Pemanfaatan Data Oriented Technology Stack dalam Game Engine Unity dan Teknologi Klasik Unity untuk Membuat Permainan Danmaku

Tantyo Nurwahyu T., Med Irzal, Lipur Sugiyanta

Prodi Ilmu Komputer, Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Jakarta, Jakarta Timur, Indonesia

Email: trippingballs@duck.com, medirzal@unj.ac.id, lipurs@unj.ac.id

Abstract

Video games are relatively resource-intensive. Unlike most common apps, they require additional quality of their resources such as bigger RAM and faster processing speed due to the nature of unpredictability. Even now, one could say that despite the advancement in hardware technologies; a number of video games still demand better resources. To aid this issue, Unity Technologies has introduced a new technology called Data Oriented Technology Stack for their game engine; or DOTS. The DOTS technology is based on the Entity Component System architecture which is an adaptation of the data-oriented paradigm for engineering a software program or for a solution. But as of the time of writing this, the technology isn't ready yet and considered unstable; so a game developer that wishes to use DOTS still need to use the classic technology of the engine. To display the usability and a degree of potential from Unity's DOTS technology, this thesis will present a danmaku game development using both DOTS and the classic MonoBehaviour technology.

Keywords: *Data Oriented Technology Stack, DOTS, Danmaku, Game, Unity, MonoBehaviour, Entity Component System, Data-Oriented Paradigm*

Abstrak

Video games atau permainan membutuhkan sumber daya yang relatif berat. Berbeda dengan aplikasi lain, permainan-permainan membutuhkan kualitas tambahan pada sumber daya yang digunakan seperti pada RAM yang lebih besar dan kecepatan pengolahan data dikarenakan permainan bisa dengan cepat menjadi susah diperkirakan kebutuhannya. Bahkan masih dapat dikatakan bahwa kemajuan teknologi *hardware* masih kurang untuk permainan sekarang. Untuk mengatasi permasalahan tersebut, Unity Technologies mempersembahkan sebuah teknologi baru bernama Data Oriented Technology Stack untuk *game engine* mereka. Teknologi DOTS didasarkan dengan arsitektur Entity Component System yang merupakan sebuah adaptasi dari paradigma berbasis data sebagai solusi pembuatan sebuah program *software*. Pada saat ini, teknologi DOTS belum siap and masih dianggap tidak stabil; sehingga, seorang *game developer* yang ingin menggunakan DOTS masih perlu menggunakan teknologi klasik *game engine* Unity. Untuk menunjukkan kegunaan dan potensi DOTS, skripsi ini akan menunjukkan pembuatan sebuah permainan *danmaku* menggunakan DOTS dan teknologi klasik MonoBehaviour.

Kata-kata kunci: Data Oriented Technology Stack, DOTS, *Danmaku*, *Game*, Unity, MonoBehaviour, Entity Component System, Paradigma Berbasis Data

PENDAHULUAN

Dasar *game engine* Unity masih menggunakan paradigma *object-oriented*. Sehingga pemrogramannya pun didasarkan pada mengorganisasikan semua instansi sebagai sebuah objek. Salah satu kelemahan yang muncul dari desain pemrograman ini ialah lambatnya jalan sebuah program (Saylor Academy, 2019). Program yang terkait, atau *game* dalam kasus ini, akan menjadi semakin lambat seiring bertambahnya instansi objek yang berada pada program yang terkait; Dan di dalam *game danmaku*, sudah dipastikan akan ada banyak instansi *bullet* setiap *stage* dikarenakan dasar tema gameplay-nya sendiri, yaitu menghindari serangan-serangan yang berpola dan berpotensi rumit. Misalkan ada 1,000 instansi dan setiap siklus *frame*, Unity diperlukan untuk memanggil rangkaian satu atau banyak fungsi hanya untuk mengubah sebuah data yang dimiliki sebuah instansi; Hal tersebut tidak memakan waktu yang banyak ketika hanya terdapat beberapa instansi. Tetapi, dalam permainan genre *danmaku*, pengolahan tersebut akan sangat memperlambat sistem.

Permainan yang memiliki kemampuan berjalan pada 60 *frames per* second merupakan kemampuan yang relatif relevan karena berdasarkan sebuah eksperimen pada tesis dari Uppsala Universitet yang dijalankan oleh Rickard Hagström mengklaim bahwa jumlah *frames per second* yang dapat ditampilkan permainan berpengaruh kepada pengalaman pemain. Rickard meneliti perbandingan performa dari rentang 18 hingga 120 *frames per second*. Beliau mengatakan 18 hingga 24 *frames per second* tergolong "*unplayable*" dan 30 *frames per second* dikatakan "*felt like a minor improvement*". Seiring bertambahnya *frames per second*, semakin halus pengalaman pemain (Hagström, 2015).

Performa mereka pun juga meningkat secara signifikan. Perbandingan skor dan kematian subjek eksperimen sangat besar. Rickard juga memperhatikan sesuatu, yaitu munculnya efek samping dari bermain dengan *frames per second* yang rendah. Subjek-subjek ada yang terlihat mengeluarkan air mata pada *frames per second* yang rendah. Efek tersebut diklaim sangat terlihat pada 30 *frame per second* (Hagström, 2015) dan ke bawah.

Secara ideal, pembuatan *game* bergenre *danmaku* memerlukan desain yang lebih fokus dalam mengolah data instansi yang ada dan yang akan datang dikarenakan potensi banyaknya jumlah instansi yang berjalan dalam *game*. Paradigma desain tersebut dikenal sebagai *data-oriented programming* atau DOP. Di dalam DOP, *programmer* didorong untuk lebih memfokuskan dirinya dalam menyelesaikan sebuah permasalahan yang didasarkan oleh data atau setidaknya yang ia anggap sebagai "data" (Straume, 2019). Dengan kata lain, daripada memfokuskan organisasi data dan instansinya berdasarkan paradigma *object-oriented*; *programmer* lebih mempertimbangkan cara menyelesaikan permasalahan-permasalahan yang muncul dari data yang tersedia.

Di dunia nyata, DOP membuahkan berbagai macam pendekatan dan solusi. Hal tersebut muncul dikarenakan paradigma *data-oriented* sebenarnya kurang diteliti oleh sisi akademia maupun industri; walau DOP merupakan konsep yang tua, baru beberapa tahun terakhir ini mendapat perhatian; terutama di industri *game development* (Straume, 2019).

Salah satu dari banyaknya pola pemrograman yang muncul dari adopsi paradigma *data-oriented* ialah pola Entity Component System atau ECS. Pola pemrograman ECS sudah mulai diaplikasikan di berbagai industri *game development* (Holopainen, 2016) dikarenakan fokusnya terhadap permasalahan performa, suatu permasalahan yang sering mewabahi banyak *game development* dikarenakan jumlah instansi program yang banyak.

Unity memiliki beberapa *package* bagi *developer* yang menginginkan *game development* yang didasarkan ECS. Sekumpulan *package-package* tersebut biasa dinamakan Data Oriented Technology Stack atau DOTS. Teknologi tersebut memberi API untuk menjalankan pola arsitektur pemrograman ECS pada Unity menggunakan C#. Namun, Unity Technologies mengatakan bahwa teknologi tersebut masih belum matang dan bahkan belum disarankan untuk dipakai untuk produksi (Unity Technologies, 2021).

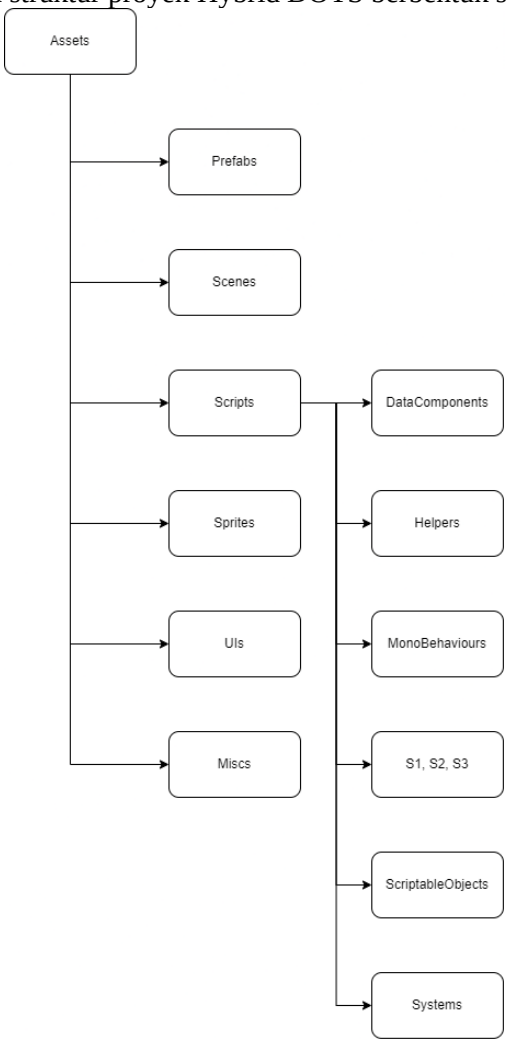
METODOLOGI DAN IMPLEMENTASI PROGRAM

Development

Untuk meminimalisir permasalahan itu, beberapa proses akan diimplementasikan menggunakan MonoBehaviour demi kemudahan dan pencegahan *error*. Pendekatan ini terkenal dalam komunitas sebagai *hybrid solution*, yaitu *development* menggunakan ECS dengan DOTS dan OOP dengan MonoBehaviour.

Keputusan ini juga akan berpengaruh kepada struktur pemrograman dikarenakan tanggung jawab pengolahan datanya berubah sebagian dari desain OOP menjadi menggunakan arsitektur ECS. Pengorganisasian OOP dapat digunakan untuk mengatur alur *gameplay* dan efisiensi arsitektur ECS dapat dimanfaatkan untuk mempercepat pengolahan data bagi CPU. Untuk menunjukkan perbandingan struktural, berikut terdapat bayangan-bayangan *class diagram* untuk perbandingan.

Development kedua permainan yang akan dihasilkan tidak dilakukan secara parallel, melainkan secara sekuensial. Sehingga, *development* permainan versi DOTS dahulu yang akan di-*develop*. *Development* versi Hybrid DOTS dimulai pada Januari 2022 dan MonoBehaviour dimulai pada Oktober 2022; *development* juga tidak berjalan kontinu tetapi ada masa-masa jeda. Selain itu ada kemungkinan besar terdapat perbedaan struktur secara minor, seperti jumlah *script* dan keberadaan *script*; dikarenakan pengintegrasian antar-*script* pada Hybrid DOTS berbeda dengan MonoBehaviour. Kurang-lebih struktur proyek Hybrid DOTS berbentuk seperti berikut;



Gambar 1. Struktur Proyek

Game Design

Outline

Pemain akan bermain sebagai Koakuma, sebuah karakter orisinil dari Touhou Project, yang ditugaskan untuk mengembalikan buku yang dicuri antagonis. Pemain perlu mengalahkan antagonis sambil menghindari serangan-serangan yang ia telah pelajari dari buku sihir yang dicurinya.

Permainan akan berbasis 2D dengan tampilan *top-down*. Target *platform* yang ditujui ialah komputer yang menjalankan *operating system* Windows 10. Permainan akan diprogramkan menggunakan C# dengan Unity.

Gameplay

Cerita dari permainan ialah terdapat sebuah buku dari perpustakaan Scarlet Devil Mansion yang berisi detail semua *spell card* Gensokyo dicuri oleh sebuah antagonis berupa *youkai*. Antagonis tersebut bersatu dengan buku dan mengintroduksi dirinya sebagai "Postulate" dengan julukan "Wayward Tsukumogami of the Library" kemudian kabur. Sebagai *familiar* dari anggota *mansion* tersebut, Koakuma ditugaskan oleh majikannya untuk mengejar Postulate dan mengembalikan buku yang dicuri.

Sebuah lagu akan mulai bermain ketika masuk ke layar menu utama. Kemudian, saat pemain memulai permainan dengan mengklik Start, tampilan akan berpindah ke layar stage dimana permainan dimulai. Koakuma akan berdialog dengan Postulate dan menyatakan bahwa ia akan mengembalikan buku yang dicurinya. Tetapi Postulate tidak akan langsung mengalah, ia memiliki kemampuan semua *spell card* yang ada di Gensokyo karena telah menyatu dengan buku sihir Patchouli.

Mekanika

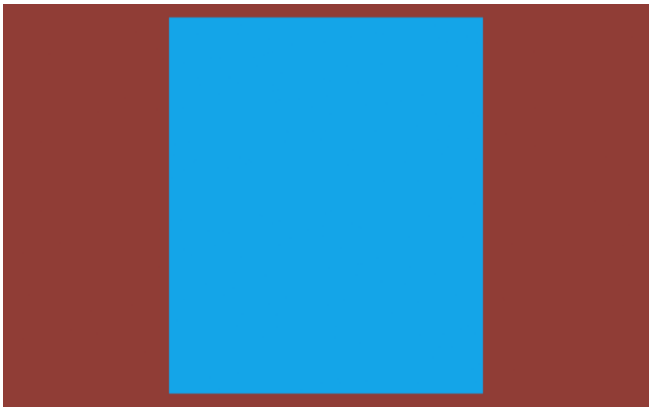
Pemain akan bermain sebagai Koakuma yang telah mempelajari beberapa *sanguine magic* dari majikannya, Patchouli. Koakuma bisa menggunakan *familiar*-nya yang berupa kelelawar untuk menembak tetesan darah kepada lawan.

Sebagai salah satu keputusan desain, nyawa pemain dapat dianggap tidak terhingga. Ketika pemain terkena tembakan, ia akan langsung *respawn* pada tengah bawah *stage*. Dapat dikatakan bahwa kekalahan sebenarnya tidak ada. Tetapi, kekalahan adalah sesuatu yang subjektif dalam permainan yang seperti ini, memiliki nyawa tak terhingga. Dapat diasumsikan bahwa pemain kalah jika terkena sebuah tembakan, jika itu definisi kalah yang ia terima.

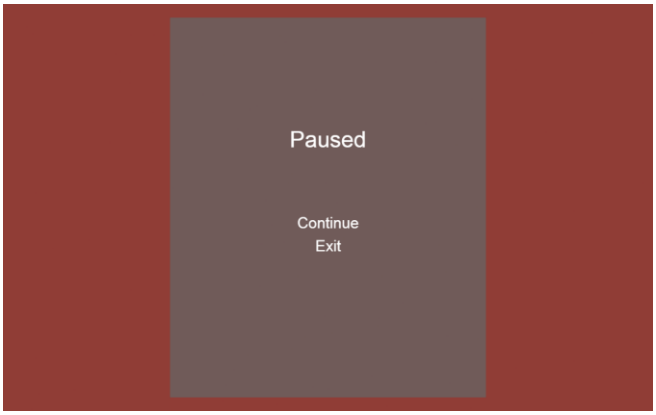
Sedangkan, untuk memenangkan permainan, pemain perlu menghabisi nyawa Postulate pada setiap *spell card*. Setelah nyawa Postulate habis pada sebuah *spell card*, permainan akan membersihkan entitas-entitas yang terkait dengan *spell card* tersebut; hal ini disebut sebagai *clean-up*. Setelah memenangi permainan, pemain akan diberi sebuah dialog penutup.

Draft Layar

Gambar 2. *Menu Draft*



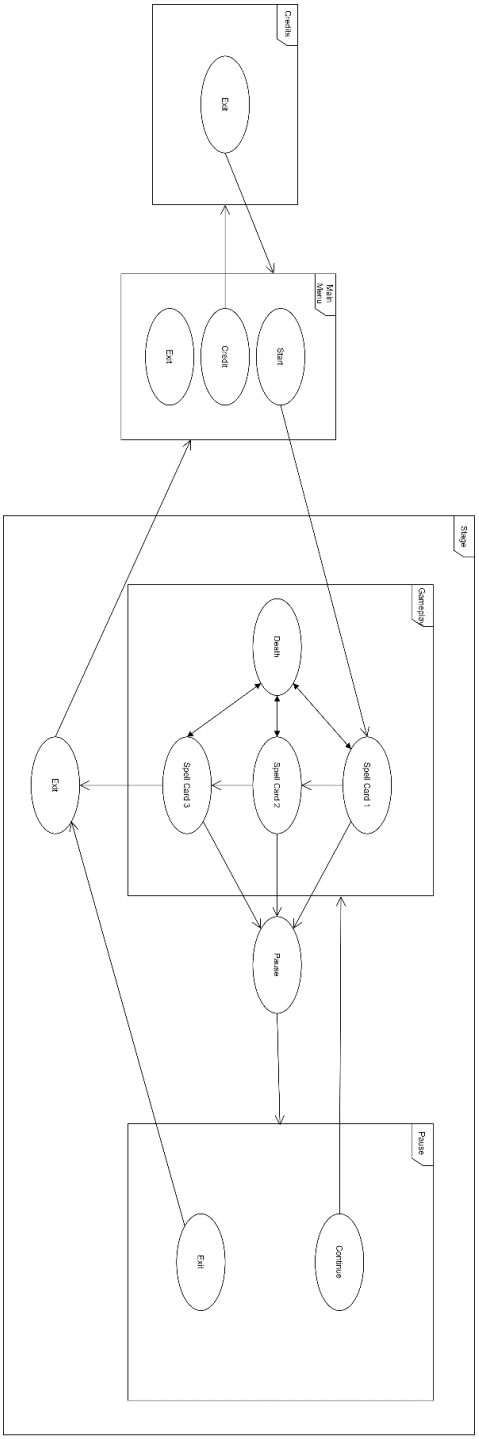
Gambar 3. *Stage Draft*



Gambar 4. *Pause Draft*

Gambar 5. *Credits Draft*

Flow Diagram



Gambar 6. *Flow Diagram*

Testing

Sesuai dengan yang tertera sebelumnya, fokus permasalahan pada karya ilmiah ini ialah mencari cara untuk memanfaatkan Hybrid DOTS untuk membuat sebuah permainan *danmaku* yang dapat berjalan dengan tingkat *frames per second* setinggi 60; Jika dipecahkan, berarti ada dua faktor yang membutuhkan pernyataan eksternal yang menyatakan kesuksesan penelitian, yaitu:

- Konfirmasi penggolongan hasil penelitian sebagai sebuah *danmaku*

- Pencapaian 60 *frames per second*

Terkait faktor pertama; seperti yang telah ditulis sebelumnya; *genre* sebuah permainan didefinisikan oleh komunitas pemain. Untuk memenuhi konfirmasi tersebut, permainan yang dihasilkan akan dicoba oleh minimal 4 pemain-pemain yang memiliki setidaknya 4000 total jam bermain dan meminta opini mereka mengenai apakah hasil akhir merupakan permainan *danmaku*.

Untuk faktor kedua, hal tersebut juga bergantung pada *hardware* pengguna. Walau dengan spesifikasi yang bagus, sebuah mesin komputer dengan tempat penyimpanan HDD yang sudah cacat akan berjalan lama. Sehingga, faktor tersebut akan dianggap terpenuhi jika mayoritas dari pencoba-pencoba mencapai 60 *frames per second*.

Sedangkan untuk mengetahui jawaban fokus permasalahan yang kedua, nanti akan diprogramkan mode khusus pengukuran pada *scene stage* untuk kedua versi permainan, MonoBehavior dan Hybrid, dalam Unity Editor yang akan menjalankan sebuah simulasi *spell card* sepanjang 30 detik untuk masing-masing *spell card*. Setiap frame akan dicatat secara programatis waktu yang digunakan oleh permainan dalam mengolah entitas/objek permainan. Program akan mengirim HTTP yang berisi data tercatat ke server lokal.

Selain itu, akan ada permainan versi kedua yang berbasis MonoBehaviour yang hanya akan digunakan untuk perbandingan performa bagi para pemain. Kedua permainan tersebut akan dimainkan oleh para pemain tanpa mereka ketahui perbandingan cara kerja *code* programnya. Kemudian akan diberi pertanyaan mengenai permainan mana yang performanya lebih halus.

HASIL IMPLEMENTASI

Kode

Dikarenakan panjang isi setiap *script*, semua *source code* dapat dilihat dalam Github untuk kenyamanan. URL *source code* kedua game dapat dilihat pada lampiran atau melalui URL berikut;

- https://github.com/TNBruh/UDWG_MB
- https://github.com/TNBruh/UnboundDevilmentWithinTheGrimoires

Ekstraksi Data

Data terekstrasi dengan membuat setiap permainan mengirim data pengukuran dengan protokol HTTP saat berada dalam mode pengukuran. Terdapat 2 *script* yang bekerja sama untuk hal itu;

- FPSMB.cs: Menghitung jumlah unit waktu delta time per detik dengan memanfaatkan fitur coroutine
- BenchMB.cs: Mengirim data yang dikumpulkan ke *server*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using UnityEditor;

public class FPSMB : MonoBehaviour
{
    [SerializeField] TextMeshProUGUI txt;
    float totalDeltaTime = 0f;
    uint frameCount = 0;

    private void Start()
    {
        totalDeltaTime = 0f;
        frameCount = 0;
```

```
        StartCoroutine(FPSMeter());
    }

    IEnumerator FPSMeter()
    {
        yield return new WaitForFixedUpdate();
        while (true)
        {
            yield return new WaitForSeconds(1);
            if (StageManager.isTesting)
            {
                BenchMB.Store(StageManager.spellPhase, totalDeltaTime,
System.GC.GetTotalMemory(false));
            }
            txt.text = (frameCount / totalDeltaTime).ToString();
            totalDeltaTime = 0f;
            frameCount = 0;
        }
    }

    private void Update()
    {
        totalDeltaTime += Time.deltaTime;
        frameCount++;
    }
}
```

Kode 1. FPSMB.cs

Data akan diterima oleh HTTP *server* yang dibuat sendiri menggunakan Rust dengan bantuan *library* tiny http dan serde kemudian diekspor ke dalam bentuk *file*.

Wawancara dengan Para *Playtester*

Wawancara dilakukan dalam Google Meet dan direkam dengan OBS pada sistem operasi Windows. Pada setiap sesi, dapat dikonfirmasi bahwa permainan mampu berjalan 60+ FPS. Subjek-subjek juga tidak mengenal arsitektur dan/atau pendekatan ECS dan MB. Setiap wawancara diberi pertanyaan relevan yang mendekati sebagai berikut;

1. *What do you think is a danmaku / bullet hell*?
2. *What danmakus / bullet hells have you played*?
3. *Do you think these are danmakus / bullet hells*?
4. *Out of these 2 games, which do you feel ran the smoothest*?

Len

1. *From what I've played, heard, and seen; I think a bullet hell is just a game where the main gameplay mechanics is dodging a large amount of projectiles on the screen with a little playable character*
2. *Several Touhou games. Not much other than that.*
3. *Yeah, they fit the description of a danmaku.*
4. *The 2nd game* (MB) *felt smoother.*

Market Piler

1. *I would say danmaku is a dodging game, usually a top-down, and your main objective is to avoid enemies rather than attack them; in a traditional sense.*
2. *Several post-2000 Touhou games and some other indie titles, like Nuclear Throne, Enter the Gungeon, and The Binding of Isaac.*
3. *For sure, yes.*
4. *It was hard to tell; but the 2nd one (MB) felt a little smoother.*

Bean

1. *Fast-paced game where you need to react quickly.*
2. *I've never played any bullet hell.*
3. *They're the exact definition of bullet hell that I think of when I hear people mention "bullet hell".*
4. *The 2nd one (MB).*

island boy / Jeff

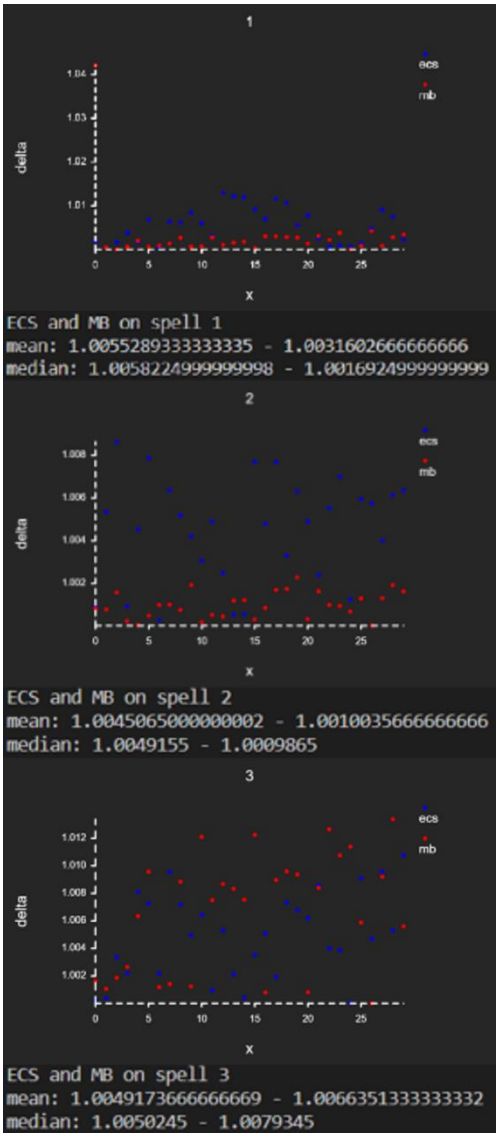
1. *High skill cap game where you have to dodge enemy projectiles in order to survive.*
2. *I've played Enter the Gungeon, it's similar to The Binding of Isaac; a bullet hell-styled rogue-like. I've also played Nier, which is a JRPG bullet hell in a 3D plane unlike most traditional bullet hells.*
3. *I would say yeah. They are what would be classified as traditional bullet hells.*
4. *I didn't notice too much of a difference.*

Data Hasil Performa

Untuk menjalankan kedua permainan yang telah dibuat, dibutuhkan sebuah mesin yang memiliki sistem operasi Windows. Berikut adalah *hardware* yang dimiliki mesin tersebut;

- Processor: Intel(R) Core(TM) i7-10750H CPU
- RAM: 16,0 GB (15,8 GB usable)
- Storage: 1 TB NVMe

Berikut adalah hasil pengolahan data yang terekstraksi dengan setiap *spell card* berjalan sekitar 30 detik;



Gambar 7. Graf dan Perhitungan Hasil Performa

KESIMPULAN

Penerapan Hybrid DOTS

Penerapan Hybrid DOTS diterapkan dengan mengkhususkan bagian DOTS untuk pengolahan entitas-entitas yang terlibat langsung dalam *gameplay* dan bagian MonoBehaviour dikhususkan untuk mengatur alur permainan. Hal-hal seperti instansiasi entitas, gerakan permain, dan penempatan *bullet* akan ditugaskan kepada fungsi-fungsi DOTS. Sedangkan fungsi yang tidak terkait langsung pada gameplay seperti orkestrasi alur *spell card*, GUI, dan perpindahan *scene* akan ditugaskan kepada sisi MonoBehaviour. Terutama GUI, karena masih belum ada solusi built-in untuk menerapkan GUI dengan DOTS.

Hasil Wawancara *Playtester*

Semua *playtester* dapat mengkonfirmasi bahwa permainan yang disediakan adalah danmaku, termasuk playtester yang mengakui bahwa beliau belum pernah memainkan genre tersebut. Para subjek juga mengkonfirmasi hal tersebut dengan tingkat keyakinan yang tinggi.

Tiga dari empat *playtester* jelas merasa lebih nyaman dengan versi yang dihasilkan dengan pendekatan MonoBehaviour tradisional dan salah satu dari mereka dapat melihat perbandingan FPS

yang beliau katakan “sangat kecil” dimana versi berbasis MonoBehaviour terasa memiliki tingkat rata-rata FPS yang sedikit lebih tinggi.

Hasil Ekstraksi Data

Hasil ekstraksi data mendukung bahwa pendekatan MonoBehaviour lebih performatif; kecuali pada *spell card* ke-3. Pengolahan berdasarkan unit waktu per detik yang dilakukan versi pendekatan MonoBehaviour lebih sedikit daripada pendekatan Hybrid ECS. Tetapi, dalam *spell card* ke-3, rata-rata dan median unit waktu per detik yang digunakan Hybrid ECS lebih kecil daripada pendekatan MonoBehaviour.

REFERENSI

Hagström, R. (2015). *Frames That Matter: The Importance of Frames per Second in Games.* Uppsala: Uppsala Universitet.

Holopainen, T. (2016). *Object-oriented programming with Unity.* Bachelor's Thesis, JAMK University of Applied Sciences, Department of Technology, Communication, and Transport.

Saylor Academy. (2019, April 15). *Advantages and Disadvantages of Object-Oriented Programming.* Retrieved from Saylor Academy: https://resources.saylor.org/wwwresources/archived/site/wp-content/uploads/2013/02/CS101-2.1.2-AdvantagesDisadvantagesOfOOP-FINAL.pdf

Straume, P.-M. (2019). *Investigating Data-Oriented Design.* Master’s thesis in Applied Computer Science, Norwegian University of Science and Technology, Department of Computer Science.

Unity Technologies. (2021, October 16). *Experimental packages.* Retrieved from Unity Documentation: https://docs.unity3d.com/2021.1/Documentation/Manual/pack-exp.html