Лабораторна робота №6

2D игра Unity: Classic Tower Defense

Мета: ознайомитися з методами створення гри у жанрі Tower Defense.

Література

Документація Unity3D https://docs.unity3d.com/ScriptReference

Зміст роботи

Завдання 1. *Створити гру Tower Defense.*

1. Створити прототип гри Tower Defense з примітивних фігур:
   - 1.1. Свторити потрібні префаби.
   - 1.2. Створити сцену гри.
   - 1.3. Створити систему ворогів.
   - 1.4. Створити систему башень та їхньої стрільби.
   - 1.5. Додати контролер гри.
   - 1.6. Додати та налаштувати користувацький інтерфейс.
2. Покращити візуальну частину гри шляхом додання спрайтів та анімацій.

Завдання 2. *Додати музичний супровід (звук знищення ворога, звук створення вежі тощо) до створеної гри.*

Методичні рекомендації

!*Це лише один з способів створити гру у жанрі Tower Defense****!***

Для комфортної розробки гри, створимо файлову систему (рис.1).

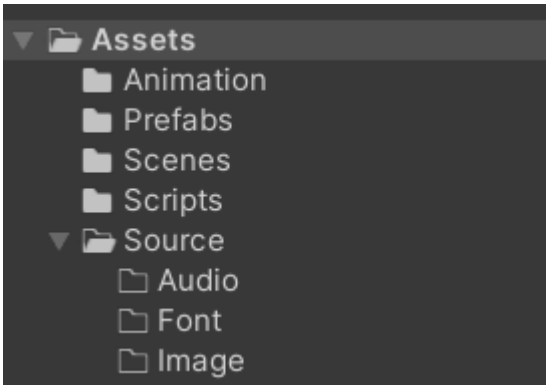


Рис.1 Файлова система проекту

Додамо на сцену наступні елементи: задній фон, місце для башні, ворога, базу, снаряд та шлях противника. Для кращого сприйняття того, що відбувається на сцені, ворога можна створити з простого спрайту (рис.2).

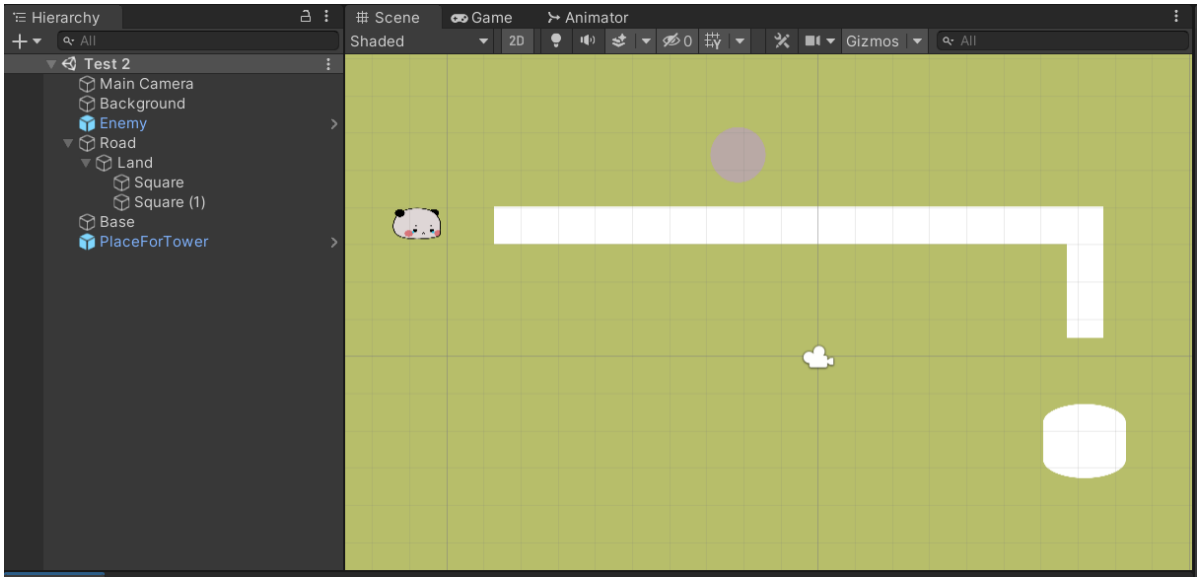


Рис.2 Сцена з примітивами.

Розробка системи ворог. Надамо можливість ворогу рухатись. В такому типі ігор, противники рухаються по точках. Від точки А до точки Б. Тому додамо на сцену Empty як точки шляху (рис.3).

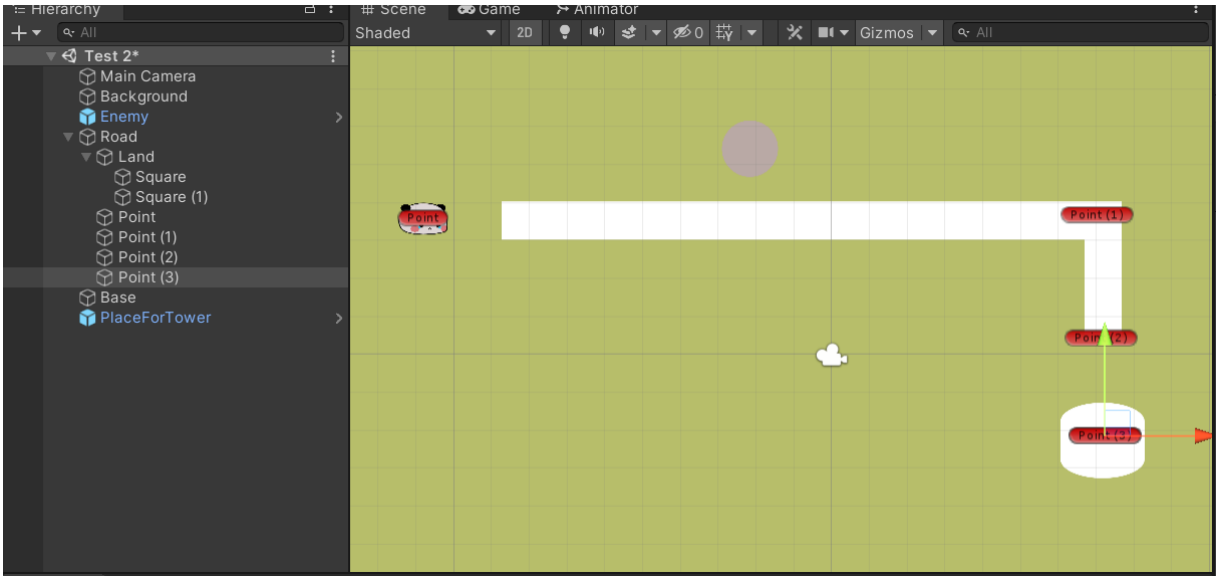


Рис.3 Точки для руху ворогра.

Для комфортної роботи з елементами типу Empty в інспекторі виберемо зручну іконку (рис.4). Переіменуємо точки відповідно до їх порядку.

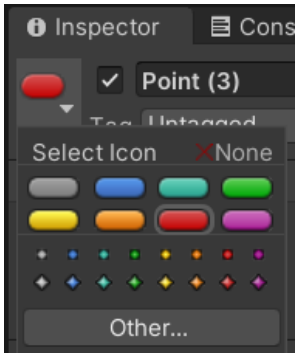


Рис. 4. Вибір іконки.

На противника додамо компонент Rigidbody 2D та Collider 2D. У Rigidbody 2D змінимо значення Gravity Scale на 0.

Перейдемо до написання сценарію для противника. Створимо відповідний C# скрипт.

Для вирішення питання руху між двома точками підійде метод Lerp, що присутній у Vector3. Для написання скрипту потрібні наступні дані (рис.5):

- Масив з точок шляху.
- Дві точки між якими здійснюється рух.
- Дожина шляху між двома точками.
- Час, що буде потрачений на подолання шляху.
- Час, коли було досягнено точку.
- Швидкість руху.
- Поточна точка.

```
public Transform[] points;
private int currentPoint = 0;
[SerializeField] private float moveSpeed = 2.0f;

private Vector3 startPosition;
private Vector3 endPosition;
private float pathLength;
private float totalTimeForPath;
private float lastWaypointSwitchTime;
```

Рис. 5. Об’явлення змінних.

При старті присвоємо значення зміній, що відповідає за останій час зміни точки(так як з неї починається відлік часу) та визвемо функцію що буде обчислювати дистанцію та час, що буде потрачений на нього. Також дана функція присваютиме значення точки А та точки Б (startPosition/endPosition).

Для обчислення дистанції між точками використаємо метод Distance, що приймає у себе дві позиції. Для обчислення часу поділимо всю довжину шляху на швидкість (рис.6).

```
Сообщение Unity | Ссылок: 0
void Start()
{
    lastWaypointSwitchTime = Time.time;
    distanceCalculation();
}
Ссылок: 2
private void distanceCalculation()
{
    //Обчислюємо координати двух точок, між якими буде здійснюватись рух
    startPosition = points[currentPoint].position;
    endPosition = points[currentPoint + 1].position;
    //Визначаємо відстань між ними, та час що буде потрачений на нього
    pathLength = Vector2.Distance(startPosition, endPosition);
    totalTimeForPath = pathLength / moveSpeed;
}
```

Рис 6. Лістинг коду

Щоб змусити об’єкт рухатись використаємо уже відомий нам метод Lerp у функції FixedUpdate.

Для цього будемо взнавати поточний час об’єкту, що рухається для співвідношення з загальним часом, що буде витрачено на весь шлях. Для того щоб перевірити чи досяг об’єкт кінцевої точки використаємо метод Equals, що перевіряє на тотожність між точками. Якщо так, то змінимо поточну точку на 1, зазначимо час досягнення точки та знову викличимо функцію обрахунку потрібних нам даних (рис.7).

```
void FixedUpdate()
{
    float currentTimeOnPath = Time.time - lastWaypointSwitchTime;
    gameObject.transform.position = Vector2.Lerp(startPosition, endPosition, currentTimeOnPath / totalTimeForPath);

    if (gameObject.transform.position.Equals(endPosition)) {
        if (currentPoint < points.Length - 2)
        {
            currentPoint++;
            lastWaypointSwitchTime = Time.time;
            distanceCalculation();

        }
    }
}
```

Рис.7. Лістинг коду

Прикріпимо скрипт до противника. В інспекторі в компоненті скрипту заповнимо відповідні поля, а саме у відповідному порядку додамо у масив точки та вкажемо швидкість руху ворога. Натиснемо кнопку Play та перевіримо коректність роботи скрипту (рис.8)

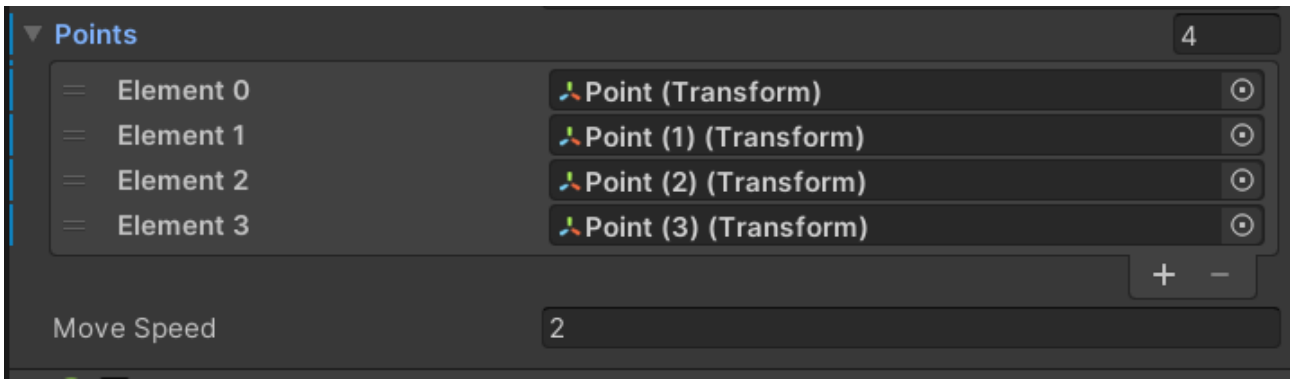


Рис. 8. Налаштування руху

Також потрібно дописати функцію, що буде повертати спрайт у напрямку до наступної точки. У ній обчислимо кут повороту та за допомогою функції AngleAxis повернемо наш об’єкт у потрібну сторону. Додамо виклик функції у функцію FixedUpdate у блоці перевірки на досягнення точки противником (рис.9).

```
private void RotateIntoMoveDirection()
{

    Vector3 newDirection = (endPosition - startPosition);

    float x = newDirection.x;
    float y = newDirection.y;
    float rotationAngle = Mathf.Atan2(y, x) * 180 / Mathf.PI;

    gameObject.transform.rotation = Quaternion.AngleAxis(rotationAngle, Vector3.forward);
}
Сообщение Unity | Ссылок: 0
void FixedUpdate()
{
    float currentTimeOnPath = Time.time - lastWaypointSwitchTime;
    gameObject.transform.position = Vector2.Lerp(startPosition, endPosition, currentTimeOnPath / totalTimeForPath);

    if (gameObject.transform.position.Equals(endPosition)) {
        if (currentPoint < points.Length - 2)
        {
            currentPoint++;
            lastWaypointSwitchTime = Time.time;
            distanceCalculation();
            RotateIntoMoveDirection();
        }
    }
}
```

Рис. 9. Лістинг коду

Перевіримо правильність роботи скрипту, перейшовши до режиму гри. Додамо об’єкту тег Enemy.

Також ворог повинен мати кількість очок здоров’я та кількість золота, що буде отримано у разі його знищення вистрілами башні.

```
public int hp = 3;
public float coinAfterKill = 10;
```

Лістинг скрипту Enemy:

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

public class Enemy : MonoBehaviour
{
    public int hp = 3;
    public float coinAfterKill = 10;

    public Transform[] points;
    private int currentPoint = 0;
    [SerializeField] private float moveSpeed = 2.0f;

    private Vector3 startPosition;
    private Vector3 endPosition;
    private float pathLength;
    private float totalTimeForPath;
    private float lastWaypointSwitchTime;

    void Start()
    {
        lastWaypointSwitchTime = Time.time;
        distanceCalculation();
    }
    private void distanceCalculation()
```

```
    {
        //Обчислюємо координати двух точок, між якими буде здійснюватись рух
        startPosition = points[currentPoint].position;
        endPosition = points[currentPoint + 1].position;
        //Визначаємо відстань між ними, та час що буде потрачений на нього
        pathLength = Vector2.Distance(startPosition, endPosition);
        totalTimeForPath = pathLength / moveSpeed;
    }
    private void RotateIntoMoveDirection()
    {

        Vector3 newDirection = (endPosition - startPosition);

        float x = newDirection.x;
        float y = newDirection.y;
        float rotationAngle = Mathf.Atan2(y, x) * 180 / Mathf.PI;

        gameObject.transform.rotation = Quaternion.AngleAxis(rotationAngle,
Vector3.forward);
    }
    void FixedUpdate()
    {
        float currentTimeOnPath = Time.time - lastWaypointSwitchTime;
        gameObject.transform.position = Vector2.Lerp(startPosition, endPosition,
currentTimeOnPath / totalTimeForPath);

        if (gameObject.transform.position.Equals(endPosition)) {
            if (currentPoint < points.Length - 2)
            {
                currentPoint++;
                lastWaypointSwitchTime = Time.time;
                distanceCalculation();
                RotateIntoMoveDirection();
            }
        }
    }
}
```

Приступимо до створення кулі. Створимо відповідний C# скрипт. Рух кулі буде здійснюватись аналогічно руху противника, відмінністю буде лиш те, що куля буде постійно змінювати кінцеву точку, тобто переслідувати противника.

Отже, об’явлемо портібні нам змінні. У ролі кінцевої точки буде ігровий об’єкт Enemy. Функція FixedUpdate міститиме у собі перевірку на те, чи існує переслідуваємий об’єкт, якщо так, то здійснюватиметься рух до нього, в іншому випадку – знищення кулі. У функції старт ми знаходимо елемент GameManager, що буде потрібний нам у подальшій розробці, а поки закоментуйте його.

На об’єкт кулі додамо компонент Collider 2D та змінимо значення is Trigger на true, шляхом натискання на відповідний чекбокс.

```
private GameContoler gameContoler;
private Vector3 startPosition;
[SerializeField] public GameObject target;
private float pathLength;
private float totalTimeForPath;
private float startTime;
private float moveSpeed = 5f;

Сообщение Unity | Ссылок: 0
void Start()
{
    gameContoler = GameObject.Find("GameManager").GetComponent<GameContoler>();
    startPosition = gameObject.transform.position;
    startTime = Time.time;
}
```

Рис. 10. Лістинг коду

```
void FixedUpdate()
{
    if (target != null)
    {
        pathLength = Vector2.Distance(startPosition, target.transform.position);
        totalTimeForPath = pathLength / moveSpeed;
        float currentTimeOnPath = Time.time - startTime;
        gameObject.transform.position = Vector2.Lerp(startPosition, target.transform.position, currentTimeOnPath / totalTimeForPath);
    }
    else
    {
        Destroy(gameObject);
    }
}
```

Рис. 11. Лістинг коду

Скористаємось функцією OnTriggerEnter2D для перевірки зіткнення кулі з цілею. Перевіримо чи є об’єкт переслідувальним противником, далі візьмемо за допомогою методу GetComponent<> отримаємо доступ до інформації ворога та віднімемо одне очко здоров’я. Якщо здоров’я рівне або нижче нуля, знищуємо об’єкт та додаємо до існуючих доступної ігрової валюти ціну за взнищення ворога (але поки закоментуємо цю стрічку допоки не буде створено відповідного сценарію).

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if (collision.gameObject == target)
    {
        Enemy enemyDate = target.GetComponent<Enemy>();
        int targetHP = enemyDate.hp--;
        if (targetHP <= 0)
        {
            gameContoler.gold += enemyDate.coinAfterKill;
            Destroy(target);

        }
        Destroy(gameObject);
    }
}
```

Рис.12. Лістинг коду

Прикріпимо скрипт до відповідного об’єкту. У поле target зазначте об’єкт противника та перевірте чи переслідує його куля.

Створіть копії об’єкту Enemy та налаштуйте їх як нові типи противників, змінивши колір для зручності. Перетяніть об’єкти Enemy та Bullet у папку Prefabs для створення префабів противників та кулі.

!**Будьте обережні, координати об’єктів повині бути відцентровані (0)**!

Лістинг сценарію Bullet:

```
using UnityEngine;

public class Bullet : MonoBehaviour
{

    private GameContoler gameContoler;
    private Vector3 startPosition;
    [SerializeField] public GameObject target;
    private float pathLength;
    private float totalTimeForPath;
    private float startTime;
    private float moveSpeed = 5f;

    void Start()
    {
        gameContoler = GameObject.Find("GameManager").GetComponent<GameContoler>();
        startPosition = gameObject.transform.position;
        startTime = Time.time;
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject == target)
        {
            Enemy enemyDate = target.GetComponent<Enemy>();
            int targetHP = enemyDate.hp--;
            if (targetHP <= 0)
            {
                gameContoler.gold += enemyDate.coinAfterKill;
                Destroy(target);

            }
            Destroy(gameObject);
        }
    }

    void FixedUpdate()
    {
        if (target != null)
        {
            pathLength = Vector2.Distance(startPosition, target.transform.position);
            totalTimeForPath = pathLength / moveSpeed;
            float currentTimeOnPath = Time.time - startTime;
            gameObject.transform.position = Vector2.Lerp(startPosition,
target.transform.position, currentTimeOnPath / totalTimeForPath);
        }
        else
        {
            Destroy(gameObject);
        }
    }
}
```

Перейдемо до створення системи башень. Створимо відповідний сценарій та перейдемо до його написання. Спершу об’явимо змінні, що нам стануть у нагоді. Башні матимуть рівні, ціну для покупки та на їх покращення та

швидкість вистрілів. Також створимо масив з ігрових об’єктів, що будуть слугувати рівнями башень. Додамо поле, що прийматиме у себе префаб кулі для вистрелів. Для відсліджування ворогів створимо список з ігрових об’єктів.

```
private GameContoler gameContoler;
[SerializeField] private int towerLVL = 0;
[SerializeField] private float goldForLVLUp;
[SerializeField] private float shootInterval;

[SerializeField] private GameObject[] towers;

[SerializeField] private List<GameObject> enemies;
[SerializeField] private GameObject bullet;
```

Рис.13. Об’явлення змінних.

На башню добавимо компонент Circle Collider 2D та збільшимо його радіус. Він слугуватиме радіусом досягнення до цілей.

За допомогою функцій OnTriggerEnter2D і OnTriggerExit2D додаватимемо до списку противників та видалятимемо.

```
Сообщение Unity | Ссылок: 0
private void OnTriggerEnter2D(Collider2D collision)
{
    enemies.Add(collision.gameObject);
}
Сообщение Unity | Ссылок: 0
private void OnTriggerExit2D(Collider2D collision)
{
    enemies.Remove(collision.gameObject);
}
```

Рис.14. Лістинг коду

Для реалізації стрільби скористаємось корутинами. Якщо в списку присутній ворог, створюватиметься об’єкт куля і в параметр target передаватиметься перший ворог у списку.

```
Ссылок: 2
IEnumerator Shoot() {
    yield return new WaitForSeconds(shootInterval);
    if (enemies.Count > 0 && towerLVL >0)
    {
        GameObject newBullet = Instantiate(bullet, gameObject.transform);
        newBullet.GetComponent<Bullet>().target = enemies[0];
    }
    StartCoroutine(Shoot());
}
```

Рис.15. Лістинг коду.

За допомогою функції OnMouseUp реалізуємо можливість покупки та покращення башні. Для початку в функції Start отримаємо компонент GameControler у якому зберігатиметься дані про наявні кошти.

При натискані на об’єкт перевірятиметься наступне:

- Чи достатньо коштів для покупки або покращення
- Чи башня не має останій рівень покращення

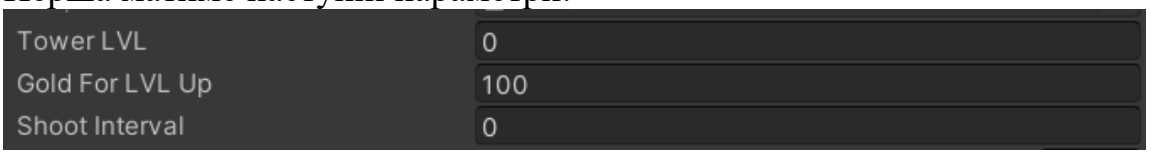
Якщо так, то об’єкт замінятиметься на наступний. Створюється башня вищого рівня, а поточна знищується.

```
private void OnMouseUp()
{
    if (gameContoler.gold >= goldForLVLUp && towerLVL < towers.Length-1)
    {
        gameContoler.gold -= goldForLVLUp;
        towerLVL++;
        GameObject newTower = Instantiate(towers[towerLVL], transform.position, Quaternion.identity);
        Destroy(gameObject);
    }
}
```

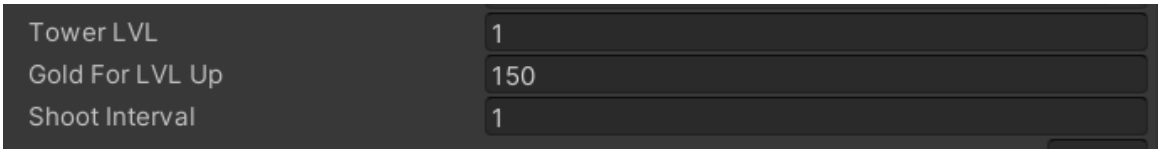
Рис.16. Лістинг коду

Перенесемо скрипт на башню та створимо ще дві її копії (Ctrl+D)

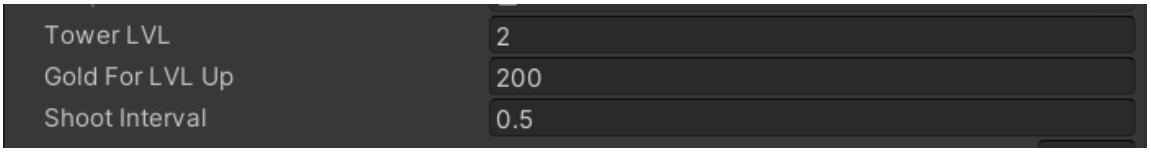
Перша матиме наступні параметри:



Друга:



Тертя:



Параметри налаштовуйте як ви забажаєте. Перша башня слугує лише місцем для будови інших (не має рівня та інтервал стільби).

Перенесемо ці об’єкти у папку Prefabs, змінивши колір для зручності у розробці.

!**Будьте обережні, координати об’єктів повині бути відцентровані (рівні 0)**!

Повернемось до параметрів. Перенесемо у масив створені префаби башень та додамо префаб кулі у поле Bullet

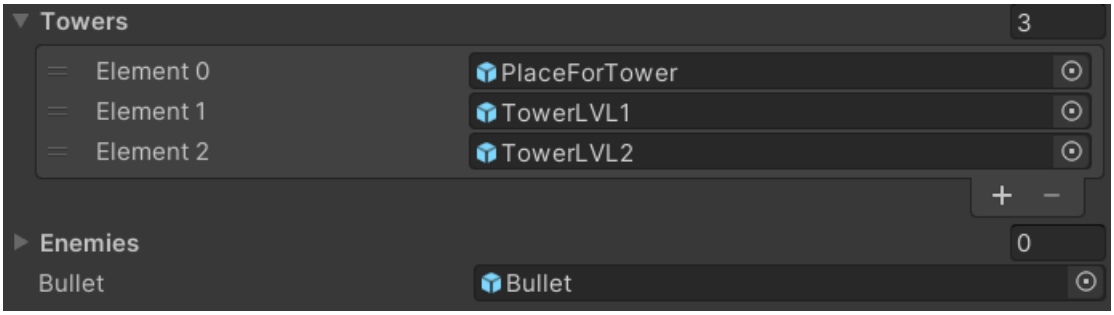


Рис.17. Налаштування башень

Повторіть аналогічно з іншими двома об’єктами башень. У скрипті закоментуйте поля зв’язані зі GameControler.

Лістинг сценарію Tower:

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

public class Tower : MonoBehaviour
{
    private GameContoler gameContoler;
    [SerializeField] private int towerLVL = 0;
    [SerializeField] private float goldForLVLUp;
    [SerializeField] private float shootInterval;

    [SerializeField] private GameObject[] towers;

    [SerializeField] private List<GameObject> enemies;
    [SerializeField] private GameObject bullet;

    private void Start()
    {
        gameContoler = GameObject.Find("GameManager").GetComponent<GameContoler>();
        StartCoroutine(Shoot());
    }

    private void OnMouseUp()
    {
        if (gameContoler.gold >= goldForLVLUp && towerLVL < towers.Length-1)
        {
            gameContoler.gold -= goldForLVLUp;
            towerLVL++;
            GameObject newTower = Instantiate(towers[towerLVL], transform.position,
Quaternion.identity);
            Destroy(gameObject);
        }
    }
    IEnumerator Shoot() {
        yield return new WaitForSeconds(shootInterval);
        if (enemies.Count > 0 && towerLVL >0)
        {
            GameObject newBullet = Instantiate(bullet, gameObject.transform);
            newBullet.GetComponent<Bullet>().target = enemies[0];
        }
        StartCoroutine(Shoot());
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        enemies.Add(collision.gameObject);
    }
    private void OnTriggerExit2D(Collider2D collision)
    {
        enemies.Remove(collision.gameObject);
    }
}
```

Створимо Empty об’єкт та назвемо його GameManager, створимо скрипт GameControler, у якому об’явимо поля currentWave, health та gold з публічним доступом. Також додамо поля gameOver та gameWon (рис.18).

```
public bool gameOver = false;
public bool gameWon = false;
public int health;
public float gold;
public int currentWave = 0;
```

Рис.18. Об’явлення змінних

Перетянемо скрипт на створений об’єкт та налаштуємо кількість здоров’я та коштів. Розкоментуйте поля що мають зв’язок з GameConroler у зарання створених скриптах. Перевірте можливість покупки та покращення башень.

Перейдемо до створення системи бази. На відповідний об’єкт додамо компонент Collider 2D, параметр is Trigger змінимо на true. Створимо скрипт, де вже за допомогою функції OnTriggerEnter2D буде здійснюватись віднімання очків здоров’я.

Лістниг сценарію Base:

```
using UnityEngine;

public class Base : MonoBehaviour
{
    private GameContoler gameManager;

    private void Start()
    {
        gameManager = GameObject.Find("GameManager").GetComponent<GameContoler>();
    }
    private void OnTriggerEnter2D(Collider2D collision)
    {
        if (collision.gameObject.tag == "Enemy") {
            if (!gameManager.gameOver)
            {
                gameManager.health--;
            }
            Destroy(collision.gameObject);
        }
    }
}
```

Приступимо до розробки генератора противників. Для цього створимо відповідний скрипт та додамо на об’єкт GameManager. Перед основним класом створимо клас Wave, що матиме у собі особливості хвилі нападу, а саме: кількість простих, швидких та товстих противників, а також інтервал їх генерації (рис.19).

```
[System.Serializable]
ссылка: 1
public class Wave
{
    public int primaryEnemy;
    public int fastEnemy;
    public int fatEnemy;
    public float spawnInterval;
}
```

Рис.19. Клас Wave

Створимо портібні нам поля, а саме масив з можливих противників, точок їх руху, масив класу Wave, підключимо скрипт gameControler, додамо поля для підрахунку кількості згенерованих окремого виду противників та загальну кількість створених, кількість можливих противників певного виду, отсаній час їх створнення, тип ворога та час між хвилями.

```
[SerializeField] private GameObject[] enemies;
[SerializeField] private Transform[] points;

[SerializeField] private Wave[] waves;

private GameContoler gameContoler;

private int countEnemy;
private int spawnedEnemy;
private float lastSpawnTime;
private int typeEnemy = 0;
private int enemyCounter = 0;
private float timeBetweenWaves = 5;
```

Рис.20. Об’явлення змінних

У функції Start виконаємо настпуні дії:

```
void Start()
{
    gameContoler = gameObject.GetComponent<GameContoler>();
    lastSpawnTime = Time.time;
}
```

Рис.21. Лістинг коду

Генерація ворогів відбуватиметься у блоці фукції Update. Спершу додамо перевірку чи не завершена гра. Далі дізнаємось поточну хвилю з сценарію GameControler. Перевіримо, чи поточна хвиля не виходить за максимальну кількість хвиль. Якщо так, то підрахуємо всю кількість ворогів за поточну хвилю. Добавимо оператор switch, що оприділятиме кількість противників відповідного типу.

Обрахуємо поточний часовий інтервал у порівняні з останньою генерацією противника та об’явимо змінну, що прийматиме інтервал між генераціями ворогів поточної хвилі.

```
if (!gameContoler.gameOver)
{
    int currentWave = gameContoler.currentWave;
    if (currentWave < waves.Length)
    {

        int maxEnemies = waves[currentWave].primaryEnemy + waves[currentWave].fastEnemy + waves[currentWave].fatEnemy;
        switch (typeEnemy)
        {
            case 0: countEnemy = waves[currentWave].primaryEnemy; break;
            case 1: countEnemy = waves[currentWave].fastEnemy; break;
            case 2: countEnemy = waves[currentWave].fatEnemy; break;
        }
        float timeInterval = Time.time - lastSpawnTime;
        float spawnInterval = waves[currentWave].spawnInterval;
        if (((spawnedEnemy == 0 && timeInterval > timeBetweenWaves) || (spawnedEnemy != 0 && timeInterval > spawnInterval)) && enemyCounter < countEnemy)
        {
            lastSpawnTime = Time.time;
            GameObject newEnemy = Instantiate(enemies[typeEnemy], points[0].position, Quaternion.identity);
            newEnemy.GetComponent<Enemy>().points = points;
            enemyCounter++;
            spawnedEnemy++;
            if (enemyCounter >= countEnemy)
            {
                typeEnemy++;
                enemyCounter = 0;
            }
        }
```

Рис.22. Лістинг коду

За допомогою перевірки, контролюватимемо дотримання інтервалу між хвилями та генераціями противника, також контролювати кількість згенерованих противників окремих видів. У цьому блоці створюватимемо ігровий об’єкт Enemy та передаватимемо йому у параметр точки для руху. При досягненні максимальної кількості поточного виду ворога, перейдемо до наступного.

Далі перевірятимемо чи згенеровано максимальну кількість противників та чи є об’єкти з тегом Enemy на сцені. Якщо так, то переходимо до наступної хвилі.

```
if (spawnedEnemy >= maxEnemies && GameObject.FindGameObjectWithTag("Enemy") == null)
{
    gameContoler.currentWave++;
    gameContoler.gold = Mathf.RoundToInt(gameContoler.gold * 1.1f);
    spawnedEnemy = 0;
    enemyCounter = 0;
    typeEnemy = 0;
    lastSpawnTime = Time.time;
}
```

Рис.23. Лістинг коду

Якщо було досягнено максимальної кількості хвиль, то гравець перемагатиме.

```
else {
    gameContoler.gameWon = true;
}
```

Рис.24. Лістинг коду

Налаштуйте елемент за своїм бажанням.

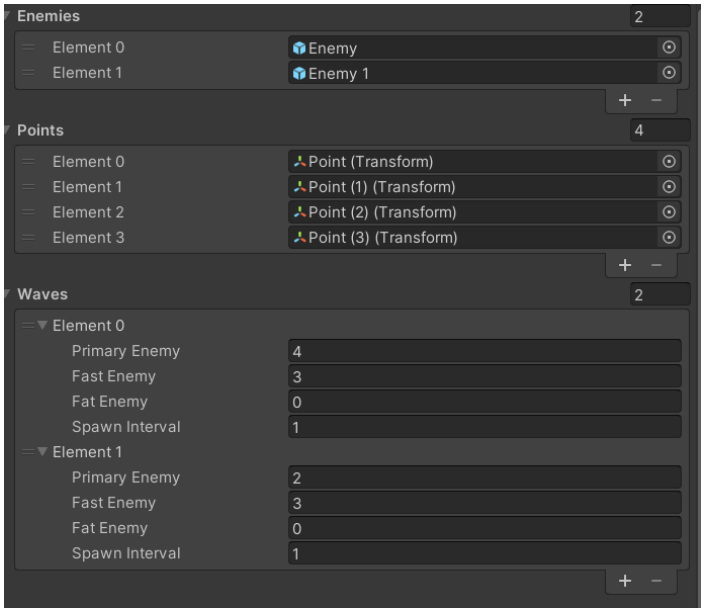


Рис.25. Приклад налаштування

Лістинг скрипту SpawnEnemy:

```
using UnityEngine;
[System.Serializable]
public class Wave
{
    public int primaryEnemy;
    public int fastEnemy;
    public int fatEnemy;
    public float spawnInterval;
}

public class SpawnEnemy : MonoBehaviour
{
    [SerializeField] private Transform spawnpoint;
    [SerializeField] private GameObject[] enemies;
    [SerializeField] private Transform[] points;
    [SerializeField] private Wave[] waves;

    private GameContoler gameContoler;

    private int countEnemy;
    private int spawnedEnemy;
    private float lastSpawnTime;
    private int typeEnemy = 0;
    private int enemyCounter = 0;
    private float timeBetweenWaves = 5;

    void Start()
    {
        gameContoler = gameObject.GetComponent<GameContoler>();
        lastSpawnTime = Time.time;
    }

    void Update()
    {
        if (!gameContoler.gameOver)
        {
            int currentWave = gameContoler.currentWave;
            if (currentWave < waves.Length)
            {

                int maxEnemies = waves[currentWave].primaryEnemy +
waves[currentWave].fastEnemy + waves[currentWave].fatEnemy;
```

```
                switch (typeEnemy)
                {
                    case 0: countEnemy = waves[currentWave].primaryEnemy; break;
                    case 1: countEnemy = waves[currentWave].fastEnemy; break;
                    case 2: countEnemy = waves[currentWave].fatEnemy; break;
                }
                float timeInterval = Time.time - lastSpawnTime;
                float spawnInterval = waves[currentWave].spawnInterval;
                if (((spawnedEnemy == 0 && timeInterval > timeBetweenWaves) || (spawnedEnemy
!= 0 && timeInterval > spawnInterval)) && enemyCounter < countEnemy)
                {
                    lastSpawnTime = Time.time;
                    GameObject newEnemy = Instantiate(enemies[typeEnemy],
spawnpoint.position, Quaternion.identity);
                    newEnemy.GetComponent<Enemy>().points = points;
                    enemyCounter++;
                    spawnedEnemy++;
                    if (enemyCounter >= countEnemy)
                    {
                        typeEnemy++;
                        enemyCounter = 0;
                    }
                }
                if (spawnedEnemy >= maxEnemies && GameObject.FindGameObjectWithTag("Enemy")
== null)
                {
                    gameContoler.currentWave++;
                    gameContoler.gold = Mathf.RoundToInt(gameContoler.gold * 1.1f);
                    spawnedEnemy = 0;
                    enemyCounter = 0;
                    typeEnemy = 0;
                    lastSpawnTime = Time.time;
                }
            }
            else {
                gameContoler.gameWon = true;
            }
        }
    }
}
```

Створіть сцену рівеня гри (рис.26).

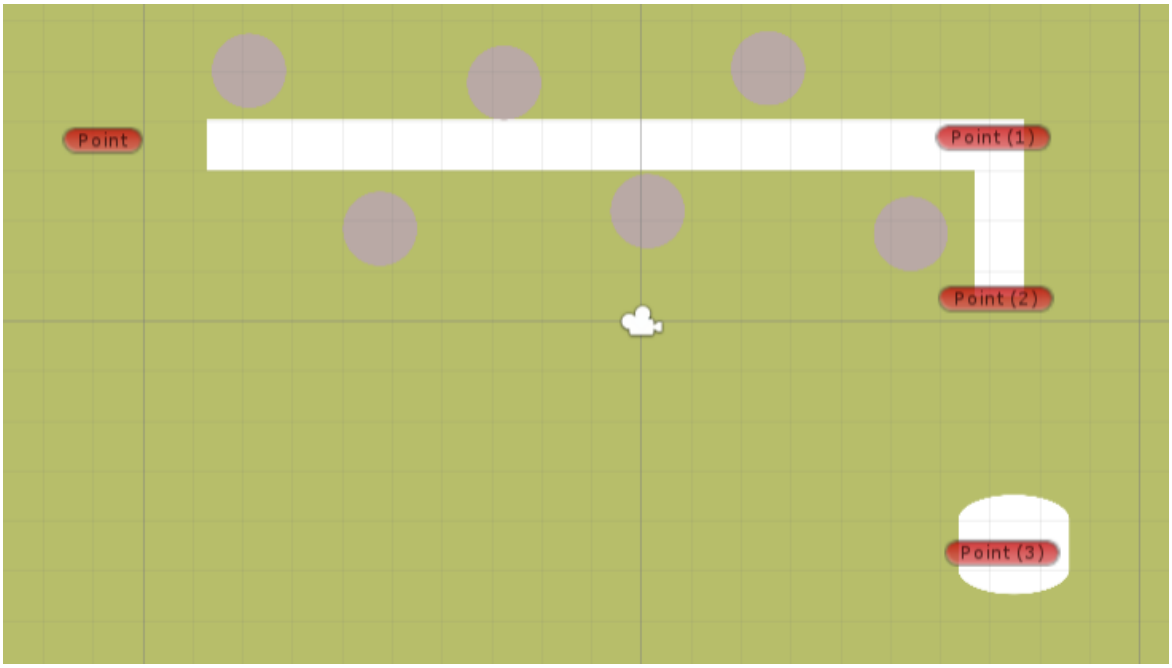


Рис.26. Приклад рівня гри

Перейдіть до режиму гри та перевірте на правильність її роботи.

Удоскональте гру спрайтами та анімаціями. Додайте користувацький інтерфейс та звуковий супровід.

Для прикладу додання звукового супроводу додамо звук при знищені ворога. Перенесемо звук на префаб противника та виключимо параметр Play in Awake (рис.27).

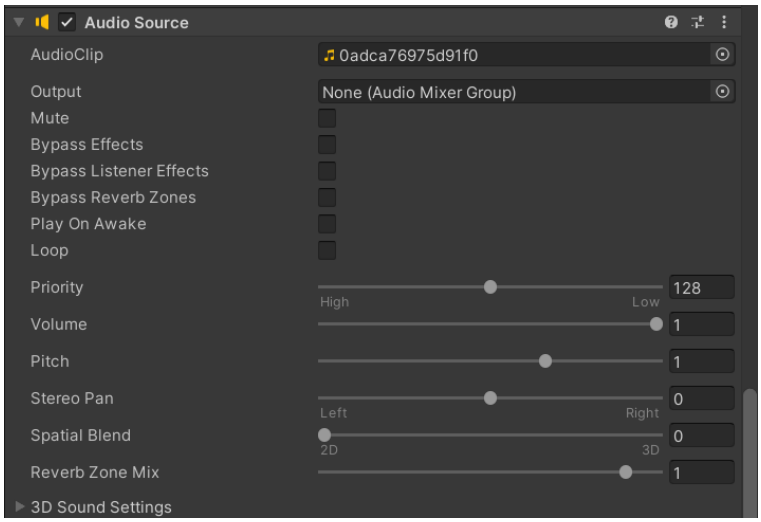


Рис.27. Налаштування аудіофайлу

У скрипт Bullet допишемо запуск звку при знищені ворога (рис.28).

```
AudioSource audioSource = target.GetComponent<AudioSource>();
AudioSource.PlayClipAtPoint(audioSource.clip, transform.position);
```

Рис.28. Лістинг коду

Контрольні питання:

1. Які особливості жанру «Tower Defense»?
2. Як створити об’єкт на сцені в процесі гри?
3. Як створити шлях, по якому будуть рухатися вороги?
4. Як додати звуковий супровід у гру?

Самостійна робота:

Ось кілька ідей, які можна розвинути на основі того, що ви зробили:

– Створити більше типів противників.

– Більше типів веж.

– Розробити декілька ворожих шляхів.

– Створити різні рівні гри.