PROGRESS PROYEK AKHIR

Implementasi *Cutscene System* Sebagai Pendukung Pengembangan *Cutscene Immersive*

Oleh:

Daffa Hanif Amel Putra

5221600041

Dosen Pembimbing:

Halimatus Sa'dyah, S.Kom, M.Kom.

199007012015042001

Rizky Yuniar Hakkun, S.Kom., MT.

198106222008121003

PROGRAM STUDI SARJANA TERAPAN TEKNOLOGI GAME
JURUSAN TEKNOLOGI MULTIMEDIA KREATIF
POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

2024

[Halaman ini sengaja dikosongkan]

PROGRESS PROYEK AKHIR

Implementasi *Cutscene System* Sebagai Pendukung Pengembangan *Cutscene Immersive*

Oleh:

Daffa Hanif Amel Putra

5221600041

Dosen Pembimbing:

Halimatus Sa'dyah, S.Kom, M.Kom.

199007012015042001

Rizky Yuniar Hakkun, S.Kom., MT.

198106222008121003

PROGRAM STUDI SARJANA TERAPAN TEKNOLOGI GAME

JURUSAN TEKNOLOGI MULTIMEDIA KREATIF

POLITEKNIK ELEKTRONIKA NEGERI SURABAYA

2024

[Halaman ini sengaja dikosongkan]

ABSTRAK

Perkembangan teknologi telah menjadikan video game sebagai media yang signifikan dalam hiburan, pendidikan, dan budaya modern, menghadirkan inovasi luar biasa dalam visual, gameplay, teknologi, dan narasi. Dalam industri game, pengalaman imersif menjadi elemen kunci untuk menarik dan mempertahankan perhatian pemain, terutama melalui narasi yang disampaikan secara mendalam. Salah satu metode penting untuk mencapai hal ini adalah dengan menggunakan sistem cutscene yang efektif. Sistem cutscene memungkinkan penyajian cerita, dialog, dan peristiwa penting secara sinematik, mendukung alur cerita tanpa mengganggu gameplay.

Penelitian ini berfokus pada pengembangan sistem cutscene untuk game naratif Project Botanical, dengan menekankan fitur utama seperti dialog interaktif, animasi, perubahan kamera, efek suara, dan musik latar. Implementasi dilakukan menggunakan Unity dengan memanfaatkan fitur seperti Timeline, Animator, dan ScriptableObject untuk memastikan sinkronisasi elemen audio-visual, fleksibilitas, dan modularitas sistem. Hasilnya adalah sistem cutscene yang mendukung transisi mulus antara gameplay dan cerita, memberikan pengalaman bermain yang kohesif dan emosional.

Studi ini menunjukkan bahwa sistem cutscene yang dirancang secara modular dan terstruktur dapat memperkaya narasi, meningkatkan imersi pemain, dan menciptakan pengalaman bermain yang mendalam. Pendekatan ini diharapkan menjadi panduan bagi pengembang dalam menciptakan game story-driven yang lebih menarik dan kompetitif di industri game.

Kata Kunci : *Cutscene System, Narrative-driven Game, Unity Game Development, Modular Design, ScriptableObject*

[Halaman ini sengaja dikosongkan]

ABSTRACT

The advancement of technology has established video games as a significant medium in entertainment, education, and modern culture, offering remarkable innovations in visuals, gameplay, technology, and storytelling. In the gaming industry, immersive experiences are key to capturing and retaining players' attention, particularly through deeply engaging narratives. One crucial method to achieve this is by implementing effective cutscene systems. Cutscenes enable the cinematic presentation of stories, dialogues, and key events, enhancing the narrative without disrupting gameplay.

This study focuses on developing a cutscene system for the narrative-driven game Project Botanical, emphasizing essential features such as interactive dialogues, animations, camera transitions, sound effects, and background music. The implementation utilizes Unity's tools, including Timeline, Animator, and ScriptableObject, to ensure synchronization of audiovisual elements, system flexibility, and modularity. The resulting system supports seamless transitions between gameplay and story, delivering a cohesive and emotional player experience.

The study demonstrates that a modular and well-structured cutscene system can enrich narratives, enhance player immersion, and create a profound gaming experience. This approach aims to serve as a guide for developers in designing more engaging and competitive story-driven games in the gaming industry.

Keywords : Cutscene System, Narrative-driven Game, Unity Game Development, Modular Design, ScriptableObject

[Halaman ini sengaja dikosongkan]

DAFTAR ISI

[Halaman ini sengaja dikosongkan]

DAFTAR GAMBAR

[Halaman ini sengaja dikosongkan]

DAFTAR TABEL

[Halaman ini sengaja dikosongkan]

DAFTAR KODE

[Halaman ini sengaja dikosongkan]

BAB I
PENDAHULUAN

1.1 Latar Belakang

Seiring dengan perkembangan teknologi, *video game* telah menjadi fenomena global dalam hiburan, pendidikan, dan budaya modern. Perkembangan *video game* menghadirkan inovasi luar biasa dalam segi *visual*, *gameplay*, teknologi, dan cerita yang digunakan. Dalam industri *game*, pengalaman imersif pemain menjadi kunci dalam menarik perhatian dan mempertahankan minat pemain, terutama dengan fokus pada sisi naratif. Salah satu elemen penting dalam menciptakan pengalaman imersif pemain adalah penggunaan sistem *cutscene*, yang memungkinkan pengembang *video game* untuk menyajikan cerita, dialog, dan peristiwa penting guna memperdalam narasi yang Dinamis (Kaźmierczak et al. 2024).

Melalui *cutscene*, pemain akan dibawa dalam cerita yang menarik dan mendalam, yang memungkinkan mereka untuk lebih terhubung dengan karakter dan alur cerita. Misalnya, *cutscene* dapat digunakan untuk menampilkan percakapan penting antara Filo dan karakter lain, atau menggambarkan momen dramatis saat Filo menemukan pengetahuan baru tentang tanaman.

Pengembangan *cutscene system* yang efektif untuk *game story-driven* seperti *Game* Project Botanical memerlukan analisis yang mendalam tentang bagaimana *cutscene* dapat mendukung plot cerita tanpa mengganggu pengalaman bermain pemain. *Cutscene* juga harus dirancang sedemikian rupa sehingga mereka mengalir secara alami dalam alur cerita, memberikan informasi penting atau mengembangkan karakter tanpa membuat pemain merasa terputus dari *gameplay*. Transisi yang halus antara *gameplay* dan *cutscene* juga penting untuk memastikan pengalaman yang Imersif bagi pemain. *Hugh Hancock* Memberikan beberapa kategori pemanfaatan *cutscene* yaitu adegan percakapan, pembuangan informasi, pengaturan adegan dan suasana hati, penghargaan, pengenalan *plot* atau elemen *gameplay* dan tempo (Říha 2014).

Untuk mendukung narasi dalam *Game* Project Botanical, *cutscene system* harus memiliki fitur utama seperti kemampuan untuk menampilkan dialog, animasi karakter, perubahan kamera, efek suara,

dan musik latar. Selain itu, sistem harus mendukung *script* yang kompleks, interaksi pemain minimal, dan opsi untuk melewati atau mengulang *cutscene*. Manajemen sinematografi dalam *cutscene* juga penting untuk meningkatkan emosi dan narasi. Ini melibatkan penggunaan sudut kamera, pencahayaan, dan gerakan kamera yang efektif untuk menyoroti emosi dan narasi.

Seperti yang terlihat pada *game Stardew Valley* dan *Pokemon*, penggunaan *cutscene* dapat membuat pengalaman bermain menjadi lebih mendalam dan menarik. *Cutscene* membantu memperkuat tema naratif dan membangun dunia dalam permainan menjadi lebih imersif. Oleh karena itu, implementasi sistem *cutscene* dalam *Game* Project Botanical memiliki potensi besar untuk meningkatkan imersi pemain dan membuat pengalaman bermain lebih mendalam (Mateas & Stern 2005).

Untuk merancang sistem *cutscene* yang *modular* dan mudah dikembangkan adalah kunci untuk memastikan fleksibilitas dan efisiensi dalam pengembangan *game*. Sistem *cutscene* harus dirancang dengan komponen yang dapat digunakan kembali dan mudah dimodifikasi, memungkinkan pengembang untuk menambahkan, menghapus, atau mengubah *cutscene* tanpa harus merombak keseluruhan sistem. Ini termasuk penggunaan *scriptable objects*, *event systems*, dan *modular code structures*. Sinkronisasi antara audio, dialog, dan animasi dalam *cutscene* juga penting untuk memastikan pengalaman yang kohesif dan imersif bagi pemain.

Pada implementasi *cutscene system* di *Unity* memerlukan pendekatan yang terstruktur dan efektif. *Unity* menyediakan berbagai alat dan fitur yang dapat digunakan untuk mengembangkan *cutscene system*, seperti *Timeline*, *Animator*, *ScriptableObject*, *Unity Event* dan berbagai komponen *scripting* lainya. Implementasi ini harus memastikan bahwa semua elemen *cutscene* dapat bekerja sesuai dengan desain awal, mulai dari animasi dan audio hingga transisi dan interaksi pemain. Memastikan performa yang optimal dan responsivitas sistem juga penting untuk menjaga pengalaman bermain yang lancar dan imersif. Dengan pendekatan yang terstruktur ini, pengembang dapat memanfaatkan alat-alat yang ada di *Unity* untuk menciptakan *cutscene* yang tidak hanya fungsional tetapi juga dapat memberikan pengalaman yang mendalam dan memuaskan bagi pemain.

Implementasi sistem *cutscene* yang efisien sangat penting dalam menciptakan pengalaman bermain pemain yang mendalam, terutama untuk *game* naratif seperti *Game* Project Botanical. Sistem *cutscene* yang baik memperkaya narasi dan karakter dalam *game* serta memastikan transisi yang mulus antara gameplay dan cerita. Teknologi di *Unity*, seperti *Timeline*, *Animator*, dan *ScriptableObject*, memberikan alat untuk merancang *cutscene* yang kompleks dan dinamis, namun memerlukan pendekatan terstruktur untuk memastikan sinkronisasi audio, dialog, animasi, serta fleksibilitas dan modularitas sistem. Dengan memahami pentingnya dan tantangan dalam pengembangan *cutscene system*, pengembang dapat menciptakan *game* yang menarik secara *visual* dan *gameplay*, sambil menyajikan cerita yang kuat dan emosional, memberikan pengalaman bermain yang tak terlupakan dan memperkuat posisi *game* dalam industri (Říha 2014).

1.2 Rumusan Masalah

Dari latar belakang yang telah dipaparkan, dibuat beberapa rumusan masalah yaitu :

1. Apa saja kebutuhan *fungsional* dan *non fungsional* dalam Project Botanical?
2. Bagaimana arsitektur sistem yang mencakup kebutuhan fungsional dan *non fungsional* Project Botanical ?
3. Bagaimana memvalidasi sistem yang telah dikembangkan untuk Project Botanical ?

1.3 Batasan Masalah

Agar penelitian ini lebih efektif, efisien, terarah dan dapat dikaji lebih mendalam maka diperlukan pembatasan masalah.

Adapun pembatasan masalah yang dikaji dalam penelitian ini adalah: Masalah yang diteliti terbatas pada pengembangan dan implementasi sistem *cutscene* dalam permainan “Project Botanical”

1.4 Tujuan Proyek Akhir

Tujuan Penelitian ini adalah memberikan pengetahuan terkait bagaimana Implementasi *Cutscene System* untuk mendukung Pengembangan *Cutscene Immersive*

1.5 Metodologi

Ada beberapa metodologi yang digunakan selama pengerjaan, berikut diantaranya

1.5.1 Analisa Kebutuhan

Di tahap ini, dilakukan analisa kebutuhan melalui diskusi dengan *game designer* dan studi literatur untuk mengidentifikasi elemen-elemen yang diperlukan dalam pengembangan sistem *cutscene*. Kebutuhan yang telah ditemukan akan dijadikan dasar untuk tahap pengembangan.

1.5.2 Perancangan Arsitektur

Pada tahap ini, dilakukkan perancangan arsitektur sistem *cutscene*, mencakup desain elemen fungsional seperti dialog, animasi, dan kamera, serta penyusunan *class diagram* untuk memetakan hubungan antar komponen.

1.5.3 Implementasi

Ini adalah tahap di mana saya melakukan implementasi sistem *cutscene* menggunakan *Unity*. Proses ini mencakup pengembangan komponen sistem, integrasi antar modul, serta pengujian fungsional sistem.

1.5.4 Dokumentasi

Di tahap ini, seluruh proses pengembangan, mulai dari analisa kebutuhan, perancangan, hingga implementasi, didokumentasikan sebagai bahan laporan akhir dan referensi untuk pengembangan lanjutan.

BAB II
TINJAUAN PUSTAKA

2.1 Penelitian Terkait

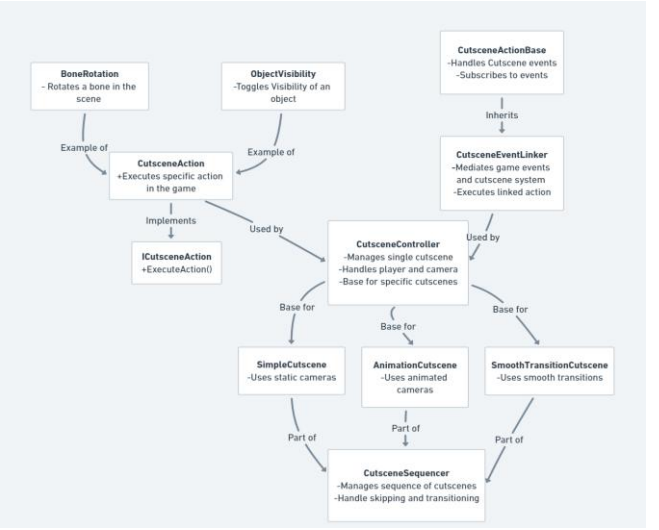
Pada bagian ini akan membahas mengenai *game* sejenis yang pernah dibuat, berikut merupakan *game*nya.

2.1.1 *Cutscenes in Video Games: Exploring Narratives, Interactivity, and Agency*

Jurnal "*Waiting for Something to Happen: Narratives, Interactivity, and Agency and the Video Game Cutscene*" mengeksplorasi hubungan antara narasi, interaktivitas, dan agensi dalam *cutscene video game*. Artikel ini menganalisis bagaimana *cutscene* berfungsi sebagai momen transisi yang menghubungkan elemen naratif dengan interaktivitas *gameplay*, serta bagaimana mereka mempengaruhi pengalaman pemain. Penulis menyoroti peran *cutscene* dalam membangun cerita, menyampaikan informasi penting, dan meningkatkan keterlibatan pemain dengan memberikan konteks naratif yang mendalam. Selain itu, jurnal ini juga membahas bagaimana *cutscene* dapat menyeimbangkan antara kontrol pemain dan narasi yang dirancang oleh pengembang, serta dampaknya terhadap agensi pemain dalam keseluruhan pengalaman bermain *game* (Cheng, n.d.).

2.1.2 *Using Narrative and Technology in Creating Interactive Scenes for 3D Educational Games*

Dalam penelitian ini sistem *cutscene* yang dikembangkan oleh peneliti memberikan banyak kemungkinan bagi pengembang untuk menciptakan elemen naratif yang imersif sambil mempertahankan fleksibilitas dan kontrol. Sifat *modular* dan integrasinya dengan peristiwa dinamis dalam *game* mendorong inovasi dan adaptasi yang penting dalam dunia permainan komputer yang terus berkembang.



Gambar 2.1 Konseptual Diagram untuk *Cutscene System*

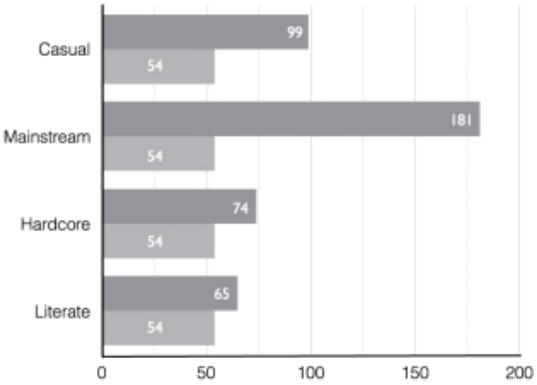
Penyusun *cutscene* tingkat lanjut menawarkan kerangka kerja yang kuat untuk mengintegrasikan elemen naratif yang kaya, meningkatkan efisiensi dan kreativitas dalam pengembangan *game*. Desain *modular* memungkinkan prototipe yang mudah dan mendorong eksperimen dengan berbagai solusi naratif, yang sangat bermanfaat dalam konteks pendidikan dan budaya di mana pengalaman *visual* yang menarik sangat penting.

2.1.3 *Exploring Cutscenes as an Information System in Video Games*

Penelitian ini membahas bagaimana *cutscene* dalam *video game*, yang biasanya digunakan untuk pengembangan naratif sinematik, juga bisa berfungsi sebagai sistem informasi dalam lingkungan permainan. dalam penelitian ini *cutscene system* memiliki berbagai fungsi seperti adegan percakapan, pembuangan informasi, pengaturan adegan dan suasana hati, Penghargaan, pengenalan plot atau elemen *gameplay* dan tempo. *Cutscene* sendiri terbagi menjadi dua kategori yaitu *cutscene* in *game* dan *pre-rendered*. keduanya memiliki keuntungan dan kerugian tergantung pada efisiensi dan biaya produksi (Říha 2014).

2.1.4 *The Relationship Between Game Motivation and Preferences for Cutscenes*

participants, 54 participants per group.



Gambar 2.2 *Classification data of game attitude of participants*

Penelitian ini berfokus pada desain *cutscene* yang berpusat pada pemain dan mengkaji faktor-faktor yang mempengaruhi preferensi pemain terhadap *cutscene*. Melalui analisis kuesioner berbasis internet dari ratusan pemain, ditemukan bahwa sikap pemain terhadap 23 deskripsi *cutscene* dipengaruhi oleh kategori pemain seperti kasual, *mainstream*, *hardcore*, dan *literate*, dengan 14 deskripsi dipengaruhi oleh sikap pemain. Sebaliknya, motivasi pemain seperti pencapai, sosialis, penjelajah, dan pembunuh hanya mempengaruhi 5 deskripsi *cutscene*. Tidak ada korelasi atau interaksi antara sikap dan motivasi pemain terhadap preferensi *cutscene*. Motivasi pemain memiliki dampak yang lebih kecil dibandingkan sikap, dan kurangnya teori umum yang mendukung klasifikasi motivasi menunjukkan bahwa temuan ini mungkin tidak sepenuhnya menggambarkan dampaknya. Oleh karena itu, diperlukan eksplorasi lebih lanjut terhadap motivasi pemain *game* aksi-petualangan. Temuan ini juga menunjukkan bahwa desain *cutscene* yang meminimalkan gangguan terhadap imersi permainan adalah penting, terutama bagi pemain *hardcore* dan *literate*. Rekomendasi untuk desain *cutscene* meliputi penyampaian informasi secara singkat di awal

permainan, penggunaan *cutscene* pendek untuk tindakan sederhana, dan penempatan *cutscene* yang tepat setelah operasi permainan yang *intens* atau saat pemain diharapkan melanjutkan permainan, untuk menjaga imersi dan mencegah kebingungan pemain (Ruan & Cho 2014).

2.1.5 Implementing the Beats & Units Framework in Narrative-Driven Game Development

<table>
<tr><td>Operational level</td><td>Surface</td><td>Layer 7 – Beats</td></tr>
<tr><td rowspan="5">Informational level</td><td>Units</td><td>Layer 6 – Units</td></tr>
<tr><td rowspan="2">Design</td><td>Layer 5 – Game plot</td></tr>
<tr><td>Layer 4 – Gameplay</td></tr>
<tr><td rowspan="2">Conceptual</td><td>Layer 3 – Story (Fable)</td></tr>
<tr><td>Layer 2 – Mechanics</td></tr>
</table>

Gambar 2.3 *Beats & Units Frameworks*

Penelitian ini berfokus pada implementasi *frameworks Beats & Units* untuk desain narasi dalam game interaktif. Pada *framework* ini membagi cerita kedalam dua komponen yaitu *beats* sebagai momen penting dalam cerita dan units sebagai detail setiap aksi dari pemain. Penelitian dilakukan melalui pengembangan prototipe *game* berbasis narasi, di mana setiap *beat* dirancang untuk merepresentasikan alur cerita utama, sementara *unit* mendukung interaksi pemain. Hasil menunjukan bahwa *framework* ini dapat meningkatkan pengalaman naratif pemain dengan menciptakan keselarasan antara cerita dan *gameplay.* Temuan ini diharapkan menjadi panduan bagi pengembang dalam menciptakan narasi game yang lebih imersif dan terstruktur.

Empat penelitian yang sudah disebutkan di atas akan diadaptasi ke kebutuhan *fungsional* yang dapat mengakomodasi kebutuhan *game designer* untuk mengembangkan *game* yang imersif (Zagalo et al. 2023).

2.2 Studi Literatur

Pada bagian ini akan membahas mengenai penelitian-penelitian yang pernah dilakukan, berikut merupakan penelitiannya.

2.2.1 Design Pattern

Design Pattern adalah sebuah solusi untuk masalah yang umum terjadi dalam desain perangkat lunak. Ini seperti cetak biru siap pakai yang dapat disesuaikan untuk memecahkan masalah desain berulang dalam kode. *Pattern* bukanlah potongan kode tertentu, melainkan konsep umum untuk memecahkan masalah tertentu. Sebaliknya, *pattern* menyediakan detail dan pendekatan yang dapat diterapkan sesuai dengan kebutuhan dan realitas program.

Pattern seringkali dikacaukan dengan algoritma, karena kedua konsep tersebut menggambarkan solusi tipikal untuk beberapa masalah yang diketahui. Meskipun algoritma selalu mendefinisikan serangkaian tindakan yang jelas yang dapat mencapai suatu tujuan, pattern adalah deskripsi solusi tingkat tinggi. Kode pattern yang sama yang diterapkan pada dua program berbeda mungkin berbeda (Refactoring Guru, 2022).

a. Observer Pattern

Observer Pattern adalah salah satu pola desain perangkat lunak yang digunakan dalam pemrograman berorientasi objek. Implementasi *observer pattern* pada *engine unity* dan *C#* dapat menggunakan fitur yang sudah disiapkan oleh *unity Event* . *Event* di dalam *unity* adalah *Unity Event* yang bisa ditampilkan di Inspector.

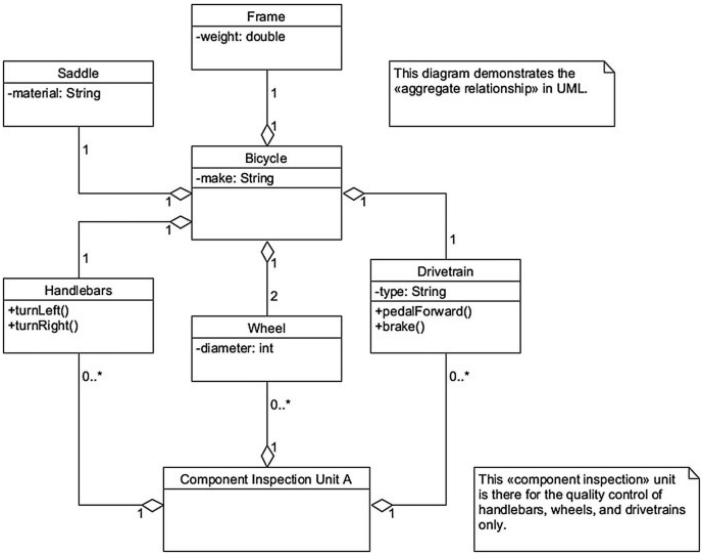
Di *Unity*, dimungkinkan untuk membuat koneksi modular antara satu *script* dengan yang lain dan objek dengan menggunakan *Event System* dan *Delegate*, yang memungkinkan memicu logika permainan saat itu terjadi, tanpa bergantung pada koneksi yang ketat antar skrip

2.2.2 UML

UML (Unified Modeling Language) adalah bahasa *visual* yang digunakan dalam rekayasa perangkat lunak untuk merancang, mendokumentasikan, dan memvisualisasikan sistem. *UML* membantu dalam mempermudah pengembangan aplikasi berkelanjutan karena sistem yang terdokumentasi dengan baik mengurangi hambatan dalam pengembangan, dimana *developer* tidak perlu menelusuri dan mempelajari setiap baris kode untuk memahami alur dalam sistem tersebut. *UML* juga berfungsi sebagai alat bantu antar *developer* untuk memahami sistem yang sedang dikembangkan.

a. Class Diagram

Diagram kelas *(class diagram)* adalah salah satu jenis diagram yang digunakan dalam *Unified Modeling Language (UML)* untuk menggambarkan struktur kelas dan hubungan antara kelas dalam sebuah sistem perangkat lunak. *Class Diagram* adalah alat yang sangat penting dalam pemodelan perangkat lunak berbasis objek dan digunakan untuk merancang struktur kelas, atribut, metode, serta hubungan antara kelas-kelas tersebut.



Gambar 2.4 *Class Diagram*

2.2.3 Throwaway Prototyping

Throwaway Prototyping adalah suatu teknik dalam pengembangan perangkat lunak yang melibatkan pembuatan *prototype* sistem sementara dengan tujuan untuk menggali pemahaman lebih baik tentang kebutuhan dari sistem dalam sebuah *game*. *Prototype* dibangun secara cepat dan tidak bertujuan untuk menjadi produk akhir, tetapi digunakan sebagai alat untuk mengklarifikasi kebutuhan dalam sebuah *game*.

2.2.4 SOLID Principle

SOLID Principle adalah sebuah prinsip yang memiliki tujuan untuk membuat *kode* yang dibuat dapat dipahami dan dikembangkan seiring pertumbuhan proyek (Robert C. Martin, 2000).
kata *SOLID* merupakan singkatan dari:

- *Single Responsibility Principle*
- *Open Closed Principle*
- *Liskov Substitution Principle*
- *Interface Segregation*
- *Dependency Inversion Principle*

2.2.5 Unity

Unity dalam sebuah *Game Engine* yang populer di dunia saat ini. *Unity Game Engine* memungkinkan pengembangan *game* dalam beberapa platform secara bersamaan.

Secara umum, *Unity* dirancang khusus untuk proses pembuatan *game*. Jika ingin membuat desain atau pemodelan aset *3D* dapat menggunakan *software* pihak ketiga seperti *3ds Max, Blender*, dan lainnya. *Unity* menawarkan banyak fitur, termasuk *Particle FX*, *Audio Reverb Zone*, *2D Sprites Maker*, *Skybox*, *Ambient Lighting*, *Shaders*, dan lain-lain. *Unity* juga memiliki text editor bernama *Monodevelop* yang dapat digunakan untuk *coding game* dan terintegrasi langsung dengan *Unity Engine,* serta *Asset Store* untuk mencari aset *2D/3D* baik yang gratis maupun berbayar. adapun beberapa fitur di dalam *engine unity* yang digunakan dalam proyek ini.

a. Event System

Event System adalah mekanisme untuk mengirimkan *event* ke objek dalam aplikasi berdasarkan berbagai input seperti keyboard, mouse, sentuhan, atau *input* khusus lainnya. *Event System* terdiri dari beberapa komponen yang bekerja sama untuk mengirimkan *event*. Ketika menambahkan komponen *Event System* ke *GameObject*, akan terlihat bahwa hanya sedikit fungsionalitas yang terekspos. Hal ini dikarenakan *Event System* dirancang sebagai pengelola dan fasilitator komunikasi antar modul *Event System (Unity - Manual: EventSystem no date)*

b. ScriptableObject

ScriptableObject adalah sebuah wadah data yang dapat digunakan untuk menyimpan sejumlah besar data, terpisah dari *instance* kelas. Salah satu kegunaan utama *ScriptableObject* adalah untuk mengurangi penggunaan memori Proyek Anda dengan menghindari duplikasi nilai. Hal ini berguna jika Proyek Anda memiliki *Prefab* yang menyimpan data yang tidak berubah dalam skrip *MonoBehaviour* yang terlampir *(Unity - Manual: ScriptableObject no date).*

Setiap kali menginstansiasi Prefab tersebut, ia akan mendapatkan salinan data tersendiri. Daripada menggunakan metode ini dan menyimpan data yang terduplikasi, ScriptableObject dapat digunakan untuk menyimpan data tersebut dan kemudian mengaksesnya dengan referensi dari semua Prefab. Ini berarti hanya ada satu salinan data di dalam memori.

c. Custom Editor

Custom editor adalah skrip terpisah yang menggantikan tampilan default editor dengan kontrol editor yang dipilih. Dengan *custom editor*, properti skrip dapat ditampilkan dan dioperasikan secara lebih terstruktur di *Inspector* Unity. Ini sangat berguna untuk menyediakan antarmuka yang lebih intuitif atau fungsional untuk pengaturan tertentu yang akan digunakan oleh pengembang lain atau di masa depan.

d. Unity ReadOnlyAttribute

Atribut *ReadOnly* memungkinkan penandaan anggota sebuah *struct* yang digunakan dalam sebuah *job* sebagai hanya-baca. Kontainer *native* secara default adalah baca-tulis saat digunakan dalam sebuah *job*. Ini berarti dua *job* yang merujuk ke kontainer yang sama tidak dapat dijadwalkan secara bersamaan. Dengan menambahkan atribut *ReadOnly* pada *field* kontainer di *struct job*, kontainer tersebut ditandai sebagai hanya-baca, memungkinkan dua *job* berjalan secara paralel untuk membaca data dari kontainer yang sama (Unity - Scripting API: ReadOnlyAttribute no date).

2.2.6 Cutscene System

Cutscene memainkan peran penting dalam penyampaian naratif *video game* dengan memberikan elemen cerita penting dan membimbing pemahaman pemain tentang dunia *game.* Namun, mereka juga memperkenalkan tantangan dengan mengganggu aliran interaktif yang imersif dan memaksa pemain untuk bergeser antara peran konfigurasional

dan *interpretatif. Cutscene* berfungsi sebagai alat naratif yang membantu membimbing pemahaman pemain tentang cerita. Mereka dapat mengungkapkan titik plot kritis, memperkenalkan karakter, atau menyiapkan peristiwa masa depan yang perlu diketahui pemain untuk memahami keseluruhan cerita *game* (Říha 2014).

2.3 Game Terkait

2.3.1 Chef RPG

Dalam *Game Chef RPG,* saya menggunakan fungsi yang terdapat di tampilan *inspector* dari sistem *cutscene* yang sudah ada. Saya kemudian mengimplementasikan fungsi-fungsi tersebut ke dalam kode yang saya buat sendiri. Pendekatan ini memungkinkan saya untuk menyesuaikan dan memperluas fungsionalitas *cutscene* agar lebih sesuai dengan kebutuhan spesifik proyek saya *(Pixel Architect 2023).*

Gambar 2.5 Game Chef RPG

2.3.2 Stardew Valley

Stardew Valley adalah *game* simulasi pertanian yang populer dikembangkan oleh *ConcernedApe*. Dalam *game* ini, pemain mengambil peran karakter yang mewarisi sebuah pertanian yang terbengkalai dan ditugaskan untuk memulihkannya. Pemain dapat terlibat dalam berbagai aktivitas seperti bertani, beternak, memancing, menambang, mencari makanan, dan berinteraksi dengan komunitas lokal. *Game* ini dikenal karena gaya seni pikselnya yang menawan, *gameplay* yang terbuka, dan kedalaman elemen sosial serta naratifnya .

Gambar 2.6 Game Stardew Valley

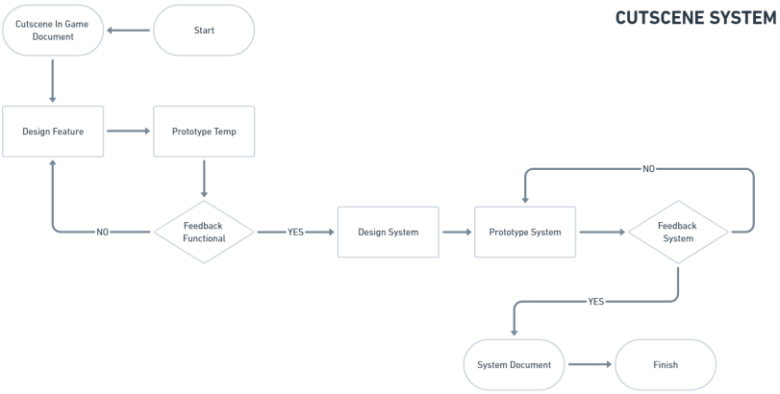
Menggunakan Sistem *Cutscene* Interaktif di *Stardew Valley*. Di *Stardew Valley*, sistem *cutscene* interaktif memainkan peran penting dalam meningkatkan narasi dan pengalaman pemain. *Cutscene* digunakan untuk mengembangkan alur cerita, membangun hubungan dengan karakter *non-pemain (NPC)*, dan memberikan konteks penting bagi tindakan pemain. *Cutscene* ini bersifat interaktif, memungkinkan pemain membuat pilihan yang dapat mempengaruhi hasil peristiwa dan hubungan mereka dengan *NPC* .

Terinspirasi oleh sistem *cutscene* interaktif di *Stardew Valley*, saya memutuskan untuk menggabungkan fitur serupa ke dalam proyek saya. Saya menganalisis bagaimana *Stardew Valley* dengan mulus mengintegrasikan *cutscene* ke dalam *gameplay* untuk menciptakan pengalaman yang lebih mendalam dan menarik. Saya kemudian mengimplementasikan prinsip-prinsip ini ke dalam kode saya sendiri, menyesuaikan dan memperluas fungsionalitasnya agar sesuai dengan kebutuhan spesifik *game* saya. Dengan memanfaatkan keunggulan sistem *cutscene Stardew Valley*, saya bertujuan untuk menciptakan pengalaman naratif yang dinamis dan interaktif yang meningkatkan keterlibatan dan imersi pemain dalam *game* saya.

BAB III
METODOLOGI PENELITIAN

3.1 Analisa Kebutuhan

3.1.1 *Workflow* Pengembangan Sistem *Cutscene*



Gambar 3.1 *Flowchart Workflow*

1. *Cutscene In Game Document*

Fase ini adalah tahapan awal dengan mendefinisikan kebutuhan naratif atau elemen *cutscene* yang akan di implementasikan ke dalam game. Informasi ini dicatat oleh *game designer* dalam bentuk *document cutscene in game* untuk memastikan semuanya terdokumentasi dengan baik dan benar.

2. *Design Feature*

Berdasarkan dokumen awal, langkah selanjutnya adalah mendesain fitur *cutscene*. Proses ini melibatkan perencanaan elemen-elemen penting, seperti interaksi pemain, alur cerita, atau mekanika yang mendukung keberadaan *cutscene* dalam *game*.

3. *Prototype Temp*

Setelah desain selesai, prototipe sementara dibuat untuk menguji kelayakan teknis dari ide yang telah dirancang. Prototipe ini bersifat sederhana dan tidak harus sempurna, namun cukup untuk menunjukkan konsep dasar yang diinginkan

4. *Feedback Functional*

Prototipe sementara diuji untuk memastikan fungsi-fungsi dasarnya berjalan dengan baik. Jika hasil pengujian menunjukkan bahwa prototipe belum memenuhi harapan (*NO*), maka perlu kembali ke tahap desain fitur untuk revisi. Jika prototipe berfungsi sesuai ekspektasi (*YES*), proses berlanjut ke tahap berikutnya.

5. *Design System*

Setelah prototipe fungsional berhasil dibuat, desain sistem dilakukan secara lebih mendalam. Tahap ini mencakup perancangan elemen teknis, logika permainan, serta integrasi dengan sistem lain dalam game untuk memastikan implementasi yang menyeluruh.

6. *Prototype System*

Berdasarkan hasil desain sistem, prototipe sistem penuh dikembangkan. Prototipe ini merupakan versi yang lebih matang dan mendekati bentuk implementasi akhir.

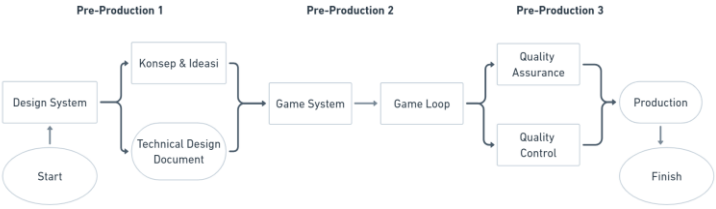
7. *Feedback System*

Prototipe sistem diuji kembali untuk mendapatkan umpan balik yang lebih mendetail. Jika ditemukan kekurangan atau masalah *(NO)*, perbaikan dilakukan dengan kembali ke tahap desain sistem. Jika hasil pengujian memuaskan *(YES)*, langkah berikutnya adalah dokumentasi.

8. *System Document*

Sistem yang telah selesai dan terverifikasi didokumentasikan secara resmi. Dokumen ini mencakup semua detail implementasi dan dapat digunakan sebagai referensi untuk pengembangan lebih lanjut atau sebagai bagian dari laporan penelitian atau skripsi.

3.1.2 *Game Design Framework*



Gambar 3.2 *Game Development Framework*

Untuk mengembangkan *Game Project Botanical* dengan *genre Adventure Story Driven*, *framework* yang menyatukan setiap elemen desain secara holistik dan mendalam sangat penting. *Framework Beats & Units* adalah pendekatan yang bagus untuk memastikan semua elemen desain bekerja bersama dengan baik.

Tabel 3.1 *Beats & Units Framework*

Operational Level	*Surface*	*Level 7 - Beats*
Informational level	*Units*	*Level 6 - Units*
	Design	*Layer 5 - Game Plot*
		Layer 4 - Gameplay
	Conceptual	*Layer 3 - Story* (*Fable*)
		Layer 2 - Mechanics
		Layer 1 - Intent (Nexus)

Berikut adalah bagaimana *game development framework* (*Pre-Production* 1, 2, dan 3) dapat disesuaikan dengan *Framework Beats* & *Units*:

1. ***Design* (*Pre-Production 1 : Design System)***

Pada tahap ini, penulis akan merancang keseluruhan visi dan struktur narasi serta *gameplay* dari *game*. Berikut adalah bagaimana tiga tahap *pre-production* dapat dimasukkan ke dalam sub bagian Desain dari *framework Beats & Units.*

- **Konsep dan Ideasi:**

Definisikan visi dan tujuan dari *cutscene* dan narasi *game*. Misalnya, bagaimana *cutscene* akan digunakan untuk memperkuat narasi dan mengembangkan karakter dalam *game*

- **Technical Design Document (TDD):**

Buat dokumen desain teknis yang lebih rinci tentang bagaimana setiap elemen akan diimplementasikan dalam *game*.

2. ***Units (Pre-Production 2 : Game System)***

Pada tahap ini, fokus utamanya adalah merancang sistem permainan yang mencakup pengembangan cerita, karakter, dunia permainan, dan mekanisme permainan. Ini termasuk menentukan gaya artistik, tema, dan bagaimana setiap elemen akan berinteraksi satu sama lain untuk menciptakan pengalaman yang koheren dan menarik bagi pemain

3. ***Surface (Pre-Production 3 : Game Loop)***

Pada tahap ini, elemen-elemen yang telah dirancang dan sistem yang telah dibuat diintegrasikan ke dalam *game loop*. *Game loop* adalah siklus utama yang dijalankan oleh *game* selama beroperasi, mencakup pembaruan permainan, render grafis, dan interaksi pemain.

3.1.3 Analisa Kebutuhan Fungsional

Pada tahap ini, dilakukan analisis kebutuhan fungsional untuk memastikan setiap elemen sistem mendukung narasi dan pengalaman pemain secara optimal. Proses ini mencakup evaluasi berbagai fitur yang diperlukan dalam sistem *cutscene*, seperti pengaturan kamera, dialog, dan interaksi karakter. Selain itu, elemen pendukung seperti transisi antar segmen dan sinkronisasi dengan aksi pemain juga diperhatikan. Dengan analisis yang mendalam, sistem *cutscene* dapat berfungsi sesuai dengan tujuan naratif dan meningkatkan pengalaman bermain secara keseluruhan

Tabel 3.2 Kebutuhan Fungsional

NO	Sistem	Deskripsi
1	*Sistem Cutscene*	● Menampilkan narasi cerita melalui dialog, Emoji, animasi, dan perubahan kamera ● Mengatur urutan *cutscene* dan transisi antar adegan secara mulus. ● Mendukung *custom script* yang kompleks untuk memastikan narasi yang koheren.
2	Sistem *Dialogue*	● Menampilkan dialog teks dengan opsi interaktif. ● Sinkronisasi dialog dengan animasi karakter dan audio.

		● Kemampuan untuk melewati atau mengulang dialog.
3	Sistem Animasi Karakter	● Mengelola berbagai gerakan dan ekspresi karakter sesuai dengan narasi. ● Mengintegrasikan animasi dengan sistem dialog dan *cutscene*. ● Mengatur *animator* yang digerakan oleh player dan yang digerakan oleh *AI(Artificial Intelegent)* secara bergantian
4	Sistem Kamera	● Mengatur sudut pandang dan gerakan kamera untuk memperkuat emosi dan narasi. ● Mendukung transisi kamera yang halus antara *gameplay* dan *cutscene*.
5	Sistem *Frame Interaction*	● Menampilkan bingkati *frame* dengan gambar dan *text* ● Sinronisasi *Text* dengan Gambar
6	Sistem Emoji	● Mengintegrasikan Emoji dengan *Character/NPC* ● Mengelola Berbagai Emoji sesuai dengan narasi
7	Sistem Pergerakan Karakter	● Menggerakan *Object/Character* sesuai dengan Posisi yang diinginkan ● Mengintegrasikan Sistem Pergerakan *Player* dengan *Cutscene*
8	*Editor Data Segment*	● *Segment Data Editor* harus dapat menfasilitasi pengeditan data *cutscene* secara langsung dari *Unity Inspector*

3.1.4 Analisa Kebutuhan Non Fungsional

Analisa kebutuhan *non-fungsional* berfokus pada aspek-aspek yang mendukung kualitas dan kinerja sistem *cutscene*. Beberapa kebutuhan non-fungsional utama meliputi:

1. **Modularitas**

Sistem *cutscene* harus dirancang dengan komponen yang dapat digunakan kembali dan mudah dimodifikasi, memungkinkan

pengembang untuk menambahkan, menghapus, atau mengubah *cutscene* tanpa harus merombak keseluruhan sistem.

2. **Responsivitas**

Cutscene harus dijalankan dengan lancar tanpa *lag* atau gangguan yang dapat mengganggu pengalaman bermain pemain.

3. **Kompatibilitas**

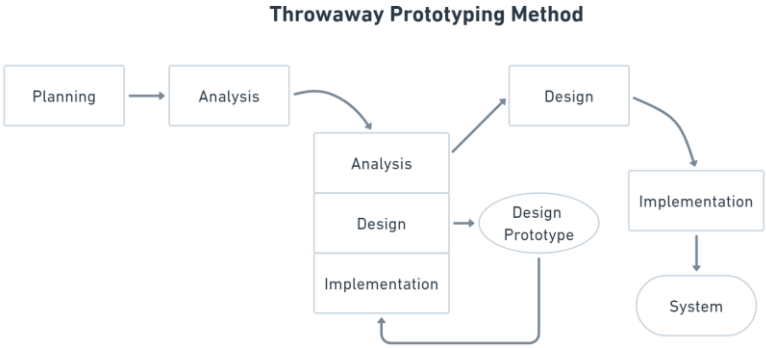
Sistem harus kompatibel dengan berbagai perangkat dan resolusi layar yang berbeda untuk memastikan pengalaman yang konsisten di semua platform.

Dengan mengidentifikasi kebutuhan fungsional dan non-fungsional, serta mengaitkannya dengan *Game Design Framework*, pengembangan sistem *cutscene* di *Game* Project Botanical diharapkan dapat memberikan pengalaman bermain yang imersif dan mendalam bagi pemain, sekaligus meningkatkan apresiasi terhadap cerita yang disajikan dalam *game*.

3.2 Perancangan Arsitektur

Sistem *cutscene* dirancang dengan mengacu pada kebutuhan yang telah diidentifikasi dalam *sub-bab* sebelumnya. Analisis kebutuhan menggarisbawahi pentingnya sistem ini dalam mendukung elemen naratif, sesuai dengan permintaan *game designer* untuk menciptakan pengalaman yang lebih mendalam dan terhubung secara emosional bagi pemain. Dalam proses ini, pendekatan teknis yang dirancang bertujuan untuk memenuhi kebutuhan tersebut dengan efisien dan efektif.

Desain sistem dimulai dengan pembuatan diagram kelas yang berfungsi sebagai representasi struktur sistem secara menyeluruh. Diagram ini mencakup elemen-elemen penting seperti hubungan antar kelas, atribut, dan metode yang akan digunakan untuk mengimplementasikan fitur *cutscene*. Dengan diagram kelas ini, pengembang dapat memastikan bahwa desain sistem tidak hanya memenuhi kebutuhan fungsional tetapi juga mampu diintegrasikan dengan elemen *game* lainnya.

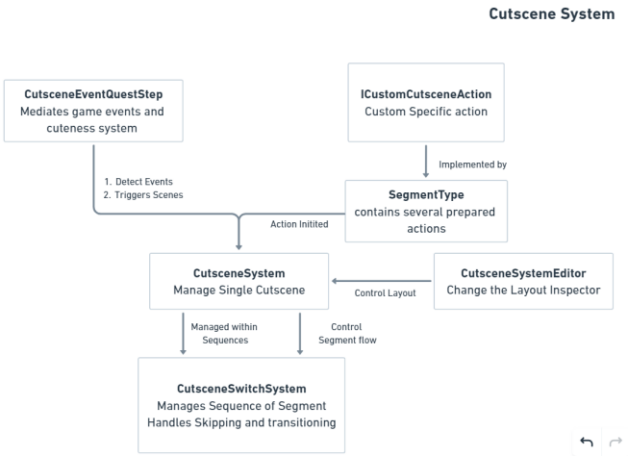


Gambar 3.3 *Throwaway Prototyping Method*

Pada bagian ini, perancangan arsitektur sistem digambarkan melalui *class diagram. Class* diagram ini akan menunjukkan hubungan antar kelas yang ada dalam sistem *cutscene* serta bagaimana mereka berinteraksi satu sama lain. Berikut adalah *class diagram* untuk sistem *cutscene*:

3.2.1 Arsitektur dalam *Cutscene System*

Arsitektur sistem *cutscene* dirancang untuk memenuhi kebutuhan fungsional dan *non-fungsional* dalam memberikan pengalaman naratif kepada pemain. Sistem ini dikembangkan menggunakan *Unity 3D* untuk memberikan kemudahan dalam pengelolaan *cutscene*, baik dari segi pembuatan maupun pengeditan. Sistem *cutscene* dirancang dengan fleksibilitas untuk menangani berbagai aksi spesifik, transisi antar segmen, dan penyesuaian alur *cutscene*.



Gambar 3.4 Konseptual Diagram *Cutscene System*

Gambar 3.4 adalah diagram konseptual dari sistem *cutscene*, yang menggambarkan hubungan antar kelas yang digunakan dalam pengembangan. Secara keseluruhan, diagram ini mencakup beberapa kelas inti, seperti *CutsceneSystem*, *CutsceneSwitchSystem*, dan *ICustomCutsceneAction*. Kelas utama dari sistem ini adalah *CutsceneSystem*, yang bertugas mengelola alur *cutscene* tunggal, dan *CutsceneSwitchSystem*, yang menangani transisi antar segmen *cutscene*.

Sistem *cutscene* dalam *game Project Botanical* dirancang dengan pendekatan modular untuk memudahkan pengelolaan dan pengembangan fitur naratif. Sistem ini terdiri dari beberapa kelas yang memiliki peran masing-masing, mulai dari pengelolaan aksi dalam cutscene hingga transisi antar segmen. Setiap kelas dirancang agar dapat berinteraksi dengan baik dan mendukung kebutuhan game designer dalam menciptakan pengalaman yang imersif bagi pemain.

Tabel 3.3 menjelaskan tugas dan fungsi dari setiap kelas yang terlibat dalam sistem cutscene, termasuk pengelolaan event, aksi, dan transisi cutscene.

Tabel 3.3 *Function in Cutscene System*

NO	*Class*	*Behaviour*

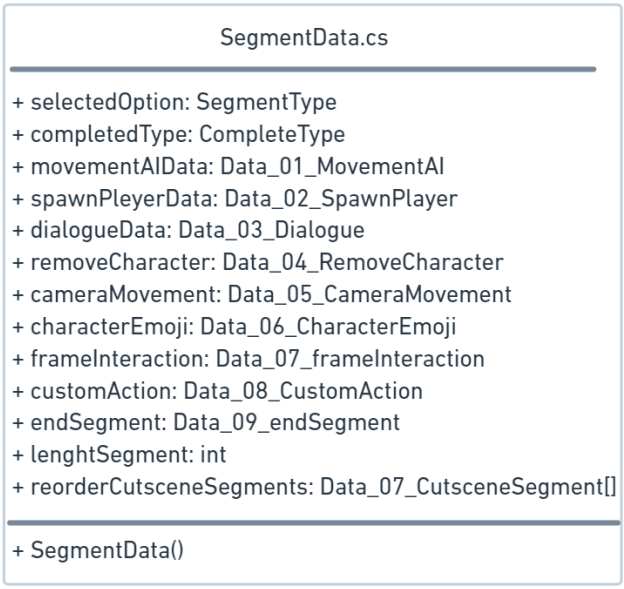
1	*CutsceneEventQuestStep.cs*	Mediator jika ada event yang terjadi pada in *game*
2	*CutsceneSystem.cs*	Kontrol satu *Cutscene*
3	*SegmentType.cs*	Menyimpan *list action* dengan *behaviour* berbeda-beda
4	*ICustomCutsceneAction.cs*	Mengatur penambahan *action*
5	*CutsceneSystemEditor.cs*	Mengatur tata letak *inspector*
6	*CutsceneSwitchSystem.cs*	Mengatur urutan *segment* dan transisi

3.2.2 Desain Segment Data

Untuk mendukung *Cutscene Manager* dalam mengelola setiap segmen *cutscene*, *Segment Data* dirancang untuk menyimpan informasi spesifik tentang setiap segmen yang akan dijalankan. Data ini mendefinisikan tipe dan aksi yang harus dilakukan dalam tiap bagian *cutscene*, memberikan fleksibilitas bagi *game designer* untuk mengatur alur cerita dengan mudah.

3.2.2.1 Struktur Segment Data

Struktur Segment Data dirancang dengan pendekatan modular, seperti yang digambarkan dalam diagram berikut:



Gambar 3.5 *Class Diagram Segment Data*

3.2.2.2 Penjelasan Komponen Segment Data

SegmentData.cs berfungsi sebagai wadah utama untuk menyimpan informasi terkait setiap segmen dalam *cutscene*. Setiap *instance* dari *SegmentData* mencakup berbagai elemen penting, seperti *selectedOption*, yang menentukan tipe segmen yang akan dijalankan, seperti dialog, pergerakan kamera, atau aksi khusus. Selain itu, terdapat *completedType*, yang mencatat status penyelesaian segmen, serta referensi ke berbagai data spesifik yang mendukung pelaksanaan segmen tersebut.

Komponen data spesifik yang direferensikan dalam *SegmentData* meliputi:

1. *movementAIData*: Komponen ini menyimpan data terkait pergerakan *AI* untuk segmen tertentu. Ini termasuk karakter yang terlibat dalam segmen, posisi target untuk karakter tersebut, serta kecepatan yang digunakan untuk pergerakan dalam segmen tersebut. Data ini memastikan bahwa *AI* bergerak sesuai dengan instruksi yang ditentukan dalam segmen spesifik.

2. *spawnPlayerData*: Komponen ini menyimpan data mengenai posisi spawn pemain dan karakter yang terlibat dalam segmen. Ini berlaku untuk segmen-segmen yang membutuhkan *spawn karakter* pemain, sehingga data ini akan disesuaikan pada segmen yang membutuhkan spawn karakter.
3. *dialogueData*: Komponen ini berfungsi untuk mengelola dialog berbasis file *JSON* pada segmen tertentu. Data ini menyimpan teks narasi yang relevan untuk segmen yang sedang diproses. Setiap segmen dengan dialog akan memiliki data dialognya sendiri, yang akan ditampilkan sesuai dengan alur cerita yang dirancang.
4. *removeCharacterData*: Komponen ini menyimpan informasi mengenai karakter yang perlu dihapus dalam segmen tertentu. Pada segmen yang membutuhkan penghapusan karakter, data ini akan menentukan karakter mana yang harus dihapus dari *scene*.
5. *cameraMovementData*: Komponen ini menyimpan data pergerakan kamera khusus untuk segmen yang melibatkan pergerakan kamera. Ini mencakup informasi mengenai *waypoint* dan target kamera untuk segmen tersebut, memastikan kamera bergerak sesuai dengan tujuan narasi yang ingin dicapai dalam segmen tersebut.
6. *characterEmojiData*: Komponen ini menyimpan informasi mengenai ekspresi karakter, seperti emoji yang digunakan dalam segmen tertentu untuk menggambarkan perasaan atau reaksi karakter. Setiap segmen yang memerlukan ekspresi karakter akan memiliki data emoji khusus yang terkait.
7. *frameInteractionData*: Komponen ini menyimpan data yang memungkinkan interaksi berbasis *frame* pada segmen tertentu. Interaksi berbasis *frame* ini akan terjadi selama *cutscene*, dan data ini akan menentukan bagaimana karakter atau elemen lain berinteraksi dengan lingkungan berdasarkan waktu atau *frame*.
8. *customActionData*: Komponen ini menyimpan data untuk pelaksanaan aksi khusus yang telah didefinisikan oleh *game designer* dalam segmen tertentu. Aksi ini bisa berupa sesuatu yang sangat spesifik untuk segmen tersebut dan tidak berlaku secara umum di semua segmen, memberikan fleksibilitas bagi *designer* untuk menyesuaikan aksi sesuai dengan kebutuhan narasi.
9. *endSegment*: Komponen ini menyimpan data yang menentukan apakah segmen tersebut adalah segmen terakhir dalam *cutscene*.

Pada segmen terakhir, data ini akan diset untuk menandai akhir dari alur *cutscene* tersebut. Segmen lainnya tidak akan membutuhkan data ini.

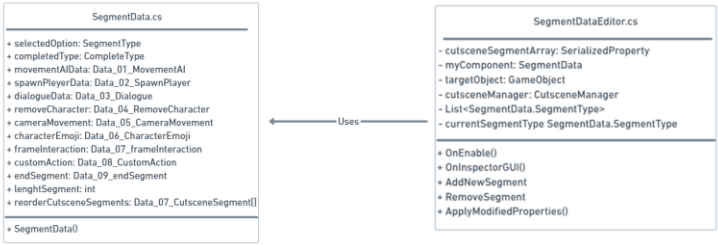
Setiap komponen data tersebut diimplementasikan melalui skrip data spesifik untuk mendukung fungsionalitasnya. *Data_01_MovementAI.cs* digunakan untuk mengatur pergerakan karakter *AI* dengan parameter seperti karakter, target posisi, dan kecepatan. *Data_02_SpawnPlayer.cs* bertugas mengatur posisi *spawn* pemain dan karakter yang terlibat. *Data_03_Dialogue.cs* menangani dialog berbasis file *JSON* untuk memberikan narasi yang dinamis. *Data_04_RemoveCharacter.cs* menghapus karakter yang tidak diperlukan dari cutscene.

Selanjutnya, *Data_05_CameraMovement.cs* bertanggung jawab mengontrol pergerakan kamera, termasuk pengaturan *waypoint* dan target kamera untuk menciptakan pengalaman sinematik yang mendalam. *Data_06_CharacterEmoji.cs* memungkinkan tampilan ekspresi karakter melalui emoji. *Data_07_FrameInteraction.cs* mengelola interaksi berbasis *frame* untuk meningkatkan detail dalam *cutscene*. *Data_08_CustomAction.cs* menjalankan aksi khusus yang telah didefinisikan sebelumnya oleh *game designer*, dan *Data_09_EndSegment.cs* menentukan apakah segmen tersebut merupakan akhir dari *cutscene*.

Melalui desain ini, *SegmentData* memastikan setiap segmen dalam *cutscene* berjalan sesuai dengan narasi dan desain yang telah direncanakan, sekaligus memberikan fleksibilitas bagi game designer untuk menyesuaikan detail *cutscene*.

3.2.3 Desain *Segment Data Editor*

Untuk memberikan fleksibilitas lebih dalam pengelolaan *Segment Data*, *game designer* membutuhkan *Segment Data Editor*. *Editor* ini memungkinkan game designer untuk mengonfigurasi dan mengedit data setiap segmen langsung di editor *Unity*, menggunakan atribut *CustomEditor* untuk menyesuaikan tampilan dan interaksi. Dengan adanya editor ini, *game designer* dapat dengan mudah menyesuaikan setiap elemen *cutscene* agar sesuai dengan keinginan dan alur cerita yang telah direncanakan.



Gambar 3.6 *Class Diagram SegmentDataEditor*

SegmentDataEditor.cs adalah skrip editor kustom yang dirancang untuk mempermudah pengelolaan data segmen dalam *cutscene* di *Unity Editor*. Skrip ini memungkinkan pengeditan properti segmen secara visual, yang mempercepat dan menyederhanakan proses penyesuaian segmen dalam alur cerita tanpa memerlukan penulisan kode manual. Beberapa komponen utama dalam *SegmentDataEditor.cs* mencakup *cutseneSegmentArray*, yaitu *array* yang menyimpan berbagai segmen *cutscene*, serta *myComponent*, yang merujuk ke *SegmentData* yang berisi informasi dan konfigurasi terkait segmen tersebut. Selain itu, terdapat targetObject, yang merujuk ke *GameObject* yang terkait dengan elemen visual segmen, dan *cutsceneManager*, yang mengelola jalannya cutscene. *List<SegmentData.SegmentType>* menyimpan daftar tipe segmen yang tersedia, sementara *currentSegmentType* menunjukkan tipe segmen yang sedang aktif untuk diedit. Fungsi-fungsi utama dalam skrip ini meliputi *OnEnable(),* yang memuat data segmen saat editor diaktifkan, serta *OnInspectorGUI()* yang menampilkan antarmuka pengguna untuk pengeditan properti segmen. Fungsi *AddNewSegment()* memungkinkan penambahan segmen baru ke dalam *cutscene*, sementara *RemoveSegment*() digunakan untuk menghapus segmen yang tidak diperlukan. Terakhir, *ApplyModifiedProperties()* memastikan bahwa perubahan yang dilakukan pada properti segmen diterapkan dengan benar. Dengan skrip ini, pengelolaan cutscene menjadi lebih efisien dan memberikan kontrol visual yang lebih baik dalam mengubah dan menyesuaikan data segmen sesuai kebutuhan narasi.

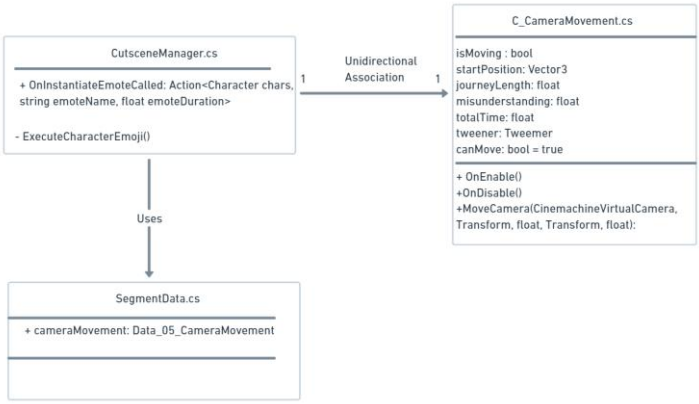
3.2.4 Desain *Cutscene Manager*

Sebagai komponen utama dalam pengelolaan *cutscene*, *game designer* membutuhkan *Cutscene Manager* untuk memastikan bahwa seluruh alur cerita dalam cutscene berjalan sesuai dengan urutan dan logika yang telah ditentukan. *Cutscene Manager* berfungsi sebagai pengontrol utama, mengelola eksekusi segmen-segmen *cutscene* yang ada, dan memastikan setiap aksi atau *event* dalam *cutscene* terjadi sesuai dengan desain yang diinginkan. Hal ini memberi *game designer* kontrol penuh dalam merancang pengalaman naratif yang imersif.

Setelah *Cutscene Manager* mengatur alur dan urutan kejadian dalam *cutscene*, langkah selanjutnya adalah memastikan setiap elemen dalam *cutscene* berfungsi secara dinamis dan terkoordinasi untuk menciptakan pengalaman yang mendalam. Ini mencakup berbagai aspek, mulai dari pergerakan kamera, animasi karakter, hingga pengaturan *event-event* spesifik. Dalam hal ini, setiap elemen saling terhubung dan berinteraksi dengan *Cutscene Manager* untuk menghasilkan sebuah narasi yang tidak hanya mengikuti alur cerita, tetapi juga memberikan dampak visual dan emosional yang mendalam bagi pemain. Salah satu elemen penting dalam mencapai hal ini adalah *Camera Movement*, yang akan dibahas lebih lanjut sebagai bagian dari desain yang lebih luas untuk mendukung kelancaran dan keselarasan seluruh *cutscene.*

a. CameraMovement

Selama *cutscene*, *game designer* membutuhkan sistem pergerakan kamera yang dapat mengikuti alur cerita secara dinamis dan sinematik. Oleh karena itu, *Camera Movement* menggunakan *Cinemachine* dan *DOTween* untuk menciptakan gerakan kamera yang halus dan terkontrol. *Camera Movement* bekerja erat dengan *Cutscene Manager* untuk memastikan bahwa gerakan kamera selalu selaras dengan kejadian dalam *cutscene*, memberikan pengalaman visual yang mendalam bagi pemain.



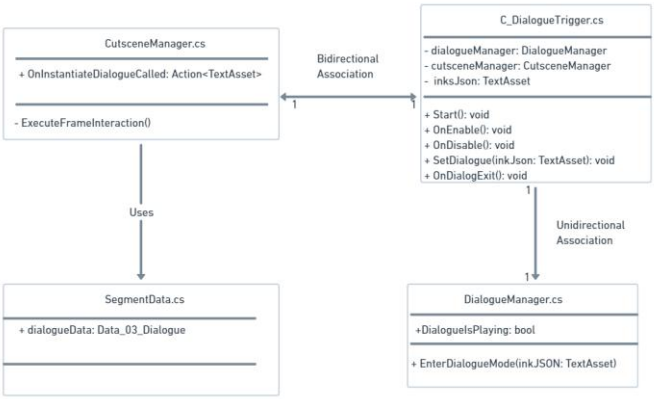
Gambar 3.7 *Class Diagram Camera Movement*

Diagram pada gambar menunjukkan hubungan *uses* dan *unidirectional association* antara tiga kelas: *CutsceneManager*, *C_CameraMovement*, dan *SegmentData*. *CutsceneManager* bertanggung jawab untuk mengelola emote karakter dalam game melalui event *OnInstantiateEmoteCalled*, yang menerima parameter seperti karakter, nama emote, dan durasi emote, serta metode *ExecuteCharacterEmoji()* untuk memicu aksi emoji. Kelas ini menggunakan *SegmentData*, yang menyimpan data terkait gerakan kamera dalam atribut *cameraMovement* dari tipe *Data_05_CameraMovement*. Selain itu, *CutsceneManager* memiliki asosiasi *unidirectional* dengan *C_CameraMovement*, yang menangani logika perpindahan kamera menggunakan atribut seperti *isMoving*, *startPosition*, dan *journeyLength*. Fungsi-fungsi seperti *MoveCamera()* digunakan untuk memindahkan kamera berdasarkan parameter virtual kamera dan transformasi tertentu. Hubungan ini menunjukkan bahwa *CutsceneManager* mengandalkan *SegmentData* untuk data konfigurasi kamera dan memanggil metode dalam *C_CameraMovement* untuk mengatur perpindahan kamera tanpa hubungan balik.

b. DialogueTrigger

Diagram ini menggambarkan hubungan *unidirectional* antara dua kelas, yaitu *DialogueTrigger* dan *DialogueManager*, yang berperan dalam sistem dialog pada *game*. Salah satu elemen penting dalam

cutscene adalah dialog antara karakter. *Dialogue Trigger* memicu dialog berdasarkan data yang diambil dari file *JSON* yang disimpan, dan mengelola transisi antar dialog menggunakan metode seperti *SetDialogue()* dan *OnDialogExit()*. *Dialogue Trigger* bergantung pada *Dialogue Manager* untuk memproses dan mengontrol status dialog. *Game designer* dapat menggunakan *Dialogue Trigger* untuk memulai dan mengakhiri percakapan dalam *cutscene* sesuai dengan alur cerita yang telah dirancang.



Gambar 3.8 *Class Diagram DialogueTrigger*

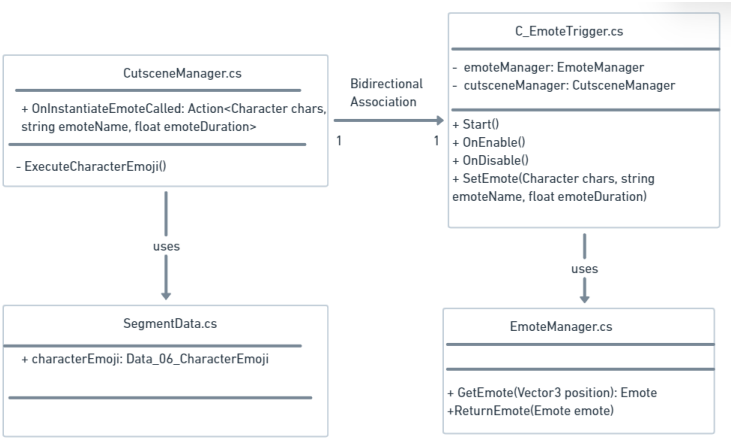
Diagram pada gambar menunjukkan hubungan *uses*, *bidirectional association*, dan *unidirectional association* antara empat kelas: *CutsceneManager*, *C_DialogueTrigger*, *DialogueManager*, dan *SegmentData*. *CutsceneManager* bertanggung jawab untuk mengelola interaksi dialog dalam game melalui event *OnInstantiateDialogueCalled* yang memproses teks dialog, serta metode *ExecuteFrameInteraction()* untuk mengelola adegan. Kelas ini menggunakan *SegmentData*, yang menyimpan data terkait dialog dalam atribut *dialogueData* dari tipe *Data_03_Dialogue*.

Selain itu, *CutsceneManager* memiliki asosiasi *bidirectional* dengan *C_DialogueTrigger*, yang menangani logika pemicu dialog menggunakan fungsi seperti *SetDialogueInkJSON()* dan *OnDialogExit()*. *C_DialogueTrigger* juga memiliki asosiasi *unidirectional* dengan

DialogueManager, yang mengelola status dialog melalui atribut *DialoguesPlaying* dan metode *EnterDialogueMode()*. Hubungan ini menunjukkan bahwa *CutsceneManager* mengandalkan *SegmentData* untuk data konfigurasi dialog, bekerja secara dua arah dengan *C_DialogueTrigger* untuk memproses pemicu dialog, serta bergantung pada *DialogueManager* untuk pengelolaan mode dialog tanpa hubungan balik.

c. EmoteTrigger

Untuk meningkatkan ekspresi visual karakter, *game designer* membutuhkan *Emote Trigger* yang memicu ekspresi atau *emote* berdasarkan nama dan durasi yang telah ditentukan. *Emote Trigger* bergantung pada *Emote Manager* untuk mendapatkan dan mengelola objek *emote* yang akan ditampilkan di layar. Dengan demikian, *game designer* dapat memanfaatkan *emote* untuk menambahkan dimensi emosional pada karakter selama *cutscene*.



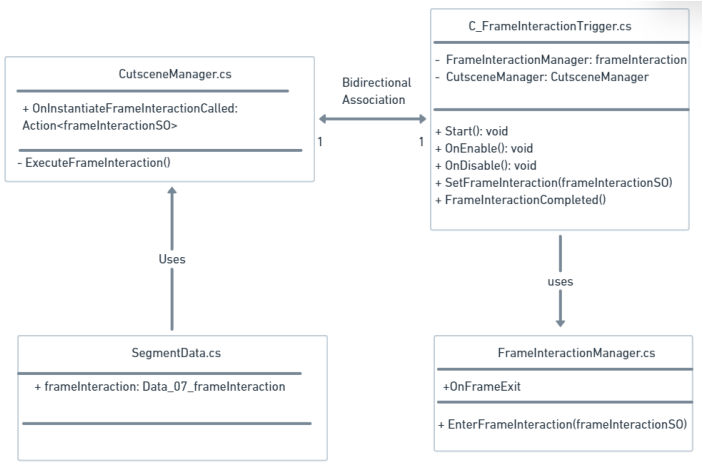
Gambar 3.9 *Class Diagram EmoteTrigger*

Diagram menunjukkan hubungan antara empat kelas *CutsceneManager*, *C_EmoteTrigger*, *EmoteManager*, dan *SegmentData*. *CutsceneManager* menggunakan *SegmentData* untuk mengakses data terkait *emote*. *CutsceneManager* juga memiliki hubungan asosiasi dua arah dengan *C_EmoteTrigger*, yang berarti kedua kelas tersebut dapat mengakses dan memanipulasi data masing-masing. *C_EmoteTrigger*

menggunakan *EmoteManager* untuk mengelola *emote*, yang berarti *C_EmoteTrigger* bergantung pada *EmoteManager*. Hubungan ini menunjukkan bahwa *CutsceneManager* mengakses *SegmentData* untuk data *emote*, berinteraksi dua arah dengan *C_EmoteTrigger* dalam mengelola *emote*, dan bergantung pada *EmoteManager* untuk mengelola *emote* secara umum.

d. FrameInteractionTrigger

Interaksi berbasis *frame* juga menjadi bagian penting dalam *cutscene*. *Frame Interaction Trigger* memungkinkan *game designer* untuk mendefinisikan interaksi berbasis frame dalam *cutscene*, menggunakan *ScriptableObject* yang menyimpan data interaksi. *Frame Interaction Trigger* bekerja dengan *Frame Interaction Manager* untuk memulai dan menyelesaikan interaksi berdasarkan data yang telah ditentukan. Hal ini memberi *game designer* kontrol lebih dalam merancang interaksi yang terjadi dalam setiap segmen *cutscene*.



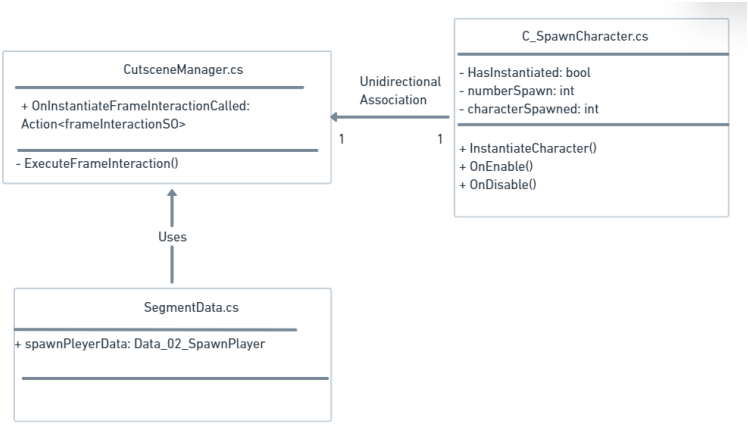
Gambar 3.10 *Class Diagram FramerInteractionTrigger*

Diagram pada Gambar 3.11 ini menunjukkan hubungan *uses* dan *bidirectional association* antara empat kelas, yaitu *CutsceneManager*, C*_FrameInteractionTrigger*, *FrameInteractionManager*, dan *SegmentData*. *CutsceneManager* bertanggung jawab untuk memicu interaksi berbasis frame dengan memanggil metode

ExecuteFrameInteraction(), yang kemudian memanggil fungsi pada C_*FrameInteractionTrigger* melalui event *OnInstantiateFrameInteractionCalled*. C_*FrameInteractionTrigger* bertugas memicu interaksi berbasis *frame*, yang dikelola melalui metode *SetFrameInteraction*(), dengan menggunakan data *frameInteractionSO*. Kelas ini juga mengelola aktifasi dan deaktivasi proses dengan metode seperti *OnEnable*() dan *OnDisable*(). *FrameInteractionManager* mengelola logika terkait interaksi berbasis frame, termasuk metode *EnterFrameInteraction*() untuk memulai interaksi dan *OnFrameExit*() untuk menangani logika ketika interaksi selesai. *SegmentData* menyimpan data interaksi berbasis frame dalam *frameInteraction*, yang kemudian digunakan oleh *CutsceneManager* untuk memicu interaksi tersebut. Hubungan *uses* antara *CutsceneManager* dan *SegmentData* serta *FrameInteractionManager* dan *C_FrameInteractionTrigger* menunjukkan bahwa *CutsceneManager* menggunakan data dari *SegmentData* untuk mengatur interaksi dan *C_FrameInteractionTrigger* menggunakan fungsi dari *FrameInteractionManager* untuk memproses interaksi berbasis frame. Hubungan *bidirectional association* mengindikasikan bahwa *C_FrameInteractionTrigger* dan *CutsceneManager* saling berinteraksi dalam pengaturan dan pemrosesan interaksi berbasis frame.

e. SpawnCharacter

Saat *cutscene* memerlukan karakter baru, *Spawn Character* digunakan untuk menginstansiasi karakter dalam *game*. Dengan atribut seperti *numberSpawn*, *game designer* dapat menentukan berapa banyak karakter yang perlu di-*spawn* pada titik tertentu dalam *cutscene*. *Spawn Character* berfungsi dengan *Cutscene Manager* untuk memastikan karakter yang di-spawn muncul di waktu yang tepat, mengikuti alur cerita yang telah ditentukan.

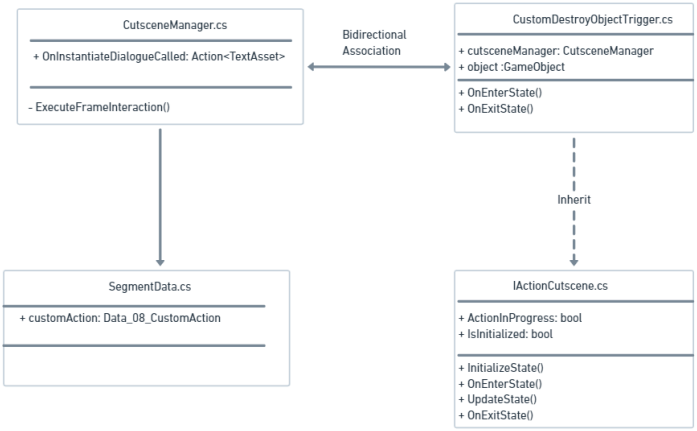


Gambar 3.11 *Class Diagram SpawnCharacter*

Diagram pada Gambar 3.12 ini menunjukkan hubungan *uses* dan *unidirectional association* antara tiga kelas, yaitu *CutsceneManager*, *C_SpawnCharacter*, dan *SegmentData*. *CutsceneManager* bertanggung jawab untuk memicu interaksi dalam *game* melalui metode *ExecuteFrameInteraction*(), yang dapat memanggil metode *OnInstantiateFrameInteractionCalled*. Kelas ini berhubungan dengan C*_SpawnCharacter*, yang mengelola proses pemunculan karakter menggunakan metode InstantiateCharacter() untuk membuat karakter baru dalam scene. C*_SpawnCharacter* juga memiliki atribut seperti HasInstantiated untuk menandakan apakah karakter sudah dipunculkan dan *numberSpawn* untuk menghitung jumlah karakter yang dipunculkan. Fungsi lainnya, seperti *OnEnable*() dan *OnDisable*(), digunakan untuk mengaktifkan dan menonaktifkan karakter. *SegmentData* menyimpan data terkait karakter yang dipunculkan, yaitu *spawnPlayerData* yang berisi informasi mengenai karakter yang akan dipunculkan, yang kemudian digunakan oleh *CutsceneManager* untuk memproses dan memanggil interaksi tersebut. Hubungan *uses* ini menunjukkan bahwa *CutsceneManager* mengandalkan data dari *SegmentData* untuk memulai interaksi, sementara hubungan unidirectional association menunjukkan bahwa *CutsceneManager* memanggil fungsi pada *C_SpawnCharacter* untuk memunculkan karakter tanpa adanya hubungan balik.

f. *CustomAction*

Untuk memenuhi kebutuhan aksi khusus yang diperlukan dalam *cutscene*, *game designer* dapat menggunakan *Custom Action*. Fitur ini melibatkan dua antarmuka, *ICustomActionTrigger* dan *IActionCutscene*, yang memungkinkan *game designer* mendefinisikan dan mengelola aksi-aksi yang terjadi selama *cutscene*. *Custom Action* memperluas kemampuan *Cutscene Manager* dengan menyediakan kontrol lebih atas logika aksi, serta melacak apakah aksi sedang berlangsung atau sudah selesai, untuk memastikan setiap aksi berjalan sesuai dengan urutan yang diinginkan.



Gambar 3.12 *Class Diagram Custom Action*

Diagram pada gambar menunjukkan hubungan *bidirectional association* dan *inheritance* antara empat kelas: *CutsceneManager*, SegmentData, *CustomDestroyObjectTrigger*, dan *IActionCutscene*. CutsceneManager bertanggung jawab untuk memicu dialog dalam game melalui event *OnInstantiateDialogueCalled* yang mengelola interaksi dialog dan fungsi *ExecuteFrameInteraction()* untuk pengelolaan adegan. Kelas ini berasosiasi dengan *SegmentData*, yang menyimpan data khusus berupa *customAction* untuk mengatur tindakan khusus dalam segmen tertentu. *CustomDestroyObjectTrigger* mengimplementasikan antarmuka *IActionCutscene* dan terhubung secara langsung dengan

CutsceneManager melalui asosiasi *bidirectional*, memungkinkan pengelolaan objek yang dihancurkan dalam adegan. *IActionCutscene* bertindak sebagai antarmuka dengan fungsi-fungsi seperti *InitializeState(), OnEnterState(), UpdateState()*, dan *OnExitState()* untuk mendefinisikan perilaku aksi dalam sebuah *cutscene*. Hubungan *inheritance* antara *CustomDestroyObjectTrigger* dan *IActionCutscene* menunjukkan bahwa *CustomDestroyObjectTrigger* menggunakan logika yang diatur oleh antarmuka ini untuk transisi status *cutscene*.

3.3 Implementasi

3.3.1 *Segment Data*

Segment Data adalah struktur data yang menyimpan informasi setiap segmen *cutscene*, seperti pergerakan kamera, dialog, dan *emote* yang ditampilkan oleh karakter. Setiap segmen dalam *cutscene* didefinisikan dengan data yang mendeskripsikan aksi yang harus terjadi pada waktu tertentu. Dengan menggunakan *ScriptableObject*, data untuk setiap segmen disimpan dan dapat dengan mudah dikelola dalam *editor Unity*.

Segment Data memungkinkan untuk menyusun dan mengurutkan setiap bagian dari cutscene, dengan mengatur atribut-atribut seperti posisi kamera, waktu transisi, teks dialog, karakter yang muncul, serta efek visual yang menyertainya. Data ini dapat digunakan oleh *Cutscene Manager* untuk memanipulasi elemen-elemen dalam scene sesuai dengan urutan yang telah ditetapkan

3.3.2 Segment Data Editor

Segment Data Editor berfungsi sebagai alat bantu dalam sistem *cutscene* ini yang memungkinkan *game designer* untuk mengelola dan mengedit *Segment Data* secara visual di *Unity Editor*. *Editor* ini mempermudah *game designer* untuk melakukan perubahan atau penyesuaian pada setiap segmen *cutscene*, seperti menambah, menghapus, atau mengubah atribut yang ada, tanpa harus menulis kode secara langsung.

Dalam konteks sistem ini, *Segment Data Editor* memungkinkan *game designer* untuk melihat dan mengedit data yang terkait dengan segmen *cutscene* dengan lebih mudah, termasuk pengaturan posisi kamera, teks dialog, dan interaksi karakter atau objek lainnya. *Game designer* dapat dengan cepat menyesuaikan properti-properti tersebut secara langsung di

editor, yang mengurangi kemungkinan kesalahan input dan mempercepat alur pengembangan *cutscene*.

Fungsi dari *Segment Data Editor* di sistem ini adalah untuk memberikan kontrol lebih besar kepada *game designer* atas konten dan urutan *cutscene*, dengan tampilan antarmuka yang intuitif, memungkinkan mereka untuk lebih efisien dalam merancang dan mengelola pengalaman cutscene tanpa perlu bergantung pada kode secara manual.

3.3.3 *Cutscene Manager*

Cutscene Manager adalah komponen yang mengelola alur dari seluruh *cutscene* dalam *game*. Script ini berfungsi untuk mengatur urutan pemanggilan setiap segmen cutscene berdasarkan data yang telah disiapkan dalam *Segment Data*. Dengan *Cutscene Manager*, pengembang dapat mengontrol eksekusi setiap langkah dalam *cutscene*, memicu berbagai event seperti pergerakan kamera, dialog, animasi, dan interaksi dengan objek lain dalam *game*.

Fungsi utama dari *Cutscene Manager* adalah sebagai pengatur alur dari *cutscene*. Manager ini akan memulai dan menghentikan *cutscene* sesuai dengan kondisi yang telah ditentukan. Sebagai contoh, saat *player* memasuki area tertentu atau mencapai tujuan tertentu, *Cutscene Manager* akan men-trigger *cutscene* yang telah diprogram.

3.3.3.1 *Script Action Cutscene*

Untuk mengatur behaviour dalam *cutscene manager* dibutuhkan script tambahan yang bernama *Segment Script.Segment Script* adalah kumpulan skrip yang digunakan untuk mengatur setiap tindakan yang terjadi dalam segmen *cutscene*. Setiap segmen memiliki skrip yang spesifik untuk menangani logika terkait dengan peristiwa yang terjadi pada segmen tersebut, seperti pergerakan kamera, pemutaran dialog, atau pemunculan karakter. Di bawah ini adalah penjelasan tentang beberapa jenis *Segment Script* yang diimplementasikan:

a. CameraMovement

CameraMovement bertanggung jawab untuk mengatur pergerakan kamera selama *cutscene*. Dalam script ini, kita menggunakan sistem *Cinemachine* untuk menciptakan pergerakan kamera yang halus dan dinamis. Kamera dapat bergerak dari satu titik ke titik lain, mengikuti

karakter atau objek tertentu, atau bahkan memutar sudut pandang kamera sesuai dengan kebutuhan *cutscene*.

Fungsi Utama:

- Memindahkan kamera sesuai dengan segmen yang ditentukan.
- Mengontrol kecepatan dan jarak pergerakan kamera.
- Mengatur transisi yang mulus antar posisi kamera.

```
   VARIABLES:
       - isMoving: Boolean to store the status of
whether the camera is moving.
       - startPosition: The starting position of the
camera.
       - journeyLength: The total distance between the
start and target positions.
       - remainingDistance: The distance left to reach
the target.
       - totalTime: The total time required to complete
the journey.
       - tweener: A Tweener object for animations (not
yet used in this script).
       - canMove: A flag to control whether the camera
is allowed to move (default: true).
```

Kode 3.1 Kode definisi *variable*

Bagian ini mendefinisikan variabel-variabel yang digunakan untuk mengatur logika pergerakan kamera, seperti status pergerakan, posisi, dan kontrol logika.

```
METHOD OnEnable():
    ADD          event          listener          for
CutsceneManager.OnMoveCameraCalled.
END METHOD
METHOD OnDisable():
    REMOVE         event          listener          for
CutsceneManager.OnMoveCameraCalled.
END METHOD
```

Kode 3.2 Kode Event Listener

Bagian ini memastikan bahwa metode *MoveCamera* hanya dipanggil saat kelas ini diaktifkan dan mencegah *error* atau *memory leak* dengan menghapus *listener* saat kelas dinonaktifkan.

```
METHOD MoveCamera(camera, targetPoint, movementSpeed,
startPositions):
    SET canMove TO true
    CONFIGURE delayTime (e.g., 1 second)
    CALL coroutine MoveCameras TO move the camera
END METHOD
```

Kode 3.3 *Move Camera function*

MoveCamera adalah metode utama yang mengatur logika awal pergerakan kamera. Metode ini menginisialisasi *flag canMove* untuk memastikan kamera dapat bergerak dan memanggil coroutine *MoveCameras* untuk memulai pergerakan.

```
METHOD MoveCameras(camera, targetPoint, movementSpeed,
startPositions, delayTime):
    CALCULATE startPosition BY adjusting the camera's Z-
axis
    CALCULATE targetPosition IN the same manner
    COMPUTE journeyLength AS the total distance
    SET isMoving TO true
    CALCULATE totalMoveTime BASED ON journeyLength AND
movementSpeed
    SET totalTime TO totalMoveTime
    SET delayTime TO 0
    WAIT FOR the next frame
    WHILE elapsedTime < totalMoveTime AND canMove:
        INCREMENT elapsedTime BY the elapsed duration
        COMPUTE fractionOfJourney BASED ON the current
progress of the journey
    END WHILE
    AFTER movement is complete:
        SET remainingDistance TO 0
        SET canMove TO false TO stop movement
        ENSURE camera's position MATCHES targetPosition
        SET isMoving TO false
END METHOD
```

Kode 3.4 *Move Cameras Function*

Coroutine ini mengatur transisi kamera secara bertahap dari posisi awal ke posisi target menggunakan interpolasi linier *(Lerp)*. Logika utama melibatkan perhitungan waktu yang berlalu (*elapsedTime*) untuk memastikan kamera bergerak sesuai kecepatan (*movementSpeed*) hingga

mencapai target. Setelah selesai, *flag* seperti *canMove* dan *isMoving* diperbarui.

b. DialogueTrigger

DialogueTrigger mengatur pemunculan dialog dalam *cutscene*. Ketika segmen *cutscene* mencapai titik tertentu, *DialogueTrigger* akan memicu tampilan dialog di layar. Dialog ini bisa berupa teks atau suara yang dibacakan oleh karakter dalam *cutscene*. *DialogueTrigger* juga mengelola timing untuk setiap kalimat atau paragraf yang muncul di layar.

Fungsi Utama:

- Memicu tampilan dialog pada waktu yang tepat dalam cutscene.
- Mengatur teks atau suara yang diputar bersamaan dengan dialog.

```
METHOD Initialize():
    IF dialogueManager IS NULL THEN
        Request dependency injection for dialogueManager
using CustomInjector
    END IF
END METHOD
```

Kode 3.5 Inisiasi *Dialogue Manager*

Fungsi ini memastikan bahwa *DialogueManager* sudah siap untuk digunakan dalam sistem. Jika *instance DialogueManager* belum tersedia, metode akan secara otomatis meminta injeksi dependensi melalui *CustomInjector* untuk memastikan semua komponen yang dibutuhkan tersedia dan dapat berfungsi dengan baik.

Fungsi ini bertugas memastikan bahwa data dialog dalam *inkJSON* tersedia sebelum melanjutkan proses. Jika data tersebut tidak ditemukan, proses dihentikan dengan menampilkan peringatan di log. Jika data dialog valid, fungsi memulai dialog dengan memanggil metode *EnterDialogueMode*() pada *DialogueManager*. Selain itu, *listener* untuk *event OnDialogExit* ditambahkan untuk memastikan proses *cutscene* dapat dilanjutkan setelah dialog selesai.

Fungsi ini bertanggung jawab untuk memastikan kebersihan sistem dengan menghapus *event listener OnDialogExit* setelah dialog selesai. Hal ini dilakukan untuk mencegah potensi kebocoran memori atau

pemanggilan berulang yang tidak diinginkan, menjaga efisiensi dan kestabilan sistem selama eksekusi *cutscene*.

c. EmoteTrigger

EmoteTrigger digunakan untuk mengontrol perubahan *emote* atau ekspresi wajah karakter dalam *cutscene*. Selama *cutscene*, karakter dapat menampilkan ekspresi yang mencerminkan perasaan mereka, seperti senyuman, kemarahan, atau ketakutan. *EmoteTrigger* akan memicu perubahan ini pada waktu yang sesuai dengan alur cerita dalam *cutscene*.

Fungsi Utama:

- Mengubah ekspresi wajah atau emote karakter selama cutscene.
- Memperkuat narasi melalui ekspresi emosional karakter.

```
    DECLARATION OF VARIABLES:
        emoteManager: Reference to the EmoteManager used
for managing character emotes.
        cutsceneManager:      Reference      to      the
CutsceneManager used for managing cutscene segments.
    END DECLARATION
```

Kode 3.6 Deklarasi *Variable Manager*

Di bagian ini, dua variabel penting dideklarasikan. emoteManager digunakan untuk mengakses dan mengelola emote yang terkait dengan karakter, sedangkan cutsceneManager digunakan untuk menyelesaikan segment cutscene.

```
    METHOD Start():
        GET  CutsceneManager  component  FROM  the  current
GameObject
        IF emoteManager IS null:
            CALL CustomInjector.RequestInject() TO inject
emoteManager
    END METHOD
```

Kode 3.7 Inisiasi *Cutscene & Emote Manager*

Pada Saat *game* dimulai, *Start()* menginisialisasi *cutsceneManager* dengan mengambil komponen yang ada pada objek yang sama. Jika

emoteManager belum terinjeksi, maka dilakukan injeksi manual menggunakan *CustomInjector.RequestInject(this).*

```
METHOD OnEnable():
    ADD                    event                    listener
CutsceneManager.OnCharacterEmoteCalled    WITH    callback
SetEmote
END METHOD
```

Kode 3.8 *Event Listener* untuk *cutscene manager*

Pada saat objek diaktifkan, *event listener* didaftarkan untuk mendengarkan pemanggilan *event OnCharacterEmoteCalled* dari *CutsceneManager*. Ketika *event* ini dipanggil, *callback SetEmote* akan dipanggil untuk menjalankan tindakan yang sesuai. Pada saat objek

```
    METHOD SetEmote(chars, emoteName, emoteDuration):
        IF chars == null OR emoteName == null THEN:
            LOG    Warning:    "[FrameInteractionTrigger]
frameSO is null !!"
            RETURN
        IF emoteDuration == 0 THEN:
            SET emoteDuration = 1
        //  Ambil  emote  dari  emoteManager  berdasarkan
posisi emoteHandler
        VAR                    emote                    =
emoteManager.GetEmote(chars.emoteHandler.position)
        // Aktifkan gameObject emote
        emote.gameObject.SetActive(true)
        // Mainkan animasi emote dengan nama emoteName
        emote.PlayAnimator(emoteName, () =>
        {
            // Setelah emoteDuration selesai
            DOVirtual.DelayedCall(emoteDuration, () =>
            {
                // Kembalikan emote ke emoteManager
                emoteManager.ReturnEmote(emote)

                // Selesaikan segment cutscene
                cutsceneManager.FinishCutsceneSegment()
            })
        })
    END METHOD
```

Kode 3.9 Kode aktivasi *emote*

dinonaktifkan event listener dihapus untuk mencegah pemanggilan event yang tidak diinginkan.

Metode *SetEmote* dipanggil saat *event OnCharacterEmoteCalled* dipicu. Langkah pertama adalah memeriksa apakah *chars* atau *emoteName null*. Jika ada yang *null*, metode akan keluar dengan menampilkan peringatan.

Selanjutnya, jika durasi *emote* (*emoteDuration*) adalah 0, maka akan diubah menjadi 1 untuk menghindari durasi 0 detik. Kemudian, *emote* diambil dari *emoteManager* berdasarkan posisi *emote* yang dimiliki karakter. *Emote* tersebut diaktifkan dan animasinya diputar sesuai dengan nama *emote* yang diberikan.

Setelah animasi selesai, *DOVirtual.DelayedCall* digunakan untuk menunggu selama *emoteDuration* detik sebelum mengembalikan *emote* ke *emoteManager* dan menyelesaikan segmen *cutscene* dengan memanggil *cutsceneManager.FinishCutsceneSegment*().

d. FrameInteractionTrigger

FrameInteractionTrigger memungkinkan interaksi dengan elemen lain dalam *cutscene*, seperti objek atau tombol. Sebagai contoh, ini dapat digunakan untuk memicu interaksi seperti membuka pintu atau mengaktifkan efek visual tertentu pada objek yang berinteraksi dengan karakter.

Fungsi Utama:

- Memicu interaksi dengan objek dalam *cutscene*.
- Menambahkan elemen *gameplay* atau efek visual yang interaktif dalam *cutscene*

```
METHOD Start():
    ASSIGN cutsceneManager WITH COMPONENT CutsceneManager
    IF frameInteraction IS NULL:
        CALL   CustomInjector.RequestInject(THIS)      //
Request dependency injection
END METHOD
```

Kode 3.10 Inisiasi *Cutscene & Frame Interaction Manager*

Metode *Start* digunakan untuk menginisialisasi variabel cutsceneManager dengan mendapatkan komponen *CutsceneManager* dari objek yang sama. Selain itu, jika *frameInteraction* belum terisi, skrip meminta *dependency injection* melalui *CustomInjector.RequestInject*.

Hal ini memastikan bahwa semua referensi yang dibutuhkan tersedia sebelum skrip mulai menjalankan fungsi lainnya.

```
METHOD OnEnable():
    SUBSCRIBE CutsceneManager.OnFrameInteractionCalled TO
SetFrameInteraction
END METHOD
METHOD OnDisable():
    UNSUBSCRIBE  CutsceneManager.OnFrameInteractionCalled
FROM SetFrameInteraction
END METHOD
```

Kode 3.11 *Event Listener* untuk *Cutscene Manager*

Metode *OnEnable* dan *OnDisable* mengatur langganan *event* untuk menangani interaksi *frame*. *OnEnable* memastikan bahwa metode *SetFrameInteraction* akan dipanggil ketika event *OnFrameInteractionCalled* dipicu. Sebaliknya, *OnDisable* memastikan untuk berhenti berlangganan ketika objek dinonaktifkan, mencegah potensi kesalahan akibat *event* yang dipanggil pada objek yang tidak aktif.

```
METHOD    SetFrameInteraction(frameInteractionSO    AS
FrameInteractionSO):
    IF frameInteractionSO IS NULL:
        LOG WARNING "[FrameInteractionTrigger] frameSO
is null !!"
        RETURN
    END IF
    CALL
frameInteraction.EnterFrameInteraction(frameInteractio
nSO)  // Begin interaction
    SUBSCRIBE      frameInteraction.OnFrameExit      TO
FrameInteractionCompleted
END        METHOD                                  CALL
frameInteraction.EnterFrameInteraction(frameInteractio
nSO)  // Begin interaction
    SUBSCRIBE      frameInteraction.OnFrameExit      TO
FrameInteractionCompleted
END METHOD
```

Kode 3.12 *Function* untuk *Frame Interaction Trigger*

Metode *SetFrameInteraction* bertanggung jawab untuk memulai interaksi frame berdasarkan data yang diterima melalui parameter

frameInteractionSO. Jika data tersebut tidak ada, skrip mencatat peringatan dan menghentikan eksekusi lebih lanjut. Jika data valid, skrip memulai interaksi melalui *EnterFrameInteraction* dan berlangganan *event OnFrameExit* untuk mendeteksi kapan interaksi selesai.

```
METHOD FrameInteractionCompleted():
    UNSUBSCRIBE    frameInteraction.OnFrameExit    FROM
FrameInteractionCompleted
    CALL cutsceneManager.FinishCutsceneSegment()
END METHOD
```

Kode 3.13 *Function* untuk *exit frame interaction*

Metode ini dipanggil ketika interaksi *frame* selesai. Langkah pertama adalah berhenti berlangganan dari event *OnFrameExit* untuk mencegah pemanggilan berulang. Kemudian, metode ini memberi tahu *cutsceneManager* bahwa segmen *cutscene* saat ini telah selesai, memungkinkan *cutscene* untuk melanjutkan ke segmen berikutnya.

e. SpawnCharacter

SpawnCharacter bertanggung jawab untuk memunculkan karakter atau *NPC* di dalam cutscene pada waktu yang tepat. Misalnya, jika ada karakter yang muncul secara tiba-tiba dalam sebuah segmen, script ini akan memanggil karakter tersebut dan menempatkannya di posisi yang telah ditentukan.

Fungsi Utama:

- Memunculkan karakter atau *NPC* pada saat yang tepat dalam cutscene.
- Mengatur posisi dan animasi karakter yang muncul.

```
DECLARE hasInstantiated AS BOOLEAN INITIALLY FALSE
DECLARE numberOfSpawn AS INTEGER INITIALLY 1
DECLARE characterSpawned AS INTEGER INITIALLY 0
```

Kode 3.14 Deklarasi *Variable Spawn Segment*

Pada bagian ini, tiga variabel dideklarasikan untuk mengelola status dan jumlah karakter yang dapat di-*spawn*:

- *hasInstantiated*: Menyimpan status apakah karakter telah diinstansiasi. Nilai awal adalah *false*.

- *numberOfSpawn*: Menentukan jumlah maksimum karakter yang dapat di-*spawn*. Nilai defaultnya adalah 1.
- *characterSpawned*: Menyimpan jumlah karakter yang telah di-*spawn*. Nilai awalnya adalah 0.

Variabel-variabel ini memungkinkan sistem untuk membatasi jumlah karakter yang dapat di-*spawn* dan menghindari penginstansian karakter yang sama lebih dari sekali.

```
METHOD Start():
    ASSIGN hasInstantiated WITH FALSE
END METHOD
```

Kode 3.15 *Start Function*

Metode *Start* mengatur ulang nilai *hasInstantiated* menjadi *false* setiap kali objek aktif atau dimulai. Ini memastikan bahwa status awal memungkinkan instansiasi karakter baru jika skrip digunakan kembali.

```
METHOD OnEnable():
    SUBSCRIBE LevelManager.OnInstantiateCharacterCalled
TO InstantiateCharacter
    SUBSCRIBE
CutsceneManager.OnInstantiateCharacterCalled         TO
InstantiateCharacter
END METHOD
METHOD OnDisable():
    UNSUBSCRIBE
LevelManager.OnInstantiateCharacterCalled          FROM
InstantiateCharacter
    UNSUBSCRIBE
CutsceneManager.OnInstantiateCharacterCalled       FROM
InstantiateCharacter
END METHOD
```

Kode 3.16 *Event Listener* untuk *Cutscene Manager*

OnEnable Mengatur langganan pada event *OnInstantiateCharacterCalled* dari *LevelManager* dan *CutsceneManager*. Ketika event ini dipicu, metode *InstantiateCharacter* akan dijalankan.

OnDisable Menghapus langganan *event* saat objek dinonaktifkan, mencegah *error* akibat pemanggilan *event* pada objek yang sudah tidak aktif

```
METHOD  InstantiateCharacter(character  AS  GameObject,
targetSpawn AS Transform):
    ASSIGN characterSpawned WITH 0
    LOG "Spawn Character"
    IF characterSpawned < numberOfSpawn THEN:
        IF NOT hasInstantiated THEN:
            INCREMENT characterSpawned BY 1
            DECLARE newCharacter AS GameObject
            ASSIGN          newCharacter          WITH
Instantiate(character,            targetSpawn.position,
targetSpawn.rotation, THIS.transform)
            ASSIGN        newCharacter.name        WITH
character.name
            ASSIGN hasInstantiated WITH TRUE
        ELSE:
            LOG     WARNING     "Character     already
instantiated."
        END IF
    ELSE:
        LOG WARNING "Character spawn limit reached."
    END IF
END METHOD
```

Kode 3.17 *Function* untuk *instantiate Character*

Metode ini bertugas untuk menginstansiasi karakter berdasarkan parameter *character* (prefab karakter) dan *targetSpawn* (posisi serta rotasi tempat *spawn*). Proses dimulai dengan mereset jumlah karakter yang telah di-*spawn* (*characterSpawned*) menjadi nol. Selanjutnya, metode memeriksa apakah jumlah karakter yang telah di-*spawn* kurang dari batas maksimal (*numberOfSpawn*). Jika belum, dan karakter belum pernah diinstansiasi sebelumnya (*hasInstantiated* = *false*), metode menambah jumlah karakter yang di-*spawn*, menginstansiasi karakter pada posisi dan rotasi *targetSpawn*, menetapkan transformasi induknya ke objek saat ini, lalu menandai bahwa karakter telah diinstansiasi (*hasInstantiated* = *true*). Jika karakter sudah diinstansiasi, metode akan mencatat peringatan bahwa karakter tersebut sudah ada. Jika batas jumlah karakter telah tercapai, metode juga mencatat peringatan bahwa spawn limit telah terpenuhi.

f. Custom Action

Custom Action adalah *script* yang memungkinkan pengembang untuk menambahkan aksi-aksi khusus yang tidak termasuk dalam segmen standar, seperti memutar efek visual, mengaktifkan suara latar tertentu, atau menjalankan animasi yang unik untuk situasi khusus dalam cutscene.

Fungsi Utama:

- Menjalankan aksi atau logika kustom yang dibutuhkan dalam *cutscene*.
- Memungkinkan pengembang untuk memperkaya *cutscene* dengan elemen-elemen unik.

```
ActionInProgress: Public boolean, menandai apakah aksi
sedang berlangsung. Digunakan untuk pengendalian alur.
IsInitialized: Private boolean dengan getter publik,
memastikan bahwa state telah diinisialisasi sebelum
memulai.
detailedManager: Public DetailedObjectManager, diatur
melalui Reflex untuk mengelola objek yang terperinci.
cutsceneManager: Protected CutsceneManager, bertugas
mengontrol dan mengelola cutscene dalam game.
```

Kode 3.18 Deklarasi *variable*

Variabel ActionInProgress memastikan hanya satu aksi yang berjalan pada suatu waktu untuk mencegah konflik antar state, sedangkan *IsInitialized* digunakan untuk memastikan bahwa aksi tidak dilakukan sebelum state siap digunakan. *Variabel detailedManager* menyediakan manajemen objek terperinci dalam *game*, sementara *cutsceneManager* memungkinkan integrasi aksi dengan sistem *cutscene game*, memastikan setiap aksi dapat berjalan selaras dengan alur narasi.

```
METHOD InitializeState():
    SET IsInitialized TO true  // Menandai bahwa state
telah siap digunakan
    ASSIGN cutsceneManager TO CutsceneManager component
pada object saat ini
END METHOD
```

Kode 3.19 Inisiasi *Cutscene Manager*

Metode ini bertujuan untuk menginisialisasi status state sebelum menjalankan aksi, sekaligus mengambil referensi dari *CutsceneManager* untuk memastikan *state* dapat berinteraksi dengan sistem *cutscene* secara optimal

```
METHOD OnEnterState():
    SET ActionInProgress TO true  // Menandai bahwa aksi
sedang berlangsung
END METHOD
```

Kode 3.20 *Enter State Function*

Metode ini menandai bahwa aksi telah dimulai sekaligus mempersiapkan *state* untuk menjalankan logika utama selama state tersebut aktif.

```
METHOD UpdateState():
    // Placeholder untuk logika runtime saat state aktif
END METHOD
```

Kode 3.21 *Update State Function*

Metode ini berfungsi sebagai titik pengembangan untuk mengimplementasikan logika yang berjalan saat state aktif, dengan memberikan fleksibilitas untuk di-*override* pada kelas turunan karena tidak memiliki implementasi default.

```
METHOD OnExitState():
    LOG "Out" TO console   // Catat bahwa state sedang
keluar
    SET ActionInProgress TO false  // Menandai bahwa aksi
telah selesai
    SET IsInitialized TO false     // Reset status
inisialisasi state
END METHOD
```

Kode 3.22 *Exit State Function*

Metode ini menangani proses akhir dari sebuah *state* dengan menandai bahwa aksi telah selesai melalui pengaturan ulang status *ActionInProgress* dan *IsInitialized*, serta memberikan *log* untuk membantu debugging dan melacak kapan *state* berakhir.

3.3.5 Engine yang digunakan

Engine yang digunakan untuk implementasi *cutscene system* adalah *unity*

BAB IV
SKENARIO PENGUJIAN

4.1 Tujuan Pengujian

Tujuan pengujian dalam pengembangan *Game* Project Botanical adalah untuk memastikan bahwa sistem *cutscene* yang dikembangkan dapat mendukung narasi secara efektif, memberikan transisi yang mulus antara *gameplay* dan cutscene, serta memenuhi semua kebutuhan fungsional dan non-fungsional yang telah diidentifikasi. Pengujian ini juga bertujuan untuk mengidentifikasi dan memperbaiki masalah yang mungkin terjadi selama implementasi, memastikan sistem yang *modular*, responsif, dan kohesif.

Pengujian ini dikaitkan dengan rumusan masalah sebagai berikut:

1. Untuk memastikan sistem *cutscene* dapat mendukung narasi secara efektif, pengujian akan mengevaluasi bagaimana *cutscene* menyajikan cerita, dialog, dan peristiwa penting dalam *game*.
2. Untuk memastikan transisi yang mulus antara *gameplay* dan *cutscene*, pengujian akan mengamati transisi antar adegan dan bagaimana pemain merasakan perpindahan antara *gameplay* dan *cutscene*.
3. Untuk memastikan fitur utama dalam sistem *cutscene*, pengujian akan mengecek keberadaan dan fungsionalitas fitur seperti dialog, animasi, perubahan kamera, efek suara, dan musik latar.
4. Untuk memastikan sistem *cutscene* yang *modular* dan mudah dikembangkan, pengujian akan mengevaluasi kemampuan sistem dalam menambahkan, menghapus, atau mengubah *cutscene* tanpa mengganggu sistem lainnya.

4.2 Lingkungan Pengujian

Pengujian ini dilakukan pada lingkungan pengujian dengan spesifikasi sebagai berikut:

1. Perangkat Komputer atau Laptop
2. Perangkat memiliki minimum *operating system* yang digunakan berupa *Windows* 10
3. Perangkat dengan minimum spesifikasi *processor intel core* i3-3200

4. Perangkat dengan minimum *RAM 4GB DDR* 3
5. Perangkat dengan *Intel Graphics* 4000.
6. Perangkat memiliki *web browser*
7. Koneksi internet yang stabil dengan kecepatan *bandwith* minimal 1 *Mbps*
8. Ruang pengujian yang tenang

4.3 Target Pengujian

Pengujian Sistem *Cutscene In Game* dari Project Botanical dilakukan oleh *game designer* dari Project Botanical sebagai pengguna utama dari *Cutscene In Game*.

4.4 Proses Pengujian

Pengujian akan dilakukan dalam dua kategori utama: pengujian fungsional dan pengujian sistem.

4.4.1 Analisa Kebutuhan Fungsional

Pengujian fungsional akan fokus pada verifikasi bahwa setiap fitur dalam sistem cutscene berfungsi sesuai dengan spesifikasi. Beberapa skenario pengujian fungsional meliputi:

1. Pengujian Dialog:
   - Verifikasi tampilan dialog teks dengan opsi interaktif.
   - Sinkronisasi dialog dengan animasi karakter dan audio.
   - Kemampuan untuk melewati atau mengulang dialog.
2. Pengujian Animasi Karakter:
   - Memastikan animasi karakter berfungsi sesuai dengan narasi.
   - Integrasi animasi dengan dialog dan *cutscene*.
3. Pengujian Kamera:
   - Verifikasi sudut pandang dan gerakan kamera yang mendukung narasi.
   - Transisi kamera yang halus antara *gameplay* dan *cutscene*.
4. Pengujian *Frame Interaction*:
   - Kemampuan untuk Menampilkan bingkati frame dengan gambar dan *text*
   - Sinkronisasi *Text* dengan Gambar

4.4.2 Pengujian Fitur

Pengujian fitur difokuskan untuk menguji fitur-fitur yang ada dalam *Cutscene In Game* yang dilakukan oleh *game designer*. Dalam proses pengujian fitur, dirancang skenario pengujian yang dapat dilihat pada tabel di bawah ini

a. *Movement Segment*

Tabel 4.1 Pengujian *Movement Segment*

Fitur	Skenario Pengujian	Indikator Keberhasilan
Trigger Movement Segment	Atur *start delay* pada *Movement Segment* menjadi 0. Jalankan sistem. Amati apakah *Movement Segment* mulai tanpa ada penundaan.	*Movement Segment* langsung mulai ketika dijalankan tanpa jeda waktu.
Movement Character	Tentukan sebuah *target point* dalam dunia *game*. Jalankan *Movement Segment* dengan *target point* yang telah ditentukan. Amati apakah karakter bergerak menuju *target point*.	Karakter bergerak menuju koordinat *target point* yang telah ditentukan
Character Animation	Pastikan *animation controller* dan *state machine* karakter terhubung dengan parameter gerakan. Jalankan *Movement Segment*. Amati apakah animasi bergerak sesuai dengan gerakan karakter.	Animasi karakter menampilkan gerakan berjalan atau berlari yang sinkron dengan pergerakan fisik karakter.

Fitur	Skenario Pengujian	Indikator Keberhasilan
Movement Stop	Tentukan *completed type* pada *Movement Segment* menjadi Movement. Jalankan *Movement Segment* dengan *target point* tertentu. Amati apakah karakter berhenti tepat di posisi *target point* ketika mencapai tujuan.	Karakter berhenti dengan akurat di lokasi *target point*.
Speed Multiplier	Atur *speed multiplier* dengan nilai 1 dan jalankan *Movement Segment*. Catat waktu yang diperlukan untuk mencapai *target point*. Ubah *speed multiplier* menjadi nilai lebih besar, misalnya 2. Jalankan kembali *Movement Segment* dan bandingkan waktu yang diperlukan untuk mencapai *target point*.	Karakter bergerak lebih cepat dengan nilai *speed multiplier* yang lebih besar.
End Condition	Rancang dua segmen: *Movement Segment* dan segmen berikutnya (misalnya *Chat Segment*). Jalankan *Movement Segment*. Amati apakah sistem melanjutkan ke segmen berikutnya setelah *Movement Segment* selesai.	Sistem secara otomatis berpindah ke segmen berikutnya setelah *Movement Segment* selesai.
Reusable Segment	Rancang dua Movement Segment dengan target yang berbeda.	*Movement Segment* dapat dijalankan kembali tanpa *error* atau masalah.

Fitur	Skenario Pengujian	Indikator Keberhasilan
	Amati apakah Movement Segment dapat berjalan dengan baik pada iterasi kedua.	

b. *Spawn Character Segment*

Tabel 4.2 Pengujian *Spawn Character Segment*

Fitur	Skenario Pengujian	Indikator Keberhasilan
Trigger Spawn Segment	Atur *start delay* pada *Spawn Character Segment* menjadi 0. Jalankan sistem. Amati apakah *Spawn Character Segment* langsung terpicu tanpa jeda.	*Spawn Character Segment* mulai segera setelah dijalankan tanpa jeda waktu.
Spawn Location	Tentukan titik spawn dalam dunia game (*spawn point*). Jalankan *Spawn Character Segment*. Amati apakah karakter muncul di lokasi *spawn point* yang telah ditentukan.	Karakter muncul di koordinat *spawn point* yang ditambahkan sebelumnya.
Configured Spawn	Buat beberapa *ScriptableObject* berisi Karakter yang berbeda. Assign salah satu *ScriptableObject* ke *Spawn Character Segment*. Jalankan segmen dan amati karakter yang di-*spawn*.	Karakter yang di-spawn sesuai dengan ScriptableObject yang telah di-diberikan sebelumnya.
Segment Complete	Rancang dua segmen: *Spawn Character Segment* dan segmen berikutnya (misalnya	Sistem secara otomatis berpindah ke segmen berikutnya setelah karakter di-spawn.

Fitur	Skenario Pengujian	Indikator Keberhasilan
	Wait Segment atau *Movement Segment*). Jalankan *Spawn Character Segment*. Amati apakah sistem berlanjut ke segmen berikutnya setelah proses spawn selesai.	
Single Spawn	Tentukan sebuah *Spawn Character Segment* dengan konfigurasi *spawn count* = 1. Jalankan segmen dan hitung jumlah karakter yang muncul.	Hanya satu karakter yang di-spawn setiap kali segmen dijalankan.
Reusable Segment	Rancang dua buah *Spawn* Character *Segment*, tetapi dengan SO yang berbeda Amati apakah proses spawn berjalan dengan benar pada iterasi kedua.	Spawn Character Segment dapat dijalankan kembali tanpa error atau masalah.

c. *Dialogue Segment*

Tabel 4.3 Pengujian *Dialogue Segment*

Fitur	Skenario Pengujian	Indikator Keberhasilan
Trigger Dialogue Segment	Atur segment start delay menjadi 0. Jalankan sistem. Amati apakah *Dialogue Segment* langsung terpicu tanpa ada jeda waktu.	*Dialogue Segment* langsung terpicu ketika dijalankan tanpa penundaan
Pop-up Dialogue	Aktifkan *Dialogue Segment*. Amati apakah dialog muncul (*pop-up*) di layar.	Dialog muncul di layar sesuai dengan desain yang diharapkan.

Fitur	Skenario Pengujian	Indikator Keberhasilan
Interactive Dialogue	Aktifkan *Dialogue Segment*. Interaksikan dialog dengan menekan tombol "*Space*" Amati apakah dialog berikutnya ditampilkan setelah interaksi dilakukan.	Dialog dapat dilanjutkan ke dialog berikutnya melalui interaksi pemain.
Predefined Dialogue	Assign sebuah file *ink* tertentu ke *Dialogue Segment*. Jalankan sistem. Amati apakah dialog yang muncul sesuai dengan teks dalam file *ink* yang di-assign.	Dialog yang muncul sesuai dengan konten *ink* yang telah ditentukan.
Segment Completion	Jalankan *Dialogue Segment*. Interaksikan semua dialog hingga dialog selesai. Amati apakah sistem secara otomatis berpindah ke segmen berikutnya.	Setelah dialog selesai, sistem melanjutkan ke segmen berikutnya tanpa hambatan.
Reusable Dialogue	Gunakan kembali *Dialogue Segment* pada segmen lain di dalam cutscene. Jalankan sistem. Amati apakah *Dialogue Segment* dapat berfungsi kembali di segmen tersebut.	*Dialogue Segment* dapat berfungsi dengan baik saat digunakan kembali di segmen lain

d. Remove Character Segment

Tabel 4.4 Pengujian *Character Segment*

Fitur	Skenario Pengujian	Indikator Keberhasilan
Trigger Remove Character	Atur segment start delay menjadi 0. Jalankan sistem. Amati apakah *Remove Character Segment* langsung terpicu tanpa ada jeda waktu.	*Remove Character Segment* langsung terpicu ketika dijalankan tanpa penundaan.
Character Deletion	Jalankan *Remove Character Segment*. Amati apakah karakter yang dimaksud telah berhasil dihapus dari permainan.	Karakter berhasil dihapus dari dunia permainan tanpa sisa game object atau visual object.
Configured Removal	Assign sebuah Scriptable Object (*SO*) yang mendefinisikan karakter tertentu ke *Remove Character Segment*. Jalankan sistem. Amati apakah karakter yang dihapus sesuai dengan *SO* yang telah di-assign sebelumnya.	Karakter yang dihapus adalah karakter yang didefinisikan dalam SO yang ditentukan.
Character Deletion	Jalankan *Remove Character Segment*. Amati apakah setelah karakter berhasil dihapus, sistem melanjutkan ke segmen berikutnya.	Sistem melanjutkan ke segmen berikutnya setelah karakter berhasil dihapus tanpa hambatan
Reusable Removal	Gunakan kembali *Remove Character Segment* pada segmen lain di dalam satu cutscene. Assign Scriptable Object baru untuk	*Remove Character Segment* dapat digunakan kembali dan berfungsi dengan baik di segmen berikutnya dengan karakter yang baru

Fitur	Skenario Pengujian	Indikator Keberhasilan
	karakter yang akan dihapus di segmen ini. Jalankan sistem. Amati apakah *Remove Character Segment* berfungsi kembali sesuai dengan konfigurasi baru.	

e. Camera Movement Segment

Tabel 4.5 Pengujian *Camera Movement Segment*

Fitur	Skenario Pengujian	Indikator Keberhasilan
Trigger Camera Movement	Atur segment start delay menjadi 0. Jalankan sistem. Amati apakah *Camera Movement Segment* langsung terpicu tanpa ada jeda waktu.	*Camera Movement Segment* langsung terpicu ketika dijalankan tanpa penundaan
Targeted Movement	Assign sebuah target point untuk *Camera Movement Segment*. Jalankan sistem. Amati apakah kamera bergerak menuju target point tersebut.	Kamera berhasil bergerak menuju target point yang telah di-*assign*
Configured Camera	Assign sebuah target point dan sebuah camera ke dalam *Camera Movement Segment*. Jalankan sistem. Amati apakah kamera yang bergerak sudah sesuai dengan yang ditambahkan sebelumnya	Kamera bergerak menuju target point yang sesuai dengan SO

Fitur	Skenario Pengujian	Indikator Keberhasilan
Initial Position	Tentukan titik awal untuk kamera (spawn point). Jalankan sistem. Amati apakah kamera muncul di titik spawn yang telah diatur sebelumnya.	Kamera muncul di titik spawn point sesuai pengaturan awal
Segment Completion	Jalankan *Camera Movement Segment*. Amati apakah setelah pergerakan kamera selesai, sistem melanjutkan ke segmen berikutnya.	Sistem melanjutkan ke segmen berikutnya setelah pergerakan kamera selesai
Reusable Movement	buat 2 *Camera Movement Segment* didalam satu *cutscene* dengan target yang berbeda Jalankan sistem. Amati apakah *Camera Movement Segment* di segment yang ke dua berfungsi kembali setelah digunakan sebelumnya.	*Camera Movement Segment* dapat digunakan kembali dan berfungsi dengna baik di segmen berikutnya dengan target point yang baru
Precise Stop	Assign target point untuk pergerakan kamera. Jalankan sistem. Amati apakah kamera berhenti tepat di koordinat target point.	Kamera berhenti tepat di target point tanpa melewati atau berhenti sebelum mencapai target

f. Character Emoji Segment

Tabel 4.6 Pengujian *Character Emoji Segment*

Fitur	Skenario Pengujian	Indikator Keberhasilan
Trigger Emoji Segment	Atur segment start delay menjadi 0.	*Character emoji Segment* langsung

Fitur	Skenario Pengujian	Indikator Keberhasilan
	Jalankan sistem. Amati apakah *Character Emoji Segment* langsung terpicu tanpa ada jeda waktu.	terpicu ketika dijalankan tanpa ada *delay*
Text-Based Emoji	Assign sebuah nama emoji untuk *Character Emoji Segment*. Jalankan sistem. Amati apakah emoji yang ditampilkan sesuai dengan nama yang telah di-assign.	Emoji yang muncul sesuai dengan nama yang sudah ditambahkan sebelumnya
Character Emoji Location	Assign *Scriptable Object (SO)* karakter tertentu untuk *Character Emoji Segment*. Jalankan sistem. Amati apakah emoji muncul di posisi karakter yang telah di-assign.	Emoji muncul tepat di posisi *SO character* yang sesuai dengan konfigurasi
Segment Completion	Jalankan *Character Emoji Segment*. Amati apakah setelah segment selesai, sistem melanjutkan ke segmen berikutnya.	Sistem melanjutkan ke segmen berikutnya setelah segment selesai
Reusable Segment	Gunakan kembali *Character Emoji Segment* pada segmen lain. Assign string emoji dan karakter baru untuk segmen tersebut. Jalankan sistem. Amati apakah *Character Emoji Segment* berfungsi	*Character Emoji Segment* dapat digunakan kembali dan berungsi dengan baik di segmen berikutnya

Fitur	Skenario Pengujian	Indikator Keberhasilan
	dengan baik di segmen baru.	
Timed Disappearance	Atur segment timer dengan nilai tertentu (contoh: 3 detik). Jalankan sistem. Amati apakah emoji menghilang tepat ketika timer mencapai 0.	Emoji menghilang tepat saat *segment timer* mencapai 0, tanpa ada keterlambatan atau penghapusan dini

g. Frame Interaction Segment

Tabel 4.7 Pengujian *Frame Interaction Segment*

Fitur	Skenario Pengujian	Indikator Keberhasilan
Trigger Frame Interaction	Atur *start delay* pada Frame Interaction Segment menjadi 0. Jalankan sistem. Amati apakah Frame Interaction Segment langsung muncul tanpa jeda waktu.	*Frame Interaction Segment* langsung muncul ketika dijalankan tanpa jeda waktu
Pop-up Frame	Assign sebuah *SO Frame Interaction* ke *Frame Interaction Segment.* Jalankan Frame Interaction Segment. Amati apakah *frame interaction* muncul sesuai pengaturan yang telah ditentukan.	*Frame Interaction* dapat muncul *(pop up)* di layar sesuai dengan konfigurasi
Segment Completion	Assign *SO Frame Interaction* dengan durasi tertentu. Jalankan *Frame Interaction Segment.* Amati apakah *Frame Interaction* otomatis selesai ketika frame habis.	*Frame Interaction* dapat melanjutkan ke segment berikutnya ketika semua bingkai sudah habis

Fitur	Skenario Pengujian	Indikator Keberhasilan
	Pastikan segment berikutnya (*next segment*) ter-*trigger* setelah *frame interaction* selesai.	
Reusable Segment	Gunakan kembali *Frame Interaction Segment* di *sequence* yang berbeda. Assign *SO Frame Interaction* yang sama atau berbeda. Jalankan kembali *Frame Interaction Segment.* Amati apakah *Frame Interaction* dapat dijalankan kembali tanpa error.	*Frame Interaction Segment* dapat dijalankan kembali di segment berikutnya tanpa error
Configured Display	Assign sebuah *SO Frame Interaction* di *frame Interaction Segment* Jalankan Frame Interaction Segment. Amati apakah Frame Interaction yang muncul sesuai dengan data yang di-assign di SO.	*Frame Interaction* yang muncul sesuai dengan konfigurasi *SO Frame Interaction* yang telah di-*assign* sebelumnya.

h. Custom Action Segment

Tabel 4.8 Pengujian *Custom Action Segment*

Fitur	Skenario Pengujian	Indikator Keberhasilan
Trigger Custom Action	Siapkan scene dengan *Custom Action Segment* yang memiliki *start delay* diatur ke *0.* Jalankan sistem dan biarkan *Custom Action Segment* ter-*trigger*.	*Custom Action Segment* langsung mulai ketika dijalankan tanpa jeda waktu.

Fitur	Skenario Pengujian	Indikator Keberhasilan
	Amati apakah segmen langsung dimulai tanpa adanya jeda waktu.	
Scripted Behavior	Siapkan scene dengan *Custom Action Segment* yang memiliki *start delay* diatur ke 0. Jalankan sistem dan biarkan *Custom Action Segment* ter-trigger. Amati apakah segmen langsung dimulai tanpa adanya jeda waktu.	*Behaviour Custom Action* berjalan sesuai *script custom action* yang di-*assign*
Segment Completion	*Assign script custom action* dengan durasi eksekusi tertentu ke dalam *Custom Action Segment*. Jalankan sistem dan biarkan *Custom Action Segment* selesai. Amati apakah segment berikutnya (*next segment*) otomatis ter-trigger setelah *Custom Action* selesai.	*Custom Action Segment* selesai dan melanjutkan ke segmen berikutnya
Reusable Action	Gunakan kembali *Custom Action Segment* pada *sequence* yang berbeda di scene yang sama atau berbeda. *Assign script custom action* yang sama atau berbeda ke dalam *Custom Action Segment*. Jalankan kembali sistem dan biarkan *Custom Action Segment* berjalan.	*Custom Action Segment* dapat berjalan kembali tanpa *error* pada segmen berikutnya

i. End Segment

Tabel 4.9 Pengujian *End Segment*

Fitur	Skenario Pengujian	Indikator Keberhasilan
Trigger End Segment	Siapkan *scene* dengan *End Segment* yang memiliki *start delay* diatur ke 0. Jalankan sistem dan biarkan *End Segment* ter-*trigger*. Amati apakah segmen langsung dimulai tanpa adanya jeda waktu.	*End Segment* langsung ter-trigger tanpa jeda waktu
Cutscene Stop	Buat sebuah *sequence* yang memiliki beberapa *segment*, dengan *End Segment* sebagai *segment* terakhir. Jalankan sistem hingga mencapai *End Segment*. Amati apakah *cutscene* berhenti setelah *End Segment* berjalan, tanpa melanjutkan *segment* berikutnya dalam *sequence.*	*Cutscene* berhenti setelah *End Segment* selesai, tanpa memproses segmen berikutnya.
Boolean Output	Siapkan *scene* dengan *End Segment* yang memiliki logika untuk mengirimkan informasi *bool* ke sistem. Atur parameter *bool* di *End Segment* untuk melanjutkan ke *cutscene* lain atau *quest* baru.	Informasi boolean terkirim dengan benar untuk memutuskan melanjutkan ke *cutscene* lain atau *quest* berikutnya.

Fitur	Skenario Pengujian	Indikator Keberhasilan
	Jalankan sistem dan amati apakah informasi *bool* terkirim dengan benar ke sistem penerima.	

4.5 Analisa Hasil Pengujian

Setelah pengujian fitur oleh *game designer*, didapat hasil sebagai berikut

a. *Movement Segment*

Tabel 4.10 Hasil Pengujian *Movement Segment*

Fitur	Status	Keterangan
Trigger Movement Segment		
Movement Character		
Character Animation		
Movement Stop		
Speed Multiplier		
End Condition		
Reusable Segment		

b. *Spawn Character Segment*

Tabel 4.11 Hasil Pengujian *Spawn Character Segment*

Fitur	Status	Keterangan
Trigger Spawn Segment		
Spawn Location		

Fitur	Status	Keterangan
Configured Spawn		
Segment Complete		
Single Spawn		
Reusable Segment		

c. *Dialogue Segment*

Tabel 4.12 Hasil Pengujian *Dialogue Segment*

Fitur	Status	Keterangan
Trigger Dialogue Segment		
Pop-up Dialogue		
Interactive Dialogue		
Predefined Dialogue		
Segment Completion		
Reusable Dialogue		

d. Remove Character Segment

Tabel 4.13 Hasil Pengujian *Character Segment*

Fitur	Status	Keterangan
Trigger Remove Character		
Character Deletion		
Configured Removal		
Character Deletion		
Reusable Removal		

e. Camera Movement Segment

Tabel 4.14 Hasil Pengujian *Camera Movement Segment*

Fitur	Status	Keterangan
Trigger Camera Movement		
Targeted Movement		
Configured Camera		
Initial Position		
Segment Completion		
Reusable Movement		
Precise Stop		

f. Character Emoji Segment

Tabel 4.15 Hasil Pengujian *Character Emoji Segment*

Fitur	Status	Keterangan
Trigger Emoji Segment		
Text-Based Emoji		
Character Emoji Location		
Segment Completion		
Reusable Segment		
Timed Disappearance		

g. Frame Interaction Segment

Tabel 4.16 Hasil Pengujian *Frame Interaction Segment*

Fitur	Status	Keterangan
Trigger Frame Interaction		
Pop-up Frame		

Fitur	Status	Keterangan
Segment Completion		
Reusable Segment		
Configured Display		

h. Custom Action Segment

Tabel 4.17 Hasil Pengujian *Custom Action Segment*

Fitur	Status	Keterangan
Trigger Custom Action		
Scripted Behavior		
Segment Completion		
Reusable Action		

i. End Segment

Tabel 4.18 Hasil Pengujian *End Segment*

Fitur	Status	Keterangan
Trigger End Segment		
Cutscene Stop		
Boolean Output		

[Halaman ini sengaja dikosongkan]

DAFTAR PUSTAKA

Kaźmierczak, R., Skowroński, R., Kowalczyk, C. & Grunwald, G., 2024, 'Creating Interactive Scenes in 3D Educational Games: Using Narrative and Technology to Explore History and Culture', *Applied Sciences (Switzerland)*, 14(11).

Mateas, M. & Stern, A., 2005, *Structuring Content in the Façade Interactive Drama Architecture Proceedings of the First Artificial Intelligence and Interactive Digital Entertainment Conference*.

Pixel Architect, 2023, *Pixel Architect (@PixelArchitect1) / X*.

Říha, D., 2014, *LNCS 8518 - Cutscenes in Computer Games as an Information System*, vol. 8518.

Ruan, X.-Y. & Cho, D.-M., 2014, 'Relation between Game Motivation and Preference to Cutscenes', *Cartoon and Animation Studies*, 36, 573–592.

Unity - Manual: EventSystem, no date, *Unity - Manual: Event System*, *Event System*.

Unity - Manual: ScriptableObject, no date, *Unity - Manual: ScriptableObject*, *Unity - Manual: ScriptableObject*.

Unity - Scripting API: ReadOnlyAttribute, no date, *Unity - Scripting API: ReadOnlyAttribute*.

Zagalo, N., Oliveira, A.P., Cardoso, P. & Vairinhos, M., 2023, 'BEATS & UNITS: A STORY-GAME DESIGN FRAMEWORK', *International Journal of Film and Media Arts*, 8(1), 52–67.