BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama masa magang di PT Kalbe Farma Tbk, posisi yang dijalankan adalah VR/XR Developer di bawah supervisi langsung Bapak Miftahul Tirta Irawan sebagai IoT Engineer. Tugas utama berfokus pada pengembangan aplikasi *Virtual Reality* (VR) untuk mensimulasikan prosedur pemasangan dan pembongkaran salah satu mesin yang digunakan di pabrik klien.

Pelaksanaan proyek memerlukan koordinasi lintas divisi yang intensif. Tim VR bertanggung jawab end-to-end mulai dari perancangan, implementasi, hingga pengujian aplikasi. Tim 3D Model menyiapkan aset mesin dalam bentuk 3D untuk integrasi ke lingkungan VR. Sementara itu, tim Project Management (PM) menyusun alur simulasi (*flow*) berdasarkan masukan dari klien serta mengoordinasikan kebutuhan teknis dan fungsional antartim.

Koordinasi dilakukan secara daring melalui WhatsApp dan Microsoft Teams, sedangkan dokumentasi serta pelacakan progres dikelola menggunakan GitHub, SharePoint, Asana, dan Notion. Perancangan *flow* memanfaatkan Draw.io, dan Unity digunakan sebagai platform utama pengembangan VR. Rapat mingguan diselenggarakan untuk pelaporan progres, pembahasan kendala teknis, dan penetapan tindak lanjut. Selain itu, terdapat keterlibatan dalam sesi uji coba aplikasi bersama pihak klien serta kegiatan kegiatan langsung di lapangan (observasi langsung) di pabrik guna mempelajari proses *Clean-Up Set-Up* (CUSU) pada mesin yang disimulasikan.

Sebagai bagian dari siklus validasi, dilakukan kegiatan langsung di lapangan untuk keperluan *testing* bersama pihak klien. Kegiatan ini mencakup pengamatan langsung alur *Clean-Up Set-Up* (CUSU), uji coba skenario pada aplikasi VR, serta pengumpulan umpan balik terstruktur dari operator dan *Subject Matter Expert* (SME). Hasil kegiatan langsung di lapangan didokumentasikan dalam bentuk temuan prioritas (mismatch urutan langkah, kebutuhan *feedback* visual/*Sound Effects* (SFX) tambahan, dan penyesuaian *collision/interaction*) yang kemudian diterjemahkan menjadi *action items* untuk iterasi perbaikan berikutnya (refactor skrip, penyelarasan menu/instruksi, dan penyesuaian aset/kolisi). Pendekatan ini memastikan tingkat kesesuaian prosedural, kenyamanan penggunaan, dan

efektivitas pelatihan sebelum melanjutkan ke tahap *User Acceptance Test* (UAT) yang lebih luas.

3.2 Tugas yang Dilakukan

Selama pelaksanaan magang di PT Kalbe Farma Tbk sebagai VR/XR Developer, beberapa tugas utama yang dilakukan antara lain sebagai berikut.

1. Mengembangkan alur *Clean-Up* dan *Set-Up* pada aplikasi VR (Unity).
2. Meningkatkan skrip interaksi dan validasi langkah (refactor, penambahan checker berbasis *counter*/event, perbaikan urutan prosedur).
3. Meningkatkan stabilitas interaksi (penataan *collider*/*rigidbody*/*socket*, matrix kolisi, dan mekanisme re-enable *collision*).
4. Meningkatkan tutorial menu (alur instruksi, indikator progres, dan standar *highlight*).
5. Meningkatkan pengalaman audiovisual (*sound effects*/SFX dan *feedback* visual) agar selaras dengan skenario produksi.
6. Kegiatan langsung di lapangan dan pengujian bersama pihak klien untuk memperoleh *feedback* dan menetapkan tindak lanjut perbaikan.

Seluruh aktivitas berfokus pada penyempurnaan fitur dan kualitas pengalaman pengguna: alur *Clean-Up/Set-Up* disiapkan untuk skenario pelatihan, logika skrip dirapikan agar validasi langkah lebih andal, serta stabilitas interaksi ditingkatkan melalui pengaturan fisika dan kolisi yang konsisten. Perbaikan menu dan SFX ditujukan untuk memperjelas instruksi, memberikan *feedback* yang informatif, dan menjaga ritme latihan operator. Integrasi aset 3D memastikan kesesuaian visual dan dimensi kerja di VR, sementara kegiatan langsung di lapangan dan sesi uji bersama pihak klien digunakan untuk memverifikasi keselarasan prosedur dan mengarahkan iterasi perbaikan berikutnya.

3.3 Uraian Pelaksanaan Magang

Tabel 3.1 menampilkan ringkasan linimasa kegiatan selama magang di PT Kalbe Farma Tbk. Isi tabel merangkum aktivitas mingguan yang menggambarkan

kemajuan bertahap pengembangan aplikasi *Virtual Reality* (VR) hingga pengujian dan dokumentasi. Melalui linimasa ini, pembaca memperoleh gambaran utuh mengenai alur kerja dan kontribusi yang diberikan sepanjang program, sekaligus melihat keterkaitan antar kegiatan dalam mendukung tercapainya tujuan proyek.

Tabel 3.1. Pekerjaan yang dilaksanakan setiap minggu

Minggu Ke -	Pekerjaan yang dilakukan
1	Kegiatan yang dilakukan berfokus pada pengembangan beberapa scene set up serta bug fixing.
2	Kegiatan masih berfokus pada pengembangan scene dan melanjutkan scene set up.
3	Kegiatan difokuskan pada internal testing scene yang sudah selesai dikerjakan dan memperbaiki berdasarkan feedback testing tersebut.
4	Kegiatan berfokus pada perbaikan scene yang telah dibuat dan dilakukan beberapa adjustment seperti ukurannya.
5	Kegiatan berfokus pada internal testing scene yang telah diperbaiki dan dilakukan penambahan sound effect.
6	Kegiatan yang dilakukan berfokus pada pergantian dan penambahan komponen pada scene dan meeting dengan pihak klien.
7	Kegiatan berfokus untuk melanjutkan scene yang belum siap untuk bagian clean up.
8	Kegiatan masih berfokus pada pembuatan scene clean up.
9	Kegiatan yang dilakukan berfokus pada pembuatan scene set up dan penambahan scene clean up.
10	Dilanjutkan dengan internal testing dan melakukan perbaikan pada bug yang ditemukan.
11	Melakukan scene splitting menjadi beberapa bagian scene dan menyelesaikan tutorial menu.
12	Dilakukan testing internal untuk menguji scene yang telah dibuat serta menambahkan sound effect tambahan.
13	Dilakukan testing dengan pihak klien dan melakukan peningkatan pada beberapa scene.
	Lanjut pada halaman berikutnya

Tabel 3.1. Pekerjaan yang dilaksanakan setiap minggu (Lanjutan)

Minggu Ke -	Pekerjaan yang dilakukan
14	Dilakukan pergantian bahasa pada tutorial menu dan melakukan peningkatan untuk UI Menu.
15	Kegiatan berfokus pada pergantian script dan mengubah overall proses systemnya.
16	Dilakukan internal testing dan memperbaiki bug yang ditemukan.
17	Dilakukan peningkatan bug fixing terhadap scene yang telah dibuat.
18	Kegiatan berfokus pada pengembangan exam mode.
19	Kegiatan yang dilakukan masih berfokus untuk mengembangkan exam mode.

Kegiatan pada Tabel 3.1 menunjukkan bahwa program magang dijalankan secara bertahap dan terstruktur, dengan penekanan pada pengembangan aplikasi VR yang fungsional serta relevan bagi kebutuhan industri. Setiap pekan diisi rangkaian aktivitas yang saling melengkapi dari observasi lapangan hingga implementasi teknis menandakan keterlibatan aktif dalam seluruh tahapan pengembangan. Pola ini sekaligus merefleksikan praktik kerja kolaboratif dan iteratif yang diterapkan pada proyek simulasi.

3.4 Perangkat Penunjang Pelaksanaan Magang

Dalam pelaksanaan magang, diperlukan beragam perangkat lunak (software), perangkat keras (hardware), serta kerangka kerja (framework) dan pustaka (library) yang masing-masing berperan penting dalam mendukung proses pengembangan VR. Rincian setiap komponen akan dijelaskan pada subbab berikut.

3.4.1 Perangkat Lunak yang Digunakan

Selama magang, berbagai perangkat lunak dimanfaatkan untuk mendukung pengembangan aplikasi Virtual Reality sekaligus kolaborasi tim. Perangkat lunak utama yang digunakan meliputi alat pengembangan, manajemen versi, dan platform komunikasi.

Tabel 3.2. Daftar Perangkat Lunak dan spesifikasinya (versi)

Nama Perangkat Lunak	Spesifikasi (Versi)
Unity	*Game engine* utama untuk membangun simulasi VR; mendukung interaksi fisika, XR Interaction Toolkit, dan integrasi model 3D. Versi 2022.3.43f1 LTS
SideQuest	Digunakan untuk *sideloading* / pemasangan aplikasi VR ke perangkat Meta Quest 3
Draw.io	Dipakai untuk menyusun diagram alur simulasi, mencakup urutan interaksi pengguna dan transisi antar tahap
Notion	Repositori dokumentasi proyek: ide, catatan rapat, tugas mingguan, serta referensi teknis
SharePoint	Media penyimpanan dokumen dan aset proyek yang terstruktur serta kolaboratif
GitHub	Sistem *version control* dan kolaborasi kode melalui *branch*, *pull request*, dan *code review*
Microsoft Teams	Platform komunikasi dan rapat daring dengan tim internal serta klien klien
WhatsApp	Kanal komunikasi informal dan koordinasi cepat antar anggota tim
Asana	Alat pelacakan tugas (*task tracking*) dan pemantauan progres

3.4.2 Perangkat Keras yang Digunakan

Untuk menunjang pengembangan aplikasi Virtual Reality, perangkat keras yang dipakai harus mampu menangani kebutuhan rendering 3D dan kompatibel dengan ekosistem XR. Perangkat keras inti mencakup komputer pengembangan, perangkat HMD, serta periferal pendukung lainnya.

Tabel 3.3. Daftar perangkat keras dan spesifikasinya

Nama Perangkat	Spesifikasi
Laptop Pengembangan	Perangkat utama untuk menjalankan Unity Editor, menulis kode, dan mengelola aset proyek. Spesifikasi ringkas: Intel Core i3, RAM 12 GB. Memadai untuk kebutuhan pengembangan dan uji VR dasar
PC Pengembangan	Workstation untuk Unity Editor, *coding*, dan manajemen aset skala lebih besar. Spesifikasi ringkas: Intel Core i7, RAM 12 GB. Sesuai untuk pengembangan dan pengujian VR yang lebih intensif
Head-Mounted Display (HMD) dan Controller	Meta Quest 3; digunakan untuk pengujian aplikasi VR secara *real-time* dan interaksi imersif dengan lingkungan virtual

3.4.3 Kerangka Kerja dan Library yang Digunakan

Pengembangan aplikasi Virtual Reality juga memanfaatkan sejumlah framework dan library agar sistem yang dibangun bersifat modular, interaktif, dan mudah dipelihara. Kerangka kerja serta pustaka kunci yang digunakan akan dipaparkan pada bagian selanjutnya.

Tabel 3.4. Daftar framework / library dan spesifikasinya (versi)

Nama Framework / Library	Spesifikasi (Versi)
XR Interaction Toolkit	3.0.8
Unity Input System	1.14.0
Mock HMD (XR Simulator)	1.2.0-preview.1
GitHub Desktop	3.4.19
Draw.io (Desktop)	27.0.5
Unity Asset Store Package	1.6.250208

3.5 Proses Pelaksanaan Magang

3.5.1 Pengembangan dan Tujuan

Fase pengembangan ini difokuskan untuk menuntaskan skenario *Clean-Up* dan *Set-Up* semua *scene*, menyelaraskan perilaku interaksi, serta menyiapkan aplikasi untuk *demo*/UAT internal dan sesi uji bersama klien. Pada periode sebelumnya, implementasi pada mesin yang disimulasikan baru mencapai sekitar 10–20% dan sejumlah *scene* lain belum siap. Oleh karena itu, pekerjaan utama pada fase ini meliputi penyelesaian *scene* yang belum lengkap, penataan ulang arsitektur interaksi, serta peningkatan pengalaman pengguna.

Perubahan inti dilakukan pada arsitektur skrip interaksi: dari pendekatan berbasis objek (*object-based*) dimigrasikan ke pendekatan berbasis *socket*. Pergeseran ini membuat logika orientasi, pemasangan, dan pelepasan komponen menjadi lebih konsisten di seluruh *scene*, mengurangi duplikasi kode, dan memudahkan *reuse*. Dari sisi pengguna, pola interaksi tetap sama; perbedaannya terutama terjadi pada proses di belakang layar sehingga validasi langkah menjadi lebih stabil.

Untuk menjamin kelengkapan prosedur, ditambahkan skrip *checker* yang memverifikasi kondisi terpasang/terlepas (attach/detach) secara menyeluruh. Mekanisme ini mendukung penilaian lulus/gagal per langkah dan mempermudah pelacakan kesalahan saat pelatihan. Di sisi antarmuka, dilakukan perbaikan menu instruksi agar lebih jelas dan terstruktur, termasuk penambahan indikator progres dan umpan balik visual. Selain itu, diperkenalkan *exam mode* sebagai mode evaluasi dengan panduan minimal (tanpa *hint*), yang dapat dilengkapi pengatur waktu.

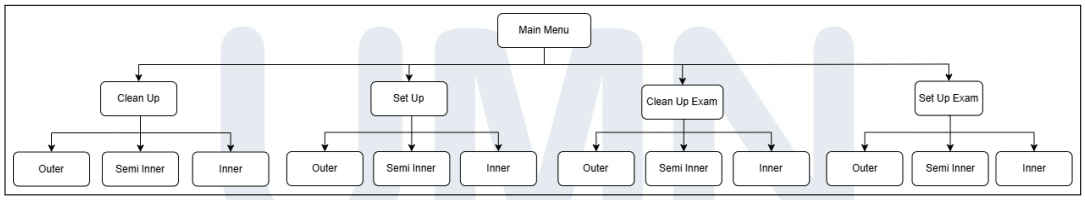
Secara ringkas, tujuan fase ini adalah:

1. Menyelesaikan *scene* yang belum siap dan menyeragamkan interaksi berbasis *socket* di seluruh skenario,
2. Meningkatkan reliabilitas validasi prosedur melalui *checker* attach/detach dan pengendali urutan langkah,
3. Memperbaiki UI/UX instruksi (menu tutorial, progres, *feedback*) dan menambahkan *exam mode* untuk evaluasi,
4. Menyiapkan *build* berkala ke perangkat HMD guna *testing* internal dan sesi uji bersama klien.

3.5.2 Development

Implementasi teknis pada fase ini mengikuti pendekatan iteratif-modular dengan penekanan pada konsistensi interaksi, stabilitas validasi, dan kemudahan pemeliharaan kode. Arsitektur pengembangan dirancang dengan logika pemasangan/pelepasan komponen dikelola oleh skrip berbasis *socket*, validasi progres ditangani oleh skrip *checker* yang independen, dan pengendali menu beroperasi terpisah dari mekanisme simulasi inti. Pendekatan ini mempermudah pengujian per modul, mengurangi risiko efek samping saat perbaikan, serta memungkinkan *reuse* komponen di berbagai skenario tanpa duplikasi kode yang signifikan.

Transisi dari arsitektur awal berbasis objek (*object-based*) ke arsitektur berbasis *socket* menjadi tonggak utama pada fase ini. Pergeseran paradigma ini memungkinkan pemusatan logika orientasi, penempatan, dan validasi pada sisi *socket interactor*, sehingga perilaku komponen menjadi lebih seragam dan dapat diprediksi di seluruh *scene*. Sistem *checker* dikembangkan sebagai lapisan validasi yang fleksibel mampu memantau kondisi terpasang (*attached*), terlepas (*detached*), atau dituntaskan (*fully hammered*) dengan mekanisme berbasis *counter* dan *event* yang responsif terhadap perubahan status komponen secara *real-time*.



Gambar 3.1. Sitemap Alur Navigasi dan Pembagian Scene Aplikasi

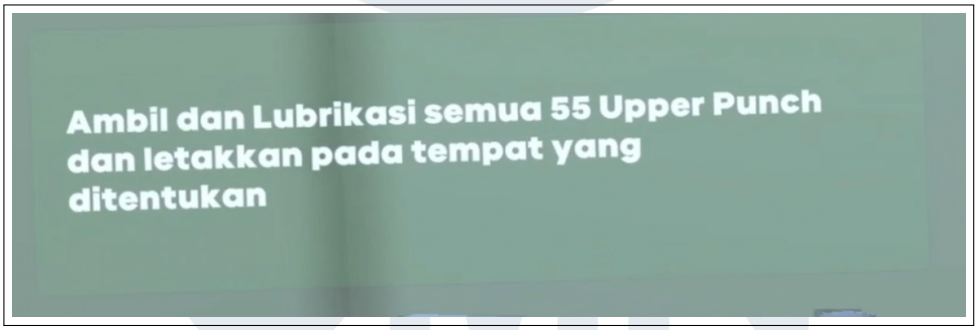
Gambar 3.1 menggambarkan alur navigasi aplikasi yang dibangun, memperlihatkan pembagian scene berdasarkan kategori seperti Clean Up, Set Up, serta Exam Mode. Setiap kategori memiliki cabang lebih lanjut, seperti Outer, Semi Inner, dan Inner, dengan masing-masing memiliki skenario yang berbeda, baik untuk proses *clean up*, *set up*, maupun *exam*. Diagram ini menunjukkan bagaimana pengguna dapat menavigasi antar scene, memilih skenario yang diinginkan, dan memandu mereka melalui tahapan simulasi secara terstruktur.

Untuk menjaga keterbacaan dan kemudahan navigasi laporan, uraian implementasi teknis dibagi ke dalam empat section utama, yaitu *UI/UX Instruksi* menguraikan sistem navigasi menu hierarki dan kanvas instruksi bertahap, *Set*

Up and Clean Up Flow menjelaskan mekanisme pemasangan dan pembongkaran komponen dan logika skrip pendukungnya, *Exam Mode* menjelaskan pendekatan berbasis konfigurasi untuk mode evaluasi tanpa panduan visual, dan *Hasil Implementasi* menyajikan tangkapan layar serta dokumentasi visual dari sistem yang telah dibangun. Setiap section dilengkapi dengan potongan kode, diagram konfigurasi Inspector, serta penjelasan alur kerja untuk memberikan gambaran menyeluruh tentang keputusan teknis dan implementasinya.

A UI/UX Instruksi

Menu tutorial seperti Gambar 3.2 ini dibangun dengan menggunakan sistem navigasi berbasis kanvas (`Canvas`) yang dapat diganti-ganti sesuai dengan langkah tertentu. Sistem ini memungkinkan pengguna untuk melihat tutorial yang berbeda, dengan setiap tutorial disajikan dalam bentuk `Canvas` terpisah yang dapat diaktifkan secara bergantian. Dengan menggunakan pendekatan ini, pengguna dapat lebih mudah mengikuti tutorial secara berurutan atau memilih tutorial tertentu berdasarkan kebutuhan mereka.



Gambar 3.2. Tampilan tutorial menu

Kode 3.1 digunakan fungsi `LookAt()` untuk memastikan bahwa menu selalu menghadap posisi kepala pengguna. Fungsi ini memutar menu sehingga menghadap ke posisi kepala pengguna dengan mengabaikan perbedaan ketinggian (`y-axis`). Setelah posisi menu diatur, kode ini juga memutar menu untuk memastikan agar menu selalu menghadap dengan benar ke pengguna. `transform.forward` dibalik untuk memastikan menu tidak terbalik, menjaga orientasi yang benar selama penggunaan.

```
menu.transform.LookAt(new Vector3(head.position.x, menu.transform.
    position.y, head.position.z));
```

```
menu.transform.forward *= -1;
```

Kode 3.1: Menjaga menu selalu menghadap pengguna

Selanjutnya, Kode 3.2 digunakan untuk mengaktifkan kanvas tutorial tertentu, terdapat fungsi `ActivateCanvas()` yang menerima parameter indeks. Fungsi ini pertama-tama melakukan validasi untuk memastikan bahwa indeks yang diberikan valid dan sesuai dengan jumlah `Canvas` yang ada. Kemudian, semua `Canvas` dinonaktifkan terlebih dahulu untuk memastikan hanya satu tutorial yang ditampilkan pada waktu tertentu. Setelah itu, `Canvas` yang sesuai dengan indeks yang diberikan akan diaktifkan, dan posisinya disesuaikan dengan posisi kepala pengguna. Fungsi ini juga memastikan bahwa suara tutorial diputar menggunakan `AudioSource`, menambah pengalaman interaktif dalam tutorial.

```
public void ActivateCanvas(int index)
{
    if (index < 0 || index >= canvases.Count) return; // Validasi indeks

    // Menonaktifkan semua canvas
    foreach (var canvas in canvases)
    {
        canvas.SetActive(false);
    }

    // Mengaktifkan canvas sesuai indeks
    canvases[index].SetActive(true);
    menu = canvases[index];
    menu.transform.position = head.position + new Vector3(head.forward.x, 0, head.forward.z).normalized * spawnDistance;
    currentCanvasIndex = index;

    // Memutar suara
    source = GetComponent<AudioSource>();
    source.PlayOneShot(clip);
}
```

Kode 3.2: Mengaktifkan kanvas tutorial tertentu

Untuk memungkinkan navigasi antar tutorial, terdapat dua fungsi tambahan, yaitu `ActivateNextCanvas()` dan `ActivatePreviousCanvas()`. Kedua fungsi ini memungkinkan pengguna untuk berpindah ke tutorial berikutnya atau sebelumnya dengan menghitung indeks yang sesuai. Fungsi

`ActivateNextCanvas()` akan menghitung indeks tutorial berikutnya dengan menambahkan satu pada indeks saat ini dan menggunakan operator modulus untuk memastikan bahwa indeks tetap valid meskipun sudah mencapai akhir daftar tutorial. Begitu juga dengan `ActivatePreviousCanvas()`, yang menghitung indeks tutorial sebelumnya dan memastikan alur navigasi tetap terjaga dengan benar.

```
public void ActivateNextCanvas()
{
    int nextIndex = (currentCanvasIndex + 1) % canvases.Count;
    ActivateCanvas(nextIndex);
}

public void ActivatePreviousCanvas()
{
    int previousIndex = (currentCanvasIndex - 1 + canvases.Count)
   % canvases.Count;
    ActivateCanvas(previousIndex);
}
```

Kode 3.3: Navigasi ke tutorial berikutnya dan sebelumnya

Di bagian Kode 3.4, fungsi `ActivateNextCanvas()` dipanggil untuk berpindah ke tutorial berikutnya. Fungsi ini dapat dipanggil dari script lain untuk memudahkan interaksi pengguna dengan menu tutorial, memungkinkan kontrol penuh atas alur navigasi berdasarkan input atau interaksi dari pengguna.

```
if (menuController != null)
{
    menuController.ActivateNextCanvas();
}
```

Kode 3.4: Panggilan fungsi untuk navigasi menu

Untuk pengaturan di `Inspector` seperti pada Gambar 3.3, elemen-elemen berikut perlu diperhatikan. Pertama, objek `Menu` yang mengandung semua elemen dan fungsionalitas tutorial harus terhubung dengan komponen yang sesuai di dalam `Inspector`. Kemudian, daftar `Canvas` yang akan digunakan harus diatur dalam array `canvases`, di mana setiap elemen array ini akan mewakili tutorial yang berbeda yang bisa diaktifkan.

Gambar 3.3. Tampilan settingan inspector menu

Dengan pendekatan ini, Anda dapat mengganti tutorial yang sedang ditampilkan berdasarkan interaksi pengguna atau alur yang diinginkan.

B Set Up and Clean Up Flow

Skrip Kode 3.5 berperan sebagai *checker* berbasis hitungan (*counter*). Prinsip kerjanya adalah memantau sekumpulan objek target (`objectsToCheck`) dan menghitung berapa yang sudah memenuhi kondisi "terpasang" (*attached*). Jika jumlah terpenuhi mencapai ambang yang ditentukan (`objectsToCheckCount`), maka langkah dinyatakan lulus, sistem mengaktifkan objek lanjutan, membersihkan *highlight* pada objek saat ini, serta menyorot objek yang menjadi sasaran berikutnya. Pendekatan ini memudahkan validasi proses yang mengharuskan beberapa subkomponen selesai sebelum beralih ke tahap selanjutnya.

Pemeriksaan utama dilakukan pada `CheckObjectAttachments`. Fungsi ini menghitung berapa objek yang sudah "terpasang", menyimpan daftar yang belum, lalu mengeksekusi transisi ketika ambang terpenuhi. Setelah kondisi tercapai, pemeriksaan periodik dihentikan menggunakan `CancelInvoke` agar tidak membebani sistem.

```
    foreach (var obj in objectsToCheck)
    {
        if (obj == null) continue;
        if (IsObjectAttached(obj)) attachedFlagCount++;
        else notAttachedNames.Add(obj.name);
```

```
    }
```

Kode 3.5: Logika pemeriksaan dan transisi langkah

Saat ambang terpenuhi, daftar `objectsToActivate` dihidupkan untuk membuka langkah berikutnya (misalnya *socket* tahap berikut, alat bantu, atau indikator). Selain mengaktifkan target baru, *highlight* pada objek saat ini dibersihkan (`objectsCurrent`) agar panduan visual tetap fokus. Sementara itu, `objectsNext` disorot untuk mengarahkan pengguna ke sasaran berikutnya.

Untuk menjaga konsistensi visual pada objek kompleks yang memiliki beberapa `Renderer` turunan, disediakan utilitas pengganti material secara menyeluruh. Dengan cara ini, proses reset dan *highlight* tidak meninggalkan bagian mesh yang terlewat.

```
    if (go == null || mat == null) return;
    var renderers = go.GetComponentsInChildren<Renderer>(true);
    foreach (var r in renderers) r.material = mat;
```

Kode 3.6: Setter material untuk semua renderer turunan

Status "terpasang" ditentukan oleh komponen domain yang relevan pada objek (contoh: `ScInsideTurretAssembly`) melalui properti `IsAttached`. Pola ini memisahkan logika *checker* dari detail implementasi setiap komponen, sehingga mudah diperluas dengan menambahkan blok pemeriksaan baru sesuai kebutuhan.

```
    var insideTurretAssembly = go.GetComponent<
    ScInsideTurretAssembly>();
    if (insideTurretAssembly != null && insideTurretAssembly.
    IsAttached) return true;
```

Kode 3.7: Pemeriksaan status attached berbasis komponen domain

Dengan Kode 3.5, validasi langkah yang melibatkan beberapa subkomponen dapat ditangani secara ringkas dan konsisten. Aktivasi objek lanjutan, pembersihan *highlight*, serta penyorotan target berikutnya dilakukan otomatis saat ambang terpenuhi, sehingga alur *Clean Up* menjadi lebih terstruktur dan mudah diikuti.

Skrip Kode 3.8 dipasang pada *GameObject* yang memiliki komponen `XRSocketInteractor`. Tujuannya adalah memusatkan logika pemasangan/pelepasan pada sisi *socket* sehingga perilaku langkah menjadi seragam di berbagai *scene*. Saat sebuah objek menempel (`selectEntered`), skrip ini mengatur arah gerak sekrup, mengelola *highlight*, mengaktifkan objek lanjutan, serta menonaktifkan objek yang sudah tidak relevan.

Ketika objek menempel, arah gerak sekrup diseragamkan berdasarkan konfigurasi pada *socket*. Dengan begitu, komponen seperti `ScScrewDetach`, akan menerima vektor arah yang sama tanpa perlu mengatur setiap objek satu per satu. Kode 3.8 di bawah menunjukkan penggantian properti `movementDirection` saat `selectEntered` terjadi.

```
    var comp = args.interactableObject as Component;
    if (comp != null)
    {
        var go = comp.gameObject;

        var detach = go.GetComponentInParent<ScScrewDetach>();
        if (detach != null)
            detach.movementDirection = screwMovementDirection;
    }
```

Kode 3.8: Override arah gerak sekrup pada saat attach

Untuk memperjelas konfigurasi arah gerak, Gambar 3.4 merupakan tangkapan layar Inspector yang menampilkan nilai `screwMovementDirection` pada skrip Kode 3.8. Gambar 3.4 membantu menjelaskan sumber arah yang digunakan oleh semua skrip sekrup saat objek menempel pada *socket*.

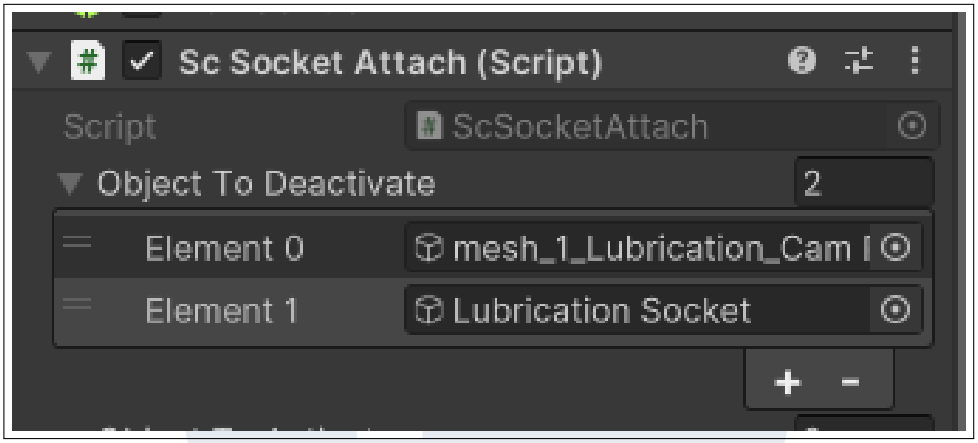
Gambar 3.4. Pengaturan `screwMovementDirection` pada `ScSocketAttach` di Unity

Setelah arah gerak ditentukan, skrip menentukan objek mana yang perlu disorot sebagai langkah berikutnya. Jika objek ber-tag `Nail` dan sudah dibaut, *highlight* dilewati agar pengguna tidak diarahkan ke langkah yang telah selesai. Jika objek adalah placeholder, sistem mengaktifkannya terlebih dahulu untuk menandai titik/area kerja. Potongan berikut menggambarkan logika tersebut.

```
    {
        var r = obj.GetComponent<Renderer>();
        if (r != null) r.material = highlightMaterial;
    }
```

Kode 3.9: Penentuan highlight dan aktivasi objek berikutnya

Konfigurasi `objectToActivate` dipakai untuk mengaktifkan objek lanjutan setelah langkah saat ini dinyatakan berhasil. Gambar 3.5 menampilkan daftar objek yang diaktifkan, seperti object, atau *socket* berikutnya.



Gambar 3.5. Contoh konfigurasi `objectToActivate` pada `ScSocketAttach`

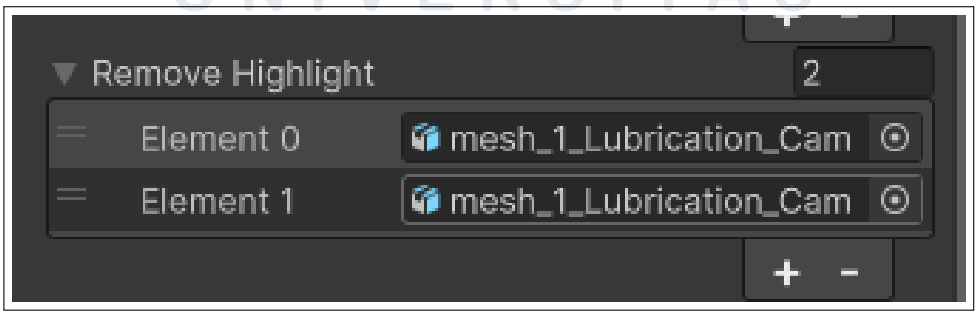
Selain memberi *highlight* pada langkah berikutnya, skrip juga menghapus *highlight* dari objek yang sudah tidak relevan agar panduan visual tetap bersih dan fokus. Kode 3.10 mengembalikan material ke `defaultMaterial`.

```

    var r = obj.GetComponent<Renderer>();
    if (r != null) r.material = defaultMaterial;
```

Kode 3.10: Menghapus highlight dari objek sebelumnya

Untuk memperjelas bagian pembersihan *highlight*, pada Gambar 3.6 berisi daftar `removeHighlight` di Inspector. Daftar ini berguna untuk menelusuri objek mana saja yang dibersihkan pasca langkah tertentu.



Gambar 3.6. Contoh konfigurasi `removeHighlight` pada `ScSocketAttach`

Penonaktifan objek tertentu dilakukan setelah jeda singkat agar proses *snap*

oleh *socket* selesai terlebih dahulu. Strategi ini mengurangi kemungkinan kondisi balapan yang dapat menyebabkan perilaku tidak stabil saat transisi status objek.

```
1
2       yield return new WaitForSeconds(0.2f);
3       if (obj == null) continue;
4       obj.SetActive(false);
```

Kode 3.11: Penundaan singkat sebelum menonaktifkan objek

Agar sistem tidak kembali menyorot langkah yang sudah selesai, skrip menyediakan fungsi pemeriksaan untuk objek bertipe `Nail`. Fungsi ini mengecek komponen terkait pada objek dan mengembalikan status dibaut berdasarkan properti `IsFullyHammered`.

```
1
2       var screwAttach = nailObject.GetComponent<ScScrewAttach>()
3                         ?? nailObject.transform.parent?.GetComponent<
    ScScrewAttach>();
4
5       return (screwAttach != null && screwAttach.IsFullyHammered);
```

Kode 3.12: Pemeriksaan apakah objek bertipe Nail sudah tuntas

Dengan pemusatan logika di *socket*, arah gerak, *highlight*, serta aktivasi/deaktivasi objek dapat dikelola secara konsisten di seluruh *scene*. Pendekatan ini mengurangi duplikasi skrip di objek, mempermudah perawatan, dan meningkatkan stabilitas validasi langkah pada alur *Clean Up*.

Kemudian, skrip Kode 3.13 berfungsi sebagai pemeriksa kemajuan khusus untuk langkah yang melibatkan sekrup. Berbeda dari yang sebelumnya yang hanya menghitung objek berstatus terpasang, skrip ini membedakan dua kondisi validasi: jumlah sekrup yang sudah berstatus terpasang (*attached*) dan jumlah sekrup yang benar-benar dituntaskan (*fully hammered*). Peralihan ke langkah berikutnya baru dilakukan setelah kedua kondisi tersebut terpenuhi, sehingga memastikan urutan kerja benar sekaligus tuntas secara mekanis.

Pemeriksaan utama dilakukan pada Kode 3.13. Fungsi ini menghitung berapa sekrup yang berstatus terpasang dengan membaca `AttachedFlag` dari setiap tipe skrip sekrup. Jika jumlah terpasang memenuhi ambang, skrip mereset material sekrup yang belum terpasang agar panduan visual tidak menyesatkan. Setelah itu, ketika jumlah sekrup yang benar-benar dituntaskan mencapai ambang, sistem mengaktifkan objek lanjutan, membersihkan material pada objek saat ini, lalu

menyorot target berikutnya. Pemeriksaan periodik dihentikan ketika kedua kondisi validasi telah terpenuhi.

```
    if (!attachedPassed && attachedFlagCount >= screwsToCheck)
    {
        attachedPassed = true;
    }

    if (!hammeredPassed && fullyHammeredCount >= screwsToCheck)
    {
        hammeredPassed = true;
    }

    if (hammeredPassed && attachedPassed)
        CancelInvoke(nameof(CheckSelectedScrews));
```

Kode 3.13: Dua tahap validasi: attached dan fully hammered

Penambahan hitungan sekrup yang benar-benar dituntaskan dilakukan melalui pemanggilan fungsi `IncrementFullyHammeredCount` oleh skrip sekrup terkait saat kondisi tuntas tercapai. Pola berbasis event ini memastikan umpan balik cepat tanpa menunggu jeda pemeriksaan periodik.

```
    if (insideScrewsToMonitor.Contains(screw))
    {
        fullyHammeredCount++;
        CheckSelectedScrews();
    }
```

Kode 3.14: Kenaikan jumlah sekrup yang fully hammered

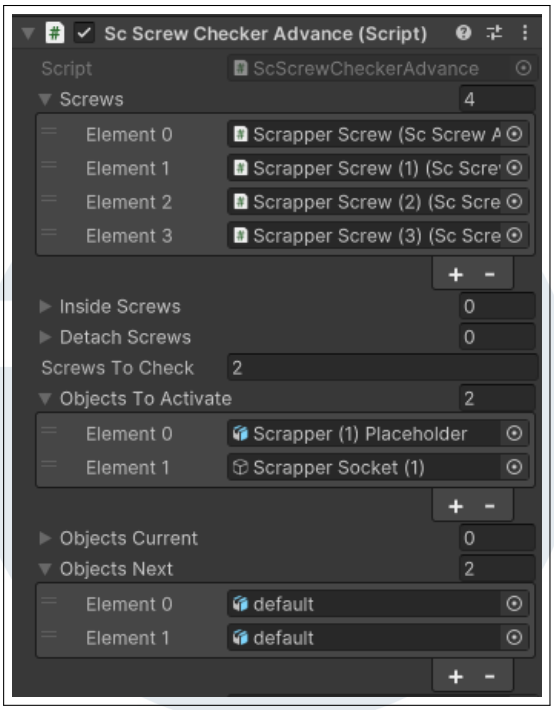
Untuk menjaga konsistensi penggantian material pada objek yang memiliki beberapa `Renderer`, disediakan utilitas pengaturan material secara menyeluruh di seluruh turunan. Dengan cara ini, pembersihan dan *highlight* tidak meninggalkan bagian mesh yang terlewat.

```
    var renderers = go.GetComponentsInChildren<Renderer>(true);
    foreach (var r in renderers) r.material = mat;
```

Kode 3.15: Penggantian material seluruh renderer turunan

Agar transisi langkah dan penandaan visual terdokumentasi jelas, Gambar 3.7 menampilkan pengaturan `objectsToActivate`, `objectsCurrent`, dan `objectsNext`. Pengaturan ini menunjukkan objek mana yang diaktifkan saat

validasi terjadi, mana yang dibersihkan *highlight*-nya, serta mana yang disorot sebagai target berikutnya.



Gambar 3.7. Pengaturan `objectsToActivate`, `objectsCurrent`, dan `objectsNext`

Pembacaan status terpasang dilakukan secara seragam melalui fungsi Kode 3.16. Fungsi ini menyatukan akses ke `AttachedFlag` dari berbagai tipe skrip sekrup sehingga logika pemeriksaan tetap ringkas dan tidak bergantung pada implementasi masing-masing komponen.

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

Kode 3.16: Pembacaan flag terpasang yang diseragamkan

Dengan dua tahapan validasi dan pemicu berbasis event, pengecekan memastikan bahwa langkah yang melibatkan sekrup tidak hanya terpasang dengan benar, tetapi juga dituntaskan secara mekanis sebelum pengguna diarahkan ke tahapan berikutnya.

Selanjutnya, skrip Kode 3.17 menggabungkan peran *handler* dan *checker* dalam satu modul. Saat sebuah *punch* ditempatkan pada `XRSocketInteractor` bertag tertentu (`requiredSocketTag`), skrip ini menghitung progres penempatan, membekukan objek agar tidak bisa diinteraksikan ulang, dan bila perlu

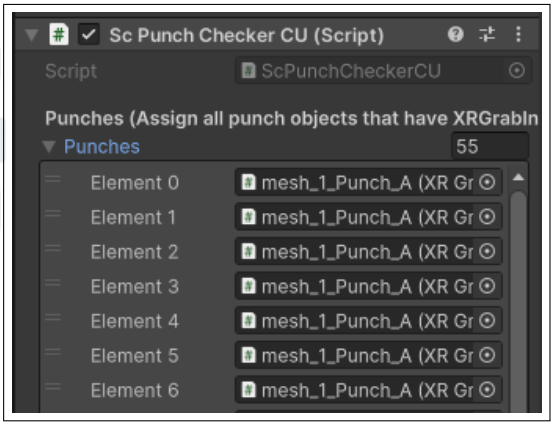
menonaktifkan *socket* penerima untuk mencegah pemakaian ganda. Setelah ambang jumlah penempatan tercapai, skrip memicu transisi langkah berikutnya dengan membersihkan *highlight* saat ini, menyorot target berikut, serta mengaktifkan/menonaktifkan objek pendukung sesuai konfigurasi.

Pada saat *punch* menempel, langkah pertama adalah memverifikasi bahwa interaktor yang menerima adalah `XRSocketInteractor` dengan tag yang sesuai. Jika valid, sistem mencegah hit ganda pada objek yang sama, memutar *SFX* singkat, lalu membekukan fisika/kolider agar *punch* tidak bergerak atau berinteraksi lagi. Setelah itu, alur dilanjutkan dan kondisi kelulusan dievaluasi.

```
1
2     if (args.interactorObject is XRSocketInteractor xrSocket &&
3         xrSocket != null && xrSocket.CompareTag(requiredSocketTag)
    )
4     {
5         GameObject punchGO = args.interactableObject.transform.
    gameObject;
6         if (!placedPunches.Contains(punchGO))
7         {
8             Kode lainnya...
9         }
10    }
```

Kode 3.17: Penanganan penempatan punch pada socket yang valid

Agar konfigurasi mudah dicek, daftar `punches` yang dipantau, nilai ambang `maxCount`, dan tag target `requiredSocketTag` ditata pada Inspector. Gambar 3.8 menampilkan pengaturan tersebut.



Gambar 3.8. Konfigurasi `punches`, `maxCount`, dan `requiredSocketTag` pada `ScPunchCheckerCU`

Setelah *snap* stabil, seleksi XRI ditutup secara aman, objek dibuat *inert* (opsional menghapus `XRGrabInteractable`/`Rigidbody`/`Collider`), *socket* penerima bisa dinonaktifkan setelah jeda singkat untuk mencegah penempatan ulang, dan objek dipindahkan ke `parentOnAttach` jika diatur. Terakhir, material objek dikembalikan ke `defaultMaterial` agar tidak tetap tersorot.

```
1
2     MakePlacedInert(punchGO);
3
4     if (autoDisableSocketOnPlace)
5         StartCoroutine(DisableSocketAfterDelay(socket.gameObject,
    socketDisableDelay));
```

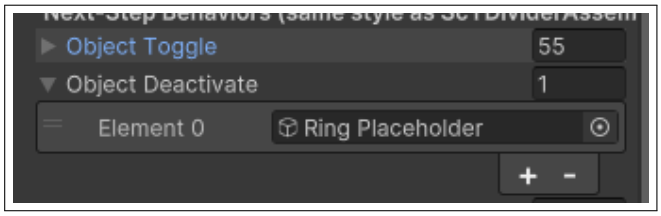
Kode 3.18: Membuat punch inert dan menonaktifkan socket penerima

Transisi ke langkah berikutnya tidak terjadi seketika setiap kali ada satu penempatan; skrip menunggu hingga jumlah penempatan mencapai ambang `maxCount` atau seluruh *punch* yang ditugaskan sudah ditempatkan. Setelah syarat terpenuhi, ada jeda `nextStepDelay` untuk memberi waktu sistem menyelesaikan *snap*/render, barulah perilaku "langkah berikutnya" dijalankan.

```
1
2     if (reachedMax || placedAllAssigned)
3     {
4         nextStepDone = true;
5         StartCoroutine(RunNextStepBehaviorsDelayed());
6     }
```

Kode 3.19: Trigger langkah berikutnya setelah penempatan terpenuhi

Perilaku "langkah berikutnya" meliputi pembersihan *highlight*/placeholder yang sedang aktif, penyorotan target berikutnya, aktivasi objek pendukung, penonaktifan objek yang tidak lagi relevan, dan bila tersedia melanjutkan kanvas instruksi melalui `menuController`. Gambar 3.9 menunjukkan contoh pemetaan `currentHighlight`/`nextHighlight`/`objectToggle`/`objectDeactivate`.



Gambar 3.9. Pengaturan `currentHighlight`, `nextHighlight`, `objectToggle`, dan `objectDeactivate`

Untuk mengantisipasi kasus *detach* dari *socket* yang bukan target (misalnya salah tempat), skrip dapat mengaktifkan kembali kolisi terhadap seluruh `Collider` di *scene* kecuali pengecualian tertentu agar lingkungan uji kembali stabil.

```
1
2       if (enableCollisionsOnWrongDetach)
3           ReenableCollisionsForPunchExcept(punchGO,
  reenableExceptionObject, reenableExceptionTag);
```

Kode 3.20: Re-enable collision ketika detach dari socket yang tidak valid

Dengan pemusatan logika pada satu skrip, `ScPunchCheckerCU` menyederhanakan alur Clean Up untuk kasus penempatan massal: validasi penempatan pada *socket* bertag, pembekuan objek agar tidak diutak-atik ulang, penonaktifan *socket* penerima, serta transisi langkah yang tertata setelah ambang penempatan tercapai.

Kemudian Skrip Kode 3.21 digunakan pada alur *Clean Up* untuk memvalidasi langkah yang menuntut objek berada pada kondisi "terlepas". Berbeda dengan checker pemasangan, modul ini memantau sekumpulan target dan menghitung berapa yang sudah memenuhi status lepas sesuai logika domain. Ketika jumlahnya mencapai ambang `objectsToCheckCount`, sistem melanjutkan ke langkah berikut dengan mengaktifkan objek lanjutan, membersihkan sorotan pada objek saat ini, serta menyorot dan menyiapkan objek berikut (`objectsNext`) agar siap diinteraksikan.

Pemeriksaan utama dilakukan pada Kode 3.21. Fungsi ini menghitung jumlah target yang telah berstatus lepas. Begitu ambang terpenuhi, skrip menjalankan transisi: mereset material pada target yang belum lepas agar tidak tersisa sorotan menyesatkan, mengaktifkan objek lanjutan, membersihkan sorotan saat ini, lalu menyorot sekaligus menyiapkan objek berikutnya menjadi dapat dipegang, dengan menambahkan komponen `Rigidbody` dan `XRGrabInteractable` bila diperlukan.

```
1     if (!attachPassed && attachedFlagCount >= objectsToCheckCount)
2     {
3         attachPassed = true;
4
5         Kode lainnya..
6     }
7
8     if (attachPassed)
```

```
9           CancelInvoke(nameof(CheckObjectAttachments));
```

Kode 3.21: Logika hitung detach dan transisi ke langkah berikut

Penentuan status "terlepas" diseragamkan melalui komponen domain Kode 3.22 . Dengan cara ini, checker tidak bergantung pada detail implementasi tiap objek, cukup membaca satu indikator status dari skrip domain.

```
1       var screwDetach = go.GetComponent<ScScrewDetach>();
2       if (screwDetach != null && screwDetach.IsAttached)
3           return true;
4   }
```

Kode 3.22: Pembacaan status detach berbasis komponen domain

Dengan pemusatan logika pada kondisi lepas, pengecekan memastikan alur *Clean Up* berjalan terstruktur: status dilepas divalidasi secara konsisten, objek lanjutan diaktifkan tepat waktu, dan target berikut disiapkan agar tahapan selanjutnya dapat berlangsung tanpa hambatan.

C Exam Mode

Mode evaluasi (*Exam Mode*) disediakan sebagai mekanisme penilaian untuk mengukur tingkat pemahaman pengguna terhadap prosedur *Clean-Up* dan *Set-Up* tanpa bantuan visual atau petunjuk langkah. Berbeda dengan mode latihan (*Training Mode*) yang menyertakan *highlight* pada objek target berikutnya, tutorial interaktif, serta menu bantuan, mode evaluasi menghilangkan seluruh elemen pemandu tersebut sehingga pengguna harus mengandalkan pemahaman prosedural yang telah dikuasai sebelumnya.

Implementasi teknis *Exam Mode* tidak memerlukan skrip baru atau logika validasi yang berbeda. Seluruh mekanisme interaksi, *checker*, dan transisi langkah tetap menggunakan arsitektur yang sama dengan mode latihan. Perbedaan utama terletak pada konfigurasi Inspector di Unity, di mana elemen-elemen pemandu secara sengaja tidak diberikan referensi atau dibiarkan kosong.

Dalam *Exam Mode*, parameter `nextHighlight` ini dibiarkan kosong sehingga sistem tidak memberikan petunjuk visual tentang objek mana yang harus diinteraksikan berikutnya. Namun, parameter seperti `objectsToActivate` tetap diisi untuk memastikan *socket* atau objek pendukung berikutnya tersedia dan dapat diakses, menjaga kelancaran alur prosedur meskipun tanpa panduan visual.

Selain penghilangan *highlight*, mode evaluasi juga menonaktifkan seluruh elemen tutorial dan menu bantuan. Kanvas instruksi bertahap yang biasanya

menampilkan deskripsi langkah, ilustrasi, atau tombol *hint* tidak dimunculkan dalam skenario evaluasi. Pengguna hanya disediakan lingkungan kerja dengan komponen-komponen yang perlu diinteraksikan, tanpa petunjuk tambahan mengenai urutan atau cara pengerjaannya.

Pendekatan ini memungkinkan evaluator atau instruktur untuk mengukur sejauh mana pengguna telah menginternalisasi prosedur kerja. Validasi langkah tetap berjalan di latar belakang menggunakan *checker* yang sama, sehingga sistem dapat mencatat progres, mendeteksi kesalahan, dan menghitung waktu penyelesaian. Data ini kemudian dapat digunakan untuk menghasilkan laporan kinerja atau skor evaluasi berdasarkan kriteria yang telah ditetapkan, seperti ketepatan urutan, jumlah kesalahan, dan durasi penyelesaian.

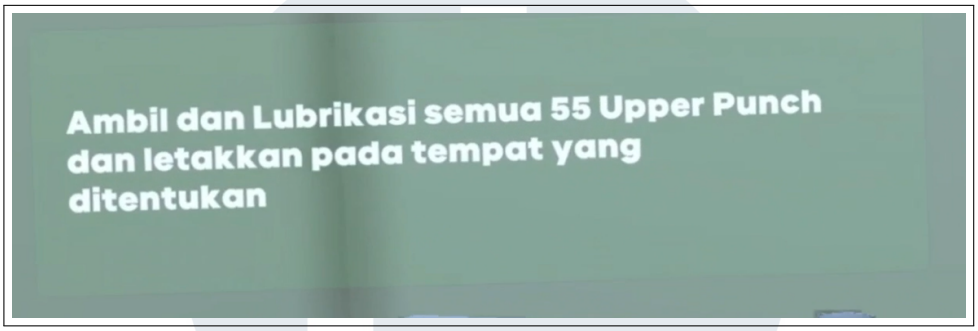
Dengan memanfaatkan konfigurasi berbasis Inspector tanpa perubahan kode, pengembangan dan pemeliharaan *Exam Mode* menjadi lebih efisien. Tim dapat dengan mudah mengonversi skenario latihan menjadi skenario evaluasi hanya dengan menyesuaikan parameter visual di Unity Editor, tanpa harus menduplikasi atau memodifikasi logika skrip yang sudah stabil.

D Hasil Implementasi

Hasil pengembangan fase ini menghasilkan aplikasi VR yang fungsional dengan cakupan skenario *Clean-Up* dan *Set-Up* yang lengkap untuk seluruh kategori komponen (Outer, Semi Inner, dan Inner). Sistem navigasi menu hierarki empat tingkat berhasil diimplementasikan dengan konsisten, memungkinkan pengguna memilih skenario latihan secara intuitif mulai dari pemilihan mode hingga komponen spesifik. Arsitektur berbasis *socket* dan *checker* yang telah dibangun terbukti stabil dalam memvalidasi urutan langkah, mengelola transisi antarlangkah, serta memberikan umpan balik visual yang jelas kepada pengguna.

Selain navigasi utama sistem latihan, implementasi fitur *Tutorial Menu* juga menjadi bagian penting dalam fase ini. Menu tutorial dikembangkan untuk memberikan panduan langkah demi langkah mengenai prosedur tertentu sebelum pengguna memasuki simulasi utama. Sistem tutorial ini dirancang menggunakan mekanisme pergantian `Canvas` secara dinamis, sehingga setiap langkah tutorial dapat disajikan sebagai panel terpisah yang muncul satu per satu sesuai interaksi pengguna. Pendekatan ini memungkinkan penyajian petunjuk yang runtut dan mudah dipahami, sekaligus mempertahankan konsistensi visual dengan antarmuka VR secara keseluruhan.

Gambar 3.10 menunjukkan salah satu contoh tampilan tutorial yang menampilkan langkah instruksi teknis. Panel tutorial ditampilkan tepat di depan pengguna dengan jarak yang telah ditentukan untuk menjamin keterbacaan dan kenyamanan saat digunakan dalam lingkungan VR. Desain visual disusun dengan fokus pada kejelasan informasi, dengan elemen teks yang langsung menyoroti inti dari langkah yang sedang dijelaskan.



Gambar 3.10. Contoh tampilan salah satu langkah tutorial

Gambar 3.11 memperlihatkan tampilan tutorial lain yang digunakan untuk menjelaskan proses atau bagian komponen tertentu. Pada panel ini, pengguna diberikan informasi dalam bentuk visual dan teks yang saling melengkapi, sehingga dapat membantu pemahaman sebelum melakukan tindakan langsung pada simulasi VR. Dengan adanya halaman-halaman tutorial seperti ini, pengguna dapat mempersiapkan diri secara mandiri tanpa perlu pendamping eksternal.

Gambar 3.11. Contoh tampilan langkah tutorial lainnya

Setelah memilih skenario, pengguna masuk ke lingkungan simulasi VR aktual. Gambar 3.12 menunjukkan contoh sesi *Set-Up* yang sedang berlangsung, di mana pengguna berinteraksi dengan komponen mesin dalam lingkungan virtual. Objek yang menjadi target langkah saat ini ditandai dengan material *highlight* berwarna berbeda, sementara komponen lainnya tetap dalam keadaan normal.

Gambar 3.12. Contoh simulasi Set-Up dalam lingkungan VR

Implementasi yang telah diselesaikan mencakup seluruh alur kerja dari navigasi menu hingga eksekusi skenario simulasi. Sistem validasi berbasis *checker* memastikan setiap langkah diverifikasi dengan tepat, baik untuk kondisi pemasangan (*attach*), pelepasan (*detach*), maupun penyelesaian sekrup (*fully hammered*). Integrasi dengan sistem menu memungkinkan transisi yang mulus antara pemilihan skenario dan pelaksanaan latihan, sementara dukungan untuk mode evaluasi (*Exam Mode*) memberikan fleksibilitas untuk keperluan penilaian tanpa memerlukan pengembangan skrip tambahan.

3.5.3 Testing

Pengujian dilaksanakan dalam dua jalur utama dan berjalan paralel dengan proses pengembangan, sehingga perbaikan dapat dilakukan secara berkala berdasarkan umpan balik yang diterima. Jalur pertama adalah pengujian internal bersama tim pengembang (Dev) dan Project Management (PM). Jalur kedua adalah pengujian terbatas bersama pihak klien untuk memverifikasi kesesuaian skenario dengan prosedur aktual di lapangan.

A Testing internal (Dev dan PM)

Pada tahap ini, pengujian difokuskan pada verifikasi fungsi inti seperti interaksi *grab*/*release*, validasi urutan langkah, navigasi menu, serta konsistensi umpan balik visual dan SFX. Pengujian dilakukan melalui perangkat VR. Setiap temuan dicatat sebagai isu, diprioritaskan, dan diterjemahkan menjadi *action items*

untuk iterasi berikutnya. Kriteria keberhasilan mencakup tuntasnya skenario Clean-Up/Set-Up tanpa *bug*, urutan langkah yang sesuai rancangan, serta kejelasan instruksi yang mudah diikuti.

Gambar 3.13. Sesi testing internal bersama tim Dev dan PM

Gambar 3.13 memperlihatkan situasi pengujian internal yang digunakan untuk mengecek fungsi dasar, menilai stabilitas fisika dan kolisi, serta menyelaraskan antar komponen sebelum dibawa ke sesi pengujian berikutnya.

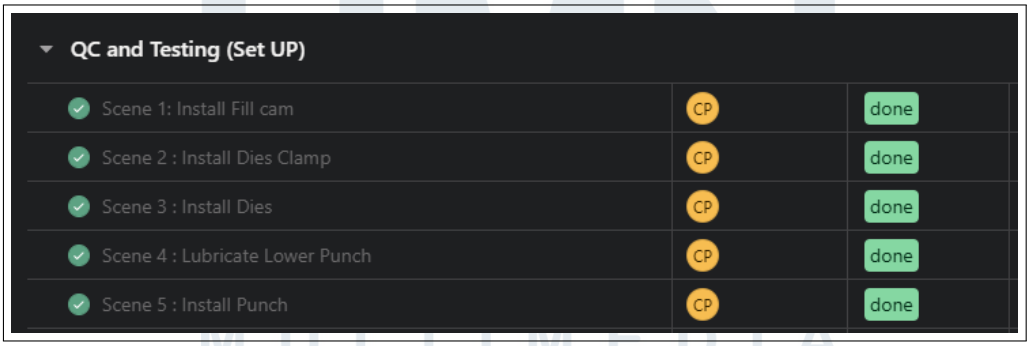
B Testing dengan Klien

Pengujian ini dilakukan untuk menilai kecocokan skenario VR dengan prosedur Clean-Up/Set-Up di lapangan. Sesi dilakukan melalui *demo*/*review* berkala dan kegiatan langsung di lapangan guna mengamati proses aktual, mengumpulkan masukan dari operator, serta mengidentifikasi penyesuaian yang diperlukan. Temuan seperti ketidaksesuaian urutan, kebutuhan umpan balik tambahan, dan penyesuaian interaksi dirangkum lalu dijadikan dasar perbaikan. Kriteria keberhasilan mencakup kesesuaian dengan SOP, kenyamanan penggunaan, dan ketiadaan isu yang menghambat kelancaran skenario.

Gambar 3.14. Sesi testing bersama pihak klien

Gambar 3.14 mendokumentasikan interaksi dengan perwakilan klien saat menilai alur simulasi, menguji kejelasan instruksi, dan memastikan perilaku sistem sejalan dengan kebutuhan operasional di pabrik.

Selain cuplikan sesi, terdapat pula tangkapan layar daftar *scene* yang menjadi objek *quality check* (QC) pada saat testing dengan klien. Daftar ini membantu memastikan cakupan pengujian merata, memudahkan pelacakan progres, dan meminimalkan adanya skenario yang terlewat.



Gambar 3.15. Tangkapan layar daftar *scene* yang di-QC pada sesi testing dengan klien

Gambar 3.15 menunjukkan daftar *scene* yang dikonfirmasi bersama untuk diuji, lengkap dengan prioritas dan catatan temuan yang kemudian dijadikan acuan dalam penjadwalan perbaikan serta pengujian ulang.

3.5.4 Deployment

Hingga penyusunan laporan ini, aplikasi belum melalui proses *deployment* atau serah terima resmi kepada pihak klien untuk penggunaan di lingkungan produksi. Aktivitas yang dilakukan sebatas *build* ke perangkat HMD (Meta Quest 3) untuk kepentingan uji coba fungsional dan validasi skenario secara berkala selama pengembangan.

Proses *build* uji coba dilakukan secara iteratif untuk menguji stabilitas interaksi, kesesuaian alur *Clean-Up/Set-Up*, serta kejelasan instruksi dan *feedback*. Distribusi *build* pengujian dilakukan secara internal dan dipasang ke perangkat menggunakan *side quest* sesuai kebutuhan *testing*. Tahap ini belum termasuk paket rilis final, dokumentasi serah terima, maupun mekanisme *rollout* di lingkungan produksi.

Rencana *deployment* resmi akan dilanjutkan setelah kriteria kesiapan terpenuhi, antara lain penyelarasan konten dengan SOP lapangan, stabilitas sistem pada uji perangkat, serta persetujuan hasil UAT. Pada fase tersebut, paket rilis akan mencakup *build* final untuk Meta Quest 3, panduan penggunaan, catatan rilis (*changelog*), serta dokumentasi teknis bagi tim terkait agar proses adopsi dan pemeliharaan berjalan lancar.

3.6 Kendala dan Solusi yang Ditemukan

3.6.1 Kendala

Selama pelaksanaan magang, muncul sejumlah kendala teknis dan koordinatif yang wajar pada proyek simulasi VR lintas divisi. Tantangan utama terletak pada kebutuhan validasi bersama antartim, kestabilan performa di perangkat HMD, serta sinkronisasi agenda dengan pihak klien. Kendala yang dihadapi adalah sebagai berikut.

1. Kesulitan dalam koordinasi lintas divisi yang memerlukan validasi kolektif sebelum perubahan skrip/menu/SFX diadopsi.

2. Terjadi variasi FPS dan *stutter* pada beberapa skenario dengan beban interaksi/visual tinggi.

3.6.2 Solusi

Sebagai respons, pendekatan penyelesaian difokuskan pada ritme kolaborasi yang terstruktur, optimasi teknis yang terukur, dan mekanisme *testing* yang konsisten. Setiap solusi dirancang agar berkelanjutan serta mudah diintegrasikan dengan proses yang sudah berjalan. Solusi yang dilakukan adalah sebagai berikut.

1. *Weekly check-in* lintas tim dengan agenda tetap (temuan prioritas, dampak ke skrip/menu/SFX) disertai notulensi ringkas untuk mempercepat keputusan bersama.
2. Optimasi performa di HMD melalui pembatasan highlight simultan, penyederhanaan mesh/LOD, dan penggunaan *collider convex* seperlunya.