

**SINKRONISASI DATA KOMPONEN *GAMEOBJECT* PADA *GAME*
MULTIPLAYER DENGAN *MULTITHREADING*
BERBASIS *RAISE EVENT***

SKRIPSI

Oleh :

NAUFAL PRATAMA PUTRA

NIM. 16650066



**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2023**

**SINKRONISASI DATA KOMPONEN *GAMEOBJECT* PADA *GAME*
MULTIPLAYER DENGAN *MULTITHREADING*
BERBASIS *RAISE EVENT***

SKRIPSI

Diajukan kepada:

Universitas Islam Negeri (UIN) Maulana Malik Ibrahim Malang

Untuk memenuhi Salah Satu Persyaratan dalam

Memperoleh Gelar Sarjana Komputer (S.Kom)

Oleh :

**NAUFAL PRATAMA PUTRA
NIM. 16650066**

**PROGRAM STUDI TEKNIK INFORMATIKA
FAKULTAS SAINS DAN TEKNOLOGI
UNIVERSITAS ISLAM NEGERI MAULANA MALIK IBRAHIM
MALANG
2023**

HALAMAN PERSETUJUAN

**SINKRONISASI DATA KOMPONEN GAMEOBJECT PADA GAME
MULTIPLAYER DENGAN MULTITHREADING
BERBASIS RAISE EVENT**

SKRIPSI

Oleh :
NAUFAL PRATAMA PUTRA
NIM. 16650066

Telah Diperiksa dan Disetujui untuk Diuji:
Tanggal: 8 Juni 2023

Pembimbing I

Juniardi Nur Fadila, M.T
NIP. 19920605 201903 1 015

Pembimbing II

Dr. Frosy Nugroho, M.T
NIP. 19710722 201101 1 001

Mengetahui,
Ketua Program Studi Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang



Dr. Fachrul Kurniawan, M.MT, IPM
NIP. 19771020 200912 1 001

HALAMAN PENGESAHAN





SINKRONISASI DATA KOMPONEN *GAMEOBJECT* PADA *GAME MULTIPLAYER* DENGAN *MULTITHREADING* BERBASIS *RAISE EVENT*

SKRIPSI

Oleh :
NAUFAL PRATAMA PUTRA
NIM. 16650066


Telah Dipertahankan di Depan Dewan Penguji Skripsi dan Dinyatakan Diterima Sebagai Salah Satu Persyaratan Untuk Memperoleh Gelar Sarjana Komputer (S.Kom)
Tanggal: 15 Juni 2023

Susunan Dewan Penguji

Ketua Penguji	: <u>Dr. Muhammad Faisal, M.T</u> NIP. 19740510 200501 1 007	()
Anggota Penguji I	: <u>Dr. Yunifa Miftachul Arif, M.T</u> NIP. 19830616 201101 1 004	()
Anggota Penguji II	: <u>Juniardi Nur Fadila, M.T</u> NIP. 19920605 201903 1 015	()
Anggota Penguji III	: <u>Dr. Fresy Nugroho, M.T</u> NIP. 19710722 201101 1 001	()

Mengetahui dan Mengesahkan,
Ketua Program Studi Teknik Informatika
Fakultas Sains dan Teknologi
Universitas Islam Negeri Maulana Malik Ibrahim Malang




Dr. Fachrul Kurniawan, M.MT, IPM
NIP. 19771020 200912 1 001

PERNYATAAN KEASLIAN TULISAN

Saya yang bertanda tangan di bawah ini:

Nama : Naufal Pratama Putra

NIM : 16650066

Fakultas / Jurusan : Sains dan Teknologi / Teknik Informatika

Judul Skripsi : Sinkronisasi Data komponen *Gameobject* pada *Game Multiplayer* dengan *Multithreading* berbasis *Raise Event*

Menyatakan dengan sebenarnya bahwa Skripsi yang saya tulis ini benar-benar merupakan hasil karya saya sendiri, bukan merupakan pengambil alihan data, tulisan, atau pikiran orang lain yang saya akui sebagai hasil tulisan atau pikiran saya sendiri, kecuali dengan mencantumkan sumber cuplikan pada daftar pustaka.

Apabila dikemudian hari terbukti atau dapat dibuktikan skripsi ini merupakan hasil jiplakan, maka saya bersedia menerima sanksi atas perbuatan tersebut.

Malang, 7 Juni 2023

Yang membuat pernyataan,



Naufal Pratama Putra
NIM.16650066

MOTTO

"Verily, it is by design that mankind has been fashioned to encounter trials and tribulations of life."

HALAMAN PERSEMBAHAN

Puji syukur kehadirat Allah *subhāhahu wata'āla*, atas limpahan karunia-Nya penulis dapat menyelesaikan skripsi ini. Shalawat serta salam tak lupa ucapkan kepada Nabi Muhammad SAW. Penulis dengan rendah hati mempersembahkan skripsi ini kepada Orang tua tercinta, Ayahanda Slamet Bawono dan Ibunda Luluk Rosyatul Umma yang tiada henti yang telah memberikan kasih sayang, doa, dan dukungan dalam setiap langkah hidup penulis, penulis mengucapkan terima kasih yang sangat mendalam. Penulis menyadari bahwa persembahan ini tidak akan seberapa dibandingkan dengan pengorbanan dan dukungan yang telah diberikan oleh mereka. Namun, penulis berharap bahwa persembahan ini dapat menjadi tanda terima kasih dan penghargaan atas segala bantuan dan doa yang telah diberikan.

KATA PENGANTAR

Assalamu'alaikum Warahmatullahi Wabarakatuh

Puji syukur penulis panjatkan kehadirat Allah *subhāhahu wata'āla*, Tuhan yang Maha Esa, yang telah melimpahkan rahmat dan karunia-Nya sehingga penulis dapat menyelesaikan penyusunan skripsi ini. Shalawat dan salam semoga selalu tercurah kepada junjungan kita Nabi Muhammad SAW, beserta keluarga, sahabat, dan para pengikutnya.

Penulis juga mengucapkan terima kasih kepada pihak-pihak yang telah memberikan dukungan dan bantuan dalam penyusunan skripsi ini, diantaranya :

1. Prof. Dr. H. M. Zainuddin MA, selaku rektor UIN Maulana Malik Ibrahim Malang.
2. Dr. Sri Harini, M.Si, selaku Dekan Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.
3. Dr. Fachrul Kurniawan, M.MT, selaku Ketua Prodi Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang.
4. Juniardi Nur Fadila, M.T dan Dr. Fresy Nugroho, M.T selaku dosen pembimbing yang telah memberikan arahan, kritik, dan saran yang sangat berharga.
5. Dr.Muhammad Faisal, M.T dan Dr.Yunifa Miftachul Arif, M.T selaku dosen penguji yang telah membimbing dan memberikan masukan kepada penulis sehingga tercapai hasil skripsi yang lebih baik.

6. Ayahanda dan Ibunda tercinta, serta keluarga penulis yang senantiasa memberikan doa dan restunya kepada penulis dalam segala hal.
7. Sahabat-sahabat terdekat penulis, serta rekan-rekan 'Satrivier Studio' yang membantu, mendukung dan mau meluangkan waktu dalam demi penulis.
8. Seluruh jajaran staf dan dosen jurusan Teknik Informatika yang secara langsung maupun tidak langsung terlibat dalam proses pengerjaan skripsi.
9. Semua pihak yang ikut membantu dalam menyelesaikan Skripsi ini yang tidak bisa penulis sebutkan satu persatu tanpa mengurangi rasa hormat dan terima kasih.

Malang, 25 Mei 2023

Penulis

DAFTAR ISI

HALAMAN JUDUL	i
HALAMAN PENGAJUAN	ii
HALAMAN PERSETUJUAN	iii
HALAMAN PENGESAHAN	iv
MOTTO... ..	vi
HALAMAN PERSEMBAHAN	vii
KATA PENGANTAR.....	viii
DAFTAR ISI.....	x
DAFTAR GAMBAR.....	xii
DAFTAR TABEL	xiv
ABSTRAK	xv
ABSTRACT	xvi
البحث مستخلص	xvii
BAB I PENDAHULUAN.....	1
1.1 Latar Belakang.....	1
1.2 Identifikasi Masalah.....	12
1.3 Batasan Masalah	12
1.4 Tujuan Penelitian	12
1.5 Manfaat Penelitian	13
BAB II STUDI PUSTAKA	14
2.1 Studi Literatur	14
2.2 Landasan Teori	16
2.2.1 <i>User Datagram Protocol</i>	16
2.2.2 <i>Delegate</i>	17
2.2.3 <i>Design Pattern</i>	18
2.2.3.1 <i>Subclass Sandbox</i>	20
2.2.3.2 <i>Service Locator</i>	21
2.2.3.3 <i>Factory</i>	22
2.2.4 <i>Event</i>	23
2.2.5 <i>Raise Event</i>	24
2.2.6 <i>Networked Multiplayer Game</i>	29
2.2.7 <i>Game Engine</i>	30
2.2.8 <i>Unity 3d</i>	32
2.2.9 <i>Photon Unity Network 2</i>	33
2.2.10 <i>Multithreading</i>	34
BAB III DESAIN DAN IMPLEMENTASI	36
3.1 Desain Sistem	36
3.1.1 <i>Login Account</i>	36
3.1.2 <i>Main Menu</i>	37
3.1.3 <i>Matchmaking</i>	37
3.1.4 <i>Enter Match Room</i>	39
3.1.5 <i>Wait other player to send 'Ready' State</i>	39
3.1.6 <i>Match Session Started</i>	39

3.1.7 <i>Combat Session</i>	40
3.1.8 <i>Match Session Finished</i>	41
3.2 Desain Pengujian	42
3.3 <i>Raise Event Pada Pemuatan Map</i>	44
3.4 <i>Raise Event Pada Pengiriman Input Character</i>	46
3.5 <i>Raise Event pada Pengambilan dan Penurunan Penumpang</i>	48
3.6 Proses Merangkum Data	49
3.7 Implementasi <i>Design Pattern</i> pada <i>Game</i>	50
3.8 Sistem <i>Photon Network Handler</i>	55
BAB IV UJI COBA DAN PEMBAHASAN	59
4.1 Implementasi	59
4.2 Perangkat Lunak yang digunakan	59
4.3 Spesifikasi Perangkat Keras	59
4.4 Implementasi Desain Pengujian	60
4.4.1 <i>Main Menu</i>	60
4.4.2 <i>Map Selection</i>	61
4.4.3 <i>Matchmaking</i>	62
4.4.4 <i>Load Map</i>	63
4.4.5 <i>Load Traveler</i>	64
4.5 Pengujian Rata-rata kecepatan kirim data pada <i>Function</i>	65
4.6 Pengujian Fungsi <i>Spawn Map</i>	68
4.7 Pengujian Fungsi Send Input	70
4.8 Pengujian Fungsi Driver Delivery	71
4.9 Pengujian Validasi Posisi <i>Map</i>	72
4.10 Pengujian Validasi Posisi Karakter.....	73
4.11 Perbandingan Hasil Pengujian	77
4.12 Integrasi Sains dan Islam	78
4.12.1 Q.S. Al-Anbiya' ayat 80 dalam Konteks Teknologi	79
4.12.2 Q.S. Al-'Alaq ayat 1-5 dalam Konteks Menuntut Ilmu	80
BAB V KESIMPULAN DAN SARAN	85
5.1 Kesimpulan	85
5.2 Saran	86
DAFTAR PUSTAKA	

DAFTAR GAMBAR

Gambar 2.1 <i>TCP vs UDP Header</i>	17
Gambar 2.2 Ilustrasi fungsi <i>Delegate</i>	18
Gambar 2.3 <i>Raise Event</i>	26
Gambar 2.4 <i>Diagram Venn dari Game Multiplayer dan Networked Game</i>	30
Gambar 2.5 <i>Photon PUN 2 Room Authority</i>	34
Gambar 2.6 <i>Multithreading sample</i>	35
Gambar 3.1 <i>Desain Sistem</i>	36
Gambar 3.2 <i>Detail Matchmaking</i>	37
Gambar 3.3 <i>Create Room Illustration</i>	38
Gambar 3.4 <i>Combat Session Detail</i>	40
Gambar 3.5 <i>Match Session Finish block diagram flow</i>	42
Gambar 3.6 <i>Distribusi metode RPC atau Raise Event pada Client</i>	43
Gambar 3.7 <i>Spawn Map Raise Event Block Diagram</i>	45
Gambar 3.8 <i>Input Character Player Raise Event Block Diagram</i>	46
Gambar 3.9 <i>Driver Delivery Raise Event</i>	48
Gambar 3.10 <i>Data Collection Concept</i>	49
Gambar 3.11 <i>NetInfo Downloader Interface</i>	50
Gambar 3.12 <i>Service Locator Class</i>	51
Gambar 3.13 <i>Service Provider Registration</i>	51
Gambar 3.14 <i>Start Match Pseudocode</i>	52
Gambar 3.15 <i>Network Factory Inheritance</i>	52
Gambar 3.16 <i>Factory Run Call Illustration</i>	53
Gambar 3.17 <i>Factory Registration Pseudocode</i>	53
Gambar 3.18 <i>Subclass Sandbox Implementation</i>	54
Gambar 3.19 <i>Call Net Statistics Function</i>	55
Gambar 3.20 <i>Raise Event pseudocode flow</i>	55
Gambar 3.21 <i>RPC flow</i>	56
Gambar 3.22 <i>Raise Event Handler</i>	57
Gambar 3.23 <i>Call via Factory</i>	58
Gambar 3.24 <i>RPC Factory Member</i>	58
Gambar 4.1 <i>Main Menu Interface</i>	61
Gambar 4.2 <i>Map Selection Interface</i>	61
Gambar 4.3 <i>Matchmaking Interface</i>	62
Gambar 4.4 <i>Cancel Matchmaking Interface</i>	63
Gambar 4.5 <i>Initializing Map Interface</i>	63
Gambar 4.6 <i>Map Loaded Interface</i>	64
Gambar 4.7 <i>Pick & Drop Traveler Interface</i>	64
Gambar 4.8 <i>Spawn Map Chart</i>	69
Gambar 4.9 <i>Send Input Chart</i>	70
Gambar 4.10 <i>Driver Delivery Chart</i>	72

Gambar 4.11 <i>Average speed saved</i>	78
Gambar 5.1 <i>Speed saved by each system</i>	82

DAFTAR TABEL

Tabel 3.1 <i>Data Type Byte Size</i>	44
Tabel 4.1 Rata-rata kecepatan <i>player</i> menggunakan <i>Raise Event</i>	65
Tabel 4.2 Rata-rata kecepatan <i>player</i> menggunakan <i>Remote Procedure Call</i>	66
Tabel 4.3 <i>Average Call Speed</i>	68
Tabel 4.4 <i>Spawn Map Analysis</i>	69
Tabel 4.5 <i>Send Input Analysis</i>	70
Tabel 4.6 <i>Driver Delivery Analysis</i>	71
Tabel 4.7 <i>Spawn Map Validation</i>	72
Tabel 4.8 <i>Player 1 Character Spawn Position Validation</i>	73
Tabel 4.9 <i>Player 2 Character Spawn Position Validation</i>	74
Tabel 4.10 <i>Player 3 Character Spawn Position Validation</i>	75
Tabel 4.11 <i>Player 4 Character Spawn Position Validation</i>	76
Tabel 4.12 <i>Sequential Average Speed</i>	77
Tabel 4.13 <i>Multithreading Average Speed</i>	78
Tabel 5.1 rata-rata kecepatan <i>RPC</i> dengan <i>Raise Event</i>	85

ABSTRAK

Putra, Naufal P. 2023. **Sinkronisasi Data komponen Gameobject pada Game Multiplayer dengan Multithreading berbasis Raise Event**. Skripsi. Jurusan Teknik Informatika Fakultas Sains dan Teknologi Universitas Islam Negeri Maulana Malik Ibrahim Malang. Pembimbing: (I) Juniardi Nur Fadila, M.T (II) Dr.Fresy Nugroho, M.T.

Kata kunci: multiplayer game, networking, Photon PUN 2, Raise Events, Remote Procedure Call (RPC).

Dengan berkembangnya teknologi pada industri pengembangan *game*, mampu mempermudah proses iterasi pada pengembangan *game*, sehingga proses tersebut menjadi lebih cepat. hal tersebut juga ikut mempengaruhi proses pengembangan *game multiplayer*, dimana telah tersedia berbagai templat sistem kode jaringan atau *netcode* yang telah mencakup kebutuhan dasar pengembangan *game multiplayer*, sehingga pengembang tidak perlu merancang dari awal kode jaringan dan hanya perlu memodifikasi *netcode* sesuai kebutuhan. namun dengan adanya kemudahan-kemudahan tersebut, tidak secara otomatis menyelesaikan tantangan pada pengembangan *game multiplayer*. salah satu tantangan tersebut ialah menyediakan transfer data yang stabil dan cepat antar *client*, *client* menuju *server* atau sebaliknya. pada penelitian ini, penulis mencoba memecahkan tantangan tersebut dengan meneliti salah satu metode pengiriman data yaitu *Raise Event* yang terdapat pada *netcode Photon PUN 2*. penulis membandingkan kecepatan pengiriman data *Raise Event* dengan metode *Remote Procedure Call* yang diimplementasikan pada *game multiplayer*. dimana pada hasil penelitian tersebut, penulis menemukan bahwa penggunaan *Raise Event* lebih cepat **47.1538140849377%** jika dibandingkan dengan *Remote Procedure Call*.

ABSTRACT

Putra, Naufal P. 2023. *Data Synchronization on Gameobject Component in Raise Event-based on Multithreaded Multiplayer Games*. Thesis. Department of Informatics Engineering, Faculty of Science and Technology, State Islamic University of Maulana Malik Ibrahim Malang. Advisor : (I) Juniardi Nur Fadila, M.T (II) Dr.Fresy Nugroho, M.T.

With the advancement of technology in the *game* development industry, the iteration process in *game* development has been made easier, resulting in a faster process. This has also affected the development of *multiplayer games*, where various *network* code templates or netcode are made available that cover the basic needs of *multiplayer game* development, so *developers* do not need to design *network* code from scratch and only need to modify the netcode as needed. However, with these conveniences, it does not automatically solve the challenges of developing *multiplayer games*. One of these challenges is providing stable and fast data transfer between *clients*, from *clients* to the *server*, or vice versa. In this study, the author attempts to solve this challenge by examining one of the data transfer methods, which is the *Raise Event* method found in the *Photon PUN 2* netcode. The author compares the speed of data transfer using the *Raise Event* method with the *Remote Procedure Call* method implemented in a *multiplayer game*. The results of this study show that the use of *Raise Event* is **47.1538140849377%** faster compared to *Remote Procedure Call*.

Keywords: multiplayer game, networking, Photon PUN 2, Raise Events, Remote Procedure Call (RPC).

البحث مستخلص

بوترا، ناوفال بي. 2023. تزامن البيانات على مكونات الكائنات اللعبة في الألعاب المتعددة اللاعبين القائمة على رفع الحدث رسالة ماجستير. قسم هندسة الحاسوب، كلية العلوم والتكنولوجيا، الجامعة الإسلامية الحكومية مولانا مالك إبراهيم مالانج M.T.، د. فريسي نوجروهو (II) M.T.، جونياردي نور فاديل (I): المشرف

الكلمات الرئيسية: لعبة جماعية، الشبكات، فوتون بان 2، رفع الأحداث، استدعاء الإجراء عن بعد

مع تقدم التكنولوجيا في صناعة تطوير الألعاب، تم تسهيل عملية التكرار في تطوير الألعاب، مما أدى إلى عملية أسرع. وقد أثر هذا أيضاً على تطوير الألعاب الجماعية، حيث يتم توفير قوالب شفرة الشبكة المختلفة أو الشفرة الشبكية التي تغطي الاحتياجات الأساسية لتطوير الألعاب الجماعية، بحيث لا يحتاج المطورون إلى تصميم شفرة الشبكة من الصفر فقط يحتاجون إلى تعديل الشفرة الشبكية حسب الحاجة. ومع ذلك، فإن هذه السهوليات لا تحل تلقائياً التحديات التي تواجه تطوير الألعاب الجماعية، واحدة من هذه التحديات هي توفير نقل بيانات مستقر وسريع بين العملاء، من العملاء إلى الخادم أو العكس. في هذه الدراسة الموجودة في شفرة *Raise Event* يحاول المؤلف حل هذا التحدي عن طريق دراسة أحد أساليب نقل البيانات، وهو طريقة *Remote* مع طريقة *Raise Event* يقارن المؤلف سرعة نقل البيانات باستخدام طريقة *Photon PUN 2* الشبكية *Remote* أسرع بالمقارنة مع *Raise Event* المطبقة في لعبة جماعية. تظهر نتائج هذه الدراسة أن استخدام *Procedure Call* *Procedure Call*.

BAB I

PENDAHULUAN

1.1 Latar Belakang

Dalam beberapa tahun terakhir, terjadi kemajuan signifikan dalam industri pengembangan *game*, yang ditandai dengan munculnya *serious games*, ekspansi pasar *game* video, dan peningkatan permintaan akan *game* berkualitas tinggi. *Serious games*, yang mengacu pada *game* yang diterapkan yang menggunakan elemen *game* untuk meningkatkan pengalaman pengguna, telah menjadi populer karena potensinya untuk menghibur dan mendidik pengguna secara simultan (Gazis & Katsiri, 2023). Hal ini telah menghasilkan evaluasi menyeluruh terhadap aplikasi, mesin *game*, dan kemajuan dalam genre *serious games* (Gazis & Katsiri, 2023).

Selain itu, minat yang semakin berkembang dalam pendidikan *game* di universitas Amerika dan Eropa telah mendorong International Game Developers Association (IDGA) untuk membuat kerangka kurikulum pengembangan *game* (Mikami et al., 2010). Kolaborasi antara industri dan akademisi telah menghasilkan hasil positif, menekankan pentingnya memiliki pemahaman mendalam tentang prosedur pengembangan *game* untuk menjaga kecepatan dengan kemajuan teknologi di industri tersebut (Mikami et al., 2010).

Untuk mengatasi kekurangan pengembang *game* yang terampil secara teknis, kurikulum pengembangan *game* yang komprehensif telah dirancang dan diimplementasikan. Kurikulum ini berfokus pada program gelar teknis yang berpusat pada teknologi/rekayasa *game* dengan penekanan pada keterampilan yang dapat dipindahkan, pemecahan masalah, matematika, rekayasa perangkat lunak

, skalabilitas, dan praktik industri (Kenwright, 2016). Kurikulum pengembangan *game* ini terhubung dengan spektrum subjek yang luas, seperti pemrograman, matematika, grafika komputer, animasi berbasis fisika, sistem paralel, dan kecerdasan buatan, untuk memenuhi kebutuhan baik penelitian maupun industri (Kenwright, 2016).

Selain itu, industri video *game* telah mengalami ekspansi dan evolusi yang signifikan, yang disebabkan oleh kemajuan teknologi yang telah memungkinkan pengembangan, distribusi, dan monetisasi video *game* (Sothmann, 2018). Industri ini telah mengalami pertumbuhan pesat selama abad ke-21 dan diproyeksikan akan terus berkembang lebih pesat di masa depan, memperluas domainnya melampaui hiburan murni untuk mencakup tujuan pendidikan, kompetitif, dan kreatif (Sothmann, 2018).

Perkembangan industri pengembangan *game* memiliki dampak signifikan pada sektor *game indie*, di mana pengembang semakin memilih untuk memproduksi *game* di luar struktur produksi dan distribusi konvensional dari perusahaan *game* mainstream. Trend ini mendorong penelitian di bidang interaksi manusia dan komputer (HCI) dan studi *game*, yang telah mengeksplorasi tantangan dan peluang unik yang terkait dengan ruang terbuka ini. Salah satu fitur penting dari *game indie* adalah penekanan mereka pada nilai-nilai seni dan budaya daripada manfaat ekonomi. Hal ini telah menghasilkan munculnya komunitas pengembangan *game indie* yang tidak didorong oleh laba dan beroperasi di bawah model produksi *game* terbuka dan partisipatif. Namun, komunitas ini juga menghadapi tantangan yang berasal dari fokus mereka pada nilai-nilai seni dan

budaya, termasuk dilema kualitas tenaga kerja yang bervariasi dan kesulitan dalam mengamankan sumber daya yang berkelanjutan serta memproduksi produk berkualitas tinggi (Freeman & Schulenberg, 2023).

Studi menunjukkan bahwa *game indie* cenderung berfokus pada seperangkat tujuan desain dan estetika yang terbatas, yang ditandai dengan mekanika *gameplay* yang khas, struktur naratif, dan gaya visual. Trend ini telah mengarah pada demokratisasi teknologi dan potensi dampaknya pada teknologi *game* di masa depan, termasuk desain, inovasi, dan proses produksi (Freeman & Schulenberg, 2023).

Industri pengembangan *game indie* adalah fenomena global yang telah melahirkan berbagai jenis *game* yang mencerminkan warisan budaya pembuatnya. Misalnya, studio *game* independen di India semakin mengintegrasikan unsur sejarah, seni, musik, dan arsitektur ke dalam *game-game* khas wilayah mereka, menciptakan *game* tunggal yang unik (Zeiler & Mukherjee, 2022).

Untuk memahami industri budaya, termasuk pengembangan *game indie*, sosok "perantara budaya" sangat penting. Perantara budaya adalah pendiri dan direktur ruang kerja *indie* yang melakukan bentuk pekerjaan yang beragam. Mobilitas terkoneksi mereka dan keterpisahan dari "lokasi" kerja tradisional memerlukan peninjauan ulang pendekatan berbasis studio standar dalam mempelajari produksi budaya *indie* (Browne & Schram, 2020).

Di antara banyak *game* yang diproduksi selama beberapa dekade, mereka dibagi berdasarkan genre. Genre *game* adalah cara mengkategorikan *game* berdasarkan mekanika *gameplay*, cerita, dan fitur lainnya (Arsenault, 2009).

Beberapa genre *game* populer termasuk *RPG*, *game FPS*, *MMORPG*, *game* simulasi, *game* strategi, dan *game* olahraga. Setiap genre memiliki fitur dan mekanika *gameplay* yang unik yang menarik bagi berbagai jenis pemain. Seiring berkembangnya industri *game*, genre *game* baru akan muncul, dan genre yang sudah ada akan terus berkembang (Apperley, 2006).

Salah satu genre *game* yang paling populer adalah *Role-playing games* (*RPGs*), dimana *RPG* ditandai oleh penekanannya pada cerita, pengembangan karakter, dan pilihan pemain (Meakin et al., 2021). *RPGs* biasanya melibatkan pemain mengambil peran karakter di dunia fiksi, di mana mereka berinteraksi dengan karakter lain dan menyelesaikan quest atau misi (Meakin et al., 2021). *The Elder Scrolls V: Skyrim* dan *The Witcher 3: Wild Hunt* menjadi beberapa contoh *RPG* yang paling sukses sepanjang masa. *RPG* dapat dibagi lagi menjadi beberapa *sub-genre*, seperti *action RPGs*, *tactical RPGs*, dan *massively multiplayer online RPGs* (*MMORPGs*) (Apperley, 2006).

Massively multiplayer online role-playing games (*MMORPGs*) adalah subgenre dari *RPG* yang memungkinkan para pemain untuk berinteraksi satu sama lain dalam dunia virtual. Salah satu contohnya adalah *World of Warcraft*, salah satu *MMORPG* yang paling sukses sepanjang masa. *MMORPG* ditandai dengan dunia terbuka yang besar, ekonomi yang didorong oleh pemain, dan interaksi sosial (De Paoli, 2013).

Menurut penelitian (Bawa et al., 2017) yang menginvestigasi penggunaan *Massively Multiplayer Online Games* (*MMOGs*) dalam pembelajaran di perguruan tinggi, ditemukan bahwa penggunaan *MMOGs* meningkatkan performa dan

interaksi pada kegiatan pembelajaran. Selain itu, menurut (Liu, 2015) implementasi sistem pembelajaran berbasis *Multiplayer Online Role Playing Game (MORPG)*, memungkinkan pengajar untuk membuat skenario *game* dan mengelola konten pembelajaran yang dapat digunakan dan digunakan kembali untuk beberapa kursus, juga pembelajaran berbasis *game* dapat dimanfaatkan sebagai lingkungan pembelajaran yang efektif, selain itu kemajuan pada bidang *game development* juga mendorong banyaknya penelitian-penelitian untuk implementasi teknologi pada *game development* tersebut, seperti pada penelitian (Nugroho et al., 2022) yang membahas pada penggunaan metode *Finite State Machine* pada penentuan tindakan atau *decision-making* pada perilaku atau *behaviour NPC*, atau pada penelitian (Faisal et al., 2022) yang menggabungkan teknologi *blockchain* pada *game* untuk memperoleh *user response* yang telah disediakan, juga pada penelitian (Fathurrahman & Miftachul Arif, 2021) yang mengimplementasikan penggunaan *decision tree* untuk penyesuaian perpindahan *scene* dan *hierarchy Finite State Machine* sebagai penentuan *behaviour* dari *NPC*, Selain itu pada bidang keagamaan sebagai contoh penelitian (Nurhayati et al., 2017) yang mengimplementasi algoritma *fuzzy* untuk pemilihan *level* pada metode pembelajaran menghafal al-quran pada *game*.

Selain di bidang pendidikan, *game multiplayer* juga telah dipelajari untuk dampak sosial dan budayanya. Sebagai contoh, sebuah penelitian dari (Dualism et al., 2015) mengeksplorasi penggunaan *game* digital sebagai sarana penelitian berbagai aspek dan kejadian keagamaan dalam *game* digital, para pemain *game*, dan praktik bermain *game* digital.

Game multiplayer itu sendiri di industri *game* telah mengalami pertumbuhan yang sangat pesat dalam beberapa tahun terakhir, Menurut laporan yang diterbitkan oleh Newzoo, pasar *game multiplayer* global menghasilkan pendapatan sebesar \$69,8 miliar pada tahun 2020. Ini mewakili peningkatan sebesar 15,7% dari tahun sebelumnya.

Pengembangan *game multiplayer* memerlukan seperangkat keterampilan dan teknologi yang unik, termasuk pemrograman jaringan, pengelolaan *database*, dan desain *game*. Salah satu tantangan utama yang dihadapi industri pengembangan *game multiplayer* adalah kebutuhan akan infrastruktur *server* yang *scalable* dan dapat diandalkan. Seiring dengan meningkatnya kompleksitas *game* dan peningkatan popularitas *game multiplayer*, tuntutan terhadap sumber daya *server* meningkat secara eksponensial. Hal ini mendorong para pengembang untuk merancang dan menerapkan arsitektur *server* yang dapat menangani jumlah pengguna simultan yang besar, sambil mempertahankan latensi rendah dan ketersediaan yang tinggi (Alexander, 2004).

Tantangan lainnya yaitu pengembang harus mempertimbangan beberapa aspek teknis yang bisa berpengaruh pada *experience user* dalam bermain *game multiplayer online*, salah satunya yang menjadi pertimbangan penting adalah waktu yang diperlukan untuk mengirim data dari *player* ke *player* lain, semakin banyak waktu yang diperlukan untuk mengirim data membuat *game* menjadi lamban dan kurang responsif (Glazer & Madhav, 2016).

Selain itu pengembang juga harus memastikan bahwa pemain memiliki cukup kesempatan untuk saling berinteraksi dengan cara yang bermakna dan bahwa

kontribusi mereka seimbang. Pengujian *game* juga menjadi lebih sulit karena diperlukan lebih banyak pengujian serta kompleksitas ruang keadaan *game* yang tumbuh secara eksponensial dengan setiap pemain baru, sehingga sulit bagi pengujian manusia untuk menguji setiap kombinasi dari setiap peristiwa (Reuter, 2016).

Populeritas permainan seluler juga memberikan dampak signifikan pada pengembangan permainan *multiplayer*. Dengan meningkatnya penggunaan smartphone dan model *free-to-play*, permainan seluler telah menjadi mode pengembangan permainan yang dominan di banyak negara, salah satu contohnya Jepang (Hartzheim, 2019). Perubahan ini telah mengakibatkan dorongan industri secara menyeluruh menuju produksi permainan seluler yang berkelanjutan, dengan banyak pengembang yang bekerja pada permainan smartphone daripada permainan konsol (Hartzheim, 2019). Popularitas permainan seluler juga telah mengarah pada pengembangan opsi permainan yang lebih santai, yang telah memungkinkan semakin banyaknya permainan santai (Martin, 2012).

Dalam hal permainan *multiplayer*, penelitian telah menunjukkan bahwa pengaruh pengalaman aliran (*flow experience*) terhadap keterlibatan pelanggan dalam permainan komputer pribadi lebih signifikan daripada dalam permainan seluler, sementara efek persepsi kemudahan penggunaan dan interaksi sosial luring pada keterlibatan pelanggan dalam permainan seluler relatif lebih tinggi (Kang et al., 2020).

Secara teknis *game online* adalah *game* yang dimainkan secara *online* melalui *Local Area Network (LAN)*, *Internet*, atau media telekomunikasi lainnya

(Freeman, 2018). sementara *game multiplayer* memungkinkan beberapa *user* yang memainkan *game* secara bersamaan akan mampu saling berinteraksi, namun hal tersebut masih terbatas pada satu perangkat (Glazer & Madhav, 2016). Sedangkan untuk *game multiplayer online* sendiri merupakan kombinasi dari kedua konsep diatas, dimana *game* mampu dimainkan lebih dari dua *player* atau *user* pada lebih dari satu perangkat dan terhubung melalui jaringan atau *network* (Armitage et al., 2006). Setiap *user* dari kalangan yang berbeda-beda tersebut bisa berinteraksi, juga berkesempatan untuk saling mengenal, mengesampingkan perbedaan, lalu bekerja sama untuk menyelesaikan tantangan pada *game* tersebut, seperti pada firman Allah Al-Quran surat Al-Hujurat ayat 13, dimana Allah *subhāhahu wata'āla* menciptakan manusia berbeda-beda agar saling mengenal

يَا أَيُّهَا النَّاسُ إِنَّا خَلَقْنَاكُمْ مِنْ ذَكَرٍ وَأُنْثَىٰ وَجَعَلْنَاكُمْ شُعُوبًا وَقَبَائِلَ لِتَعَارَفُوا ۗ إِنَّ أَكْرَمَكُمْ عِنْدَ اللَّهِ أَتْقَاكُمْ ۗ إِنَّ اللَّهَ عَلِيمٌ خَبِيرٌ

“Hai manusia, sesungguhnya Kami menciptakan kamu dari seorang laki-laki dan seorang perempuan dan menjadikan kamu berbangsa-bangsa dan bersuku-suku supaya kamu saling kenal-mengenal. Sesungguhnya orang yang paling mulia diantara kamu disisi Allah ialah orang yang paling takwa diantara kamu. Sesungguhnya Allah Maha Mengetahui lagi Maha Mengenal.”
(Q.S Al-Hujurat: 13).

Agar antar *user* mampu berinteraksi menggunakan jaringan internet, maka perlu adanya pengiriman data yang efektif dan efisien. Efektif dalam pengiriman dan tepat sasaran, efisien dalam waktu kirim dan memproses data. Dalam *game*, hal-hal diatas sangat mempengaruhi *user experience* saat memainkan *game*, seperti yang dijelaskan Glazer (2016) pengiriman yang lambat akan membuat *game* menjadi lamban dan tidak responsif, oleh karena itu perlu pertimbangan yang

matang dalam membuat *game multiplayer online* mulai dari teknologi *game engine* yang digunakan, serta teknologi pada segi teknis *networking*. Salah satu *game engine* yang populer dan menyediakan pengembangan *game multiplayer* yaitu *Unity*. *Game engine Unity* semakin populer dalam beberapa tahun terakhir karena fleksibilitas dan kemudahan penggunaannya. Ini adalah mesin *game* profesional dan multi-platform yang memiliki *audio*, video, grafik, efek pencahayaan, dan efek fisik yang dapat mensimulasikan dunia fisik sehingga pengguna merasa tenggelam dalam permainan. *Unity* menjadi populer karena komunitasnya yang besar dan aktif, sehingga banyak produk terlahir menggunakan *Unity*, Aplikasi dan produk yang dikembangkan dengan menggunakan *Unity* digunakan dalam produksi film, pengembangan *game*, dan di sektor pendidikan (Singh & Kaur, 2022).

Salah satu alasan keberhasilan *Unity* adalah kemampuannya untuk beradaptasi dengan teknologi baru. Misalnya, teknologi *VR* telah menjadi bidang yang signifikan dalam beberapa tahun terakhir, dan *Unity* telah mampu beradaptasi dengan teknologi baru ini. Teknologi desain *audio* juga harus berkembang dan beradaptasi dengan persyaratan baru. Pembuatan desain *audio* pada *game VR* memerlukan Teknik yang berbeda dengan *game non-VR* pada umumnya, dan *Unity* telah dapat memberikan alat yang diperlukan bagi desainer *audio* untuk membuat desain *audio* 3D yang dapat dipercaya (Nuora, 2018).

Unity juga telah dapat beradaptasi dengan kebutuhan yang berubah dari pengembang *game*. Misalnya, sebuah sistem untuk mempopulasikan lingkungan *Unity* secara dinamis pada waktu *runtime* menggunakan standar *Open Web3D* telah dikembangkan. Sistem ini mampu menampilkan *asset Environment* secara dinamis

pada waktu *runtime* dari repositori secara *remote*. Hal tersebut memungkinkan untuk membangun pemutar yang dapat dengan mudah memvisualisasikan revisi berbasis 3D dari database yang terkontrol secara versi di *Cloud* (Friston et al., 2017).

Unity juga digunakan dalam pengembangan sistem yang mampu menangkap gerakan dan menentukan Gerakan yang tepat pada kegiatan olahraga. Sistem menggunakan *webcam SHDR* untuk menangkap gerakan yang dilakukan oleh pengguna secara *real-time*, yang diproses untuk melacak pose dan sendi menggunakan *Machine Learning* melalui *library MediaPipe* (José et al., 2023).

Unity juga mampu digunakan untuk mengembangkan *game multiplayer*, fitur *multiplayer* di *Unity3D* memungkinkan pengembang untuk membuat berbagai macam *game online*, mulai dari *MMO* sosial yang santai hingga penembak kompetitif yang menegangkan. Fitur *multiplayer* di *Unity3D* memungkinkan untuk membuat *game* dengan *server* yang berdiri sendiri atau *dedicated server*, *client* sebagai *server* atau *host* dan teknologi *Cloud server* (Stagner, 2013).

Pada pengembangan *game multilpayer* terdapat beberapa tantangan yang harus bisa diselesaikan oleh *developer* agar mampu menghasilkan *game* yang mampu berjalan dengan baik. Selain dari tantangan dari pembuatan *game* secara umum, seperti *scope* dari *game*, *game design*, keterbatasan dari *target hardware*, dan sebagainya. *game multiplayer* juga menambahkan tantangan-tantangan baru, seperti *matchmaking*, *player balancing*, *network scalability*, pada penelitian ini, akan lebih banyak membahas untuk pemecahan masalah pada bagian *network synchronization*. Salah satu metode sinkronisasi pada *game multiplayer* adalah

menggunakan *RPC*, dimana salah satu contoh implementasi metode ini ada pada penggunaan *Photon PUN 2*. Apabila pada sistem didesain tanpa optimasi menghasilkan *jitter* dan *delay* yang cukup terlihat (Aaltonen, 2022).

Salah satu *framework* yang digunakan untuk pengembangan *game multiplayer online* adalah *Photon*. *Photon* memberikan kemudahan bagi pengembang dengan mempermudah pengiriman paket data pada objek *game* atau *gameobject* antar jaringan menggunakan beberapa fiturnya, salah satunya adalah adanya layanan *Photon Cloud*, yang membuat *developer* fokus untuk pengembangan *game multiplayer*, tanpa perlu pusing untuk manajemen *server* (Azmi et al., 2016).

Seperti saat *Player* mengirimkan pesan singkat atau chat pada *Player* lainnya, *Photon* menyediakan metode untuk mengirimkan pesan tersebut, yaitu *Remote Procedure Calls* dan *Raise Events*.

Perbedaan mendasar dari kedua metode tersebut ada pada tingkat fleksibilitas dan kompleksitasnya. Metode *Remote Procedure Calls* relatif lebih mudah karena hanya memerlukan komponen tambahan yaitu *PhotonView* pada objek *game* yang akan mengirimkan paket data maupun yang menerimanya, namun hal tersebut mengurangi fleksibilitasnya. Sementara metode *Raise Events* tidak memerlukan *PhotonView* sehingga penggunaan metode ini lebih fleksibel, namun sebagai gantinya *Raise Events* memerlukan beberapa kondisi tambahan agar bisa dieksekusi, sehingga meningkatkan kompleksitasnya (Exit Games, 2022). Pada penelitian ini akan berfokus pada penggunaan *Raise Events* pada *game multiplayer online*. Bagaimana penggunaan *Raise Event* agar mampu menyelesaikan tantangan

game multiplayer online seperti meningkatkan kecepatan pengiriman data antar *player*.

1.2 Identifikasi Masalah

Berdasarkan latar belakang yang telah dijelaskan pada poin sebelumnya, maka mampu diketahui permasalahan yang diangkat pada penelitian ini adalah sebagai berikut :

1. Menangani kecepatan transfer data dan sinkronisasi untuk *game multiplayer*.

1.3 Batasan Masalah

Agar pembahasan ini terfokus pada topik yang dipilih, maka dibuatlah batasan-batasan sebagai berikut :

1. Tes dilakukan dengan jumlah *player* minimal 2 dan maksimal 4
2. Tes dilakukan sebanyak 10 *match* pada *game*.
3. Tiap *match* akan diamati kecepatan pengiriman data pada beberapa fungsi yang dikirim dengan metode *Raise Event* dan *RPC* sebagai pembandingnya.

1.4 Tujuan Penelitian

Adapun tujuan dilaksanakannya penelitian ini adalah sebagai berikut :

1. Mampu menangani kecepatan transfer data dan sinkronisasi yang lebih baik untuk *game multiplayer*.

1.5 Manfaat Penelitian

Adapun manfaat dari penelitian ini nantinya mampu diambil adalah sebagai berikut :

1. Membantu pengembang untuk membangun sistem *networking* yang menggunakan *Photon* agar mampu mengaplikasikan Metode *Raise Event* dengan baik.
2. Memberikan referensi untuk performa dari metode *Raise Event* pada pengembangan *game multiplayer online*.

BAB II

STUDI PUSTAKA

2.1 Studi Literatur

Adapun beberapa penelitian sebelumnya yang menjadi referensi penelitian ini.

- ***Developing MOBA games using the Unity game engine***

Penelitian (Polancec & Mekterovic, 2017) berfokus untuk pembuatan *Multiplayer game* yang bergenre *Multiplayer Online Battle Arena* (MOBA) menggunakan Photon PUN, tetapi pada desain sistem masih belum berfokus pada pemanfaatan *Raise Event*, penelitian tersebut menggunakan desain sistem *multiplayer* yang bertumpu pada RPC, berbeda dengan penelitian dari penulis yang akan menggunakan RPC dan *Raise Event*.

- ***Collaborative Handheld Augmented Reality for Interactive Furniture Interior Design***

Pada penelitian (Syahrul et al., 2018) dimana penelitian *Multiplayer* tersebut memiliki desain sistem serupa yaitu mengadopsi bentuk *Room-Based Multiplayer* untuk menghubungkan *player* dengan *player* lainnya, pada penggunaan PUN di penelitian tersebut masih belum terlalu detail, dikarenakan pembahasan lebih fokus pada kolaborasi interaktif *Augmented Reality*.

- ***Multiplayer Online War Simulator Based On Unity3D***

Pada penelitian (Azmi et al., 2016) dalam penggunaan fungsi yang bertugas untuk sinkronisasi masih mengandalkan fungsi RPC dari *Photon PUN*. Hal ini berbeda dengan penelitian penulis yang dimana tidak hanya menggunakan RPC, tetapi juga menggunakan *Raise Event*.

- ***A Scoring System for Multiplayer Game Base on Blockchain Technology***

Pada penelitian (Arif et al., 2021) meneliti tentang pembuatan sistem penilaian pada *game multiplayer online* dengan memanfaatkan teknologi *blockchain*. Pada penelitian tersebut *game* menggunakan *Photon PUN*, namun penelitian tersebut lebih berfokus pada pembuatan sistem penilaian dengan *blockchain* daripada *Photon PUN* itu sendiri.

- ***Unity Multiplayer Games***

Buku karya (Stagner, 2013) berisi tentang pembuatan *game multiplayer* menggunakan *game engine Unity* dengan memanfaatkan beberapa netcode, diantara *Player.IO*, *PubNub*, *Photon Server*, dan *Photon PUN*, selain itu buku tersebut juga menjelaskan beberapa solusi dari tantangan dalam pengembangan *game multiplayer*, seperti *Client-Side Prediction*, *Server Authorative Movement*, *Server-Hit Detection*. Pembahasan *Photon PUN* sendiri masih belum detail seperti pada penelitian ini.

Dari semua penelitian terkait diatas, penggunaan Photon PUN masih belum maksimal, ditandai dengan *multiplayer* tidak sebagai fokus utama. Hal ini yang akan membedakan penelitian penulis dengan penelitian-penelitian sebelumnya. Penulis akan fokus pada bagian pengiriman data antar *player* dengan menggunakan *Raise Events* .

2.2 Landasan Teori

Bagian ini akan menjelaskan tentang teori-teori yang berkaitan dengan penelitian ini.

2.2.1. *User Datagram Protocol*

User Datagram Protocol atau UDP adalah salah satu protokol lapisan *transport* TCP/IP yang mendukung komunikasi yang tidak andal (*unreliable*) yang berarti Pesan-pesan UDP akan dikirimkan sebagai datagram tanpa adanya nomor urut atau pesan *acknowledgment*. tanpa koneksi (*connectionless*) yang berarti Pesan-pesan UDP akan dikirimkan tanpa harus dilakukan proses negosiasi koneksi antara dua host yang hendak bertukar informasi (Kelly & Kumar, 2022). UDP secara sederhana adalah mengirim pesan ke komputer lain. Pesan-pesan ini sebagian besar tidak dicek, di luar *checksum* sederhana untuk memastikan bahwa pesan tersebut tidak rusak. Oleh karena itu, pesan tidak dijamin sampai, juga tidak dijamin hanya sampai satu kali (terkadang satu pesan dapat dikirimkan dua kali atau lebih), atau bahkan dalam urutan tertentu. TCP, di sisi lain, menjamin setiap pesan diterima hanya sekali, dan dalam urutan pengiriman yang tepat,

meskipun hal ini dapat mengakibatkan pesan harus dikirim ulang beberapa kali jika gagal mencapai target, dan pesan harus disangga saat diterima, agar diproses sesuai urutan pengirimannya (Stagner, 2013). Berikut gambar 2.1 yang menunjukkan perbedaan struktur *header* dari TCP dan UDP.

Source port	Destination port	Source port	Destination port
Sequence number		Length	Checksum
Acknowledgment number			
Flags	Window size		
Checksum	Urgent pointer		
Options (if any)			

Gambar 2.1 *TCP vs UDP Header*

Struktur dari UDP jauh lebih ringkas bila dibanding dengan TCP. *Header* dari UDP sendiri memiliki Panjang 16 bits atau 8 bytes, sehingga untuk ukuran packet yang dikirim selalu Panjang data ditambah ukuran (Fox & Hao, 2017).

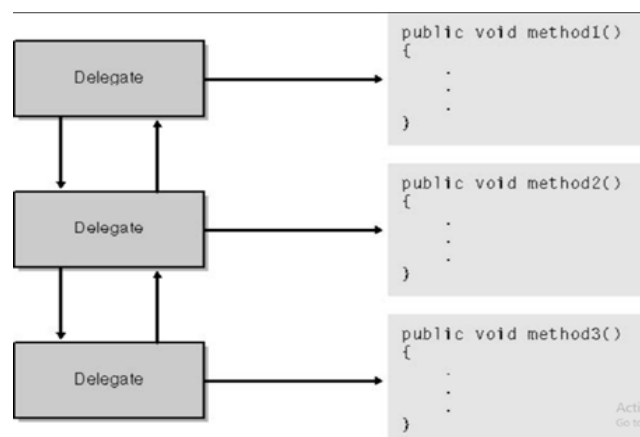
2.2.2. *Delegate*

Delegate merupakan kelas yang berfungsi untuk enkapsulasi sepotong kode sehingga dapat diedarkan dan dieksekusi seperlunya dengan cara yang aman dalam hal tipe dan parameter yang dikembalikan (Skeet, 2019), sebagai contoh dalam lingkup pemerintahan, *delegate* adalah perwakilan yang diberi wewenang untuk membuat pilihan untuk instansi yang diwakili oleh delegasi tersebut. Misalnya, negara bagian mengirim

delegasi ke konvensi pencalonan presiden. Ketika delegasi manusia tiba di sebuah konvensi, mereka bebas membuat pilihan di menit-menit terakhir

berdasarkan kondisi saat ini. Demikian pula, *delegate* pada C# menyediakan cara bagi program untuk mengambil metode alternatif yang tidak ditentukan hingga runtime. Saat menulis sebuah metode, penulis tidak selalu tahu tindakan mana yang akan terjadi saat runtime, sehingga penulis memberikan wewenang kepada *delegate* untuk menjalankan metode yang benar (Joyce Farrell, 2017).

Berikut Gambar 2.2 yang mengilustrasikan fungsi *Delegate*.



Gambar 2.2 Ilustrasi fungsi *Delegate*

2.2.3. Design Pattern

Pola atau *pattern* didefinisikan oleh serangkaian informasi, yaitu konteks, tujuan dan solusi. Konteks menjelaskan tentang kapan *pattern* tersebut berlaku, misal ingin membuat jendela pada dinding. selanjutnya didefinisikan tujuan yang akan dicapai, misal bagian bawah jendela tidak boleh terlalu tinggi atau terlalu rendah, terakhir solusi menunjukkan pada

tingkat abstrak bagaimana masalah telah berhasil dipecahkan di masa lalu, misal bagian bawah jendela berada pada ketinggian antara 12 dan 20 inci.

Pattern bukanlah dogma yang harus diikuti. Sebaliknya, mungkin ada alasan untuk sengaja melanggar aturan 12-20 inci. Misalnya, hanya atap kaca sering dipasang di ruang tertentu: penjara atau ruang mandi kolam renang yang dipasang diluar aturan 12-20 inci.

Pattern tidak memberikan rekomendasi tindakan yang spesifik. Dalam contoh diatas, perancangan ambang jendela dibuat dengan tinggi 12-14 inci atau sekitar 20 inci. sehingga, ketika merancang rencana bangunan, seorang arsitek menemukan rentang yang luas untuk dipertimbangkan tergantung pada konteksnya. Dengan demikian, *pattern* bukanlah algoritma atau implementasi konkret, melainkan solusi yang diusulkan secara abstrak yang harus disesuaikan dengan masalah yang ada (Musch, 2023).

Pattern itu sendiri hanya menjelaskan apa tujuan yang ingin dicapai, apa yang harus dicari ketika menerapkan pola, dan apa kelebihan dan kekurangannya. Namun, bagaimana cara mewujudkan *pattern* tersebut sepenuhnya tergantung pada praktisinya. Dengan cara ini *transfer* ilmu tentang *pattern* tersebut dapat diwujudkan ke semua bahasa dengan kekhasan masing-masing. Seperti halnya dengan rencana konstruksi sebuah rumah, yang menggambarkan di mana letak dinding, tetapi tidak menjelaskan bagaimana dinding tersebut harus dibangun.

Sehingga *Design Pattern* sendiri berarti suatu *pattern* yang menggambarkan solusi untuk masalah-masalah pemrograman yang sering

muncul secara abstrak. Hal ini memudahkan dalam mentransfer pengalaman dan mengandalkan solusi yang terbukti, sebagai contohnya dengan cara menggunakannya kembali solusi yang telah dipecahkan sebelumnya (Musch, 2023).

Pada penelitian ini, terdapat beberapa *design pattern* yang digunakan pada aplikasi, diantaranya :

2.2.3.1. Subclass Sandbox

Pada *pattern* ini, sebuah kelas dasar (*base class*) mendefinisikan sebuah metode abstrak yang disebut *sandbox* dan menyediakan beberapa metode operasi. tiap operasi-operasi tersebut ditandai sebagai *protected* untuk memastikan bahwa operasi-operasi tersebut hanya digunakan oleh kelas-kelas turunannya. Dikarenakan metode *sandbox* tersebut ditandai sebagai *abstract*, sehingga setiap kelas turunan perlu menimpa atau *override* metode *sandbox* dengan menggunakan operasi-operasi yang disediakan (Nystrom & Robert, 2014).

Subclass Sandbox adalah sebuah *design pattern* yang sederhana dan umum ditemukan dalam banyak *codebase*, bahkan di luar domain *game*. *Subclass Sandbox* cocok digunakan dalam situasi berikut:

- Terdapat sebuah kelas dasar yang memiliki beberapa kelas turunan.

- Kelas dasar mampu menyediakan semua operasi yang mungkin diperlukan oleh kelas turunan.
- Terdapat kesamaan perilaku di antara kelas-kelas turunan dan terdapat gagasan dari *programmer* untuk berbagi kode di antara kelas-kelas tersebut.
- *Programmer* ingin meminimalisir ketergantungan (*coupling*) antara kelas-kelas turunan tersebut dengan bagian lain dari program.

2.2.3.2. Service Locator

Service Locator merupakan design pattern yang dimana sebuah Kelas akan menyediakan layanan (*services*) dari kelas-kelas yang dianggap sebagai penyedia layanan (*service provider*). Kelas penyedia layanan didefinisikan dengan mengimplementasi interface khusus. *Service Locator* menyediakan akses ke layanan-layanan tersebut dan menemukan penyedia yang sesuai sambil menyembunyikan proses dari kelas layanan tersebut.

Penggunaan dari *Service Locator* sendiri perlu beberapa pertimbangan, daripada menggunakan mekanisme global untuk memberikan akses kode tertentu ke objek yang membutuhkannya, lebih baik untuk mengirimkan objek tersebut secara langsung. Hal ini sangat sederhana dan membuat hubungan antar kelas menjadi jelas.

Namun ada beberapa kasus dimana mengirimkan *object* secara langsung menjadi sia-sia atau justru membuat kode sulit dibaca. Beberapa sistem, seperti *logging* atau manajemen memori, sebaiknya tidak menjadi bagian dari sistem yang bersifat *public*, atau Parameter dalam sistem *rendering* seharusnya hanya berkaitan dengan *rendering*, bukan hal-hal lain seperti *logging*.

Dalam kasus-kasus seperti itu, *Service Locator* ini dapat membantu menyederhanakan akses tiap sistem. *Design pattern* ini memiliki cara kerja seperti *design pattern Singleton* namun lebih fleksibel dan lebih dapat dikonfigurasi. Ketika digunakan dengan baik, *design pattern* ini dapat membuat *codebase* lebih fleksibel dengan biaya *runtime* yang kecil.

2.2.3.3. Factory

Design Pattern factory mengasumsikan bahwa kelas-kelas dalam hierarki *factory* menciptakan objek-objek dari kelas-kelas hasil yang berbeda. *Pattern factory* sendiri adalah sebuah kelas yang mengenkapsulasi penggunaan konstruktor. Ketika sebuah kelas membutuhkan untuk membuat sebuah objek, ia memanggil sebuah metode dari kelas *factory* yang sesuai. Kelas-kelas *factory* dapat bersifat statis atau non-statis.

Kebutuhan akan *pattern factory* muncul ketika klien dapat membuat objek hasil tanpa mengetahui nama-nama setiap kelas turunan hasil. *Factory* menyatakan bahwa sistem harus membuat

hierarki *factory* paralel sehingga klien dapat membuat objek hasil dengan memanggil metode *create* pada objek *factory* yang sesuai (Sciore, 2019).

Factory dapat digunakan ketika:

- Sebuah kelas tidak dapat memprediksi kelas objek yang harus diciptakannya.
- Sebuah kelas ingin subclass-subclassnya menentukan objek-objek yang diciptakannya.
- Kelas-kelas mendelegasikan tanggung jawab kepada salah satu dari beberapa subclass bantuan, dan Anda ingin membatasi pengetahuan tentang subclass bantuan mana yang menjadi delegasi (Gamma et al., 1998).

2.2.4. Event

Event digunakan seperti *Observer Pattern*. sejumlah listener yang telah terdaftar akan memperoleh pesan notifikasi apabila *event* dipanggil. object yang akan menerima notifikasi tersebut akan mendaftarkan sebuah *delegate* pada *event*. sebuah *event* selalu didefinisikan dengan *delegate* yang harus didefinisikan dan bisa diakses. Berbeda dengan *delegate*, setiap *event* hanya bisa dipanggil di dalam kelas yang mendeklarasikan *event* tersebut. ketika *event* dipanggil, semua *delegate* yang terdaftar pada *event* tersebut akan dipanggil juga. *event* punya nilai null apabila tidak didapat listener

yang terdaftar (Bryne, 2022). Dalam C#, *event* merupakan bentuk reaksi terhadap suatu kejadian dalam suatu program. Suatu *event* terjadi ketika sesuatu yang 'menarik' terjadi pada suatu objek. hal yang 'menarik' tersebut ditentukan saat pembuatan kelas, Sebagai contoh, saat pembuatan kelas form, kelas dibuat untuk memproses user ketika menekan objek tombol dan tidak memproses user ketika menekan objek label, dalam hal ini bisa dikatakan kelas form tersebut mendefinisikan proses menekan objek tombol sebagai 'menarik' dan menekan objek label tidak 'menarik'.

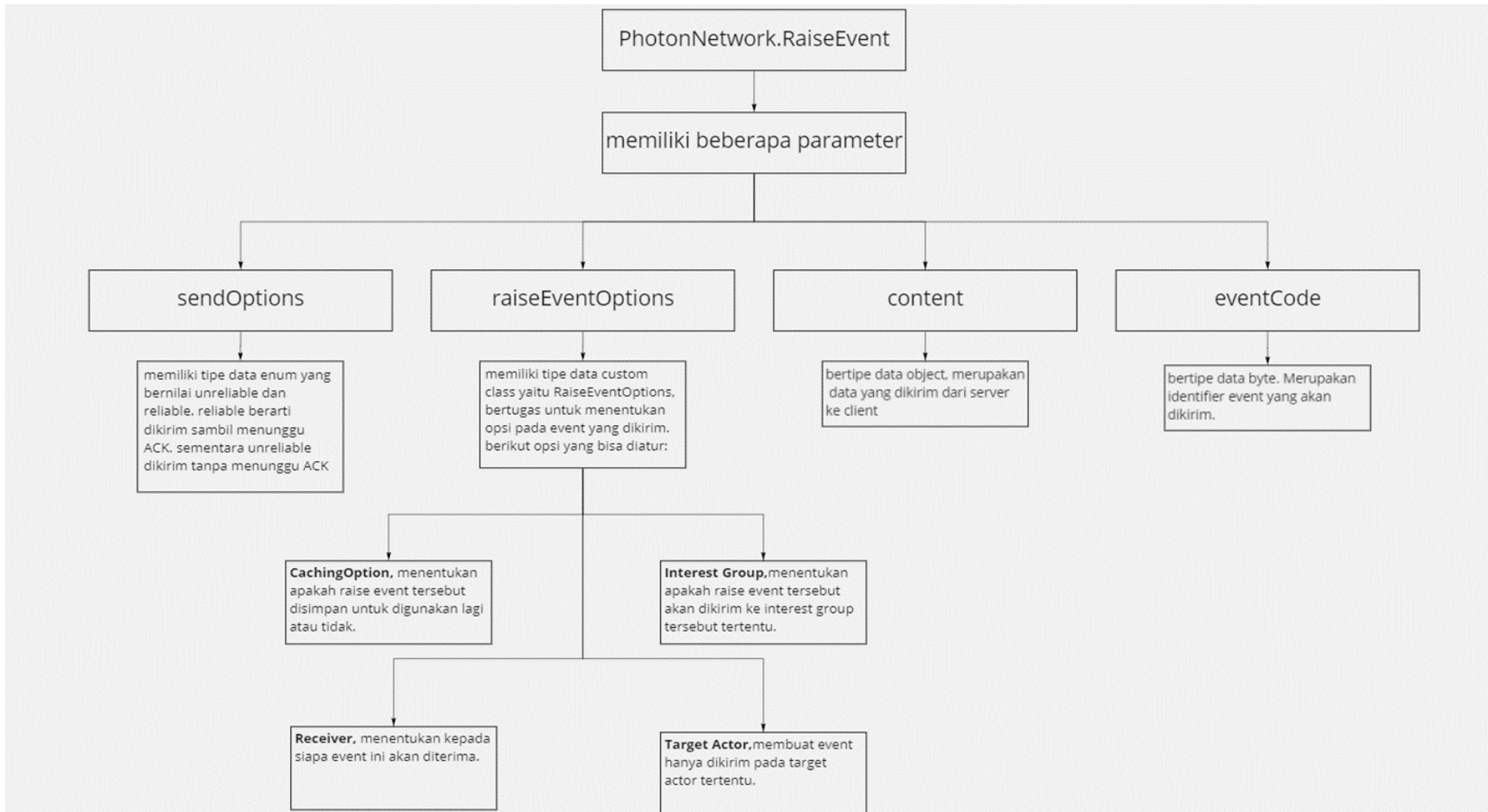
Suatu program menerapkan *event* untuk mengirim notifikasi kepada klien ketika sesuatu terjadi pada suatu objek. *event* sering digunakan pada program yang mengimplementasikan tampilan visual yang interaktif, sebagai contoh, suatu program akan mengirim notifikasi saat pengguna menekan tombol atau memilih opsi dari ListBox. Selain itu, penggunaan *event* bisa dilakukan dengan kelas biasa yang tidak mewakili kontrol grafis antarmuka dari user. Ketika objek suatu klien ingin mengetahui tentang perubahan apa pun yang terjadi pada objek, *event* memungkinkan objek memberi sinyal atau notifikasi kepada objek suatu klien (Joyce Farrell, 2017).

2.2.5. Raise Event

Raise Event adalah di mana *server* bekerja sebagai sender dan klien bekerja sebagai subscriber. *Raise Event* acara berfungsi mirip dengan *event* pada umumnya, secara umum cara kerja *event* adalah calon subscriber pertama-tama harus menentukan fungsi yang akan digunakan, kemudian

mendaftarkannya (subscribe) ke *event* di sender, satu *event* mampu menampung lebih dari satu subscriber. Kemudian ketika kondisi yang sesuai terpenuhi, sender akan memberi tahu subscriber tersebut dan subscriber akan menjalankan fungsinya.

Adapun pada *Raise Event*, klien harus menentukan *Event code*, *Event code* berfungsi sebagai kunci unik agar baik *server* maupun klien mampu mengenali *Raise Event* tersebut. Di Photon, kode peristiwa ini digambarkan sebagai nilai byte, yang memungkinkan hingga 256 peristiwa berbeda. Namun beberapa di antaranya sudah digunakan oleh Photon sendiri, sehingga *Event code* yang bisa digunakan sekitar 200 kode [0..199] (Exit Games, 2022). Untuk mengetahui lebih jelas, berikut Gambar 2.3 tentang penjelasan *Raise Event*.



Gambar 2.3 Raise Event

Seperti yang digambarkan pada gambar 2.3, Raise Event memiliki beberapa parameter yang dikirimkan pada receiver atau penerima, berikut penjelasan singkat perihal parameter-parameter tersebut.

- ***Send Options***

Menentukan pengiriman apakah akan bersifat *reliable* atau *unreliable*, perbedaan dari keduanya yaitu ketika *Send Options* bernilai *reliable*, maka sistem perlu menunggu *Acknowledgement* atau *ACK* dari sistem *Photon PUN* yang menandakan bahwa pesan telah terkirim ke penerima. Sementara bila *Send Options* bernilai *unreliable*, maka pesan dikirim tanpa perlu menunggu *Acknowledgement*.

- ***Raise Events Options***

Merupakan sebuah *custom class* yang menentukan beberapa opsi pada data yang akan dikirim. Berikut opsi-opsi tersebut.

- ***Caching Options***

Menentukan apakah *Raise Event* tersebut akan disimpan, sehingga *user* yang tergabung setelah *match* dimulai bisa menerima *Raise Event* dengan urutan yang sama seperti *user* yang tergabung tepat saat *match* dimulai. *Caching* sendiri bisa disimpan pada *room* atau secara *global*, bila disimpan pada *room*, apabila *room* telah selesai digunakan dan dihapus, maka *cache* tersebut akan ikut terhapus, sementara bila disimpan

secara *global*, *cache* tersebut akan terus ada selama *game* masih aktif.

- ***Interest Group***

Interest Group merupakan bentuk alternatif dari penerima *Raise Event* tersebut. dengan adanya *interest group*, *target* penerima jadi lebih terorganisir sesuai desain dari pengembang. parameter ini bernilai opsional.

- ***Receiver***

Parameter ini menentukan siapa yang akan menerima *Raise Event* yang dikirim. Terdapat tiga nilai yang bisa ditentukan, yaitu *All*, *Other*, *MasterClient*. *All* berarti semua *user* yang di dalam *room* akan menerima *Raise Event*, termasuk pengirim. *Other* berarti semua *user* **kecuali pengirim** yang akan menerima *Raise Event*. Sementara *MasterClient* berarti hanya *user* yang berstatus sebagai *MasterClient* yang akan menerima *Raise Event* tersebut.

- ***Target Actors***

Parameter ini menentukan secara spesifik siapa *user* yang akan menerima *Raise Event* tersebut. nilai parameter ini bisa berisi *id* dari *Actor* yang akan menerima *Raise Event*. Parameter ini bersifat opsional.

- ***Content***

Content merupakan pesan atau data yang dikirimkan kepada penerima. Sebagai contoh apabila akan mengirim suatu koordinat posisi kepada penerima agar penerima bisa bergerak menuju koordinat posisi tersebut, maka data yang dikirimkan bisa berupa *Vector3*.

- ***Event Code***

Parameter ini menentukan *Raise Event* yang akan dikirim, perlu ditandai bahwa *Raise Event* sendiri perlu di deklarasikan sebagaimana cara kerja *event*. Parameter ini bernilai *byte*.

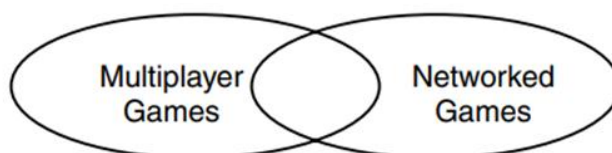
2.2.6. *Networked Multiplayer Game*

Game multiplayer adalah *game* yang membuat dua pemain atau lebih bermain di mesin fisik yang sama, sebagai contoh pemain satu akan mengambil gilirannya untuk menyelesaikan objektif dari permainan sementara pemain kedua menunggu gilirannya, contoh lainnya untuk permainan yang berlangsung secara simultan, baik secara kooperatif maupun kompetitif, setiap pemain akan melihat avatar mereka di layar yang sama atau layar akan 'dibagi' menjadi wilayah terpisah untuk setiap pemain.

Sementara itu *game networked* adalah permainan yang terhubung dengan jaringan internet, menghubungkan pemain ke *server* jarak jauh yang mengontrol berbagai aspek *gameplay*. *Game* itu sendiri, bagaimanapun, bisa sepenuhnya menjadi *game single-player* di mana tidak ada interaksi langsung dengan pemain lain atau avatar mereka. Pada sistem komputer

modern saat ini, pemain dapat menjalankan *game* secara lokal dan terhubung ke *server* untuk konten map atau berinteraksi dengan unit *Artificial intelligence (AI)* yang dikendalikan oleh *server*.

Tidak semua *game multiplayer* dimainkan lewat jaringan, maupun *game* yang dimainkan lewat jaringan tergolong *game multiplayer*. Sehingga bila digambarkan maka akan nampak seperti pada Gambar 2.4.



Gambar 2.4 Diagram Venn dari *Game Multiplayer* dan *Networked Game*

Game multiplayer saling tumpang tindih dengan *game networked*, dimana pada penelitian ini akan berfokus pada irisan dari Gambar 2.4, yaitu *game multiplayer* yang juga termasuk *game networked*.

Networked Multiplayer Game, merupakan permainan yang menghubungkan dua pemain atau lebih melalui jaringan internet, untuk bermain secara bersamaan (Armitage et al., 2006).

2.2.7. *Game Engine*

Istilah "*game engine*" muncul pada pertengahan tahun 1990-an dalam kaitannya dengan permainan penembak orang pertama (*FPS*) seperti *Doom* yang sangat populer oleh *Id Software*. *Doom* dirancang dengan memisahkan antara komponen perangkat lunak inti (seperti sistem *rendering* grafis tiga dimensi, sistem deteksi tabrakan (*collision*), atau sistem audio) dengan aset gambar dan *object 3d*, peta *game*, dan aturan

bermain yang membentuk pengalaman bermain *player*. Hal ini bertujuan agar pengembang dapat membuat produk baru tanpa harus membuat semua sistem dari awal, tapi dengan menciptakan aset gambar atau *object* 3d baru, tata letak dunia pada *game*, senjata, karakter, kendaraan, dan aturan permainan dengan perubahan minimal pada *game engine* itu sendiri. Hal ini juga menandai lahirnya "komunitas *modding*", sekelompok individu atau studio independen kecil yang membangun permainan baru dengan memodifikasi permainan yang sudah ada, menggunakan *toolkit* gratis yang disediakan oleh pengembang aslinya (Gregory, 2019).

Menurut (Gregory, 2019) pada bukunya yang berjudul *Game Engine Architecture*, tidak semua *game* dibuat dengan *game engine*. Beberapa *game* itu dibuat dengan kode-kode program yang didesain untuk menyelesaikan suatu tantangan secara spesifik, sebagai contoh sebuah *game* bisa melakukan *render* suatu *object* dengan secara spesifik, namun apabila *game* dibuat pada *game engine*, perintah *render* tersebut bisa menjadi sistem sendiri, dimana *engine* hanya menyediakan *material* untuk kepentingan umum (*general-purpose*) dan *shader*, sisanya untuk mewujudkan *object* didefinisikan berdasarkan data. Sehingga hal yang membedakan dari *game engine* dengan suatu *software game* namun bukan *engine* adalah sifat *reusability* atau sifat modular kode maupun sistem pada *game* tersebut.

2.2.8. *Unity 3d*

Unity adalah *game engine* yang populer dan mendukung berbagai *platform*. Dengan menggunakan *Unity*, pengembang dapat menerapkan permainan di *platform mobile* (*Apple iOS, Google Android*), *konsol* (*Microsoft Xbox 360 dan Xbox One, Sony PlayStation 3, PlayStation 4, Nintendo Wii, Wii U*), *platform* permainan portabel (misalnya, *Playstation Vita, Nintendo Switch*), komputer *desktop* (*Microsoft Windows, Apple Macintosh, dan Linux*), dan sistem realitas *virtual* (*VR*) (misalnya, *Oculus Rift, Steam VR, Gear VR*).

Unity didesain adalah memberikan kemudahan pengembangan dan penyebaran permainan lintas *platform*, Sehingga *Unity* menyediakan *editor* terintegrasi yang mudah digunakan, di mana pengembang dapat membuat dan memanipulasi aset dan entitas yang membentuk dunia permainan *game* dan dengan cepat melihat pratinjau permainan di dalam *editor* atau langsung di perangkat *target*. *Unity* juga menyediakan seperangkat alat untuk menganalisis dan mengoptimalkan permainan di setiap *platform target*, kemampuan untuk mengelola kualitas kinerja secara unik di setiap *platform* penyebaran. *Unity* mendukung *scripting* dalam *C#*. *Unity* juga mempunyai sistem animasi yang kuat dengan dukungan *retargeting* animasi (kemampuan untuk memutar animasi yang dibuat untuk suatu karakter dan menggunakannya pada karakter yang benar-benar berbeda), dan dukungan untuk permainan *multiplayer* yang terhubung jaringan.

Unity telah digunakan untuk membuat berbagai macam permainan yang telah diterbitkan, termasuk *Deus Ex: The Fall* oleh *N-Fusion/Eidos Montreal*, *Hollow Knight* oleh *Team Cherry*, dan *Cuphead* yang oleh *Studio MDHR*. Film pendek pemenang penghargaan *Webby, Adam*, di-render secara *real-time* menggunakan *Unity*.

2.2.9. Photon Unity Network 2

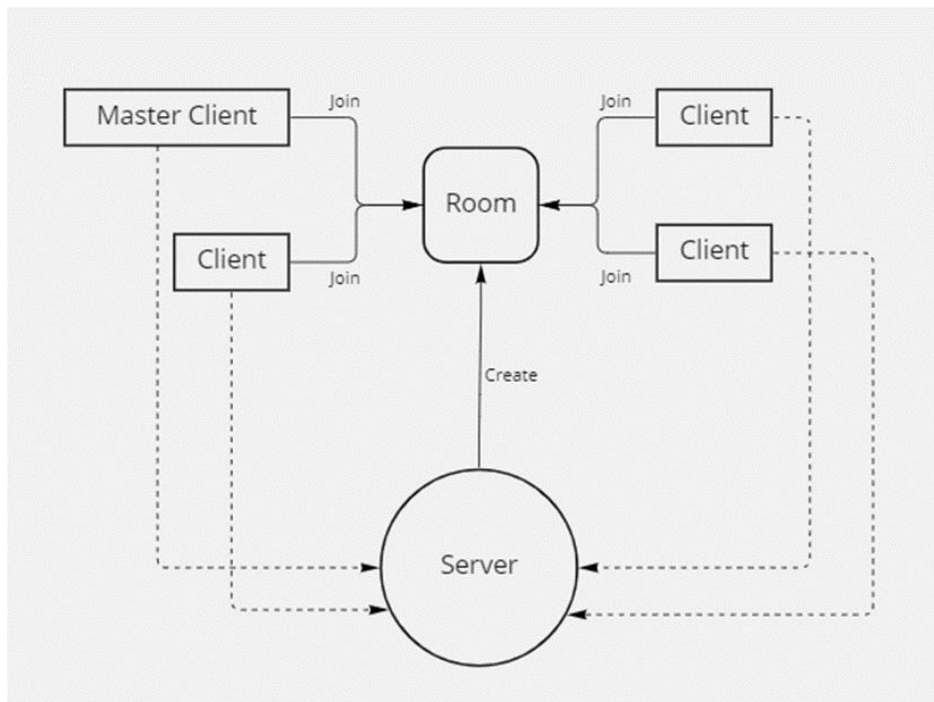
Untuk menciptakan realitas yang sama antara pemain-pemain yang terhubung melintasi jarak, seperti sistem *gameplay* yang memindahkan karakter atau memunculkan objek lalu disinkronkan ke klien lain dengan mengirimkan paket data untuk mereka.

Meskipun mungkin untuk mengirim paket-paket tersebut secara manual dengan memanggil fungsi kirim secara langsung, hal ini dapat membuat *programmer* menjadi kewalahan apabila memiliki sedikit pengalaman untuk memprogram *game multiplayer*, juga kurang efisien jika tiap pembuatan *game multiplayer* perlu membuat fungsi pengiriman secara manual.

Maka untuk mengatasinya dibuatlah suatu sistem yang menampu tugas pada sisi jaringan *game multiplayer*, sistem itu disebut kode jaringan atau netcode.

Photon Unity Network 2 (PUN 2) sendiri adalah netcode dibuat khusus untuk *game engine unity*, yang selain memiliki tugas membantu sinkronisasi data antar pemain, juga menyediakan fitur lainnya, seperti *matchmaking*, *management room*, dan sebagainya.

Secara sistem, *PUN 2* memiliki *server* yang terletak diberbagai belahan dunia atau *region*. Setiap *client* akan terkoneksi ke *region* tersebut, dan akan dihubungkan dengan *client-client* yang terkoneksi dengan *region* yang sama. Tiap *client* bisa membuat atau join pada *room* untuk memulai permainan. *Client* yang membuat *room* disebut *master client*, dimana dia bisa mempunyai tugas khusus dari *client* biasa, seperti memulai permainan, dan sebagainya (Exit Games, 2022). Bila diilustrasikan seperti pada Gambar 2.5.

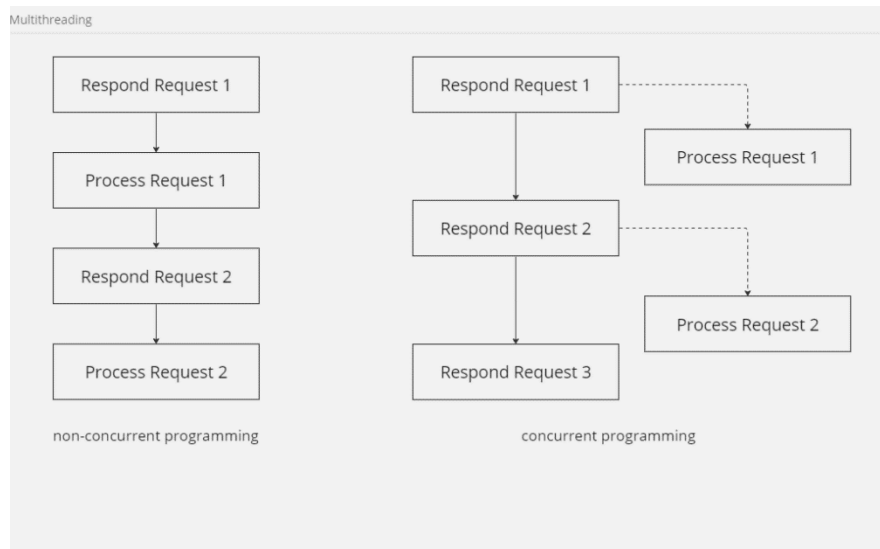


Gambar 2.5 Photon PUN 2 Room Authority

2.2.10. Multithreading

Multithreading merupakan teknik pemrograman yang membuat sistem mampu mengeksekusi fungsi berjalan secara bersamaan, atau *concurrent*, sebagai contoh sistem akan merespon suatu *request* dengan

mengerjakan *request* sebelumnya secara bersamaan (Cleary, 2019), seperti yang diilustrasikan pada gambar 2.6



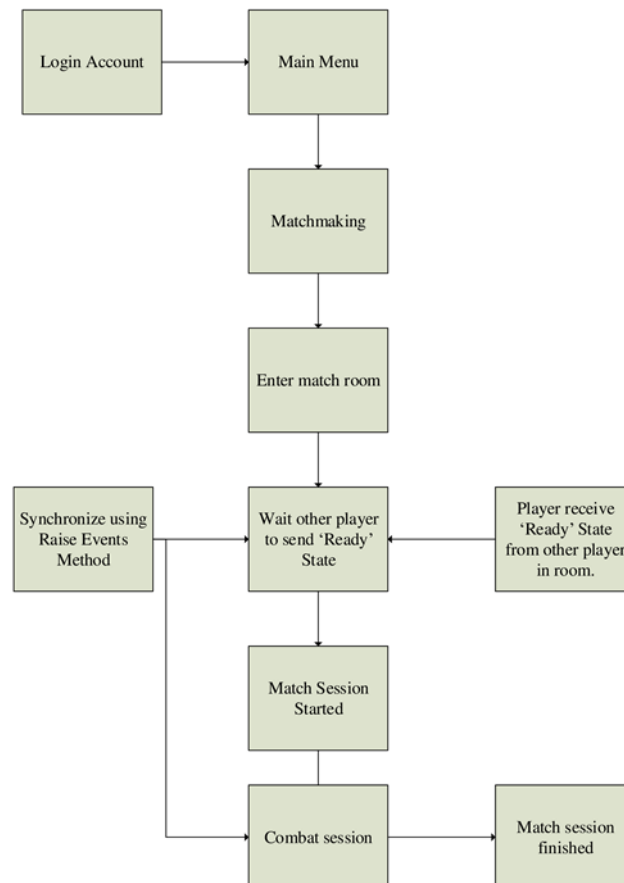
Gambar 2.6 *Multithreading sample*

BAB III

DESAIN DAN IMPLEMENTASI

3.1 Desain Sistem

Berikut Gambar 3.1 Desain sistem yang diterapkan pada penelitian Sinkronisasi Data Komponen *Gameobject* pada *Multiplayer Game* berbasis *Raise Event*.



Gambar 3.1 Desain Sistem

3.1.1. Login Account

Player melakukan aksi *login* akun dengan memasukkan *username*.

Server akan mendaftarkan *username player* apabila *username* tersebut

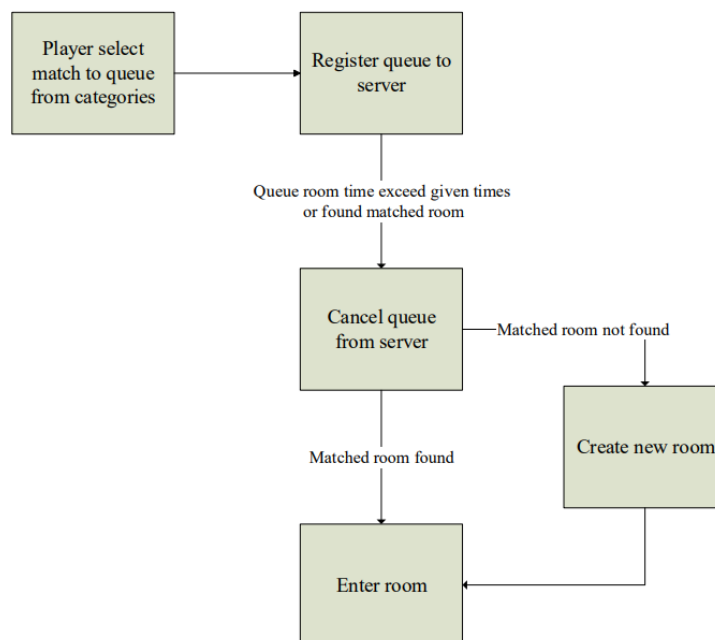
belum terdaftar pada *database*. Setelah proses login berhasil, *player* akan diarahkan menuju *main menu*.

3.1.2. Main Menu

Pada tahap ini, *player* bebas memilih mode permainan yang ingin dimainkan. Setelah menekan mode permainan yang diinginkan, *player* akan masuk pada tahap *matchmaking*.

3.1.3. Matchmaking

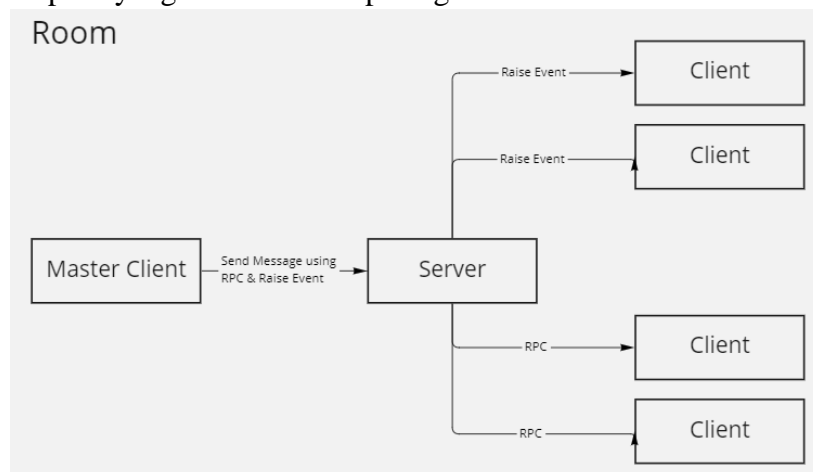
Berikutnya adalah tahap *matchmaking*, seperti yang diilustrasikan pada Gambar 3.2, Pada bagian awal *player* memilih mode *match* yang tersedia, pemilihan mode tersebut mempengaruhi jumlah *player* yang akan bertanding pada satu sesi *match* dengan *player*.



Gambar 3.2 Detail *Matchmaking*

Setelah memilih mode, *player* mengirim sinyal ke *server* agar *player* mulai ‘mengantri’ untuk digolongkan dengan *player* lainnya yang memilih mode serupa. bila *server* berhasil menemukan *room* yang sesuai kriteria *player*, maka *player* akan dimasukkan ke dalam *room* tersebut. bila *server* tidak menemukan *room* yang sesuai kriteria, maka *player* akan dibuat *room* baru lalu menunggu *player* lain yang memiliki kriteria sama dengan *player* untuk masuk ke dalam *room* baru tersebut.

Setelah itu, *server* akan mengakhiri sesi antrian *player* setelah waktu antrian telah melebihi batas atau *server* telah menemukan *room* yang sesuai, atau *server* akan membuat *room* baru bagi *player* bila tidak menemukan *room* yang sesuai kriteria antrian *player*. Pada pembuatan *room*, tiap *player* akan mengirim data menggunakan salah satu dari *RPC* atau *Raise Events*, seperti yang diilustrasikan pada gambar 3.3.



Gambar 3.3 Create Room Illustration

3.1.4. Enter Match Room

Player-player yang telah terseleksi pada mode serupa akan dibuatkan *Room* sebelum memulai permainan. *Player* yang paling lama waktu antriannya akan menjadi *host*. untuk dapat memulai permainan semua *player* dalam *room* harus dalam mode *ready* terlebih dahulu.

3.1.5. Wait other player to send 'Ready' State

Player yang ingin melanjutkan ke sesi permainan harus menunggu *player* lainnya untuk menjadi mode '*ready*' terlebih dahulu. pada tahap ini *player* harus menekan tombol *ready* setelah memastikan semua persiapan telah selesai. agar *player* lain mampu mengetahui bahwa *player* telah masuk ke mode *ready*, maka *player* perlu mengirim data '*ready*' tersebut ke *server*, lalu dari *server* akan mengirimkan ke seluruh *player* yang ada pada *room* tersebut. proses tersebut dinamakan sinkronisasi data. pada *game*, terdapat 2 jenis metode untuk melakukan sinkronisasi data, yaitu metode *RPC* dan *Raise Event*.

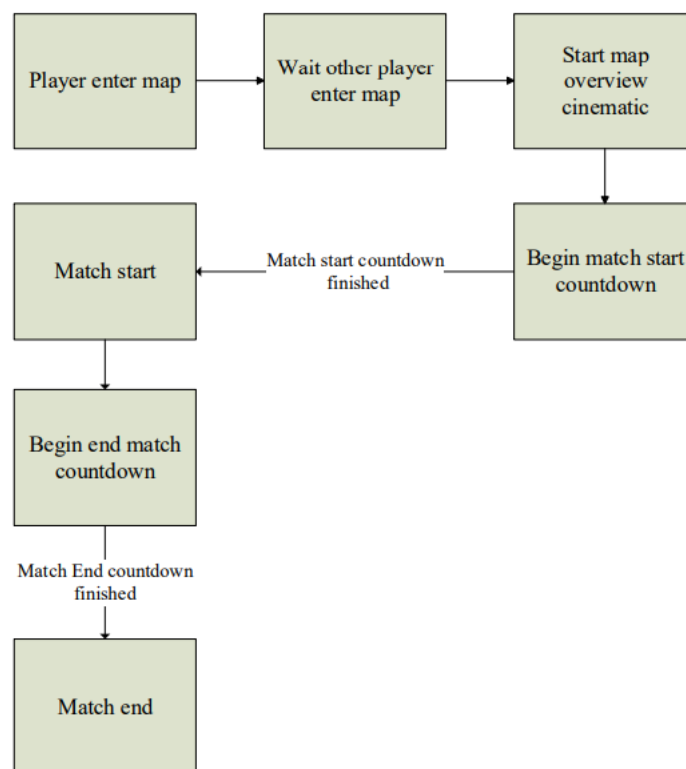
3.1.6. Match Session Started

Setelah semua *player* masuk mode '*ready*', *server* secara otomatis akan memulai pertandingan. pertandingan berlangsung selama durasi menit yang telah ditentukan. selama sesi pertandingan, *player* diperbolehkan menggunakan fitur-fitur dalam *game* untuk mencapai skor tertinggi. *player* harus menemukan pelanggan dan mengantarkannya menuju poin yang telah

ditentukan. selain itu *player* diperbolehkan untuk mengganggu *player* lainnya dengan menggunakan fitur '*combat*'.

3.1.7. *Combat Session*

Pada Gambar 3.4, *server* akan memindahkan *player* ke dalam *map* sesuai pada *room*.



Gambar 3.4 *Combat Session Detail*

Player akan mengirimkan pesan ke *server* apabila telah berhasil masuk ke dalam *map*. Sebelum *match* dimulai, setiap *player* menunggu seluruh anggota *room* untuk masuk ke dalam *map*.

Server akan mengetahui bahwa semua *player* telah berhasil masuk melalui pesan yang dikirim oleh tiap *player* yang berhasil masuk *map*. jika

jumlah pesan sama dengan jumlah *player* pada *room* maka *match* segera dimulai. Sambil menunggu seluruh *player* masuk ke dalam *map*, *player* yang telah masuk *map* akan masuk mode cinematic secara otomatis melihat bentuk *map* secara keseluruhan.

Setelah semua *player* masuk *map*, penumpang akan disebarakan secara acak pada *map*.

Untuk perhitungan durasi *match* dimulai ketika waktu hitung mundur permulaan selesai. tiap detik *server* akan melakukan perhitungan durasi *match* lalu dikirim ke tiap *player* dalam *room*. Pengambilan penumpang terjadi bila *player* mengirim *input* ambil penumpang, lalu sistem akan melakukan pengecekan untuk mengambil penumpang yang paling dekat dengan *player* berdasarkan *pick-up radius* yang telah ditentukan sistem. Hal yang sama juga berlaku untuk proses mengantar dari penumpang, bila *player* berada pada posisi yang telah ditentukan penumpang sebagai tujuan, *player* mengirim *input* untuk menurunkan penumpang, dan sistem akan melakukan validasi bahwa posisi *player* cukup dekat dengan posisi yang menjadi tujuan dari penumpang.

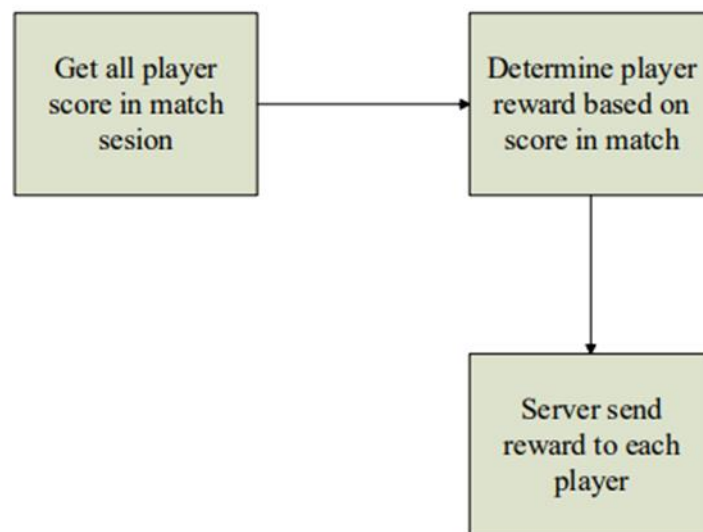
Match akan berakhir ketika durasi *match* habis. selanjutnya *player-player* akan dipersiapkan menuju fase berikutnya.

3.1.8. Match Session Finished

Setelah durasi waktu pertandingan mencapai nol, maka pertandingan akan berakhir. *player* akan menerima notifikasi dari *server*,

lalu akan lanjut ke tahap perhitungan skor. skor dihitung dari banyaknya *player* mengantar pelanggan, menjatuhkan pelanggan dari *player* lain.

Berikut rincian fase pada sesi diatas sesuai yang tertera pada Gambar 3.5, pada bagian *Get All Player Score in Match Session*, *server* mengambil skor semua *player* pada sesi *match* tersebut.



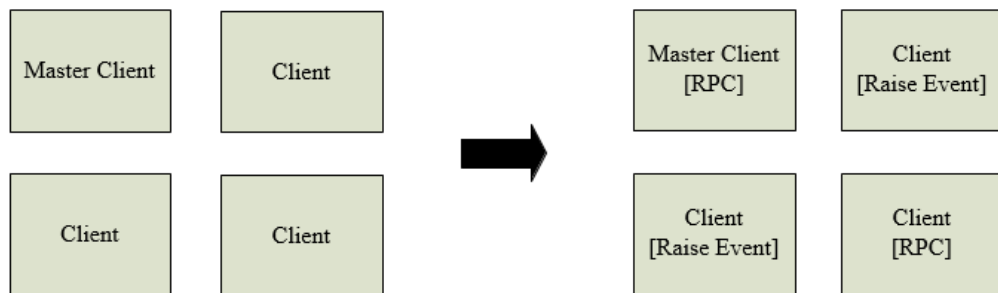
Gambar 3.5 Match Session Finish block diagram flow

Pada tahap *Determine Player Reward Based on Score in Match*, *server* akan mengkalkulasi hadiah untuk tiap *player* sepadan dengan skor yang didapat pada sesi *match*. Lalu hadiah akan dikirimkan ke tiap-tiap *player* seperti pada tahap *Server Send Reward to Each Player*.

3.2 Desain Pengujian

Penelitian ini akan menguji pada sistem pemuatan *map* pada tiap-tiap *client* dan pengiriman paket data perintah pada karakter *player* di dalam *map*. *Master client* atau *room master* akan mengirim paket data menggunakan *RPC* dan *Raise*

Event ke tiap *client*, namun *client* tidak menerima pesan dari kedua metode tersebut, *client* hanya menerima paket data yang dikirim oleh salah satu metode saja, untuk menentukan metode pengiriman yang akan diterima oleh *client*, ditentukan sebelum pemuatan *map* dieksekusi, dibagi secara acak dan merata, seperti yang diilustrasikan pada Gambar 3.6.



Gambar 3.6 Distribusi metode *RPC* atau *Raise Event* pada *Client*.

Pada pengujian pemuatan *map*, *master client* akan mengirim perintah untuk memuat *map* pada semua *client* di *room* menggunakan metode *RPC* dan *Raise Event*, lalu kecepatan keduanya akan dibandingkan. Hal serupa akan dilakukan juga pada pengiriman paket data perintah pada karakter dan pengiriman paket untuk pengambilan atau penurunan penumpang di dalam *map*.

Untuk menghitung kecepatan dilakukan dengan mengambil *timestamp* sebelum fungsi dijalankan ($t1$), lalu akan diambil selisih (*diff*) dari *timestamp* saat selesai dengan *timestamp* saat awal eksekusi ($t2$). *Timestamp* dikirim bersamaan dengan *parameter-parameter* yang dibutuhkan oleh fungsi, berikut tabel 3.1 yang menampilkan tipe data yang digunakan oleh fungsi-fungsi dan besar ukurannya dalam *byte* menurut (Albahari, 2012).

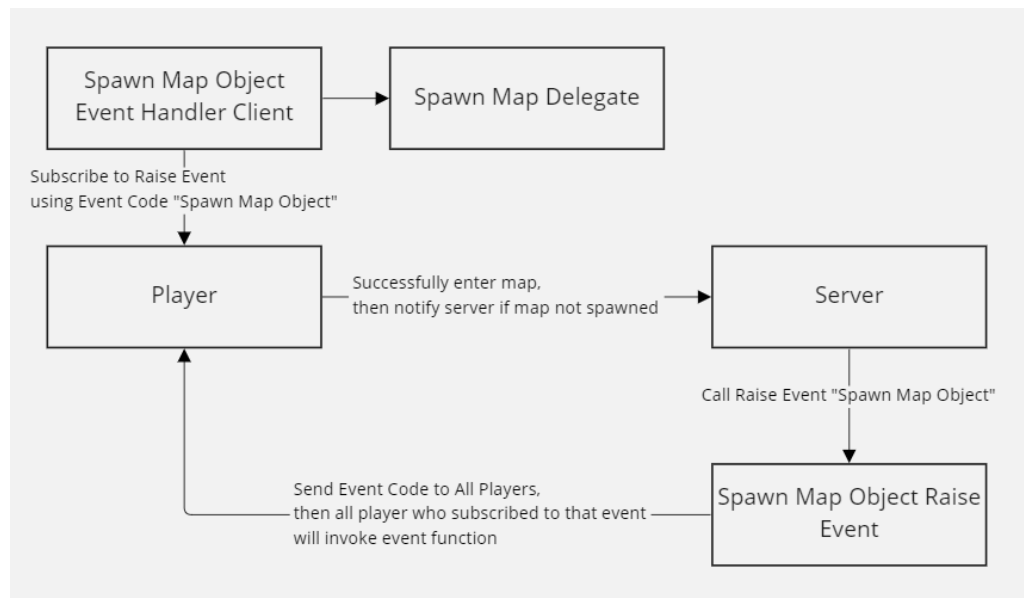
Tabel 3.1 *Data Type Byte Size*

<i>Data Types</i>	<i>Byte Size</i>
<i>Char</i>	2
<i>Integer</i>	4
<i>Bool</i>	1
<i>Float</i>	4
<i>String</i>	(Panjang <i>string</i> x 2)

3.3 *Raise Event Pada Pemuatan Map*

Raise Event akan dieksekusi ketika pada fungsi pemuatan *map*, yang dimaksud pemuatan *map* adalah memunculkan objek yang terdaftar sebagai anggota pada *map* tersebut, hal itu bisa berupa objek bersifat *environment* (mobil, rumah, pohon), ataupun objek *Non-Playable Character (NPC)* yang hanya berfungsi untuk mengisi populasi pada *map* tersebut. *NPC* memiliki logika tersendiri dan diantara mereka ada yang bisa interaksi dengan *player*.

Pada Gambar 3.7 untuk mengilustrasikan proses pemuatan map menggunakan *Raise Event*.



Gambar 3.7 *Spawn Map Raise Event Block Diagram*

Player dari awal terkoneksi dengan *server* sudah *subscribe event* untuk pemuatan *map*, setelah *player* berhasil masuk kedalam *scene map*, *server* akan menjalankan fungsi *Raise Event* yang berisikan kode untuk pemanggilan fungsi *map*, kode tersebut akan dikirimkan ke tiap *player* pada *room*, setelah itu *player* akan menjalankan fungsi pemuatan yaitu *Spawn Map Delegate* itu sendiri. Untuk ukuran data yang dikirim ditentukan dari jenis-jenis *variable* yang dikirim, untuk fungsi *Spawn Map*, *variable* yang dikirim yaitu *Timestamp* yang berupa *string* dan *playerId* yang berupa *integer*, yang merupakan *id* dari pengirim.

$$\text{Ukuran Data} = \text{timestamp} + \text{PlayerId} + \text{UDP header}$$

Timestamp sendiri memiliki format penulisan *yyyy-mm-dd hh:mm:ss.0000000* sehingga panjang *timestamp* yaitu 27 karakter, sementara ukuran satu karakter sendiri menurut *Encoding UTF-16* merupakan 16 bits atau 2

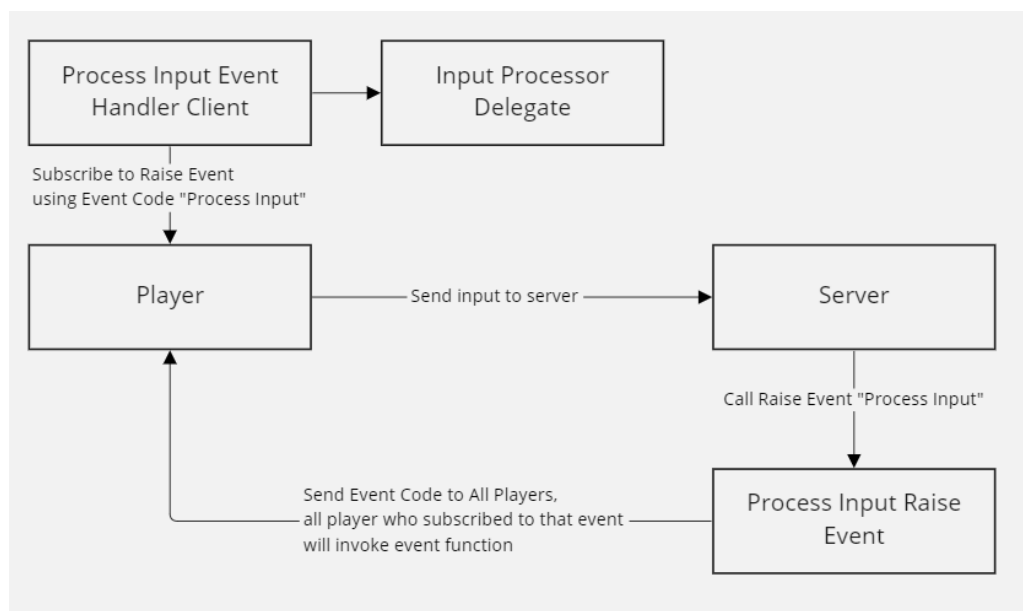
byte. Lalu *PlayerId* merupakan *integer* yang panjangnya sebanyak 4 byte, dan panjang *UDP Header* selalu 8 byte, sehingga dijumlahkan sebagai berikut.

$$\text{Ukuran Data} = (27 \times 2) + 4 + 8$$

$$\text{Ukuran Data} = 54 + 4 + 8$$

$$= 66 \text{ byte}$$

3.4 Raise Event Pada Pengiriman Input Character



Gambar 3.8 Input Character Player Raise Event Block Diagram

Fungsi ini memiliki kesamaan logika seperti fungsi pemuatan *map*, yang membedakan dari fungsi pemuatan *map* adalah *Event Code* yang dikirimkan pada tiap *player* di *room*. Seperti pada Gambar 3.8 untuk mengilustrasikan fungsi *Raise Event* pada pengiriman *input*.

Setiap suatu *player* menggerakkan *character*-nya, sistem secara langsung akan mengirimkan paket data *Raise Event* ke *server* yang nantinya akan dikirimkan lagi dari *server* ke seluruh *player* pada *room* untuk menggerakkan *character* dari *player*

yang mengirimkan paket data tersebut. Untuk ukuran data yang dikirim adalah sebagai berikut.

$$\text{Ukuran Data} = \textit{Timestamp} + \textit{PlayerId} + \textit{Steering} + \textit{Motor} + \textit{Rotx} + \\ \textit{Roty} + \textit{Rotz} + \textit{TravelerCheck} + \textit{header}$$

Definisi :

Timestamp : waktu pengiriman, panjang data 54 *byte*.

PlayerId : *id* dari pengirim, panjang 4 *byte*.

Steering : menunjukkan arah setir kendaraan, bernilai 4 *byte*.

Motor : menunjukkan arah laju kendaraan, bernilai 4 *byte*.

Rotx : rotasi sumbu x dari kendaraan.

Roty : rotasi sumbu y dari kendaraan.

Rotz : rotasi sumbu z dari kendaraan.

TravelerCheck : menunjukkan apakah *player* menekan tombol ambil penumpang, bernilai 1 *byte*.

Header : merupakan panjang dari *header UDP*, memiliki nilai 8 *byte*

Apabila dijumlahkan sebagai berikut.

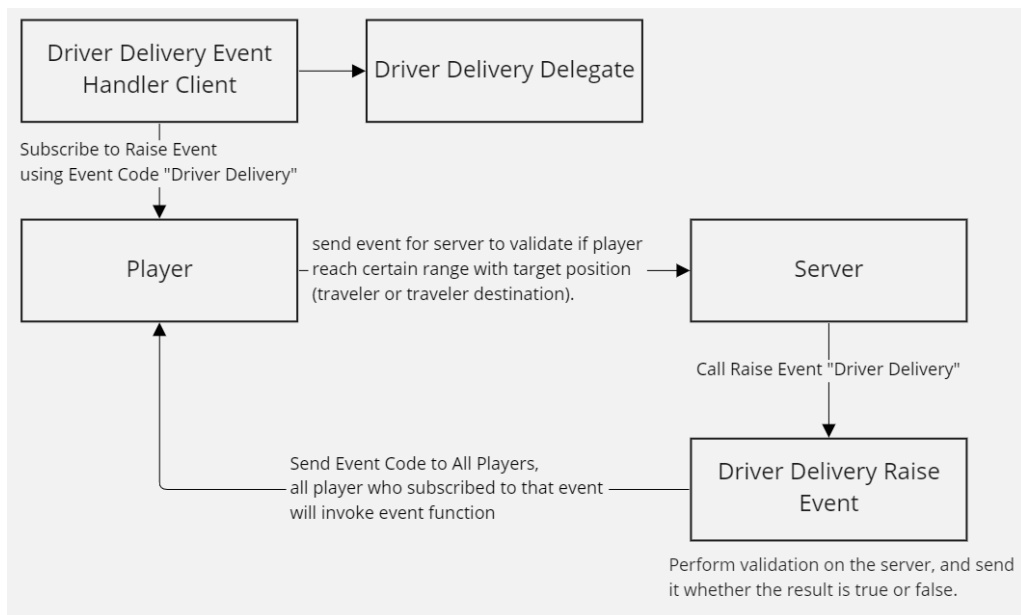
$$\text{Ukuran Data} = (27 \times 2) + 4 + 4 + 4 + 4 + 4 + 4 + 1 + 8$$

$$\text{Ukuran Data} = 54 + 4 + 4 + 4 + 4 + 4 + 4 + 1 + 8$$

$$\text{Ukuran Data} = 87 \text{ bytes}$$

3.5 Raise Event pada Pengambilan dan Penurunan Penumpang

Fungsi ini mengirim *request* untuk validasi pada pengambilan atau penurunan penumpang. Fungsi dikirim dari *player* lalu validasi dilakukan pada *server*. Validasi berupa apakah posisi dari *player* telah sesuai dengan kondisi yang telah ditentukan, *player* dinyatakan akan ‘mengambil’ penumpang apabila *player* sedang tidak membawa penumpang dan posisi *player* berada dalam radius pengambilan penumpang, lalu *player* dinyatakan ‘menurunkan’ penumpang bila *player* sedang membawa penumpang dan berada pada radius yang telah ditentukan dengan posisi tujuan penumpang sebagai porosnya. Berikut gambar 3.9 yang mengilustrasikan fungsi pengiriman dan penurunan penumpang. Selain itu untuk ukuran data yang



Gambar 3.9 Driver Delivery Raise Event

dikirim adalah sebagai berikut.

$$\text{Ukuran Data} = \text{Timestamp} + \text{PlayerId} + \text{TravelerId} + \text{DeliveryCheck} + \text{Header}$$

Definisi :

Timestamp : waktu pengiriman, panjang data 54 *byte*.

PlayerId : *Id* dari pengirim, panjang data 4 *byte*.

TravelerId : *Id* dari penumpang yang dituju, panjang 4 *byte*.

DeliveryCheck : Pengecekan apakah penumpang untuk diambil atau turun, panjang 1 *byte*.

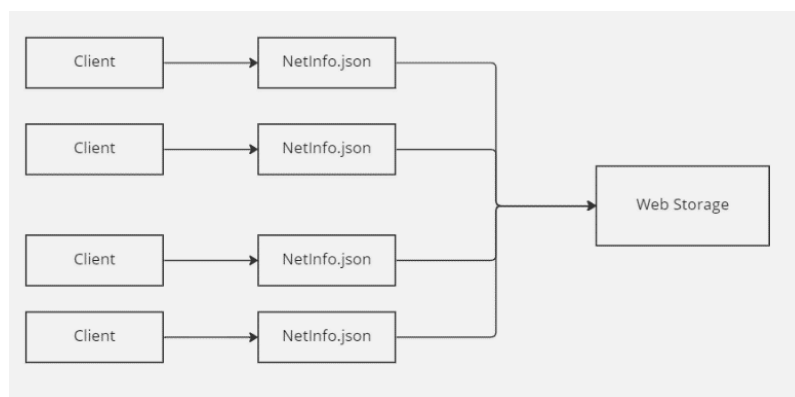
Apabila dijumlahkan maka akan bernilai sebagai berikut.

$$\text{Ukuran Data} = 54 + 4 + 4 + 1 + 8$$

$$\text{Ukuran Data} = 71 \text{ byte}$$

3.6 Proses Merangkum Data

Setiap fungsi-fungsi diatas dijalankan oleh *player*, sistem akan mencatat dan merangkum secara otomatis detail fungsi-fungsi tersebut dan nantinya akan dikirim pada penyimpanan *website* dalam bentuk *json*. Berikut gambar 3.10 tentang proses merangkum data.



Gambar 3.10 Data Collection Concept

Tiap *player* akan mengirim *file NetInfo.json* yang berisi *match id*, *ip address* dan *player id* dan semua fungsi-fungsi *Raise Event* dan *Remote Procedure Call* yang dipanggil oleh *player* tersebut. *NetInfo.json* tersebut akan disimpan pada penyimpanan *website* yang telah disiapkan. Untuk rangkuman seluruh data tersebut diambil menggunakan *downloader* khusus seperti yang ditampilkan pada gambar 3.11 lalu dirubah menjadi bentuk *Comma Separated Values* atau *CSV* sehingga bisa diolah lagi menggunakan *software spreadsheet* seperti *Microsoft Excel*, *Open Office Calc*, dan sebagainya.



Gambar 3.11 *NetInfo Downloader Interface*

3.7 Implementasi *Design Pattern* pada *Game*

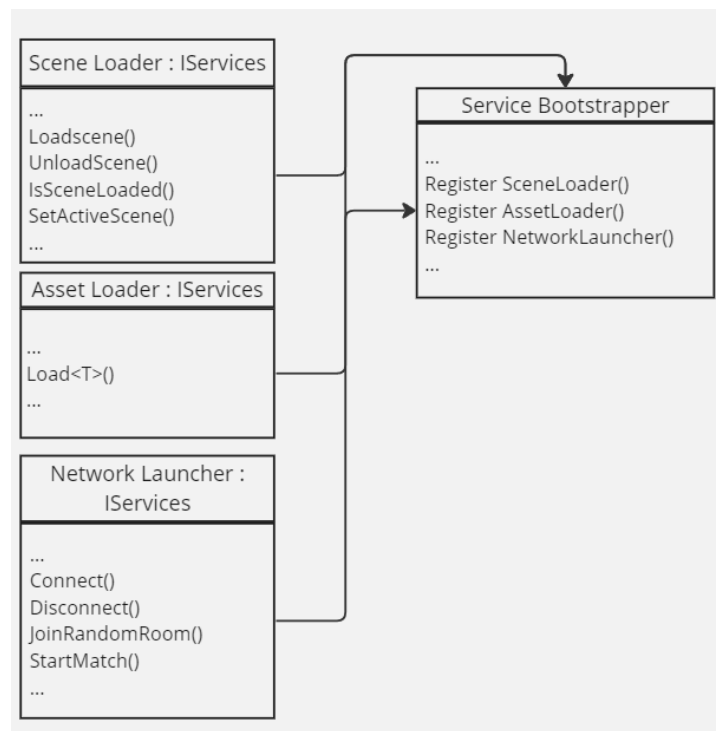
Seperti yang telah dijelaskan pada bab sebelumnya, *game* mengimplementasikan beberapa *design pattern*, salah satunya yaitu *Service Locator*. *Service Locator* bertugas untuk sebagai perantara suatu sistem pada sistem lain, umumnya *Service Locator* memiliki 2 metode yang sering digunakan yaitu

metode *Register* dan *Get* seperti yang ditampilkan pada gambar 3.12 tentang kelas *Service Locator*.

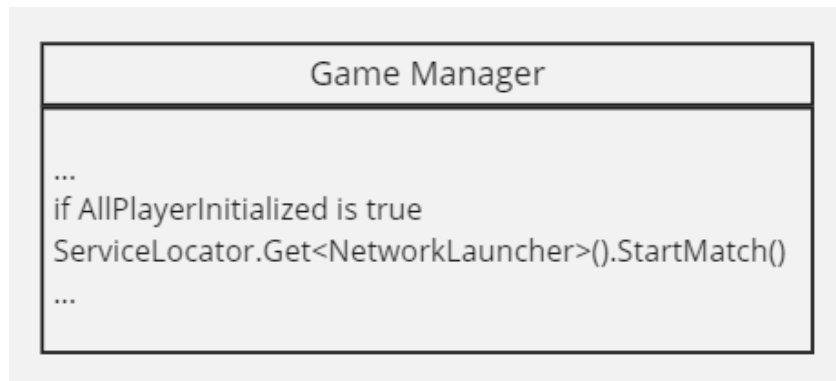


Gambar 3.12 *Service Locator Class*

Fungsi *register* akan mendaftarkan suatu kelas yang mengimplementasi interface penanda penyedia layanan (*service provider*) yaitu *IServices*, Registrasi dari kelas tersebut cukup dilakukan sekali saat tiap aplikasi dibuka, seperti yang diilustrasikan pada gambar 3.13.

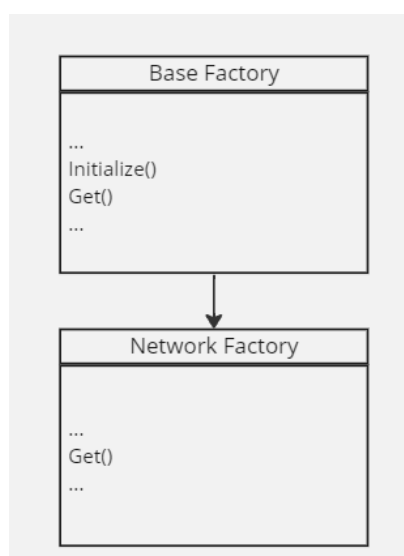


Gambar 3.13 *Service Provider Registration*

Gambar 3.14 *Start Match Pseudocode*

Selanjutnya untuk pemanggilan kelas *service provider* bisa diakses melalui metode dari kelas *service locator* yaitu metode *Get*. Sebagai contoh pada gambar 3.14, dimana ditampilkan *pseudocode* pada kelas *GameManager* pada metode *StartMatch* milik kelas *NetworkLauncher* akan dijalankan Ketika semua *player* telah terkoneksi dan masuk pada *room*.

Selain *Service Locator*, *game* juga mengimplementasi *design pattern* lainnya yaitu *Factory Pattern*. *Factory pattern* diwujudkan pada sistem pemanggilan dari fungsi-fungsi yang berhubungan dengan *Raise Event* maupun *RPC*, seperti yang

Gambar 3.15 *Network Factory Inheritance*

ditampilkan pada gambar 3.15. lalu pada gambar 3.16 ditampilkan bentuk *pseudocode* dari fungsi *Initialize* dimana kelas yang mengimplementasi *interface* khusus yang menandakan sebagai anggota dari *factory* akan dianggap sebagai bagian dari *factory* tersebut.

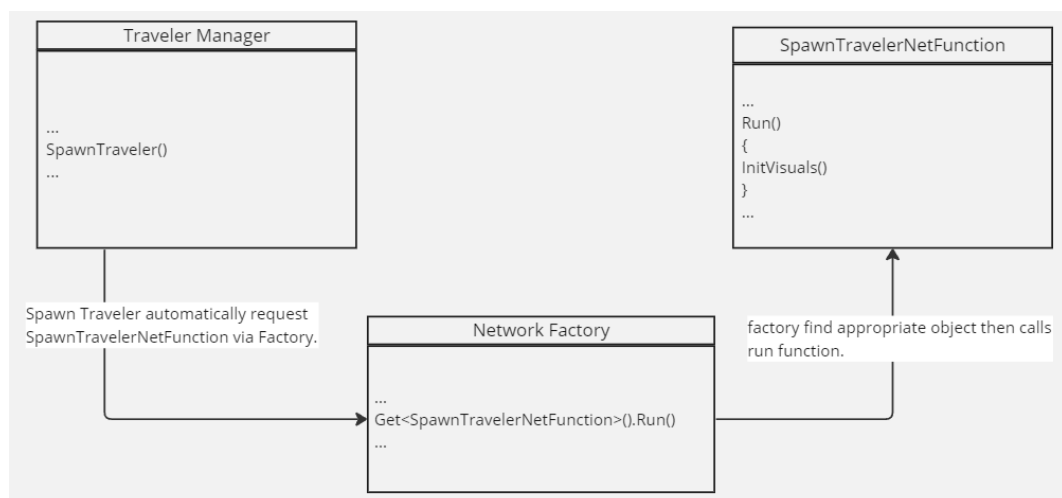
```

...
foreach class isSubClassOf FactoryMember
add class as Member
...

```

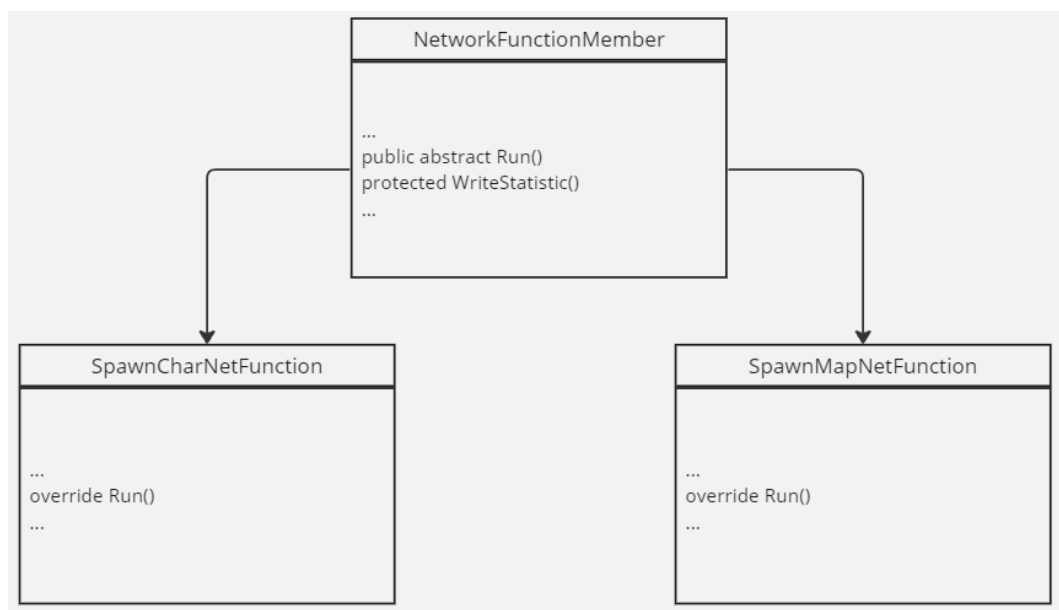
Gambar 3.16 *Factory Run Call Illustration*

Selanjutnya apabila sistem membutuhkan *object* dari *factory* tersebut, sistem perlu memanggilnya menggunakan fungsi *Get*, seperti yang digambarkan pada gambar 3.17, class *TravelerManager* berhasil memunculkan *object* dari *traveler*, langkah selanjutnya adalah memberitahu seluruh *remote client* untuk memunculkan *visual* dari *traveler*, hal itu dilakukan oleh *object SpawnTravelerNetFunction* yang telah dimunculkan melalui *factory*.



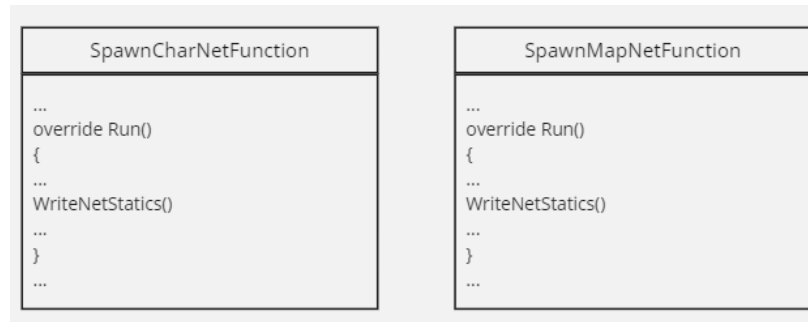
Gambar 3.17 *Factory Registration Pseudocode*

Pemanggilan *factory* pada fungsi *traveler* merupakan Sebagian contoh kecil dari penggunaan *factory pattern*, selain pada fungsi *traveler*, *factory* juga dimanfaatkan untuk fungsi *raise event* lainnya, seperti pemanggilan karakter, *environment map*, sinkronisasi *timer*, dan sebagainya. *Design Pattern* berikutnya yang diimplementasi adalah *Subclass Sandbox*. *Design Pattern Subclass Sandbox* diimplementasi pada beberapa *subclass* dari fungsi-fungsi *network* yang terdapat pada *factory*. Gambar 3.18 menampilkan 2 fungsi yang mengimplementasikan *pattern subclass sandbox*, kedua kelas tersebut adalah *SpawnCharacterNetFunction* dan *SpawnMapNetFunction*, dimana fungsi *Run* merupakan bentuk *sandbox* dan fungsi *WriteNetStatistics* merupakan fungsi pendukung dari *sandbox*.



Gambar 3.18 *Subclass Sandbox Implementation*

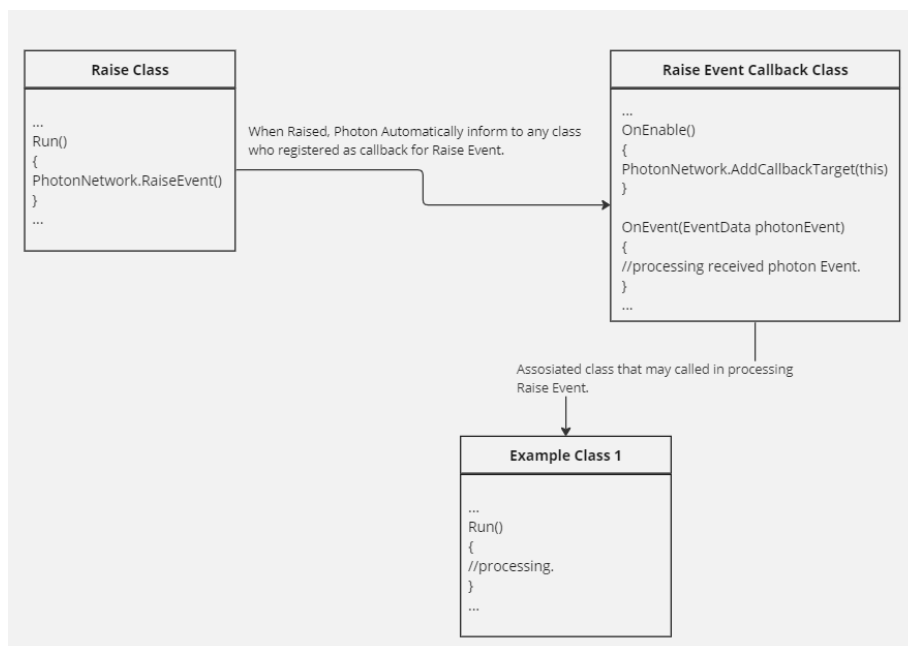
Fungsi dari *WriteNetStatistics* adalah memanggil fungsi *logging* kelas *logger* yang akan dikirim pada *web* ketika *match* telah selesai, seperti yang ditampilkan pada gambar 3.19.



Gambar 3.19 Call Net Statistics Function

3.8 Sistem Photon Network Handler

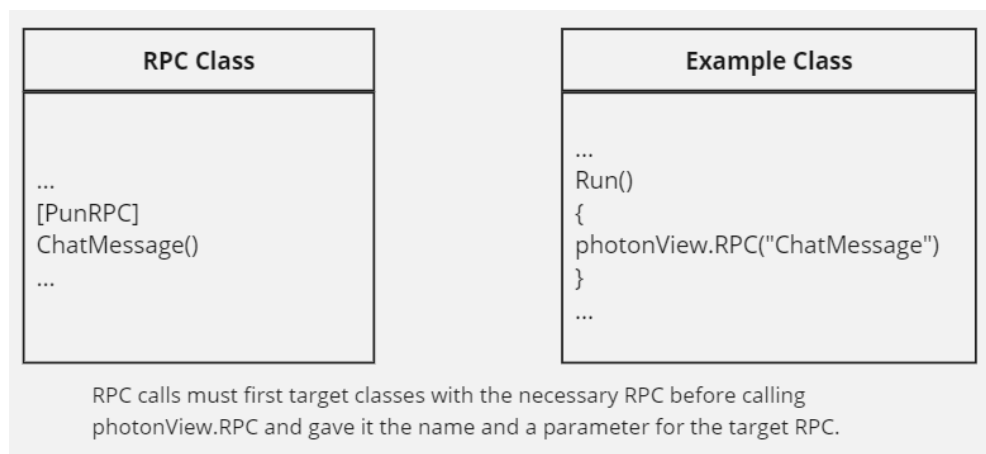
Pada penelitian ini, *game* menggunakan kedua fitur pengiriman data antar *player* dari *photon*, yaitu *Raise Event* dan *Remote Procedure Call* atau *RPC*. Selain dari *framework photon PUN* sendiri, kedua fungsi diatas juga perlu didesain implementasinya sehingga mempermudah dalam memprogram fitur-fitur lainnya



Gambar 3.20 Raise Event pseudocode flow

maupun dalam pembacaan kode pada tiap fitur didalam *codebase*. Berikut gambar 3.20 yang menampilkan alur *pseudocode Raise Event*.

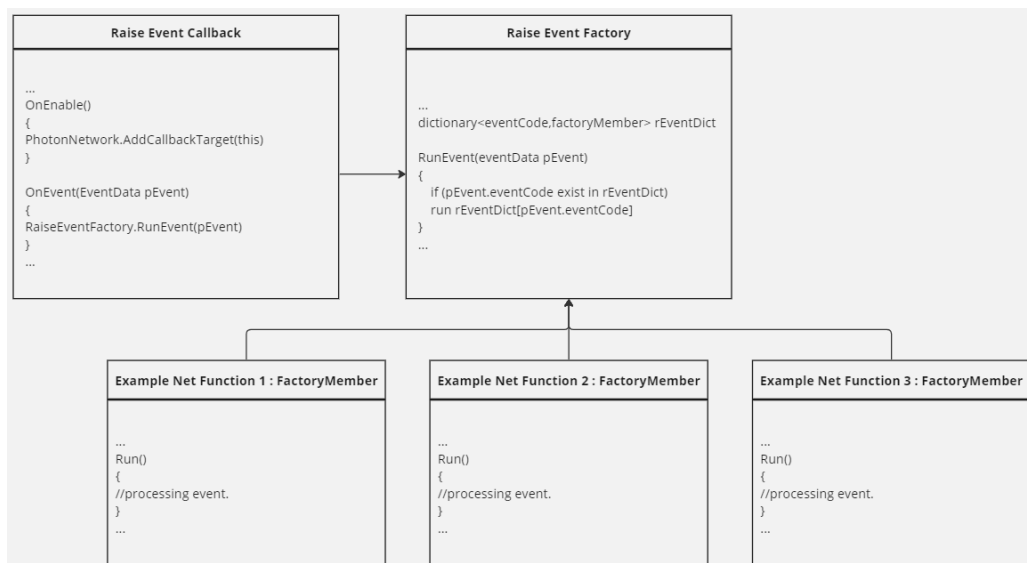
Pada gambar 3.20, *Raise Event* dijalankan saat suatu kelas memanggil fungsi *Raise Event*, lalu *Raise Event* akan mengirim *event* tersebut kepada seluruh kelas yang telah terdaftar sebagai kelas yang menerima *event*, dan setelah kelas tersebut menerima *event*, maka selanjutnya akan memproses *event* tersebut sesuai kebutuhan. Hal yang serupa juga berlaku untuk sistem *RPC*, seperti yang ditampilkan pada gambar 3.21.



Gambar 3.21 *RPC flow*

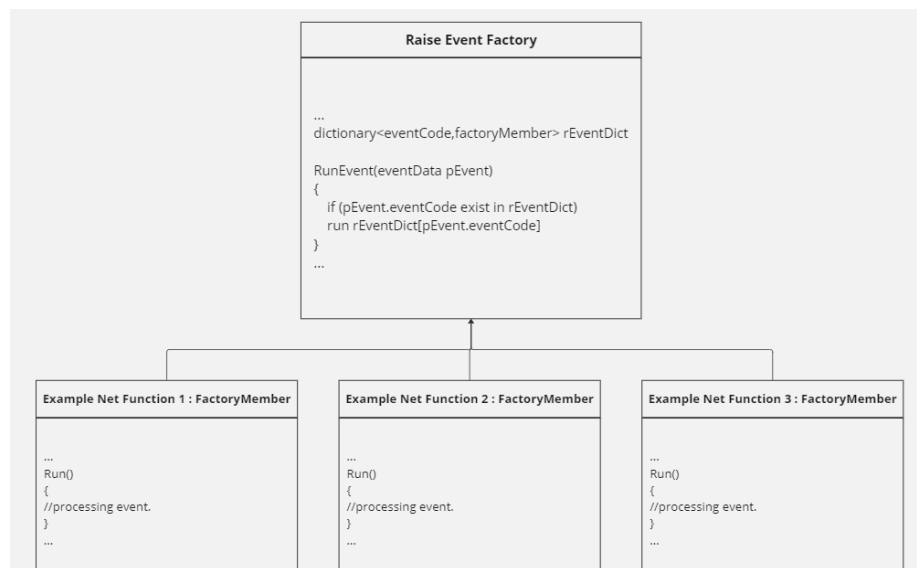
Pada gambar 3.21, fungsi *RPC* harus ditandai dengan *[PunRPC]* diatas nama fungsi sebelum bisa diproses oleh *photon*, perlu diketahui juga kelas *RPC* juga harus memiliki komponen *PhotonView* sebelum bisa menggunakan fitur *RPC*. selanjutnya pada kelas lain perlu mereferensi *PhotonView* dari kelas yang ingin dipanggil fungsi *RPC* nya. Kedua fungsi tersebut, *Raise Event* dan *RPC* mengisyaratkan perlu banyak kelas yang dimana tiap kelas tersebut perlu mengikuti persyaratan penggunaan dari *Raise Event* dan *RPC* agar fitur bisa digunakan.

Pada penelitian ini, didesain sebuah sistem yang menyederhanakan persyaratan tersebut, sehingga hanya perlu sekali pemanggilan persyaratan tersebut, sehingga memudahkan pemrograman fitur-fitur pada *game* kedepannya, seperti yang ditampilkan pada gambar 3.22, dimana terdapat suatu *factory* yang berisi kumpulan kelas-kelas yang dijalankan menggunakan *Raise Event*, tiap kelas tersebut diberi identitas sesuai dengan *event code* dari *Raise Event*, dimana nanti untuk pemanggilan kelas-kelas tersebut melalui *factory* dengan menyebut *event code* dari kelas tersebut. Dengan desain sistem yang seperti ini, komponen *PhotonView* hanya diperlukan untuk kelas penerima *Raise Event*, dari kelas itu bisa meneruskan *Raise Event* yang kemudian diterima dan diproses oleh *factory*.

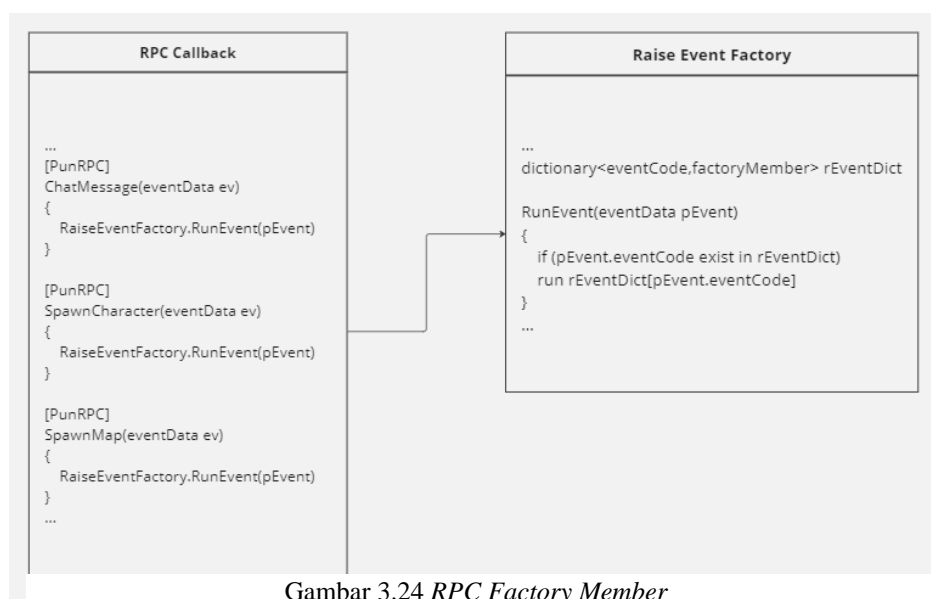


Gambar 3.22 *Raise Event Handler*

Selain pada *Raise Event*, fungsi *RPC* juga didesain serupa dengan *Raise Event*, dengan menyediakan satu kelas yang akan memproses perintah *RPC*, dimana tiap fungsi yang ditandai dengan [*PunRPC*] akan berjalan dengan melalui *factory*, seperti pada gambar 3.23, dimana fungsi-fungsi sebelumnya telah didaftarkan pada kelas *factory*, seperti yang ditampilkan pada gambar 3.24.



Gambar 3.23 *Call via Factory*



Gambar 3.24 *RPC Factory Member*

BAB IV

UJI COBA DAN PEMBAHASAN

Pada bab ini menjelaskan implementasi dan evaluasi terhadap sistem yang telah dirancang sebelumnya. Hal tersebut dilakukan untuk mengetahui kecepatan kirim dari metode *Raise Event* dan metode *Remote Procedure Call*.

4.1 Implementasi

Tahap ini berfokus implementasi dari desain sistem yang telah dibuat. Pada bagian ini juga akan menampilkan tahapan proses yang dijalankan pada *game*.

4.2 Perangkat Lunak yang digunakan

Berikut perangkat lunak yang digunakan untuk pengembangan *game*.

<i>Operating System</i>	<i>Windows 10 Pro 64-bit</i>
<i>Game Engine</i>	<i>Unity 3D 2020.3.20f1</i>
<i>Text Editor</i>	<i>Microsoft Visual Studio Community 2019</i>
<i>Image Editor</i>	<i>Adobe Photoshop CC 2015</i>
<i>3D Modeller</i>	<i>Blender 3D v2.8</i>

4.3 Spesifikasi Perangkat Keras

Terdapat 4 perangkat keras yang digunakan dalam pengujian penelitian, hal tersebut mengingat bahwa *game multiplayer* membutuhkan lebih dari satu perangkat untuk mengetahui apakah *game* berjalan dengan baik atau tidak.

Keempat perangkat keras tersebut memiliki spesifikasi yang sama yaitu sebagai berikut :

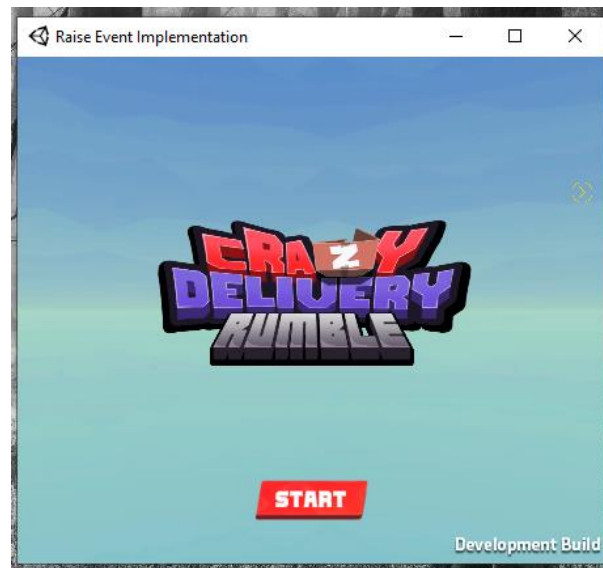
<i>Type</i>	<i>Personal Computer</i>
<i>Operating System</i>	<i>Windows 10 Pro 64-Bit</i>
<i>CPU</i>	<i>Intel Core i7 9700</i>
<i>GPU</i>	<i>NVIDIA Geforce 1050 Ti</i>
<i>RAM</i>	<i>16 Gb</i>

4.4 Implementasi Desain Pengujian

Pada tahap ini akan menjalankan *game* dengan menerapkan desain pengujian yang telah dibuat.

4.4.1 Main Menu

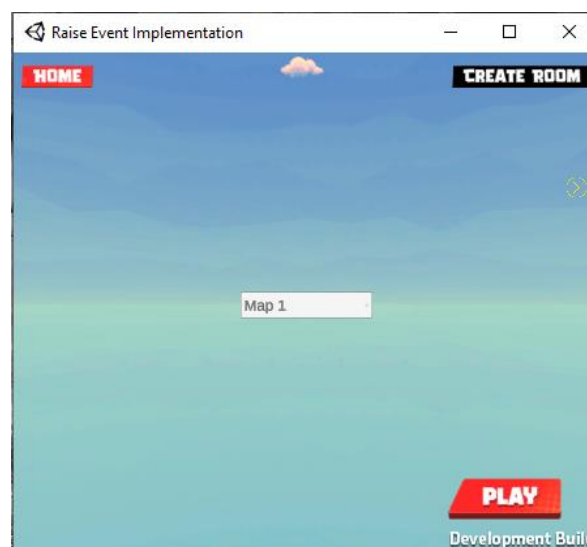
Tahap ini merupakan tahap awal ketika *game* dibuka, klik pada tombol *start* untuk transisi pada bagian pemilihan *map*. Pada saat *player* menekan tombol *start*, *game* secara otomatis mengkoneksikan *player* terhadap *server* utama dari *Photon Unity Network 2*. Selain itu untuk mempermudah uji coba *input username* untuk *player* dilakukan otomatis pada setelan *default* pada *game*. Berikut gambar 4.1 yang menampilkan *main menu* pada *game*



Gambar 4.1 *Main Menu Interface*

4.4.2 *Map Selection*

Setelah menekan tombol *start*, tampilan *main menu* akan berubah menjadi tampilan *map selection* seperti pada gambar 4.2.



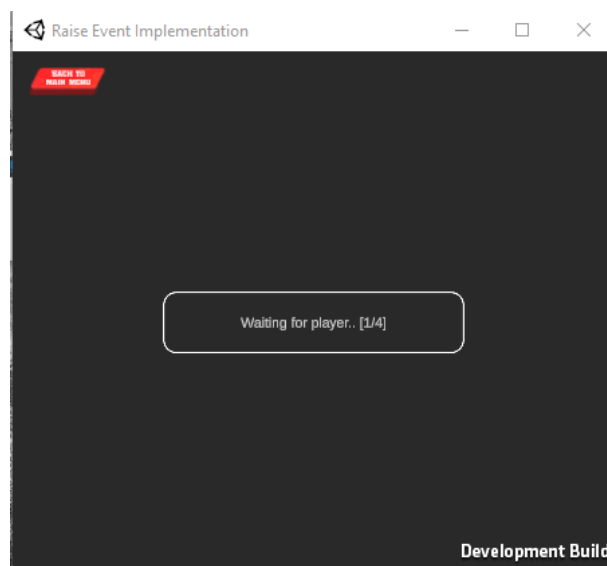
Gambar 4.2 *Map Selection Interface*

Player bisa memilih *map* yang tersedia dengan menekan *input dropdown* yang terdapat pada bagian tengah layar, setelah memilih *map* yang

diinginkan, *player* bisa melanjutkan menuju tahap *matchmaking* dengan menekan tombol *play*.

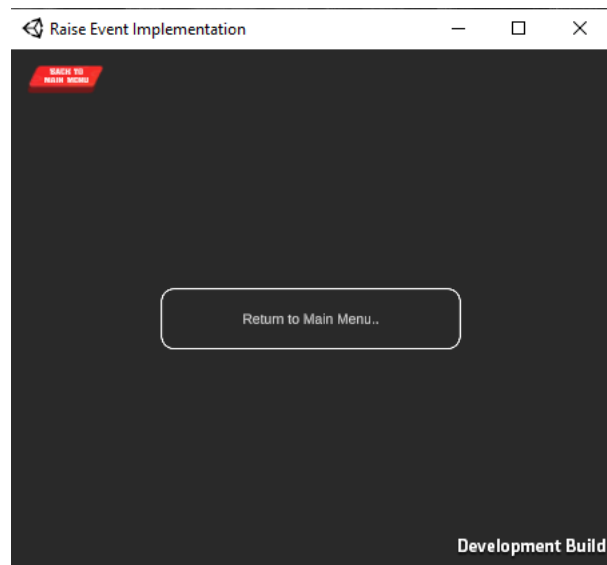
4.4.3 Matchmaking

Selanjutnya pada tahap *matchmaking*, *player* akan menunggu *player* lain yang memilih map serupa untuk bisa masuk dalam satu sesi permainan seperti yang diilustrasikan pada gambar 4.3.



Gambar 4.3 Matchmaking Interface

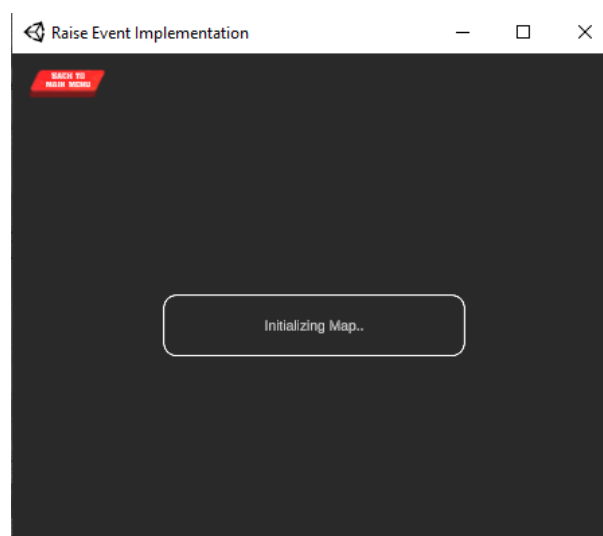
Player juga bisa membatalkan proses *Matchmaking* dengan menekan tombol *Back to Main Menu* pada pojok kiri atas, menekan tombol tersebut selain membatalkan *matchmaking* juga akan mengirim *player* kembali ke *Main Menu*. tampilan pembatalan *matchmaking* diilustrasikan pada gambar 4.4.



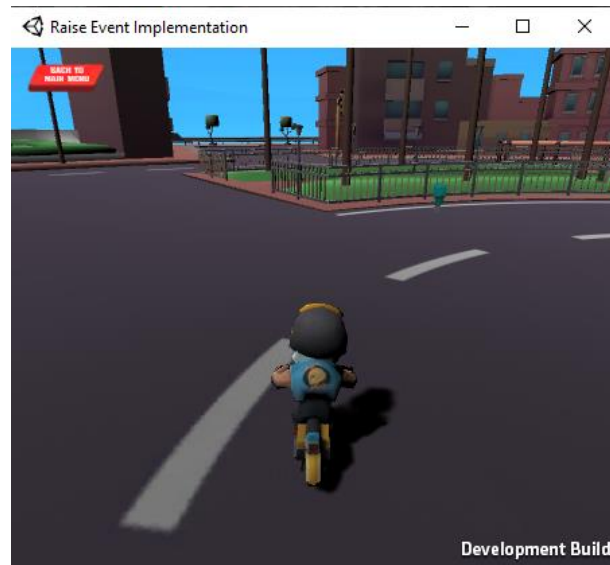
Gambar 4.4 *Cancel Matchmaking Interface*

4.4.4 *Load Map*

Setelah jumlah *player* telah memenuhi syarat untuk memulai *match*, maka tahap selanjutnya adalah memunculkan *gameobject* yang berisi lokasi *match* dan karakter dari tiap-tiap *player* dalam satu *room*. Pada gambar 4.5



Gambar 4.5 *Initializing Map Interface*

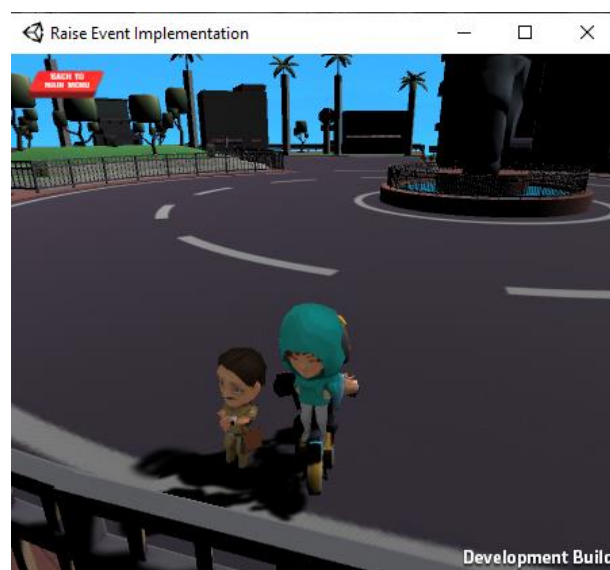


Gambar 4.6 *Map Loaded Interface*

menunjukkan proses menunggu dari pemanggilan *map*, sementara pada gambar 4.6 menggambarkan pemanggilan *map* yang telah selesai.

4.4.5 *Load Traveler*

Pada tiap *interval* tertentu, sistem akan memanggil dan menyebar penumpang atau *traveler* sebagai *objective* dari *player*. *Player* ditugaskan untuk



Gambar 4.7 *Pick & Drop Traveler Interface*

mengambil dan mengantar menuju lokasi yang telah ditentukan oleh *traveler*. Seperti yang diilustrasikan oleh gambar 4.7.

4.5 Pengujian Rata-rata kecepatan kirim data pada *Function*

Pengujian dilakukan untuk mengetahui kecepatan kirim data dari fungsi *Raise Event* dan *Remote Procedure Call* yang telah disiapkan pada desain pengujian. Data penggunaan dari fungsi-fungsi tersebut telah dirangkum dalam bentuk *Comma Separated Values* atau *CSV* sehingga bisa diproses lebih lanjut, Adapun data untuk *Player* yang menggunakan metode *Raise Event* tersebut bisa dilihat pada tabel 4.1 berikut.

Tabel 4.1 Rata-rata kecepatan *player* menggunakan *Raise Event*

No	Match Id	Player ID	Rata-rata kecepatan (ms)		
			SpawnMap	SendInput	DriverDelivery
1	a6e82c36-3fab-4a56-8b22-24a5f8504037	Player 1	71.1129	59.2519963	57.7409
		Player 3	71.1136	43.83481478	57.7409
2	e57839c3-055c-494a-bd62-d7e1dde8eec4	Player 1	46.2031	63.99116154	57.7297
		Player 3	15.9643	30.07018795	57.7297
3	99a1c6f9-27e7-455e-8c91-197f616faf51	Player 1	30.5171	62.73867115	70.433975
		Player 3	16.7811	34.73926722	67.014925
4	d72e4acd-cda5-4e5a-bdf0-3c5ce58fa816	Player 1	73.0409	67.69292708	68.73111667
		Player 3	73.0405	39.16766831	55.18736667
5	1d55a08c-a40d-44c1-b210-d5a8081b7794	Player 1	43.0178	57.82254359	68.7033
		Player 3	43.0172	28.89204444	68.7033

No	Match Id	Player ID	Rata-rata kecepatan (ms)		
			SpawnMap	SendInput	DriverDelivery
6	7b90d13f-16e0-4cea-8196-4b51424610ff	Player 1	59.001	64.805935	61.980175
		Player 3	59.081	8.937924803	61.980175
7	45eac943-d342-4ab7-84a3-cc5f231cdfdd	Player 1	50.7645	60.60257083	65.96328
		Player 3	5.7624	0.001192479	65.96328
8	cc76c924-9829-49a7-9891-6abcdf596c17	Player 1	82.8411	58.990466	53.85574
		Player 3	14.1861	35.921939	53.85574
9	ef337950-d042-4253-bd56-9c3e481195a5	Player 1	43.7739	57.10435128	63.0363
		Player 3	12.7275	32.28295047	63.0363
10	b9d23542-042f-4110-9c4e-8902030c677a	Player 1	55.4317	72.66019556	66.466925
		Player 3	55.4281	34.71561237	66.466925

Selain itu, berikut tabel 4.2 yang menampilkan data *player* yang menggunakan metode *Remote Procedure Call*.

Tabel 4.2 Rata-rata kecepatan *player* menggunakan *Remote Procedure Call*

No	Match ID	Player ID	Rata-rata kecepatan kirim (ms)		
			SpawnMap	SendInput	DriverDelivery
1	a6e82c36-3fab-4a56-8b22-24a5f8504037	Player 2	11.0441	64.33762382	15.051
		Player 4	2.9482	151.5259843	10.03028
2		Player 2	11.5221	11.35745344	9.2102

No	Match ID	Player ID	Rata-rata kecepatan kirim (ms)		
			SpawnMap	SendInput	DriverDelivery
	e57839c3-055c-494a-bd62-d7e1dde8eec4	Player 4	16.0322	156.760408	10.762225
3	99a1c6f9-27e7-455e-8c91-197f616faf51	Player 2	6.7024	6.696895133	11.2506
		Player 4	15.83	161.743772	9.2644
4	d72e4acd-cda5-4e5a-bdf0-3c5ce58fa816	Player 2	7.6729	42.96813977	10.51174
		Player 4	4.3203	158.3967237	9.3239
5	1d55a08c-a40d-44c1-b210-d5a8081b7794	Player 2	5.5143	11.34329239	9.5639
		Player 4	1.1295	156.9354396	8.0133
6	7b90d13f-16e0-4cea-8196-4b51424610ff	Player 2	2.7916	28.5692061	9.074575
		Player 4	1.1154	160.8609263	9.887566667
7	45eac943-d342-4ab7-84a3-cc5f231cdfdd	Player 2	9.0494	11.69302522	8.34785
		Player 4	14.5846	150.2283667	7.676933333
8	cc76c924-9829-49a7-9891-6abcdef596c17	Player 2	7.8699	12.61541075	11.86715
		Player 4	15.3569	161.040529	10.54665
9	ef337950-d042-4253-bd56-9c3e481195a5	Player 2	12.3259	20.60035045	9.083883333
		Player 4	3.8699	154.3167948	10.06637143
10		Player 2	11.8195	36.93349279	5.146166667

No	Match ID	Player ID	Rata-rata kecepatan kirim (ms)		
			SpawnMap	SendInput	DriverDelivery
	b9d23542-042f-4110-9c4e-8902030c677a	Player 4	9.1894	160.7818944	4.6714

Selanjutnya adalah rata-rata kecepatan seluruh fungsi yang dijalankan oleh *Raise Event* dan *Remote Procedure Call* yang akan ditampilkan pada tabel 4.3 berikut.

Tabel 4.3 *Average Call Speed*

Method	Average (ms)
<i>Raise Event</i>	52.88506193
<i>Remote Procedure Call</i>	77.82238571
<i>Difference</i>	24.93732378
<i>Percentage</i>	47.1538140849377%

Dari tabel diatas diketahui *Raise Event* memiliki rata-rata kecepatan panggilan lebih sedikit nilainya jika dibandingkan dengan *Remote Procedure Call*.

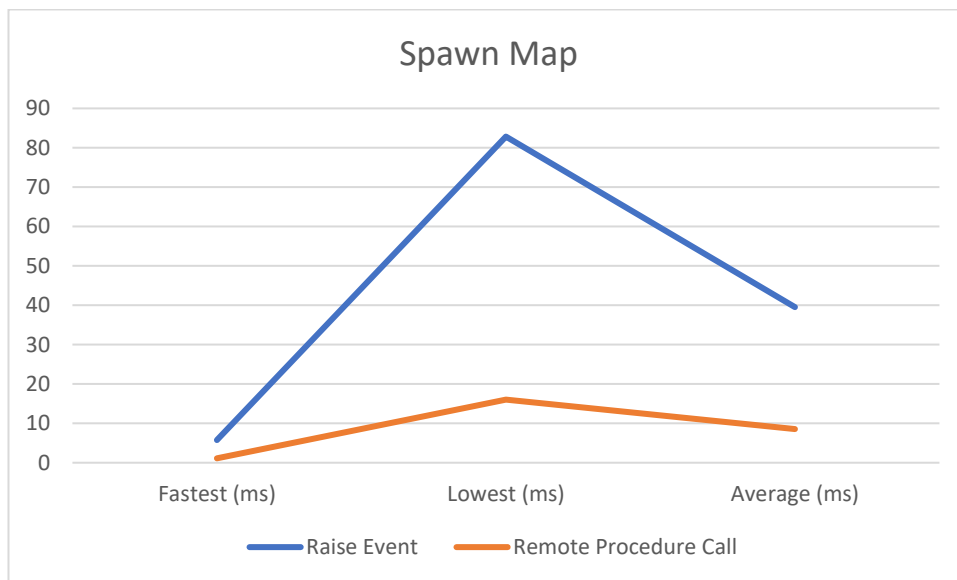
4.6 Pengujian Fungsi *Spawn Map*

Pengujian berikutnya adalah melihat waktu eksekusi tercepat (*Fastest*), waktu eksekusi terlama (*Slowest*), rata-rata kecepatan (*Average*), ukuran data yang dikirim (*Size*) dan berapa banyak fungsi dipanggil (*Calls*). Hal ini bertujuan untuk melihat bagaimana performa metode *Raise Event* dan *Remote Procedure Call* dalam mengeksekusi fungsi *Spawn Map*. Berikut tabel 4.4 yang menunjukkan performa metode *Raise Event* dan *Remote Procedure Call* dalam memproses fungsi *Spawn Map*.

Tabel 4.4 *Spawn Map Analysis*

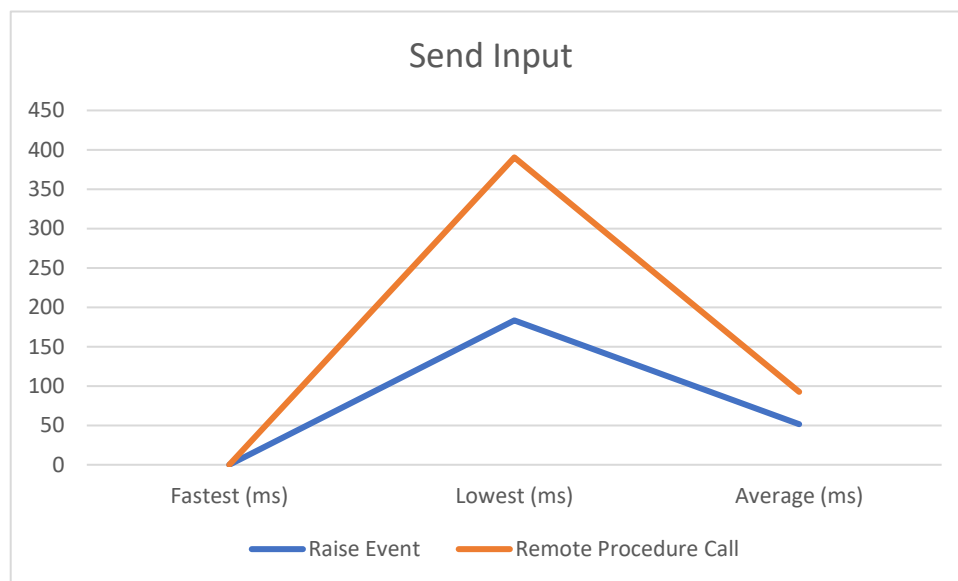
<i>Method</i>	<i>Size (Byte)</i>	<i>Calls</i>	<i>Fastest (ms)</i>	<i>Slowest (ms)</i>	<i>Average (ms)</i>
<i>Raise Event</i>	78	26	5.7624	82.8411	39.529088
<i>Remote Procedure Call</i>	78	21	1.1154	16.0322	8.534425

Pada tabel diatas menunjukkan, dengan ukuran data yang sama, *Remote Procedure Call* memiliki waktu tercepat dalam eksekusi fungsi *Spawn Map*, *Remote Procedure Call* juga memiliki catatan waktu terlama yang lebih sedikit dibandingkan dengan *Raise Event* untuk menjalankan *Spawn Map*. sementara itu *Raise Event* memiliki jumlah pemanggilan yang lebih banyak dibandingkan *Remote Procedure Call*. Berikut gambar 4.8 yang menunjukkan grafik dari hasil pengujian diatas.

Gambar 4.8 *Spawn Map Chart*

4.7 Pengujian Fungsi Send Input

Selanjutnya pengujian dilakukan pada fungsi *Send Input*, seperti halnya pada fungsi *Spawn Map*, pengujian ini dilakukan untuk melihat waktu tercepat, terlama, rata-rata waktu yang digunakan, jumlah pemanggilan fungsi dan jumlah data yang dikirim oleh fungsi *Send Input*. Berikut hasil pengujian yang ditampilkan oleh tael 4.5.



Gambar 4.9 *Send Input Chart*

Tabel 4.5 *Send Input Analysis*

<i>Method</i>	<i>Size (Byte)</i>	<i>Calls</i>	<i>Fastest (ms)</i>	<i>Slowest (ms)</i>	<i>Average (ms)</i>
<i>Raise Event</i>	87	723	0.001192187	183.3307	51.40742197
<i>Remote Procedure Call</i>	87	761	0.001100787	390.3479	92.68621208

Raise Event memiliki catatan waktu terlama lebih banyak dibanding dengan *Remote Procedure Call*. Untuk rata-rata waktu yang diperlukan oleh *Raise Event* lebih sedikit jika dibanding dengan *Remote Procedure Call*. Sementara itu jumlah

pemanggilan *Remote Procedure Call* lebih banyak dari *Raise Event*. Berikut gambar 4.9 mengenai bentuk grafik dari hasil pengujian diatas.

4.8 Pengujian Fungsi Driver Delivery

Seperti halnya seperti pengujian-pengujian sebelumnya, pengujian ini dilakukan untuk mengetahui waktu tercepat, terlama, lalu rata-rata waktu yang diperlukan, selain juga menghitung jumlah pemanggilan fungsi dan jumlah data yang dikirim. Berikut hasil pengujian yang ditampilkan oleh tabel 4.6.

Tabel 4.6 *Driver Delivery Analysis*

<i>Method</i>	<i>Size (Byte)</i>	<i>Calls</i>	<i>Fastest (ms)</i>	<i>Slowest (ms)</i>	<i>Average (ms)</i>
<i>Raise Event</i>	71	103	16.4844	97.7469	63.75139706
<i>Remote Procedure Call</i>	71	106	0.7992	17.8869	9.976608571

Pada tabel diatas, *Raise Event* memiliki catatan waktu tercepat sekaligus waktu terlama yang lebih banyak daripada *Remote Procedural Call*, namun *Raise Event* memiliki jumlah pemanggilan yang lebih sedikit dari *Remote Procedure Call*. Berikut gambar 4.10 mengenai hasil grafik dari pengujian diatas.

<i>Match Id</i>	<i>Target Position</i>	<i>Spawn Position on Client</i>				<i>Conclusion</i>
		<i>Player 1</i>	<i>Player 2</i>	<i>Player 3</i>	<i>Player 4</i>	
d72e4acd-cda5-4e5a-bdf0-3c5ce58fa816	(-150.0, 83.6, 39.0)	(-150.0, 83.6, 39.0)	(-150.0, 83.6, 39.0)	(-150.0, 83.6, 39.0)	(-150.0, 83.6, 39.0)	<i>Correct</i>

4.11 Perbandingan Hasil Pengujian

Pada bagian ini akan menjelaskan tentang hasil perbandingan dari kecepatan rata-rata pada metode sinkronisasi transfer data menggunakan *multithreading* dengan tidak menggunakan *multithreading* atau *sequential*.

Pengujian pada sistem *sequential* memiliki ketentuan yang sama sebagaimana dengan sistem *multithreading*, yang membedakan adalah sistem masih dijalankan secara *sequential* atau berurutan.

Pada pengujian *sequential*, didapat sebagaimana yang ditampilkan pada tabel 4.12, dimana diketahui *Raise Event* secara rata-rata lebih cepat dengan nilai **2.38 ms** atau **4.1931%** lebih cepat dari *Remote Procedure Call*.

Tabel 4.12 *Sequential Average Speed*

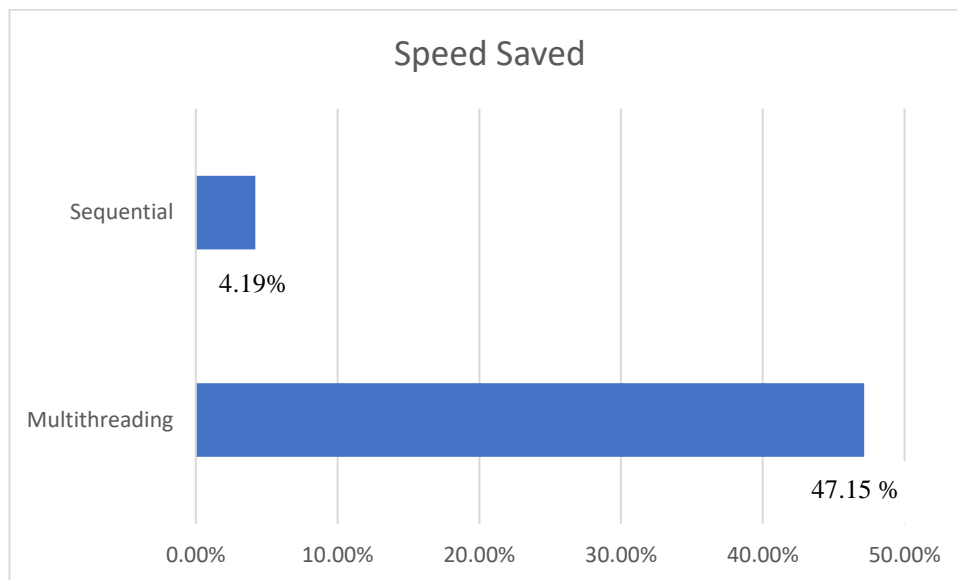
<i>Method</i>	<i>Average (ms)</i>
<i>Raise Event</i>	56.71
<i>Remote Procedure Call</i>	59.08498495
<i>Difference</i>	2.38
<i>Percentage</i>	4.1931%

Sementara itu pada pengujian menggunakan sistem *multithreading*, didapat seperti pada tabel 4.13, dimana *Raise Event* secara rata-rata lebih cepat dengan nilai **24.93732378** atau **47.1538140849377%** lebih cepat daripada *Remote Procedure Call*.

Tabel 4.13 *Multithreading Average Speed*

<i>Method</i>	<i>Average (ms)</i>
<i>Raise Event</i>	52.88506193
<i>Remote Procedure Call</i>	77.82238571
<i>Difference</i>	24.93732378
<i>Percentage</i>	47.1538140849377%

Dimana bisa digambarkan pada gambar 4.11 seberapa banyak nilai yang persentase dari kedua pengujian diatas.

Gambar 4.11 *Average speed saved*

Sehingga perbedaan kecepatan pada kedua sistem yaitu sistem *multithreading* unggul **42.96%** lebih cepat jika dibandingkan dengan sistem *sequential*.

4.12 Integrasi Sains dan Islam

Proses integrasi ini memungkinkan penulis untuk mengeksplorasi kebenaran yang terdapat dalam penemuan ilmiah, sambil tetap mengakui dan menghormati nilai-nilai dan prinsip-prinsip agama. Berikut integrasi sains dan islam pada penelitian ini :

4.12.1 Q.S. Al-Anbiya' ayat 80 dalam Konteks Teknologi

وَعَلَّمْنَاهُ صَنْعَةَ لَبُوسٍ لَكُمْ لِتُحْصِنَكُمْ مِنْ بَأْسِكُمْ فَهَلْ أَنْتُمْ شَاكِرُونَ

“Dan telah Kami ajarkan kepada Daud membuat baju besi untuk kamu, guna memelihara kamu dalam peperanganmu; Maka hendaklah kamu bersyukur (kepada Allah).” (Q.S. Al-Anbiya' ayat 80)

Berikut adalah tafsir Al-Jalalain untuk Q.S. Al-Anbiya' ayat 80:

Jika membaca *Linhshinakum*, maka Dhamir itu kembali kepada Allah, yaitu agar Kami dapat melindungimu. (Dan Kami ajarkan kepada Daud membuat baju besi), yaitu pakaian yang terbuat dari besi; dia adalah orang pertama yang membuatnya; dahulu hanya berupa lempengan-lempengan besi (untuk kamu); khusus untuk sekelompok orang (untuk melindungi diri sendiri). Dan jika *Lituhshinahum* dibaca, Dhamir kembali ke baju zirah, artinya baju zirah itu akan melindungi kalian. Menurut *Liyuhshinakum*, Dhamir kembali ke Nabi Daud untuk mencari perlindungan dari musuh kalian. Wahai penduduk Makkah, mohon (bersyukur) atas nikmat-Ku dengan beriman kepada Rasulullah. Maksudnya bersyukurlah kalian atas hal tersebut kepada-Ku (Tafsirq, 2023)

Apabila dalam konteks teknologi informasi, pengajaran nabi Daud pada pembuatan zirah besi menunjukkan bahwa Allah *subhāhahu wata'āla* mendorong inovasi teknologi terhadap tantangan yang sedang dihadapi, iterasi dari inovasi akan terus dilakukan untuk memperoleh solusi yang terbaik, seperti pada pengembangan sistem penelitian ini, yang bertujuan untuk memperoleh hasil sinkronisasi data yang tepat dan cepat sehingga memberikan kontribusi pada kemajuan teknologi untuk umat manusia.

4.12.2 Q.S. Al-'Alaq ayat 1-5 dalam Konteks Menuntut Ilmu

اقْرَأْ بِاسْمِ رَبِّكَ الَّذِي خَلَقَ (1) خَلَقَ الْإِنْسَانَ مِنْ عَلَقٍ (2) اقْرَأْ وَرَبُّكَ الْأَكْرَمُ (3) الَّذِي عَلَّمَ بِالْقَلَمِ (4) عَلَّمَ
الْإِنْسَانَ مَا لَمْ يَعْلَمْ (5)

“Bacalah dengan (menyebut) nama Tuhanmu yang menciptakan!(1) Dia menciptakan manusia dari segumpal darah.(2) Bacalah! Tuhanmulah Yang Mahamulia,(3) yang mengajar (manusia) dengan pena.(4) Dia mengajarkan manusia apa yang tidak diketahuinya.(5)” (QS Al-‘Alaq ayat 1-5)

Menurut tafsir Ibnu Katsir tentang surat Al-‘Alaq 1-5 :

Bacalah dengan (menyebut) nama Tuhanmu Yang Menciptakan. Dia telah menciptakan manusia dari segumpal darah. Bacalah, dan Tuhanmulah Yang Maha Pemurah, Yang mengajar (manusia) dengan perantaraan qalam. Dia mengajarkan kepada manusia apa yang tidak diketahuinya. Imam Ahmad mengatakan, telah menceritakan kepada kami Abdur Razzaq, telah menceritakan kepada kami Ma'mar, dari Az-Zuhri, dari Urwah, dari Aisyah yang menceritakan bahwa permulaan wahyu yang disampaikan kepada Rasulullah ﷺ berupa mimpi yang benar dalam tidurnya. Dan beliau tidak sekali-kali melihat suatu mimpi, melainkan datangnya mimpi itu bagaikan sinar pagi hari. Kemudian dijadikan baginya suka menyendiri, dan beliau sering datang ke Gua Hira, lalu melakukan ibadah di dalamnya selama beberapa malam yang berbilang dan untuk itu beliau membawa perbekalan secukupnya. Kemudian beliau pulang ke rumah Khadijah (istrinya) dan mengambil bekal lagi untuk melakukan hal yang sama. Pada suatu hari ia dikejutkan dengan datangnya wahyu saat berada di Gua Hira. Malaikat pembawa

wahyu masuk ke dalam gua menemuinya, lalu berkata, "Bacalah!" Rasulullah ﷺ melanjutkan kisahnya, bahwa ia menjawabnya, "Aku bukanlah orang yang pandai membaca." Maka malaikat itu memeganku dan mendekapku sehingga aku benar-benar kepayahan olehnya, setelah itu ia melepaskan diriku dan berkata lagi, "Bacalah!" Nabi ﷺ menjawab, "Aku bukanlah orang yang pandai membaca." Malaikat itu kembali mendekapku untuk kedua kalinya hingga benar-benar aku kepayahan, lalu melepaskan aku dan berkata, "Bacalah!" Aku menjawab, "Aku bukanlah orang yang pandai membaca." Malaikat itu kembali mendekapku untuk ketiga kalinya hingga aku benar-benar kepayahan, lalu dia melepaskan aku dan berkata: Bacalah dengan (menyebut) nama Tuhanmu Yang Menciptakan. (Al-'Alaq: 1) sampai dengan firman-Nya: apa yang tidak diketahuinya. (Al-'Alaq: 5) Maka setelah itu Nabi ﷺ pulang dengan hati yang gemetar hingga masuk menemui Khadijah, lalu bersabda: Selimutilah aku, selimutilah aku! Maka mereka menyelimutinya hingga rasa takutnya lenyap. Lalu setelah rasa takutnya lenyap, Khadijah bertanya, "Mengapa engkau?" Maka Nabi ﷺ menceritakan kepadanya kejadian yang baru dialaminya dan bersabda, "Sesungguhnya aku merasa takut terhadap (keselamatan) diriku." Khadijah berkata, "Tidak demikian, bergembiralah engkau, maka demi Allah, Dia tidak akan mengecewakanmu selama-lamanya. Sesungguhnya engkau adalah orang yang suka bersilaturahmi, benar dalam berbicara, suka menolong orang yang kesusahan, gemar menghormati tamu, dan membantu orang-orang yang tertimpa musibah." Kemudian Khadijah membawanya kepada Waraqah ibnu Naufal ibnu Asad ibnu Abdul Uzza ibnu Qusay.

Waraqah adalah saudara sepupu Khadijah dari pihak ayahnya, dan dia adalah seorang yang telah masuk agama Nasrani di masa Jahiliah dan pandai menulis Arab, lalu ia menerjemahkan kitab Injil ke dalam bahasa Arab seperti apa yang telah ditakdirkan oleh Allah, dan dia adalah seorang yang telah lanjut usia dan tuna netra. Khadijah bertanya, "Wahai anak pamanku, dengarlah apa yang dikatakan oleh anak saudaramu ini." Waraqah bertanya, "Wahai anak saudaraku, apakah yang telah engkau lihat?" Maka Nabi ﷺ menceritakan kepadanya apa yang telah dialami dan dilihatnya.

Setelah itu Waraqah berkata, "Dialah Namus (Malaikat Jibril) yang pernah turun kepada Musa. Aduhai, sekiranya diriku masih muda. Dan aduhai, sekiranya diriku masih hidup di saat kaummu mengusirmu." Rasulullah ﷺ memotong pembicaraan, "Apakah benar mereka akan mengusirku?" Waraqah menjawab, "Ya, tidak sekali-kali ada seseorang lelaki yang mendatangkan hal seperti apa yang engkau sampaikan, melainkan ia pasti dimusuhi. Dan jika aku dapat menjumpai harimu itu, maka aku akan menolongmu dengan pertolongan yang sekuat-kuatnya." Tidak lama kemudian Waraqah wafat, dan wahyu pun terhenti untuk sementara waktu hingga Rasulullah ﷺ merasa sangat sedih.

Menurut berita yang sampai kepada kami, karena kesedihannya yang sangat, maka berulang kali ia mencoba untuk menjatuhkan dirinya dari puncak bukit yang tinggi. Akan tetapi, setiap kali beliau sampai di puncak bukit untuk menjatuhkan dirinya dari atasnya, maka Jibril menampakkan dirinya dan berkata kepadanya, "Wahai Muhammad, sesungguhnya engkau adalah utusan Allah yang sebenarnya," maka tenanglah hati beliau karena berita itu, lalu kembali pulang ke rumah keluarganya.

Dan manakala wahyu datang terlambat lagi, maka beliau berangkat untuk melakukan hal yang sama. Tetapi bila telah sampai di puncak bukit, kembali Malaikat Jibril menampakkan diri kepadanya dan mengatakan kepadanya hal yang sama. Hadits ini diketengahkan di dalam kitab Sahihain melalui Az-Zuhri; dan kami telah membicarakan tentang hadits ini ditinjau dari segi sanad, matan, dan maknanya pada permulaan kitab syarah kami, yaitu Syarah Bukhari dengan pembahasan yang lengkap.

Maka bagi yang ingin mendapatkan keterangan lebih lanjut, dipersilakan untuk merujuk kepada kitab itu, semuanya tertulis di sana. Mula-mula wahyu Al-Qur'an yang diturunkan adalah ayat-ayat ini yang mulia lagi diberkati, ayat-ayat ini merupakan permulaan rahmat yang diturunkan oleh Allah karena kasih sayang kepada hamba-hamba-Nya, dan merupakan nikmat yang mula-mula diberikan oleh Allah kepada mereka. Di dalam surat ini terkandung peringatan yang menggugah manusia kepada asal mula penciptaan manusia, yaitu dari 'alaqah.

Dan bahwa di antara kemurahan Allah *subhāhahu wata'āla* ialah Dia telah mengajarkan kepada manusia apa yang tidak diketahuinya. Hal ini berarti Allah telah memuliakan dan menghormati manusia dengan ilmu. Dan ilmu merupakan bobot tersendiri yang membedakan antara Abul Basyar (Adam) dengan malaikat. Ilmu itu adakalanya berada di hati, adakalanya berada di lisan, adakalanya pula berada di dalam tulisan tangan. Berarti ilmu itu mencakup tiga aspek, yaitu di hati, di lisan, dan di tulisan. Sedangkan yang di tulisan membuktikan adanya penguasaan pada kedua aspek lainnya, tetapi tidak sebaliknya.

Karena itulah disebutkan dalam firman-Nya: Bacalah, dan Tuhanmulah Yang Maha Pemurah, Yang mengajar (manusia) dengan perantaraan qalam. Dia mengajarkan kepada manusia apa yang tidak diketahuinya. (Al-'Alaq: 3-5) Di dalam sebuah atsar disebutkan, "Ikatlah ilmu dengan tulisan." Dan masih disebutkan pula dalam atsar, bahwa barang siapa yang mengamalkan ilmu yang dikuasainya, maka Allah akan memberikan kepadanya ilmu yang belum diketahuinya (Learn Quran, 2022).

BAB V

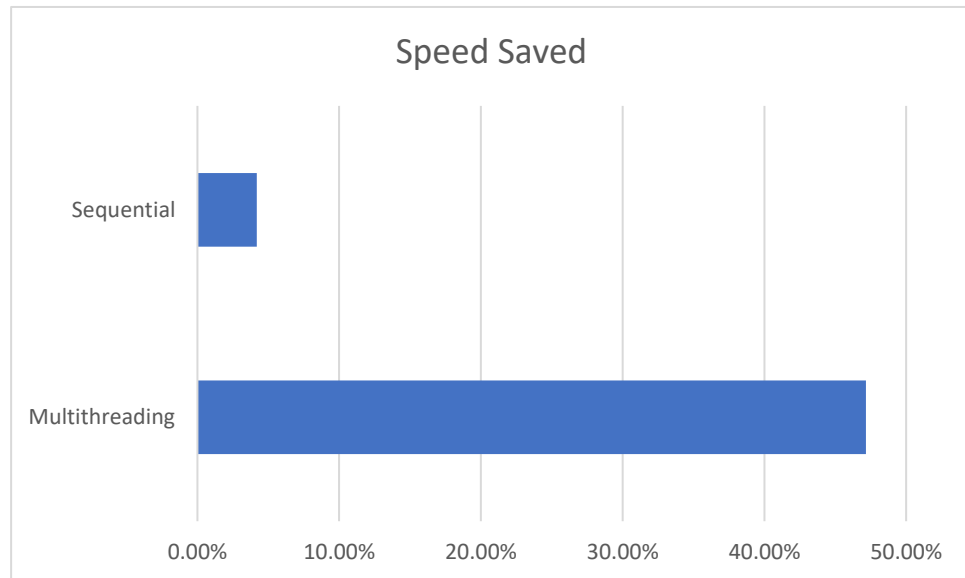
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Penelitian ini merupakan implementasi penggunaan metode *Raise Event* untuk sinkronisasi data komponen pada *gameobject*, penelitian dilakukan pada *game multiplayer* dan dijalankan pada 4 perangkat keras untuk pengiriman dan sinkronisasi data. Sebagai pembandingan, selain menggunakan *Raise Event*, juga digunakan metode *Remote Procedure Call* dalam pengujian. Berdasarkan hasil pengujian, dapat diambil kesimpulan bahwa penggunaan metode *Raise Event* bisa lebih cepat dari pada penggunaan dari metode *Remote Procedure Call* dengan **47.1538140849377%** lebih cepat dibandingkan dengan *Remote Procedure Call* seperti yang terlihat pada tabel 5.1, selain itu hasil dari pengujian juga menyatakan bahwa data mampu dihasilkan secara sinkron, seperti yang terlihat pada pengujian posisi karakter dan *map*, dimana keduanya mampu muncul pada posisi sesuai sasaran. Selain itu pada perbandingan antara sistem *sequential* dengan *multithreading*, dapat disimpulkan bahwa *multithreading* lebih cepat dalam menjalankan fungsi sinkronisasi dengan perbedaan **42.96%** lebih cepat daripada sistem *sequential*, seperti yang digambarkan pada gambar 5.1

Tabel 5.1 Rata-rata kecepatan *RPC* dengan *Raise Event*

<i>Method</i>	<i>Average (ms)</i>
<i>Raise Event</i>	52.88506193
<i>Remote Procedure Call</i>	77.82238571
<i>Difference</i>	24.93732378
<i>Percentage</i>	47.1538140849377%



Gambar 5.1 *Speed saved by each system*

5.2 Saran

Dari hasil penelitian tentang Sinkronisasi Data komponen *Gameobject* pada *Game Multiplayer* berbasis *Raise Event*, ada beberapa hal yang masih bisa ditingkatkan untuk bisa memperoleh hasil yang lebih baik, diantaranya :

1. Percobaan dilakukan pada jumlah *player* yang lebih banyak.
2. Percobaan dilakukan pada sistem *game* yang lebih kompleks.
3. Percobaan bisa dilakukan dengan lokasi antar *player* yang lebih variatif.

DAFTAR PUSTAKA

- Aaltonen, P. (2022). NETWORKING TOOLS PERFORMANCE EVALUATION IN A VR APPLICATION. *Turku University of Applied Sciences*.
- Albahari, J. A. & B. (2012). C# 7.0 In a Nutshell. In *Naturschutz und Landschaftsplanung* (Vol. 44, Issue 10).
- Alexander, T. (2004). *Massively Multiplayer Game Development*.
- Apperley, T. (2006). Genre and game studies: Toward a critical approach to video game genres. *Simulation & Gaming - Simulat Gaming*, 37, 6–23. <https://doi.org/10.1177/1046878105282278>
- Arif, Y. M., Firdaus, M. N., & Nurhayati, H. (2021). A Scoring System for Multiplayer Game Base on Blockchain Technology. *Proceedings - 2021 IEEE Asia Pacific Conference on Wireless and Mobile, APWiMob 2021*, 200–205. <https://doi.org/10.1109/APWiMob51111.2021.9435249>
- Armitage, G., Claypool, M., & Branch, P. (2006). *Networking and Online Games: Understanding and Engineering Multiplayer Internet Games*. Wiley.
- Arsenault, D. (2009). Video Game Genre, Evolution and Innovation. *Eludamos. Journal for Computer Game Culture*, 3, 149–176. <https://doi.org/10.7557/23.6003>
- Azmi, S., Bakhtiar, F., & Hossain, A. (2016). *Multiplayer Online War Simulator Based On Unity3D*.
- Bawa, P., Watson, W., Sunnie, P., & Watson, L. (2017). To Game or Not to Game? How Using Massively Multiplayer Online Games Helped Motivation and Performance in a College Writing Course: A Mixed Methods Study. *Journal of Research Initiatives*, 3(1), 12.
- Browne, P., & Schram, B. R. (2020). Intermediating the Everyday: Indie Game Development and the Labour of Co-Working Spaces. In *Game Production Studies* (pp. 83–100).
- Bryne, G. (2022). *Target C#: Simple Hands-On Programming with Visual Studio 2022*. Apress.
- Cleary, S. (2019). *Concurrency in C# Cookbook*.

- De Paoli, S. (2013). Automatic-Play and Player Deskillling in MMORPGs. *The International Journal of Computer Game Research*, 13(1). https://gamestudies.org/1301/articles/depaoli_automatic_play
- Dualism, C., May, D., Game, C., Games, G. R., & Themes, R. (2015). “Venturing into the Unknown”(?) Method(olog)ical Reflections on Religion and Digital Games, Gamers and Gaming. *Online - Heidelberg Journal of Religions on the Internet*, 7.
- Exit Games. (2022). *RPCs and RaiseEvent (Online)*. <https://doc.photonengine.com/en-us/pun/current/gameplay/rpcsandraiseevent>
- Faisal, M., Imammudin, M., & Perdana, R. (2022). Increasing Users Response of Tourism Game. *IEESE International Journal of Science and Technology (IJSTE)*, 11(1), 6–16.
- Fathurrahman, & Miftachul Arif, Y. (2021). Game Promosi Wisata Kota Malang “Kakang Mbakyu” Dengan Menggunakan Decission Tree dan Hierarchy Finite State. *Systemic: Information System and Informatics Journal*, 6(1), 51–57. <https://doi.org/10.29080/systemic.v6i1.958>
- Fox, R., & Hao, W. (2017). Internet infrastructure: Networking, Web services, and cloud computing. In *Internet Infrastructure: Networking, Web Services, and Cloud Computing*. <https://doi.org/10.1201/9781315175577>
- Freeman, G. (2018). *Multiplayer Online Games - Origins, Players, & Social Dynamics*.
- Freeman, G., & Schulenberg, K. (2023). Understanding and Mitigating Challenges for Non-Profit Driven Indie Game Development to Innovate Game Production. *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. <https://doi.org/10.1145/3544548.3580976>
- Friston, S., Fan, C., Dobös, J., Scully, T., & Steed, A. (2017). 3DRepo4Unity: Dynamic Loading of Version Controlled 3D Assets into the Unity Game Engine. *Proceedings - Web3D 2017: 22nd International Conference on 3D Web Technology*. <https://doi.org/10.1145/3055624.3075941>
- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. M. (1998). *Design Patterns Elements of Reusable Object-Oriented Software*.
- Gazis, A., & Katsiri, E. (2023). Serious Games in Digital Gaming: A Comprehensive Review of Applications, Game Engines and Advancements. *WSEAS Transactions on Computer Research*, 11, 10–22. <https://doi.org/10.37394/232018.2023.11.2>

- Glazer, J., & Madhav, S. (2016). *Multiplayer Game Programming*. Addison-Wesley.
- Gregory, J. (2019). *Game Engine Architecture* (3rd Edition). CRC Press.
- Hartzheim, B. H. (2019). At Your Service: Event-Based Design in Japanese Mobile Games. *Proceedings of DiGRA 2019*. https://www.easychair.org/publications/preprint_download/D4ZI
- José, G., Urriola, M., Jose, M., Bonilla, V., Iván, C., Trejos, P., Iván, C., & Trejos, P. (2023). *NONINTRUSIVE VIRTUAL TRAINER PROTOTYPE FOR EXERCISE ROUTINES*. 10, 13–20.
- Joyce Farrell. (2017). *Microsoft® Visual C# 2017 : an introduction to object-oriented programming*.
- Kang, K., Lu, J., Guo, L., & Zhao, J. (2020). How to improve customer engagement: A comparison of playing games on personal computers and on mobile phones. *Journal of Theoretical and Applied Electronic Commerce Research*, 15(2), 76–92. <https://doi.org/10.4067/S0718-18762020000200106>
- Kelly, S., & Kumar, K. (2022). Unity Networking Fundamentals. In *Unity Networking Fundamentals*. <https://doi.org/10.1007/978-1-4842-7358-6>
- Kenwright, B. (2016). Holistic game development curriculum. *SA 2016 - SIGGRAPH ASIA 2016 Symposium on Education*. <https://doi.org/10.1145/2993352.2993354>
- Learn Quran. (2022). *Tafsir Al-'Alaq ayat 1-5 menurut Ibnu Katsir*. <https://tafsir.learn-quran.co/id/surat-96-al-alaq/ayat-1>
- Liu, K.-Y. (2015). The MORPG-based Learning System for Multiple Courses: A Case Study on Computer Science Curriculum. *International Journal of Distance Education Technologies (IJDET)*, 13(1). <https://www.igi-global.com/gateway/article/123210>
- Martin, J. (2012). Game On: The Challenges and Benefits of Video Games. *Bulletin of Science, Technology and Society*, 32(5), 343–344. <https://doi.org/10.1177/0270467612469066>
- Meakin, E., Vaughan, B., & Cullen, C. (2021). “Understanding” Narrative; Applying Poetics to Hellblade: Senua’s Sacrifice. *The International Journal of Computer Game Research*, 21(2). https://gamestudies.org/2102/articles/meakin_vaughan_cullen
- Mikami, K., Watanabe, T., Yamaji, K., Ozawa, K., Ito, A., Kawashima, M., Takeuchi, R., Kondo, K., & Kaneko, M. (2010). Construction trial of a

- practical education curriculum for game development by industryuniversity collaboration in Japan. *Computers and Graphics (Pergamon)*, 34(6), 791–799. <https://doi.org/10.1016/j.cag.2010.09.015>
- Musch, O. (2023). Design Patterns with Java. In *Design Patterns with Java*. <https://doi.org/10.1007/978-3-658-39829-3>
- Nugroho, F., Basid, P. M. N. S. A., Bahtiar, F. S., Simamora, R. N. Z., Kurniawan, R. F., Janitra, G. A., & Fadilah, J. N. (2022). 2D Game “Omar’s Adventure” design using the Finite State Machine Method. *JOURNAL OF INFORMATICS AND TELECOMMUNICATION ENGINEERING*, 6(1), 18–26. <https://doi.org/10.31289/jite.v6i1.6327>
- Nuora, J. (2018). *Introduction to sound design for virtual reality games : a look into 3D sound, spatializer plugins and their implementation in Unity game engine*.
- Nurhayati, H., Faisal, M., & Romadhoni, M. (2017). Q-PUZZLE GAME FOR MEMORIZING THE RECITATION OF QUR’AN SURAH. *Journal of Theoretical and Applied Information Technology*, 31, 20. www.jatit.org
- Nystrom, & Robert. (2014). *Game Programming Patterns*.
- Polancec, D., & Mekterovic, I. (2017). Developing MOBA games using the Unity game engine. *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2017 - Proceedings*, 1510–1515. <https://doi.org/10.23919/MIPRO.2017.7973661>
- Reuter, C. (2016). *Authoring collaborative multiplayer games: game design patterns, structural verification, collaborative balancing and rapid prototyping*.
- Sciore, E. (2019). Java Program Design Principles, Polymorphism, and Patterns. In *Java Program Design*. Apress. <https://doi.org/10.1007/978-1-4842-4143-1>
- Singh, S., & Kaur, A. (2022). Game Development using Unity Game Engine. *2022 3rd International Conference on Computing, Analytics and Networks (ICAN)*, 1–6.
- Skeet, J. (2019). C# in Depth - Fourth Edition. In *Magnetic Resonance Imaging* (Vol. 44, Issue August).
- Sothmann, A. (2018). *Business Models in the Video Game Industry*. 1–31.
- Stagner, A. R. (2013). *Unity Multiplayer Games*. In *Packt Publishing*.

- Syahrul, M., Mohd, H., & Yusof, C. S. (2018). Collaborative Handheld Augmented Reality for Interactive Furniture Interior Design. *UTM Computing Proceedings Innovations in Computing Technology and Applications*, 3, 1–7.
- Tafsirq. (2023, June 21). *Q.S. Al-Anbiya' ayat 80*. <https://tafsirq.com/21-al-anbiya/ayat-80>
- Zeiler, X., & Mukherjee, S. (2022). Video Game Development in India: A Cultural and Creative Industry Embracing Regional Cultural Heritage(s). *Games and Culture*, 17(4), 509–527. <https://doi.org/10.1177/15554120211045143>