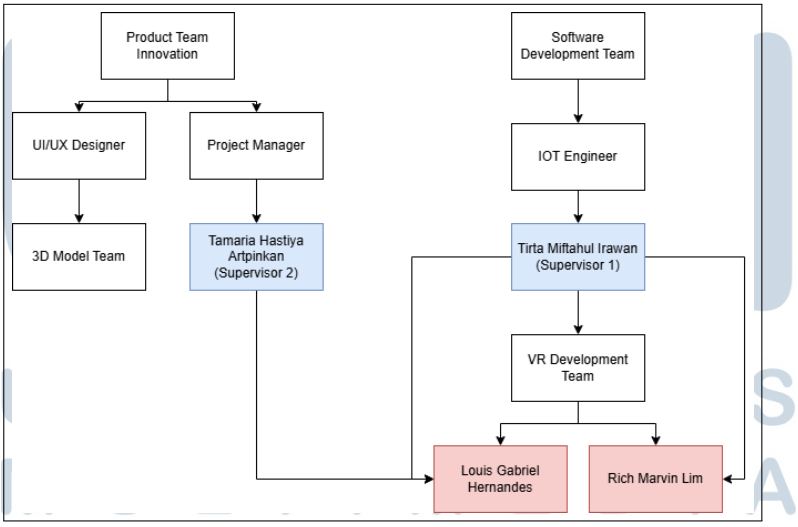
BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Dalam pelaksanaan kegiatan magang di PT Saka Farma Laboratories (*Kalbe Consumer Health*), posisi *VR Developer Intern* dibimbing langsung oleh tim *IOT Engineer*. Penugasan ini merupakan bagian dari inisiatif strategis perusahaan untuk mengembangkan media promosi dan pelatihan berbasis *Virtual Reality* yang ditujukan untuk sarana yang efektif untuk memperkenalkan PT Saka Farma Laboratories kepada audiens.

Proyek ini dilaksanakan melalui kolaborasi lintas divisi, termasuk tim *Software Development*, *Project Management*, serta tim desain visual. Kegiatan pengembangan dilakukan bersama seorang rekan magang dari *Universitas Multimedia Nusantara*, *Rich Marvin Lim*, yang juga mengemban peran sebagai *VR Developer Intern*. Dalam struktur organisasi, proyek berada di bawah pengawasan *Supervisor 1*, *Tirta Miftahul Irawan* dari tim *IOT Engineer*, dan dikoordinasikan bersama *Supervisor 2*, *Tamaria Hastiya Artpinkan* dari divisi *Project Management*. Struktur organisasi tempat pelaksanaan magang ditampilkan pada Gambar 3.1.



Gambar 3.1. Struktur Organisasi Perusahaan PT Saka Farma Laboratories (Kalbe Consumer Health)

Sumber: [11]

Tim *VR Development* bertanggung jawab untuk merancang dan membangun aplikasi *Virtual Reality* menggunakan *game engine*, mencakup scripting, pemrograman interaktif, optimasi performa, dan integrasi aset 3D. Untuk mendukung visualisasi simulasi, tim ini bekerja sama erat dengan tim *3D Model* di bawah koordinasi *UI/UX Designer*, yang menyediakan model tiga dimensi komponen mesin berdasarkan referensi nyata. Kolaborasi antara tim *VR Development* dan tim *3D Model* dilakukan secara sinkron dan iteratif, guna memastikan setiap elemen visual yang dikembangkan sesuai dari segi bentuk, skala, dan fungsi dalam lingkungan virtual. Pendekatan lintas departemen ini mencerminkan penerapan kerja kolaboratif dan integratif dalam mendukung inovasi teknologi pelatihan di perusahaan.

3.2 Tugas yang Dilakukan

Tugas utama selama magang di PT Saka Farma Laboratories (Kalbe Consumer Health) sebagai *VR Developer Intern* berfokus pada pengembangan simulasi pelatihan teknisi berbasis *Virtual Reality* sebagai bagian dari inisiatif peningkatan efektivitas pelatihan internal. Simulasi dirancang secara imersif dan interaktif menggunakan *Unity* untuk merepresentasikan operasional mesin produksi. Dalam pengembangannya, digunakan berbagai perangkat pendukung seperti *Unity* sebagai *development platform*, *Visual Studio Code* untuk penulisan kode, serta *Git* dan *GitHub* sebagai sistem *version control* dan kolaborasi proyek. Manajemen kode dilakukan menggunakan sistem *branching* sesuai standar tim teknis Kalbe.

Simulasi ini mengadaptasi proses perakitan mesin secara menyeluruh, mulai dari interaksi dengan komponen virtual menggunakan *XR Grab Interactable*, hingga penyusunan logika perakitan (*assembly logic*) dan validasi prosedural berbasis skenario. Selain itu, dikembangkan animasi tangan kontekstual, serta menu panduan *step-by-step* untuk memandu teknisi selama pelatihan. Komponen-komponen penting dari *XR Toolkit* seperti *XR Ray Interactor*, *XR Origin Rig*, dan *Unity Input System* digunakan untuk menciptakan pengalaman pengguna yang realistis dan ergonomis.

Pada tahap awal, dilakukan pengenalan struktur organisasi dan proyek terkait, disertai pembelajaran mandiri mengenai fitur teknis seperti *teleportation*, interaksi berbasis *physics*, serta integrasi animasi. Tugas pertama mencakup pembuatan prototipe simulasi dasar menggunakan *raycasting* dan *grab interaction*. Pengembangan berlanjut ke fitur lanjutan seperti sistem interaksi *step-based*,

optimisasi performa grafis untuk perangkat *VR standalone*, dan pengujian modularitas antar komponen. Proses ini didampingi oleh *code review* dan diskusi rutin bersama pembimbing teknis. Pengembangan dilakukan secara daring dan luring dengan koordinasi melalui *Microsoft Teams* dan *WhatsApp*, memungkinkan kolaborasi langsung dengan tim pengembang serta kontribusi aktif dalam realisasi solusi pelatihan berbasis teknologi imersif. Rincian kegiatan mingguan selama proses ini dapat dilihat pada Tabel 3.1.

Tabel 3.1. Rincian kegiatan mingguan selama periode magang

Minggu	Kegiatan
1	Pengenalan lingkungan kerja, struktur organisasi, dan setup awal *workstation*; instalasi *Unity Hub*, *Unity Editor*, serta penggunaan *Notion* untuk manajemen tugas.
2	Eksplorasi *Unity Editor* dan komponennya; penataan struktur proyek; konfigurasi awal *Input System* dan *XR Interaction Toolkit*.
3	Setup *Proof of Concept* dengan *XR Rig*, *teleportation*, serta *debugging* interaksi menggunakan *collider* dan *layer*; pengenalan *prefab*.
4	Pencarian dan konversi aset mesin ke format kompatibel dengan *Unity*; proses *import*, penyesuaian *scale*, *pivot*, dan *collider*.
5	Observasi langsung di lapangan (*Genba*); dokumentasi proses teknisi; penyusunan *flowchart* alur simulasi berdasarkan prosedur nyata.
6	Implementasi interaksi dasar: *grab*, *drop*, dan *attach*; penyusunan skrip interaktif dengan *C#*; penerapan *Rigidbody*, *Collider*, dan *Tag*.
7	*Kick-off* internal proyek VR Saka; penataan *folder* (`Assets`, `Prefabs`, `Scripts`); persiapan demo awal; integrasi *version control* dengan Git dan GitHub.
8	Pembuatan sistem interaksi modular untuk *disassembly*; penulisan skrip *step-by-step*; uji sistem urutan menggunakan *State Machine*.
9	Integrasi aset dari pemodelan eksternal; pengelolaan *prefab variants*; pengaturan *layer* agar alat hanya bekerja pada objek tertentu.

Tabel 3.1. Rincian kegiatan mingguan selama periode magang (lanjutan)

Minggu	Kegiatan
10	Pengembangan *teleportation* berbasis *pointer*; penambahan *reticle* dan umpan balik visual; optimisasi *physics* dan aktivasi *occlusion culling*.
11	Penyusunan panduan teknis penggunaan simulasi; dokumentasi struktur dan *interaction pipeline*; pembuatan *README* dan *markdown files* di *repository*.
12	Penerapan *Level of Detail (LOD)* untuk optimalisasi; pengaturan skala kamera dan posisi *XR Camera* untuk kenyamanan pengguna.
13	Pengembangan *reusable framework* untuk *disassembly* dan *assembly*; generalisasi langkah prosedural; validasi *prefab*.
14	Implementasi interaksi berbasis *physics* lanjutan: *constraints*, *hinge joints*, dan *spring*; penambahan *visual cue* untuk umpan balik.
15	Finalisasi proyek: pembersihan *scene*, optimisasi performa, dan *bug fixing*; evaluasi bersama tim; persiapan *showcase* simulasi.

3.3 Perangkat Penunjang Pelaksanaan Magang

Pengembangan simulasi *Virtual Reality* pada kegiatan magang di Kalbe Consumer Health didukung oleh perangkat lunak, perangkat keras, serta kerangka kerja dan pustaka pendukung. Komponen-komponen ini digunakan untuk membangun lingkungan virtual yang imersif dan interaktif sesuai kebutuhan pelatihan teknisi. Rincian perangkat dan teknologi tercantum pada tabel 3.2, 3.3, dan 3.4.

Tabel 3.2. Daftar perangkat lunak, spesifikasi (versi), dan fungsinya

Nama Perangkat	Spesifikasi (Versi)	Fungsi
Visual Studio Code (VS Code)	1.99.3	*Code editor* untuk pengembangan dan *debugging* skrip
Unity	2022.3.43f1	*Development platform* utama untuk simulasi *VR*
GitHub Desktop	3.4.18 (x64)	Antarmuka *GUI* untuk *version control* menggunakan *Git*
C#	8.0	Bahasa pemrograman utama untuk logika dan interaksi
Blender	3.6 LTS	Perangkat pembuat dan pengelola *3D model*

Tabel 3.3. Daftar perangkat keras dan spesifikasinya

Nama Perangkat	Spesifikasi
Laptop	ASUS TUF Gaming FX505DD
Processor	AMD Ryzen 5 3550H, 2.1 GHz (4MB L3 Cache, hingga 3.7 GHz, 4 core, 8 thread)
RAM	16 GB DDR4
Storage	512 GB *PCIe 4.0 NVMe M.2 SSD* dan 1 TB Toshiba *SATA HDD*
Operating System	*Windows 11 Home* 64-bit
Display	15.6” *FHD* 120Hz (1920 × 1080) *IPS Panel* (non-touch)
GPU	NVIDIA® *GeForce GTX 1050*

Tabel 3.4. Daftar *framework*, *library*, dan ekstensi VS Code untuk pengembangan simulasi *VR*

Nama *Framework* / *Library* / Ekstensi	Spesifikasi (Versi)
Unity Engine	2022.3 LTS (*Long Term Support*)
XR Interaction Toolkit	2.4.3 (*official Unity package for VR interaction*)
Input System	1.5.1 (*Unitys new input management for VR controllers*)
Oculus Integration	57.0 (*SDK* dari Meta untuk *Meta Quest*)
ProBuilder	5.2.2 (untuk pembuatan *level design* dan *geometry prototyping*)
Cinemachine	2.10.0 (pengaturan kamera dinamis)
Shader Graph	14.0.8 (visualisasi efek dan material untuk *URP*)
Universal Render Pipeline (URP)	14.0.8 (*lightweight render pipeline* untuk performa optimal *VR*)
TextMeshPro	3.2.0 (penampilan teks dalam *world-space* dan *UI*)
VS Code Extension – C# for Visual Studio Code	v1.26.0 (ekstensi resmi *OmniSharp*)
VS Code Extension – Unity Tools	v1.2.2 (*auto-complete* dan *snippets* untuk *Unity*)
VS Code Extension – Debugger for Unity	v3.0.0 (*debugging* langsung dari *VS Code*)
Newtonsoft.Json (JSON.NET)	13.0.1 (untuk *data parsing* dan serialisasi)
DOTween (Demigiant)	1.2.755 (animasi ringan seperti umpan balik *grab* atau efek transisi)

3.4 User Requirements

User requirements (kebutuhan pengguna) adalah kumpulan spesifikasi yang menggambarkan apa saja yang diinginkan dan dibutuhkan oleh pengguna akhir dari sistem atau aplikasi yang dikembangkan. *User requirements* mencakup elemen-elemen yang harus dihadirkan untuk memastikan pengalaman penggunaan yang realistis dan sesuai dengan kebutuhan operasional. Setiap kebutuhan diharapkan dapat diimplementasikan secara rinci agar sistem mampu mencerminkan aktivitas manusia dan kondisi nyata dalam lingkungan kerja. Berikut adalah daftar kebutuhan pengguna yang diperoleh:

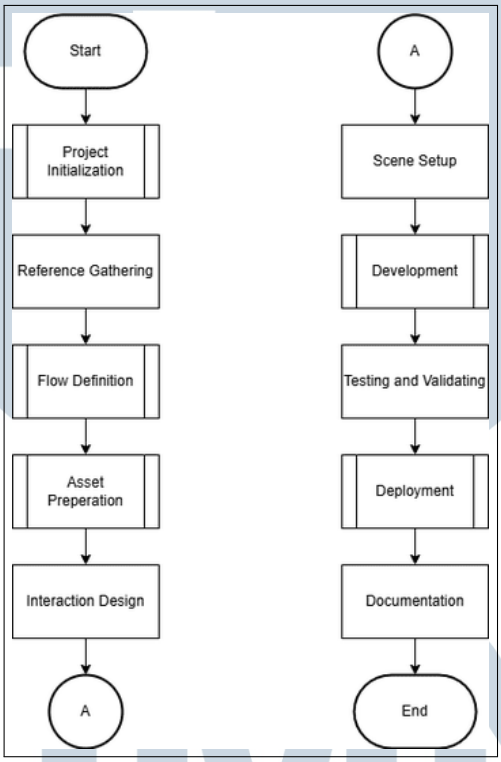
1. Ruangan di *virtual reality* dibuat serupa dengan tampilan di pabrik SAKA.
2. Animasi rotasi diterapkan pada elemen seperti baut untuk menambah realisme gerakan.
3. Efek audio untuk benda yang jatuh perlu dilengkapi.
4. Tersedia tangga lipat portabel yang dibawa operator saat naik atau turun.
5. Fitur menu *scroll wheel* untuk ganti peralatan dan juga fitur *inventory*.
6. Menyusun mekanisme agar operator tetap dapat berinteraksi atau mengambil benda meski *Scene* telah selesai.
7. Fitur *scoreboard* yang menampilkan waktu penyelesaian, dan kesalahan yang dibuat.

3.5 Proses Pelaksanaan Magang

Bagian ini menjelaskan tahapan pelaksanaan proyek pengembangan simulasi *Virtual Reality* (VR) yang dilakukan selama kegiatan magang di *Kalbe Consumer Health*. Tujuan utama proyek ini adalah menciptakan media pelatihan berbasis VR bagi teknisi internal agar memahami proses perakitan dan pembongkaran mesin produksi.

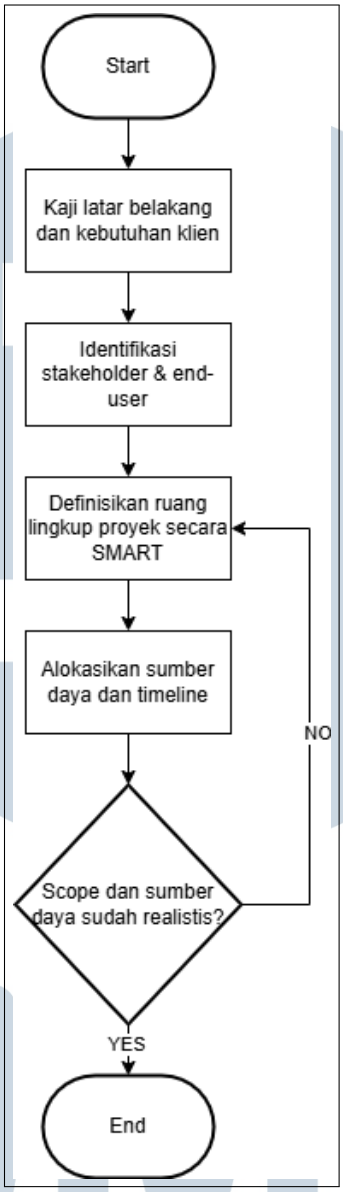
3.5.1 Alur Perancangan

Tahapan perancangan dimulai dengan proses perencanaan umum alur kerja pengembangan proyek VR. Pada tahap awal ini, alur keseluruhan proses dari inisiasi hingga dokumentasi ditetapkan secara garis besar, agar seluruh tim memahami tahapan dan dependensi antar proses. Diagram alur proses kerja secara umum ditunjukkan pada Gambar 3.2.



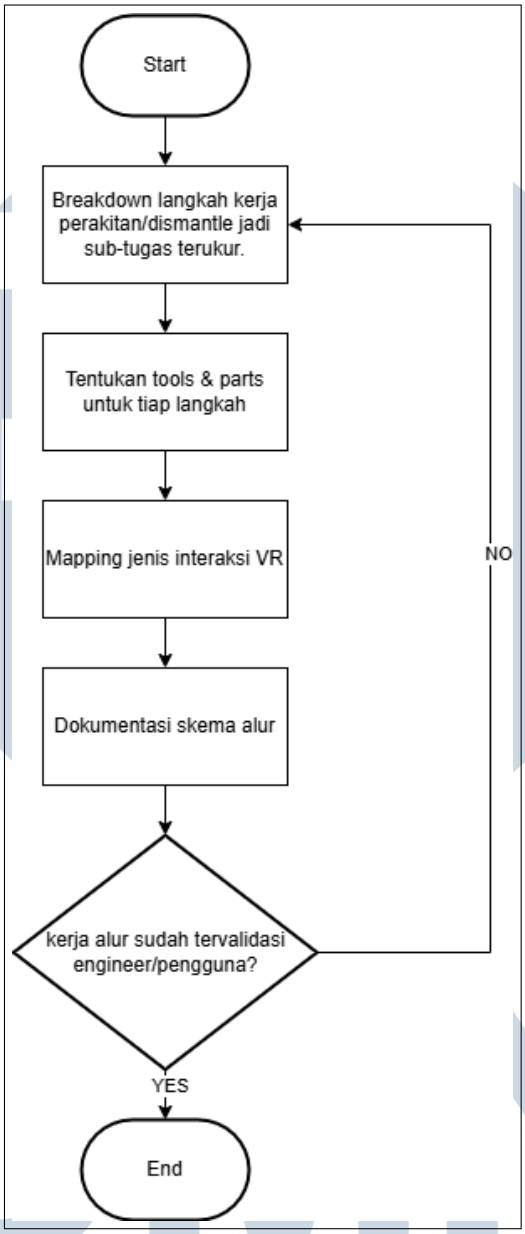
Gambar 3.2. Diagram alur proses kerja pengembangan proyek VR.

Setelah perencanaan umum dilakukan, tahapan perancangan dilanjutkan dengan proses inisiasi proyek, di mana dilakukan pengumpulan kebutuhan dari klien dan identifikasi pemangku kepentingan terkait. Ruang lingkup proyek dirumuskan berdasarkan prinsip *SMART* (Specific, Measurable, Achievable, Relevant, Time-Bound) untuk memastikan setiap target pengembangan dapat dicapai dengan realistis dan terukur. Evaluasi sumber daya dan kapabilitas tim juga dilakukan guna memastikan kesiapan dalam mengerjakan proyek secara berkelanjutan. Proses ini divisualisasikan dalam Gambar 3.3.



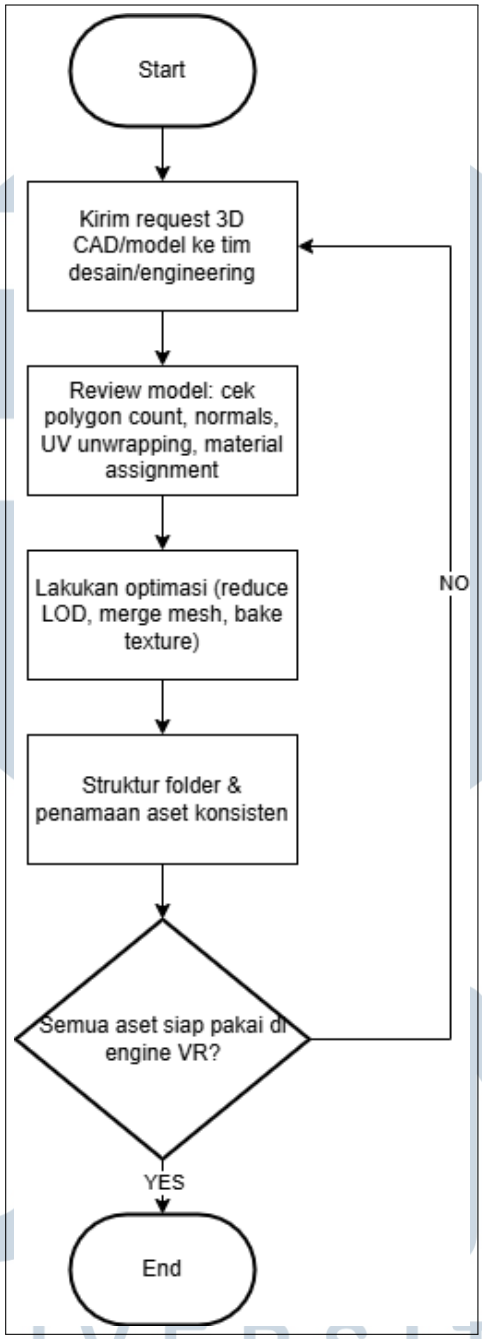
Gambar 3.3. Diagram alur proses *Project Initialization*.

Setelah ruang lingkup ditetapkan, dilakukan pendefinisian alur kerja proses perakitan dan pembongkaran mesin. Setiap langkah dipetakan dan diklasifikasikan ke dalam bentuk tugas-tugas kecil agar dapat diimplementasikan dengan efisien dalam lingkungan *VR*. Interaksi yang sesuai juga dirancang, seperti *grabbing*, *socketing*, hingga pemicu visual/audio. Proses ini menghasilkan dokumen teknis yang nantinya divalidasi bersama tim teknis dan pengguna akhir. Visualisasi alurnya ditampilkan pada Gambar 3.4.



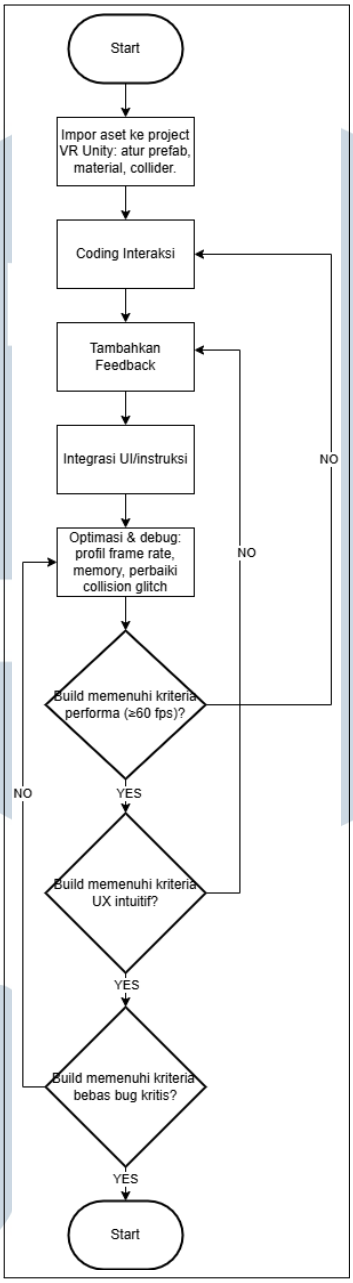
Gambar 3.4. Diagram alur proses *Flow Definition*.

Selanjutnya, dilakukan tahap pengolahan aset visual berupa model 3D mesin. Aset ini diperoleh dari tim desain dan kemudian dievaluasi untuk memastikan kesesuaian dengan spesifikasi teknis VR, seperti jumlah poligon, skala, serta pemetaan material. Untuk mengoptimalkan performa, diterapkan teknik pengurangan *Level of Detail* (LOD), penggabungan *mesh*, serta *baking texture*. Alur pengerjaan ini ditunjukkan pada Gambar 3.5.



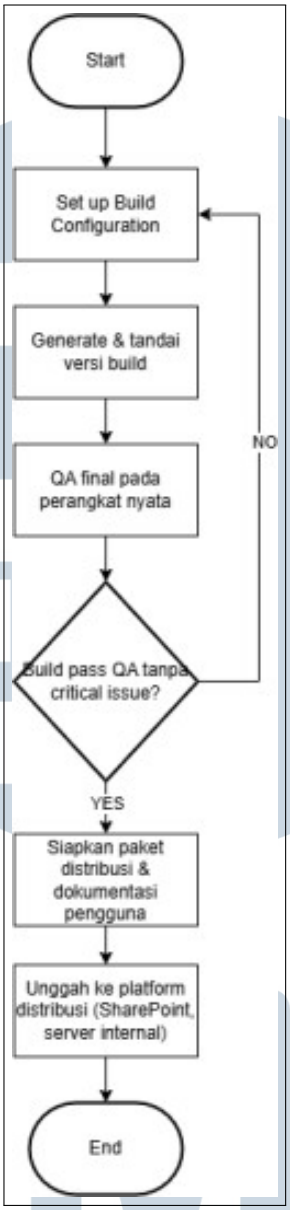
Gambar 3.5. Diagram alur proses *Asset Preparation*.

Tahapan selanjutnya adalah pengembangan teknis menggunakan *Unity Engine*. Aset-aset yang telah dioptimasi diimpor ke dalam proyek, lalu dilakukan konfigurasi komponen seperti *collider*, *prefab*, serta penambahan skrip interaksi. Selain itu, ditambahkan juga elemen umpan balik seperti suara dan getaran, serta sistem instruksi untuk mendukung navigasi pengguna. Proses ini digambarkan pada Gambar 3.6.



Gambar 3.6. Diagram alur proses *Development*.

Tahap akhir dari perancangan adalah proses *deployment*, yaitu membangun file aplikasi eksekusi untuk perangkat target dan melakukan uji coba akhir di perangkat VR secara langsung. Setelah hasil pengujian menunjukkan performa yang stabil dan fitur berfungsi sebagaimana mestinya, build akhir dikemas bersama dokumentasi pengguna dan disebarluaskan melalui sistem distribusi internal. Gambar 3.7 menunjukkan visualisasi proses ini.



Gambar 3.7. Diagram alur proses *Deployment*.

3.5.2 Persiapan dan Konfigurasi Proyek Virtual Reality

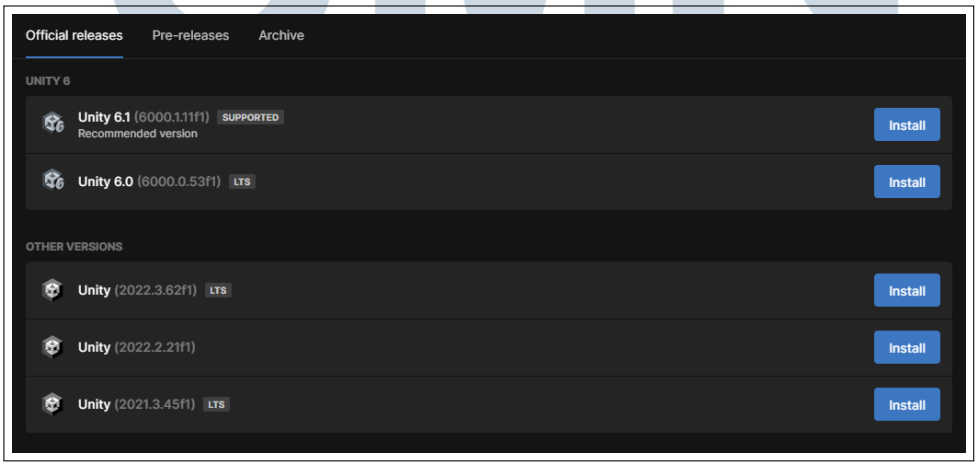
Tahapan awal dalam proses pengembangan aplikasi VR dimulai dengan menyiapkan proyek Unity yang telah dikonfigurasi agar kompatibel dengan perangkat *VR standalone* seperti Meta Quest 3. Tujuan utama dari fase ini adalah memastikan seluruh komponen penting untuk pengembangan berbasis *Virtual Reality* telah tersedia dan berfungsi optimal dalam lingkungan Unity.

A. Setup VR Project (Editor, SDK (Software Development Kit), Library)

Langkah awal meliputi pembuatan proyek Unity baru dengan pengaturan yang mendukung Meta Quest 3. Untuk memastikan kelancaran dalam proses pengembangan serta kompilasi ke perangkat VR, konfigurasi awal dilakukan melalui *Unity Hub* dan *Unity Editor*, dengan memperhatikan komponen berikut:

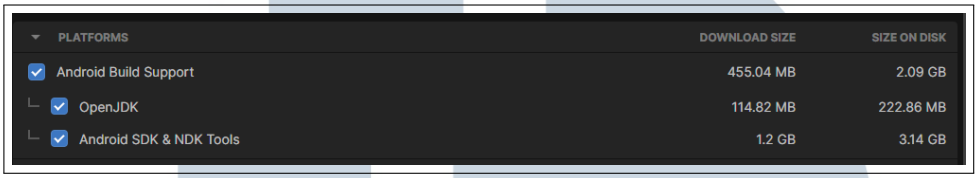
1. Penggunaan *Unity Editor* versi terbaru (versi 2022.3.62f1 LTS).
2. Aktivasi Android Build Support beserta dependensi *Android SDK NDK Toolsm* dan*OpenJDK (Java Development Kit)*
3. Instalasi *XR (Extended Reality) Plugin Management* (untuk mendukung perangkat XR seperti Meta Quest).
4. Penggunaan *XR Interaction Toolkit* versi 3.0.8, melalui *Unity Package Manager.*
5. Integrasi *Unity Input System* versi 1.4.0 atau lebih tinggi, agar kompatibel dengan kontroler dan *input* VR.
6. Penggunaan *Mock HMD (Head-Mounted Display) XR Plugin* untuk melakukan simulasi dan pengujian di editor tanpa perangkat HMD fisik.

Karena aplikasi VR akan dijalankan langsung di Meta Quest yang berbasis *Android*, konfigurasi *Android Build Support* menjadi sangat penting. Unity Editor yang digunakan adalah versi 2022.3.62f1 LTS. Versi ini dapat diinstal melalui tab Official Releases seperti tampak pada Gambar 3.8.



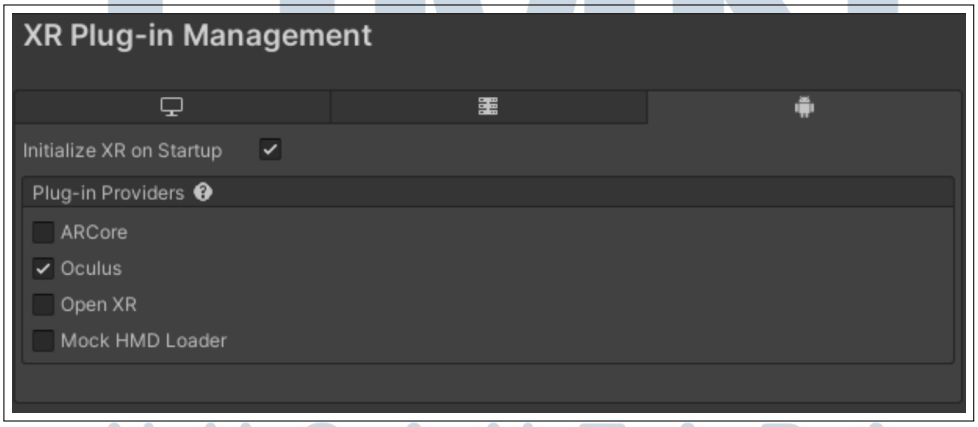
Gambar 3.8. Instalasi *Unity Editor*.

Ketika melakukan instalasi melalui *Unity Hub*, pastikan komponen seperti SDK, *NDK (Native Development Kit)*, dan *OpenJDK* dicentang agar proses kompilasi dan deployment ke perangkat dapat berjalan tanpa kendala. Tampilan pemilihan komponen tersebut ditunjukkan pada Gambar 3.9.



Gambar 3.9. Pengaturan modul *Android Build Support* pada *Unity Hub* untuk kompilasi ke perangkat Meta Quest.

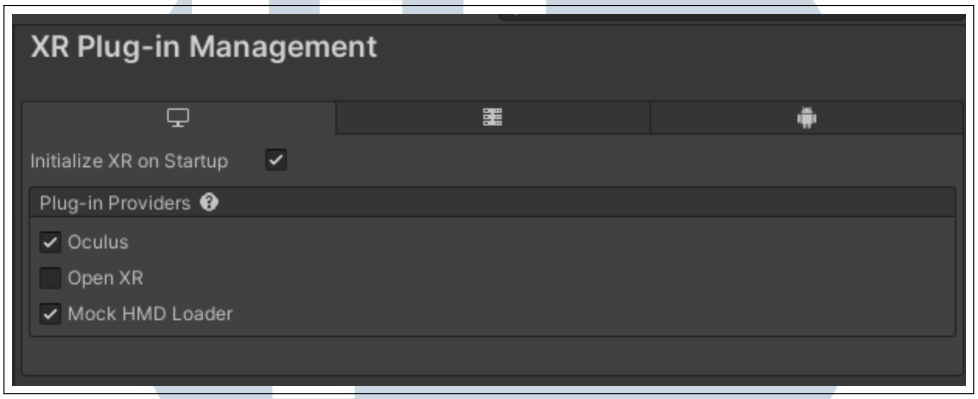
Setelah pemasangan komponen utama selesai, langkah penting selanjutnya adalah pengaturan Manajemen *XR Plug-in*. Fitur ini bisa ditemukan di menu `Edit > Project Settings > XR Plug-in Management`, dan berperan sebagai pengatur koneksi antara Unity dan perangkat vR seperti Meta Quest 3. *Plugin* ini menawarkan antarmuka untuk memilih penyedia XR yang akan digunakan dalam proyek, baik untuk kebutuhan pengujian dalam editor maupun *runtime*. Pada *platform Android*, yang menjadi target utama Meta Quest, menggunakan opsi Oculus sebagai penyedia XR utama. Ini memungkinkan Unity untuk mendeteksi Meta Quest sebagai perangkat Oculus dan mengaktifkan semua layanan yang diperlukan untuk *rendering* stereoskopik serta pelacakan kontroler. Pemilihan komponen dapat dilihat pada Gambar 3.10



Gambar 3.10. Pengaturan XR Plug-in Management untuk platform Android menggunakan Oculus sebagai penyedia utama.

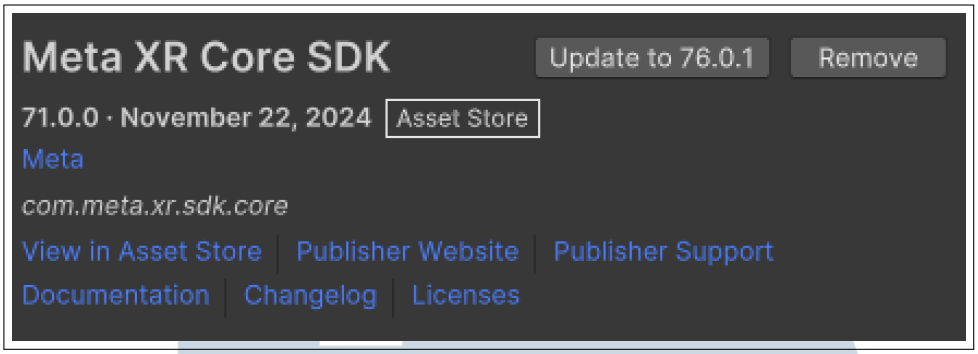
Aktifkan *plugin Mock HMD Loader* untuk *platform PC (standalone)*, terutama saat pengujian dilakukan langsung di editor tanpa menggunakan perangkat *headset*.

Plugin ini mensimulasikan perangkat HMD dan kontroler virtual, memungkinkan pengembang untuk melakukan pengujian dan debugging tanpa harus menjalankan build langsung ke *headset*. Dalam kondisi seperti ini, Oculus masih digunakan sebagai alternatif saat HMD sesungguhnya tersedia. Pengaturan dapat dilihat pada Gambar 3.11



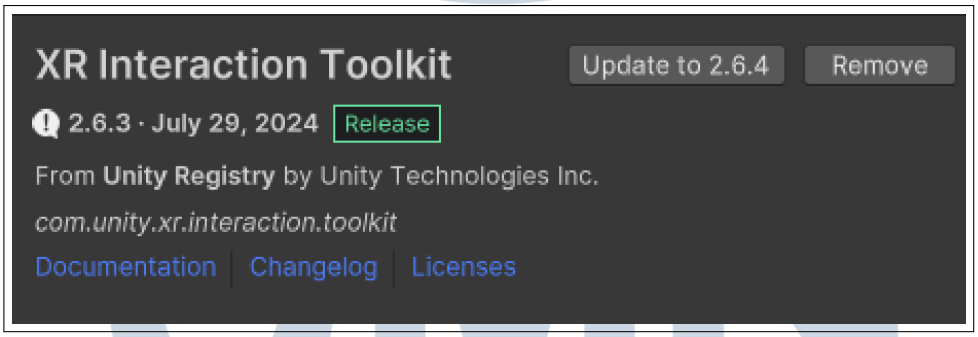
Gambar 3.11. Pengaturan *XR Plug-in Management* untuk *platform PC* dengan *Mock HMD* dan *Oculus* diaktifkan.

Untuk pengujian awal, *Mock HMD* dapat digunakan, dan dengan mengubah target *platform* menjadi *Android*, pengujian langsung dapat dengan cepat dilakukan di perangkat Meta Quest. Setelah konfigurasi Manajemen *Plug-in XR* selesai, langkah selanjutnya adalah menginstal dan mengintegrasikan SDK dan pustaka interaksi yang diperlukan untuk mendukung pengembangan antarmuka pengguna berbasis XR. Ini dilakukan melalui *Asset Store* dan *Unity Package Manager*. Komponen utama yang harus diimpor adalah *Meta XR Core SDK*, yang dikembangkan secara langsung oleh pihak Meta (Facebook). SDK ini menyediakan berbagai modul penting untuk perangkat Oculus dan Meta Quest, seperti sistem pengawasan, input tangan, *stereo rendering*, dan integrasi dengan layanan runtime Oculus. Versi 71.0.0 digunakan untuk proyek ini, seperti yang ditunjukkan pada Gambar 3.12.



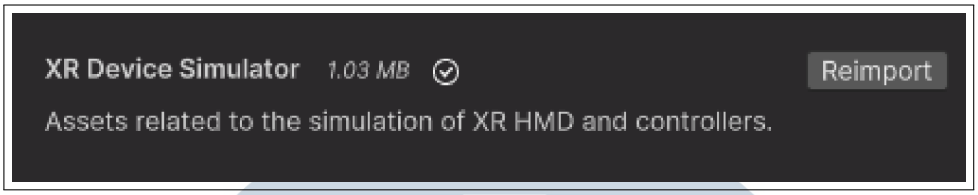
Gambar 3.12. *Meta XR Core SDK* dari *Unity Asset Store* digunakan untuk mendukung interaksi dan tracking di perangkat Meta Quest.

Langkah berikutnya adalah menginstal *XR Interaction Toolkit*, yang merupakan pustaka standar dari Unity yang menangani interaksi di lingkungan XR, baik dalam *virtual reality* maupun *augmented reality*. Dengan bantuan *XR Interaction Toolkit*, pengembang dapat dengan mudah menerapkan sistem seperti *grab*, *hover, socket*, teleporasi, dan banyak lagi tanpa harus membangun semua sistem dari awal. Versi 2.6.3, yang dapat diunduh langsung dari Registry Unity, unduhan bisa dilihat pada Gambar 3.13



Gambar 3.13. *XR Interaction Toolkit* digunakan sebagai basis sistem interaksi dalam simulasi VR.

Untuk mendukung proses pengembangan Unity menyediakan *XR Device Simulator*, yang merupakan komponen dari *XR Interaction Toolkit*. Fitur ini memungkinkan pengguna mengendalikan simulasi XR hanya dengan *keyboard* dan *mouse*, mendukung proses pengembangan tanpa memerlukan headset atau kontroler fisik. Simulator ini dapat digunakan untuk navigasi, mengambil interaksi, dan validasi logika interaksi di dalam *Unity Editor*. Ini sangat membantu saat pengujian awal sebelum*deploy* ke perangkat Meta Quest. *Import XR device Simulator* dapat dilihat pada Gambar 3.14



Gambar 3.14. *XR Device Simulator* digunakan untuk mendukung pengembangan dan pengujian tanpa HMD dan kontroler fisik.

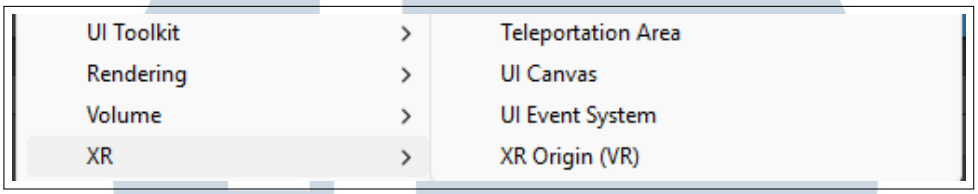
Komponen dasar sistem interaksi yang dikembangkan dalam proyek ini terdiri dari *Meta XR Core SDK, XR Interaction Toolkit*, dan *XR Device Simulator*. Selama proses pengembangan dan pengujian aplikasi VR, penggabungan pustaka resmi Meta dan *toolkit native Unity* memastikan stabilitas, kompatibilitas, dan efisiensi. Selain itu, untuk menjaga kompatibilitas dengan sistem dan perangkat terbaru, Unity dapat memberikan fitur *update* langsung melalui *Unity Package Manager*. Dengan kombinasi pengaturan ini, tim pengembang dapat berkonsentrasi pada logika bisnis dan interaksi pengguna tanpa khawatir tentang masalah teknis *low-level* yang terkait dengan integrasi perangkat XR.

B. Setup Player and Controls

Pada Unity, *Player* dan *Controls* direpresentasikan oleh *XR Rig*, *prefab* yang menyediakan struktur dasar untuk kamera dan kontroler berbasis XR. Pengaturan *XR Rig* sangat penting karena komponen ini memungkinkan pengguna berinteraksi secara langsung dengan lingkungan virtual, baik melalui pergerakan kepala (*head tracking*) maupun tangan (*controller input*). Berikut adalah beberapa komponen utama *XR Rig*:

1. Kamera utama: Kamera ini berfungsi sebagai titik pandang pengguna (*HMD/headset*) dalam ruang 3D dan bergerak secara *real-time* mengikuti posisi dan rotasi kepala pengguna.
2. Kontroler Kiri dan Kanan mewakili tangan kiri dan kanan pengguna. Kontroler memiliki kemampuan untuk mengendalikan objek, melakukan *grab*, teleportasi, dan interaksi khusus lainnya dengan menggunakan posisi, rotasi, dan tombolnya.
3. *XRRig Object*: objek induk ini menampung semua elemen XR, seperti kontroler dan kamera. Pengguna akan ditempatkan di mana dalam *scene* saat aplikasi dimulai berdasarkan posisi awal *XR Origin*.

Prefab XR Origin (berbasis aksi) dari *XR Interaction Toolkit* dimasukkan ke dalam scene untuk memulai pengaturan awal. Setelah itu, setiap komponen, seperti kontroler dan kamera, diatur agar merespons input sistem *Unity Input System* yang telah diaktifkan sebelumnya. Penambahan XR Rig dapat dilakukan dengan cara `klik kanan area Hierarchy > XR > XR Origin (VR)` dapat dilihat pada Gambar 3.15

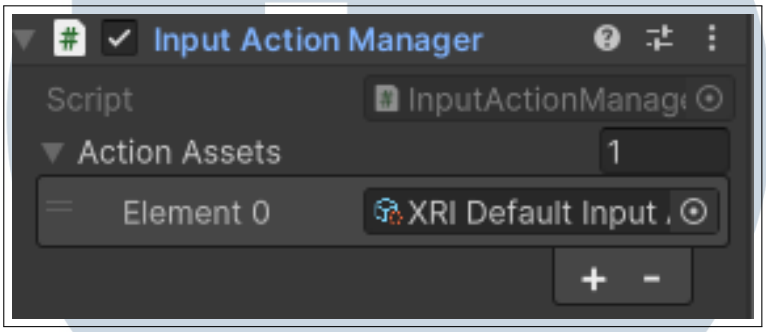


Gambar 3.15. Menambahkan *XR Rig* ke dalam scene.

Parameter *Tracking Origin Mode* komponen *XR Origin* harus disesuaikan untuk memastikan kamera mengikuti tinggi kepala pengguna dengan akurat. Nilai *Device* dan mengatur *offset camera* agar tinggi sesuai yang diinginkan di dunia virtual. Akibatnya, posisi kamera secara otomatis akan disesuaikan berdasarkan posisi vertikal kepala pengguna di luar perangkat. Karena memungkinkan sistem menyesuaikan perspektif secara alami, pengaturan ini sangat penting untuk mendukung pengalaman VR imersif. Sebagai contoh, kamera akan mengamati perubahan tinggi ketika pengguna berdiri, duduk, atau membungkuk, dan memperbarui sudut pandang secara *real-time* melalui pelacakan HMD (*Head-Mounted Display*). Seperti yang ditunjukkan pada Gambar 3.16, pengaturan ini dapat dilakukan langsung melalui inspector Unity.

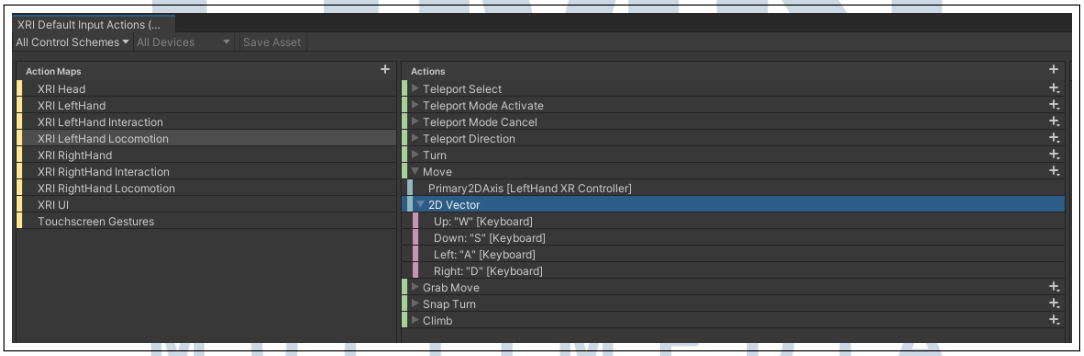
Gambar 3.16. Pengaturan komponen *XR Origin* ke mode *Device* dan mengatur *Offset* Kamera

Langkah selanjutnya adalah menambahkan komponen *Input Action Manager* ke dalam *XR Rig*. Ini berfungsi sebagai penghubung antara sistem input (*Unity Input System*) dengan aksi-aksi XR yang telah didefinisikan di dalam *XRI Default Input Actions*. Tanpa komponen ini, input dari perangkat seperti kontroler atau headset tidak akan dapat dikenali dengan benar oleh Unity. Untuk mengaktifkan sistem input secara penuh, tambahkan *Input Action Manager* ke dalam *Field Action Assets* dengan cara ini. Komponen *Input Action Manager* dapat dilihat pada Gambar 3.17



Gambar 3.17. Menambahkan komponen Input Action Manager dan mengatur *XRI Default Input Actions* sebagai aset aksi.

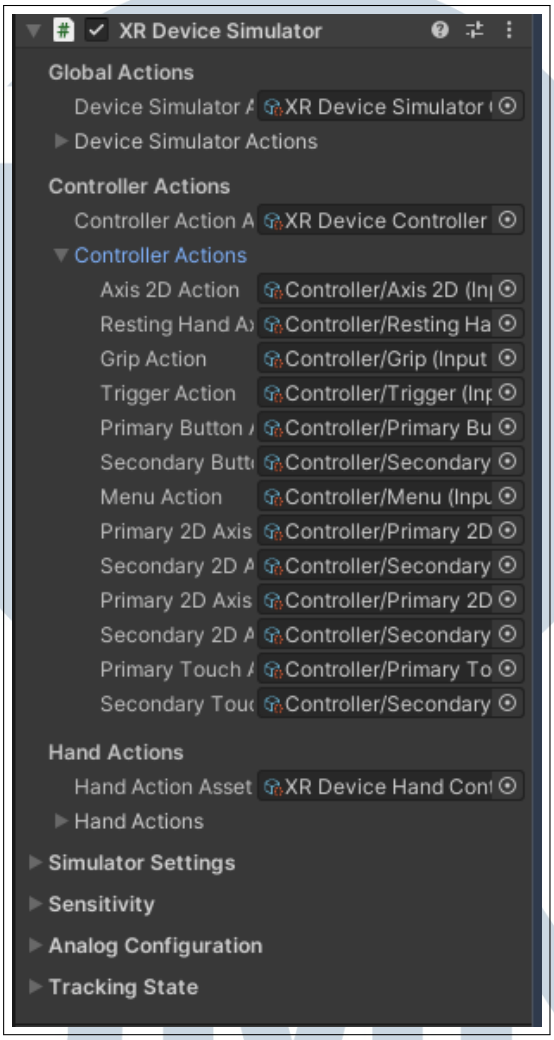
Selanjutnya, pada bagian *XRI Default Input Actions*, ditambah sebuah *Action* agar sistem dapat mendeteksi pergerakan pengguna di dalam dunia virtual ketika menggunakan *mouse* dan *keyboard* dengan menggunakan *2D Vector* `WASD`, khususnya untuk pengujian langsung pada perangkat *PC standalone*. Menu *XRI Default Input Actions* dapat dilihat pada Gambar 3.18



Gambar 3.18. Menambahkan *2D Vector* pada *XRI Lefthand Locomotion* untuk pergerakan *keyboard*

Meskipun kita sudah dapat bergerak menggunakan tombol keyboard WASD, diperlukan metode lain untuk bergerak dengan tombol-tombol seperti interaksi *select, grip*, dan lainnya. Untuk itu, kita menggunakan *XR Device Simulator*

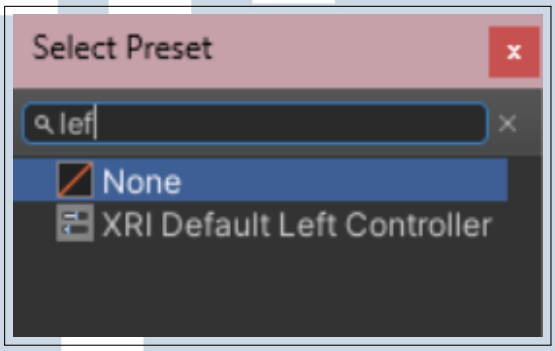
sebagai representasi karakter kita saat melakukan pengujian pada *PC standalone*. Penambahan komponen *XR Device Simulator* dapat dilihat pada Gambar 3.19



Gambar 3.19. Menambahkan komponen *XR Device Simulator* untuk Pengujian *PC Standalone*

Selain konfigurasi *XR Rig* dan *Tracking Origin Mode*, pengaturan input pada pengontrol juga merupakan elemen penting dalam simulasi VR. Untuk mempercepat proses pengaturan dan menghindari konfigurasi manual yang rumit, Unity menyediakan fitur *Preset* pada *XR Controller*, yang merupakan kumpulan konfigurasi standar untuk input yang dapat langsung diterapkan pada objek pengontrol. Preset ini dapat diakses melalui komponen *XR Controller* (*Action-Based*) yang bisa dilihat pada Gambar 3.20, baik untuk tangan kiri maupun tangan kanan. Unity menawarkan dua *preset* default utama, yaitu:

1. *XRI Default Left Controller* – Digunakan untuk mengatur input pada pengontrol kiri.
2. *XRI Default Right Controller* – Digunakan untuk mengatur input pada pengontrol kanan.



Gambar 3.20. Pemilihan *preset XRI Default Left Controller* pada kontroler kiri

Untuk menampilkan model tangan, cukup menambahkan *prefab* atau objek 3D tangan sebagai *child* dari *Left Controller* dan *Right Controller*. Unity menyediakan beberapa model tangan *default* yang dapat diimpor dari *XR Interaction Toolkit Samples* atau *Asset Store*. Selain itu, pengguna juga dapat menggunakan model tangan kustom yang diunduh dari internet atau dibuat sendiri untuk mencapai tampilan yang lebih realistis dan sesuai dengan kebutuhan. Hierarchy dapat dilihat pada Gambar 3.21

Gambar 3.21. *Hierarchy* dari *XR Rig Controller*

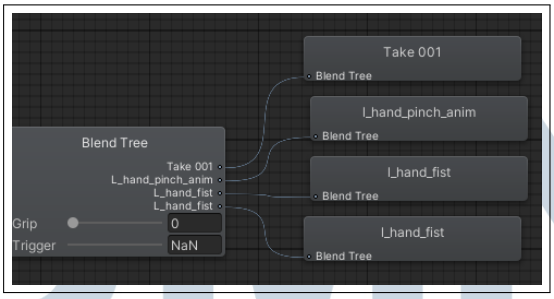
Untuk meningkatkan realisme interaksi dalam simulasi VR, disertakan juga animasi dinamis pada tangan virtual, seperti menggenggam atau menekan. Animasi ini memungkinkan model tangan untuk merespons input pengguna secara real-time, sehingga menciptakan pengalaman visual yang lebih natural saat berinteraksi dengan objek di dunia virtual. Implementasi dilakukan dengan menambahkan

skrip bernama `ScHandAnimator` yang dapat di lihat di Kode 3.1. Skrip ini membaca nilai dari *input trigger* (*pinch*) dan *grip* (genggaman) pada kontroler VR menggunakan *InputActionProperty*, lalu meneruskan nilai tersebut ke parameter *animator*. Komponen *Animator* pada skrip ini akan memudahkan proses penerapan dan pengaturan animasi yang sesuai.

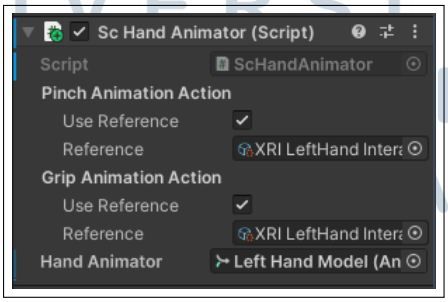
```
1 float triggerValue = pinchAnimationAction . action . ReadValue <
    float
2 >() ;
3 handAnimator . SetFloat ("Trigger", triggerValue );
4
5 float gripValue = gripAnimationAction . action . ReadValue <float
    >() ;
6 handAnimator . SetFloat ("Grip", gripValue );
```

Kode 3.1: Menonaktifkan tumbukan antar collider.

Agar dapat berfungsi dengan baik, referensi *Animator* pada model tangan harus dikaitkan yang bisa dilihat pada Gambar 3.22, dan parameter animator seperti *Trigger* dan *Grip* perlu sudah ada dalam animasi model tangan. Gambar 3.23 menunjukkan konfigurasi skrip *animator* yang telah dikaitkan dengan model tangan kiri.

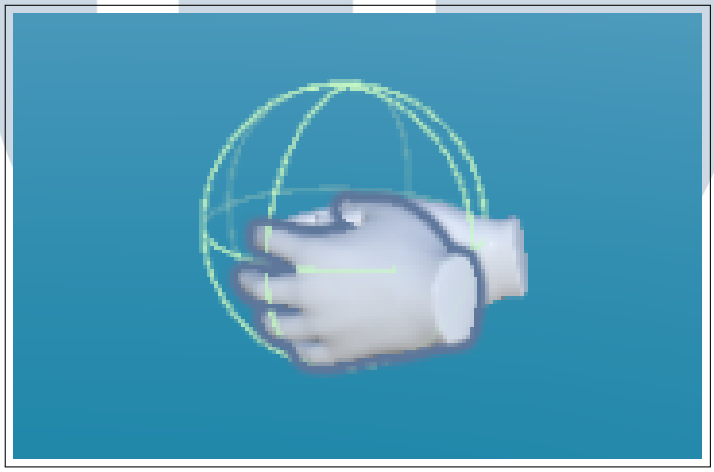


Gambar 3.22. Konfigurasi animasi di *Animator*



Gambar 3.23. Konfigurasi skrip animasi di Komponen *ScHandAnimator*

Untuk mendukung interaksi fisik yang akurat di dalam simulasi VR, pengaturan *collider* pada berbagai objek dan model sangat penting untuk dilakukan. Pada model tangan virtual, digunakan *Sphere Collider* yang ditempatkan di ujung jari-jari tangan. Penambahan *Sphere Collider* ini berfungsi untuk mendeteksi keberadaan objek di sekitarnya, sehingga sistem dapat merespons interaksi seperti *hover*, *grab*, maupun *release* dengan lebih responsif. Konfigurasi ini juga membantu meningkatkan presisi saat pengguna berinteraksi dengan komponen mesin yang kecil atau memiliki bentuk kompleks. Contoh implementasi penambahan *Sphere Collider* pada model tangan ditunjukkan pada Gambar 3.24.



Gambar 3.24. Contoh penambahan *Sphere Collider* pada model tangan untuk mendeteksi objek di sekitarnya.

Collider ini umumnya diatur sebagai Trigger untuk mencegah terjadinya benturan fisik, namun tetap memungkinkan untuk mendeteksi apakah objek berada dalam jangkauan. Ketika pengguna menekan tombol grab (biasanya grip atau trigger), sistem akan mencari objek yang ada di dalam area collider dan memiliki komponen interaktif seperti XRGrabInteractable.

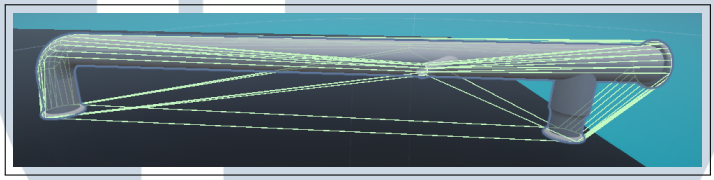
C. Setup Machine Parts

Setelah sistem interaksi dasar seperti *XR Rig* dan *input* selesai dikonfigurasi, langkah berikutnya adalah mengatur aset 3D dari bagian mesin yang akan digunakan dalam simulasi. Aset-aset ini dikembangkan oleh tim desainer eksternal dan disediakan dalam format 3D yang siap diimpor ke Unity. Proses impor dilakukan secara manual dengan cara drag-and-drop ke *folder Assets* dalam *Unity*

Project, kemudian ditempatkan dalam *Scene Hierarchy* sesuai dengan struktur perakitan yang diinginkan.

Setiap bagian mesin dikonfigurasi secara terpisah untuk memastikan interaksi yang realistis di dunia virtual. Beberapa komponen penting yang ditambahkan atau disesuaikan pada setiap objek meliputi:

1. *Collider*: Penggunaan *Box Collider*, *Capsule Collider*, atau *Mesh Collider* ditentukan berdasarkan bentuk fisik objek. Komponen ini memungkinkan deteksi tumbukan dan interaksi dengan tangan virtual pengguna. Contoh dapat dilihat pada Gambar 3.25.



Gambar 3.25. Contoh objek mesin dengan *mesh collider*

2. *Rigidbody*: Ditambahkan untuk memungkinkan interaksi fisik seperti jatuh akibat gravitasi, tergeser, atau tertarik saat di-*grab*. Parameter seperti massa, *drag*, dan penggunaan gravitasi disesuaikan untuk meniru perilaku objek nyata.

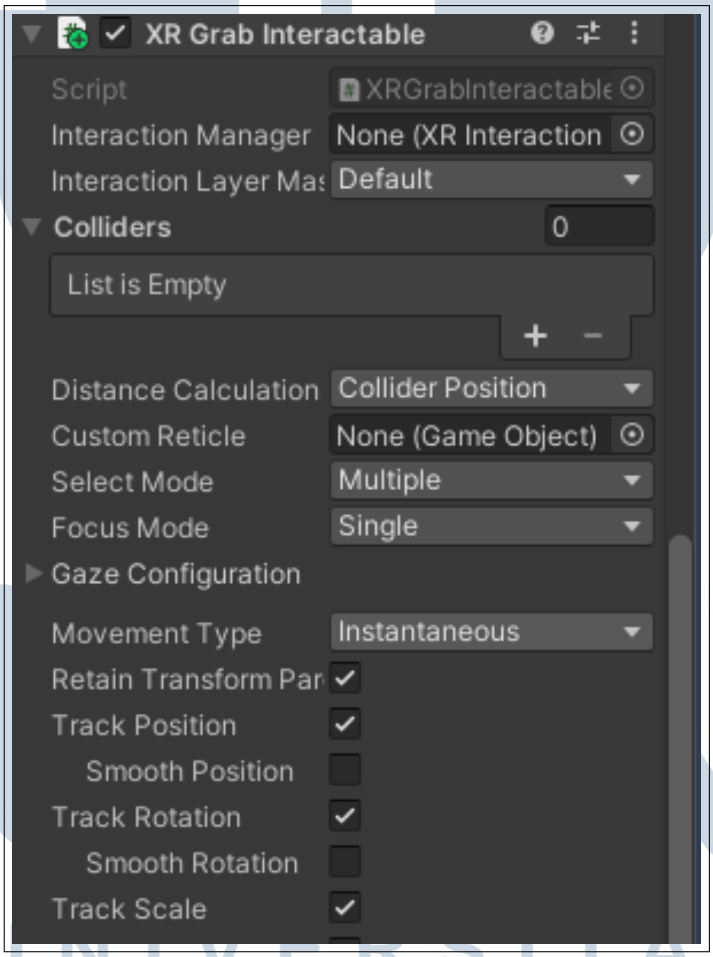
3. *XR Grab Interactable*: Komponen ini memungkinkan objek untuk diambil, dilepaskan, dan diletakkan kembali menggunakan kontroler. Interaksi seperti *hover* dan *grab* ditangani secara otomatis oleh komponen ini melalui *XR Interaction Toolkit.*

4. *XR Socket Interactor*: Pada objek mesin yang berfungsi sebagai tempat pemasangan bagian lain, *socket* disiapkan sebagai tempat *docking* untuk menerima komponen yang di-*grab* oleh pengguna. *Socket* ini akan mencocokkan bentuk dan *tag* objek interaktif yang mendekat.

5. *Layer* dan *Tag*: Setiap objek interaktif diberikan *Layer* khusus seperti *Parts*, *Grabbable*, atau *Socket* untuk memudahkan penyaringan dalam sistem *event* Unity. Penamaan *Tag* juga dilakukan untuk mendukung skrip validasi dan interaksi berdasarkan jenis objek.

Komponen *XR Grab Interactable* memungkinkan objek untuk dikenali oleh sistem XR sebagai objek interaktif. Dalam implementasi proyek ini, beberapa

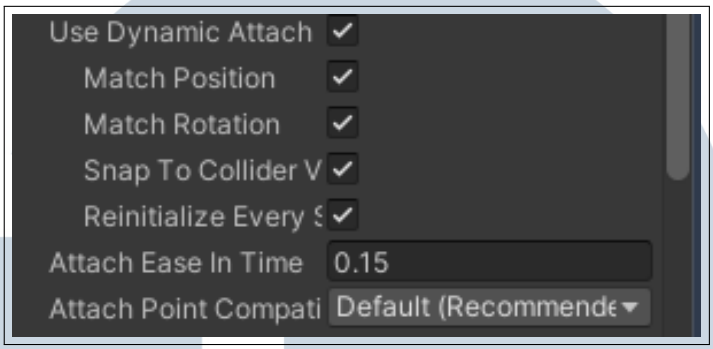
pengaturan khusus diterapkan untuk meningkatkan realisme dan fleksibilitas interaksi. Pertama, Select Mode diatur ke Multiple, memungkinkan objek untuk di-grab menggunakan kedua tangan secara bersamaan, yang sangat berguna terutama untuk objek besar atau berat. Kedua, Use Dynamic Attach dicentang agar posisi dan orientasi objek mengikuti secara natural posisi tangan pengguna saat objek diambil, menciptakan pengalaman interaksi yang lebih realistis dan intuitif. Gambar 3.26 akan menjelaskan lebih lanjut tentang pengaturan dan implementasi komponen ini dalam simulasi.



Gambar 3.26. Konfigurasi komponen *XR Grab Interactable* pada objek mesin, dengan *Select Mode* diatur ke *Multiple.*

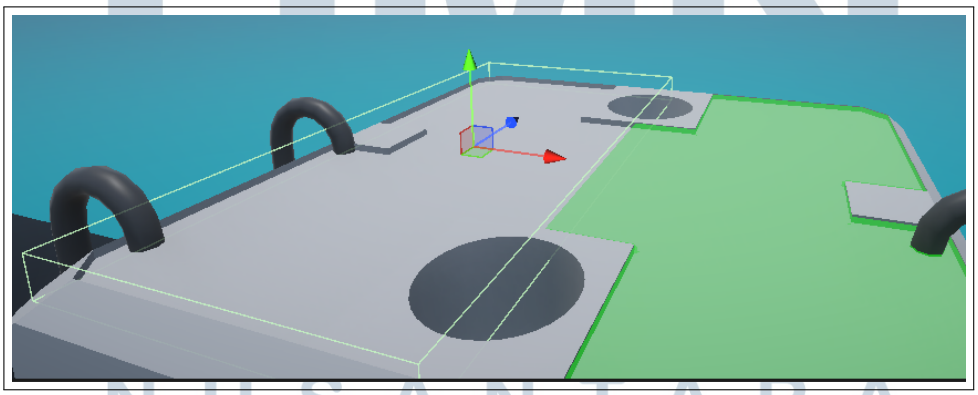
Setelah mengaktifkan pengaturan Select Mode pada komponen XR Grab Interactable, opsi Dynamic Attach juga diaktifkan. Pengaturan ini memungkinkan objek yang di-grab untuk menyesuaikan posisi dan rotasinya secara dinamis dengan titik interaksi pengguna, menciptakan pengalaman interaksi yang lebih natural dan realistis. Fitur ini sangat berguna, terutama dalam simulasi kerja manual

seperti pemasangan atau pembongkaran komponen mesin, karena memberikan kesan bahwa pengguna benar-benar memegang dan mengarahkan objek secara langsung. Tampilan antarmuka dari pengaturan ini dapat dilihat pada Gambar 3.27.



Gambar 3.27. Konfigurasi komponen *XR Grab Interactable* pada objek mesin, dengan opsi *Dynamic Attach* diaktifkan.

Langkah berikutnya dalam proses pengaturan interaksi adalah menambahkan *XR Socket Interactor* pada lokasi-lokasi tertentu dalam *scene* sebagai titik penempatan objek. Komponen ini berfungsi sebagai *socket* atau ”tempat tujuan” bagi objek yang dapat di-*grab* oleh pengguna. Ketika objek dibawa mendekati *socket*, sistem secara otomatis akan mendeteksi kompatibilitas dan memungkinkan pengguna untuk melepaskan objek ke dalam *socket* dengan tingkat presisi yang tinggi. Socket Interactor sangat penting dalam simulasi prosedural, seperti perakitan mesin, karena memastikan objek ditempatkan pada lokasi yang tepat dan terkunci baik secara visual maupun fungsional. Gambar 3.28 memberi contoh tentang *socket*



Gambar 3.28. Tata Letak *Socket* ketika *parts* mesin ingin di masukan

Tag digunakan untuk mengidentifikasi jenis objek tertentu, seperti *Parts, Socket, Tool, atau Screw*. Penamaan tag ini kemudian diterapkan dalam skrip untuk

melakukan validasi atau pemrosesan kondisi tertentu, misalnya untuk mendeteksi apakah objek yang di-grab merupakan komponen yang sesuai. Di sisi lain, Layer Mask digunakan untuk mengatur interaksi antar layer melalui sistem fisika dan XR Interaction Toolkit. Dengan menetapkan layer seperti *Grabable, SocketOnly, atau Screw*, kita dapat menentukan objek mana yang dapat di-interaksikan atau dikenali oleh komponen seperti XR Socket Interactor dan XR Ray Interactor.

D. Code Scripts

Bagian ini memuat cuplikan kode yang diimplementasikan dalam proyek VR ini. Semua kode dikembangkan menggunakan bahasa `C#` dengan memanfaatkan *XR Interaction Toolkit* dari Unity, serta didesain untuk menciptakan interaksi yang menyerupai dunia nyata.

Pada proses pemasangan objek ke dalam *socket*, perlu dilakukan penonaktifan tumbukan antar *collider*. Tujuannya adalah agar interaksi tidak mengakibatkan efek fisika seperti pantulan atau gesekan yang tidak realistis. Cuplikan pada Kode 3.2 menunjukkan bagaimana setiap *collider* dalam *socket* diatur untuk mengabaikan tumbukan dengan objek yang sedang dipasang.

```
foreach (GameObject socket in sockets)
{
    Collider[] colliders = socket.GetComponentsInChildren<Collider
    >();
    foreach (Collider col in colliders)
    {
        Physics.IgnoreCollision(currentCollider, col);
    }
}
```

Kode 3.2: Menonaktifkan tumbukan antar collider.

Setelah objek berhasil terpasang, sistem memberikan umpan balik berupa perubahan warna menjadi hijau. Hal ini dilakukan dengan mengubah material objek melalui komponen `Renderer`. Kode 3.3 menggambarkan mekanisme ini untuk memberi kejelasan status pemasangan objek kepada pengguna.

```
Renderer objRenderer = obj.GetComponent<Renderer>();
if (objRenderer != null)
{
    objRenderer.material = snapGreenMaterial;
    obj.SetActive(true);
```

```
}
```

Kode 3.3: Perubahan material objek ke hijau.

Skrip `ScNail.cs` bertugas mensimulasikan pemukulan paku. Setiap kali paku dipukul, posisinya dimodifikasi agar seolah masuk lebih dalam. Kode 3.4 juga mencakup pemutaran efek suara dan pemanggilan event `OnHammered`.

```
public void HitNail()
{
    if (!IsFullyHammered && currentSocket != null)
    {
        hammerCount++;
        currentSocket.transform.localPosition += movementDirection
    ;
        source.PlayOneShot(hit);
        OnHammered?.Invoke();
    }
}
```

Kode 3.4: Menangani logika pemukulan paku.

Untuk memastikan bahwa bagian kabinet menyatu secara fisik dan mengikuti induknya, digunakan komponen `FixedJoint`. Kode 3.5 memperlihatkan inisialisasi komponen ini tanpa batas gaya atau torsi agar tidak mudah terputus.

```
fixedJoint = gameObject.AddComponent<FixedJoint>();
fixedJoint.connectedBody = parentRigidbody;
fixedJoint.breakForce = Mathf.Infinity;
fixedJoint.breakTorque = Mathf.Infinity;
```

Kode 3.5: Penggabungan kabinet menggunakan FixedJoint.

Deteksi kontak antara kepala palu dan paku penting untuk menentukan kapan fungsi `HitNail()` dipanggil. Seperti ditunjukkan pada Kode 3.6, sistem hanya menanggapi jika kontak terjadi secara tepat antara dua objek tersebut.

```
if (contact.thisCollider == hammerHeadCollider && hitObject.
    CompareTag("Nail"))
{
    ScNail nail = hitObject.GetComponent<ScNail>();
    if (nail != null) nail.HitNail();
}
```

Kode 3.6: Deteksi tumbukan palu dengan paku.

Agar menu selalu muncul dalam jangkauan pengguna, sistem menempatkannya relatif terhadap posisi HMD. Implementasi pada Kode 3.7 memperlihatkan perhitungan vektor posisi dan rotasi menu.

```
menu.transform.position = head.position + new Vector3(head.forward
    .x, 0, head.forward.z).normalized * spawnDistance;
menu.transform.LookAt(new Vector3(head.position.x, menu.transform.
    position.y, head.position.z));
menu.transform.forward *= -1;
```

Kode 3.7: Penempatan menu dinamis.

Untuk memperjelas objek mana yang sedang dalam status interaktif, skrip akan mengganti material saat *hover*. Seperti pada Kode 3.8, objek yang disentuh oleh pengontrol akan disorot secara visual.

```
if (args.interactorObject.transform.CompareTag("Controller"))
{
    originalMaterial = objectRenderer.material;
    objectRenderer.material = hoverMaterial;
}
```

Kode 3.8: Perubahan material saat hover.

Ketika pengguna melepaskan objek dari socket, fungsi `OnDetached()` akan dijalankan untuk mengatur ulang status objek. Kode 3.9 menunjukkan bahwa tag objek dikembalikan menjadi `Parts`, material dipulihkan, serta tumbukan antar collider diaktifkan kembali. Setelah itu, proses pembongkaran dilanjutkan secara bertahap.

```
    if (args.interactorObject is XRSocketInteractor xrSocket)m {
        gameObject.tag = "Parts";
        foreach (GameObject obj in currentHighlight) {
            Renderer renderer = obj.GetComponent<Renderer>();
            if (renderer != null){
                renderer.material = defaultMaterial; }}
        Collider currentCollider = GetComponent<Collider>();
        GameObject[] sockets = GameObject.FindGameObjectsWithTag("
    Socket");
        foreach (GameObject socket in sockets){
            Collider[] colliders = socket.GetComponentsInChildren<
    Collider>();
            foreach (Collider col in colliders){
                Physics.IgnoreCollision(currentCollider, col,
    false);}}
```

```
13          StartCoroutine(ProcessDisassembly());}}
```

Kode 3.9: Disassembly: fungsi OnDetached().

Setelah pembongkaran, objek baru harus diregistrasi ulang agar bisa merespons event interaksi. Seperti ditunjukkan pada Kode 3.10, listener untuk seleksi, hover, dan pelepasan disambungkan kembali ke metode-metode terkait.

```
1  ScDiesDisassembly disassemblyScript = obj.GetComponent<
       ScDiesDisassembly>();
2  if (disassemblyScript != null)
3  {
4      objGrab.selectEntered.AddListener(disassemblyScriPTOnAttached)
       ;
5      objGrab.selectExited.AddListener(disassemblyScriPTOnDetached);
6  }
7
8  ScHoverObject hoverScript = obj.GetComponent<ScHoverObject>();
9  if (hoverScript != null)
10 {
11     objGrab.hoverEntered.AddListener(hoverScriPTOnHoverEnter);
12     objGrab.hoverExited.AddListener(hoverScriPTOnHoverExit);
13     objGrab.selectExited.AddListener(hoverScriPTOnDetach);
14 }
```

Kode 3.10: Register ulang event listener.

Saat objek dipasang ke dalam socket, fungsi `OnAttached()` akan mengatur ulang status objek, mengganti material, dan menonaktifkan tumbukan dengan socket. Kode 3.11 merepresentasikan proses ini secara lengkap.

```
1  public void OnAttached(SelectEnterEventArgs args){
2      gameObject.tag = "Attached";
3      foreach (GameObject obj in currentHighlight){
4          Renderer renderer = obj.GetComponent<Renderer>();
5          if (renderer != null){
6              renderer.material = highlightMaterial;}}
7      Collider currentCollider = GetComponent<Collider>();
8      GameObject[] sockets = GameObject.FindGameObjectsWithTag("
       Socket");
9      foreach (GameObject socket in sockets){
10         Collider[] colliders = socket.GetComponentsInChildren<
       Collider>();
11         foreach (Collider col in colliders){
```

```
12              Physics.IgnoreCollision(currentCollider, col);}}}
```

Kode 3.11: Assembly: fungsi OnAttached().

Setelah perakitan selesai, objek digabungkan secara fisik ke induknya menggunakan `FixedJoint` agar seluruh bagian bergerak sebagai satu kesatuan. Implementasinya dapat dilihat pada Kode 3.12.

```
1 fixedJoint = gameObject.AddComponent<FixedJoint>();
2 fixedJoint.connectedBody = parentRigidbody;
3 fixedJoint.breakForce = Mathf.Infinity;
4 fixedJoint.breakTorque = Mathf.Infinity;
```

Kode 3.12: Assembly: fungsi Combine().

Fungsi `HitNail()` pada skrip `ScScrewRemove.cs` menangani animasi dan suara saat sekrup diputar dalam proses pembongkaran. Kode 3.13 memperlihatkan detail pemrosesan tiap kali sekrup diputar.

```
1  public void HitNail()
2  {
3      if (!IsFullyHammered && currentSocket != null)
4      {
5          hammerCount++;
6          currentSocket.transform.localPosition += movementDirection
       ;
7          source.PlayOneShot(hit);
8          OnHammered?.Invoke();
9      }
10 }
```

Kode 3.13: Disassembly screw: fungsi HitNail().

Setelah semua sekrup dilepas, fungsi `CheckAllScrews()` pada Kode 3.14 akan memverifikasi status masing-masing dan memicu event kelanjutan bila semua selesai.

```
1 public void CheckAllScrews(){
2     int screwCount = 0;
3     foreach (GameObject screw in screws){
4         ScScrewRemove screwScript = screw.GetComponent<
      ScScrewRemove>();
5         if (screwScript != null && screwScriPTIsFullyRemoved){
6             screwCount++;}}
7     if (screwCount == screws.Length){
8         OnAllScrewsRemoved?.Invoke();}}
```

Kode 3.14: Disassembly screw: fungsi CheckAllScrews().

Deteksi awal saat kunci pas menyentuh kepala sekrup dilakukan melalui fungsi `OnTriggerEnter()`. Kode 3.15 menangkap momen interaksi tersebut dan memicu proses pembongkaran.

```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Screw"))
    {
        ScScrewRemove screw = other.GetComponent<ScScrewRemove>();
        if (screw != null) screw.HitNail();
    }
}
```

Kode 3.15: Disassembly screw: fungsi OnTriggerEnter().

Pada tahap perakitan, fungsi `HitNail()` dalam skrip `ScScrewAttach.cs` digunakan untuk memutar sekrup hingga terpasang sempurna. Mekanisme ini menambahkan realisme pada simulasi dengan memutar posisi sekrup serta memutar suara. Kode 3.16 menunjukkan implementasinya.

```
public void HitNail()
{
    if (!IsFullyHammered && currentSocket != null)
    {
        hammerCount++;
        currentSocket.transform.localPosition += movementDirection
    ;
        source.PlayOneShot(hit);
        OnHammered?.Invoke();
    }
}
```

Kode 3.16: Assembly screw: fungsi HitNail().

Untuk memastikan bahwa semua sekrup sudah dipasang, sistem menggunakan fungsi `CheckAllScrews()` untuk menghitung jumlah sekrup yang telah dalam status terpasang. Bila semua sekrup telah terpasang sempurna, event `OnAllScrewsHammered` akan dipanggil untuk menandai akhir proses pemasangan. Lihat Kode 3.17.

```
public void CheckAllScrews()
{
    int screwCount = 0;
    foreach (GameObject screw in screws)
    {
        ScScrewAttach screwScript = screw.GetComponent<
    ScScrewAttach>();
        if (screwScript != null && screwScriPTIsFullyHammered)
        {
            screwCount++;
        }
    }

    if (screwCount == screws.Length)
    {
        OnAllScrewsHammered?.Invoke();
    }
}
```

Kode 3.17: Assembly screw: fungsi CheckAllScrews().

Untuk meningkatkan pengalaman interaksi pengguna, sistem menyediakan feedback visual ketika objek sedang di-hover oleh pengontrol. Fungsi `OnHoverEnter()` dari skrip `ScHoverNew.cs` menangani perubahan material menjadi material highlight ketika kondisi aktif dan kontak oleh pengontrol terdeteksi. Detail implementasi dapat dilihat pada Kode 3.18.

```
public void OnHoverEnter(HoverEnterEventArgs args)
{
    if (isActive && args.interactorObject.transform.CompareTag("
    Controller"))
    {
        Renderer renderer = GetComponent<Renderer>();
        if (renderer != null)
        {
            renderer.material = highlightMaterial;
        }
    }
}
```

Kode 3.18: Fungsi OnHoverEnter untuk umpan balik visual.

E. Hasil Implementasi

Hasil dari proses pengembangan aplikasi Virtual Reality (VR) meliputi implementasi sebuah prototipe yang dapat digunakan untuk memvisualisasikan proses perakitan dan pembongkaran mesin produksi secara interaktif, realistis, dan imersif. Aplikasi ini dirancang agar dapat berfungsi sebagai media pelatihan bagi operator maupun teknisi, sekaligus menjadi alat bantu untuk mempermudah proses pemahaman alur kerja mesin tanpa harus menggunakan mesin fisik secara langsung.

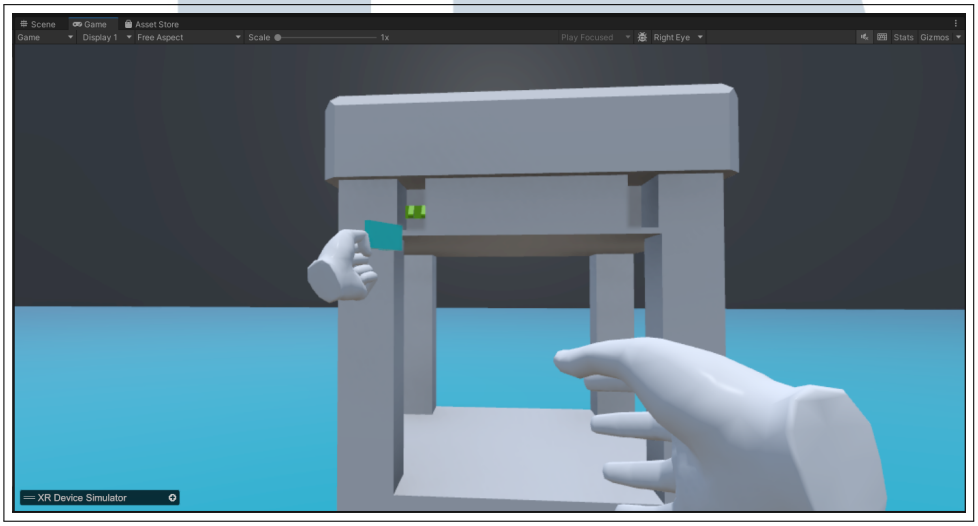
Seluruh pengembangan difokuskan pada pencapaian pengalaman pengguna *user experience* yang optimal, dengan mempertimbangkan aspek ergonomi, keakuratan interaksi, serta performa sistem di perangkat VR. Beberapa komponen utama yang telah berhasil diimplementasikan dalam prototipe ini meliputi antarmuka pengguna, prosedur interaktif, umpan balik sensorik, serta sistem dokumentasi digital.

Tahap pertama yang diimplementasikan adalah antarmuka utama simulasi. Antarmuka ini dirancang agar pengguna dapat langsung memasuki lingkungan kerja virtual secara intuitif. Elemen-elemen penting seperti menu navigasi, instruksi pelatihan, serta pengaturan dasar ditampilkan dalam ruang VR dengan tata letak yang mudah diakses. Pengembangan antarmuka ini memanfaatkan komponen dari *XR Interaction Toolkit* yang dikombinasikan dengan elemen kustom agar sesuai dengan kebutuhan simulasi. Tampilan antarmuka utama dari simulasi VR ditunjukkan secara visual pada Gambar 3.29.

Gambar 3.29. Tampilan antarmuka utama simulasi VR.

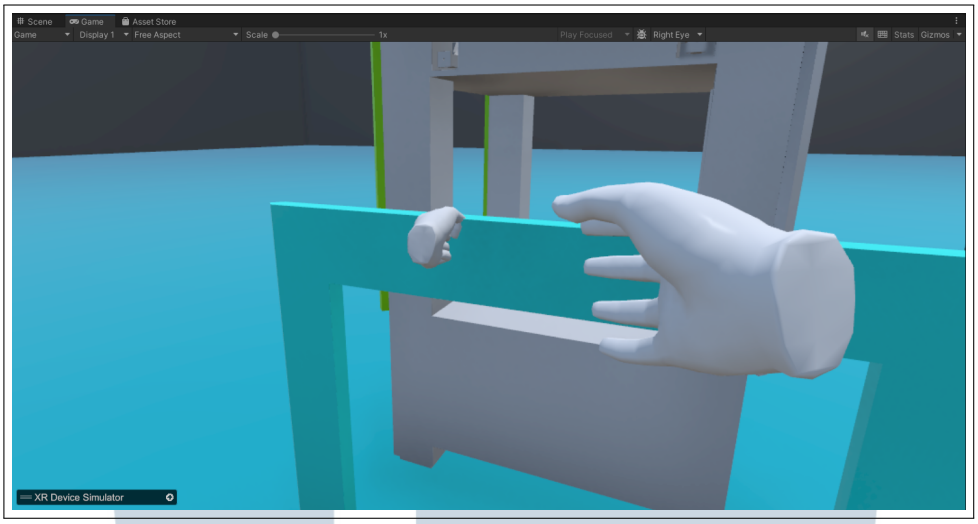
Selanjutnya, implementasi prosedur interaktif perakitan menjadi fokus utama pengembangan. Pada tahap ini, seluruh alur perakitan mesin divisualisasikan secara

dinamis di dalam lingkungan VR. Pengguna dipandu melalui langkah-langkah perakitan menggunakan kombinasi instruksi teks, elemen penunjuk visual seperti *highlight* dan panah arah, serta kontrol interaksi berbasis *controller*. Pengguna dapat mengambil, memindahkan, memutar, dan memasang komponen mesin dengan tingkat presisi yang tinggi. Seluruh interaksi ini didesain untuk menyerupai proses kerja nyata, sehingga mendukung proses pelatihan yang efektif dan aman. Ilustrasi dari prosedur perakitan interaktif ini dapat dilihat pada Gambar 3.30.



Gambar 3.30. Prosedur interaktif perakitan mesin.

Selain proses perakitan, sistem juga memungkinkan pengguna untuk melakukan proses pembongkaran mesin secara interaktif. Mirip dengan prosedur perakitan, proses pembongkaran dilakukan dengan panduan visual yang jelas, sehingga pengguna dapat memahami urutan yang benar dalam melepas komponen. Validasi dilakukan secara otomatis agar urutan yang dijalankan sesuai dengan prosedur yang telah ditentukan, sehingga kesalahan pembongkaran dapat dihindari. Pengguna juga mendapatkan umpan balik langsung jika mencoba melakukan tindakan yang tidak sesuai. Visualisasi dari prosedur interaktif pembongkaran mesin ditampilkan pada Gambar 3.31.



Gambar 3.31. Prosedur interaktif pembongkaran mesin.

Implementasi *Interface Guide*, panduan atau instruksi yang dirancang untuk membantu pengguna memahami cara berinteraksi dengan antarmuka perangkat lunak atau sistem. Panduan ini biasanya mencakup petunjuk visual dan deskriptif mengenai elemen-elemen antarmuka, seperti tombol, menu, dan ikon, serta cara penggunaannya. *Interface Guide* bertujuan untuk memastikan pengalaman pengguna yang intuitif dan efisien dengan memberikan informasi yang jelas tentang fungsi dan tujuan setiap komponen pada antarmuka. Dalam konteks pengembangan aplikasi atau perangkat lunak, *Interface Guide* juga berperan penting dalam desain user experience (UX), membantu pengguna untuk lebih mudah menavigasi sistem dan mencapai tujuan mereka tanpa kebingunguan atau kesalahan. Implementasi *interface guide* bisa dilihat pada Gambar 3.32

Gambar 3.32. Prosedur interaktif instruksi pembongkaran dan perakitan mesin.

F. Pengujian dan Validasi

Setelah proses implementasi selesai dilakukan, dilakukan tahap pengujian dan validasi untuk memastikan bahwa seluruh fungsi aplikasi VR berjalan sesuai dengan rancangan serta memenuhi ekspektasi pengguna. Proses pengujian ini bertujuan untuk mengidentifikasi potensi *bug*, memastikan interaksi yang intuitif, serta menjamin performa aplikasi tetap optimal selama digunakan.

Metodologi pengujian yang diterapkan meliputi serangkaian *test case* yang telah dirumuskan secara sistematis. Setiap *test case* dirancang untuk mengevaluasi aspek-aspek penting aplikasi, mulai dari proses inisialisasi, interaksi pengguna, penggunaan alat virtual, umpan balik, hingga performa sistem secara keseluruhan.

Rincian lengkap dari seluruh skenario pengujian tersebut disajikan pada Tabel 3.5 yang berisi daftar *test case* beserta deskripsi masing-masing tujuan pengujian.

Tabel 3.5. Dokumentasi *test case*

Test Case	Deskripsi
TC-01	Memverifikasi bahwa aplikasi dapat diluncurkan dengan sukses tanpa mengalami *crash*.
TC-02	Memastikan bahwa lingkungan VR terinisialisasi dengan benar, termasuk pelacakan *controllers* dan *headset*.
TC-03	Memvalidasi bahwa layar pemuatan menampilkan progres atau informasi relevan selama proses inisialisasi.
TC-04	Memverifikasi bahwa objek dimuat dengan benar di lingkungan VR.
TC-05	Memastikan bahwa pengguna dapat memilih dan menyorot komponen individual dari objek.
TC-06	Menguji kemampuan untuk mengambil, memindahkan, dan memutar komponen objek menggunakan *VR controllers*.
TC-07	Memvalidasi presisi interaksi, seperti kemampuan untuk memasang komponen dengan tepat *snapping*.
TC-08	Memastikan bahwa tindakan yang salah memicu umpan balik yang sesuai (misalnya, getaran, suara, atau pesan kesalahan).
TC-09	Memverifikasi bahwa komponen dapat dirakit dalam urutan yang benar.

Tabel 3.5. Dokumentasi *test case* (lanjutan)

Test Case	Deskripsi
TC-10	Memastikan bahwa panduan langkah demi langkah atau *hints* tersedia untuk membantu proses perakitan (jika diterapkan).
TC-11	Memeriksa apakah percobaan perakitan yang tidak sejajar diblokir atau dikoreksi secara otomatis.
TC-12	Menguji fungsi alat virtual (seperti obeng, kunci pas, dan sebagainya), jika ada.
TC-13	Memvalidasi notifikasi atau indikasi penyelesaian setelah proses perakitan selesai.
TC-14	Memverifikasi bahwa komponen dapat dibongkar dalam urutan yang benar.
TC-15	Memastikan bahwa alat yang dibutuhkan tersedia dan berfungsi dengan baik selama proses pembongkaran.
TC-16	Memeriksa apakah pengguna dapat melepas komponen dengan bebas tanpa merusak alur proses.
TC-17	Memvalidasi bahwa peringatan atau instruksi muncul jika pengguna melakukan tindakan pembongkaran yang salah.
TC-18	Memverifikasi bahwa kontrol VR (*buttons, joysticks, gestures*) berfungsi sebagaimana mestinya.
TC-19	Memastikan bahwa menu, *toolbar*, atau elemen *HUD (Head-Up Display)* dapat diakses dan digunakan dengan nyaman dalam VR.
TC-20	Memvalidasi bahwa teks *tooltip* atau instruksi dapat dibaca dengan jelas dan diposisikan dengan benar dalam ruang VR.
TC-21	Memeriksa apakah lingkungan VR bebas dari gangguan visual atau glitch grafis.
TC-22	Memastikan bahwa pencahayaan, bayangan, dan tekstur dirender dengan benar.
TC-23	Memvalidasi umpan balik audio untuk tindakan seperti mengambil, merakit, atau membongkar komponen.
TC-24	Menguji apakah suara lingkungan atau musik latar bersifat imersif dan tidak mengganggu.
TC-25	Memverifikasi bahwa aplikasi mempertahankan laju *frame rate* yang stabil selama digunakan.

Tabel 3.5. Dokumentasi *test case* (lanjutan)

Test Case	Deskripsi
TC-26	Memeriksa apakah penggunaan memori dan GPU berada dalam batas yang dapat diterima.
TC-27	Menguji tingkat latensi pada interaksi (misalnya saat mengambil atau memasang komponen).
TC-28	Memastikan bahwa aplikasi dapat menangani skenario seperti koneksi yang terputus atau *headset disconnection* dengan baik.
TC-29	Memverifikasi bahwa pesan kesalahan jelas dan membantu pengguna dalam menyelesaikan masalah.
TC-30	Memeriksa apakah tindakan tidak valid atau tidak didukung tidak menyebabkan aplikasi *crash*.
TC-31	Menguji kejelasan instruksi yang diberikan untuk proses perakitan dan pembongkaran.
TC-32	Memverifikasi kemudahan akses fungsi *restart* atau *reset* saat menjalankan tugas.
TC-33	Memastikan bahwa pengguna dapat keluar dari aplikasi dengan lancar dan menyimpan progres jika diperlukan.
TC-34	Memvalidasi bahwa teks dan elemen UI dapat dibaca oleh pengguna dengan berbagai tinggi badan.
TC-35	Memastikan kompatibilitas dengan konfigurasi *VR controller* untuk pengguna bertangan kiri maupun kanan.
TC-36	Menguji ketersediaan opsi ramah bagi pengguna tunanetra warna, seperti mode kontras tinggi.

Dengan pengujian yang komprehensif seperti ini, diharapkan aplikasi VR dapat memberikan pengalaman yang mulus, interaktif, dan intuitif bagi pengguna akhir. Selain itu, proses ini juga membantu tim pengembang dalam mengidentifikasi potensi perbaikan dan optimalisasi sebelum aplikasi dirilis secara resmi.

3.6 Kendala dan Solusi yang Ditemukan

Selama proses pengembangan aplikasi Virtual Reality (VR), tim pengembang menghadapi berbagai kendala yang cukup kompleks, baik dari sisi teknis maupun non-teknis. Beberapa hambatan utama yang dihadapi dirangkum sebagai berikut:

1. Kurangnya Dokumentasi Teknis Mesin Produksi
   Informasi teknis terkait mesin produksi yang akan disimulasikan sering kali tidak memadai. Data yang diberikan oleh pengguna atau pemilik mesin cenderung terbatas, tidak terstruktur, atau kurang mendalam. Akibatnya, tim pengembang kesulitan dalam memetakan alur kerja mesin secara akurat untuk keperluan simulasi VR. Proses desain interaksi pun menjadi lebih lambat karena pengembang harus memahami mekanisme mesin secara manual terlebih dahulu.

2. Kompleksitas Aset 3D dari Tim Desain Awal
   Model 3D yang diterima memiliki tingkat kompleksitas yang tinggi, seperti jumlah poligon yang besar, material beresolusi tinggi, dan struktur *mesh* yang tidak teroptimalkan. Saat diimpor ke *Unity Engine*, hal ini menyebabkan penurunan performa yang signifikan pada perangkat VR, terutama pada *headset* standalone. Dampaknya meliputi penurunan *frame rate*, latensi tinggi, dan pengalaman pengguna yang tidak nyaman.

3. Kesulitan Implementasi Interaksi Kompleks di VR
   Tidak semua interaksi di dunia nyata dapat langsung diterapkan ke lingkungan digital VR. Aksi fisik seperti rotasi multi-aksi, pemasangan presisi, atau deteksi interaksi bersyarat memerlukan logika pemrograman yang kompleks. Selain itu, keterbatasan sistem fisika di *Unity Engine* membuat simulasi dengan akurasi tinggi menjadi menantang.

Untuk mengatasi berbagai hambatan tersebut, tim pengembang menerapkan sejumlah pendekatan strategis yang berfokus pada efisiensi teknis, peningkatan performa, dan akurasi interaksi. Langkah-langkah yang diambil adalah sebagai berikut:

1. Observasi Lapangan dan Dokumentasi Visual
   Melalui kunjungan ke pabrik dan wawancara dengan teknisi berpengalaman, diperoleh pemahaman yang lebih mendalam mengenai proses kerja dan karakteristik komponen mesin. Dokumentasi visual berupa foto dan video dari proses perakitan dan pembongkaran juga dilakukan, sehingga dapat dijadikan referensi utama dalam pengembangan alur interaksi di aplikasi VR.

2. Optimasi Aset 3D untuk Performa Maksimal
   Dilakukan serangkaian proses optimasi seperti *mesh decimation* untuk

mengurangi jumlah poligon tanpa mengorbankan kualitas visual secara signifikan. Teknik *texture baking* juga diterapkan guna menyederhanakan lapisan material dan pencahayaan. Selain itu, sistem *Level of Detail (LOD)* dinamis digunakan untuk menampilkan model dalam kualitas berbeda sesuai jaraknya dari pengguna.

3. Pengembangan Skrip Kustom dan Pengujian Pengguna
   Tim mengembangkan skrip kustom di atas *XR Interaction Toolkit* untuk menangani interaksi fisik kompleks seperti rotasi bersyarat dan validasi urutan perakitan. Interaksi diuji secara iteratif melalui *user testing* dengan melibatkan pengguna internal maupun eksternal, guna memastikan pengalaman yang natural dan intuitif.

Dengan kombinasi berbagai strategi tersebut, berbagai kendala utama yang dihadapi selama pengembangan aplikasi VR dapat diatasi secara efektif. Proses ini sekaligus menjadi pembelajaran penting bagi tim pengembang dalam menghadapi tantangan pengembangan aplikasi VR di masa mendatang.