screenshot		
		

Tabel 4. 7 Pengujian *Blackbox* peta *biome.*

Nama	Hasil	Keterangan
Peta *biome*	✓	Berhasil. Peta *biome* tampil.
Screenshot		
		

Tabel 4. 8 Pengujian *Blackbox* generasi *terrain.*

Nama	Hasil	Keterangan
Generasi *Terrain*	✓	Berhasil. Permukaan medan terbentuk sesuai dengan peta.
Screenshot		
		

Tabel 4. 9 Pengujian *Blackbox* pemunculan objek.

Nama	Hasil	Keterangan
Pemunculan Objek	✓	Berhasil. Objek muncul pada *biome* yang tepat
Screenshot		
		

Tabel 4. 10 Pengujian *Blackbox* pemunculan musuh.

Nama	Hasil	Keterangan
Pemunculan Musuh	✓	Berhasil. Musuh muncul pada *biome* yang tepat.
Screenshot		
		

Tabel 4. 11 Pengujian *Blackbox* ambil *item*.

Nama	Hasil	Keterangan
Ambil *Item*	✓	Berhasil.
Screenshot		
		

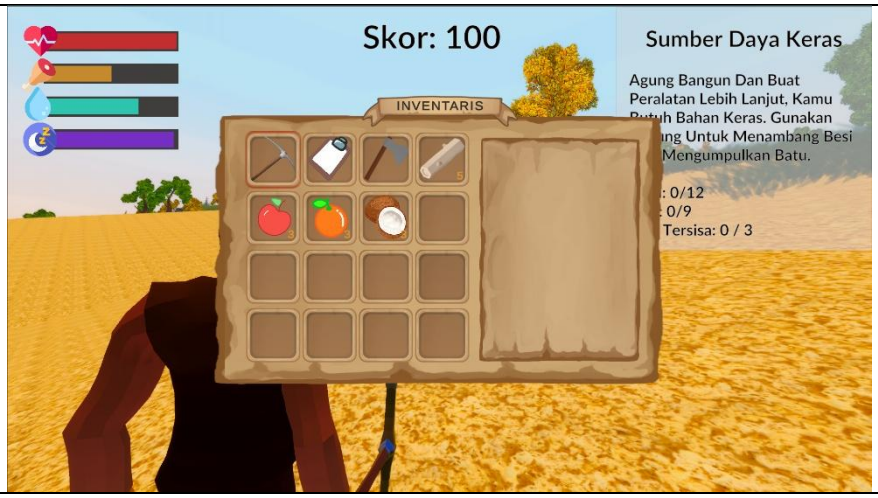
Tabel 4. 12 Pengujian *Blackbox* bunuh musuh.

Nama	Hasil	Keterangan
Bunuh musuh	✓	Berhasil.
Screenshot		
		

Tabel 4. 13 Pengujian *Blackbox* tambah Skor.

Nama	Hasil	Keterangan
Tambah Skor	✓	Berhasil. Tampilan jumlah Skor bertambah.
Screenshot		
		

Tabel 4. 14 Pengujian *Blackbox* inventaris.

Nama	Hasil	Keterangan
Inventaris	✓	Berhasil. Inventaris terbuka
Screenshot		
		

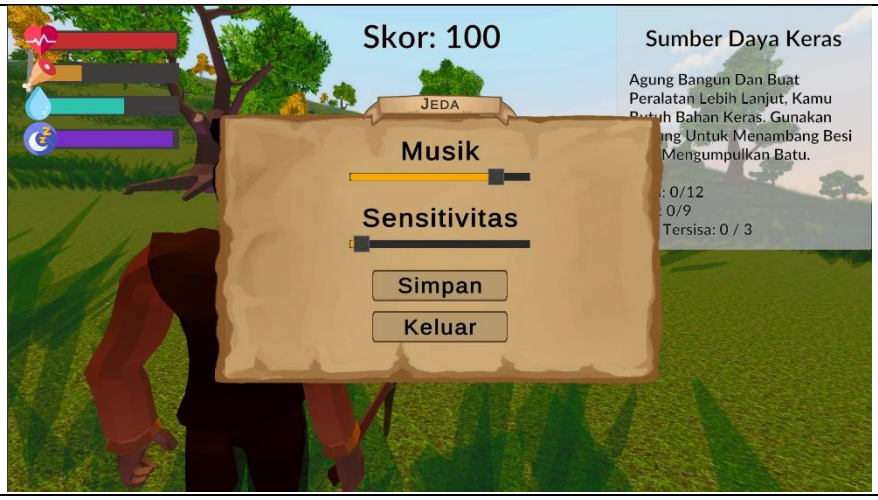
Tabel 4. 15 Pengujian *Blackbox* diserang musuh.

Nama	Hasil	Keterangan
Diserang musuh	✓	Berhasil. Layar berdarah dan *health* berkurang.
Screenshot		
		

Tabel 4. 16 Pengujian *Blackbox* jeda.

Nama	Hasil	Keterangan
Jeda	✓	Berhasil. Panel jeda terbuka dan *game* terhenti.
Screenshot		
		

Tabel 4. 17 Pengujian *Blackbox* simpan.

Nama	Hasil	Keterangan
Simpan	✓	Berhasil. Data tersimpan di penyimpanan lokal
Screenshot		
		

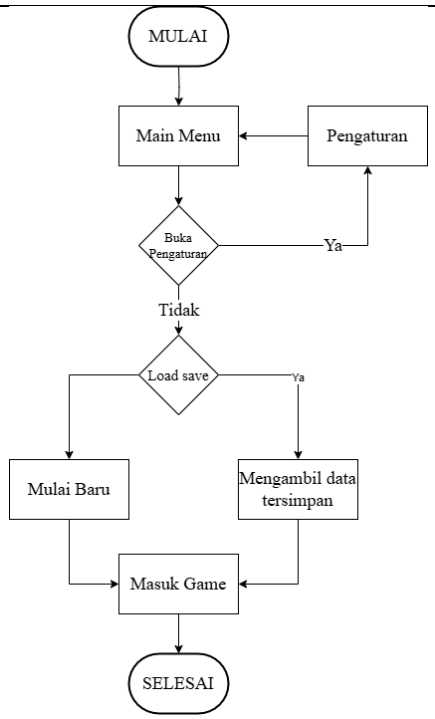
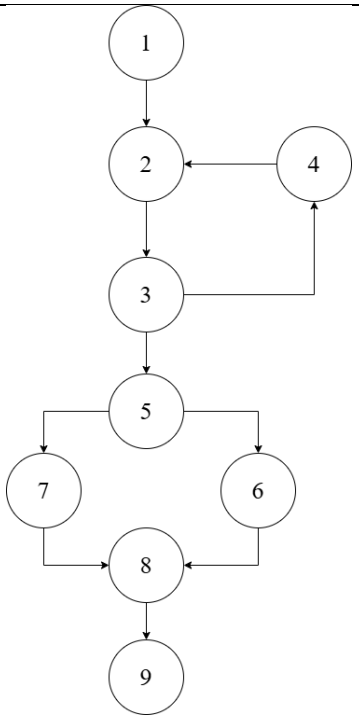
Tabel 4. 18 Pengujian *Blackbox game over*.

Nama	Hasil	Keterangan
Game Over	✓	Berhasil. Jumlah Skor dan lama bertahan muncul.
Screenshot		
		

3. White Box

a. Flowchart dan Flowgraph Menu Utama

Tabel 4. 19 *Flowchart* dan *flowgraph* menu utama.

Flowchart	*Flowgraph*
	

Dari *flowgraph* menu utama di atas dapat dilakukan proses perhitungan sebagai berikut:

1) Menentukan *Cyclomatic Complexity* V(G) berdasarkan jumlah *Edge* dan *Node* menggunakan perhitungan yang sesuai.

   Rumus: V(G) = E – N + 2

   E (*edge*) = 10

   N (node) = 9

   P (Predikat *node*) = 2

   Penyelesaian:

V(G) = E – N + 2

= 10 – 9 + 2

= 3

Predikat (P) = P + 1

= 2 + 1

= 3

2) Berdasarkan perhitungan *Cyclomatic Complexity* dari *flowgraph* menu utama memiliki *Region* = 3.

3) *Independent path flowgraph* di atas adalah:

Path 1 = 1 – 2 – 3 – 5 – 6 – 8 – 9.

Path 2 = 1 – 2 – 3 – 4 – 2 – 3 – 5 – 6 – 8 – 9.

Path 3 = 1 – 2 – 3 – 5 – 7 – 8 – 9.

4) Grafik Matrix Menu Utama.

Tabel 4. 20 Grafik matriks *whitebox* menu utama.

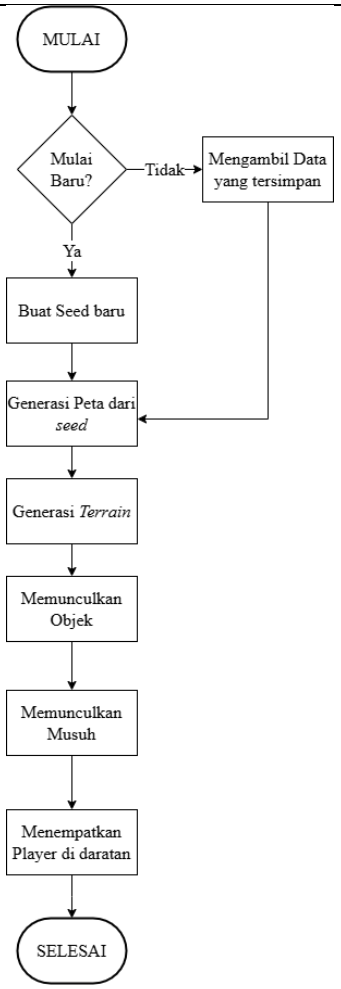
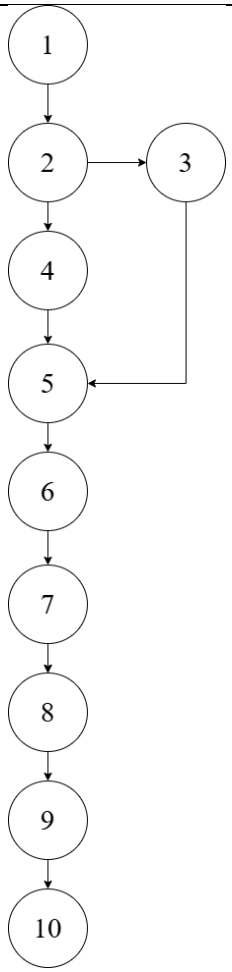
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**E – 1**
1		1								1-1=0
2			1							1-1=0
3				1	1					2-1=1
4		1								1-1=0
5						1	1			2-1=1
6								1		1-1=0
7								1		1-1=0

Lanjutan Tabel 4. 20

	1	2	3	4	5	6	7	8	9	E – 1
8									1	1-1=0
9										
SUM(E+1)										**2+1=3**

b. Flowchart dan FlowGraph *Procedural Generation*

Tabel 4. 21 *Flowchart* dan *flowgraph procedural generation*.

Flowchart	*Flowgraph*
	

Dari *flowgraph procedural generation* di atas dapat dilakukan proses perhitungan sebagai berikut:

1) Menentukan *Cyclomatic Complexity* V(G) berdasarkan jumlah *Edge* dan *Node* menggunakan perhitungan yang sesuai.

   Rumus: V(G) = E – N + 2

   E (*edge*) = 10

   N (*node*) = 10

   P (Predikat *node*) = 1

   Penyelesaian:

   V(G) = E – N + 2

   = 10 – 10 + 2

   = 2

   Predikat (P) = P + 1

   = 1 + 1

   = 2

2) Berdasarkan perhitungan *Cyclomatic Complexity* dari flowgraph menu utama memiliki *Region* = 2.

3) *Independent path flowgraph* di atas adalah:

   Path 1 = 1 – 2 – 4 – 5 – 6 – 7 – 8 – 9 – 10.

   Path 2 = 1 – 2 – 3 – 5 – 6 – 7 – 8 – 9 – 10.

4) Grafik Matrix *Procedural Generation.*

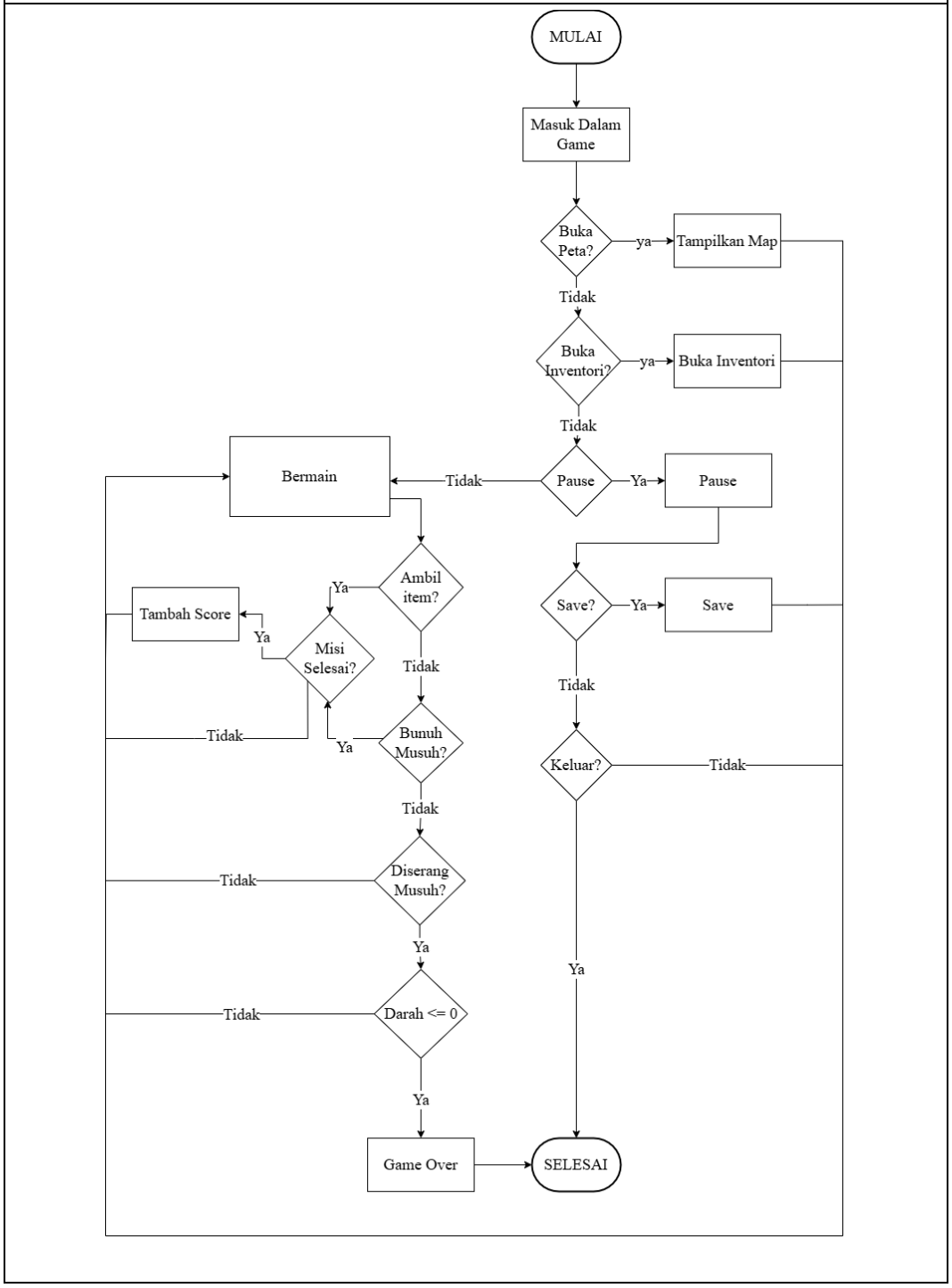
Tabel 4. 22 Grafik matriks *whitebox procedural generation.*

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**E – 1**
1		1									1–1=0
2			1	1							2–1=1
3					1						1–1=0
4					1						1–1=0
5						1					1–1=0
6							1				1–1=0
7								1			1–1=0
8									1		1–1=0
9										1	1–1=0
10											
SUM (E + 1)											**1+1=2**

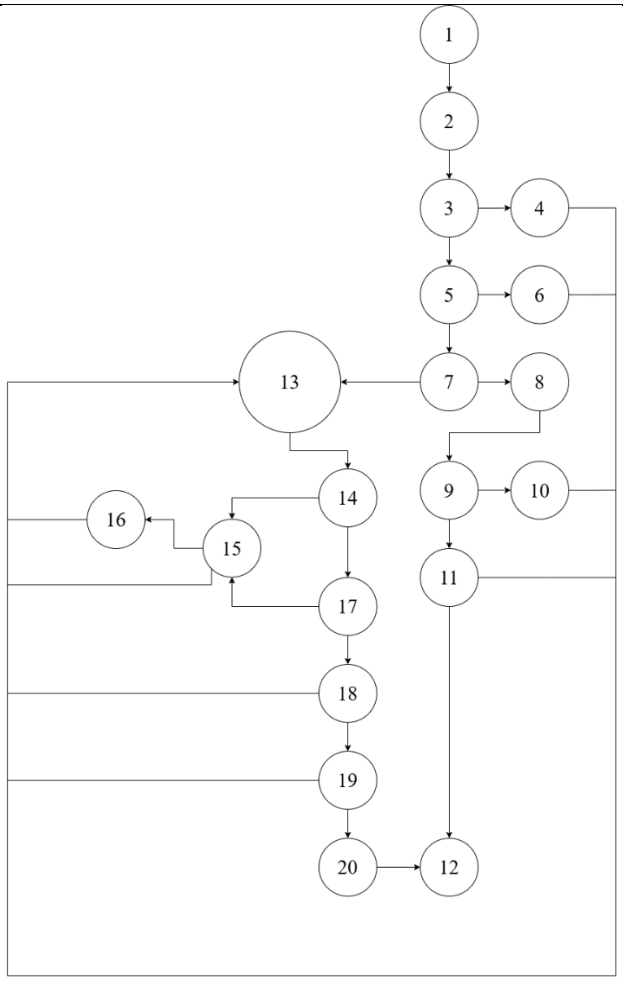
c. Flowchart dan FlowGraph *Game*

Tabel 4. 23 *Flowchart game.*

Flowchart



Tabel 4. 24 *Flowgraph game*.

Flowgraph


Dari *flowgraph game* di atas dapat dilakukan proses perhitungan sebagai berikut:

1) Menentukan *Cyclomatic Complexity* V(G) berdasarkan jumlah *Edge* dan *Node* menggunakan perhitungan yang sesuai.

   Rumus: V(G) = E – N + 2

   E (*edge*) = 29

   N (*node*) = 20

P (Predikat *node*) = 10

Penyelesaian: V(G) = E – N + 2

= 29 – 20 + 2 = 11

Predikat (P) = P + 1

= 10 + 1 = 11

2) Berdasarkan perhitungan *Cyclomatic Complexity* dari flowgraph menu utama memiliki *Region* = 11.

3) *Independent path flowgraph* di atas adalah:

Path 1 = 1 – 2 – 3 – 5 – 7 – 8 – 9 – 11 – 12.

Path 2 = 1 – 2 – 3 – 4 – 13 – 14 – 17 – 18 – 19 – 20 – 12.

Path 3 = 1 – 2 – 3 – 5 – 6 – 13 – 14 – 17 – 18 – 19 – 20 – 12.

Path 4 = 1 – 2 – 3 – 5 – 7 – 8 – 9 – 10 – 13 – 14 – 17 – 18 – 19 – 20 – 12.

Path 5 = 1 – 2 – 3 – 5 – 7 – 8 – 9 – 11 – 13 – 14 – 17 – 18 – 19 – 20 – 12.

Path 6 = 1 – 2 – 3 – 5 – 7 – 13 – 14 – 17 – 18 – 19 – 20 – 12.

Path 7 = 1 – 2 – 3 – 5 – 7 – 13 – 14 – 15 – 13 – 14 – 17 – 18 – 19 – 20 – 12.

Path 8 = 1 – 2 – 3 – 5 – 7 – 13 – 14 – 15– 16 – 13 – 14 – 17 – 18 – 19 – 20 – 12.

Path 9 = 1 – 2 – 3 – 5 – 7 – 13 – 14 – 17 – 15 – 16 – 13 – 14 – 17 – 18 – 19 – 20 – 12.

Path 10 = 1 – 2 – 3 – 5 – 7 – 13 – 14 – 17 – 18 – 13 – 14 – 17 – 18 – 19 – 20 – 12.

Path 11 = 1 – 2 – 3 – 5 – 7 – 13 – 14 – 17 – 18 – 19 – 13 – 14 – 17 – 18 – 19 – 20 – 12.

5) Grafik Matrix *Game.*

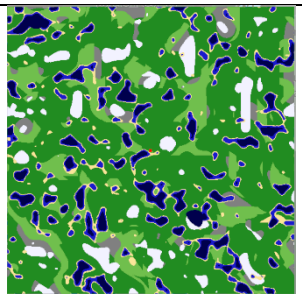
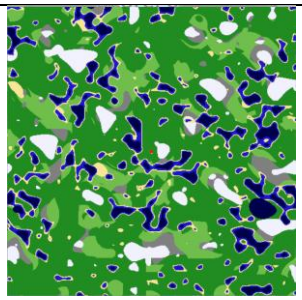
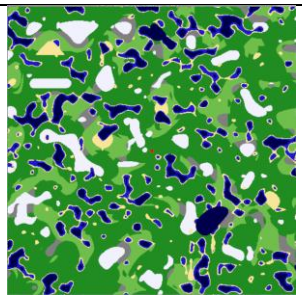
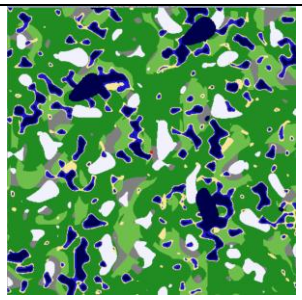
Tabel 4. 25 Grafik matriks *whitebox game*.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	E - 1
1		1																			1 - 1 = 0
2			1																		1 - 1 = 0
3				1	1																2 - 1 = 1
4													1								1 - 1 = 0
5						1	1														2 - 1 = 1
6													1								1 - 1 = 0
7								1					1								2 - 1 = 1
8									1										1		1 - 1 = 0
9										1	1										2 - 1 = 1
10													1								1 - 1 = 0
11												1	1								2 - 1 = 1
12																					
13														1							1 - 1 = 0
14															1		1				2 - 1 = 1
15																1					2 - 1 = 1
16													1								1 - 1 = 0
17															1			1			2 - 1 = 1
18																			1		2 - 1 = 1
19													1							1	2 - 1 = 1
20												1									1 - 1 = 0
	SUM (E + 1)																				10 + 1 = 11

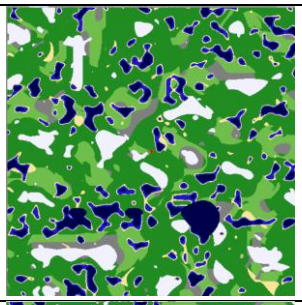
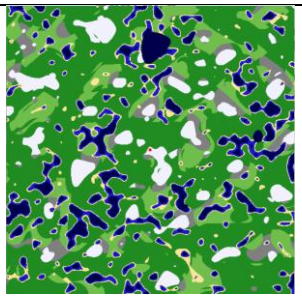
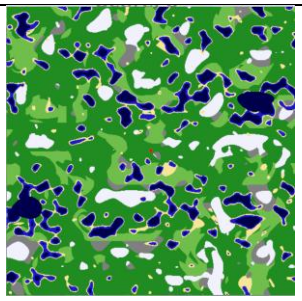
4. Pengujian Variasi Hasil Peta

Pengujian ini dilakukan untuk melihat bagaimana algoritma menghasilkan variasi peta berdasarkan *seed* yang berbeda. Berikut adalah hasilnya:

Tabel 4. 26 Hasil pengujian pembuatan peta.

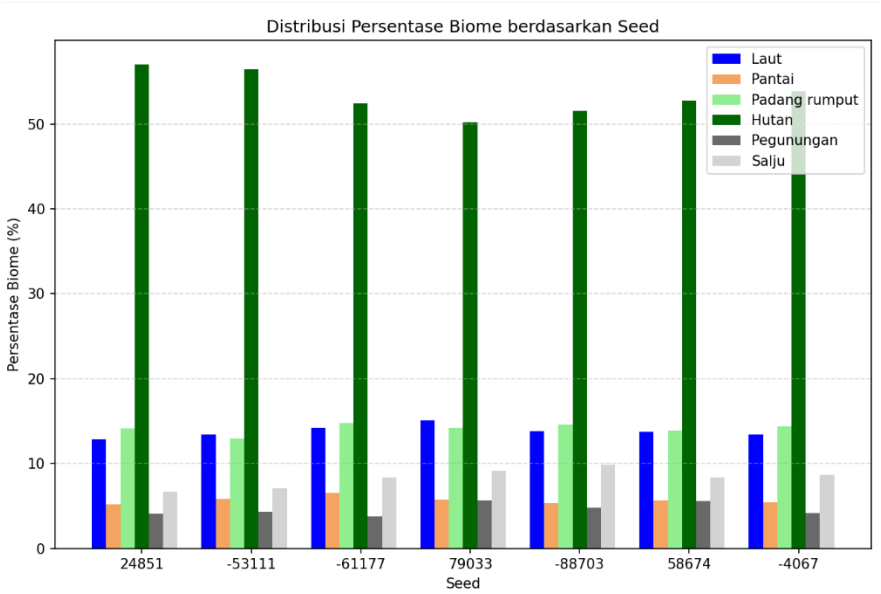
Seed	**Persentase *Biome***	**Peta**
24851	Laut: 12,9% Pantai: 5,17% Padang rumput: 14,12% Hutan: 57% Pegunungan: 4,09% Salju: 6.68%	
-53111	Laut: 13,43% Pantai: 5,79% Padang rumput: 12,92% Hutan: 56,45% Pegunungan: 4,29% Salju: 7.08%	
-61177	Laut: 14,19% Pantai: 6,52% Padang rumput: 14,75% Hutan: 52,39% Pegunungan: 3,8% Salju: 8.33%	
79033	Laut: 15,05% Pantai: 5,76% Padang rumput: 14,19% Hutan: 50,22% Pegunungan: 5,65% Salju: 9.14%	

Lanjutan Tabel 4. 26

Seed	**Persentase *Biome***	**Peta**
-88703	Laut: 13,8% Pantai: 5,34% Padang rumput: 14,6% Hutan: 51,57% Pegunungan: 4,79% Salju: 9.87%	
58674	Laut: 13,74% Pantai: 5,67% Padang rumput: 13,92% Hutan: 52,71% Pegunungan: 5,55% Salju: 8,38%	
-4067	Laut: 13,41% Pantai: 5,45% Padang rumput: 14,38% Hutan: 53,87% Pegunungan: 4,17% Salju: 8,69%	

Tabel 4. 27 Hasil distribusi persentase *biome*

No	***Seed***	**Persentase *Biome* (%)**					
		Laut	**Pantai**	**Padang rumput**	**Hutan**	**Pegunungan**	**Salju**
1	24851	12,9	5,17	14,12	57	4,09	6,68
2	-53111	13,43	5,79	12,92	56,45	4,29	7.08
3	-61177	14,19	6,52	14,75	52,39	3,8	8.33
4	79033	15,05	5,76	14,19	50,22	5,65	9.14
5	-88703	13,8	5,34	14,6	51,57	4,79	9.87
6	58674	13,74	5,67	13,92	52,71	5,55	8,38
7	-4067	13,41	5,45	14,38	53,87	4,17	8,69



Gambar 4. 106 Distribusi persentase *biome* berdasarkan *seed.*

Berdasarkan pengujian dengan 7 *seed* berbeda, waktu rata-rata untuk menghasilkan satu peta adalah 0,91 detik. Distribusi *biome* menunjukkan rata-rata persentase sebagai berikut:

- Laut: 13,79%
- Pantai: 5,67%
- Padang Rumput: 14,13%
- Hutan: 53,46%
- Pegunungan: 4,62%
- Salju: 8,31%

BAB V

KESIMPULAN

A. Kesimpulan

Berdasarkan hasil implementasi dan pengujian yang telah dilakukan, sistem *procedural generation* pada *game* aksi *survival* mampu membentuk dunia permainan yang dinamis, luas, dan variatif tanpa desain manual. Empat elemen utama berhasil dibuat secara procedural yaitu peta *biome*, permukaan medan *(terrain)* dengan sistem *chunk*, distribusi objek, dan pemunculan musuh berdasarkan karakteristik biome.

Game yang dikembangkan juga dilengkapi fitur *gameplay* seperti menu utama, HUD, status pemain, sistem inventaris, *equipment*, interaksi objek, musuh yang dapat mendeteksi dan menyerang pemain, misi, siklus siang-malam, serta sistem penyimpanan, dan dukungan penggunaan kontroler *gamepad* juga berhasil diimplementasikan dengan baik. Pengujian menggunakan metode *Blackbox*, *Whitebox*, serta uji variasi peta pada perangkat berspesifikasi minimum *processor* 2 *core*, RAM 8 GB, dan VRAM 128 MB menunjukkan bahwa game berjalan dengan baik, dengan variasi peta yang berbeda untuk setiap *seed*.

B. Saran

Pada penelitian ini masih banyak teknologii-teknologi *Procedural Generation* yang tidak digunakan, beberapa saran yang dapat dijadikan pertimbangan:

1. Disarankan untuk mengeksplorasi metode *Procedural Generation* lain seperti *Simplex Noise* atau *Wave Function Collapse* guna membandingkannya dengan *Perlin Noise*.
2. Penerapan sistem dapat ditingkatkan dengan mengintegrasikan kecerdasan buatan (AI) agar musuh memiliki perilaku yang lebih kompleks.

DAFTAR PUSTAKA

Arifudin, D., Rizky, D. R., Riyanto, A. D., & Suliswaningsih. (2024). Implementasi Algoritma Random Walk untuk Procedural Generation dalam Genre *Game* Dungeon Crawler 2D. *Infotekmesin*, 15(2): 321–329.

Fedianto, M. H. S., Aditiawan, F. P., & Al Haromainy, M. M. (2024). Pengujian Sistem Jaringan Dokumentasi Dan Informasi Menggunakan *Black Box Testing* Dan *White Box Testing*. *Jurnal Publikasi Sistem Informasi dan Manajemen Bisnis*, 3(1), 213-221.

Gumin, M. (2016). *Wave function Collapse: Algorithm for Structured Pattern Generation in Game Maps*. Diperoleh dari: https://github.com/mxgmn/WaveFunctionCollapse/. Diakses tanggal 25 Februari 2025.

Hendrix, M., Meijer, S., Velden, J. V. D., & Iosup, A. (2013). Procedural Content Generation for *Game*s: A Survey. *ACM Computing Surveys*, 45(1): 1–29.

Juwiantho, H., Liliana, L., & Budiono, M. (2023). Procedural content generation in tower defense *game* using Perlin *Noise* and Floyd Warshall algorithm. *Journal of Animation and Games Studies*, 9(1): 11–22.

Mojang. (2011). *Minecraft: Java Edition Technical Documentation*. Diperoleh dari: https://www.minecraft.net/en-us/article/java-edition-technically-updated. Diakses tanggal 25 Februari 2025.

Praniffa, A. C., Syahri, A., Sandes, F., Fariha, U., Giansyah, Q. A., & Hamzah, M. (2023). Pengujian Sistem Informasi Parkir Berbasis Web Pada UIN SUSKA RIAU Menggunakan White Box dan Black Box Testing. *Jurnal Testing dan Implementasi Sistem Informasi*, 1(1), 1–16.

Ramadhan, D. A., & Indriyanti, A. D. (2022). Procedural content generation in *game* world exploration sandbox using Perlin *Noise*. *Journal of Informatics and Computer Science*, 4(1): 86–90.

Red Blob *Game*s. (2013). *Minecraft: Java Edition Technical Documentation*. Diperoleh dari: https://www.redblob*game*s.com/maps/terrain-from-*noise*/. Diakses tanggal 25 Februari 2025.

Sampaio, P., Baffa, A., Feijó, B., & Lana, M. (2017). A Fast Approach for Automatic Generation of Populated Maps with *Seed*-based Procedural Content Generation. *IEEE Transactions on Computational Intelligence and AI in Games*, 10(2), 233–242.

Saptarini, N. G. A. P. H., Hidayat, R. A., & Ciptayani, P. I. (2019). Ajarincode: aplikasi pembelajaran bahasa pemrograman berbasis web. *Just TI (Jurnal Sains Terap. Teknol. Informasi)*, 10(2), 21.

Selviana, R., Lugata, D. B. F., & Alimin, A. (2023). Implementasi Generate Map dan Pemunculan Objek secara Acak pada *Game* 3D menggunakan Bahasa C# dan Metode Perlin *Noise* di Unity: Studi Kasus pada *Game* Development. *SPIRIT*, 15(1), 18–22.

Setiawan, H. S. (2018). Analisis Dampak Pengaruh *Game* Mobile Terhadap Aktifitas Pergaulan Siswa SDN Tanjung Barat 07 Jakarta. *Faktor exacta*, 11(2), 146–157.

Syakir, A. A. (2015). *Bahasa Pemrograman C# Berbasis Windows Application Menggunakan Editor SharpDevelop 4.4 (Program Kalkulator Sederhana)*. Sekolah Tinggi Ilmu Komputer PGRI.

Unity Technologies. (2018). *Nested Prefabs in Unity 2018.3 Documentation*. Diperoleh dari: https://docs.unity3d.com/2018.3/Documentation/Manual/NestedPrefabs.html. Diakses tanggal 25 Februari 2025.

Wahyudi, R., Utami, E., & Arief, M. R. (2016). Sistem pakar e-tourism pada Dinas Pariwisata DIY Menggunakan Metode Forward Chaining. *Data Manajemen dan Teknologi Informasi (DASI)*, 17(2), 67–75.

Wijaya, W., & Rahman, A. (2018). Analisis Perbandingan Perlin *Noise* dan Simplex *Noise* untuk Penciptaan Permukaan Daratan pada Pembuatan *Game*. *Konferensi Nasional Sistem Informasi (KNSI)*, 2018: 345–349.

Wintana, D., Pribadi, D., & Nurhadi, M. Y. (2022). Analisis Perbandingan Efektifitas *White-Box Testing* dan *Black-Box Testing*. *Jurnal Larik Ladang Artikel Ilmu Komputer*, 2(1), 8-16.