BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Koordinasi

Selama periode kerja magang di PT Surya Digital Indonesia, praktikan berkedudukan sebagai mahasiswa magang *(internship)* yang terlibat langsung dalam proyek "Rancang Bangun *Game* Interaktif untuk Stimulasi Kognitif Anak Usia Dini". Tanggung jawab yang diemban oleh praktikan meliputi pelaksanaan serangkaian tugas mulai dari tahap perancangan hingga implementasi produk.

Dalam pelaksanaannya, seluruh kegiatan dikoordinasikan secara intensif dengan Bapak Ade Surya Wibawa selaku pembimbing lapangan yang ditunjuk oleh perusahaan. Pembimbing lapangan berperan sebagai *mentor* yang memberikan arahan, masukan, serta evaluasi terhadap progres kerja praktikan. Proses koordinasi tersebut mencakup diskusi awal, bimbingan reguler, pelaporan progres, serta penerimaan umpan balik untuk iterasi dan perbaikan produk.

3.2 Tugas yang Dilakukan

Tugas utama selama kerja magang adalah merancang dan mengembangkan sebuah *Game* edukasi interaktif menggunakan *Unity*, mulai dari proses desain *UI/UX* hingga pembuatan *e-course* sebagai materi pendukung. Lingkup pekerjaan mencakup pengembangan beberapa modul permainan (*Game writing*, *matching*, *comparing*, *fitting*), implementasi fitur *audio* dan *store*, hingga proses *deployment* aplikasi menjadi format *APK*.

3.3 Uraian Pelaksanaan Magang

Pelaksanaan kerja magang difokuskan pada perancangan dan pengembangan *Game* interaktif untuk stimulasi kognitif anak usia dini serta pembuatan *e-course* pendukung. Proses ini mengikuti tahapan-tahapan yang telah diuraikan dalam prosedur pelaksanaan kerja magang (Bab 1.3.2), dengan penekanan pada aspek teknis pengembangan. Tabel 3.1 mendokumentasikan pekerjaan *intern* selama periode magang, dari minggu ke-1 hingga ke-17.

Tabel 3.1. Uraian Pelaksanaan Magang

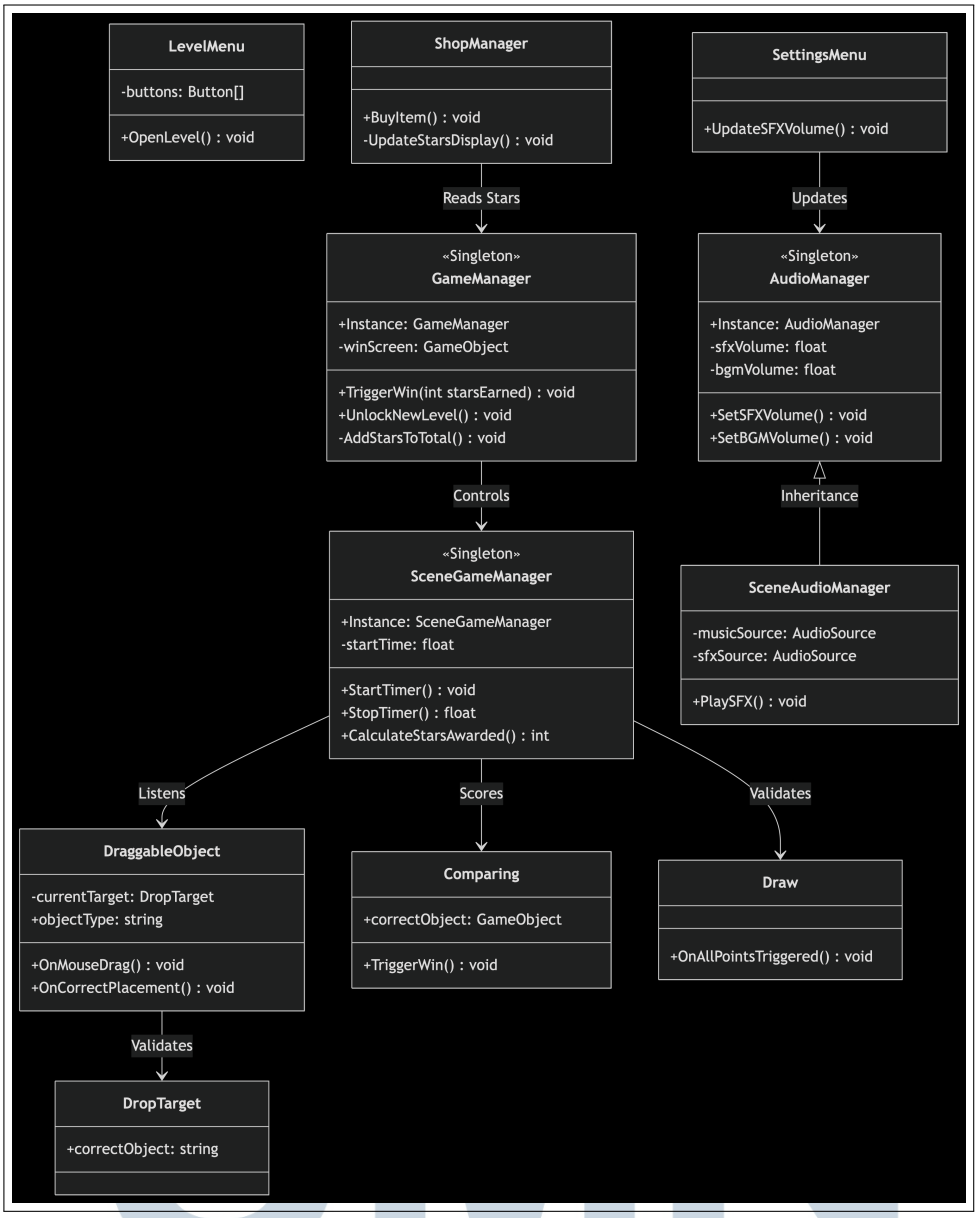
Minggu	Kegiatan
1	*Working on a mobile apps for sports booking*. *Creating a mobile education Game*.
2	*Creating a mobile education Game* menggunakan *Unity*.
3	*Creating a mobile education Game* menggunakan *Unity*.
4	*Creating Income Expenses Company* menggunakan *NextJS, Clerk, PostgreSQL, Neon*. Pengembangan kartu *Digital Website (Full stack + Deployment)*.
5	Melanjutkan kartu *Digital Website (Full stack + Deployment)*. *Creating a mobile education Game* menggunakan *Unity*.
6	*Turkish HVAC Indonesia Landing Page* menggunakan *WordPress, Google Analytics, GA4*.
7	Melanjutkan *Turkish HVAC Indonesia Landing Page* menggunakan *WordPress, Google Analytics, GA4*.
8	*Material Inovasi Industri Landing Page* menggunakan *WordPress*.
9	*Tisec.com Website* menggunakan *WordPress*.
10	*Game Landing Page* untuk *Kun Lof Brands* menggunakan *PhaserJS*. kartu *Digital Website (Full stack + Deployment)*.
11	Melanjutkan *Game Landing Page* untuk *Kun Lof Brands* menggunakan *PhaserJS*.
12	*Developing Landing Page* untuk *Ads Film Karate Kid* menggunakan *JavaScript* dan *HTML5*.
13	Melanjutkan *Landing Page* untuk *Ads Film Karate Kid* menggunakan *JavaScript* dan *HTML5*.
14	*Website Tunas Bangsa School Development* menggunakan *WordPress*.
15	Melanjutkan *Website Tunas Bangsa School Development* menggunakan *WordPress*.
16-17	*Developing Software SAAS Photobooth (Multiapp Platform)* menggunakan *NextJS, NestJS, Electron, PostgreSQL.* Finalisasi dan pengujian *platform Photobooth*.

Tabel ini menunjukkan berbagai tugas, termasuk proyek utama membuat *game* edukasi *mobile* menggunakan *Unity*, serta proyek lain seperti mengembangkan aplikasi pemesanan olahraga dengan *Android Studio* dan *Kotlin*, membangun situs *web full-stack* dengan *NextJS*, dan membuat halaman arahan menggunakan *WordPress*. Hal ini menunjukkan beragamnya tanggung jawab dan teknologi yang ditangani selama magang.

3.3.1 Flowchart Algoritma

A Flowchart Sistem *Game*

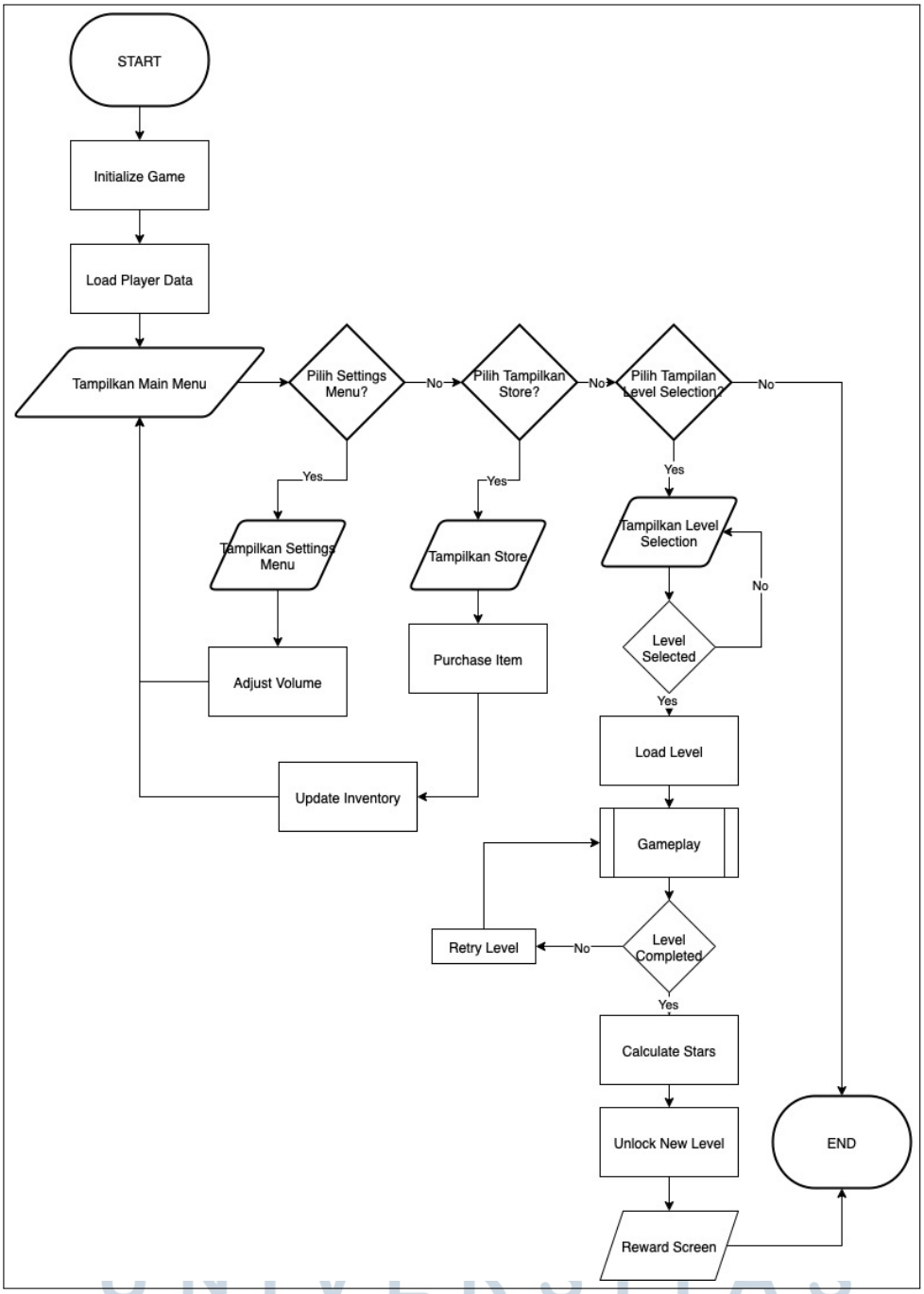
Gambar 3.1 menyajikan diagram *class* yang menguraikan komponen perangkat lunak inti. Diagram ini menunjukkan *GameManager* sebagai pengontrol pusat yang mengelola *SceneGameManager* dan *AudioManager*. Sementara itu, *SceneGameManager* berkoordinasi dengan setiap *MiniGame*, mengilustrasikan penerapan pola desain *Singleton* dan arsitektur berbasis kejadian (*event-driven*).



Gambar 3.1. Class Diagram Sistem *Game*

B *Flowchart User Flow*

Gambar 3.2 memetakan alur pengguna (*user flow*) di dalam *game*. Diagram alir ini dimulai saat aplikasi dibuka, berlanjut ke menu utama, lalu bercabang ke berbagai bagian seperti pemilihan *level*, toko, atau pengaturan *audio*. Alur kemudian mengikuti jalur bermain sebuah *game*, memeriksa kondisi kemenangan, memperbarui progres, dan membuka *level* berikutnya sebelum mengembalikan pengguna ke layar pemilihan *level*.



Gambar 3.2. *Flowchart Game Scene*

C Flowchart Game Scene

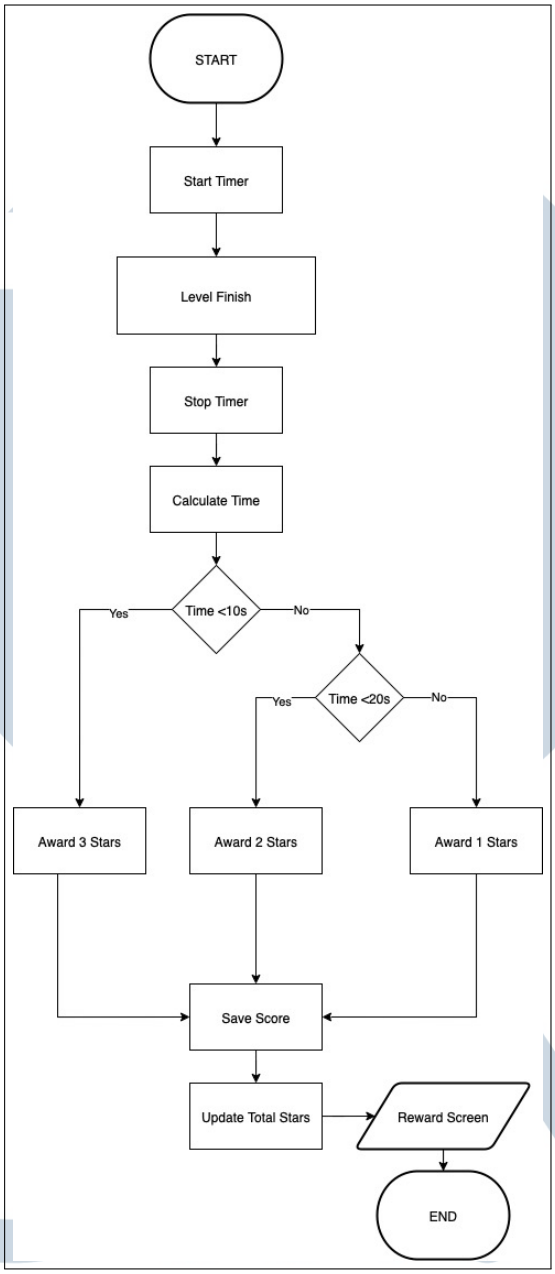
Flowchart pada Gambar 3.3 menggambarkan alur interaksi pengguna dengan sistem secara linear. Proses dimulai dari Start, kemudian sistem menampilkan karakter beserta animasinya (Munculkan Karakter dengan Animasi). Selanjutnya, misi ditampilkan kepada pengguna (Munculkan Misi untuk

Diselesaikan), dan pengguna menyelesaikan misi tersebut (User Menyelesaikan Misi). Alur berakhir di End. Setiap langkah melibatkan proses seperti render animasi, penampilan antarmuka, dan validasi input user.

Gambar 3.3. *Flowchart Game Scene*

D Flowchart Scoring System

Gambar 3.4 merinci logika untuk menghitung skor pemain. Proses ini dimulai oleh interaksi pertama pemain (`OnMouseDown`), yang akan memulai sebuah *timer*. Setelah permainan diselesaikan dengan benar, *timer* akan berhenti, skor bintang akan dihitung berdasarkan waktu yang telah berlalu, layar kemenangan ditampilkan, dan kemajuan pemain disimpan.



Gambar 3.4. *Flowchart Scoring System*

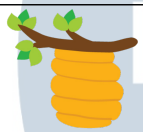
3.3.2 Proses Desain *UI/UX* dengan *Figma*

A Daftar *Assets*

Tabel 3.2, yang berjudul ”Daftar *Assets*”, secara visual membuat katalog aset-aset grafis yang digunakan dalam *game*. Katalog ini mencakup gambar dan nama untuk 39 aset yang berbeda, seperti karakter (Lebah, Beruang Besar, Monyet), objek (Sarang Lebah, Tulang, Wortel), dan elemen *UI* (*Button Close*, *Button Next*,

Icon Setting). Tabel ini menyediakan inventaris yang jelas dari komponen visual yang membentuk dunia *game*.

Tabel 3.2. Daftar *Assets*

No	*Asset*	*Asset Name*
1		*Choose Level 1-10*
2		*Bee*
3		*Beehive*
4		*Dog House*
5		*Puddle*
6		*Background Draw*
7		*Background Main Menu*
8		*Background with Water*
9		*Big Bear*
10		*Bone*

Lanjutan di halaman berikutnya

Tabel 3.2 – Daftar *Assets* (lanjutan)

No	*Asset*	*Asset Name*
11		*Button Close*
12		*Button Continue*
13		*Button Exit*
14		*Button Info*
15		*Button Main Menu*
16		*Button Next*
17		*Bunny*
18		*Carrot*
19		*Cat*
20		*Choose Bigger*
21		*Choose Smaller*
22		*Crocodile*

Lanjutan di halaman berikutnya

Tabel 3.2 – Daftar *Assets* (lanjutan)

No	*Asset*	*Asset Name*
23		*Dog*
24		*Finish*
25		*Fish*
26		*Frog*
27		*Green Frog*
28		*Icon Setting*
29		*Koala*
30		*Apple*
31		*Monkey Run*
32		*Monkey Forward*

Lanjutan di halaman berikutnya

Tabel 3.2 – Daftar *Assets* (lanjutan)

No	*Asset*	*Asset Name*
33		*Setting Panel*
34		*Shop Background*
35		*Star*
36		*game Title*
37		*Coin*
38		*Banana*
39		*Monkey*

B Desain *UIUX Main Menu*

Gambar 3.5 menunjukkan layar menu utama *game* yang diberi merek ”*Jungle Adventure*”. Desain ini menampilkan latar belakang hutan yang cerah dengan tombol ”*PLAY*” dan ”*EXIT*” yang menonjol, bersama dengan karakter *game* seperti monyet dan koala untuk menciptakan pengalaman yang menarik bagi pemain sejak awal.

Gambar 3.5. Desain *UIUX Main Menu*

C Desain *UIUX* Pilihan *Game*

Gambar 3.6 menampilkan layar pemilihan *game*. Antarmuka ini menggunakan menu geser (*carousel*) dinamis di mana pemain dapat menggulir berbagai mode permainan, dengan *game* "*Matching*" ditampilkan sebagai pilihan aktif di tengah.

Gambar 3.6. Desain *UIUX* Pilihan *Game*

D Desain *UIUX Level*

Gambar 3.7 menyajikan layar pemilihan *level* untuk mode permainan ”*Writing*”. Desain ini menggunakan ikon-ikon bernomor dari 1 hingga 10 yang dapat diklik, memungkinkan pemain untuk memilih *level* yang ingin dimainkan.

Gambar 3.7. Desain *UIUX Level*

E Desain *UIUX Matching Game*

Gambar 3.8 mengilustrasikan antarmuka dari ”*Matching Game*”. Adegan ini menampilkan karakter monyet di samping beberapa katak dengan warna berbeda yang harus dicocokkan pemain ke daun teratai yang sesuai, sebuah tugas yang dirancang untuk mengembangkan logika dan pengenalan warna.

Gambar 3.8. Desain *UIUX Matching Game*

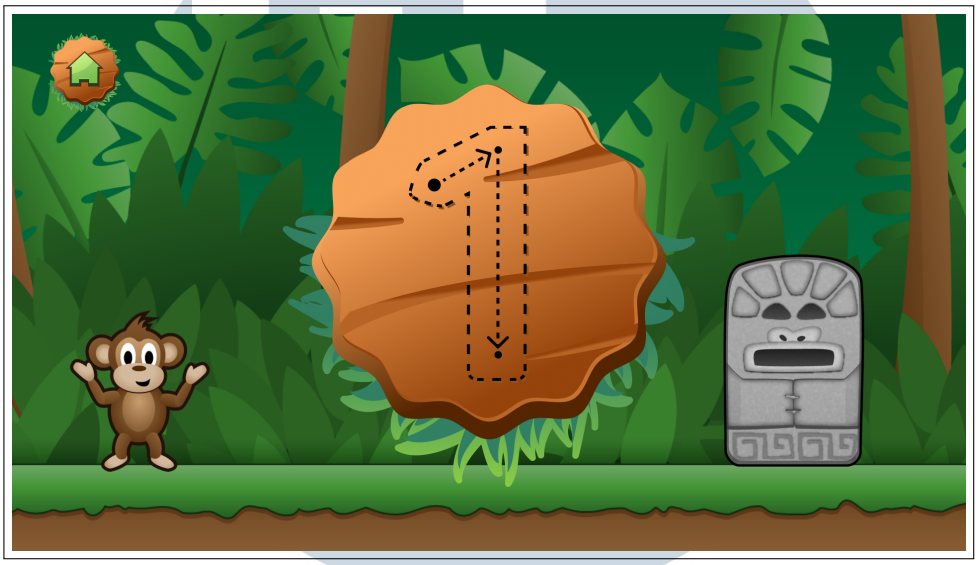
F Desain *UIUX Fitting Game*

Gambar 3.9 menunjukkan permainan ”*Fitting Game*”. Ini adalah sebuah permainan teka-teki (*puzzle*) di mana pemain harus menyeret bentuk-bentuk (bintang, lingkaran, segitiga) dan melepaskannya ke dalam siluet yang cocok, bertujuan untuk mengasah kemampuan spasial.

Gambar 3.9. Desain *UIUX Fitting Game*

G Desain *UIUX Writing Game*

Gambar 3.10 menggambarkan permainan ”*Writing Game*”, di mana pemain diajak untuk menjiplak bentuk angka (’1’) dengan mengikuti jalur putus-putus untuk membantu mengembangkan keterampilan motorik halus.



Gambar 3.10. Desain *UIUX Writing Game*

3.3.3 Arsitektur Sistem *Game*

GameManager bertindak sebagai pusat kendali global yang mengkoordinasikan alur permainan dan kemajuan pemain. *SceneGameManager* menangani logika spesifik *level* seperti *timer* dan penilaian. *AudioManager* mengatur volume suara secara global yang persisten di semua *scene*. Pola *singleton* memastikan hanya satu *instance* yang ada selama *runtime*.

Game edukasi ini dirancang khusus untuk anak usia dini dengan fokus pada pengembangan keterampilan dasar melalui 4 jenis permainan:

1. *Writing Game*: Melatih kemampuan motorik halus dengan menghubungkan titik untuk membentuk huruf/angka
2. *Matching Game*: Mengembangkan logika melalui pencocokan objek berdasarkan warna
3. *Comparing Game*: Melatih pengenalan ukuran dengan mengidentifikasi objek terbesar/terkecil

4. *Fitting Game*: Mengasah kemampuan spasial dengan menempatkan objek ke bentuk yang sesuai

Sistem ini dibangun menggunakan *Unity Engine* dengan arsitektur modular yang memisahkan komponen inti

3.3.4 Implementasi Modul Utama

A *Swipe Menu* dengan Animasi Dinamis

Algoritma pada Kode 3.1 ini menciptakan efek visual dimana elemen di tengah layar (yang sedang fokus) akan membesar (*scale* 1:1), sementara elemen di sampingnya mengecil (*scale* 0.8:0.8). Perhitungan posisi dinamis memastikan adaptasi terhadap jumlah *item* yang berubah. *Lerp function* memberikan transisi animasi yang halus.

```
public class SwipeMenu : MonoBehaviour {
    public GameObject scrollBar;
    float scroll_pos = 0;
    float[] pos;

    void Update() {
        // Hitung posisi relatif tiap elemen
        float distance = 1f / (pos.Length - 1f);
        for (int i = 0; i < pos.Length; i++) {
            pos[i] = distance * i;
        }

        // Animasi scaling untuk elemen aktif
        if (scroll_pos < pos[i] + (distance/2) && scroll_pos > pos
    [i] - (distance/2)) {
            transform.GetChild(i).localScale = Vector2.Lerp(
                transform.GetChild(i).localScale,
                new Vector2(1f, 1f),
                0.1f
            );
        } else {
            transform.GetChild(i).localScale = Vector2.Lerp(
                transform.GetChild(i).localScale,
                new Vector2(0.8f, 0.8f),
                0.1f
            );
```

```
        }
    }
}
```

Kode 3.1: Kode SwipeMenu.cs mengimplementasikan *carousel* responsif

B Sistem Manajemen *Scene*

Pada kode 3.2, setiap transisi *scene* menggunakan animasi *fade-in/fade-out* untuk pengalaman pengguna yang mulus. *Coroutine* memastikan animasi selesai sebelum *scene* baru dimuat. Pemain dapat bernavigasi antara menu utama, pemilihan *level*, toko, dan *scene* permainan dengan alur yang intuitif.

```
public class LoadScene : MonoBehaviour {
    [SerializeField] private string sceneName;

    public void LoadLevel() {
        // Animasi fade-out sebelum load scene
        StartCoroutine(FadeTransition());
    }

    IEnumerator FadeTransition() {
        CanvasGroup fadePanel = GameObject.Find("FadePanel").
    GetComponent<CanvasGroup>();
        float duration = 0.5f;

        // Fade to black
        while (fadePanel.alpha < 1) {
            fadePanel.alpha += Time.deltaTime / duration;
            yield return null;
        }

        // Load scene setelah fade complete
        SceneManager.LoadScene(sceneName);
    }
}
```

Kode 3.2: Kode LoadScene.cs menangani transisi antar *scene*

3.3.5 Sistem Progres dan *Level Management*

A Mekanisme *Unlock Level*

Pada kode 3.3, sistem menyimpan data *unlock level* menggunakan *PlayerPrefs* dengan kunci unik berdasarkan tipe *Game* (*Writing*, *Matching*, dll). Setiap *level* menampilkan bintang yang diperoleh pemain melalui aktivasi *GameObject* anak pada tombol *level*. *Level* terkunci (*interactable=false*) ditampilkan dengan *grayscale*.

```
public class LevelMenu : MonoBehaviour {
    public Button[] buttons;
    public string levelPrefix = "Writing";

    void Awake() {
        int unlockedLevel = PlayerPrefs.GetInt("UnlockedLevel" + levelPrefix, 1);

        for (int i = 0; i < buttons.Length; i++) {
            buttons[i].interactable = (i < unlockedLevel);

            // Tampilkan bintang yang diperoleh
            int stars = PlayerPrefs.GetInt(levelPrefix + i + "_Stars", 0);
            for (int s = 0; s < stars; s++) {
                buttons[i].transform.GetChild(s).gameObject.SetActive(true);
            }
        }
    }
}
```

Kode 3.3: Kode LevelMenu.cs mengatur pembukaan *level*

B *Scoring* dan Bintang

Kode 3.4 memili penilaian berbasis waktu menyelesaikan *level* dengan 3 *tier* bintang. Semakin cepat pemain menyelesaikan *level*, semakin banyak bintang diperoleh. Data disimpan dengan kunci unik berdasarkan nama *scene* untuk mempertahankan progres antar sesi.

```
public int CalculateStarsAwarded(float elapsedTime) {
```

```
    if (elapsedTime <= 10f) return 3;
    else if (elapsedTime <= 20f) return 2;
    else return 1;
}

public void FinalizeLevel() {
    float time = StopTimer();
    int stars = CalculateStarsAwarded(time);

    // Simpan progress
    string sceneName = SceneManager.GetActiveScene().name;
    PlayerPrefs.SetInt(sceneName + "_Stars", stars);

    // Unlock level berikutnya
    GameManager.Instance.UnlockNewLevel();
}
```

Kode 3.4: Kode SceneGameManager.cs menghitung penilaian

3.3.6 Implementasi *Mini-Games*

A *Writing Game (Draw.cs)*

Kode 3.5 memiliki sistem yang menggunakan *LineRenderer* untuk membuat garis dinamis saat pemain menyeret jari. *Collider2D* memvalidasi apakah gambar masih dalam area yang diizinkan. Ketika semua titik *trigger* tersentuh, *level* dianggap selesai dan animasi sukses dipicu.

```
public class Draw : MonoBehaviour {
    private LineRenderer currentLine;

    void OnMouseDrag() {
        Vector3 newPos = GetMousePosition();

        // Validasi dalam area gambar
        if (!drawingArea.OverlapPoint(newPos)) {
            ResetDrawing();
            return;
        }

        // Tambah titik garis
        currentLine.positionCount++;
```

```
15          currentLine.SetPosition(currentLine.positionCount - 1,
    newPos);
16      }
17
18      void CheckTriggerCollision() {
19          foreach (GameObject trigger in triggerPoints) {
20              if (triggerCollider.OverlapPoint(currentLine.
    GetPosition(...))) {
21                  triggeredPoints.Add(trigger);
22                  if (triggeredPoints.Count >= triggerPoints.Count)
    {
23                      OnAllPointsTriggered(); // Semua titik
    tersambung
24                  }
25              }
26          }
27      }
28  }
```

Kode 3.5: Algoritma menggambar dengan *LineRenderer*

B *Matching Game (DraggableObject.cs)*

Untuk menangani interaksi pengguna, setiap objek yang dapat diseret dilengkapi dengan sebuah collider yang berfungsi untuk mendeteksi input. Ketika objek dilepas oleh pengguna, sistem akan memeriksa apakah posisinya berada di atas target yang benar melalui komponen DropTarget. Mekanisme ini, yang juga memberikan feedback visual melalui perubahan posisi, diimplementasikan secara rinci pada Kode 3.6.

```
1   public class DraggableObject : MonoBehaviour {
2       private Vector3 startPosition;
3
4       void OnMouseDrag() {
5           // Update posisi mengikuti jari
6           Vector3 mousePos = Camera.main.ScreenToWorldPoint(Input.
    mousePosition);
7           transform.position = new Vector3(mousePos.x, mousePos.y,
    0);
8       }
9
10      void OnMouseUp() {
```

```
        if (currentTarget != null && currentTarget.correctObject
    == objectType) {
            // Snap ke target jika benar
            transform.position = currentTarget.transform.position;
            GetComponent<Collider2D>().enabled = false;
            SceneGameManager.Instance.ItemPlacedCorrectly();
        } else {
            // Kembali ke posisi awal jika salah
            transform.position = startPosition;
        }
    }
}
```

Kode 3.6: Mekanisme *drag-and-drop*

C *Comparing Game (Comparing.cs)*

Untuk mendeteksi objek yang disentuh oleh pengguna, sistem memanfaatkan teknik raycast. Sebagai feedback langsung, warna objek akan berubah menjadi hijau untuk jawaban yang benar atau merah untuk jawaban yang salah. Agar konsistensi visual tetap terjaga, sebuah coroutine digunakan untuk mengembalikan warna objek ke kondisi aslinya setelah 0.5 detik, seperti yang diimplementasikan pada Kode 3.7.

```
public class Comparing : MonoBehaviour {
    public GameObject correctObject;

    void Update() {
        if (Input.GetMouseButtonDown(0)) {
            RaycastHit2D hit = Physics2D.Raycast(...);

            if (hit.collider != null) {
                if (hit.collider.gameObject == correctObject) {
                    StartCoroutine(ShowFeedback(hit.collider.
    GetComponent<SpriteRenderer>(), Color.green));
                    SceneGameManager.Instance.CompareCorrectly();
                } else {
                    StartCoroutine(ShowFeedback(hit.collider.
    GetComponent<SpriteRenderer>(), Color.red));
                }
            }
        }
```

```
    }

    IEnumerator ShowFeedback(SpriteRenderer renderer, Color color)
    {
        Color original = renderer.color;
        renderer.color = color; // Ubah warna
        yield return new WaitForSeconds(0.5f);
        renderer.color = original; // Kembalikan warna
    }
}
```

Kode 3.7: Logika perbandingan ukuran

3.3.7 Sistem *Audio* Terintegrasi

A *Audio Manager* Global

Untuk mengelola audio secara terpusat, sistem menerapkan pattern Singleton yang memastikan hanya ada satu instance AudioManager selama permainan berjalan. Penggunaan DontDestroyOnLoad menjaga konsistensi pengaturan volume antar scene, sementara PlayerPrefs menyimpan nilai volume agar tetap bertahan antar sesi permainan. Keseluruhan logika untuk manajer audio global ini dapat dilihat pada Kode 3.8.

```
public class AudioManager : MonoBehaviour {
    public static AudioManager Instance;

    private float sfxVolume = 1f;
    private float bgmVolume = 1f;

    void Awake() {
        if (Instance == null) {
            Instance = this;
            DontDestroyOnLoad(gameObject);
            LoadVolumeSettings();
        }
    }

    void LoadVolumeSettings() {
        sfxVolume = PlayerPrefs.GetFloat("SFXVolume", 1f);
        bgmVolume = PlayerPrefs.GetFloat("BGMVolume", 1f);
    }
```

```
19
20      public void SetSFXVolume(float volume) {
21          sfxVolume = volume;
22          PlayerPrefs.SetFloat("SFXVolume", volume);
23          PlayerPrefs.Save();
24      }
25  }
```

Kode 3.8: Kode AudioManager.cs mengatur volume global

B Implementasi *Per-Scene*

Selain manajer audio global, setiap scene memiliki AudioManager lokal yang mengontrol audio spesifik untuk level tersebut. Komponen ini mengambil pengaturan volume dari AudioManager global untuk menjaga konsistensi, serta menyediakan metode PlaySFX() yang memungkinkan efek suara dipicu dari berbagai event dalam game. Logika untuk manajer audio level ini diimplementasikan dalam Kode 3.9.

```
1  public class SceneAudioManager : MonoBehaviour {
2      [SerializeField] private AudioSource musicSource;
3      [SerializeField] private AudioSource sfxSource;
4
5      void Start() {
6          musicSource.volume = AudioManager.Instance.GetBGMVolume();
7          musicSource.clip = Backsound;
8          musicSource.Play();
9      }
10
11     public void PlaySFX(AudioClip clip) {
12         sfxSource.PlayOneShot(clip);
13     }
14 }
```

Kode 3.9: Kode SceneAudioManager.cs menangani *audio* spesifik *level*

3.4 Kendala dan Solusi yang Ditemukan

Selama pelaksanaan kerja magang dalam proyek ”Rancang Bangun *Game* Interaktif untuk Stimulasi Kognitif Anak Usia Dini” dan pembuatan *e-course* pendukung, praktikan menghadapi beberapa kendala. Namun, dengan berbagai upaya, kendala-kendala tersebut berhasil diatasi.

3.4.1 Kendala yang Ditemukan:

Selama pelaksanaan magang, terdapat beberapa kendala teknis yang dihadapi dalam proses pengembangan proyek, baik dari sisi teknis pemrograman, desain, maupun manajemen waktu. Berikut adalah rincian kendala yang dihadapi:

1. Kesulitan teknis dalam menerjemahkan logika permainan yang kompleks ke dalam skrip *C* di *Unity* secara efisien.
2. Tantangan dalam manajemen waktu akibat lingkup proyek yang luas, yang mencakup pengembangan beberapa modul *Game* dan *e-course* secara bersamaan.
3. Keterbatasan dalam melakukan pengujian *usabilitas* langsung dengan target pengguna (anak usia dini), sehingga evaluasi desain lebih bergantung pada teori dan masukan pembimbing.

3.4.2 Solusi yang Diterapkan

Untuk mengatasi berbagai kendala yang muncul selama pelaksanaan magang, praktikan menerapkan beberapa strategi dan pendekatan sebagai berikut:

1. Kendala teknis diatasi dengan melakukan riset secara proaktif melalui dokumentasi resmi *Unity*, forum pengembang, dan tutorial. Selain itu, dilakukan konsultasi rutin dengan pembimbing lapangan untuk mendapatkan arahan dan validasi atas pendekatan teknis yang diambil.
2. Untuk mengelola lingkup proyek, diterapkan manajemen waktu yang terstruktur dengan membuat rencana kerja mingguan. Tugas-tugas diprioritaskan berdasarkan urgensi dan dependensinya, dengan fokus pada penyelesaian fitur-fitur inti terlebih dahulu.
3. Untuk mengatasi keterbatasan pengujian, desain *UI/UX* didasarkan pada studi literatur dan *best practices* yang ada untuk desain antarmuka anak. Simulasi penggunaan dan umpan balik dari pembimbing juga digunakan sebagai metode evaluasi internal untuk memastikan produk tetap ramah pengguna.