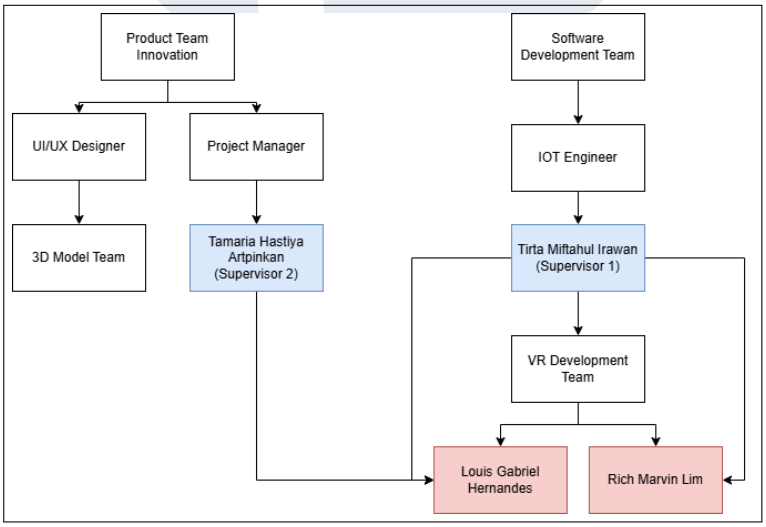
BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama masa pelaksanaan program magang di PT Kalbe Farma Tbk (Saka Farma Laboratories), posisi yang dijalankan adalah VR/XR Developer di bawah bimbingan langsung Bapak Miftahul Tirta Irawan dari tim IoT Engineer. Penempatan ini merupakan bagian dari inisiatif perusahaan dalam mengembangkan media pelatihan berbasis Virtual Reality (VR) yang bertujuan untuk menyimulasikan prosedur pemasangan serta pembongkaran mesin tablet pressing di pabrik Saka Farma. Proyek ini mendukung transformasi digital proses pelatihan teknis melalui teknologi imersif yang interaktif dan mudah diadaptasi oleh pengguna lapangan. Struktur menggambarkan hubungan koordinatif dalam pelaksanaan magang dapat dilihat pada Gambar 3.1.



Gambar 3.1. Struktur Organisasi PT Saka Farma Laboratories (Kalbe Consumer Health)
Sumber: [10]

Kegiatan pengembangan dilakukan melalui kolaborasi lintas divisi yang mencakup tim *VR Development*, *3D Model*, dan *Project Management* (PM). Tim VR bertanggung jawab terhadap keseluruhan proses pengembangan mulai dari perancangan fitur, implementasi logika interaktif, hingga proses pengujian aplikasi. Tim 3D Model berfokus pada pembuatan aset tiga dimensi mesin secara akurat

sesuai bentuk dan fungsi aslinya untuk diintegrasikan ke dalam lingkungan virtual. Sementara itu, tim PM berperan dalam penyusunan alur simulasi (*flow*), koordinasi kebutuhan teknis lintas tim, dan validasi skenario berdasarkan masukan dari Saka Farma.

Selama proses pengembangan, komunikasi antar tim dilakukan secara daring menggunakan WhatsApp dan Microsoft Teams, sedangkan dokumentasi serta pelacakan kemajuan proyek dikelola melalui GitHub, SharePoint, Asana, dan Notion. Perancangan flow simulasi dilakukan menggunakan Draw.io, sedangkan Unity menjadi platform utama untuk membangun dan menguji aplikasi VR. Rapat koordinasi mingguan diadakan untuk evaluasi progres, pengkajian kendala teknis, serta penentuan langkah tindak lanjut berdasarkan hasil uji coba internal maupun eksternal.

Sebagai bagian dari tahap validasi, dilakukan kegiatan lapangan (on-site testing) bersama tim Saka Farma untuk memastikan kesesuaian prosedural antara simulasi virtual dan proses kerja aktual. Aktivitas ini mencakup observasi langsung terhadap proses *Clean-Up Set-Up* (CUSU), pengujian skenario interaksi, serta pengumpulan umpan balik dari operator dan *Subject Matter Expert* (SME). Catatan temuan dari hasil uji lapangan—seperti ketidaksesuaian urutan langkah, kebutuhan tambahan efek suara atau visual, serta penyempurnaan sistem interaksi—dijadikan dasar bagi iterasi pengembangan berikutnya. Melalui pendekatan iteratif dan kolaboratif ini, pengembangan aplikasi VR dapat berjalan lebih efisien, presisi, dan selaras dengan kebutuhan pengguna industri.

3.2 Tugas yang Dilakukan

Selama menjalani program magang di PT Saka Farma Laboratories (*Kalbe Consumer Health*) pada posisi *VR Developer Intern*, berbagai aktivitas utama yang dilakukan dapat dirangkum sebagai berikut.

1. Mengembangkan media pelatihan berbasis *Virtual Reality* (VR) yang berfungsi untuk mensimulasikan prosedur operasional mesin produksi. Aplikasi ini dibuat secara interaktif serta imersif agar peserta pelatihan dapat memahami tahapan kerja dengan lebih praktis melalui visualisasi tiga dimensi yang menyerupai kondisi sebenarnya.

2. Memanfaatkan *Unity* sebagai platform utama pengembangan, dengan dukungan *Visual Studio Code* untuk pengelolaan kode program dan

Git/*GitHub* sebagai sistem pengendalian versi dan manajemen kolaborasi. Pengaturan versi dilakukan menggunakan metode *branching* sesuai praktik pengembangan perangkat lunak di lingkungan teknis Kalbe.

3. Merancang alur logika perakitan (*assembly logic*) dan sistem validasi berbasis skenario agar alur simulasi berjalan sesuai urutan prosedur sebenarnya. Komponen interaksi utama menggunakan *XR Grab Interactable* yang memungkinkan pengguna melakukan perakitan mesin secara realistis.

4. Mengintegrasikan fitur tambahan seperti animasi tangan kontekstual yang merespons gerakan pengguna, serta *step-by-step guide menu* sebagai panduan interaktif dalam proses pelatihan virtual.

5. Menerapkan modul teknologi dari *XR Toolkit* termasuk *XR Ray Interactor*, *XR Origin Rig*, dan *Unity Input System* untuk menghadirkan pengalaman pengguna yang ergonomis, responsif, dan menyerupai kondisi kerja di dunia nyata.

6. Mengembangkan beberapa fitur eksperimental, di antaranya *object snapping precision*, *dynamic assembly feedback*, serta optimalisasi *render pipeline* guna menjaga performa visual tetap stabil tanpa menurunkan kualitas tampilan.

7. Terlibat dalam sesi rutin seperti *code review*, diskusi teknis, serta evaluasi bersama supervisor untuk memastikan arah pengembangan dan kualitas implementasi sesuai target proyek.

8. Melaksanakan koordinasi tim secara daring dan luring melalui *Microsoft Teams* serta *WhatsApp*. Model komunikasi ini memastikan kolaborasi berjalan efektif, khususnya dalam pembaruan progres, pembagian tugas, dan penyelesaian kendala teknis selama proses pengembangan.

Seluruh kegiatan berfokus pada peningkatan kualitas dan stabilitas sistem VR sebagai media pelatihan internal di lingkungan *Kalbe Consumer Health*. Rincian aktivitas mingguan selama periode proyek dapat dilihat pada Tabel 3.1.

Tabel 3.1. Rincian kegiatan mingguan selama periode magang

Minggu	Kegiatan
1	Kegiatan awal difokuskan pada penyusunan struktur proyek dan pengembangan awal beberapa *scene* pelatihan. Selain itu dilakukan pula perbaikan awal terhadap bug yang ditemukan selama proses implementasi.
2	Melanjutkan proses pembuatan dan penyesuaian *scene setup* serta melakukan penyempurnaan elemen interaktif agar lebih responsif terhadap mekanisme pelatihan.
3	Melaksanakan *internal testing* terhadap beberapa *scene* yang telah selesai dikembangkan dan memperbaiki hasil berdasarkan masukan dari tahap pengujian tersebut.
4	Fokus kegiatan pada minggu ini adalah perbaikan visual dan fungsi pada *scene* yang sudah ada, termasuk penyesuaian ukuran dan tata letak komponen untuk meningkatkan konsistensi tampilan.
5	Melanjutkan tahapan *internal testing* setelah revisi dilakukan, dengan tambahan *sound effect* untuk meningkatkan pengalaman audiovisual pengguna.
6	Melakukan penggantian beberapa komponen dan menambahkan elemen baru pada *scene*; dilaksanakan pula pertemuan koordinasi dengan pihak *Saka Farma* untuk sinkronisasi kebutuhan teknis proyek.
7	Meneruskan pengembangan elemen pada bagian *Clean-Up process*, meliputi penyusunan urutan langkah serta optimalisasi interaksi antar objek.
8	Kegiatan masih berfokus pada penyempurnaan dan finalisasi *scene clean-up* agar seluruh logika dan interaksinya berjalan sesuai alur prosedural.
9	Mengembangkan *scene setup* tambahan serta memperluas cakupan *Clean-Up scene* agar keduanya memiliki kesinambungan logika yang baik dan terintegrasi secara fungsional.
10	Melakukan *internal testing* lanjutan serta memperbaiki berbagai bug yang teridentifikasi selama proses uji coba sebelumnya.

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

Minggu	Kegiatan
11	Menyusun ulang struktur proyek dengan membagi *scene* menjadi beberapa bagian agar manajemen proyek lebih efisien; menyelesaikan pengembangan menu *tutorial* interaktif.
12	Melaksanakan *internal testing* terhadap keseluruhan *scene* yang telah dikembangkan, sekaligus menambahkan *sound effect* pelengkap untuk memberikan umpan balik yang lebih realistis.
13	Mengadakan sesi uji coba bersama pihak *Saka Farma* guna mendapatkan masukan langsung; hasil pengujian digunakan untuk melakukan peningkatan kualitas dan stabilitas beberapa *scene utama*.
14	Melakukan pembaruan bahasa pada *tutorial menu* agar selaras dengan kebutuhan pengguna serta melakukan peningkatan desain dan navigasi pada antarmuka pengguna (*UI Menu*).
15	Mengubah dan memperbarui sejumlah *script logic* serta menyesuaikan alur proses sistem agar lebih efisien dan mudah dikelola.
16	Melaksanakan *internal testing* lanjutan untuk memastikan kestabilan aplikasi dan memperbaiki bug yang masih ditemukan selama pengujian.
17	Melakukan *bug fixing* tambahan terhadap *scene-scene* yang telah dikembangkan sebelumnya, memastikan performa sistem lebih konsisten dan reliabel.
18	Fokus kegiatan berpindah pada tahap awal pengembangan *Exam Mode*, termasuk penyusunan alur serta mekanisme evaluasi digital di dalam aplikasi VR.
19	Melanjutkan proses penyempurnaan dan ekspansi fitur pada *Exam Mode*, dengan peningkatan logika penilaian serta optimisasi fungsionalitas interaktifnya.

3.3 Perangkat Penunjang Pelaksanaan Magang

Pengembangan simulasi *Virtual Reality* (VR) selama pelaksanaan magang di *Kalbe Consumer Health* didukung oleh kombinasi berbagai perangkat lunak, perangkat keras, serta kerangka kerja dan pustaka pengembangan. Rincian lengkap mengenai perangkat dan teknologi yang digunakan dapat dilihat pada Tabel 3.2, Tabel 3.3, Tabel 3.4 dan Tabel 3.5.

Tabel 3.2. Daftar perangkat lunak dan fungsinya untuk pengembangan VR Unity

Nama/Versi	**Deskripsi dan Fungsi**
Visual Studio Code (v1.100.0)	Editor kode utama yang mendukung integrasi Unity, C#, serta ekstensi AI coding assistant untuk mempercepat pengembangan dan debugging
Unity Editor LTS (2023.2.18f1)	Platform utama pengembangan, simulasi, serta pengujian aplikasi VR modern
GitHub Desktop (v3.5.10)	Antarmuka GUI untuk version control Git dan kerja tim secara kolaboratif
C# (v12.0)	Bahasa pemrograman scripting untuk logika aplikasi serta interaksi VR
Blender (v4.0 LTS)	Alat pembuatan, editing, dan animasi model 3D untuk asset VR
OpenXR Tools for Unity (v1.10)	Mendukung standar multiplatform VR; memudahkan integrasi headset VR
Meta Quest Link (v63.0)	Debug serta tes aplikasi langsung pada device Meta Quest 2/3 atau Pro
NVIDIA Omniverse Connect (2025.1)	Penghubung workflow Unity ke pipeline produksi 3D dan AI rendering berbasis RTX

Tabel 3.3. Spesifikasi perangkat keras untuk pengembangan VR Unity

Kategori	Spesifikasi
Minimum PC/Laptop	Intel Core i5-12400 / AMD Ryzen 5 5600X (6 core), RAM 16 GB DDR4, GPU GTX 1660 Super / Radeon RX 580, SSD 512GB + HDD 1TB, Windows 10/11
Recommended PC/Laptop	Intel Core i7-13700K / AMD Ryzen 7 7800X (8 core/16 thread), RAM 32 GB DDR5, GPU RTX 4070/4080 / Radeon RX 7900 XT, SSD NVMe 1TB, Windows 11 Pro
VR Headset	Meta Quest 2/3, Valve Index, HTC Vive Pro 2, PICO 4, support OpenXR
Display	Full-HD (1920x1080) minimal, QHD/4K 120Hz+ direkomendasikan

Tabel 3.4. Minimum dan Rekomendasi Spesifikasi untuk VR Unity

Kategori	Minimum	Rekomendasi
CPU	Quad-core, 3 GHz (i5-12400 / Ryzen 5 5600X)	Octa-core, 4+ GHz (i7-13700K / Ryzen 7 7800X)
RAM	16 GB DDR4	32 GB DDR5
GPU	GTX 1660 Super / RX 580	RTX 4070/4080 / RX 7900 XT
Storage	512GB SSD NVMe + 1TB HDD	1TB SSD NVMe
OS	Windows 10/11 64-bit	Windows 11 Pro 64-bit
VR Headset	Meta Quest 2/3, Valve Index, HTC Vive, PICO 4	Meta Quest 3, Valve Index, HTC Vive Pro 2
Display	Full-HD (1920x1080)	QHD/4K, 120Hz+ refresh rate

Tabel 3.5. Framework, library, dan ekstensi VS Code untuk VR Unity

Nama / Versi	Fungsi / Deskripsi Singkat
Unity Engine (2023.2 LTS)	Game engine utama untuk simulasi, eskperimen, dan pengembangan VR modern
XR Interaction Toolkit (3.0.1)	Toolkit resmi Unity untuk interaksi dan navigasi VR/XR

Tabel 3.5 Framework, library, dan ekstensi VS Code untuk VR Unity (lanjutan)

Nama / Versi	Fungsi / Deskripsi Singkat
Input System (1.6.1)	Sistem input modern Unity untuk perangkat kontrol VR/XR
OpenXR Plugin (1.10)	Mendukung build multiplatform VR di Unity
Oculus/Meta Integration (63.0)	SDK VR device Meta Quest generasi terbaru
ProBuilder (5.3.0)	Pembuatan dan prototyping geometry/level design
Cinemachine (2.11.0)	Kontrol kamera dinamis dalam scene VR
Shader Graph (15.0.5)	Visualisasi shader/material, optimal untuk URP/VR
Universal Render Pipeline (15.0.5)	Lightweight rendering pipeline untuk performa VR
TextMeshPro (4.0.0)	Penampilan teks di world-space dan UI VR
VS Code Extension – C# (1.27.0)	Ekstensi IDE C# untuk VS Code dan Unity
VS Code Extension – Unity Tools (1.3.2)	Snippets dan auto-complete untuk Unity development
VS Code Extension – Debugger (3.2.0)	Debugging Unity langsung dalam VS Code
Newtonsoft.Json (13.0.3)	Library serialisasi dan parsing data JSON
DOTween (1.3.0)	Library animasi ringan untuk transisi/efek VR/XR
PolyNav (2.0.6)	Pathfinding dan AI navigation ringan
Unity Asset Bundle Browser (1.8.0)	Tools manajemen paket asset bundle Unity

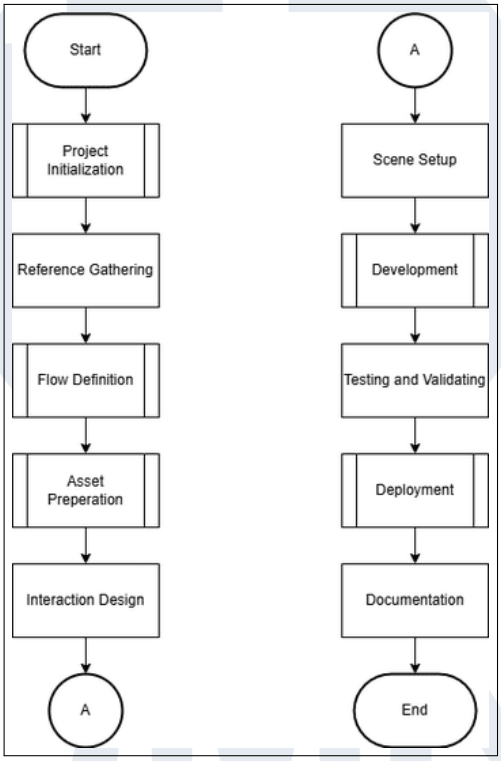
3.4 User Requirements

User requirements atau kebutuhan pengguna merupakan kumpulan spesifikasi yang menjelaskan fungsi, fitur, serta pengalaman penggunaan yang diharapkan dari sistem atau aplikasi yang dikembangkan. Kebutuhan ini berperan sebagai acuan utama dalam memastikan bahwa simulasi yang dibangun relevan dengan kondisi kerja aktual dan mampu memberikan pengalaman pelatihan yang imersif serta interaktif. Setiap elemen dirancang agar mendukung alur kerja teknisi secara realistis dan efisien di dalam lingkungan *Virtual Reality* (VR). Daftar berikut merupakan kebutuhan pengguna yang telah disusun berdasarkan hasil observasi dan diskusi dengan pihak *Saka Farma Laboratories*.

1. Lingkungan virtual dirancang menyerupai area kerja di pabrik *Saka Farma*, mencakup tata letak ruangan, peralatan, serta posisi mesin untuk memberikan kesan autentik.
2. Gerakan rotasi diterapkan pada komponen seperti baut dan mur guna meningkatkan realisme animasi selama proses perakitan maupun pembongkaran mesin.
3. Efek suara ditambahkan pada objek yang jatuh atau terlepas agar umpan balik auditif lebih alami dan membantu pengguna mengenali tindakan yang dilakukan.
4. Termasuk tangga lipat portabel yang dapat digunakan oleh operator ketika harus menjangkau area kerja di ketinggian, sesuai dengan prosedur di lapangan.
5. Menyediakan menu berbasis *scroll wheel* untuk mempermudah pergantian alat, lengkap dengan fitur *inventory* yang menampilkan daftar peralatan yang sedang dibawa.
6. Menyusun mekanisme agar operator dapat tetap berinteraksi dan mengeksplorasi lingkungan meskipun skenario utama telah selesai dijalankan, guna meningkatkan fleksibilitas pelatihan.
7. Menambahkan fitur *scoreboard* yang menampilkan waktu penyelesaian, jumlah kesalahan, dan tingkat keakuratan langkah sebagai sarana evaluasi performa peserta.

3.5 Proses Pelaksanaan Magang

Proses desain proyek dimulai dengan merumuskan alur kerja umum yang akan diikuti dalam pengembangan simulasi VR rakit dan bongkar mesin obat. Pada tahap ini, seluruh rangkaian proses dari inisiasi hingga dokumentasi didefinisikan secara garis besar agar setiap anggota tim dapat memahami urutan dan keterkaitan tiap fase kerja. Diagram alur induk dapat dilihat pada Gambar 3.2.

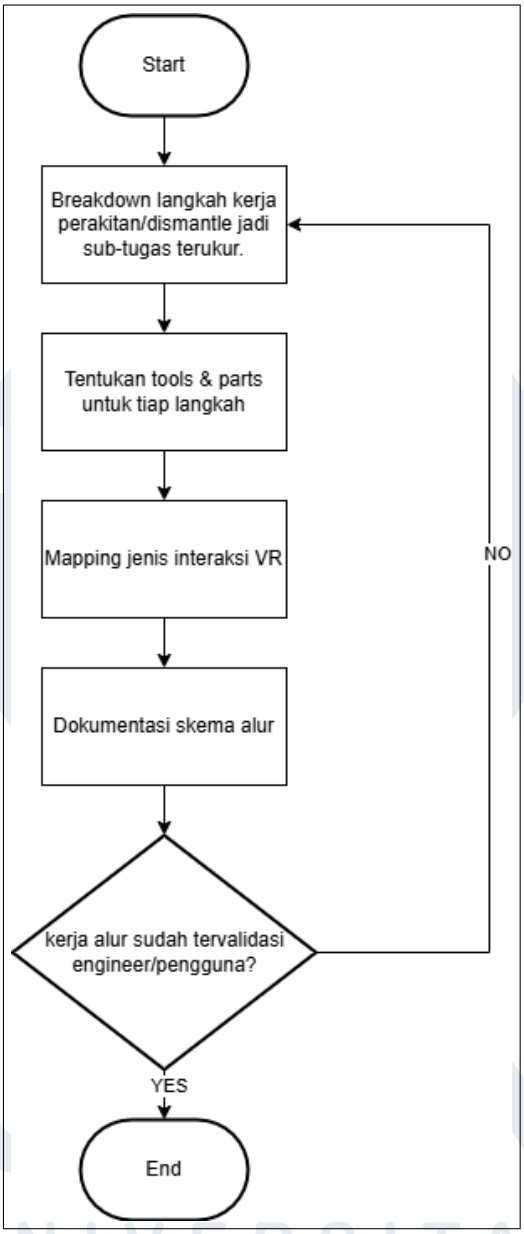


Gambar 3.2. Diagram alur utama pengembangan proyek VR.

Tahapan berikutnya yaitu inisiasi proyek, yang meliputi pengumpulan kebutuhan dari klien serta identifikasi pihak-pihak yang terlibat. Ruang lingkup kerja dirancang sesuai prinsip *SMART* (Specific, Measurable, Achievable, Relevant, Time-Bound) guna memastikan semua target tercapai secara realistis dan terukur. Selain itu, evaluasi sumber daya dan kemampuan tim dilakukan untuk menjamin kesiapan eksekusi proyek secara menyeluruh. Proses ini divisualisasikan pada Gambar 3.3.

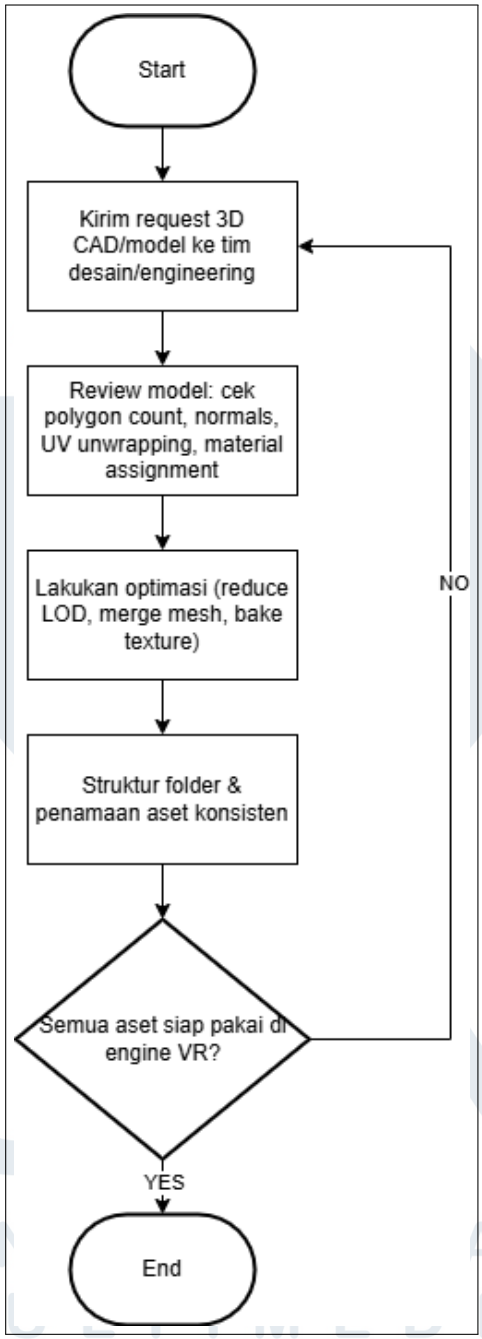
Gambar 3.3. Diagram alur inisiasi proyek.

Setelah cakupan proyek ditetapkan, langkah selanjutnya adalah pemetaan alur kerja perakitan dan pembongkaran mesin. Setiap proses dipecah menjadi tugas-tugas detail agar pelaksanaan di simulasi VR lebih efisien. Interaksi utama seperti mengambil, memasang objek, serta efek visual dan suara juga dirancang. Hasil pemetaan alur ini kemudian dituangkan ke dokumen teknis untuk divalidasi bersama tim teknis dan pengguna. Visualisasinya terdapat pada Gambar 3.4.



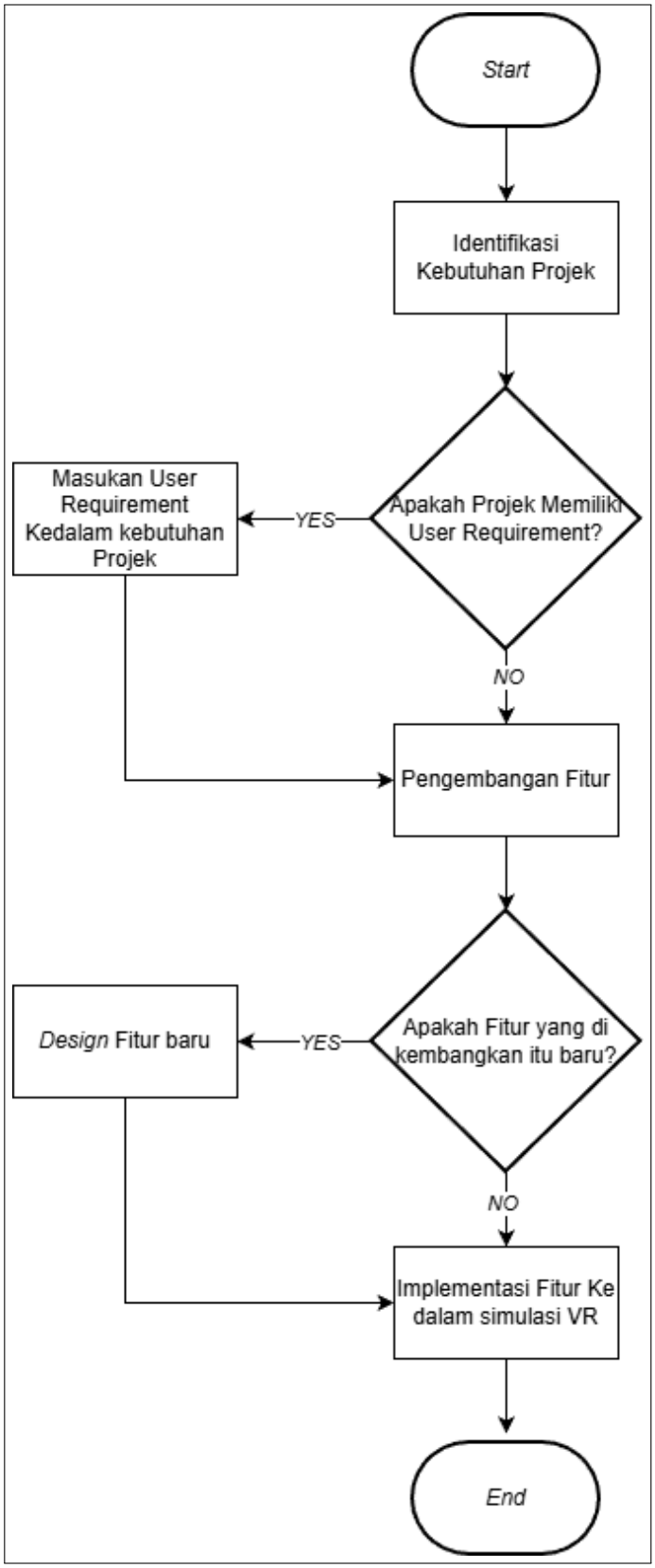
Gambar 3.4. Diagram proses pemetaan alur kerja mesin.

Selanjutnya adalah tahap pengelolaan aset visual berupa model 3D dari mesin. Hasil desain diperiksa agar sesuai dengan kebutuhan teknis simulasi VR, seperti jumlah poligon, skala, dan bahan material. Untuk mengoptimalkan performa sistem, diterapkan pendekatan Level of Detail (LOD), penggabungan objek *mesh*, serta baking tekstur bila dibutuhkan. Rangkaian alurnya tertera pada Gambar 3.5.



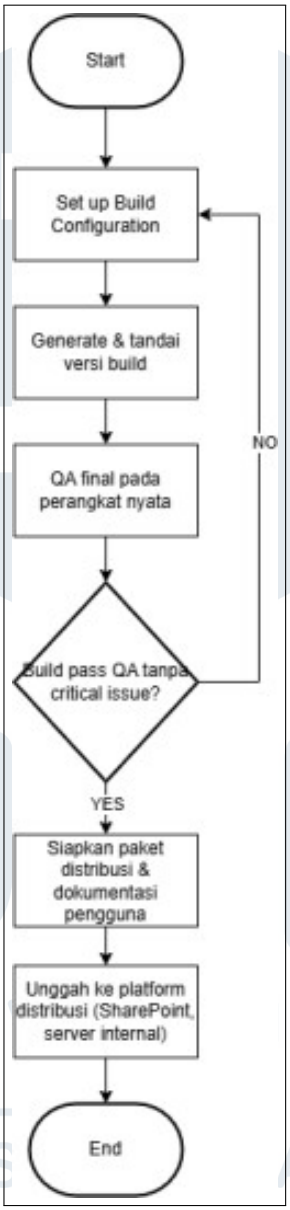
Gambar 3.5. Diagram alur pengelolaan aset visual.

Pada tahap pengembangan di Unity, aset yang telah dioptimasi diimpor lalu dikonfigurasi komponennya seperti collider, prefab, serta penambahan script untuk interaksi. Fitur suara, getaran, dan instruksi juga diintegrasikan agar navigasi dan pengalaman pengguna makin baik. Diagram pengembangan tertampil pada Gambar 3.6.



Gambar 3.6. Diagram proses pengembangan di Unity.

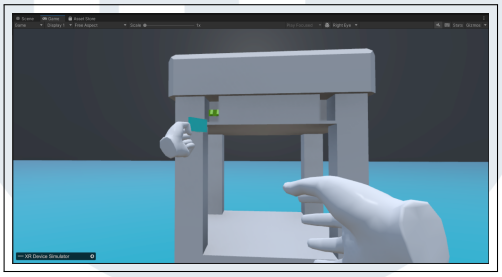
Tahap final adalah deployment, di mana file aplikasi dibuat untuk perangkat user dan diujicoba secara langsung di VR. Setelah performa dan fitur dinyatakan stabil, versi akhir beserta dokumentasi dikemas dan didistribusikan melalui kanal internal. Visualisasi alur deployment dapat dilihat pada Gambar 3.7.



Gambar 3.7. Diagram alur deployment aplikasi.

3.5.1 Fase Pengembangan dan Tujuan

Fase pengembangan ini difokuskan pada penyelesaian skenario *Clean Up* (disassembly) dan *Set Up* (assembly) di seluruh *scene*, penyelarasan perilaku interaksi antarkomponen, serta persiapan aplikasi untuk keperluan demo, UAT internal, dan sesi uji coba bersama tim Saka. Pada tahap rancang bangun, implementasi simulasi pada mesin obat baru mencapai sekitar 50–60%, sementara sejumlah *scene* lainnya masih belum siap, bisa di lihat pada Gambar 3.8. Oleh karena itu, pekerjaan utama pada tahap ini mencakup penyelesaian *scene* yang belum lengkap, penataan ulang arsitektur interaksi, serta peningkatan pengalaman pengguna secara menyeluruh.



Gambar 3.8. Aplikasi pada fase rancang bangun.

Perubahan besar dilakukan pada arsitektur skrip interaksi, dengan migrasi dari pendekatan berbasis objek (*object-based*) menuju pendekatan berbasis *socket*. Pergeseran ini membuat logika orientasi, pemasangan (*attach*), dan pelepasan (*detach*) komponen menjadi lebih konsisten di seluruh *scene*. Selain mengurangi duplikasi kode, pendekatan baru ini juga mempermudah proses *reuse* dan pemeliharaan sistem. Dari sisi pengguna, pola interaksi tetap familiar; perbedaan utama terjadi pada mekanisme di belakang layar yang kini lebih stabil dalam memvalidasi setiap langkah prosedur.

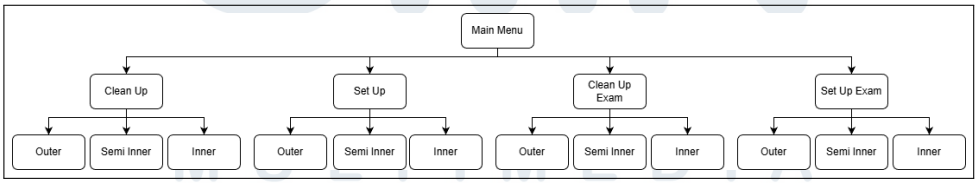
Untuk menjamin kelengkapan setiap tahapan proses, ditambahkan skrip *checker* yang berfungsi memverifikasi kondisi terpasang dan terlepas secara menyeluruh. Mekanisme ini mendukung sistem penilaian lulus atau gagal pada setiap langkah, sekaligus memudahkan pelacakan kesalahan selama sesi pelatihan. Dari aspek antarmuka, menu instruksi diperbarui agar lebih informatif dan terstruktur, dilengkapi dengan indikator progres serta umpan balik visual. Selain itu, diperkenalkan *exam mode* sebagai mode evaluasi dengan panduan minimal (tanpa *hint*) yang dapat dilengkapi dengan pengatur waktu.

3.5.2 Development

Implementasi teknis pada fase ini dilakukan dengan pendekatan *iterative-modular*, yang menitikberatkan pada konsistensi interaksi, stabilitas proses validasi, serta kemudahan pemeliharaan kode. Arsitektur sistem dirancang secara terstruktur: logika pemasangan dan pelepasan komponen dikendalikan oleh skrip berbasis *socket*, validasi progres dijalankan oleh skrip *checker* yang bersifat independen, sementara pengendali menu dikelola secara terpisah dari mekanisme simulasi inti. Pendekatan modular ini mempermudah pengujian setiap bagian secara terisolasi, mengurangi potensi efek samping saat dilakukan pembaruan, dan memungkinkan pemanfaatan ulang komponen di berbagai skenario tanpa perlu duplikasi kode yang berlebihan.

Sistem *checker* dikembangkan sebagai lapisan validasi yang fleksibel, dengan kemampuan memantau berbagai kondisi komponen seperti terpasang (*attached*), terlepas (*detached*), dan tuntas dipasang (*fully hammered*). Validasi ini dijalankan dengan mekanisme berbasis *counter* dan *event listener* yang responsif terhadap setiap perubahan status secara *real-time*.

Gambar 3.9 memperlihatkan alur navigasi aplikasi yang dikembangkan, menampilkan pembagian *scene* berdasarkan kategori seperti *Clean Up*, *Set Up*, dan *Exam Mode*. Setiap kategori memiliki cabang lebih lanjut seperti *Outer*, *Semi Inner*, dan *Inner*, di mana masing-masing mencakup skenario berbeda sesuai proses *clean up*, *set up*, maupun *exam*. Diagram ini menggambarkan bagaimana pengguna dapat berpindah antar *scene*, memilih skenario yang diinginkan, serta mengikuti alur simulasi secara sistematis.



Gambar 3.9. *Sitemap* Alur Navigasi dan Pembagian *Scene* Aplikasi

Untuk menjaga keterbacaan dan mempermudah navigasi laporan, pembahasan implementasi teknis dibagi menjadi lima bagian utama. Bagian pertama, *Set Up Flow*, menjelaskan mekanisme pemasangan komponen beserta logika skrip yang mendukung proses tersebut. Selanjutnya, *Clean Up Flow* membahas tahapan pembongkaran komponen dengan validasi pelepasan yang dibuat konsisten di seluruh skenario. Bagian *UI/UX Instruksi* menguraikan sistem navigasi

berbasis menu hierarki dan kanvas instruksi bertahap yang membantu pengguna memahami alur simulasi. Kemudian, *Exam Mode* memaparkan pendekatan berbasis konfigurasi untuk mode evaluasi tanpa panduan visual, yang dirancang guna menilai pemahaman pengguna secara mandiri. Terakhir, bagian *Hasil Implementasi* menyajikan tangkapan layar, diagram, serta dokumentasi visual dari sistem yang telah dikembangkan.

A. Set Up Flow (Assembly)

Pada proses *set up flow*, skrip `ScSocketAttach` digunakan untuk menstandarisasi pengaturan arah gerak sekrup pada berbagai skenario pemasangan dan pelepasan. Ketika event `SelectEnterEventArgs` dipicu, fungsi `OnObjectAttached` akan memeriksa apakah objek yang dipasang memiliki komponen terkait, seperti `ScScrewAttach`, `ScInsideScrewAttach`, atau `ScScrewDetach`. Jika komponen tersebut ditemukan, skrip akan menerapkan arah gerak yang telah dikonfigurasi pada `socket` ke semua jenis komponen sekrup yang terhubung pada objek tersebut.

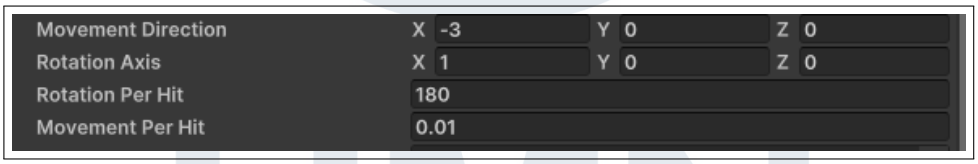
Dengan pendekatan ini, vektor arah (`screwMovementDirection`) hanya perlu dikonfigurasi satu kali pada sisi `socket`, sehingga seluruh objek sekrup yang dipasang, dilepas, maupun yang dioperasikan secara internal akan mendapatkan parameter arah yang sama secara konsisten. Implementasi fungsi ini dapat dilihat pada Kode 3.1, di mana setiap komponen terkait diberikan parameter arah gerak yang seragam.

```
1  private void OnObjectAttached(SelectEnterEventArgs args)
2  {
3      var comp = args.interactableObject as Component;
4      if (comp != null)
5      {
6          var go = comp.gameObject;
7          // ScScrewAttach
8          var attach = go.GetComponentInParent<ScScrewAttach>();
9          if (attach != null)
10         {
11             attach.movementDirection = screwMovementDirection;
12             Debug.Log($"Applied socket dir {screwMovementDirection
       } to {attach.name}");
13         }
14         // ScInsideScrewAttach
```

```
15          var inside = go.GetComponentInParent<ScInsideScrewAttach
    >();
16          if (inside != null)
17          {
18              inside.movementDirection = screwMovementDirection;
19          }
20          // ScScrewDetach
21          var detach = go.GetComponentInParent<ScScrewDetach>();
22          if (detach != null)
23          {
24              detach.movementDirection = screwMovementDirection;
25          }}}
```

Kode 3.1: Pengaturan arah gerak pada berbagai komponen sekrup dalam fungsi OnObjectAttached milik `ScSocketAttach`.

Untuk memperjelas konfigurasi arah gerak, Gambar 3.10 menunjukkan tampilan *Inspector* yang memuat nilai *screwMovementDirection* pada skrip *ScSocketAttach*. Hal ini menjelaskan sumber arah yang digunakan semua skrip sekrup ketika objek melekat pada *socket*.



Gambar 3.10. Pengaturan `screwMovementDirection` pada `ScSocketAttach` di Unity

Selain mengatur arah gerak komponen sekrup, skrip `ScSocketAttach` juga bertanggung jawab untuk menyorot objek yang menjadi target pada langkah perakitan berikutnya. Proses penyorotan ini dijalankan di dalam fungsi `OnObjectAttached` setelah seluruh proses pengaturan arah selesai dilakukan. Dengan demikian, sistem dapat memberikan petunjuk visual yang lebih informatif dan tepat sasaran kepada pengguna.

Logika yang diterapkan memastikan bahwa objek berlabel *Nail* yang sudah sempurna terpasang tidak akan disorot kembali, sehingga tidak membingungkan pengguna atau mengulang petunjuk untuk langkah yang telah lewat. Sementara itu, objek yang belum selesai, khususnya yang memiliki komponen `Renderer`, akan mendapatkan material sorotan sehingga mudah dikenali sebagai target selanjutnya. Bila objek yang akan diproses berupa *placeholder*, objek tersebut akan diaktifkan lebih dulu untuk menandai titik kerja aktual.

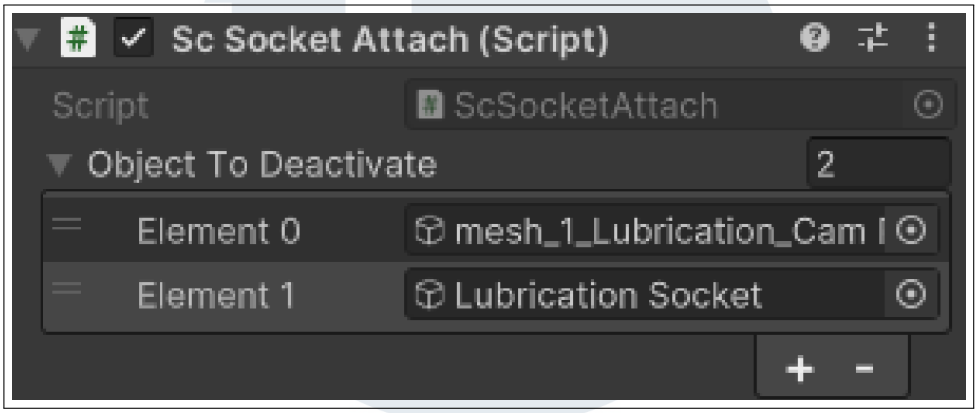
Adapun untuk objek lain, sistem memeriksa apakah terdapat komponen `Rigidbody` pada objek atau parent-nya sebelum melakukan penyorotan. Hal ini memastikan bahwa hanya objek yang relevan secara fisika dan visual yang mendapatkan efek sorotan, sehingga panduan bagi pengguna tetap fokus dan tidak tumpang-tindih. Implementasi detail proses tersebut dapat dilihat pada Kode 3.2, yang merupakan bagian integral dari fungsi `OnObjectAttached` pada skrip `ScSocketAttach`.

```
foreach (GameObject obj in nextHighlight)
{
    if (obj != null)
    {
        if (obj.CompareTag("Nail"))
        {
            if (IsNailFullyScrewed(obj))
            {
                Debug.Log($"Skipping already screwed nail: {obj.
    name}");
                continue;
            } else {
                Renderer renderer = obj.GetComponent<Renderer>();
                if (renderer != null)
                {
                    renderer.material = highlightMaterial;
                }
            }
        }
        if (obj.name.Contains("Placeholder"))
        {
            obj.SetActive(true);
            Debug.Log($"Enabled placeholder object: {obj.name}");
        }
        else
        {
            if (obj == null) continue;
            var rb = obj.GetComponent<Rigidbody>();
            if (rb == null){
                rb = obj.GetComponentInParent<Rigidbody>();
            }
            if (rb != null)
            {
                Renderer renderer = obj.GetComponent<Renderer>();
                if (renderer != null)
```

```
                {
                    renderer.material = highlightMaterial;
                }}}}}
```

Kode 3.2: Logika penyorotan objek target pada langkah berikutnya (bagian dari fungsi `OnObjectAttached` pada `ScSocketAttach`).

Konfigurasi *objectToActivate* berperan dalam mengaktifkan objek selanjutnya setelah langkah kini selesai. Gambar 3.11 menampilkan daftar objek yang akan diaktifkan, seperti objek-objek lanjutan atau *socket* berikutnya.



Gambar 3.11. Contoh konfigurasi `objectToActivate` pada `ScSocketAttach`

Selain menyorot langkah berikutnya, skrip juga menghapus sorotan pada objek yang sudah tidak diperlukan, agar panduan visual tetap jelas dan fokus. Proses ini dilakukan dengan mengembalikan material objek ke *defaultMaterial*, seperti ditunjukkan pada Kode 3.3.

```
foreach (GameObject obj in removeHighlight)
{
    if (obj != null)
    {
        Renderer renderer = obj.GetComponent<Renderer>();
        if (renderer != null)
        {
            renderer.material = defaultMaterial;
        }
    }
}
```

Kode 3.3: Mengembalikan material objek ke *defaultMaterial* untuk membersihkan *highlight*.

Pada proses pemasangan objek, skrip `ScSocketAttach` menonaktifkan objek tertentu setelah jeda singkat. Penundaan selama beberapa milidetik diberikan agar proses *snap* objek ke socket selesai terlebih dahulu dan untuk mencegah konflik fisika atau kondisi yang tidak stabil. Implementasi logika ini terdapat pada fungsi `Delayed`, seperti pada Kode 3.4, yang melakukan penonaktifan semua objek di dalam daftar `objectToDeactivate` setelah penundaan.

```
private IEnumerator Delayed()
{
    yield return new WaitForSeconds(0.2f); // short delay allows
    snapping to complete

    // Deactivate all objects in the objectToDeactivate array when
     an object attaches
    foreach (GameObject obj in objectToDeactivate)
    {
        if (obj != null)
        {
            obj.SetActive(false); // Deactivate the object
            Debug.Log($"Deactivated object: {obj.name}");
        }
        else
        {
            Debug.LogWarning("ScSocketAttach: Found null object in
     objectToDeactivate array!");
        }
    }
}
```

Kode 3.4: Penonaktifan objek setelah delay agar *snap* selesai.

Agar langkah yang sudah selesai tidak kembali disorot, skrip menyediakan fungsi pengecekan status objek berlabel *Nail*. Fungsi ini akan memeriksa apakah objek tersebut (beserta parent-nya) memiliki salah satu skrip terkait sekrup, seperti `ScScrewAttach`, `ScInsideScrewAttach`, atau `ScScrewDetach`, lalu mengembalikan status pemasangan dari properti `IsFullyHammered`. Implementasinya dapat dilihat pada Kode 3.5. Fungsi ini menjaga agar panduan visual tetap relevan dan tidak mengulang pada komponen yang sudah dinyatakan selesai proses pemasangannya.

```
private bool IsNailFullyScrewed(GameObject nailObject)
{
    // Check if the nail object has any of the screw-related
    scripts
    ScScrewAttach screwAttach = nailObject.GetComponent<
    ScScrewAttach>();
    ScInsideScrewAttach insideScrewAttach = nailObject.
    GetComponent<ScInsideScrewAttach>();
    ScScrewDetach screwDetach = nailObject.GetComponent<
    ScScrewDetach>();

    // If no script found on the object, check its parent
    if (screwAttach == null)
        screwAttach = nailObject.transform.parent?.GetComponent<
    ScScrewAttach>();

    if (insideScrewAttach == null)
        insideScrewAttach = nailObject.transform.parent?.
    GetComponent<ScInsideScrewAttach>();

    if (screwDetach == null)
        screwDetach = nailObject.transform.parent?.GetComponent<
    ScScrewDetach>();

    // Check if any of the screw scripts has the screw fully
    hammered
    if ((screwAttach != null && screwAttach.IsFullyHammered) ||
        (insideScrewAttach != null && insideScrewAttach.
    IsFullyHammered) ||
        (screwDetach != null && screwDetach.IsFullyHammered))
    {
        return true; // Nail is fully screwed
    }

    return false; // Nail is not fully screwed
}
```

Kode 3.5: Fungsi pengecekan status pemasangan pada objek bertipe *Nail*.

Pada bagian `Start`, skrip `ScAttachCheckerAdvance` melakukan inisialisasi pemantauan objek target yang akan dicek status pemasangannya. Skrip menampilkan jumlah objek yang dipantau beserta nilai ambang batas (`objectsToCheckCount`), lalu melakukan setup awal untuk setiap objek jika

diperlukan. Referensi ke `menuController` juga diambil dari scene, dan jika mode debug aktif, informasi detail log dicetak.

Selanjutnya, fungsi `CheckObjectAttachments` akan dipanggil secara berkala menggunakan `InvokeRepeating`, sehingga status pemasangan objek bisa dipantau secara otomatis tanpa membebani performa dengan pemanggilan di setiap frame. Inisialisasi ini dijabarkan dalam Kode 3.6.

```
private void Start()
{
    Debug.Log(
        $"ScAttachCheckerAdvance: Starting with {objectsToCheck?.Length ?? 0} objects to check, threshold={objectsToCheckCount}"
    );

    // Register objects for checking
    if (objectsToCheck != null)
    {
        foreach (var obj in objectsToCheck)
        {
            if (obj == null) continue;

            // Add listeners or setup logic if needed
            // Here we check for attachment directly
        }
    }

    menuController = FindObjectOfType<ScMenuNew>();

    if (debugLogs)
    {
        Debug.Log($"[ScAttachCheckerAdvance] Total objects to check: {objectsToCheck?.Length ?? 0}. Threshold={objectsToCheckCount}");
    }

    InvokeRepeating(nameof(CheckObjectAttachments), 1f, 0.5f);
}
```

Kode 3.6: Inisialisasi dan *loop* pengecekan pada *Start* di `ScAttachCheckerAdvance`.

Fungsi `CheckObjectAttachments` bertugas menghitung berapa banyak objek dalam `objectsToCheck` yang sudah terpasang, sekaligus mencatat objek mana yang belum selesai. Jika jumlah objek yang terpasang sudah memenuhi

syarat, maka langkah dianggap lulus—sistem akan menampilkan pesan selesai, reset material objek yang belum terpasang, aktivasi dan nonaktifkan objek sesuai konfigurasi, lalu highlight target berikutnya. Setelah kondisi tercapai, `CancelInvoke` digunakan untuk menghentikan loop pemeriksaan agar performa tetap efisien. Implementasi fungsi ini dapat dilihat pada Kode 3.7.

```
private void CheckObjectAttachments()
{
    int attachedFlagCount = 0;
    var notAttachedNames = new List<string>();

    // Check the attachment status of each object
    foreach (var obj in objectsToCheck)
    {
        if (obj == null) continue;

        if (IsObjectAttached(obj)) attachedFlagCount++;
        else notAttachedNames.Add(obj.name);
    }

    if (debugLogs)
        Debug.Log($"[ScAttachCheckerAdvance] AttachedFlagCount={
    attachedFlagCount} / Target={objectsToCheckCount} | NotAttached
    : [{string.Join(", ", notAttachedNames)}]");

    // When the required number of objects are attached
    if (!attachPassed && attachedFlagCount >= objectsToCheckCount)
    {
        attachPassed = true;

        if (endCanvas != null && doneCanvas != null)
        {
            // Stop the timer when the condition is met
            timerActive = false;

            // Deactivate the endCanvas and activate the
    doneCanvas
            if (endCanvas != null) endCanvas.SetActive(false);
            if (doneCanvas != null) doneCanvas.SetActive(true);

            int minutes = Mathf.FloorToInt(timer / 60);
            int seconds = Mathf.FloorToInt(timer % 60);

```

```
35            TextMeshProUGUI completionTimeText = doneCanvas.
   GetComponentInChildren<TextMeshProUGUI>();
36            if (completionTimeText != null)
37            {
38                completionTimeText.text = $"Exam Completed !!\
   nTime Taken: {minutes} min {seconds} sec";
39            }
40        }
41
42        // Reset NON-attached objects to default material
43        foreach (var obj in objectsToCheck)
44            if (obj != null && !IsObjectAttached(obj))
45                SetMaterialAllRenderers(obj, defaultMaterial);
46
47        // Activate next objects
48        foreach (var obj in objectsToActivate)
49            if (obj != null) obj.SetActive(true);
50
51        foreach (var obj in objectsToRemove)
52            if (obj != null) obj.SetActive(false);
53        // Reset material for current objects
54        foreach (var obj in objectsCurrent)
55            SetMaterialAllRenderers(obj, defaultMaterial);
56        // Highlight next objects
57        foreach (var obj in objectsNext)
58        {
59            if (obj == null) continue;
60            var rb = obj.GetComponent<Rigidbody>() ?? obj.
   transform.parent?.GetComponent<Rigidbody>();
61            if (rb != null)
62                SetMaterialAllRenderers(obj, highlightMaterial);
63            else if (debugLogs)
64                Debug.Log($"ScAttachCheckerAdvance: Skipped {obj.
   name} (no Rigidbody found)");
65        }
66    }
67    if (attachPassed)
68        CancelInvoke(nameof(CheckObjectAttachments));}
```

Kode 3.7: Pemeriksaan dan transisi status *attachment* objek pada `ScAttachCheckerAdvance`.

Fungsi `IsObjectAttached` digunakan untuk menilai status pemasangan setiap objek target. Fungsi ini akan memeriksa apakah objek memiliki komponen `ScYDividerAssembly` atau `ScInsideTurretAssembly` yang keduanya memiliki properti `IsAttached`. Jika salah satu dari komponen tersebut dan statusnya aktif, maka objek dinyatakan telah terpasang. Jika tidak, hasil pemeriksaan adalah false. Implementasi detailnya terlihat pada Kode 3.8.

```
private bool IsObjectAttached(GameObject go)
{
    if (go == null) return false;

    // Check if the object has the ScYDividerAssembly component
    var yDividerAssembly = go.GetComponent<ScYDividerAssembly>();
    if (yDividerAssembly != null && yDividerAssembly.IsAttached)
    {
        return true;
    }

    // Check if the object has the ScInsideTurretAssembly component
    var insideTurretAssembly = go.GetComponent<ScInsideTurretAssembly>();
    if (insideTurretAssembly != null && insideTurretAssembly.IsAttached)
    {
        return true;
    }
    // If neither script has IsAttached set to true, return false
    return false;
}
```

Kode 3.8: Fungsi pengecekan objek sudah terpasang melalui komponen.

Skrip `ScScrewCheckerAdvance` digunakan untuk memantau langkah perakitan yang melibatkan sekrup. Berbeda dengan `ScAttachCheckerAdvance` yang hanya menghitung jumlah objek terpasang, skrip ini membedakan dua tahap validasi: jumlah sekrup yang sudah dipasang (*attached*) dan jumlah sekrup yang benar-benar selesai (*fully hammered*). Langkah hanya dinyatakan lulus bila kedua kondisi terpenuhi, sehingga urutan kerja dan proses mekanis benar-benar tuntas.

Pada fungsi `Start`, masing-masing sekrup target didaftarkan ke dalam daftar pemantauan dan listener `OnHammered` dipasang untuk setiap skrip sekrup, agar sistem bisa langsung merespons event sekrup yang dituntaskan. Selain event

listener, pemeriksaan periodik tetap dijalankan dengan `InvokeRepeating` untuk menjaga konsistensi status. Implementasi inisialisasi dapat dilihat pada Kode 3.9.

```
private void Start()
{
    Debug.Log(
        $"ScScrewCheckerAdvance: Starting with {screws?.Length ?? 0} screws, {insideScrews?.Length ?? 0} inside screws, {detachScrews?.Length ?? 0} detach screws, insideDisasm:{insideDisassemblyScrews?.Length ?? 0}, threshold={screwsToCheck}"
    );

    // Register attach screws
    if (screws != null)
    {
        foreach (var sc in screws)
        {
            if (sc == null) continue;
            screwsToMonitor.Add(sc);
            if (!sc.assignedCheckers.Contains(this))
                sc.assignedCheckers.Add(this);
            sc.OnHammered += CheckSelectedScrews;
        }
    }

    // Register inside screws
    if (insideScrews != null)
    {
        foreach (var sc in insideScrews)
        {
            if (sc == null) continue;
            insideScrewsToMonitor.Add(sc);
            if (!sc.assignedCheckers.Contains(this))
                sc.assignedCheckers.Add(this);
            sc.OnHammered += CheckSelectedScrews;
        }
    }

    // Register detach screws
    if (detachScrews != null)
    {
        foreach (var sc in detachScrews)
        {
```

```
            if (sc == null) continue;
            detachScrewsToMonitor.Add(sc);
            if (!sc.assignedCheckers.Contains(this))
                sc.assignedCheckers.Add(this);
            sc.OnHammered += CheckSelectedScrews;
        }
    }

    // Register inside disassembly screws
    if (insideDisassemblyScrews != null)
    {
        foreach (var sc in insideDisassemblyScrews)
        {
            if (sc == null) continue;
            insideDisassemblyScrewsToMonitor.Add(sc);
            if (!sc.assignedCheckers.Contains(this))
                sc.assignedCheckers.Add(this);
            sc.OnHammered += CheckSelectedScrews;
        }
    }

    menuController = FindObjectOfType<ScMenuNew>();

    if (debugLogs)
    {
        Debug.Log($"[ScScrewCheckerAdvance] Totals -> attach:{
    screwsToMonitor.Count}, inside:{insideScrewsToMonitor.Count},
    detach:{detachScrewsToMonitor.Count}, insideDisasm:{
    insideDisassemblyScrewsToMonitor.Count}. Threshold={
    screwsToCheck}");
    }

    Invoke(nameof(CheckSelectedScrews), 0.1f);
    InvokeRepeating(nameof(CheckSelectedScrews), 1f, 0.5f);
}
```

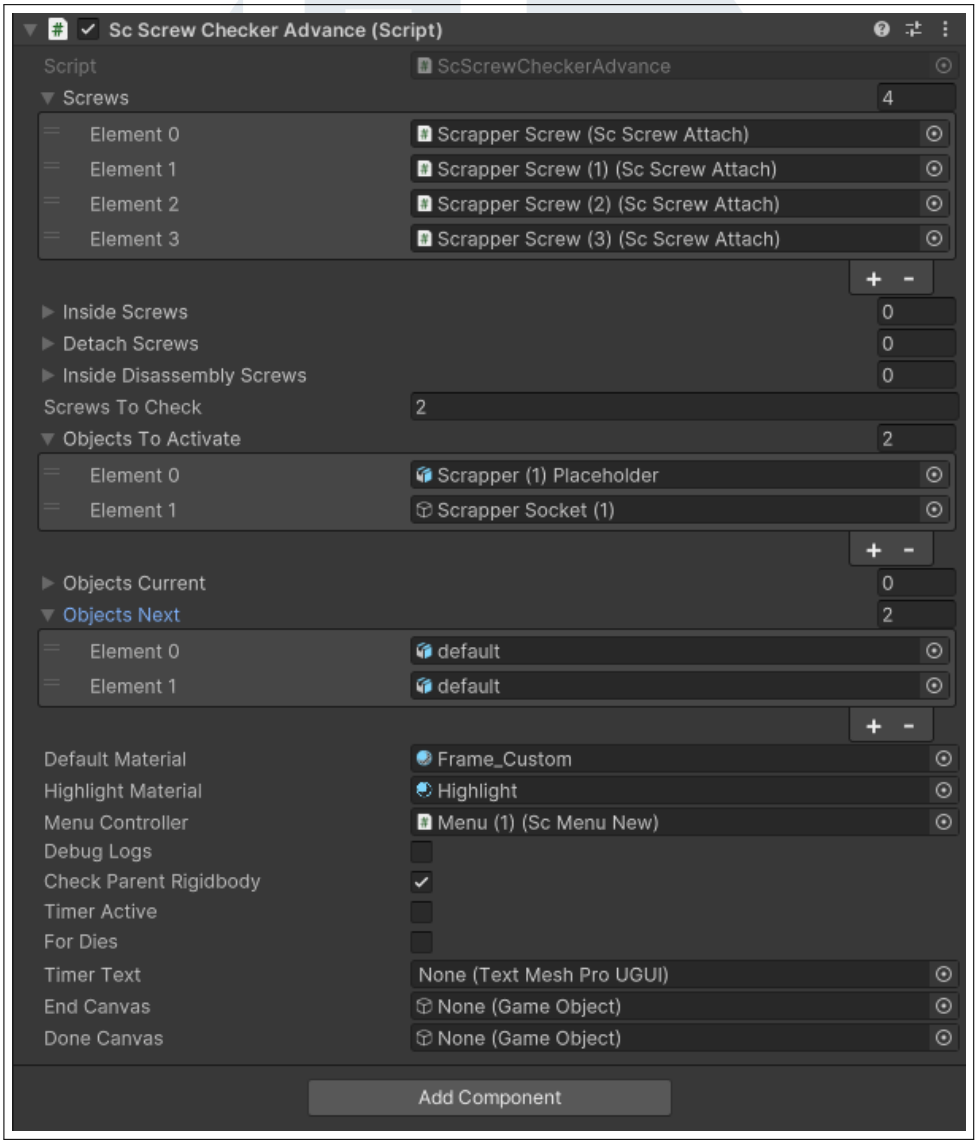
Kode 3.9: Inisialisasi pemantauan sekrup pada `ScScrewCheckerAdvance`.

Proses pemeriksaan kemajuan langkah pada `ScScrewCheckerAdvance` dijalankan melalui fungsi `CheckSelectedScrews`. Pada setiap eksekusi, skrip menghitung jumlah sekrup yang telah terpasang (*attached*) dan jumlah sekrup yang benar-benar sudah selesai dipasang (*fully hammered*). Kedua parameter ini dibandingkan dengan ambang yang ditentukan, dan langkah baru dinyatakan lulus jika seluruh syarat terpenuhi. Setelah validasi sukses, sistem akan mengaktifkan proses berikutnya dan menghentikan loop pemeriksaan untuk menjaga efisiensi aplikasi. Implementasi logika pemeriksaan ini dapat dilihat pada Kode 3.10.

```
private void CheckSelectedScrews()
{
    int attachedFlagCount = 0, fullyHammeredCount = 0;

    // Cek jumlah sekrup yang terpasang dan yang sudah full hammered
    foreach (var s in screwsToMonitor)
    {
        if (s == null) continue;
        if (GetAttachedFlagFromGO(s.gameObject)) attachedFlagCount++;
        if (s.IsFullyHammered) fullyHammeredCount++;
    }

    // Langkah dinyatakan lulus jika jumlah syarat terpenuhi
    if (!attachedPassed && attachedFlagCount >= screwsToCheck)
        attachedPassed = true;

    if (!hammeredPassed && fullyHammeredCount >= screwsToCheck)
    {
        hammeredPassed = true;
        // Aktivasi langkah berikutnya: tampilkan pesan selesai, aktifkan objek, highlight, dsb.
        // ... (detail implementasi disesuaikan dengan kebutuhan UI dan logika transisi)
    }

    // Hentikan pemeriksaan jika semua syarat sudah terpenuhi
    if (hammeredPassed && attachedPassed)
        CancelInvoke(nameof(CheckSelectedScrews));
}
```

Kode 3.10: Logika ringkas pemeriksaan kemajuan sekrup pada `ScScrewCheckerAdvance`.

Gambar 3.12 mendokumentasikan susunan objek pada tahap assembly dengan penekanan visual yang jelas. Pada ilustrasi tersebut, objek-objek dalam daftar *objectsToActivate* merupakan elemen yang akan diaktifkan ketika proses validasi berlangsung. Sementara itu, objek-objek pada *objectsCurrent* mengalami penghapusan efek highlight sebagai tanda pembersihan. Objek-objek di *objectsNext* kemudian diberikan efek sorotan guna menandai target yang akan diproses selanjutnya.



Gambar 3.12. Ilustrasi pengaturan `objectsToActivate`, `objectsCurrent`, dan *objectsNext* pada proses validasi langkah VR assembly.

Fungsi `IncrementFullyHammeredCount` pada `ScScrewCheckerAdvance` berfungsi untuk memantau setiap perubahan status sekrup yang telah selesai dipalu sepenuhnya (*fully hammered*). Saat fungsi ini dipanggil, sistem terlebih dahulu memverifikasi apakah sekrup yang diterima termasuk dalam daftar pemantauan (`screwsToMonitor`). Jika valid, penghitung `fullyHammeredCount` akan bertambah satu dan sistem kemudian memanggil `CheckSelectedScrews()` untuk memeriksa apakah kondisi penyelesaian langkah telah terpenuhi. Proses ini juga dilengkapi dengan log peringatan untuk mendeteksi apabila terjadi percobaan peningkatan jumlah pada objek yang tidak terdaftar. Implementasi lengkap dari fungsi ini dapat dilihat pada Kode 3.11.

```
public void IncrementFullyHammeredCount(ScScrewAttach screw)
{
    // Hanya menambah jumlah jika sekrup termasuk dalam daftar pemantauan
    if (screwsToMonitor.Contains(screw))
    {
        fullyHammeredCount++;
        if (debugLogs) Debug.Log($"Screws fully hammered: {fullyHammeredCount}/{screwsToCheck}");
        CheckSelectedScrews();
    }
    else
    {
        Debug.LogWarning("Tried to increment count for a screw not monitored by this checker.");
    }
}
```

Kode 3.11: Fungsi `IncrementFullyHammeredCount` untuk menghitung jumlah sekrup yang sepenuhnya terpasang.

Fungsi `GetAttachedFlagFromGO` pada kelas `ScScrewCheckerAdvance` digunakan untuk memeriksa status keterpasangan (*attached*) dari suatu objek sekrup berdasarkan referensi `GameObject` yang dikirimkan. Fungsi ini diawali dengan pemeriksaan awal untuk memastikan bahwa objek tidak bernilai `null`. Selanjutnya, sistem mencari komponen terkait yang mewakili berbagai tipe interaksi sekrup seperti `ScScrewAttach`, `ScInsideScrewAttach`, `ScScrewDetach`, maupun `ScInsideScrewDisassembly`. Jika salah satu komponen ditemukan, nilai `AttachedFlag` dari komponen tersebut akan dikembalikan sebagai hasil pemeriksaan. Pendekatan ini memungkinkan fungsi

untuk menangani berbagai jenis sekrup dengan fleksibilitas tinggi tanpa perlu pemisahan logika tambahan. Implementasi fungsi ini ditunjukkan pada Kode 3.12.

```
private bool GetAttachedFlagFromGO(GameObject go)
{
    if (go == null) return false;

    // 1) Direct known types
    var a = go.GetComponent<ScScrewAttach>();
    if (a != null) return a.AttachedFlag;

    var i = go.GetComponent<ScInsideScrewAttach>();
    if (i != null) return i.AttachedFlag;

    var d = go.GetComponent<ScScrewDetach>();
    if (d != null) return d.AttachedFlag;

    // NEW:
    var id = go.GetComponent<ScInsideScrewDisassembly>(); // NEW
    if (id != null) return id.AttachedFlag;

    return false;
}
```

Kode 3.12: Fungsi `GetAttachedFlagFromGO` untuk memeriksa status attached dari objek sekrup berdasarkan jenis komponennya.

B. Clean Up Flow (Disassembly)

Skrip `ScAttachCheckerAdvance` memegang peran sentral dalam proses validasi baik pada tahap Set Up maupun Clean Up, seperti yang telah dijelaskan pada Kode 3.9. Pada tahap inisialisasi, sebagaimana yang terlihat pada Kode 3.13, sistem menerima daftar objek yang akan diperiksa, lalu menyiapkan rutinitas pemeriksaan secara berkala melalui fungsi `InvokeRepeating` untuk memantau status keterpasangan setiap objek (*attachments*). Berbeda dengan checker pada tahap pemasangan, validasi pada Clean Up menggunakan pendekatan serupa untuk memastikan objek yang perlu dilepas telah benar-benar mencapai status "terlepas". Jika jumlah objek yang memenuhi syarat mencapai ambang `objectsToCheckCount`, langkah lanjutan akan diaktifkan: sistem membersihkan highlight objek yang telah diperiksa dan mempersiapkan objek berikutnya (`objectsNext`) agar siap untuk interaksi selanjutnya.

```
private void Start()
{
    Debug.Log(
        $"ScAttachCheckerAdvance: Starting with {objectsToCheck?.
    Length ?? 0} objects to check, threshold={objectsToCheckCount}"
    );

    // Register objects for checking
    if (objectsToCheck != null)
    {
        foreach (var obj in objectsToCheck)
        {
            if (obj == null) continue;

            // Add listeners or setup logic if needed
            // Here we check for attachment directly
        }
    }

    menuController = FindObjectOfType<ScMenuNew>();

    if (debugLogs)
    {
        Debug.Log($"[ScAttachCheckerAdvance] Total objects to
    check: {objectsToCheck?.Length ?? 0}. Threshold={
    objectsToCheckCount}");
    }

    InvokeRepeating(nameof(CheckObjectAttachments), 1f, 0.5f);
}
```

Kode 3.13: Inisialisasi pemeriksaan objek pada `ScAttachCheckerAdvance` dengan rutinitas pemantauan status keterpasangan.

Logika utama pemeriksaan status pemasangan objek pada skrip `ScAttachCheckerAdvance` dilakukan secara berkala melalui fungsi `CheckObjectAttachments` Kode 3.9, yang dipanggil rutin oleh sistem setelah inisialisasi lihat Kode 3.13. Fungsi ini menghitung jumlah objek yang telah memenuhi status terpasang (melalui `IsObjectAttached`), lalu membandingkannya dengan ambang minimum `objectsToCheckCount`.

Jika jumlah objek terpasang sudah sesuai syarat, status `attachPassed` diaktifkan. Selanjutnya, sistem menghentikan pengecekan, memproses transisi

aplikasi, seperti menampilkan waktu penyelesaian, mengatur ulang material objek yang belum terpasang, mengaktifkan objek berikut (`objectsToActivate`), serta menyorot dan menyiapkan interaksi untuk langkah selanjutnya. Dengan demikian, fungsi ini merupakan penghubung antara keberhasilan validasi pemasangan dan proses penyiapan tahap berikutnya.

```
private void CheckObjectAttachments()
{
    int attachedFlagCount = 0;

    // Periksa semua objek, hitung yang terpasang
    foreach (var obj in objectsToCheck)
        if (obj != null && IsObjectAttached(obj))
    attachedFlagCount++;

    // Jika seluruh syarat tercapai
    if (!attachPassed && attachedFlagCount >= objectsToCheckCount)
    {
        attachPassed = true;
        // Proses transisi: tampilkan waktu, reset material,
    aktifkan objek berikut dst.
        // (Detail implementasi disesuaikan kebutuhan aplikasi)
    }

    // Hentikan pemeriksaan jika selesai
    if (attachPassed)
        CancelInvoke(nameof(CheckObjectAttachments));
}
```

Kode 3.14: Rutinitas pemeriksaan status yang mengatur transisi antar langkah pada `ScAttachCheckerAdvance`.

Fungsi `IsObjectAttached` dalam skrip `ScAttachCheckerAdvance` bertugas menentukan status keterpasangan suatu objek berdasarkan komponen yang terpasang pada `GameObject` terkait, dan merupakan pendekatan modular lanjutan dari pemeriksaan status seperti pada Kode 3.9. Pada implementasinya Kode 3.15, fungsi ini akan mengembalikan nilai benar apabila objek memiliki komponen `ScScrewDetach` atau `ScSemiDisassembly` dengan indikator `IsAttached` aktif. Jika tidak, maka objek akan dinyatakan belum terpasang. Dengan cara ini, sistem dapat secara fleksibel menangani berbagai tipe interaksi assembly dan disassembly yang mungkin digunakan dalam satu alur validasi.

```
private bool IsObjectAttached(GameObject go)
{
    if (go == null) return false;

    var screwDetach = go.GetComponent<ScScrewDetach>();
    if (screwDetach != null && screwDetach.IsAttached)
    {
        return true;
    }

    var semiDisassembly = go.GetComponent<ScSemiDisassembly>();
    if (semiDisassembly != null && semiDisassembly.IsAttached)
    {
        return true;
    }

    return false;
}
```

Kode 3.15: Fungsi `IsObjectAttached` untuk memeriksa status keterpasangan objek berdasarkan komponen yang relevan.

Pada Kode 3.16, fungsi `Start` digunakan untuk melakukan inisialisasi berbagai komponen serta pengaturan event pada sistem VR assembly yang melibatkan objek punch. Fungsi ini memastikan bahwa `menuController` terhubung dengan objek yang relevan di scene, kemudian mendaftarkan listener untuk event `selectEntered` dan `selectExited` pada setiap punch agar dapat menangani aksi pemasangan dan pelepasan secara interaktif. Selain itu, informasi parent dari target juga dicatat jika diperlukan untuk logika penempatan. Fungsi tersebut juga menangani pencarian dan penyimpanan collider bertag tertentu yang akan diabaikan pada saat proses attach, guna menjaga performa dan logika interaksi. Terakhir, `AudioSource` pada objek diinisialisasi agar sistem bisa memutar suara ketika aksi dijalankan.

```
void Start()
{
    if (menuController == null) menuController = FindObjectOfType<
    ScMenuNew>();

    // Subscribe to select events for each punch
    if (punches != null)
    {
        foreach (var p in punches)
```

```
        {
            if (p == null) continue;
            p.selectEntered.AddListener(OnPunchSelectEntered);
            p.selectExited.AddListener(OnPunchSelectExited);
        }
    }

    if (parentOnAttach != null)
    {
        originalParentOfTarget = parentOnAttach.parent;
    }

    // Cache tagged target colliders once if desired
    if (ignoreCollisionsOnAttach && !searchIgnoreTargetsEveryTime)
    {
        cachedIgnoreTargets = FindCollidersByTag(
   ignoreCollisionsWithTag);
    }

    source = GetComponent<AudioSource>();
}
```

Kode 3.16: Inisialisasi *event* dan komponen penting pada sistem *punch assembly* di fungsi *Start*.

Fungsi `OnPunchSelectEntered` Kode 3.17 berfungsi untuk menangani logika setiap kali objek punch berhasil dimasukkan ke socket yang sesuai dalam proses VR assembly. Pertama, fungsi ini memeriksa apakah interaksi dilakukan oleh `XRSocketInteractor` serta memastikan socket tersebut memiliki tag yang dibutuhkan (`requiredSocketTag`). Jika kondisi ini terpenuhi, objek punch diidentifikasi dan dicek apakah belum pernah dicatat sebagai punch yang telah ditempatkan. Jika masih baru, sistem memutar efek suara konfirmasi, menambahkan punch ke dalam daftar `placedPunches`, serta mengunci seluruh pergerakan dengan membekukan `Rigidbody` dan menonaktifkan `Collider` untuk menghindari interaksi fisik selanjutnya. Proses selanjutnya melibatkan coroutine `HandlePunchPlacement` untuk penanganan transisi, serta pengecekan kondisi penyelesaian langkah menggunakan `TryFinishIfComplete`. Pendekatan ini menjamin respons interaktif dan transisi logis setiap kali punch berhasil ditempatkan pada socket yang benar.

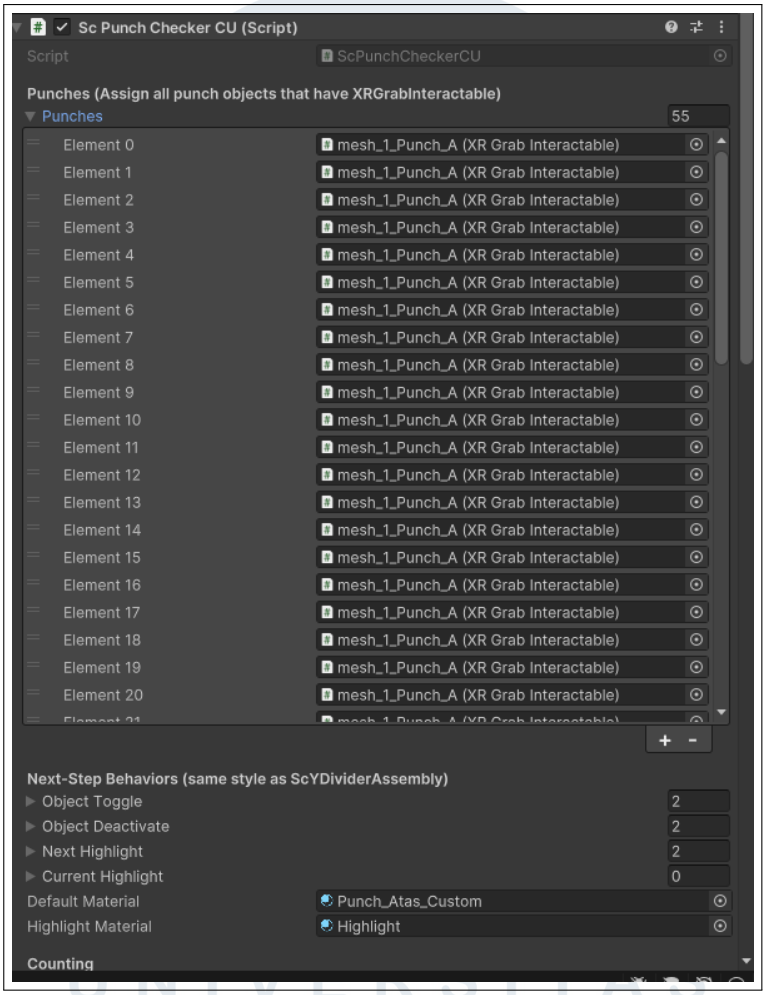
```
private void OnPunchSelectEntered(SelectEnterEventArgs args)
{
    // Hanya memproses jika interaktor berupa socket dengan tag sesuai
    if (args.interactorObject is XRSocketInteractor xrSocket &&
        xrSocket != null &&
        xrSocket.CompareTag(requiredSocketTag))
    {
        GameObject punchGO = args.interactableObject.transform.gameObject;

        // Hanya dihitung sekali per punch
        if (!placedPunches.Contains(punchGO))
        {
            source.PlayOneShot(snap);
            placedPunches.Add(punchGO);

            var rb = punchGO.GetComponent<Rigidbody>();
            if (rb) rb.constraints = RigidbodyConstraints.FreezeAll;

            var col = punchGO.GetComponent<Collider>();
            if (col) col.enabled = false;

            StartCoroutine(HandlePunchPlacement(punchGO, xrSocket));
            TryFinishIfComplete();
        }
    }
}
```

Kode 3.17: Logika penanganan interaksi saat punch masuk ke *socket* melalui *OnPunchSelectEntered*.

Untuk mendukung proses validasi dan pemeriksaan logika assembly pada simulator VR, seluruh konfigurasi penting ditata langsung melalui Inspector Unity, sebagaimana diperlihatkan pada Gambar 3.13. Pada tampilan tersebut, daftar objek punch yang dipantau (`Punches`) diisi dengan komponen `XRGrabInteractable` sehingga setiap objek dapat diinteraksikan dalam lingkungan VR. Seluruh elemen punch terdaftar secara sistematis untuk memastikan proses counting dan validasi berjalan konsisten.

Selain itu, bagian konfigurasi berikutnya menampilkan parameter behaviour

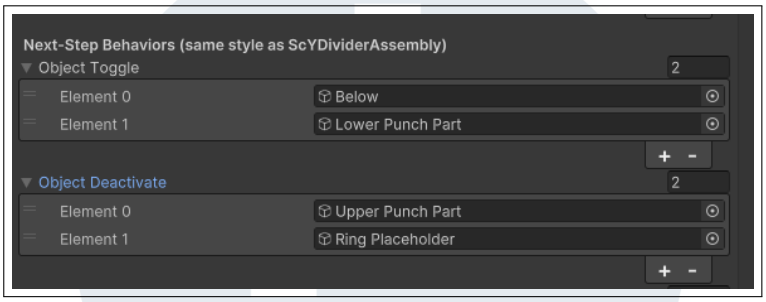
untuk langkah berikut, seperti daftar objek yang akan diaktifkan atau dinonaktifkan (`Object Toggle` dan `Object Deactivate`), serta pengaturan visual, yaitu pemilihan material untuk kondisi default dan highlight. Penataan ini memudahkan tim pengembang dalam memantau, memverifikasi, serta melakukan penyesuaian sesuai kebutuhan scenario pada tahap setup dan clean up.



Gambar 3.13. Pengaturan *Inspector* pada `ScPunchCheckerCU`, termasuk daftar *punch*, *behaviour* langkah berikut, dan konfigurasi material visual.

Gambar 3.14 memperlihatkan pengaturan *Next-Step Behaviors* pada Inspector, yang digunakan untuk mengelola perubahan status objek setelah suatu langkah assembly selesai dijalankan. Pada pengaturan ini terdapat dua bagian utama: `Object Toggle` dan `Object Deactivate`. Kumpulan pada `Object Toggle` memuat objek-objek (seperti `Below` dan `Lower Punch Part`) yang akan diaktifkan pada langkah berikutnya, sehingga pengguna dapat melanjutkan proses assembly dengan objek tersebut.

Sebaliknya, daftar `Object Deactivate` berisi objek-objek (misalnya `Upper Punch Part` dan `Ring Placeholder`) yang akan dinonaktifkan ketika transisi langkah terjadi, guna mencegah interaksi lanjutan yang tidak diperlukan. Pola penataan seperti ini membantu tim developer dalam menetapkan logika transisi antar-tahap assembly secara visual dan terstruktur, sehingga perubahan status objek dapat dikendalikan dengan mudah tanpa penyesuaian kode tambahan.



Gambar 3.14. Pengaturan *Next-Step Behaviors* pada *Inspector*: daftar objek yang diaktifkan dan dinonaktifkan setelah transisi langkah *assembly*.

Fungsi `HandlePunchPlacement` Kode 3.18 adalah coroutine yang bertugas memastikan semua efek penempatan objek punch dilakukan secara berurutan dan mulus setelah punch ditempatkan dalam socket yang tepat. Fungsi ini pertama-tama memberikan jeda singkat agar proses snap pada socket benar-benar selesai sebelum melanjutkan langkah berikutnya. Selanjutnya, bila objek punch bertipe `XRGrabInteractable`, sistem melakukan `SelectExit` pada `interactionManager` untuk menutup interaksi dengan benar.

Setelah itu, fungsi `MakePlacedInert` dipanggil untuk mengubah perilaku objek punch agar tidak bisa dipindahkan atau diinteraksikan secara fisik. Jika fitur otomatis disable socket diaktifkan (`autoDisableSocketOnPlace`), socket yang menerima punch akan dinonaktifkan setelah delay tertentu. Fungsi juga menangani proses reparent atau pemindahan parent objek punch dan target agar hierarki transform tetap konsisten. Terakhir, material objek punch diubah ke default untuk penanda visual bahwa penempatan telah selesai dan objek sudah siap pada status pasif.

```
1 private IEnumerator HandlePunchPlacement(GameObject punchGO,
      XRSocketInteractor socket)
2 {
3     // Tunggu sebentar agar proses snap selesai
4     yield return new WaitForSeconds(0.1f);
5
```

```
    // Selesaikan interaksi seleksi dengan XRI
    var interactable = punchGO.GetComponent<XRGrabInteractable>();
    if (interactable != null && socket != null && socket.
    interactionManager != null)
    {
        socket.interactionManager.SelectExit(socket, interactable)
    ;
    }

    yield return null;

    // Buat punch tidak lagi bisa diinteraksi (inert)
    MakePlacedInert(punchGO);

    // Otomatis disable socket jika diaktifkan
    if (autoDisableSocketOnPlace)
    {
        StartCoroutine(DisableSocketAfterDelay(socket.gameObject,
    socketDisableDelay));
    }

    // Reparent punch ke target transform
    if (parentOnAttach != null)
    {
        if (parentOnAttach.parent != originalParentOfTarget)
            parentOnAttach.SetParent(originalParentOfTarget);

        punchGO.transform.SetParent(parentOnAttach);
    }

    // Ubah material ke default sebagai penanda visual
    var renderer = punchGO.GetComponent<Renderer>();
    if (renderer != null && defaultMaterial != null)
        renderer.material = defaultMaterial;
}
```

Kode 3.18: *Coroutine* `HandlePunchPlacement` yang menangani efek penempatan dan transisi visual *punch*.

Fungsi `ReenableCollisionsForPunchExcept` Kode 3.19 digunakan untuk mengaktifkan kembali (reenable) interaksi fisik antar komponen `Collider` pada objek punch, kecuali terhadap objek tertentu dan/atau objek yang memiliki tag tertentu. Pada implementasinya, fungsi mencari semua collider pada objek punch

dan seluruh collider yang ada di scene. Sistem kemudian membangun daftar pengecualian berdasarkan objek tertentu (`exceptObject`) dan tag yang ingin diabaikan (`exceptTag`).

Selanjutnya, untuk setiap pasangan collider punch dan collider lain di scene, pengecekan dilakukan: jika collider tergolong dalam pengecualian, interaksi fisik tidak diaktifkan kembali; jika bukan, maka `Physics.IgnoreCollision` disetting ke `false` sehingga punch dapat berinteraksi secara fisik dengan collider tersebut. Fungsi ini penting untuk menjaga logika interaksi VR, dimana punch perlu dibatasi agar tidak berbenturan dengan objek tertentu, tetapi tetap berfungsi alami dengan objek lain. Pada akhir proses, log hasil jumlah collider yang diaktifkan kembali juga dicatat untuk kepentingan debugging.

```
private void ReenableCollisionsForPunchExcept(GameObject punchGO,
    GameObject exceptObject, string exceptTag)
{
    if (!punchGO) return;

    var punchCols = punchGO.GetComponentsInChildren<Collider>(true
    );
    if (punchCols == null || punchCols.Length == 0) return;

    Collider[] allColliders = FindObjectsOfType<Collider>(true);

    // Build exception set for specific object (and its children)
    HashSet<Collider> exceptSet = null;
    if (exceptObject != null)
        exceptSet = new HashSet<Collider>(exceptObject.
    GetComponentsInChildren<Collider>(true));

    int reenabled = 0;
    foreach (var pc in punchCols)
    {
        if (!pc) continue;

        foreach (var oc in allColliders)
        {
            if (!oc || oc == pc) continue;

            bool isException = false;

            // Exception: explicit object
            if (exceptSet != null && exceptSet.Contains(oc))
```

```
                isException = true;

            // Exception: tag (on object or its root)
            if (!isException && !string.IsNullOrEmpty(exceptTag))
            {
                var go = oc.gameObject;
                if (go.CompareTag(exceptTag) || go.transform.root.
    CompareTag(exceptTag))
                    isException = true;
            }

            if (isException) continue;

            Physics.IgnoreCollision(pc, oc, false);
            reenabled++;
        }
    }

    Debug.Log($"ScPunchCheckerCU: Re-enabled collisions for {
    punchGO.name} against {reenabled} colliders (except object '{
    exceptObject?.name}' and tag '{exceptTag}').");
}
```

Kode 3.19: Fungsi `ReenableCollisionsForPunchExcept` untuk mengaktifkan kembali collision objek punch dengan pengecualian tertentu.

Fungsi `TryFinishIfComplete` Kode 3.20 bertugas memvalidasi apakah seluruh ketentuan penyelesaian langkah punch sudah terpenuhi. Pada setiap pemanggilan, fungsi ini segera keluar jika status `nextStepDone` sudah aktif sebagai tanda proses telah selesai sebelumnya. Dua kondisi utama yang diperiksa: apakah jumlah punch yang berhasil ditempatkan sudah mencapai batas maksimum (`maxCount`) atau apakah seluruh punch yang ditugaskan sudah berhasil diletakkan pada tempatnya.

Jika salah satu kondisi tersebut tercapai, status langkah berikutnya diaktifkan dan sistem menampilkan pesan kelulusan di log. Fungsi juga menangani transisi visual pada UI, yaitu menghentikan timer, menonaktifkan `endCanvas`, dan mengaktifkan `doneCanvas` sambil menampilkan waktu penyelesaian secara detail. Terakhir, fungsi ini memulai proses perilaku lanjutan secara terjadwal menggunakan coroutine `RunNextStepBehaviorsDelayed`, yang akan mengatur langkah-langkah yang dibutuhkan setelah semua punch sukses ditempatkan. Dengan demikian, fungsi ini merupakan pengontrol utama alur kelulusan pada tahap punch assembly.

```
private void TryFinishIfComplete()
{
    if (nextStepDone) return;

    // Selesai jika mencapai batas atau seluruh punch telah dipasang
    bool reachedMax = placedPunches.Count >= maxCount;
    bool placedAllAssigned = punches != null && placedPunches.Count >= punches.Length;

    if (reachedMax || placedAllAssigned)
    {
        nextStepDone = true;
        Debug.Log("ScPunchCheckerSU: All punches placed (or max reached). Starting delayed next step behaviors.");

        if (endCanvas != null && doneCanvas != null)
        {
            timerActive = false;

            if (endCanvas != null) endCanvas.SetActive(false);
            if (doneCanvas != null) doneCanvas.SetActive(true);

            int minutes = Mathf.FloorToInt(timer / 60); // Get the minutes
            int seconds = Mathf.FloorToInt(timer % 60);

            // Tampilkan waktu selesai
            TextMeshProUGUI completionTimeText = doneCanvas.GetComponentInChildren<TextMeshProUGUI>();
            if (completionTimeText != null)
            {
                completionTimeText.text = $"Exam Completed !!\nTime Taken: {minutes} min {seconds} sec";
            }
        }

        // Mulai langkah lanjutan secara terjadwal
        StartCoroutine(RunNextStepBehaviorsDelayed());
    }
}
```

Kode 3.20: Fungsi `TryFinishIfComplete` untuk validasi kelulusan dan transisi ke langkah berikutnya pada *punch assembly*.

Fungsi `RunNextStepBehaviors` Kode 3.21 bertugas mengatur transisi antar langkah dalam proses assembly dengan mengubah status visual dan aktif objek-objek terkait. Pertama, fungsi ini mematikan placeholder pada objek yang sedang di-highlight atau mengembalikan material ke `defaultMaterial`. Selanjutnya, objek yang menjadi target highlight berikut akan diaktifkan atau diubah materialnya ke `highlightMaterial` agar menonjol secara visual. Sistem juga mengaktifkan dan menonaktifkan kumpulan objek sesuai kebutuhan langkah lanjutan, serta memanggil fungsi lanjutan pada `menuController` jika tersedia, untuk menampilkan menu atau instruksi berikutnya.

```
private void RunNextStepBehaviors()
{
    // De-highlight dan nonaktifkan placeholder
    foreach (var obj in currentHighlight)
        if (obj != null)
            obj.SetActive(!obj.name.Contains("Placeholder"));

    // Highlight dan aktifkan target berikut
    foreach (var obj in nextHighlight)
        if (obj != null)
            obj.SetActive(true);

    // Aktifkan dan nonaktifkan objek sesuai daftar
    foreach (var obj in objectToggle)
        if (obj != null) obj.SetActive(true);
    foreach (var obj in objectDeactivate)
        if (obj != null) obj.SetActive(false);

    // Lanjutkan ke menu berikutnya
    menuController?.ActivateNextCanvas();
}
```

Kode 3.21: Fungsi `RunNextStepBehaviors` untuk mengelola transisi dan status objek antar langkah *assembly*.

Pemusatan seluruh logika pada satu skrip, yaitu `ScPunchCheckerCU`, membuat proses Clean Up menjadi lebih efisien untuk skenario penempatan objek secara massal. Skrip ini menangani validasi penempatan pada socket yang sesuai tag, penguncian objek agar tidak dapat dipindahkan lagi, penonaktifan socket setelah berhasil menerima item, dan pengelolaan transisi antar langkah yang terstruktur setelah jumlah penempatan mencapai ambang yang ditentukan.

C. Pathing Menu

Struktur navigasi menu dirancang bertingkat empat untuk memudahkan pengguna dalam memilih skenario latihan yang diinginkan. Proses pemilihan dimulai dari panel utama, dilanjutkan dengan pemilihan mode (*Set Up* atau *Clean Up*), kemudian pengguna menentukan kategori komponen (*Outer, Semi Inner,* atau *Inner*), dan akhirnya memilih komponen tertentu atau opsi skenario lengkap (*Full*) yang mencakup semua langkah dalam satu kategori. Pendekatan hierarkis seperti ini membantu mengurangi beban kognitif pengguna, karena setiap pilihan disajikan secara bertahap sehingga proses seleksi tetap jelas dan mudah diikuti, bahkan ketika jumlah skenario banyak.

Pengelolaan transisi antar panel menu dan pemuatan scene berdasarkan pilihan pengguna dilakukan melalui skrip `ScMainMenu`. Fungsi `ShowOnly` digunakan untuk mengatur visibilitas setiap panel menu, sementara fungsi `LoadScene` menangani perpindahan ke scene simulasi. Pada tahap inisialisasi, panel utama langsung ditampilkan dan setiap tombol navigasi didaftarkan listener-nya. Dengan susunan ini, navigasi selalu diawali dari posisi yang konsisten dan jalur seleksi pengguna dapat dipantau secara sistematis.

Pada Kode 3.22, fungsi `Start()` menginisialisasi sistem navigasi menu dengan mengatur *event listener* untuk setiap tombol pada berbagai tingkat panel menu. Proses dimulai dengan hanya menampilkan `mainMenuPanel` menggunakan fungsi `ShowOnly`. Selanjutnya, setiap tombol disetting agar, saat diklik, hanya panel tujuan yang relevan yang akan muncul. Sebagai contoh, menekan `startButton` akan menampilkan `modeSelectionPanel`, sedangkan berbagai tombol lain menangani navigasi ke kategori dan subkategori seperti mode, jenis latihan (assembly/disassembly), serta panel spesifik untuk *level* berikutnya.

```
1  void Start()
2  {
3      ShowOnly(mainMenuPanel);
4
5      // Level 1
6      startButton.onClick.AddListener(() => ShowOnly(
       modeSelectionPanel));
7
8      // Level Selection
9      NormalButton.onClick.AddListener(() => ShowOnly(Normal));
10     ExamButton.onClick.AddListener(() => ShowOnly(Exam));
11
```

```
12      // Level 2
13      assemblyButton.onClick.AddListener(() => ShowOnly(
        assemblyPanel));
14      disassemblyButton.onClick.AddListener(() => ShowOnly(
        disassemblyPanel));
15      assemblyButtonExam.onClick.AddListener(() => ShowOnly(
        assemblyPanelExam));
16      disassemblyButtonExam.onClick.AddListener(() => ShowOnly(
        disassemblyPanelExam));
17
18      // Level 3 - Show Level 4 Panel
19      outerAssemblyButton.onClick.AddListener(() => ShowOnly(
        outerAssemblyPanel4));
20      semiInnerAssemblyButton.onClick.AddListener(() => ShowOnly(
        semiInnerAssemblyPanel4));
21      innerAssemblyButton.onClick.AddListener(() => ShowOnly(
        innerAssemblyPanel4));
22      outerDisassemblyButton.onClick.AddListener(() => ShowOnly(
        outerDisassemblyPanel4));
23      semiInnerDisassemblyButton.onClick.AddListener(() => ShowOnly(
        semiInnerDisassemblyPanel4));
24      innerDisassemblyButton.onClick.AddListener(() => ShowOnly(
        innerDisassemblyPanel4));
25
26      outerAssemblyButtonExam.onClick.AddListener(() => ShowOnly(
        outerAssemblyPanel4Exam));
27      semiInnerAssemblyButtonExam.onClick.AddListener(() => ShowOnly
        (semiInnerAssemblyPanel4Exam));
28      innerAssemblyButtonExam.onClick.AddListener(() => ShowOnly(
        innerAssemblyPanel4Exam));
29      outerDisassemblyButtonExam.onClick.AddListener(() => ShowOnly(
        outerDisassemblyPanel4Exam));
30      semiInnerDisassemblyButtonExam.onClick.AddListener(() =>
        ShowOnly(semiInnerDisassemblyPanel4Exam));
31      innerDisassemblyButtonExam.onClick.AddListener(() => ShowOnly(
        innerDisassemblyPanel4Exam));
32  }
```

Kode 3.22: Inisialisasi dan pengaturan *event* navigasi pada sistem menu VR.

Penanganan tombol Level 4 pada menu VR dilakukan dengan mendaftarkan setiap tombol untuk langsung memuat scene yang sesuai, baik untuk mode latihan biasa maupun exam. Mekanisme ini memungkinkan setiap tombol (misal `Dies`,

`Ejection Cam`, dll.) pada kategori assembly dan disassembly mengarahkan pengguna ke skenario praktikum yang spesifik hanya dengan satu klik. Fungsi `LoadScene` dipanggil secara langsung lewat `AddListener` pada event `onClick` tiap tombol, sehingga struktur menu tetap modular dan mudah dikembangkan. Implementasi singkatnya dapat dilihat pada Kode 3.23.

```
// Contoh singkat event handler Level 4
assemblyFullButton.onClick.AddListener(() => LoadScene("Assembly Full"));
assemblyDiesButton.onClick.AddListener(() => LoadScene("Assembly Dies"));
assemblyFillCamButton.onClick.AddListener(() => LoadScene("Assembly Fill Cam"));

disassemblyFullButton.onClick.AddListener(() => LoadScene("Disassembly Full"));
disassemblyDiesButton.onClick.AddListener(() => LoadScene("Disassembly Dies"));
disassemblyFillCamButton.onClick.AddListener(() => LoadScene("Disassembly Fill Cam"));

// Exam mode
assemblyFullExamButton.onClick.AddListener(() => LoadScene("Assembly Full Exam"));
disassemblyFullExamButton.onClick.AddListener(() => LoadScene("Disassembly Full Exam"));
```

Kode 3.23: Registrasi *event* tombol *Level* 4 untuk pemuatan *scene* pada *menu* VR

Fungsi ini mengatur tombol ”Back” pada level 3 dan level 4 menu sehingga ketika ditekan, panel akan kembali ke panel sebelumnya secara hierarkis. Tombol level 3 mengarahkan kembali pengguna ke panel pemilihan mode utama, baik untuk mode biasa maupun exam. Sedangkan tombol level 4 mengembalikan tampilan ke panel assembly atau disassembly sesuai konteks dan mode yang sedang aktif. Pendekatan ini menjaga navigasi tetap intuitif dan konsisten, sehingga pengguna dapat dengan mudah mundur antar lapisan menu tanpa kebingungan. Kode singkat pengaturannya dapat dilihat di Kode 3.24.

```
assemblyBackButton.onClick.AddListener(() => ShowOnly(modeSelectionPanel));
disassemblyBackButton.onClick.AddListener(() => ShowOnly(modeSelectionPanel));
```

```
assemblyBackButtonExam.onClick.AddListener(() => ShowOnly(
    modeSelectionPanel));
disassemblyBackButtonExam.onClick.AddListener(() => ShowOnly(
    modeSelectionPanel));

// Level 4 Back buttons
outerAssemblyBackButton.onClick.AddListener(() => ShowOnly(
    assemblyPanel));
semiInnerAssemblyBackButton.onClick.AddListener(() => ShowOnly(
    assemblyPanel));
innerAssemblyBackButton.onClick.AddListener(() => ShowOnly(
    assemblyPanel));
outerDisassemblyBackButton.onClick.AddListener(() => ShowOnly(
    disassemblyPanel));
semiInnerDisassemblyBackButton.onClick.AddListener(() => ShowOnly(
    disassemblyPanel));
innerDisassemblyBackButton.onClick.AddListener(() => ShowOnly(
    disassemblyPanel));

outerAssemblyBackButtonExam.onClick.AddListener(() => ShowOnly(
    assemblyPanelExam));
semiInnerAssemblyBackButtonExam.onClick.AddListener(() => ShowOnly
    (assemblyPanelExam));
innerAssemblyBackButtonExam.onClick.AddListener(() => ShowOnly(
    assemblyPanelExam));
outerDisassemblyBackButtonExam.onClick.AddListener(() => ShowOnly(
    disassemblyPanelExam));
semiInnerDisassemblyBackButtonExam.onClick.AddListener(() =>
    ShowOnly(disassemblyPanelExam));
innerDisassemblyBackButtonExam.onClick.AddListener(() => ShowOnly(
    disassemblyPanelExam));
```

Kode 3.24: Pengaturan tombol *Back* untuk navigasi antar panel *menu* VR

D. Exam Mode

Sistem evaluasi (*Exam Mode*) digunakan sebagai sarana penilaian yang menguji seberapa paham pengguna terhadap prosedur *Clean Up* dan *Set Up*, tanpa adanya fitur bantuan visual ataupun petunjuk langkah. Pada mode ini, tidak seperti mode pelatihan *(Training Mode)* yang menyediakan *highlight* objek berikut, *tutorial*, dan menu bantuan, seluruh elemen pemandu dihilangkan sehingga peserta benar-benar bergantung pada penguasaan langkah yang telah mereka pelajari.

Di sisi implementasi, *Exam Mode* memanfaatkan skrip serta mekanisme validasi yang sama persis dengan mode latihan; tidak diperlukan skrip tambahan atau logika baru. Perbedaan utama hanya ada pada pengaturan elemen di Inspector Unity. Pada mode evaluasi, komponen seperti daftar objek untuk *highlight* (misal, `nextHighlight` di skrip `ScSocketAttach`, `ScAttachCheckerAdvance`, dan `ScScrewCheckerAdvance`) sengaja dibiarkan kosong, sementara pada mode latihan biasanya diisi agar objek dapat disorot sesudah satu langkah selesai. Untuk memastikan alur tetap berlanjut, parameter `objectsToActivate` tetap dikonfigurasi sehingga objek selanjutnya tetap bisa digunakan.

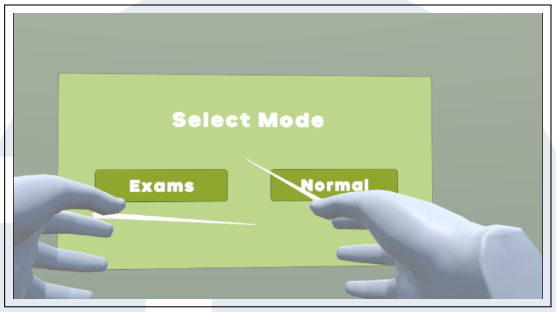
Tidak hanya menghapus *highlight*, pada mode evaluasi semua *tutorial* dan panel bantuan juga dinonaktifkan. Instruksi bertahap, ilustrasi, maupun tombol petunjuk tidak muncul, sehingga pengguna hanya akan dihadapkan pada lingkungan tugas tanpa bantuan pola kerja eksplisit. Model ini memungkinkan instruktur menilai secara obyektif kemampuan pengguna dalam mengikuti alur kerja dari awal hingga akhir. Selama proses evaluasi berlangsung, sistem tetap menjalankan pengecekan *back-end* seperti pada mode latihan untuk merekam progres, mendeteksi kesalahan, dan menghitung waktu penyelesaian. Hasil pengujian ini dapat dipakai sebagai bahan laporan maupun penentuan nilai berdasarkan kriteria seperti ketepatan langkah, tingkat kesalahan, dan durasi kerja.

Karena pengaturannya cukup dengan menyesuaikan parameter di *Inspector* tanpa mengubah kode sumber, proses pengembangan *Exam Mode* sangat praktis. Tim pengembang dapat dengan cepat menjadikan skenario latihan sebagai skenario evaluasi hanya lewat perubahan konfigurasi di *Unity Editor*, tanpa perlu memodifikasi atau menduplikasi logika program yang sudah ada.

E. Hasil Implementasi

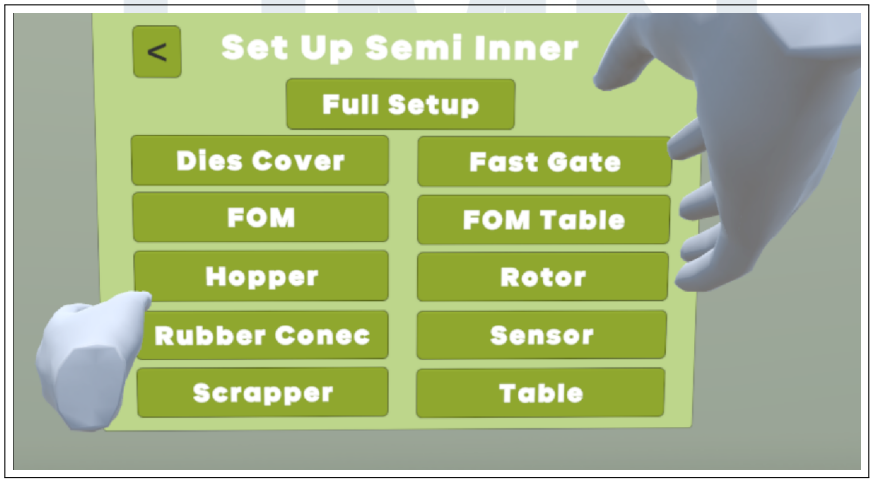
Fase pengembangan ini telah menghasilkan aplikasi VR yang telah berfungsi secara optimal dan mencakup skenario *Clean-Up* serta *Set-Up* bagi seluruh kategori komponen, meliputi *Outer, Semi Inner,* dan *Inner*. Implementasi sistem navigasi menu yang terstruktur dalam empat tingkatan berjalan dengan konsisten, sehingga pengguna dapat menelusuri pilihan skenario latihan mulai dari pemilihan mode hingga seleksi komponen tertentu secara efisien dan intuitif.

Gambar 3.15 memperlihatkan tampilan panel pemilihan mode pada aplikasi VR. Pada layar ini, pengguna dihadapkan pada dua pilihan utama: `Exams` dan `Normal`, yang masing-masing diwakili oleh tombol dengan warna hijau dan tulisan tebal. Judul "*Select Mode*" di bagian atas panel memberikan arahan yang jelas terkait langkah awal yang harus dipilih peserta sebelum memulai simulasi.



Gambar 3.15. Panel pemilihan mode aplikasi VR dengan tombol Exams dan Normal, serta interaksi tangan virtual untuk seleksi skenario.

Gambar 3.16 menampilkan antarmuka menu VR pada tahap pemilihan skenario "*Set Up Semi Inner.*" Pengguna diberikan beberapa opsi interaktif untuk memulai simulasi, di antaranya tombol `Full Setup` untuk menjalankan seluruh proses setup sekaligus, serta tombol-tombol lain yang mewakili komponen individual seperti `Dies Cover`, `Fast Gate`, `FOM`, `Hopper`, dan sebagainya.



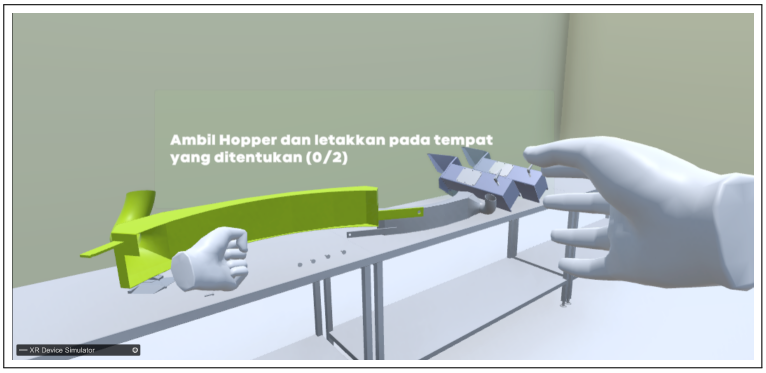
Gambar 3.16. Tampilan menu pemilihan skenario *Set Up Semi Inner* pada aplikasi VR dengan pilihan komponen setup yang dapat diakses melalui gesture tangan virtual.

Gambar 3.17 dan Gambar 3.18 memperlihatkan aktivitas pengguna pada tahap *Set Up Semi Inner Full* dalam aplikasi simulasi VR. Dalam skenario ini, pengguna diminta melakukan perakitan (*assembly*) komponen Semi Inner secara lengkap, salah satunya dengan menempatkan bagian *Hopper* pada posisi yang telah ditentukan di mesin utama.

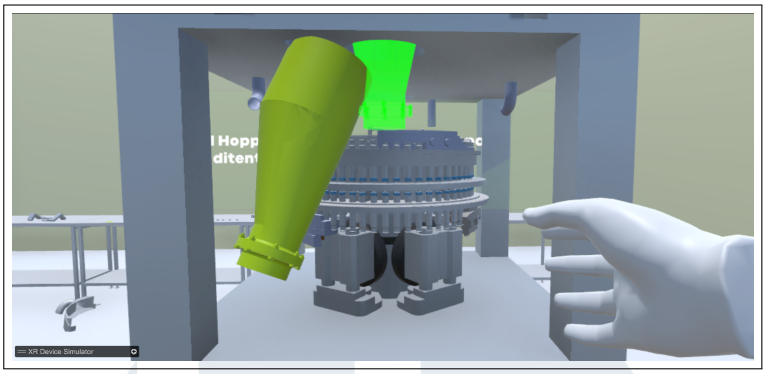
Pada gambar pertama, terlihat tangan virtual sedang melakukan aksi *grab* terhadap model *Hopper* yang disorot dengan warna hijau cerah. Sistem menggunakan fitur *XR Grab Interactable,* sehingga objek *Hopper* bisa diambil, digerakkan, dan diposisikan secara langsung oleh pengguna lewat gestur tangan di lingkungan VR. Warna hijau digunakan sebagai *highlight* visual untuk menandai komponen utama yang sedang aktif atau perlu diinteraksikan, mengikuti praktik umum dalam *VR training* sebagai penanda langkah kerja.

Gambar kedua menunjukkan instruksi pada layar: ”Ambil *Hopper* dan letakkan pada tempat yang ditentukan (0/2)”. Teks ini memberikan arahan langsung ke pengguna bersama progres yang sedang berlangsung (belum ada *Hopper* yang dipasang dari dua yang harus dipasang). Di sini, tangan virtual juga sedang bersiap melakukan interaksi *grab* pada objek *Hopper* yang masih di atas meja kerja. *Highlight* warna hijau tetap digunakan untuk membedakan *Hopper* sebagai objek yang bisa diambil dan dipindahkan pada tahap ini.

Penerapan teknik *highlight* dan *grab interactable* tidak hanya mendukung navigasi visual namun juga memfasilitasi alur assembly secara lebih intuitif. Sistem ini memastikan pengguna memahami bagian mana yang harus diambil dan dipasang, sekaligus mengasah keterampilan dalam mengenali urutan perakitan komponen Semi Inner sehingga proses pelatihan dapat berjalan efektif.



Gambar 3.17. Aksi grab dan penempatan Hopper pada proses *Set Up Semi Inner Full*: highlight hijau menandakan objek target yang aktif untuk dipasang.



Gambar 3.18. Instruksi interaktif "Ambil *Hopper* dan letakkan pada tempat yang ditentukan (0/2)" pada tahap *Set Up Semi Inner Full*, dengan *Hopper* yang dapat di-*grab* dan dipasang oleh pengguna.

3.5.3 Testing dan Validasi

Proses pengujian dilakukan melalui dua lintasan utama yang berlangsung secara bersamaan dengan tahap pengembangan aplikasi, sehingga tim dapat secara dinamis melakukan penyesuaian berdasarkan masukan yang diperoleh. Jalur pertama melibatkan pengujian internal oleh tim pengembang dan manajer proyek, yang berfungsi untuk memastikan stabilitas serta konsistensi fitur selama proses iterasi. Sementara itu, jalur kedua dilakukan secara terbatas bersama perwakilan klien, dengan tujuan untuk memastikan bahwa setiap skenario yang dikembangkan benar-benar sudah selaras dengan prosedur kerja aktual di lapangan. Selain menggunakan perangkat VR secara langsung, pengujian juga memanfaatkan fitur *casting* pada *web Horizon Meta*.

A. Testing Internal

Pada tahap validasi ini, pendekatan *black box testing* digunakan untuk menilai ketepatan fungsional aplikasi tanpa memeriksa detail kode internalnya. Fokus utama uji coba adalah memastikan interaksi ambil dan lepas objek (*grab/release*), urutan prosedural, navigasi menu, serta konsistensi antara tampilan visual dan efek *audio* berjalan sesuai ekspektasi pengguna. Semua pengujian dilakukan langsung menggunakan perangkat *Virtual Reality* sehingga evaluasi benar-benar mencerminkan pengalaman pengguna akhir.

Setiap kasus yang ditemukan selama pengujian dicatat dan dikategorikan berdasarkan urgensi, lalu dituangkan sebagai daftar perbaikan pada siklus pengembangan berikutnya. Keberhasilan diuji berdasarkan kelancaran pelaksanaan

skenario *Clean-Up* dan *Set-Up* tanpa ditemui bug, akurasi urutan langkah kerja sesuai rancangan, dan kejelasan instruksi yang mudah diikuti peserta. Metode pengujian ini sangat relevan untuk aplikasi VR, karena menitikberatkan pada hasil interaksi serta *output* sistem yang diamati dari sisi pengguna, bukan dari sisi logika pemrogramannya.

Gambar 3.20 mendokumentasikan sesi pengujian internal yang digunakan untuk menilai fungsi utama aplikasi, termasuk kestabilan simulasi fisika, interaksi kolisi, dan konsistensi antar komponen sebelum aplikasi diuji lebih lanjut.

Gambar 3.19. Sesi pengujian internal aplikasi VR dengan metode *black box*: evaluasi interaksi, validasi urutan kerja, dan penyesuaian antar komponen dalam lingkungan VR

Gambar 3.20. Situasi pengujian internal aplikasi VR: verifikasi interaksi dasar, pengujian stabilitas fisika dan kolisi, serta sinkronisasi komponen utama sebelum uji eksternal.

B. Testing dengan Client

Tahap pengujian ini memanfaatkan pendekatan *white box testing* untuk menguji kesesuaian dan integritas skenario VR terhadap prosedur *Clean-Up* dan *Set-Up* yang diterapkan di lingkungan kerja sesungguhnya. Selain demonstrasi dan *review* berkala, penguji dari pihak klien diberikan akses untuk mengamati serta mengevaluasi struktur internal aplikasi, logika kode, serta jalur eksekusi setiap fungsi utama yang berkaitan langsung dengan *workflow* operasional. Proses ini melibatkan observasi langsung di lapangan, diskusi mendalam, serta pengumpulan masukan dari operator terkait, sehingga optimalisasi sistem dapat dilakukan secara tepat sesuai kebutuhan nyata.

Setiap temuan, mulai dari ketidaksesuaian urutan langkah, kebutuhan tambahan umpan balik, hingga penyesuaian interaksi, didokumentasikan secara sistematis sebagai dasar perbaikan selanjutnya. Gambar 3.21 menunjukkan sesi evaluasi bersama klien saat menguji alur simulasi, menilai kejelasan instruksi, dan memastikan perilaku sistem sudah sejalan dengan kebutuhan operasional di lapangan.

Gambar 3.21. Sesi evaluasi white box bersama klien: analisis alur simulasi, validasi instruksi, dan *review scene* pada proses QC.

C. Test Case

Dalam pengembangan aplikasi Virtual Reality (VR), proses pengujian dirancang untuk memastikan bahwa seluruh fungsi, alur interaksi, serta aspek teknis aplikasi berjalan sesuai standar yang ditetapkan dan kebutuhan pengguna akhir. Pengujian dilakukan secara menyeluruh dengan memanfaatkan dua pendekatan utama: *black box testing*, yang berfokus pada evaluasi fungsi dari sisi pengguna tanpa melihat struktur kode internal, serta *white box testing*, yang menguji keandalan, jalur logika, dan stabilitas skrip pada tingkat kode sumber.

Setiap skenario pengujian dituangkan ke dalam serangkaian *test case*, mulai dari verifikasi peluncuran aplikasi, presisi interaksi, kestabilan performa grafis dan *audio*, hingga uji keamanan *event* dan *recovery* dari skenario anomali. Selain itu, daftar pengujian ini juga memuat aspek penting yang krusial dalam *VR development*, seperti *bug finding*, *bug fixing*, validasi *feedback* sensorik, serta penggunaan sumber daya perangkat secara efisien. Dengan dokumentasi test case ini, pengembang dapat memantau kemajuan pengujian, mengidentifikasi gangguan atau *bug* secara sistematis, serta menjalankan proses perbaikan berdasarkan temuan yang konkret. Rangkuman dan kategorisasi lengkap seluruh *test case* yang diterapkan dapat dilihat pada Tabel 3.6.

Tabel 3.6. Kategorisasi dan Dokumentasi *test case* untuk VR

Kategori	**Test Case**	**Deskripsi**
Black Box	TC-01	Memverifikasi aplikasi dapat diluncurkan tanpa *crash*.
	TC-02	Memastikan lingkungan VR (*controller, headset*) terinisialisasi dengan benar.
	TC-03	Memvalidasi tampilan layar pemuatan selama proses inisialisasi.
	TC-04	Memverifikasi objek dimuat sempurna di lingkungan VR.
	TC-05	Memastikan pengguna dapat memilih dan menyorot komponen.
	TC-06	Menguji interaksi *grab, move,* dan *rotate* menggunakan *VR controllers.*
	TC-07	Memvalidasi presisi interaksi (*snapping* komponen).

Tabel 3.6: Kategorisasi dan Dokumentasi *test case* untuk VR (lanjutan)

Kategori	**Test Case**	**Deskripsi**
	TC-08	Memastikan tindakan salah memicu umpan balik (getaran, suara, pesan *error*).
	TC-09	Verifikasi urutan perakitan komponen (*sequence logic*).
	TC-10	Pengecekan ketersediaan petunjuk (*hint/step guide*) jika diterapkan.
	TC-11	Memastikan perakitan tidak presisi diblokir atau dikoreksi otomatis.
	TC-12	Pengujian alat virtual (obeng, kunci, dsb.) bila ada.
	TC-13	Cek notifikasi penyelesaian perakitan.
	TC-14	Verifikasi pembongkaran komponen sesuai urutan.
	TC-15	Memastikan alat pembongkaran tersedia dan bisa dipakai.
	TC-16	Pengujian kebebasan melepas komponen tanpa mengganggu alur.
	TC-17	Validasi kemunculan peringatan pada tindakan salah saat pembongkaran.
	TC-18	Cek fungsi kontrol VR (tombol, *joystick*, *gesture*).
	TC-19	Memastikan *menu/HUD/toolbar* mudah diakses dan digunakan.
	TC-20	Verifikasi kejelasan teks *tooltip*/instruksi dan penempatan *UI* di ruang VR.
	TC-21	Pengujian tampilan dari sisi pengguna (*visual glitch*, tekstur, *lighting*, *audio, FPS*).
	TC-22	Validasi *feedback audio* (aksi dan *ambience*).
	TC-23	Pengujian performa: *frame rate*, latensi, penggunaan memori dan GPU saat berinteraksi.
	TC-24	Memastikan aplikasi tidak *crash* saat aksi tidak valid.

Tabel 3.6: Kategorisasi dan Dokumentasi *test case* untuk VR (lanjutan)

Kategori	**Test Case**	**Deskripsi**
White Box	TC-W01	Pengujian unit *(unit test) script logic* kritikal, seperti urutan validasi, *checking* komponen, *socket events*, dan *scene management*.
	TC-W02	Analisis *code coverage* untuk memastikan semua cabang logika telah diuji.
	TC-W03	Penelusuran (*tracing*) bug pada fungsi utama, pencarian *hidden bugs* pada *branching code*.
	TC-W04	Uji integrasi antar modul *script*, misal sinkronisasi status objek, menu, dan *checker*.
	TC-W05	Pemastian *exception handling* pada fungsi *asynchronous* (*coroutine, event handler*).
	TC-W06	Pemeriksaan efektivitas *logging*, fiksasi *debugging output*, dan *error reporting.*
	TC-W07	Validasi synchronisasi *event* antar objek pada *frame-rate* tinggi (*thread-safety testing*).
	TC-W08	Review efisiensi dan redundansi kode, serta deteksi *memory leak* pada *script Unity*.
	TC-W09	Pengujian *fallback* untuk skenario *edge case* (misal, kehilangan input, *failover* initialisasi ulang).
	TC-W10	*Review handling state transitions* ketika user keluar/masuk scene, atau *recovery* dari *crash* ringan.
	TC-W11	Uji fitur *diagnostic internal* yang membantu finding dan *bug fixing* pada development VR.

Tabel 3.6: Kategorisasi dan Dokumentasi *test case* untuk VR (lanjutan)

Kategori	**Test Case**	**Deskripsi**
Catatan Tambahan: • *Issue Tracking*: Setiap temuan bug, anomali visual, atau ketidaksesuaian UX/waktu respon didokumentasikan dan ditindaklanjuti secara iteratif. • *Bug Fixing/Finding*: Pengujian regresi perlu dijalankan pada setiap patch/iterasi baru. • *Performance Monitoring*: Pastikan alat profiling aktif selama uji berat (stress test VR), pantau resource spikes, memory leak, dan anomali interaksi. • *Usability/Accessibility*: Selain lewat black box dan white box, integrasikan pengujian usability (kenyamanan, presence, user journey) serta fitur aksesibilitas warna, suara, dan tinggi badan.		

3.6 Kendala dan Solusi yang Ditemukan

Selama proses pengembangan aplikasi *Virtual Reality* (VR), tim menghadapi berbagai tantangan yang cukup kompleks baik di aspek teknis maupun non-teknis. Berikut ini adalah ringkasan hambatan utama yang dialami:

1. Keterbatasan Dokumentasi Teknis Mesin Produksi
   Data terkait mesin yang akan disimulasikan sering kali kurang lengkap, tidak terstruktur, atau minim detail. Informasi yang diberikan oleh pengguna maupun pemilik mesin kadang hanya mencakup hal-hal umum, sehingga pengembang harus mempelajari mekanisme kerja mesin secara manual. Hal ini memperlambat pemetaan alur dan desain interaksi untuk simulasi VR.

2. Model 3D yang Tidak Teroptimasi
   Tim desain awal sering kali memberikan model 3D dengan tingkat detail yang terlalu tinggi, misalnya jumlah poligon yang besar, material dengan resolusi tinggi, dan struktur *mesh* yang rumit. Ketika aset tersebut diimpor ke *Unity Engine*, performa headset VR menurun tajam, seperti *frame rate* rendah atau lag tinggi, yang berdampak pada kenyamanan pengguna.

3. Implementasi Interaksi yang Rumit pada VR
   Tidak semua aksi di dunia nyata dapat langsung diterapkan ke VR secara digital. Beberapa gerakan seperti rotasi kompleks, pemasangan komponen presisi, atau interaksi yang bergantung pada kondisi tertentu, memerlukan logika pemrograman khusus. Selain itu, kendala pada sistem fisika bawaan *Unity Engine* membuat simulasi dengan ketelitian tinggi cukup menantang.

4. Konfigurasi *Layer* dan *Collision Matrix* yang Kurang Tepat
   Kesalahan dalam pengaturan *layer* atau *collision matrix* dapat menyebabkan objek berinteraksi secara tidak diinginkan, seperti *avatar* melayang atau tertarik ke arah tertentu.

Untuk mengatasi kendala-kendala tersebut, tim pengembang melakukan sejumlah langkah strategis agar proses pengembangan lebih efisien, performa aplikasi meningkat, serta interaksi di VR menjadi lebih akurat. Langkah yang dilaksanakan meliputi:

1. Observasi Langsung dan Dokumentasi Visual
   Pengembang melakukan kunjungan ke pabrik dan berdialog dengan teknisi untuk memahami proses dan komponen mesin secara detail. Dokumentasi visual berupa foto dan video jalannya perakitan serta pembongkaran juga dikumpulkan sebagai referensi utama dalam merancang alur interaksi VR.

2. Optimalisasi Aset 3D demi Performa
   Penyesuaian dilakukan dengan mengurangi jumlah poligon pada model (*mesh decimation*), menggabungkan tekstur dan cahaya (*texture baking*), serta menerapkan sistem *Level of Detail (LOD)* agar tampilan model menyesuaikan jarak dari pengguna tanpa mengorbankan visual penting.

3. Pengembangan Skrip Kustom dan Uji Coba Pengguna
   Tim membuat skrip khusus berbasis *XR Interaction Toolkit* untuk mendukung interaksi yang kompleks, seperti rotasi bersyarat dan validasi urutan perakitan. Interaksi tersebut diuji secara bertahap dengan melibatkan pengguna internal maupun eksternal demi menghasilkan pengalaman yang alami dan mudah digunakan.

4. Pengaturan Layer dan Collision Matrix secara Spesifik
   Pengembang membuat kategori *layer* yang terpisah sehingga objek dengan *collider* tidak saling bertabrakan atau menimbulkan *bug*.