BAB 3
PELAKSANAAN KERJA MAGANG

3.1 Kedudukan dan Organisasi

Pada magang kali ini, *Game Programmer* bertugas untuk mengembangkan fitur-fitur *game* yang diperlukan, seperti gerakan pemain, kesehatan musuh, sistem penempatan, pemindahan item, dan lainnya. Selain itu, sebagai *Game Programmer* di PT Anoa Interactive Studio, terdapat dua pembimbing, Gde Danendra Arya Nugraha sebagai CEO dan Auryan Pratama sebagai CTO, bertanggung jawab untuk mengevaluasi fitur-fitur yang telah dibuat sehingga *game* yang dibuat menjadi berkualitas tinggi dan memberikan pengalaman bermain yang menyenangkan. Komunikasi, baik dalam rapat atau sehari-hari, dilakukan melalui aplikasi Discord dan Google Meet selama pelaksanaan magang.

3.2 Tugas yang Dilakukan

Sesuai dengan proyek yang sedang dikembangkan, tugas yang dilakukan adalah membuat dan mengembangkan fitur. Salah satu fiturnya adalah gerakan karakter, bullet pool, menembak, darah musuh, sistem penempatan, dan lainnya. Bahasa pemrograman yang digunakan adalah C-Sharp. Setelah melewati tahap pembuatan dan pengembangan, langkah selanjutnya adalah menguji dan menyesuaikan fitur-fitur yang telah dibuat secara langsung melalui Unity Game Engine. Proses ini dilakukan untuk memastikan bahwa fitur sesuai dengan tujuan, memastikan bahwa fitur berjalan tanpa masalah, dan untuk mengidentifikasi *bug*. Jika ditemukan, *bug* akan diperbaiki.

Tahapan selanjutnya adalah menciptakan area atau lingkungan yang digunakan sebagai area permainan. Area yang akan dikembangkan adalah sebuah area yang menggunakan *Grid*. *Grid* ini akan menjadi sebuah tempat bagi pemain untuk bisa menempatkan berbagai balok yang akan digunakan sebagai senjata. Tipe permainan yang akan dikembangkan dan mekanik permainan yang akan dikembangkan harus disesuaikan dengan pengembangan lingkungan. Menggunakan sumber daya untuk melengkapi permainan yang dikembangkan, seperti penambahan karakter, efek suara, efek visual, bahan, texture, dan lainnya, dilakukan selain pengembangan area.

Selain itu, tahapan selanjutnya yang akan dilakukan adalah pengujian

performa. Pengujian ini bertujuan untuk meningkatkan pengalaman bermain. Pengujian akan dilakukan terhadap FPS dan kinerja komputer untuk mencegah crash, yang dikarenakan kinerja komputer yang terlalu tinggi. Jika hasilnya menunjukkan performa yang buruk misalnya, FPS di bawah 60, lag disebabkan oleh proses rendering gambar yang terlalu banyak [5], maka pengujian akan dilakukan untuk meningkatkan pengalaman bermain. Tugas berikutnya adalah melakukan check in menggunakan aplikasi Plastic SCM agar anggota tim lainnya dapat melakukannya. Selanjutnya, hal yang juga sering dilakukan adalah mengisi tugas harian menggunakan aplikasi Notion, yang memungkinkan Anda melaporkan apa yang sudah dilakukan selama satu hari.

3.3 Uraian Pelaksanaan Magang

Tabel 3.1 merupakan uraian pelaksanaan magang dari minggu satu hingga minggu enam belas.

Tabel 3.1. Deskripsi pekerjaan mingguan

Minggu	**Deskripsi Pekerjaan**
1	Pembahasan proyek yang akan dikembangkan serta melakukan pengaturan versi dari Unity Game Engine
2	Membuat tempat permainan berupa *grid base layout* yang akan digunakan sebagai area permainan
3	Membuat *bullet pool, bullets*, dan *bulletSpawner*
4	Membuat *change parents, object database, disable buttons*, dan *disable mouse*
5	Membuat *enemy health, enemy movement*, dan *game over conditions*
6	Membuat *game over menu, pause menu*, dan *home UI*
7	Membuat *grid data, input manager*, dan *kill counter*
8	Membuat *transparent Shadder, grid visualization*, dan *grid indicator*
9	Membuat *moving floor, object database*, dan *pause system*
10	Membuat *object placer, placement state*, dan *placement system*
	Lanjut pada halaman berikutnya

Tabel 3.1 Deskripsi pekerjaan mingguan (lanjutan)

Minggu	Deskripsi Pekerjaan
11	Membuat *player movement, player shoot*, dan *preview system*
12	Membuat *removing state, transform weapon position*, dan *next level*
13	Melakukan pengujian dan optimalisasi, serta melakukan *bug testing and bug fixing*
14	Membuat *customize weapon* dan *add weapon effect*
15	Membuat *weapon upgrade*, dan *unlock more slot*
16	Melakukan pengujian dan optimalisasi, serta melakukan *bug testing and bug fixing*

3.3.1 Perangkat Yang Digunakan

Pada pelaksanaan kerja magang, perangkat keras maupun perangkat lunak akan digunakan sebagai media pelaksanaan kerja magang. Berikut adalah perangkat keras yang digunakan adalah laptop Asus TUF FX506II.

1. Processor AMD Ryzen™ 8 5800 MHZ 16 Core
2. RAM 16GB DD4 4200 MHZ
3. Storage SSD 2000GB
4. GPU 3650 TI
5. Mouse Logitech Hero 503G

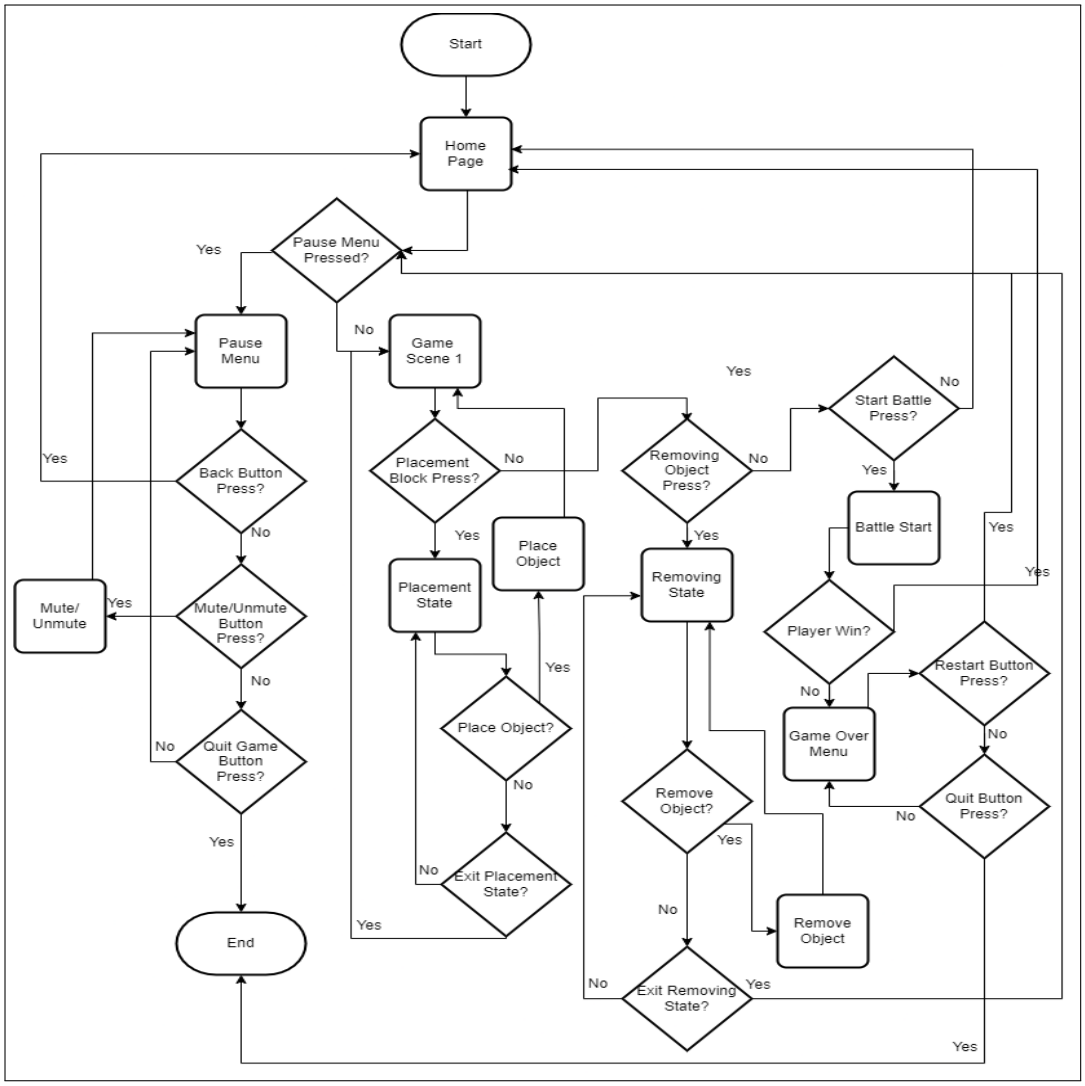
Berikut adalah perangkat lunak yang digunakan dalam pelaksanaan kerja magang.

1. Unity Hub 3.6.1
2. Unity Editor 2022.3.11f1
3. Visual Studio Code
4. Discord

5. Notion
6. Plastic SCM
7. Windows 11 64-bit
8. Microsoft Edge
9. Unity Asset Store
10. Unity Documentation

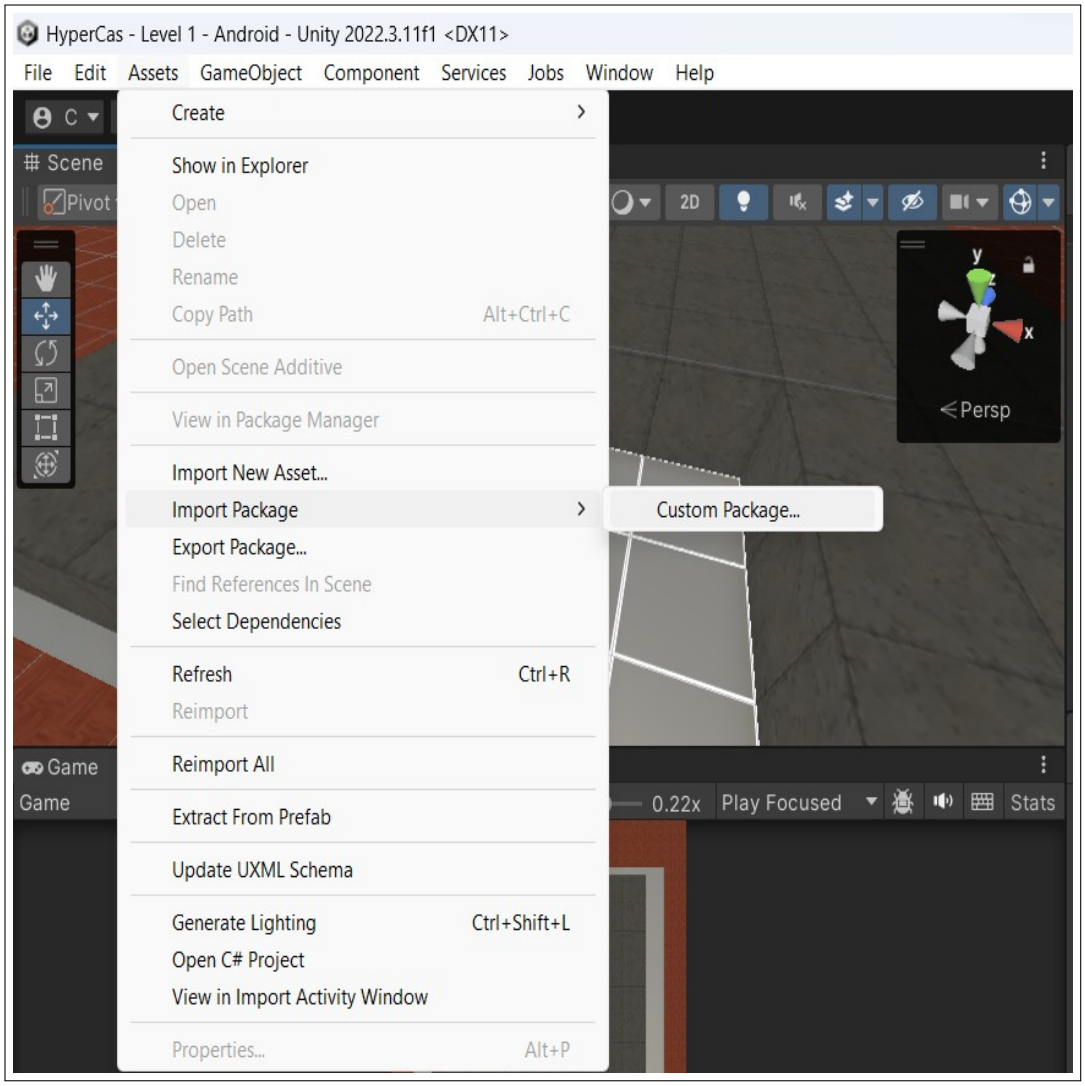
3.3.2 Pengembangan Proyek Magang

Berikut ini Gambar 3.1 merupakan *flowchart* sebagai gambaran umum dari *game* yang akan dibuat. Pada Gambar 3.1 diawali dengan menjalankan permainan, pemain akan masuk menuju halaman *home* pada halaman ini para pemain bisa memilih beberapa pilihan seperti, *pause menu, placement block,* dan *removing object*. Pada *pause menu* pemain bisa mengatur menyalakan ataupun mematikan suara, pemain juga bisa menekan tombol *quit game* untuk keluar dari permainan, jika pemain menekan tombol *back* maka pemain akan kembali ke halaman *home*. Jika pemain memilih untuk memasang balok, pemain akan memasuki fitur *placemement state*. Sebaliknya, pemain juga bisa melakukan penghapusan terhadap balok yang sudah terpasang dengan memasuki fitur *removing state*. Jika pemain sudah melakukan penyusunan balok, pemain bisa menekan tombol *battle start* untuk memulai permainan.



Gambar 3.1. *Flowchart game flow*

Pada tahap awal pengembangan proyek magang, pengembang harus memahami *Game Design Document*, yang dimaksudkan untuk memberikan gambaran keseluruhan mengenai proyek yang akan dikerjakan. Dengan memahami *Game Design Document*, dengan begitu maka pengembang dapat menyusun rancangan fitur apa saja yang perlu dilakukan terlebih dahulu. *Game Design Document* juga dapat membantu dalam membagi fitur utama dengan fitur tambahan. Gambar 3.2 menunjukkan proses selanjutnya setelah mempelajari *Game Design Document* adalah membuat proyek baru pada aplikasi Unity Hub [6].

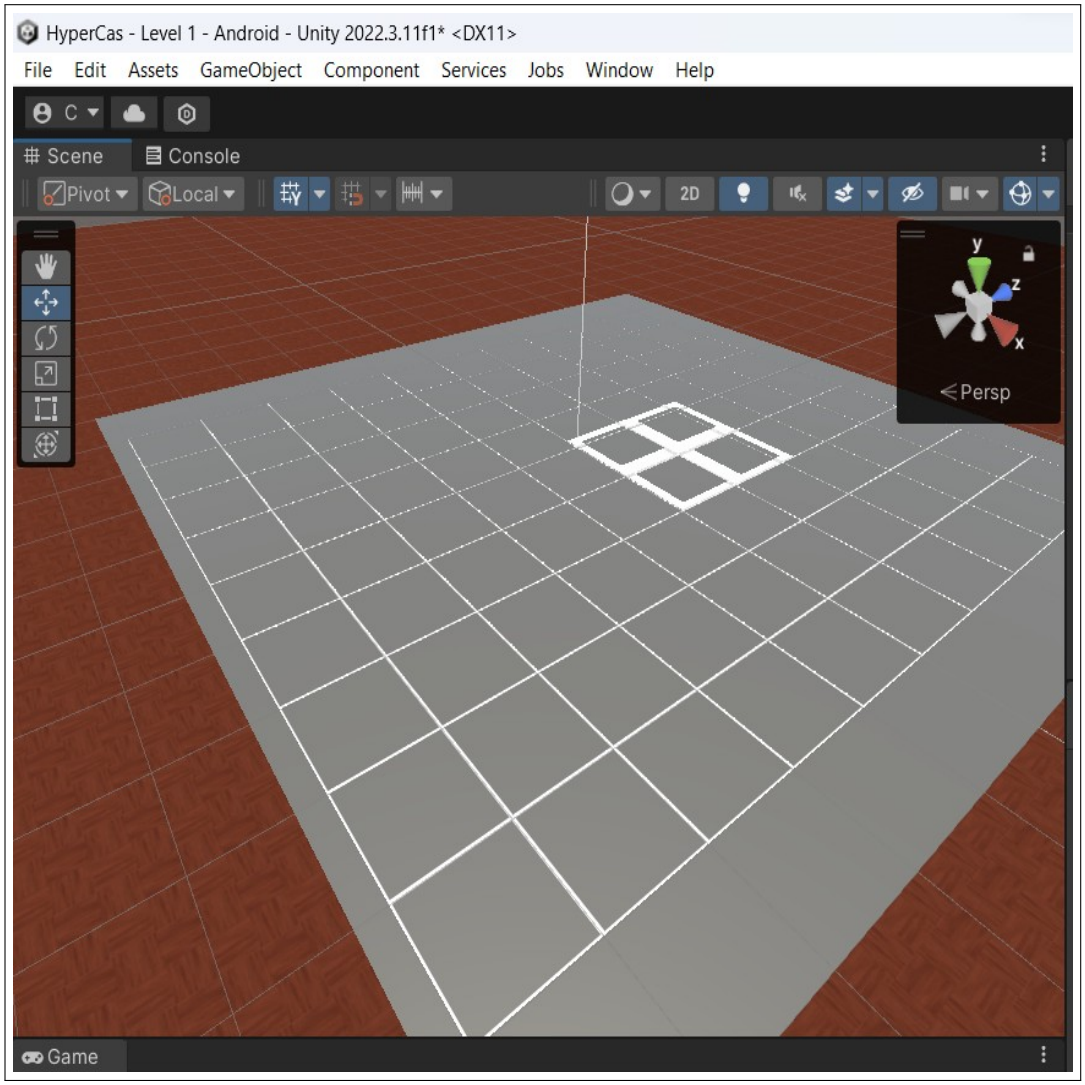


Gambar 3.2. *Import package*

Pada Gambar 3.2 akan menunjukkan gambaran tahapan *import* terhadap *asset* yang digunakan. *Import asset* dilakukan dengan menekan menu *Assets, Import New Package, Custom Package*. *Asset* bisa didapatkan melalui *Unity Assets Store*, *Unity Assets Store* menyediakan berbagai jenis *asset* seperti, animasi, *model, texture, editor extensions*, dan *tutorials* [7].

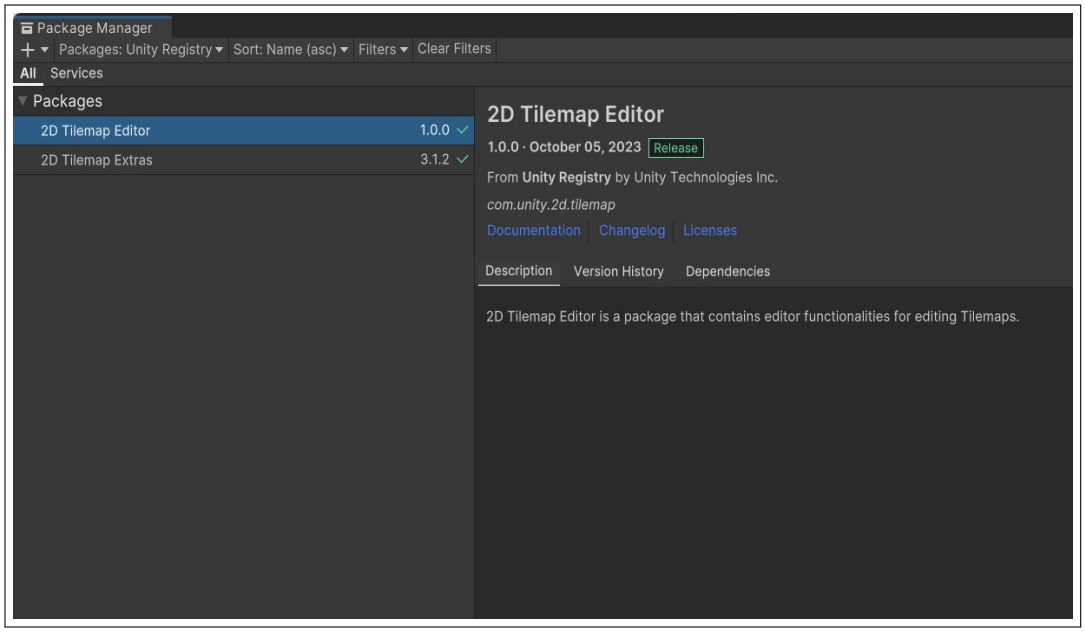
Setelah melalui tahapan *import packages*, fitur-fitur *game* sudah bisa dibuat. Tahapan selanjutnya adalah menyusun dan mengembangkan *environment* yang akan digunakan sebagai area permainan kali ini. Pada proyek kali ini area yang akan dikembangkan adalah *Grid Base Layout*. *Grid Base Layout* ini akan digunakan para pemain untuk meletakkan berbagai balok yang nantinya balok tersebut akan menjadi senjata utama dari pemain. Gambaran dari *Grid Base Layout* bisa dilihat

pada Gambar 3.3.



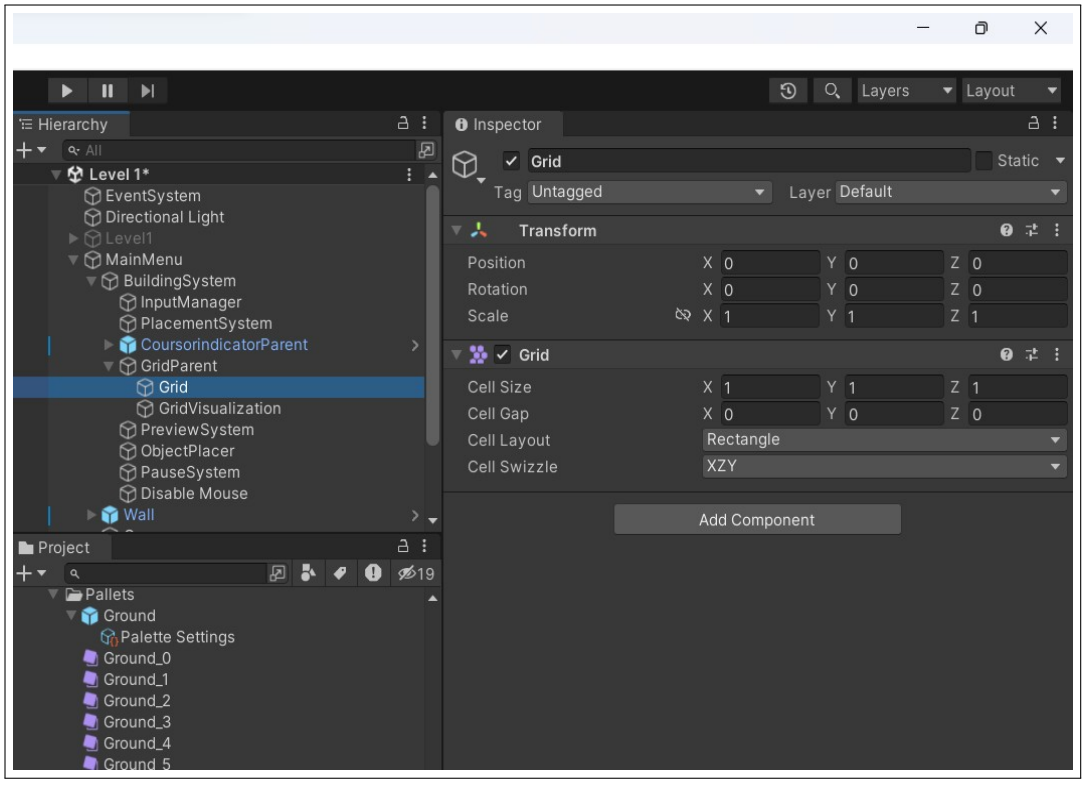
Gambar 3.3. *Grid base layout*

Pada proses pembuatan *Grid Base Layout* hal pertama yang dilakukan adalah memasang *2D Tilemap Editor* yang ada pada menu *Window, Package Manager, Unity Registry*, kemudian cari *2D Tilemap Editor* dan tekan *install*. Jika sudah berhasil maka sudah bisa menggunakan fungsi untuk melakukan *editing* pada *tilemaps*. Gambaran dari *2D Tilemap Editor* bisa dilihat pada Gambar 3.4.



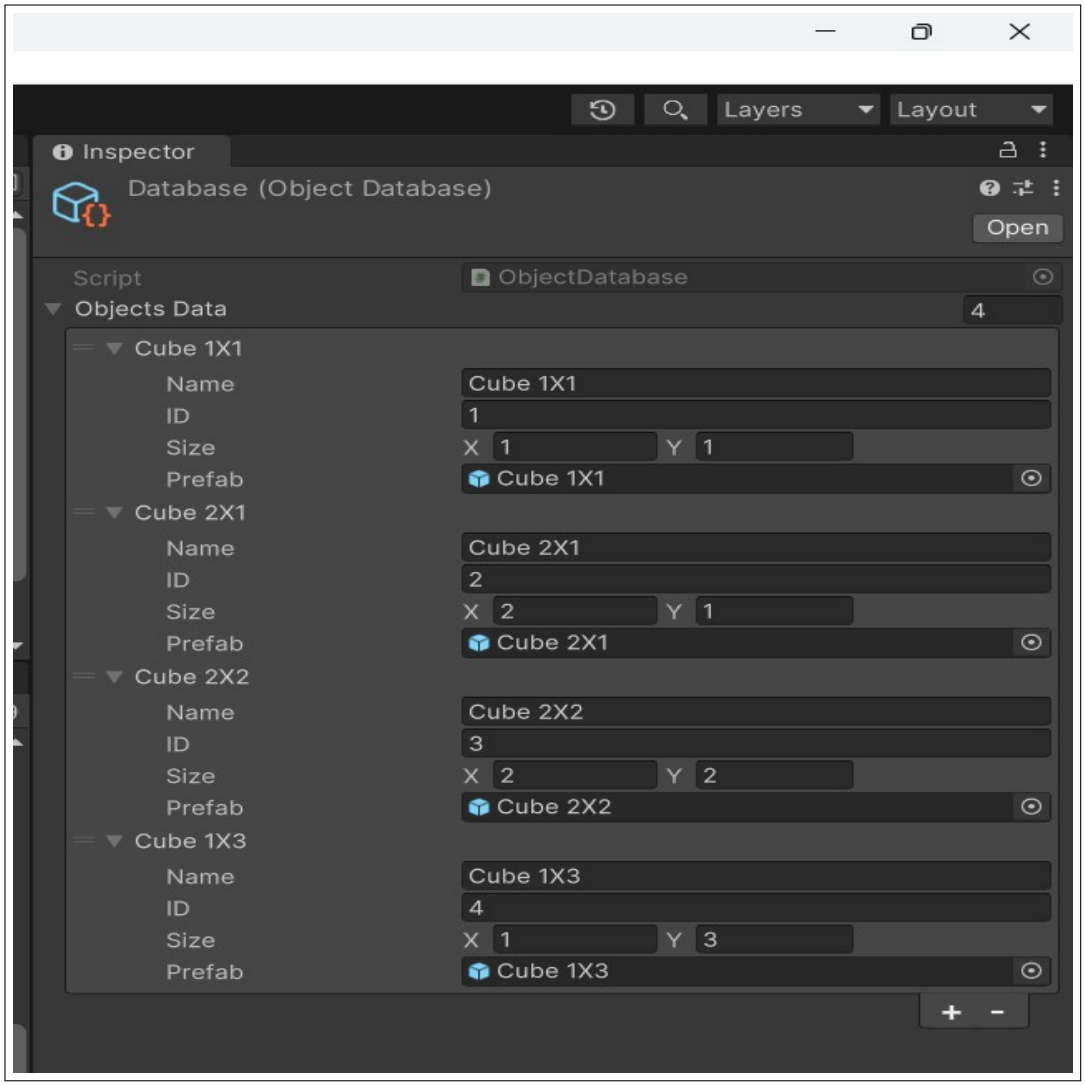
Gambar 3.4. *2D Tilemap editor*

Selanjutnya, hal yang akan dilakukan adalah memasang komponen *grid*. Komponen *grid* akan digunakan sebagai dasar dari area permainan yang akan dikembangkan. Penambahan komponen bisa dilakukan dengan cara melakukan *add component* kemudian memilih *grid*. *Grid* adalah sebuah fondasi yang digunakan untuk memberikan sebuah *layout*. *Grid component* akan menyimpan data dari *grid* yang nantinya akan membantu *function* dalam mengakses berbagai informasi seperti, lokasi sel dan posisi lokal dari sebuah objek [8]. Pada *grid* kita juga bisa mengatur berbagai macam hal pada menu *Inspector* seperti, ukuran sel, jarak antar sel, bentuk sel, dan *layout* sel. Gambar 3.5 menunjukkan gambaran dari pengaturan pada *grid component*.



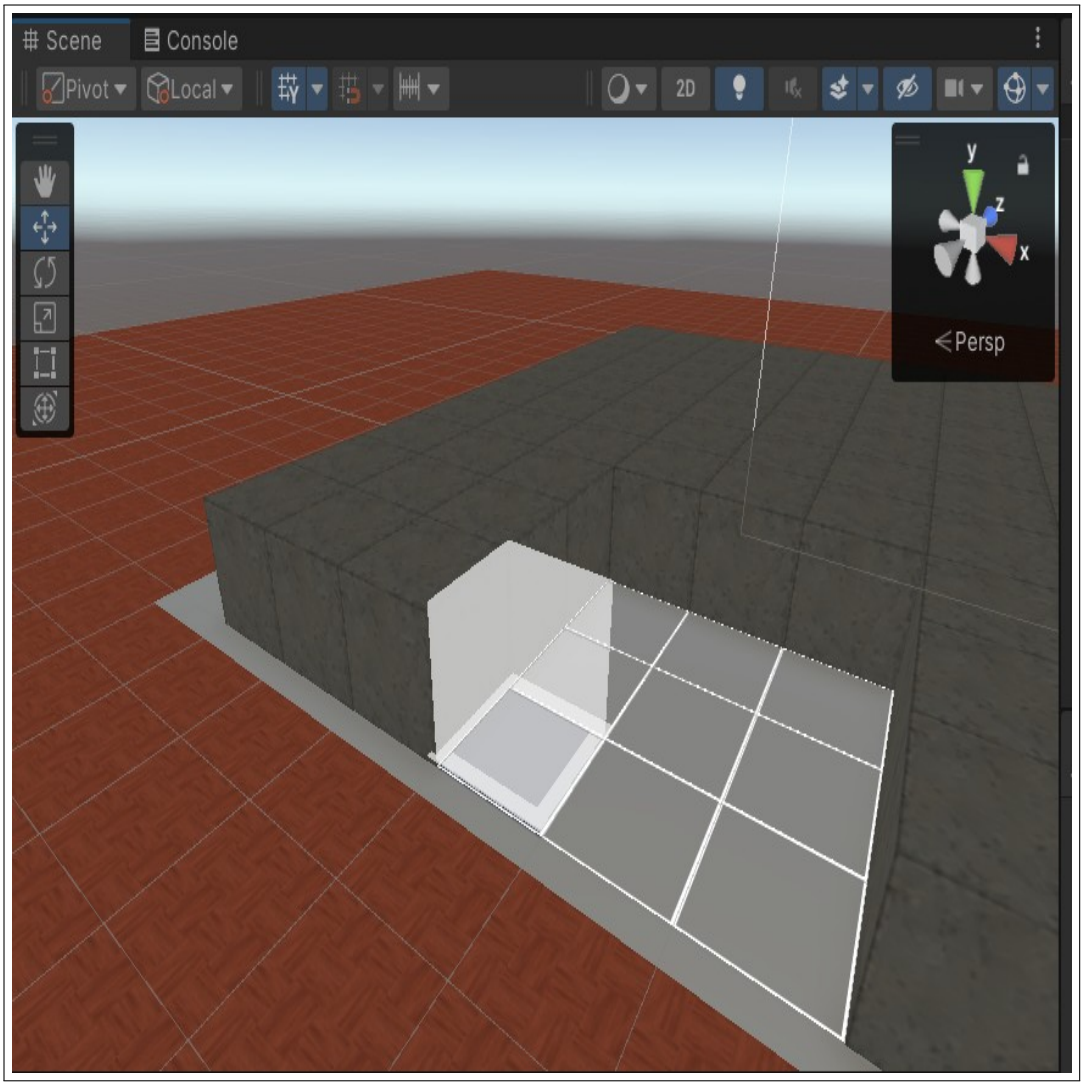
Gambar 3.5. *Grid inspector*

Tahapan selanjutnya adalah pembuatan *database*, *database* ini akan digunakan sebagai tempat penyimpan informasi dari berbagai objek yang akan digunakan sebagai balok senjata. Pembuatan *database* ini akan dilakukan dengan pembuatan *scriptable object*, dengan penggunaan *scriptable object* data dapat disesuaikan secara spesifik tergantung data apa yang diperlukan. Gambar 3.6 menunjukkan gambaran dari *scriptable object database*.



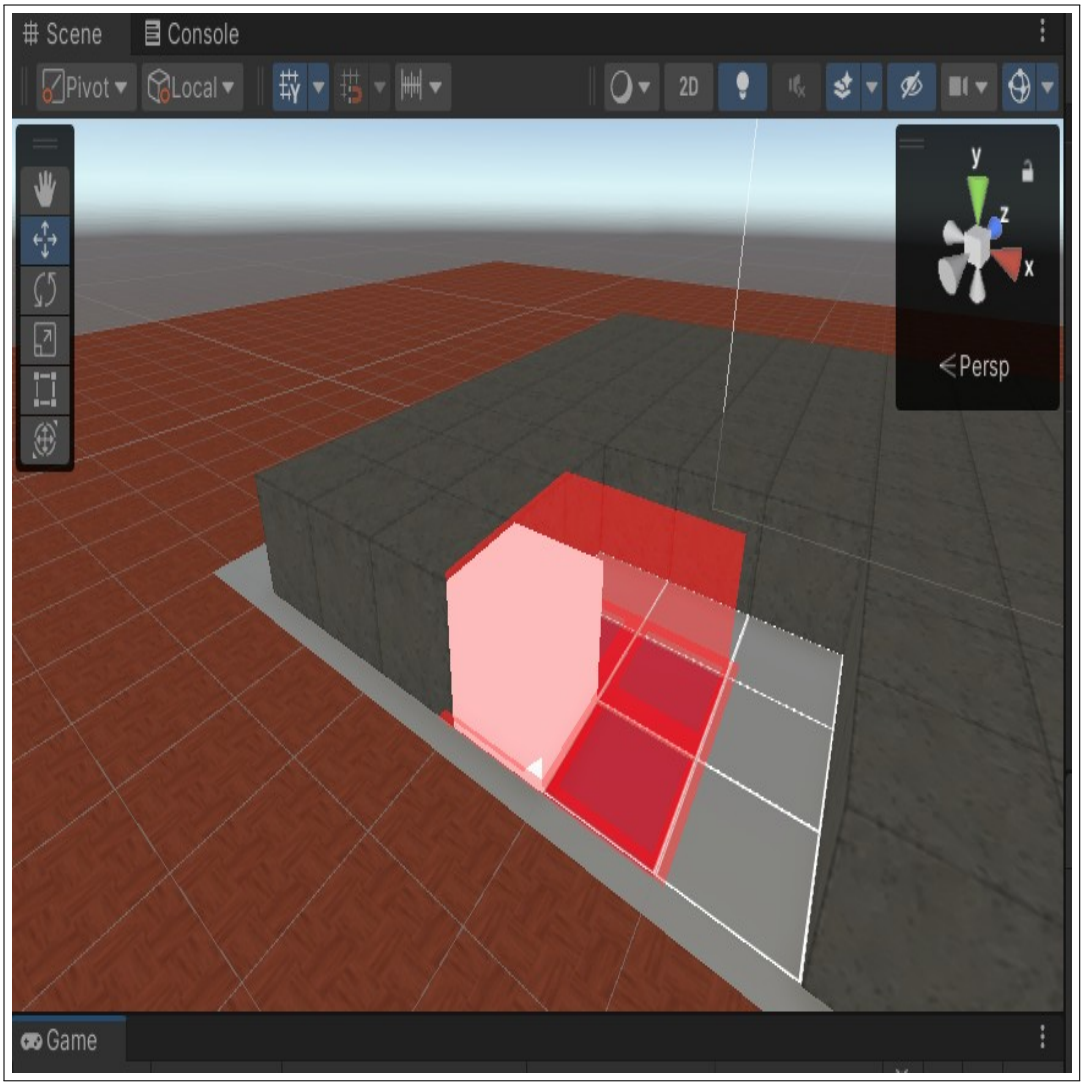
Gambar 3.6. *Scriptable object database*

Database yang dibuat akan menampung berbagai informasi seperti nama, ID, ukuran, serta *prefab* atau objek yang akan digunakan. Setelah membuat *database* hal selanjutnya yang akan dibuat adalah *placement system*, fitur ini berfungsi agar pemain bisa memilih balok yang akan ditempatkan pada *grid* yang sudah dibuat sebelumnya. *Placement system* akan mengambil informasi sesuai dengan *database* yang sudah kita buat sebelumnya. Tahapan selanjutnya adalah menambahkan *object preview*, hal ini bertujuan agar pemain bisa melihat benda apa yang akan ditaruh pada *grid* yang tersedia. Gambar 3.7 menunjukkan gambaran dari *object preview*.



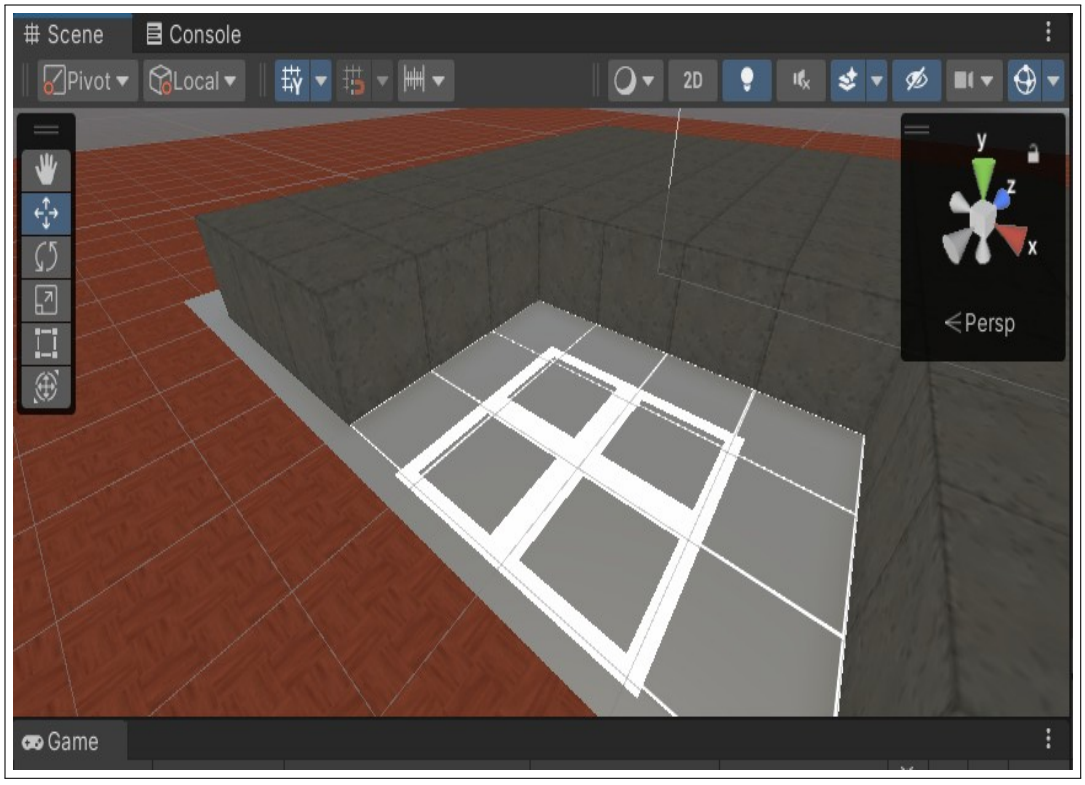
Gambar 3.7. *Object preview*

Object preview akan memberikan informasi mengenai *grid* yang sudah ditempati oleh balok lainnya, hal ini bisa terlihat pada Gambar 3.8 yang menunjukkan *object preview* akan berubah warna menjadi merah yang menandakan bahwa titik tersebut sudah ditempati oleh balok lainnya.



Gambar 3.8. *Object preview restrictions*

Selanjutnya, hal yang akan dikembangkan adalah penambahan indikator dari sebuah sel, yang terlihat pada Gambar 3.9. Penambahan indikator ini bertujuan agar pemain mengetahui dengan jelas posisi dari kursor. Indikator akan mengambil informasi dari posisi kursor pemain, Indikator juga akan menampilkan ukuran balok yang dipilih saat ini sesuai dengan ukuran asli dari balok yang ada pada *database*.



Gambar 3.9. *Cell indicator*

Pada tahapan selanjutnya adalah pengembangan *movement*, pengembangan *movement* terbagi menjadi dua jenis yaitu, *movement* untuk pemain dan *movement* untuk musuh. *Movement* yang digunakan pada pemain adalah pergerakan secara *horizontal*, pemain hanya bergerak ke arah kiri ataupun ke arah kanan sedangkan, *movement* pada lawan hanya akan bergerak ke arah depan. Contoh dari *script player movement* bisa dilihat pada Kode 3.1.

```
1
2  using System.Collections;
3  using System.Collections.Generic;
4  using UnityEngine;
5
6  public class PlayerMovement : MonoBehaviour
7  {
8      [SerializeField] private float FltMoveSpeed;
9      [SerializeField] private float FltHorizontalRange;
10
11     public Joystick joystick;
12
13     void FixedUpdate ()
14     {
```

```
15          float HorizontalInput = joystick.Horizontal;
16
17          float HorizontalOffset = HorizontalInput * FltMoveSpeed *
    Time.deltaTime;
18
19          float RawHorizontalPos = transform.position.x +
    HorizontalOffset;
20          float clampedHorizontalPos = Mathf.Clamp(RawHorizontalPos,
     -FltHorizontalRange, FltHorizontalRange);
21
22          transform.position = new Vector3(clampedHorizontalPos,
    transform.position.y, transform.position.z);
23      }
24  }
```

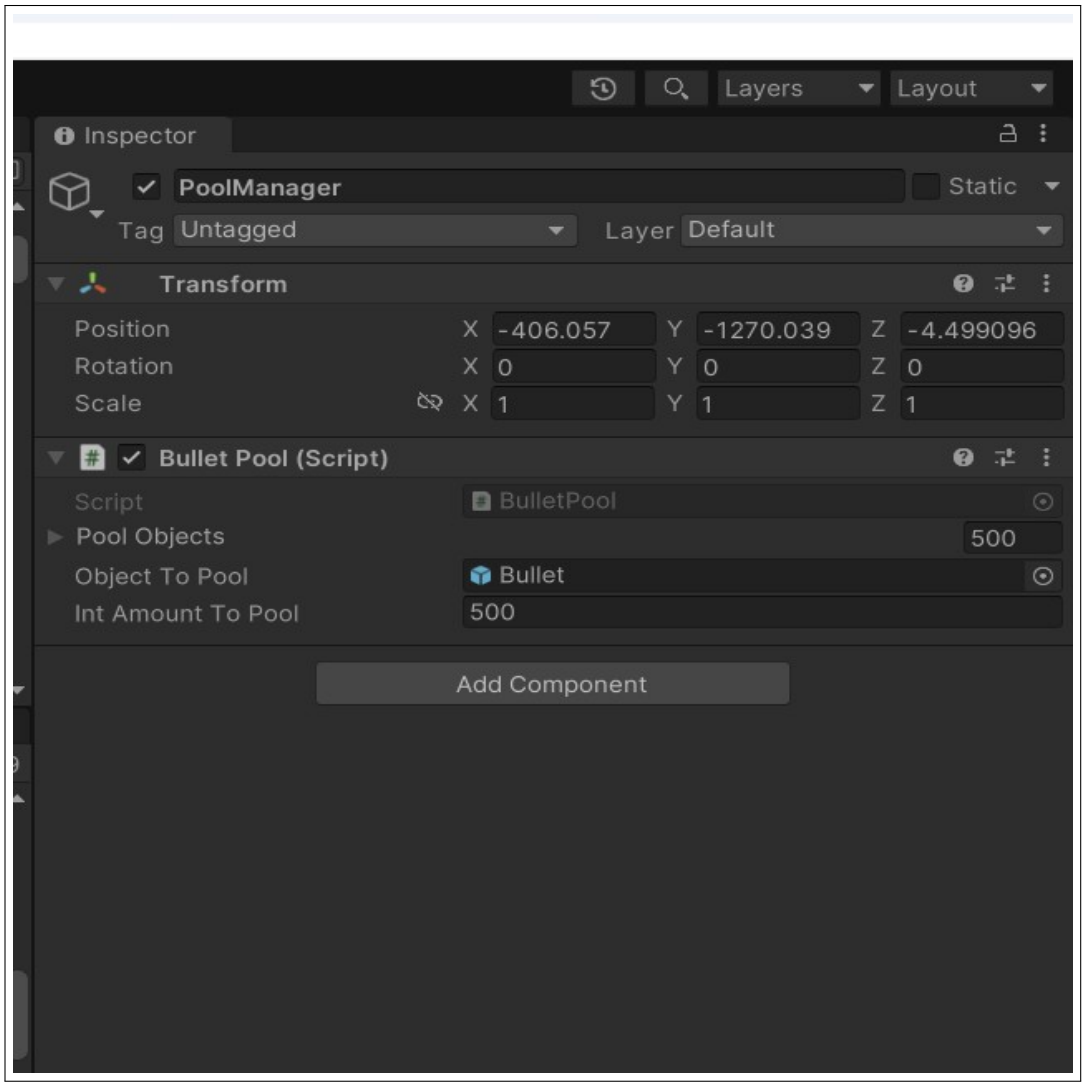
Kode 3.1: Player movement script

Pada Kode 3.1 terdapat beberapa variabel yang dibutuhkan untuk membuat *player movement* seperti, *float movespeed* dan *float horizontalrange*. *Float movespeed* bertujuan untuk mengatur kecepatan dari pergerakan pemain dan *float horizontal range* bertujuan untuk membatasi sejauh mana pemain bisa bergerak dari kiri hingga ke kanan. Pada fungsi *FixedUpdate* hal yang akan diproses adalah penerimaan *input* menggunakan *joystick* dengan arah *horizontal* maka posisi dari pemain akan berubah sesuai dengan *input* ke arah kiri maupun kanan. *Time.deltaTime* berfungsi agar kecepatan pergerakan pemain tidak dipengaruhi oleh jumlah *frame per second* dari perangkat yang digunakan.

Selanjutnya adalah pembatasan jarak bagi pemain dengan menggunakan *Mathf.Clamp*, hal ini bekerja dengan memberikan rentang nilai maksimal dan rentang nilai minimal sehingga akan membuat pergerakan dari pemain menjadi terbatas [9]. Untuk pergerakan pada musuh yang ada akan menggunakan *transform.forward* hal ini dikarenakan musuh hanya bergerak maju ke arah pemain dan tidak membutuhkan pergerakan lainnya. *Transform.forward* sendiri akan mengembalikan nilai dari *axis* berwarna biru yang ada pada *world space*.

Tahapan selanjutnya adalah pembuatan *bullet pool* yang akan digunakan sebagai kotak peluru untuk pemain. Konsep *pool* adalah konsep di mana sebuah *gameobject* hanya akan diciptakan sekali dan ketika objek tersebut digunakan objek tersebut tidak akan dihapus melainkan akan digunakan kembali ketika ingin digunakan kembali. Dengan menggunakan konsep *pool* maka akan menghemat kinerja dari *CPU* karena *CPU* tidak perlu melakukan proses menciptakan objek dan penghapusan objek secara berulang [10]. Pada *pool* kita juga bisa mengatur jumlah

objek yang akan dibuat pertama kali saat menjalankan permainan yang terlihat pada Gambar 3.10 sebagai gambaran dari *bullet pool*.



Gambar 3.10. *Bullet pool*

Tahapan selanjutnya adalah pembuatan fitur *player shoot*, sistem ini akan menggunakan *bullet pool* sebagai peluru yang akan digunakan saat menembak. Contoh dari *script player shoot* bisa dilihat pada Kode 3.2.

```
1
2 using System.Collections;
3 using System.Collections.Generic;
4 using UnityEngine;
5 using UnityEngine.Pool;
6
```

```
public class PlayerShoot : MonoBehaviour
{
    [SerializeField] private Transform BulletPosition;
    [SerializeField] private float time;

    public float FiringRate;
    public bool shoot;
    public GameObject BulletSpawnPosition;
    MoveBulletSpawnPosition Move;

    private void Start()
    {
        Move = BulletSpawnPosition.GetComponent<
    MoveBulletSpawnPosition>();
    }

    void FixedUpdate()
    {
        shoot = Move.DisableMove;
        time += Time.deltaTime;
        float NextTimeToFire = 1 / FiringRate;
        if (time >= NextTimeToFire)
        {
            if(shoot)
            {
                Shoot();
                time = 0;
            }
        }
    }
    private void Shoot()
    {
        GameObject bullet = BulletPool.SharedInstance.
    GetPoolObjects();
        if (bullet != null)
        {
            bullet.transform.position = BulletPosition.position;
            bullet.transform.rotation = BulletPosition.rotation;
            bullet.SetActive(true);
        }
    }
}
```

Kode 3.2: Player shoot script

Pada Kode 3.2 terdapat beberapa variabel yang akan digunakan untuk pembuatan *player shoot* seperti, *float time, float FiringRate, bool shoot, string tag*, dan lain sebagainya. Pada *function Start* akan dilakukan pencarian terhadap sebuah *gameobject* yang akan digunakan untuk menggerakkan lokasi *spawn* dari peluru. Selanjutnya, pada *function FixedUpdate* akan dilakukan penyesuaian terhadap *boolean* dan melakukan perhitungan terhadap kecepatan menembak. Setelah itu, akan dilakukan pemeriksaan menggunakan *if statement*, jika *boolean shoot* yang artinya jika bernilai benar maka akan memanggil *function shoot* serta melakukan *reset* terhadap waktu.

Pada Kode 3.2 pada *function shoot* hal pertama yang akan dilakukan adalah mengambil objek dari *bullet pool*, selanjutnya akan dilakukan pemeriksaan jika *bullet* tidak *null* maka posisi dan rotasi dari *bullet* akan diubah sesuai dengan posisi dari *BulletPosition*. Langkah selanjutnya adalah mengaktifkan *bullet* agar bisa digunakan untuk menembak. Tahapan selanjutnya adalah pembuatan *script* untuk peluru. Pembuatan *script* ini bertujuan agar peluru bisa bergerak dan bisa memberikan *damage* terhadap lawan yang terkena, Kode 3.3 merupakan gambaran dari *script* peluru.

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class Bullets : MonoBehaviour
{

    [SerializeField] private float Speed;
    [SerializeField] private float BulletLifeTime;
    [SerializeField] private float BulletDamage;
    private Rigidbody _rigidbody;

    private void Awake()
    {
        _rigidbody = GetComponent<Rigidbody>();
    }
    private void Start()
    {
        ApplyVelocity();
    }

```

```
23      private void OnTriggerEnter(Collider other)
24      {
25          if (other.gameObject.tag == "Enemy")
26          {
27              if (other.gameObject != null)
28              {
29                  other.gameObject.GetComponent<EnemyHealth>().
    TakeDamage(BulletDamage);
30              }
31              hideBullet();
32          }
33      }
34      private void ApplyVelocity()
35      {
36          _rigidbody.velocity = Speed * transform.forward;
37      }
38
39      private void OnEnable()
40      {
41          Invoke("hideBullet", BulletLifeTime);
42      }
43
44      void hideBullet()
45      {
46          gameObject.SetActive(false);
47      }
48
49      private void OnDisable()
50      {
51          CancelInvoke();
52      }
53  }
```

Kode 3.3: Bullet script

Pada Kode 3.3 terdapat beberapa variabel yang digunakan seperti, *float Speed, float BulletLifeTime, BulletDamage* dan *rigidbody*. Tahapan pertama yang akan diproses adalah tahapan inisialisasi terhadap komponen *rigidbody*, selanjutnya adalah pada *function Start* akan memanggil *ApplyVelocity*. *ApplyVelocity* bertujuan untuk memberikan kecepatan terhadap peluru agar bisa bergerak ke arah depan. Selanjutnya, jika peluru memiliki kontak dengan musuh, maka akan memberikan *damage* terhadap darah musuh. Jika peluru sudah mengenai lawan maka peluru akan dihilangkan, peluru juga akan hilang jika tidak mengenai lawan dalam kurun

batas waktu yang sudah ditentukan. Tahapan selanjutnya adalah membuat darah pada lawan. Pada Kode 3.4 akan menunjukkan *script* mengenai darah lawan.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

public class EnemyHealth : MonoBehaviour
{
    public TextMeshProUGUI EnemyHealthText;
    [SerializeField] float CurrentHealth;
    [SerializeField] float MaxHealth;

    public GameObject KCO;

    KillCounter KillCounter;

    private void Start()
    {
        CurrentHealth = MaxHealth;
        EnemyHealthText.text = CurrentHealth.ToString();
        KillCounter = KCO.GetComponent<KillCounter>();
    }

    public void TakeDamage(float damage)
    {
        CurrentHealth -= damage;
        EnemyHealthText.text = CurrentHealth.ToString();
        if (CurrentHealth <= 0)
        {
            Debug.Log("Enemy Destroyed");
            KillCounter.AddKill();
            this.gameObject.SetActive(false);
        }
    }
}
```

Kode 3.4: Enemy health script

Pada Kode 3.4 terdapat beberapa variabel yang akan digunakan pada pembuatan *script enemy health* seperti, *TextMeshProUGUI EnemyHealthText, float CurrentHealth, float MaxHealth* dan *GameObject KCO*. Pada metode *Start* akan dilakukan inisialisasi terhadap darah sekarang dan terhadap indikator *text* yang akan

menampilkan informasi dari jumlah darah yang tersisa. Pada fungsi *TakeDamage* akan dipanggil ketika peluru mengenai musuh, pada fungsi ini darah musuh akan berkurang sesuai dengan *damage* yang diberikan oleh peluru serta akan melakukan pemeriksaan apakah darah sudah habis atau belum. Jika darah sudah habis atau sama dengan nol, maka otomatis musuh akan mati dan akan menambah angka skor *kill*. Skor *kill* ini nantinya akan digunakan sebagai *winning condition*. Pemain akan menang jika semua musuh sudah berhasil dikalahkan. Sebaliknya, pemain akan kalah jika ada musuh yang berhasil menyentuh pemain. Gambar 3.11 menunjukkan menu *game over*. Pemain bisa memilih untuk melakukan *restart* untuk mengulangi permainan atau menekan tombol *quit* untuk keluar dari permainan.

Gambar 3.11. *Game over*

3.3.3 Kendala yang dihadapi

Selama melaksanakan kerja magang terdapat beberapa kesulitan yang ditemukan, kendala berskala besar dan kecil, masing-masing. Berikut adalah beberapa tantangan selama pelaksanaan kerja magang.

1. Terjadinya beberapa error terhadap aplikasi dan sistem komputer.
2. Pembelajaran terhadap cara pembuatan fitur agar bisa berjalan dengan optimal.

3.3.4 Solusi dari kendala yang dihadapi

Dari berbagai kendala yang dihadapi dalam pelaksanaan kegiatan kerja magang yang dialami, Berikut beberapa solusi yang bisa dilakukan untuk mengatasi kendala tersebut.

1. Mencari solusi melalui internet, menonton video *tutorial* dan melakukan *update* serta *troubleshooting* terhadap aplikasi.
2. Bertanya kepada *supervisor* serta melakukan pencarian pada media internet.