Peningkatan Fleksibilitas dan Kecepatan Pengembangan Permainan Melalui Penerapan Pola Desain Pada Komponen Unity UI

Adam Shidqul Aziz[1*], Andhik Ampuh Yunanto[2], Umi Sa'adah[3], Sabila Jamal[4], Fadilah Fahrul Hardiansyah[5], Desy Intan Permatasari[6], Nailussa'ada[7]

[1,2,3,4,5,6,7] Program Studi Teknik Informatika, Politeknik Elektronika Negeri Surabaya, Surabaya, Jawa Timur
Email: [1*] adam@pens.ac.id, [2] andhik@pens.ac.id, [3] umi@pens.ac.id, [4] sabilajml@gmail.com, [5] fahrul@pens.ac.id, [6] desy@pens.ac.id, [7] nailus@pens.ac.id

(Naskah masuk: 3 Agu 2023, direvisi: 16 Okt 2023, 1 Nov 2023, diterima: 6 Nov 2023)

Abstrak

Industri permainan telah mengalami perkembangan pesat dalam beberapa tahun terakhir, ditandai dengan munculnya berbagai jenis permainan yang menarik dan inovatif. Salah satu tren yang sedang meningkat adalah popularitas *game midcore* yang berhasil menarik perhatian pemain dari berbagai kelompok. Karena daya tariknya, banyak studio *game* beralih ke pengembangan *game midcore* untuk mengeksplorasi potensi pasar yang luas dan mencapai kesuksesan yang lebih besar. Salah satu tantangan utama yang dihadapi dalam pengembangan *game midcore* oleh perusahaan adalah perancangan dan pembuatan komponen antarmuka pengguna (UI) yang efektif. Bagian UI merupakan aspek kritis dalam permainan karena secara langsung mempengaruhi pengalaman bermain dan keterlibatan pemain. Di samping itu, pada saat ini belum ada standar universal atau *framework* yang sepenuhnya mendukung pengembangan UI pada *game midcore*. Kurangnya alat bantu yang efisien dapat menyebabkan penundaan dan peningkatan beban kerja pada tim pengembang, yang akhirnya dapat berdampak pada keseluruhan pengalaman bermain dan kualitas produk. Dalam menghadapi tantangan pengembangan UI pada *game midcore*, penelitian ini menyajikan solusi dengan menerapkan *design pattern* untuk mengembangkan komponen UI *game* yang akan menjadi *framework* pada *Unity Engine*. Hasil validasi menunjukkan bahwa penggunaan *framework* ini menyebabkan efisiensi pada waktu pengembangan dan *interaction cost*. Efisiensi *interaction cost* sebesar 10% hingga 66% dan waktu pengembangan sebesar 28% hingga 91%. Selain itu, dari hasil survei pelanggan menggunakan metode perhitungan *Customer Satisfaction Score* (CSAT), diperoleh tingkat kepuasan pelanggan sebesar 78%. Nilai 78% dalam skala CSAT dianggap sebagai tingkat kepuasan "*excellent response*," yang menyiratkan bahwa menurut pelanggan, *framework* ini telah memiliki performa yang sangat baik.

Kata Kunci: *Unity Framework, Midcore Game, Design Pattern,* Waktu Pengembangan, *Interaction Cost*.

Enhancement of Reusability and Development Speed of Games through the Implementation of Design Patterns in Unity UI Components

Abstract

The gaming industry has experienced rapid development in recent years, marked by the emergence of various types of captivating and innovative games. One of the rising trends is the popularity of midcore games, successfully attracting players from various demographics. Due to their appeal, many game studios have shifted towards developing midcore games to explore vast market potential and achieve greater success. One of the main challenges faced in midcore game development is the effective design and creation of User Interface (UI) components. The UI plays a critical role in the gaming experience, directly influencing player engagement. Moreover, there is currently no universal standard or fully supportive framework for midcore game UI development. The lack of efficient tools can lead to delays and increased workload for the development team, ultimately impacting the overall gaming experience and product quality. To address the challenges of midcore game UI development, this research presents a solution by applying design patterns to develop UI game components that will become part of the Unity

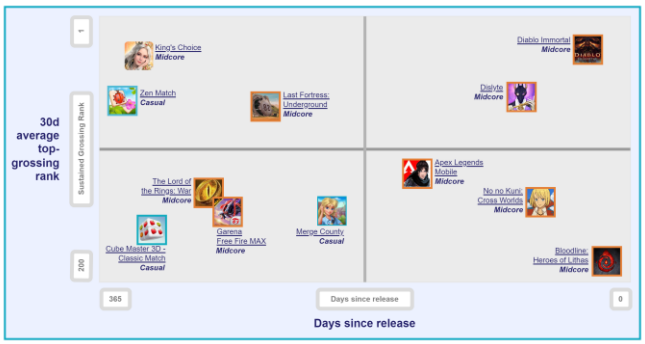
Engine framework. Validation results indicate that the use of this framework improves efficiency in development time and interaction cost. Interaction cost efficiency ranges from 10% to 66%, while development time efficiency ranges from 28% to 91%. Additionally, a survey using the Customer Satisfaction Score (CSAT) method reveals a customer satisfaction rate of 78%. A CSAT score of 78% is considered an "excellent response" in satisfaction, indicating that customers perceive the framework to perform exceptionally well.

Keywords: *Unity Framework, Midcore Game, Design Pattern, Time Development, Interaction Cost.*

I. PENDAHULUAN

Industri permainan telah mengalami perkembangan yang signifikan, sebagaimana terdokumentasikan dalam *Outlook* Pariwisata & Ekonomi Kreatif 2021/2022. Laporan tersebut mencatat bahwa industri permainan telah berkontribusi sebesar 31,25 triliun rupiah terhadap Produk Domestik Bruto (PDB) pada tahun 2021, dengan tingkat pertumbuhan kedua tertinggi sebesar 9,17% [1].

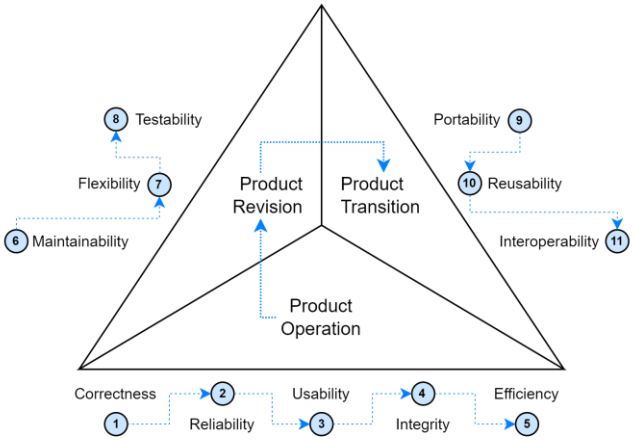
GameRefinery A Liftoff Company sebuah perusahaan yang berfokus dalam penelitian pada pasar *game*, mengungkapkan bahwa sebelumnya pasar permainan *mobile* didominasi oleh permainan *Casual* dan *Hyper Casual* seperti *Candy Crush* dan *Merge County*. Meskipun permainan-permainan ini masih populer, namun kini ada jenis permainan baru yang semakin populer bernama *Midcore* seperti *Diablo Immortal*, *Call of Duty*: *Mobile*, dan *Apex Legends Mobile*. Kepopuleran permainan *Midcore* ini menandakan bahwa pasar permainan *mobile* terus berkembang dan menyediakan pengalaman baru yang sesuai dengan kebutuhan para pemain dari PC dan konsol [2].



Gambar 1. Permainan *Mobile Midcore* dan *Casual* yang Dirilis Dalam Satu Tahun Terakhir dan Tetap Mempertahankan Posisinya di 200 Peringkat Dengan Pendapatan Tertinggi (*iOS*, Amerika Serikat). Source: *GameRefinery SaaS Platform* [2]

Visualisasi pada Gambar 1 menunjukkan bahwa permainan *Midcore* menghasilkan pendapatan yang tinggi dan lebih populer dibandingkan permainan *Casual*. Terdapat lebih banyak permainan *Midcore* yang masuk ke dalam daftar 200 permainan dengan pendapatan tertinggi dibandingkan permainan *Casual*. Dalam sumber tersebut juga dijelaskan bahwa meskipun *midcore* mendominasi, bukan berarti bahwa permainan *Casual* dan *Hyper Casual* tidak populer lagi. Tetapi kepopuleran permainan *midcore* menunjukkan bahwa para pengembang dan penerbit permainan besar dari dunia *game* mulai menganggap pasar *mobile* lebih serius.

Perkembangan industri permainan dengan genre *midcore* yang pesat juga terjadi pada negara Amerika Serikat. Hal ini dibuktikan permainan genre *midcore* mewakili 35% dari total pendapatan permainan di Amerika Serikat untuk *platform iOS* [3]. Pasar permainan *midcore* yang semakin menjanjikan menciptakan persaingan yang kompetitif bagi pada pengembang. Kondisi ini menuntut inovasi dan kreasi para pengembang dalam menciptakan permainan berkualitas dalam rentang waktu yang terbatas.



Gambar 2. *Triangle McCall Quality Model* [4]

McCall Quality Model adalah sebuah kerangka kerja yang digunakan untuk mengukur kualitas perangkat lunak, seperti terlihat pada Gambar 2. Model ini terdiri dari 11 aspek yang mencakup segala aspek yang berkaitan dengan kualitas perangkat lunak, seperti keandalan, efisiensi, pemeliharaan, dan dokumentasi [4]. Dalam pengembangan perangkat lunak, *McCall Quality Model* berperan penting dalam membantu tim pengembang untuk memahami, mengukur, dan meningkatkan kualitas perangkat lunak yang dihasilkan. Dengan menganalisis aspek-aspek kualitas yang terdefinisi dalam model ini, pengembang dapat mengidentifikasi potensi masalah, mengambil langkah-langkah perbaikan, dan

memastikan bahwa produk akhir memenuhi standar kualitas yang ditetapkan.

Berdasarkan kerangka kerja *McCall Quality Model*, aspek yang memiliki dampak signifikan dalam pengembangan perangkat lunak berkualitas tinggi dalam waktu terbatas adalah fleksibilitas dan reusabilitas. Fleksibilitas mengacu pada kemampuan perangkat lunak untuk dengan mudah dan efisien beradaptasi dengan perubahan kebutuhan atau perubahan dalam lingkungan sekitarnya. Sedangkan reusabilitas merujuk pada kemampuan komponen atau modul perangkat lunak supaya dapat digunakan kembali dalam berbagai konteks atau proyek yang berbeda [4]. Penerapan aspek fleksibilitas dan reusabilitas dalam pengembangan perangkat lunak, dengan pendekatan *Clean Architecture* sebagai kerangka arsitektur perangkat lunak. *Clean Architecture* dikenal karena memisahkan komponen-komponen perangkat lunak ke dalam lapisan-lapisan yang berbeda, seperti lapisan presentasi, lapisan bisnis, dan lapisan data [5]. Penerapan aspek fleksibilitas dalam *Clean Architecture* memungkinkan pengembang untuk lebih mudah mengubah atau menambahkan fitur baru tanpa mengganggu bagian-bagian yang sudah ada. Reusabilitas, di sisi lain, diperkuat oleh *Clean Architecture* dengan mempromosikan pembuatan komponen-komponen yang independen dan dapat digunakan kembali. Implementasi fleksibilitas dan reusabilitas melalui pendekatan *clean architecture* dilakukan dengan membuat komponen-komponen yang dirancang menggunakan prinsip-prinsip *design pattern*. Hal ini bertujuan supaya komponen tersebut dapat dengan mudah diintegrasikan ke dalam proyek-proyek pengembangan permainan lainnya. Metode ini memberikan manfaat berupa mengurangi kerja pengembangan yang berulang, meningkatkan efisiensi, dan mempercepat pengembangan perangkat lunak.

Unity Engine dikenal sebagai *development tools* permainan yang familiar dan banyak digunakan oleh pengembang. Alat pengembangan ini memiliki basis bahasa pemrograman berupa C#. Pada *Unity Engine*, komponen UI terdiri dari berbagai elemen seperti tombol (*button*), teks, dan lain-lain. Namun, saat digunakan secara *native*, komponen UI ini memiliki dependensi yang cukup tinggi terhadap komponen-komponen lain, seperti *Script* atau *Game Object* tertentu yang ada dalam proyek.

Stressful work pace merupakan salah satu isu yang sering terjadi dalam pengembangan permainan. Isu ini merujuk pada tekanan dan kecepatan kerja yang tinggi yang sering dialami oleh para profesional dalam industri permainan [6]. Apabila tidak tersolusikan, maka akan berdampak pada kualitas dari permainan akan menurun dan tentu akan menciptakan kerugian bagi industri tersebut.

Penelitian ini dikembangkan sebagai tanggapan terhadap isu *stressful work pace*, dimana durasi waktu pengembangan menjadi konstrain yang perlu disolusikan. Hal ini bertujuan supaya para pengembang dapat menciptakan produk permainan yang berkualitas dengan durasi waktu yang terbatas. Penelitian ini menjawab isu tersebut melalui pendekatan *reusable component* yang diterapkan dalam proses pengembangan permainan. Kondisi ini telah menjadi perbincangan utama di kalangan pengembang, yang secara intens mempertimbangkan berbagai pendekatan dalam pengembangan komponen yang bersifat *reusable* [7] [8] [9].

II. PENELITIAN TERKAIT

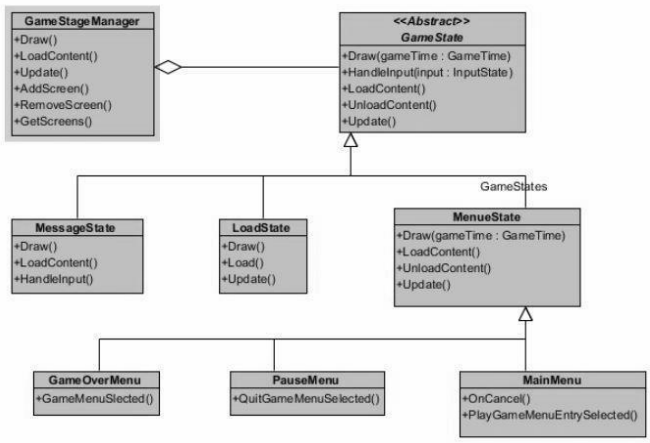
A. *Game Design Patterns - Utilizing Design Patterns in Game Programming*

```
using System.Collections;
using UnityEngine;

public class Singleton : MonoBehaviour {

    public static Singleton uniqueInstance;

    // simple example data to verify that the Singleton operates correctly:
    public int health;


    void Awake () {

        Debug.Log ("Singleton: Awake called.");

        if (uniqueInstance != null) {
            Debug.Log ("There already is a unique istance of this class, destroying this instance.");
            // if an instance of this class already exists, destroy it:
            Destroy (gameObject);
        } else {
            Debug.Log ("No previous instance, making this the unique instance.");
            // there is no instance yet, so assign it:
            uniqueInstance = this;
            // prevent the automatic destruction of the object target when loading a new scene:
            DontDestroyOnLoad (this);
        }
    }
}
```

Gambar 3. Implementasi Kode *Singleton Pattern* Pada Proyek

Penelitian ini menunjukkan eksplorasi konsep *design patterns* dan evaluasi potensinya dalam konteks *game programming* guna meningkatkan keterbacaan kode serta mempermudah modifikasi dan pemeliharaan. Pendekatan dilakukan dengan menerapkan prinsip-prinsip teoritis yang telah diidentifikasi, melalui tahap *refactor* pada proyek permainan di *Unity Engine* yang memiliki kompleksitas tertentu, dengan menggunakan *design patterns* sebagai panduan. Hasil penelitian ini membuktikan peran signifikan *design patterns* dalam meningkatkan efisiensi pengembangan permainan pada aspek maintainabilitas kode yang mudah dipahami dan dapat dijaga kemudahannya [10] seperti terlihat pada Gambar 3.

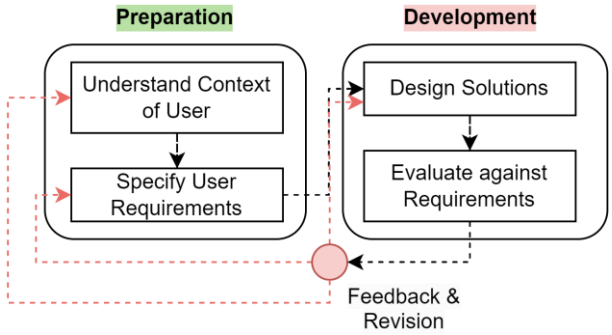
B. *Applying Design Patterns in Game Programming*



Gambar 4. Implementasi *State Pattern* Untuk *Game State Management*

Penelitian ini mengeksplorasi penerapan desain berorientasi objek pada pengembangan *game* menggunakan bahasa pemrograman C# dan platform XNA dengan menggunakan berbagai pola desain. Dalam penelitian ini, penulis menerapkan pola-pola desain struktural, kreasi, dan perilaku untuk mengembangkan elemen *sprite* dalam *game*, serta pola desain *state* untuk mengelola keadaan permainan dan elemen *sprite*. Selain itu, penelitian ini mendiskusikan penggunaan pola desain dalam penanganan komunikasi antara elemen *sprite* dan NPC dengan memanfaatkan pola *observer* dan mediator [11]. Penelitian ini menunjukkan bahwa *object oriented design* menggunakan *design pattern* dapat digunakan dalam menunjang pengembangan permainan seperti terlihat pada Gambar 4.

III. METODE PENELITIAN



Gambar 5. Metode *User Centered Design*

Penelitian ini menerapkan metode *User Centered Design* selama pengembangan dilakukan. Metode UCD dipilih karena proses pengembangan yang dilakukan berfokus pada kebutuhan pengguna [12]. Dalam metode UCD ini, peneliti harus secara aktif berinteraksi dengan pengguna dan mengelola data yang diperoleh guna menentukan kebutuhan pengguna. Metode ini melibatkan serangkaian iterasi, dengan evaluasi yang terus-menerus dilakukan dalam setiap langkah prosesnya. Berdasarkan Gambar 5, metode UCD terbagi atas 4 bagian, yaitu:

- *Understand Context of Use*
- *Specify User Requirement*
- *Design Solution*
- *Evaluate Against Requirements*

A. *Understand Context of Use*

Metode pengumpulan data kebutuhan pengguna dalam penelitian ini menggunakan wawancara non-struktural atau wawancara tidak terstruktur. Dengan pendekatan ini, wawancara memberikan kebebasan yang lebih besar untuk mengajukan pertanyaan kepada pengguna, tanpa harus mengikuti format atau struktur tertentu. Pengguna dalam penelitian adalah para profesional dalam industri permainan.

B. *Specify User Requirement*

Berdasarkan tahapan wawancara yang telah dilakukan, maka didapatkan kebutuhan pengguna terkait komponen apa saja yang perlu disediakan dalam proses pengembangan permainan. Secara umum terdapat empat komponen utama yang dibutuhkan pengguna, yaitu:

- Komponen *Button*
- Komponen *Canvas*
- Komponen *Text*
- Komponen *ScrollView*

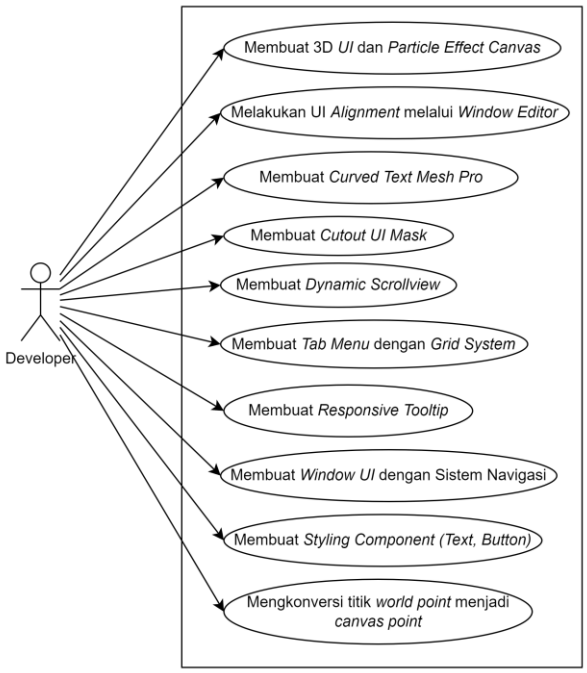
C. *Design Solutions*

Kebutuhan pengguna yang didapatkan, kemudian didefinisikan solusinya dalam bentuk komponen-komponen yang *reusable*. Pada tahapan ini terdapat tiga ruang lingkup besar yaitu:

- *Use Case Diagram*
- Paket *Framework*
- Implementasi beberapa jenis *Design Pattern*, yaitu *builder pattern*, *decorator pattern*, dan *observer pattern*.

C.1. *Use Case Diagram*

Use Case digunakan untuk memberikan visualisasi terhadap fungsi-fungsi yang akan dikembangkan.



Gambar 6. *Use Case Diagram*

Gambar 6 menunjukkan bahwa *use case diagram* hanyak memiliki satu *actor* yaitu *developer*. Terdapat beberapa fungsi yang terdapat pada *use case* yang secara umum dapat melakukan hal-hal berikut:

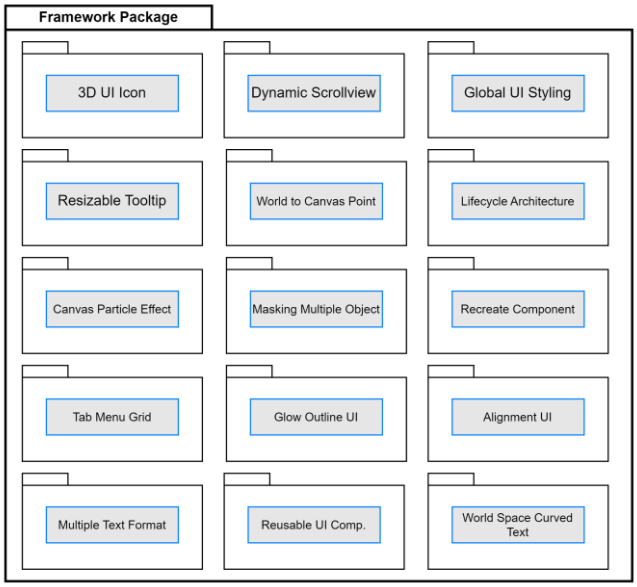
- Instansiasi Komponen *Button*:
  Developer dapat membuat instan komponen *button* yang ada pada *framework*.
- Instansiasi Komponen *Canvas*:

Developer dapat membuat instan komponen *canvas* yang ada pada *framework*.

- Instansiasi Komponen *Text*:
  Developer dapat membuat instan komponen *text* yang ada pada *framework*.
- Instansiasi Komponen *Scrollview*:
  Developer dapat membuat instan komponen *scrollview* yang ada pada *framework*.

C.2. Paket *Framework*

Dari komponen yang telah diuraikan pada gambar 6, akan diorganisir ke dalam beberapa paket dalam kerangka kerja seperti pada Gambar 7. Dengan demikian, objek-objek ini tidak hanya terbatas pada elemen UI *native*, melainkan juga menyertakan berbagai fitur khusus yang lebih lengkap.



Gambar 7. *Framework Package*

Ilustrasi pada Gambar 7 menggambarkan rangkaian paket yang terintegrasi dalam kerangka kerja ini. Setiap paket mengandung skrip-skrip dan elemen UI yang telah siap untuk diterapkan. Lebih jauh mengenai komponen dalam setiap paket ini, informasinya dapat ditemukan dalam Tabel 1.

Tabel 1. Komponen Dalam *Package*

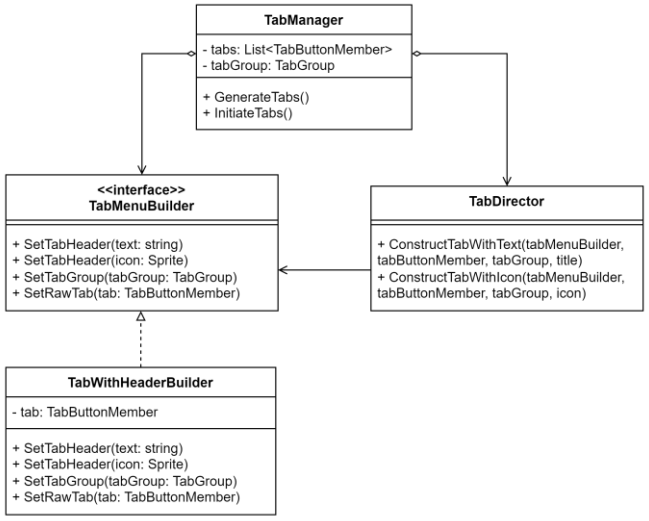
No.	*Package*	Rincian Komponen
1.	*3D UI Icon*	- *Prefab Objek Canvas*
2.	*Auto-resizable Tooltip*	- *Prefab Objek Tooltip* - *Script Tooltip Trigger*
3.	*Particle Effect UI pada Canvas*	- *Prefab Objek Canvas Particle Effect*
4.	*UI Menu Tab dengan Grid System*	- *Prefab Objek Menu Tab* - *Script Tab Group* - *Script Tab Button* - *Script Flexible Grid Layout*
5.	*World Space to Canvas Converter*	- *Script World To Canvas Converter*
6.	*World Space Curve Text*	- *Script TMP on A Circle*
7.	*Dynamic Scrollview*	- *Script Scrollview Content Fitter*
8.	*Alignment Multiple UI Object*	- *Editor Window Alignment*
9.	*Flexible UI Styling*	- *Script UI Style Manager* - *Scriptable Object Button & Text*
10.	*UI Lifecycle*	- *Script Event Listener* - *Script Event Manager* - *Script Content*

Package ini menyajikan komponen-komponen dengan tambahan fungsi yang belum tersedia secara bawaan di dalam *Unity Engine*. Keberadaan komponen-komponen ini sangat penting bagi para pemangku kepentingan dalam mengembangkan *game midcore*. Komponen-komponen ini dirancang untuk memastikan bahwa pemangku kepentingan tidak perlu merancang ulang komponen serupa dari awal, melainkan dapat langsung memanfaatkan dan mengubah komponen yang sudah ada.

C.3. Implementasi beberapa jenis *Design Pattern*

C.3.1. Implementasi *Builder Pattern*

Dalam konteks ini, terlihat gambaran implementasi *class* diagram salah satu pola yang dianggap relevan, yaitu *Builder Pattern*. Untuk merangkai pola ini, tahap pertama melibatkan pembuatan antarmuka (*interface*) sebagai langkah awal, yang kemudian menentukan fungsi-fungsi yang akan dimiliki oleh *builder*. Pada tahap berikutnya, saya merancang *class Director* yang bertindak sebagai penghubung *builder* saat merakit komponen yang dibutuhkan. Akhirnya, *class concrete builder* dirancang sesuai spesifikasi. Seiring perkembangan, untuk memperlihatkan sifat ekstensibilitasnya, *developer* berpeluang untuk membuat *builder* lain yang sesuai dengan kebutuhan, dengan melakukan *ekstend* antarmuka yang telah disediakan.

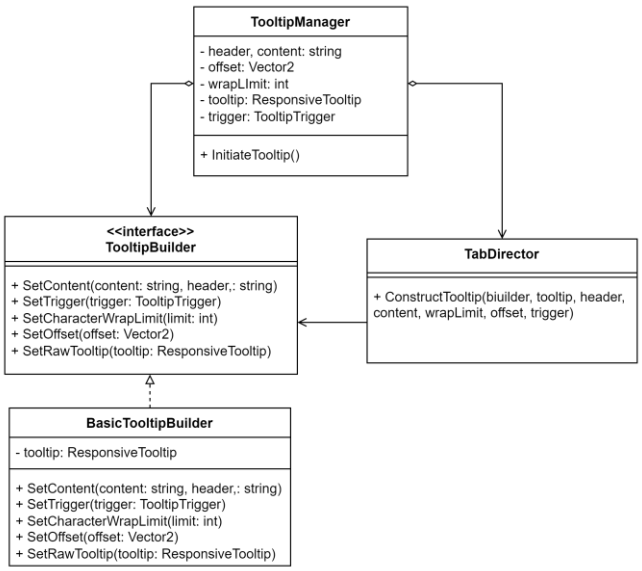


Gambar 8. Implementasi *Builder Pattern*

Gambar 8 mengilustrasikan implementasi *Builder Pattern* untuk fitur *Tab* Menu. *TabMenuBuilder* mewakili antarmuka

yang bakal diimplementasikan oleh *concrete builder*, *TabWithHeaderBuilder* mempresentasikan *class builder* untuk merancang menu *tab* dengan *header*. *TabDirector* berfungsi sebagai *class director* yang diinisiasi untuk mengarahkan proses pembangunan *tab*, sementara *TabManager* bertindak sebagai *class* tempat pembangunan *tab* dilakukan.

Pada ilustrasi dalam Gambar 9 dilakukan implementasi *Builder Pattern* yang disesuaikan untuk fitur *Tooltip*. Dalam membangun pola ini, tahapan dimulai dengan penyusunan antarmuka (*interface*) yang akan diimplementasikan oleh *concrete builder*, yang direpresentasikan oleh *TooltipBuilder*. Lanjut ke tahap berikutnya, dirancang *class builder*, dalam hal ini *BasicTooltipBuilder*, yang bertugas membangun *tooltip* responsif sesuai spesifikasi. Kemudian, ada *TooltipDirector* sebagai *class director* yang memandu dan mengarahkan proses konstruksi *tooltip*. Sementara itu, peran *TooltipManager* terletak sebagai *class* yang menampung proses pembangunan *tooltip*. Semua komponen ini bekerja bersama untuk menghasilkan fitur *Tooltip* dengan desain dan fungsionalitas yang diinginkan.

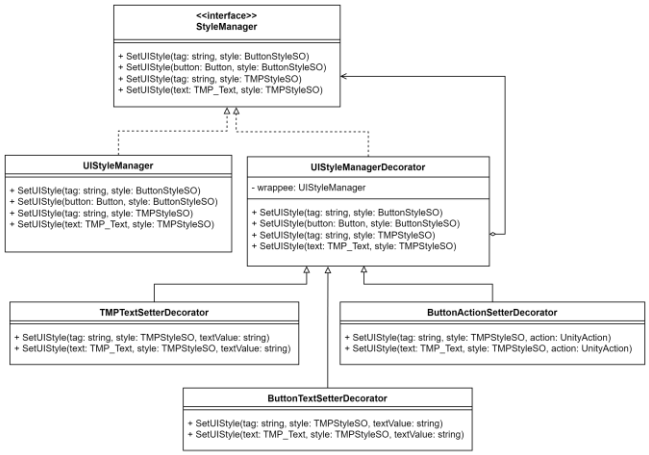


Gambar 9. Implementasi *Builder Pattern*

C.3.2. Implementasi *Decorator Pattern*

Berikut gambaran mengenai implementasi *class diagram* dari *pattern* selanjutnya, yaitu *Decorator Pattern*. Untuk membuat *pattern* ini, langkah awalnya adalah dengan membuat *interface* komponen yang nantinya akan diberi dekorator. Dalam kasus ini, komponen tersebut adalah *style manager*. Selanjutnya, saya buat *concrete class* yang mengimplementasikan *interface* tersebut. *Class* ini sudah memiliki fungsi-fungsi dasar yang bisa dilakukan oleh komponen ini, yaitu mengganti *style*. Berikutnya, saya buat *base class* dekorator untuk membungkus komponen dasar dan menambahkan kemampuan tambahan pada fungsi yang sudah ada. *Class* inilah yang bisa di-*extend* untuk membuat dekorator - dekorator baru sesuai kebutuhan. Sebagai contoh, dalam kasus ini saya buat komponen untuk menambahkan fungsi mengatur teks dan *action button*.

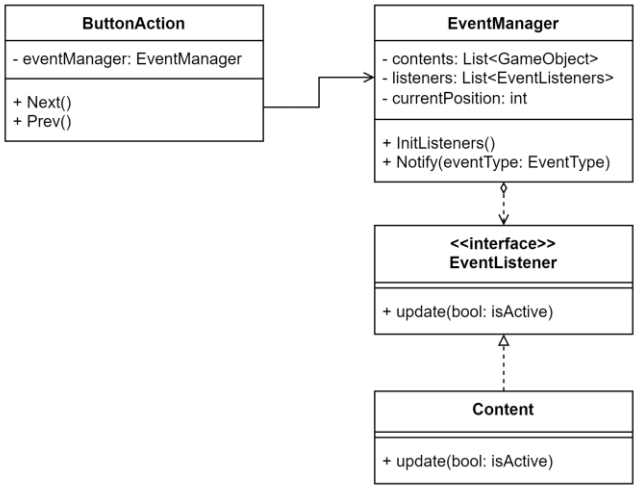
Pada Gambar 10 merupakan implementasi *decorator pattern* pada fitur UI *Styling*. *StyleManager* merupakan *interface* yang akan diimplementasikan oleh *class concrete* dan *decorator*-nya. *UIStyleManager* merupakan *Concrete class* dari *style manager*, dan *UIStylrManagerDecorator* merupakan *base decorator* dari *style manager* tersebut. Tiga *class* di bawahnya, *TMPTextSetterDecorator*, *ButtonTextSetterDecorator*, dan *ButtonActionDecorator* merupakan *child class decorator* dengan fungsi tambahan masing-masing.



Gambar 10. Implementasi *Decorator Pattern*

C.3.3. Implementasi *Observer Pattern*

Berikut gambaran mengenai implementasi *class diagram* dari salah satu *pattern* selanjutnya, yaitu *Observer Pattern*. Untuk membuat *pattern* ini, langkah awalnya adalah dengan membuat *interface* untuk objek *subscriber*-nya terlebih dahulu. Berikutnya, saya buat implementasi *konkrit* untuk *subscriber* yang mengimplementasikan *interface* yang sudah dibuat tadi. Lalu, saya buat *class* untuk *publisher* yang menyimpan *instance* dari objek-objek *subscriber*.



Gambar 11. Implementasi *Observer Pattern*

Pada Gambar 11 merupakan implementasi *observer pattern* untuk fitur UI *Lifecycle*. *EventListener* merupakan *interface* dari *observer* pada *observer pattern*. *Content* adalah implementasi konkret dari *EventListener* yang bertindak sebagai *observer*. *EventManager* adalah *class* yang berperan untuk mengawasi *event* yang terjadi pada *content* sebagai *publisher*, sedangkan *ButtonAction* adalah *class* yang menggunakan bantuan *EventManager* untuk melakukan *tracking* tiap kali terjadi *event button click*.

D. *Evaluate Againts Requirements*

Setelah mendefinisikan dan pengembangan terhadap kebutuhan pengguna, maka dilakukan implementasi oleh pengguna dalam proses pengembangan permainan. Pengembangan ini dilakukan evaluasi secara berkala untuk mendapatkan umpan balik sehingga diketahui ketercapaian dari tujuan dan kebutuhan pengguna.

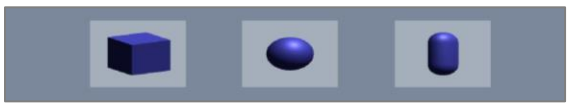
Dalam penelitian ini, para pengguna berpartisipasi aktif dengan peneliti dalam setiap siklus evaluasi melalui teknik uji coba sistem dan wawancara tatap muka. Evaluasi hasil yang diperoleh dari pengguna akan diaplikasikan pada sistem untuk tujuan perbaikan dan peningkatan performa sistem.

IV. HASIL DAN PEMBAHASAN

Pada bab ini akan ditunjukkan contoh dari penggunaan setiap komponen yang telah dikembangkan.

A. 3D *UI Icons*

3D *UI Icons* merupakan komponen yang memungkinkan penempatan objek 3D pada *Unity* UI, mengatasi pembatasan sebelumnya dimana komponen UI hanya dapat berisi aset 2D.



Gambar 12. 3D *UI Icons*

Komponen ini tidak memiliki script, melainkan hanya *prefab object*, yaitu objek *template* yang dapat langsung digunakan kembali tanpa perlu pengaturan ulang. Dengan bantuan komponen ini, developer dapat langsung membuat *canvas* yang dapat menampung objek - objek 3D pada tampilan UI seperti terlihat pada Gambar 12.

B. *Auto-resizable Tooltip*

Auto-resizable Tooltip merupakan komponen *tooltip* yang dapat menyesuaikan ukurannya sesuai dengan isi teks yang dimasukkan.

Gambar 13. *Responsive Tooltip*

Gambar 14. *Responsive Tooltip Long*

Pada gambar 13, 14 terlihat bahwa saat *mouse hover* pada objek target, maka akan muncul *tooltip* berisi teks yang diinginkan. Jika teks lebih panjang, pakai *tooltip* akan menyesuaikan ukuran lebarnya tanpa memperkecil ukuran teks. Jika teks lebih panjang dari lebar *tooltip* maksimal yang diinginkan, maka teks akan diperkecil menyesuaikan ukuran lebar maksimal *tooltip*. Komponen ini berupa *script* yang nantinya ditautkan pada objek target *hover* dan *prefab* dari *tooltip*-nya sendiri yang tampilannya dapat diubah-ubah. Solusi ini penulis dapatkan dengan bantuan dari kanal *youtube Game Dev Guide*. Komponen ini menerapkan *BuilderPattern*. *Pattern* ini memudahkan developer dalam membangun *tooltip* dengan *style* yang berbeda-beda. Berikut adalah contoh penggunaan *tooltip* pada *game minecraft*.

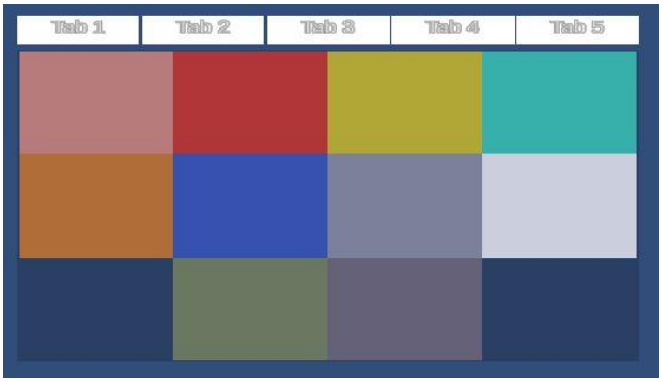
C. *Particle Effect UI* pada *Canvas*

Gambar 15. *UI Particle Effect*

Particle Effect UI pada *Canvas* merupakan komponen berupa efek partikel yang ditampilkan pada *UI Canvas*, atau layar *UI* pada *Unity*. Pada Gambar 15, titik-titik putih pada layar merupakan efek partikel dari *Unity*, namun normalnya partikel *Unity* hanya bisa ditampilkan pada *World Space* saja atau *layer* penempatan *game object*-nya. Komponen ini berupa *prefab canvas* yang dapat menampilkan efek partikel pada layer UI. Dengan bantuan komponen ini, *developer* dapat langsung membuat *canvas* yang dapat menampung efek partikel pada tampilan UI.

D. *Template UI Menu Tab* dengan *Grid System*

Template UI Menu Tab dengan *Grid System* merupakan komponen menu *tab* dengan panel yang memiliki *grid system* sehingga tatanan objek UI di dalamnya lebih rapi.

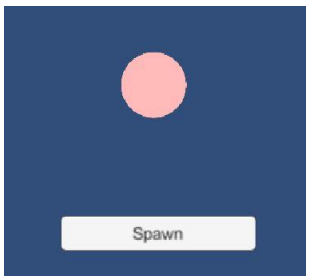


Gambar 16. Menu *Tab Grid Layout*

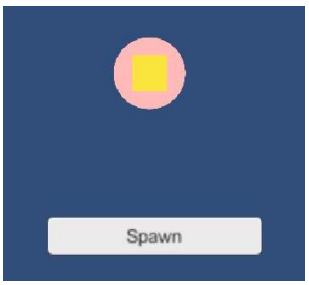
Pada Gambar 16 terlihat ada beberapa *tab* yang dapat diatur warnanya saat *selected*, *hover*, maupun *inactive*. Pada *tab* 1 terdapat panel 1 dengan *child object* nya yang ukuran dan posisinya seragam dalam sistem *grid*. Jika tombol *tab* di klik, maka menu akan menavigasi halaman konten di bawahnya sesuai dengan halaman-halaman untuk masing-masing *tab*. Komponen ini berupa *script* untuk *tab group*, *tab button*, dan *panel group* yang dapat ditautkan pada saat membuat objek menu *tab*. Solusi ini penulis dapatkan dengan bantuan dari kanal *youtube Game Dev Guide*. Komponen ini menerapkan *BuilderPattern*. *Pattern* ini membuat *developer* lebih mudah membangun *tab menu* dengan tipe UI yang berbeda-beda.

E. *World Space to Canvas Converter*

World Space to Canvas Converter merupakan komponen untuk menginstansiasi objek UI pada posisi *World Space*. Dalam *unity*, lokasi atau titik poin untuk objek pada *world space* dan *canvas space* dibedakan. Posisi *world space* sendiri merupakan posisi dimana objek - objek yang ada dalam dunia *game* berada, sedangkan *canvas space* merupakan posisi dimana objek - objek UI yang bukan bagian dari dunia game berada. Pada umumnya, saat kita menginstansiasi objek UI melalui kode di *unity* maka objek tersebut akan ter-instansiasi pada posisi terhadap *Canvas*. Komponen ini mengkonversi posisi *World Space* yang diinputkan agar saat meng instansiasi pada posisi *Canvas* letaknya sesuai dengan posisi *World Space*-nya.



Gambar 17. Contoh Objek Pada *World Space*



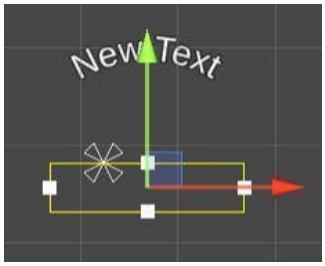
Gambar 18. Instansiasi Objek UI

Pada Gambar 17 terdapat objek lingkaran yang diletakkan pada *world space*. Saat ditekan *spawn*, maka akan menginstansiasi objek UI pada *canvas space* seperti berikut.

Objek kotak warna kuning pada Gambar 18 merupakan UI *Image* yang diinstansiasi di *canvas space* sesuai dengan posisi dari lingkaran pada *world space*. Komponen ini berupa *script static* yang dapat langsung dipanggil.

F. *World Space Curve Text*

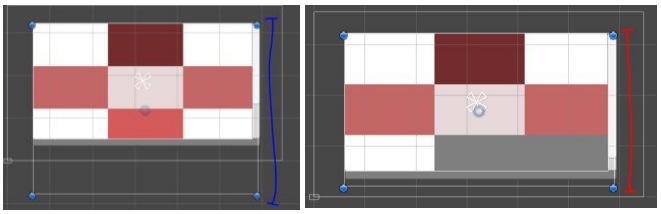
World Space Curve Text merupakan komponen teks yang bisa membentuk kurva. Pada *umumnya* dalam *Unity* teks hanya bisa berbentuk lurus. Komponen ini berupa *script* yang ditautkan pada teks yang ingin dibuat menjadi kurva. Bisa dilihat pada Gambar 19 bahwa teks membentuk garis lengkung yang dapat diatur radiusnya.



Gambar 19. *Pivot Point Text*

G. *Dynamic Scrollview*

Dynamic Scrollview merupakan komponen *scrollview* yang dapat menyesuaikan ukuran *viewport content*-nya sesuai dengan objek yang ditambahkan.



(a) Panjang (b) Pendek

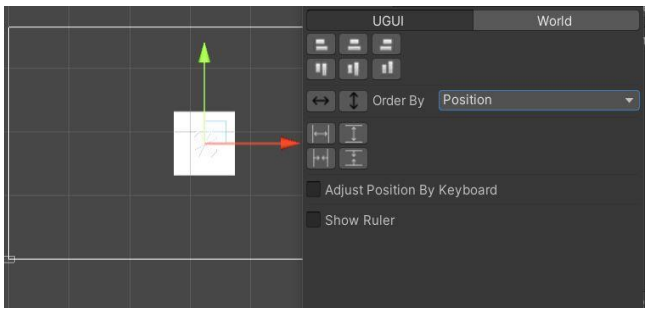
Gambar 20. Implementasi *Dynamic Scrollview*

Pada Gambar 20(a) terdapat banyak objek kotak-kotak yang terdapat di dalam *viewport content scrollview* hingga melebihi ukuran *scrollview*, sehingga ada objek yang tidak terlihat dan baru akan terlihat saat di *scroll*. Ukuran *container*-nya menyesuaikan agar bisa menampung seluruh objek tanpa mengubah ukurannya.

Pada Gambar 20(b), jumlah objek dalam *container* dikurangi, sehingga panjang *viewport content* dari *scrollview* turut berkurang juga. Komponen ini berupa *script* yang ditautkan pada *scrollview* yang dibuat. Komponen ini menerapkan *BuilderPattern*.

H. *Alignment Multiple UI Object*

Alignment Multiple UI Object merupakan komponen untuk mengatur *alignment* dari objek-objek UI yang terdapat pada layar (Gambar 21).



Gambar 21. *UI Alignment*

Normalnya, *Unity* tidak menyediakan fitur GUI untuk *alignment* objek UI, sehingga jika kita ingin mengatur *alignment* UI kita harus memindahkan posisi masing - masing objek secara manual. Komponen ini berupa *window* yang

dapat mengatur letak *alignment* objek yang diinginkan dengan cara memilih objek tersebut lalu memilih *alignment* yang diinginkan. Solusi ini penulis dapatkan dengan bantuan dari repository *open source Litefeel*.

I. *Flexible UI Styling*

Flexible UI Styling merupakan komponen berbentuk *script* untuk melakukan *styling* pada beberapa objek UI sekaligus. Komponen ini akan memilih objek UI dengan *tag* tertentu yang nantinya tampilannya akan diubah sesuai pengaturan yang diinginkan yang diletakkan pada satu objek *style* sehingga kita tidak harus mengubah *styling* objek UI satu per satu dan bisa melakukannya sekaligus.

Gambar 22. *Flexible UI Styling*

Pada Gambar 22, *button* yang tengah memiliki *tag* yang berbeda dengan *button* lainnya. Begitu juga untuk teksnya, oleh karena itu tampilannya berbeda. Dalam fitur ini, terdapat *script decorator* terpisah, dimana selain mengubah *style* dari komponen UI, kita juga bisa merubah propertinya, misal merubah teks untuk komponen *TextMeshPro* dan merubah teks pada *button*, atau menambahkan fungsi *onclick* pada *button*. Komponen ini menerapkan *Decorator Pattern*. Pattern ini mempermudah *developer* untuk menambahkan fungsi tertentu pada objek yang diinginkan, seperti mengubah teks atau menambahkan *event action*.

J. *UI Lifecycle*

UI Lifecycle Komponen ini merupakan komponen yang bertindak seperti *manager* dimana pada suatu *window* terdapat komponen - komponen penyusun lainnya, berikut dengan tombol navigasi *next* dan *previous* (Gambar 23).

Gambar 23. *UI Lifecycle*

Komponen ini terdiri dari dua *script* penyusun utama, yaitu *EventManager* yang ditautkan pada *parent object* atau *container* terluar dari konten, dan *EventListener* yang ditautkan pada konten. Komponen ini membantu *developer* untuk lebih mudah membuat *logic* navigasi antar halaman. Komponen ini menerapkan *Observer Pattern*. *Pattern* ini mempermudah developer untuk mengatur *event* yang akan dilakukan saat terjadi perubahan pada objek yang diawasi.

V. EVALUASI DAN VALIDASI

Proses validasi dilakukan dengan cara melakukan uji coba pada *framework* yang telah dibuat dan memberikan perbandingan pada *interaction cost (IC)* dan waktu pengerjaan yang dibutuhkan dengan tidak menggunakan *framework* atau *vanilla*. Validasi ini akan memberikan hasil efisiensi dalam bentuk persentase pada dua parameter yang akan diuji. Formula yang digunakan dapat dilihat pada formula (1).

$$\frac{\Delta Nilai}{Nilai\ Awal} \times 100\% \quad (1)$$

Dimana $\Delta Nilai = Nilai\ Akhir - Nilai\ Awal$

Hasil percobaan tersebut adalah sebagaimana dijelaskan pada Tabel 2.

Tabel 2. Hasil Uji Coba

No.	Komponen	Dengan *Framework*		Tanpa Framework		*Efficiency (%)*	
		IC	Waktu	*IC*	Waktu	*IC*	Waktu
1.	*3D UI Icon*	4	00:31	8	01:22	50%	54%
2.	*Auto-resizable Tooltip*	4	00:52	5	07:55	20%	89%
3.	*Particle Effect UI pada Canvas*	2	00:22	6	01:11	66%	69%
4.	*UI Menu Tab* dengan *Grid System*	9	04:55	10	18:22	10%	73%
5.	*World Space to Canvas Converter*	5	01:10	6	2:33	16%	54%
6.	*World Space Curve Text*	4	00:36	4	07:05	0%	91%
7.	*Dynamic Scrollview*	7	01:27	9	15:07	22%	90%
8.	*Alignment Multiple UI Object*	4	00:41	3	01:07	-25%	38%
9.	*Flexible UI Styling*	6	02:17	7	01:01	14%	-55%

Selain komponen-komponen pada Tabel 2, terdapat juga komponen yang sudah dimodifikasi dengan *penggunaan design pattern*. Hasil percobaan untuk komponen-komponen tersebut adalah sebagaimana dijelaskan pada Tabel 3.

Tabel 3. Hasil Uji Coba Dengan *Pattern*

No.	Komponen	Dengan *Framework*		Tanpa Framework		*Efficiency (%)*	
		IC	Waktu	*IC*	Waktu	*IC*	Waktu
1.	*UI Menu Tab* dengan *Grid System*	10	05:45	12	20:07	17%	73%
2.	*Auto-resizable Tooltip*	5	01:22	8	10:05	38%	89%

3.	*Flexible UI Styling*	7	03:50	9	03:07	33%	-44%
4.	*UI Lifecycle*	6	04:25	6	07:02	0%	28%

Percobaan di atas dilakukan dengan pemahaman penuh akan kegunaan *framework* saat menggunakan *framework*, dan pemahaman penuh atas algoritma yang digunakan saat tidak menggunakan *framework*. Proses validasi kepada pengguna dilakukan dengan melemparkan pertanyaan yang bersifat kuantitatif dengan 1-10. Adapun hasil dari survei kuantitatif dapat dilihat pada Tabel 4.

Tabel 4. Nilai Skala Kepuasan

No.	Pertanyaan	Jawaban			Total	Nilai Max
		Anggakara Hendra	Murpy Nugraha	Darmawan S		
1	Seberapa mudah penggunaan *framework* ini ?	6	9	9	**24**	30
2	seberapa besar *framework* ini dapat membantu meningkatkan kecepatan *development* ?	8	7	9	**24**	30
3	seberapa besar relevansi *framework* ini untuk membantu *development* UI pada *midcore game* ?	10	8	8	**26**	30
4	seberapa besar tingkat *maintainability* dari komponen - komponen dalam *framework* ini ?	7	7	7	**21**	30
5	seberapa besar tingkat *reusability* dari komponen - komponen dalam *framework* ini ?	7	7	9	**23**	30
				Total	**118**	150

Sehingga nilai CSAT yang didapatkan adalah:

$$\frac{118}{150} \times 100\% = 78\%$$

Dari standar CSAT, 25% - 50% dinilai sebagai *good response*, atau nilai yang cukup baik, selebihnya dianggap sebagai *excellent response rate*, yaitu performa yang sangat baik [14]. Survei menunjukkan nilai CSAT *framework* ini adalah 78%, sehingga bisa disimpulkan bahwa menurut *customer*, *framework* ini sudah memiliki performa yang sangat baik.

Selain melakukan uji coba pada setiap komponen, uji coba juga dilakukan dengan mengembangkan *prototype* dengan dan tanpa *framework*. Nilai efisiensi didapatkan dengan menggunakan formula (1).

Tabel 5. Perbandingan Kecepatan *Develop Prototype*

No	Nama	Dengan *Framework*	Tanpa *Framework*	Efisiensi (%)
1.	Anggakara Hendra	02:42:00	03:18:39	77%
2.	Murpy Satria Nugraha	01:01:29	01:35:47	35%
3.	Darmawan S	01:18:44	01:19:25	0,8%

Pada Tabel 5 dapat dilihat bahwa dari seluruh hasil uji coba *user*, terdapat peningkatan dari yang paling signifikan, yaitu 77%, hingga yang kurang signifikan, yaitu 0.8%, sehingga dapat disimpulkan bahwa *framework* ini sudah memenuhi tujuan awal untuk meningkatkan kecepatan *development*.

V. KESIMPULAN

Setelah melakukan pengembangan *design pattern* menggunakan metode UCD pada komponen-komponen penyusun *framework* didapatkan simpulan hasil sebagai berikut:

- Peningkatan Efisiensi aspek fleksibilitas yang diukur melalui waktu pengembangan dan didapatkan penghematan durasi pengembangan 28% - 91%.
- Peningkatan Efisiensi aspek reusabilitas yang diukur melalui parameter *interaction cost* dan didapatkan penghematan iteraksi 10% - 66%.

Dari simpulan hasil di atas, dapat disimpulkan bahwa *design pattern* terbukti dapat menjadi jawaban dari isu *stressful work pace*. Sehingga para pengembang dapat menghasilkan produk permainan yang berkualitas dalam durasi waktu yang terbatas.

REFERENSI

[1] Kemenparekraf/Baparekraf RI, "https://kemenparekraf.go.id/," Kementerian Pariwisata dan Ekonomi Kreatif Republik Indonesia, [Online]. Available: https://kemenparekraf.go.id/en/articles/development-of-the-game-industry-in-indonesia-achieves-international-award. [Accessed 10 Oktober 2023].

[2] GameRefinery Team, "The Rise of Midcore Mobile Games Snapshot Report: July 2022," GameRefinery, 7 Juli 2022. [Online]. Available: https://www.gamerefinery.com/the-rise-of-midcore-mobile-games-snapshot-report-july-2022/. [Accessed 1 Agustus 2023].

[3] R. Tang, "liftoff," [Online]. Available: https://liftoff.io/blog/2023-midcore-gaming-apps-report-announcement/. [Accessed 31 10 2023].

[4] J. McCall, P. Ricards and G. Walters, "Factors in software quality: concept and definitions of software quality," *General Electric Company*, 1977.

[5] R. C. Martin, Clean Architecture: A Craftsman's Guide to Software Structure and Design, Pearson; 1st edition , 2017.

[6] C. M. Kanode and H. M. Haddad, "Software Engineering Challenges in Game Development," in *Sixth International Conference on Information Technology: New Generations*, Kennesaw, 2009.

[7] K. Karisson, "Reusing GameObjects in Unity3D," 20 Maret 2017. [Online]. Available: https://medium.com/@utterbbq/reusing-gameobjects-in-unity3d-468e0317024f. [Accessed 10 15 2023].

[8] J. West, "Revisiting the Power of Interfaces in C# for Unity: Tips and Techniques for Creating Reusable Components," 14 Desember 2022. [Online]. Available: https://valdarixgames.medium.com/revisiting-the-power-of-interfaces-in-c-for-unity-tips-and-techniques-for-creating-reusable-b4775c12e3ab. [Accessed 10 Oktober 2023].

[9] Learn to Create Games, "Creating maintainable and reusable code," 14 Februari 2018. [Online]. Available: https://learntocreategames.com/optimizing-your-code-2/. [Accessed 15 Oktober 2023].

[10] A. Rautakopra, "Game Design Patterns : Utilizing Design Patterns in Game Programming," 2018.

[11] J. Qu, Y. Song and Y. Wei, "Applying Design Patterns in Game Programming," *17th IEEE.ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, 2013.

[12] A. Chammas, . M. Quaresma and C. Mont'Alvão, "A Closer Look on the User Centred Design," *Procedia Manufacturing*, vol. 3, pp. 5397-5404, 2015.