UNIVERSITAS INDONESIA

Tower Defence
Game Design Document

Adi Nugroho (2306208546)

Alfonsus Tanara Gultom (2306267126) (**tidak kerja**)

Fauzan Farras Hakim Budi Handoyo (2306250610)

Ruben Kristanto (2306214624)

Fakultas Teknik

Teknik Komputer

2024

1. Pendahuluan
   1.1 Deskripsi Singkat

   Tower Defence merupakan game berbasis shooter, dimana pemain akan mengendalikan sebuah tower bersenjata untuk mengalahkan musuh-musuh yang ada. Terdapat beberapa tipe senjata yang dapat dipilih oleh pemain untuk menambah variasi permainan dengan 4 jenis musuh yang akan muncul secara acak dan mendekat ke player secara perlahan, jika sampai kepada sprite player musuh akan sedikit demi sedikit mengurangi hp player, selain itu terdapat 1 boss yang akan spawn enemy tanpa henti hingga boss tersebut mati.

   1.2 Asset yang digunakan

   https://foozlecc.itch.io/spire-tower-pack-1
   https://foozlecc.itch.io/spire-tower-pack-2
   https://foozlecc.itch.io/spire-tower-pack-3
   https://foozlecc.itch.io/spire-tower-pack-4
   https://foozlecc.itch.io/spire-tileset-1

2. Deskripsi Fitur
   2.1 Sistem Karakter Player

   Game Object Player akan memiliki 2 child object, yaitu Tower sebagai tampilan player serta Weapon sebagai senjata yang digunakan player. Weapon akan bervariasi tergantung dari tipe senjata apa yang dipilih di awal game. Selama permainan, player hanya akan diam di posisinya dan tidak bisa bergerak, namun masih dapat menembak peluru ke arah musuh untuk mengalahkannya. Pada script Player, terdapat implementasi singleton pattern pada playerInstance untuk memastikan hanya ada 1 player dalam game.

   Untuk peluru yang ditembak oleh object player diterapkan metode object pool dimana ada set object misal 9 peluru yang akan terus didaur ulang sehingga tidak terus menerus membuat object peluru.

   untuk setiap objek yang membutuhkan Health dan Defense(endurance) dapat dipasangkan dengan Health dan endurance component.

Ketika player mati scene akan diload ulang dan wave counternya akan kembali ke 0 untuk mempersiapkan permainan berikutnya.

```
public class Player : MonoBehaviour, IEndurance
{
    public static Player playerInstance { get; private set; }
    private PlayerLevel playerLevel;
    private EnduranceComponent enduranceComponent;

    [SerializeField] private int maxHealth = 100;
    [SerializeField] private int defense = 10;
    [SerializeField] private int currentHealth;

    public delegate void HealthChanged(int currentHealth, int
maxHealth);
    public event HealthChanged OnHealthChanged;

    private void Awake()
    {
        if (playerInstance == null)
        {
            playerInstance = this;
            DontDestroyOnLoad(gameObject);
        }
        else
        {
            Destroy(gameObject);
        }

        enduranceComponent =
GetComponentInChildren<EnduranceComponent>();
        if (enduranceComponent == null)
        {
            Debug.LogError("EnduranceComponent not found in child
object.");

            #if UNITY_EDITOR
                UnityEditor.EditorApplication.isPlaying = false;
            #endif
        }

        playerLevel = GetComponentInChildren<PlayerLevel>();
        if (playerLevel == null)
```

```
        {
            Debug.LogError("PlayerLevel component not found in
child object.");

            #if UNITY_EDITOR
                UnityEditor.EditorApplication.isPlaying = false;
            #endif
        }
    }

    private void Start()
    {
        currentHealth = maxHealth; // Initialize health at the
start
    }

    private void OnDestroy()
    {
        if (playerInstance == this)
        {
            playerInstance = null;
        }
    }

    public int GetMaxHealth()
    {
        return maxHealth;
    }

    public int GetCurrentHealth()
    {
        return currentHealth;
    }

    public int GetDefense()
    {
        return defense;
    }

    public void IncreaseMaxHealth(int amount)
    {
        maxHealth += amount;
```

```
        OnHealthChanged?.Invoke(currentHealth, maxHealth); //
Notify listeners
    }

    public void IncreaseDefense(int amount)
    {
        defense += amount;
    }

    public void RestoreHealthToMax()
    {
        currentHealth = maxHealth;
        OnHealthChanged?.Invoke(currentHealth, maxHealth); //
Notify listeners
    }

    public void TakeDamage(int damage)
    {
        int effectiveDamage = Mathf.Max(damage - defense, 0); //
Consider defense
        currentHealth = Mathf.Max(currentHealth -
effectiveDamage, 0); // Prevent health from going below 0

        OnHealthChanged?.Invoke(currentHealth, maxHealth); //
Notify listeners

        if (currentHealth <= 0)
        {
            Die();
        }
    }

    private void Die()
    {
        Debug.Log("Player has died.");

        WaveManager.Instance.ResetWaveNumber();

        SceneManager.LoadScene("MainMenu");

    }
```

```
    public void LevelUp()
    {
        if (playerLevel != null)
        {
            playerLevel.LevelUp(this);
        }
    }
}
```

2.2 Sistem Kontrol

Kontrol pada game ini berfokus pada mengendalikan arah tembakan dari senjata milik player. Player akan menggunakan kursor mouse yang akan berfungsi sebagai target dari peluru milik player. Jika tombol kiri mouse ditekan, maka senjata player akan menembakkan peluru ke posisi kursor saat menembak.

Pertama-tama dalam script MouseControl digunakan function SetCursor() untuk menetapkan texture, mode, dan koordinat titik tengah kursor dan dijalankan ketika start sehingga hanya sekali saja dijalankannya.

```
public class MouseControl : MonoBehaviour
{
    [SerializeField] private Texture2D crosshairTexture; // using
Texture2D because SetCursor method requires it
    private Vector2 cursorHotspot = Vector2.zero; // hotspot is
the point in the cursor that is the target point
    private CursorMode cursorMode = CursorMode.Auto; // set to
auto so that the cursor will be displayed as the OS default
cursor

    // Start is called before the first frame update
    private void Start()
    {
        SetCursor();
```

```
    }

    // method to set the cursor display to use the crosshair
texture
    private void SetCursor()
    {
        if (crosshairTexture != null)
        {
            Cursor.SetCursor(crosshairTexture, cursorHotspot,
cursorMode);
        }
    }
}
```

Kemudian dalam script CombatManager, function update mengimplementasikan mengecek input dari mouse berupa left click dan penerapan jeda antar tembakan jika input dari mouse terlalu cepat. Tetapi sebelum itu diambil referensi dari player object pada Start() function.

```
public class CombatManager : MonoBehaviour
{
    [SerializeField] private Transform playerTransform;
    private float shootCooldown = 0.2f;
    private float nextShootTime;
    private Weapon currentWeapon;
    private Animator weaponAnimator;

    private void Start()
    {
        playerTransform = GameObject.Find("Player").transform;
    }

    // Update is called once per frame
    private void Update() // shooting need to be responsive so
its using Update() instead of FixedUpdate()
```

```
    {
        if (Input.GetKeyDown(KeyCode.Mouse0))
        {
            weaponAnimator?.SetBool("isShooting", true);
            Shoot();
        }

        if (Input.GetKey(KeyCode.Mouse0) && Time.time >=
nextShootTime) // Assuming "Fire1" is configured in Input
settings
        {
            Shoot();
        }

        if (Input.GetKeyUp(KeyCode.Mouse0))
        {
            weaponAnimator?.SetBool("isShooting", false);
        }
    }
```

2.4 Sistem Musuh

Musuh dikendalikan oleh 2 script yaitu enemy manager dan enemy behavior, enemy manager berfungsi untuk mengatur wave dan spawn enemy sedangkan enemy behavior berfungsi untuk mengatur collision dari object enemy yang dibuat dengan bullet dari player maupun enviroment seperti object player. ketika enemy bertemu dengan bullet maka enemy akan menerima damage dan healthnya akan berkurang jika sampai 0 enemy akan mati dan object akan dihapus, disini kode tidak reuse object seperti jika menggunakan object pool, tetapi langsung menghapus dan membuat object enemy baru .

Enemy akan dispawn randomly dengan parameter jarak tertentu sehingga tidak spawn out of bounds atau diluar kamera.

Enemy manager membuat array list enemy yang aktif, sehingga tidak akan ada musuh yang spawn dengan jumblah tak terhingga. Selain batasan tersebut kode ini juga mengimplementasikan scaling difficulty dimana jumblah enemy akan terus bertambah di setiap wave sebanyak dua enemy.

pada function startwave, Wave diatur dengan Wave Manager yang akan mengambil jumblah wave yang sudah terjadi pada awal function kemudian mengincrementnya setiap kali wave selesai.

kode:
```

```
public class EnemyManager : MonoBehaviour
{
    [Header("Enemy Spawners")]
    [SerializeField] private Transform[] spawnPoints; // Array of
spawn points
    [SerializeField] private List<GameObject> enemyPrefabs; //
Enemy prefabs to spawn

    [Header("Wave System")]
    [SerializeField] private int totalEnemies = 4; // Base number
of enemies per wave

    private List<GameObject> activeEnemies = new
List<GameObject>(); // Track active enemies
    private bool isWeaponSelected = false; // Tracks if a weapon
has been selected

    private void Update()
    {
        // Wait until the weapon is selected before starting
waves
        if (!isWeaponSelected) return;

        // Check if all enemies are killed to start the next wave
        activeEnemies.RemoveAll(enemy => enemy == null); //
Remove null references (destroyed enemies)
        if (activeEnemies.Count == 0)
        {
            StartWave();
        }
    }

    private void StartWave()
    {
        int waveNumber = WaveManager.Instance.GetWaveNumber();
        Debug.Log($"Starting Wave: {waveNumber}");
```
```

```
        for (int i = 0; i < totalEnemies; i++)
        {
            SpawnEnemy(enemyPrefabs[0]); // Assuming index 0 is
the basic enemy
        }

        totalEnemies += 2; // Increase enemies per wave for
scaling difficulty
        WaveManager.Instance.IncrementWave(); // Notify
WaveManager to increment the wave
    }

    private void SpawnEnemy(GameObject enemyPrefab)
    {
        // Select a random spawn point
        Transform spawnPoint = spawnPoints[Random.Range(0,
spawnPoints.Length)];

        // Set a fixed z position (e.g., z = 1)
        Vector3 spawnPosition = spawnPoint.position;
        spawnPosition.z = 1f;

        // Instantiate the enemy prefab at the chosen spawn point
        GameObject spawnedEnemy = Instantiate(enemyPrefab,
spawnPosition, spawnPoint.rotation);

        // Add the spawned enemy to the active enemies list (if
applicable)
        activeEnemies.Add(spawnedEnemy);
    }

    // Public method to notify that a weapon has been selected
    public void SetWeaponSelected(bool selected)
    {
        isWeaponSelected = selected;
        Debug.Log("Weapon has been selected! Enemies will now
start spawning.");
    }
}
```

```
```

```
public class EnemyBehavior : MonoBehaviour
{
    [SerializeField] private float speed = 3f;          // Movement speed of the enemy
    [SerializeField] private int health = 50;          // Enemy's health
    [SerializeField] private int attackDamage = 10;    // Damage dealt to the player on collision

    private Transform player;                          // Reference to the player's transform
    private bool isNearWall = false;                   // To check if the enemy is near the invisible wall

    private void Start()
    {
        // Find the player by tag
        GameObject playerObject = GameObject.FindGameObjectWithTag("Player");
        if (playerObject != null)
        {
            player = playerObject.transform;
        }
        else
        {
            Debug.LogError("Player not found in the scene! Make sure the Player GameObject has the tag 'Player'.");
        }
    }

    private void Update()
    {
        // Follow the player unless the enemy is near the invisible wall
        if (player != null && !isNearWall)
        {
            FollowPlayer();
        }
    }

    private void FollowPlayer()
    {
```

```
        // Move towards the player
        transform.position =
Vector3.MoveTowards(transform.position, player.position, speed *
Time.deltaTime);
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("InvisibleWall"))
        {
            // Stop the enemy from moving closer to the player
            isNearWall = true;
        }
        else if (collision.CompareTag("Player"))
        {
            // When the enemy collides with the player, deal
damage to the player
            Player playerScript =
collision.GetComponent<Player>();
            if (playerScript != null)
            {
                playerScript.TakeDamage(attackDamage); // Reduce
player's health
                Debug.Log("Player hit! Dealt " + attackDamage + "
damage.");

            }
        }
        else if (collision.CompareTag("PlayerBullet")) // Check
if the enemy is hit by a bullet
        {
            Bullet bullet = collision.GetComponent<Bullet>();
            if (bullet != null)
            {
                TakeDamage(bullet.GetAttackDamage());
                Destroy(collision.gameObject); // Destroy the
bullet upon hitting the enemy
            }
        }
}


    private void OnTriggerExit2D(Collider2D collision)
```

```
    {
        // Allow the enemy to move again if it leaves the wall
area
        if (collision.CompareTag("InvisibleWall"))
        {
            isNearWall = false;
        }
    }

    public void TakeDamage(int damageAmount)
    {
        // Reduce enemy health
        health -= damageAmount;

        if (health <= 0)
        {
            Die();
        }
    }

    private void Die()
    {
        Debug.Log("Enemy has died.");
        Destroy(gameObject); // Destroy the enemy GameObject
    }
}
```

```

2.5 Boss Battle

Dalam Tower Defence, boss battle merupakan scene melawan boss akhir dari project ini, boss akan memunculkan musuh-musuh dengan interval tertentu tanpa henti hingga boss tersebut sendiri mati, boss tersebut sama seperti tower yaitu statis tidak bergerak sama sekali.

berikut adalah implementasi kode:

Boss Manager

```public class BossManager : MonoBehaviour
{
    [Header("Boss Settings")]
```
```

```
    [SerializeField] private GameObject bossPrefab; // Boss
prefab to spawn
    [SerializeField] private Transform spawnPoint; // Boss spawn
point

    [Header("Basic Enemy Settings")]
    [SerializeField] private GameObject[] basicEnemyPrefabs; //
Array of basic enemy prefabs
    [SerializeField] private Transform[] enemySpawnPoints; //
Spawn points for basic enemies
    [SerializeField] private float spawnInterval = 5f; // Time
interval for spawning basic enemies

    [Header("Scene Settings")]
    [SerializeField] private string mainSceneName = "MainBattle";
// Name of the main battle scene

    private GameObject bossInstance;
    private bool isBossAlive = true; // Tracks if the boss is
alive

    private void Start()
    {
        SpawnBoss();
        StartCoroutine(EnemySpawnerRoutine());
    }

    private void SpawnBoss()
    {
        if (bossPrefab == null || spawnPoint == null)
        {
            Debug.LogError("BossPrefab or SpawnPoint not
assigned.");
            return;
        }

        // Spawn the boss
        bossInstance = Instantiate(bossPrefab,
spawnPoint.position, spawnPoint.rotation);
        Debug.Log("Boss spawned.");
    }

    private IEnumerator EnemySpawnerRoutine()
```

```
    {
        while (isBossAlive)
        {
            yield return new WaitForSeconds(spawnInterval);
            SpawnBasicEnemies();
        }
    }

    private void SpawnBasicEnemies()
    {
        if (basicEnemyPrefabs.Length == 0 ||
enemySpawnPoints.Length == 0)
        {
            Debug.LogError("BasicEnemyPrefabs or EnemySpawnPoints
not assigned.");
            return;
        }

        for (int i = 0; i < 2; i++) // Spawn two basic enemies
        {
            // Select a random enemy prefab
            GameObject randomEnemyPrefab =
basicEnemyPrefabs[Random.Range(0, basicEnemyPrefabs.Length)];

            // Select a random spawn point
            Transform spawnPoint =
enemySpawnPoints[Random.Range(0, enemySpawnPoints.Length)];

            // Instantiate the enemy
            Instantiate(randomEnemyPrefab, spawnPoint.position,
spawnPoint.rotation);
        }

        Debug.Log("Basic enemies spawned by the boss.");
    }

    public void OnBossDefeated()
    {
        Debug.Log("Boss defeated! Returning to the main battle
scene...");
        isBossAlive = false; // Stop the spawning routine
        SceneManager.LoadScene(mainSceneName); // Return to the
main battle scene
```

```
    }
}
```

Boss Behaviour

```
public class BossBehavior : MonoBehaviour
{
    [SerializeField] private int health = 1000;          // Boss
health
    [SerializeField] private int attackDamage = 10;     // Damage
dealt to the player on collision

    private Transform player;                            //
Reference to the player's transform
    private bool isNearWall = false;                     // To
check if the boss is near the invisible wall

    private void Start()
    {
        // Find the player by tag
        GameObject playerObject =
GameObject.FindGameObjectWithTag("Player");
        if (playerObject != null)
        {
            player = playerObject.transform;
        }
        else
        {
            Debug.LogError("Player not found in the scene! Make
sure the Player GameObject has the tag 'Player'.");
        }
    }

    private void Update()
    {
        // No movement logic here to make the boss static
    }

    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.CompareTag("Player"))
        {
```

```
            // When the boss collides with the player, deal
damage to the player
            Player playerScript =
collision.GetComponent<Player>();
            if (playerScript != null)
            {
                playerScript.TakeDamage(attackDamage); // Reduce
player's health
                Debug.Log("Player hit! Dealt " + attackDamage + "
damage.");
            }
        }
        else if (collision.CompareTag("PlayerBullet")) // Check
if the boss is hit by a bullet
        {
            Bullet bullet = collision.GetComponent<Bullet>();
            if (bullet != null)
            {
                TakeDamage(bullet.GetAttackDamage());
                Destroy(collision.gameObject); // Destroy the
bullet upon hitting the boss
            }
        }
    }

    public void TakeDamage(int damageAmount)
    {
        // Reduce boss health
        health -= damageAmount;

        if (health <= 0)
        {
            Die();
        }
    }

    private void Die()
    {
        Debug.Log("Boss has been defeated!");
        Destroy(gameObject); // Destroy the boss GameObject
    }
}
```

3. Kesimpulan

   Project Tower Defence merupakan implementasi dari prinsip OOP yang telah dipelajari selama semester ini. Dengan mengimplementasikan design pattern serta berbagai practice pemrograman yang baik, program yang dihasilkan diharapkan dapat scalable, secure, low coupling serta memiliki fokus yang jelas untuk setiap bagiannya. Untuk kedepannya, kekurangan seperti bug ataupun fitur yang belum lengkap akan menjadi bahan evaluasi kami.