中華大學資訊工程學系

104 學年度專題製作期末報告

應用手勢在手機遊戲 APP 之開發

指導老師：梁秋國教授

專案組員：

B10102070 陳孟君

B10102078 邱勢棠

B10102080 宋竑慶

中華民國一百零五年一月

目錄

第一章 序論

1.1 研究動機

現在的手機遊戲百百種，有卡牌類、解謎類、戰鬥類等，然而想要在這些遊戲中脫穎而出是一件非常難的事情，不是要有強大的美術做背景，就是要有豐富又有趣的故事內容，再者則是令人意想不到的特殊玩法，而我們這個團隊想要的則是第三個“特殊玩法”，於是開始了找資料的旅程，後來發現ＲＰＧ類型的遊戲比較符合我們的胃口，在這其中又以動作類（ＡＲＰＧ）的最想做，因此就決定了我們專題一開始的方向。

在特殊的方法這一塊我們在眾多遊戲中找到了一款遊戲，他遊玩的方式是在手機螢幕上畫出對應的圖形，這時我們就想到一個點子“把手勢加進遊戲中，使得角色放招是透過手勢”，而這個點子一出來就獲得了全數人的同意，因為我們平常玩的ＡＲＰＧ大多是搖桿加按紐，其中搖桿是角色移動，按鈕是放出技能，這樣的方式我們覺得有些枯燥乏味，感覺玩家與遊戲間的互動沒有很多，然而透過手勢讓玩家放技能不再只是單純的點點手指就可以了，而是要畫上對映到的圖案才能，這樣的想法應該是一種前所未有的突破。

1.2 研究目標

現在的３Ｄ遊戲大多是按鈕，所以我們決定要用手勢來做代替，讓玩家可以更進一步跟遊戲做互動，使我們做出來的遊戲更加有趣，以下是我們增加手勢後所產生的效果：

1. 增加趣味性

   手勢讓遊戲不無聊，同時讓玩家感到不會像現在一些遊戲一樣的膩。

2. 增加玩家與遊戲的互動

   一般來說都是按下按鈕來發動技能，沒有與技能做連結的感覺，而透過手勢來讓玩家覺得此技能是自己所「畫」出來的，而不是單純點點手指而已。

第二章 開發環境

2.1 開發平台 - Unity5.3

Unity 是一套跨平台的遊戲引擎，可開發執行於 PC、Mac OS 單機遊戲，或是 iOS、Android 手機或平板電腦的遊戲。Unity 提供了人性化的操作介面，支援 PhysX 物理引擎、粒子系統，並且提供網路多人連線的功能，不需要學習複雜的程式語言，符合遊戲製作上的各項需求。Unity 大幅降低了遊戲開發的門檻，即使是個人工作室製作遊戲也不再是夢想。

2.2 使用語言

Visual Studio 2013 C#

2.3 硬體設備

Android 手機、Windows PC、MacBook Air

第三章 專案分工

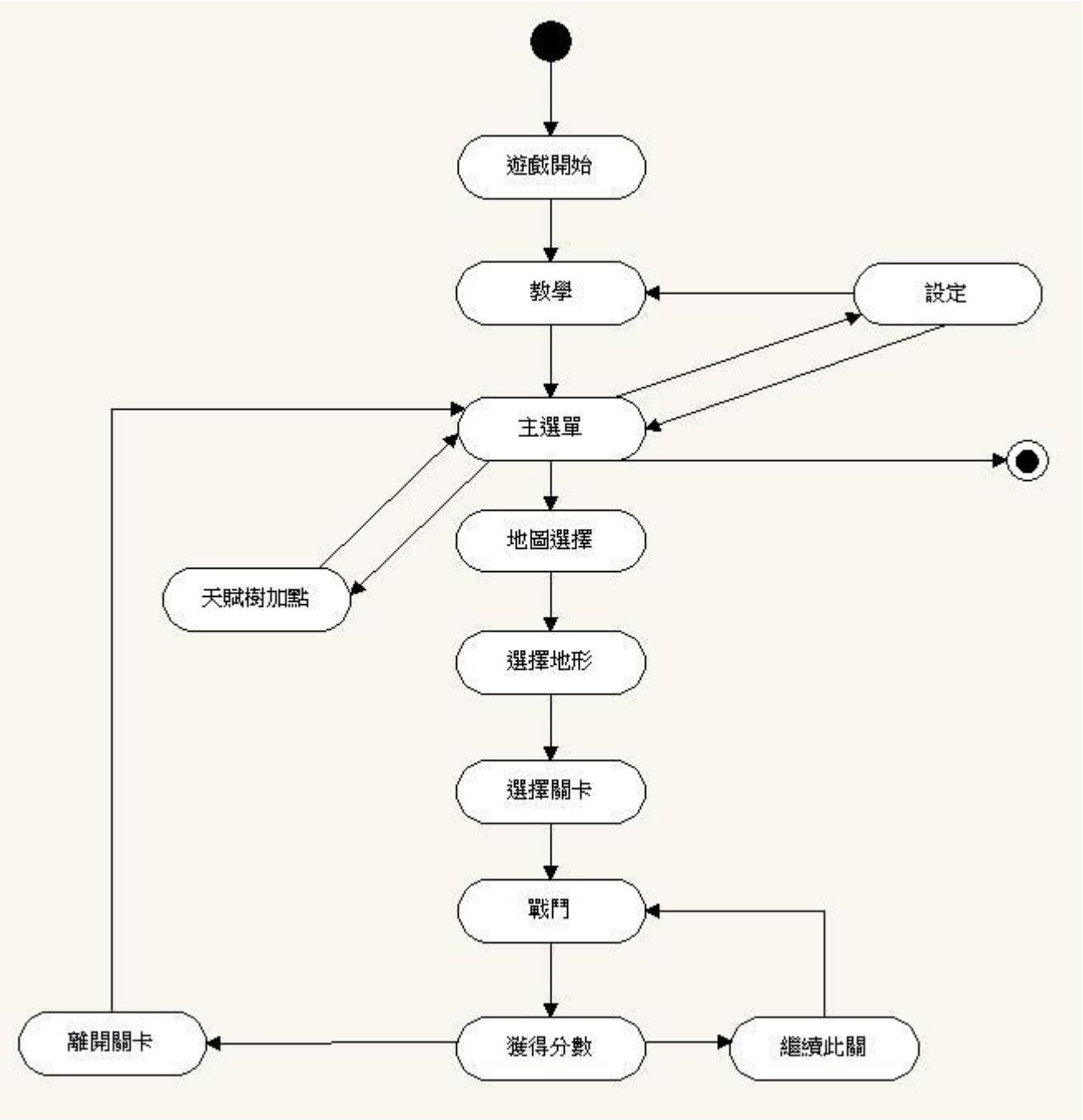
組長：陳孟君 程式撰寫、人物動作設定、專題報告
組員：邱勢棠 程式撰寫、美工、AI 製作
組員：宋竑慶 3D 模組製作、人物動畫

時間	104/01	104/02	104/03	104/04	104/05	104/06
陳孟君	學 IOS		學 COCOS-2D(搖桿+手勢)			
邱勢棠	學 IOS		學 COCOS-Studio			
宋竑慶	學 2D 建模		學習 MAYA			

時間	104/07	104/08	104/09	104/10	104/11	104/12	105/01
陳孟君	學習 Unity			撰寫手勢+搖桿		合併/除錯	
邱勢棠	學習 Unity			撰寫 AI		合併/除錯	
宋竑慶	學習 MAYA		製作 3D 模型				除錯

第四章 遊戲流程圖

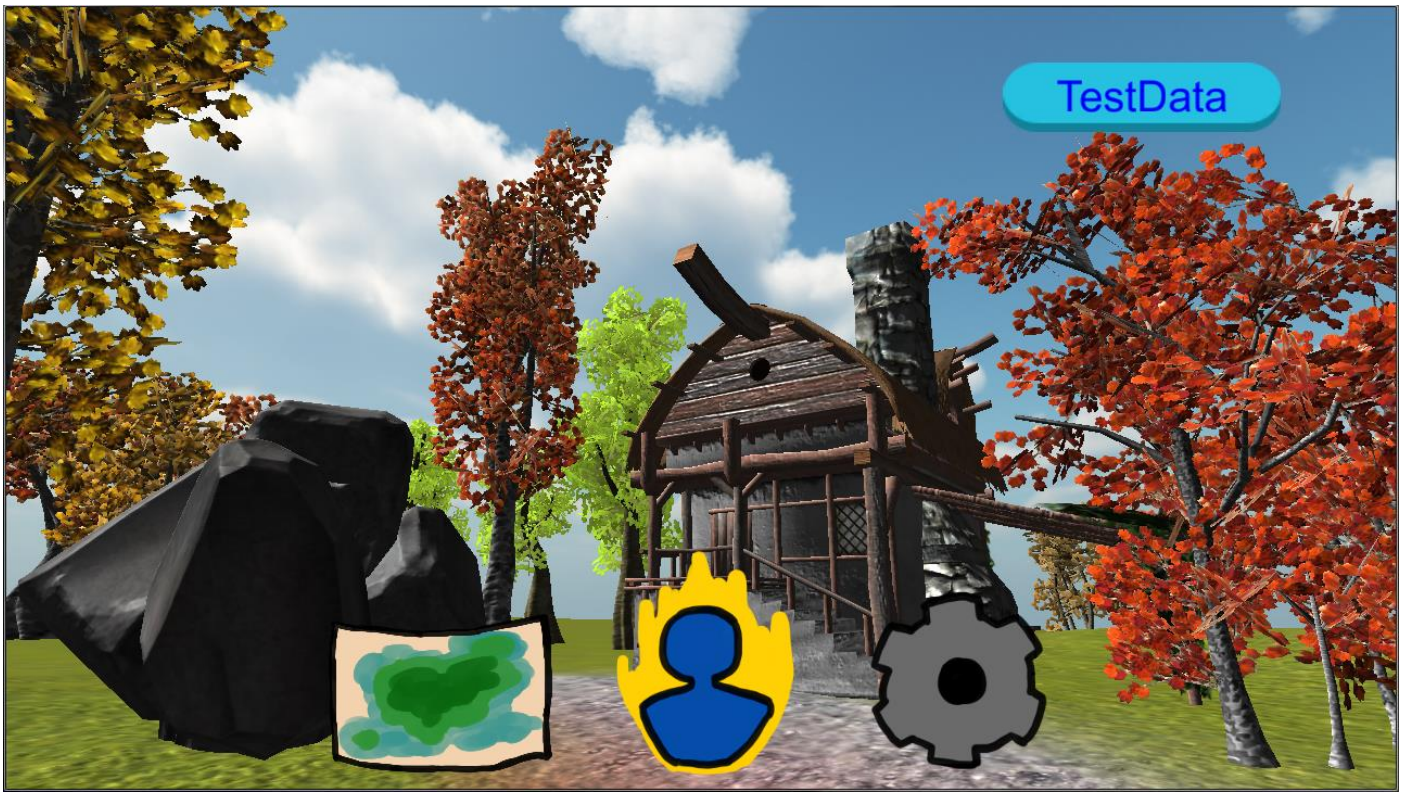
4.1 遊戲流程圖

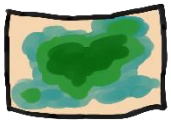


4.2 系統功能

主選單

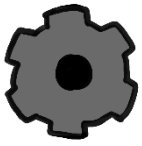
簡單明瞭易操作





點擊後前往地圖選單

點擊後前往技能選單



點擊後出現設定選單

教學畫面

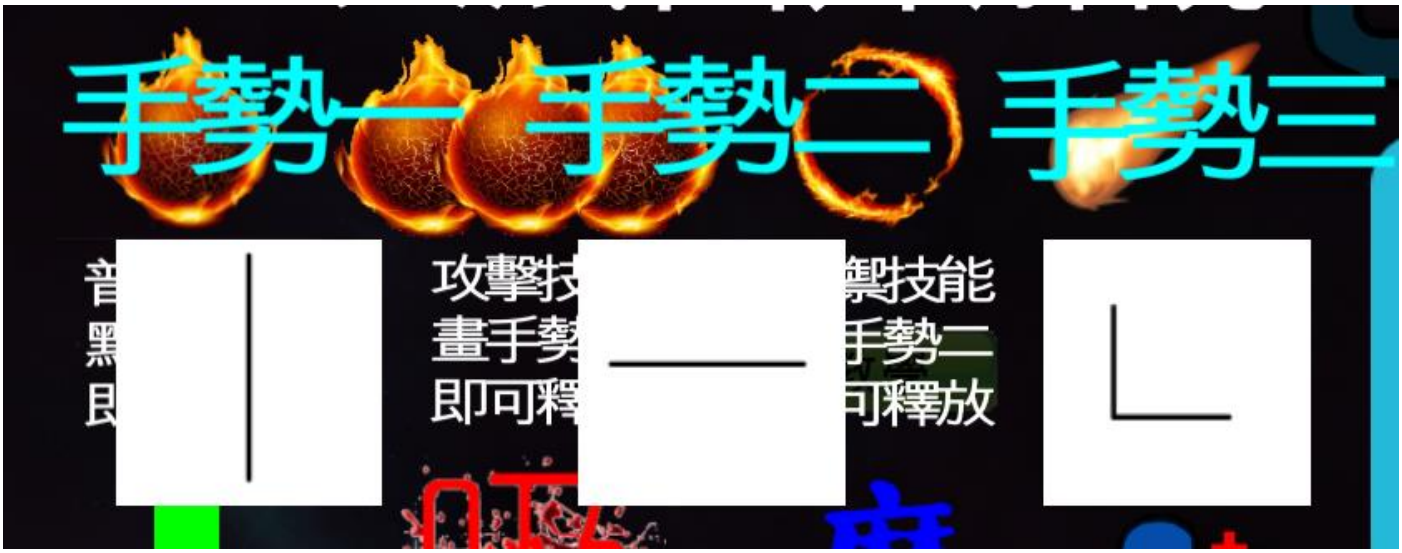

技能選單教學
主畫面教學
地圖畫面教學

技能選單教學


天賦圖片解說
返回
手勢圖
普通攻擊
點擊螢幕
即可釋放
攻擊技能
畫手勢一
即可釋放
防禦技能
畫手勢二
即可釋放
大決技能
畫手勢三
即可釋放
增加血量
吸血
回魔
突破上限
(角色能力上升)
地圖畫面教學

手勢圖

點擊後顯示手勢圖，再一次點擊手勢圖會消失



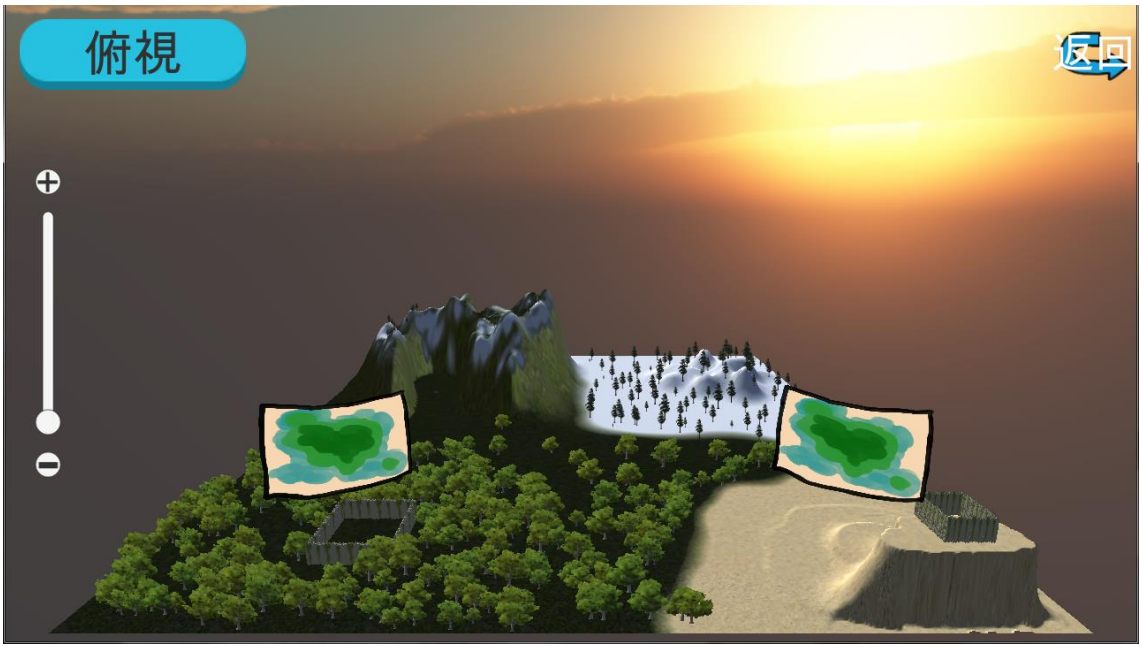
主畫面教學

地圖畫面教學


地圖圖示解說
返回
20
選擇等級
(數字是怪物等級)
俯視/側視
可以切換
視角的按鈕
選擇地形
有鎖頭代
表未解鎖
地圖畫面教學

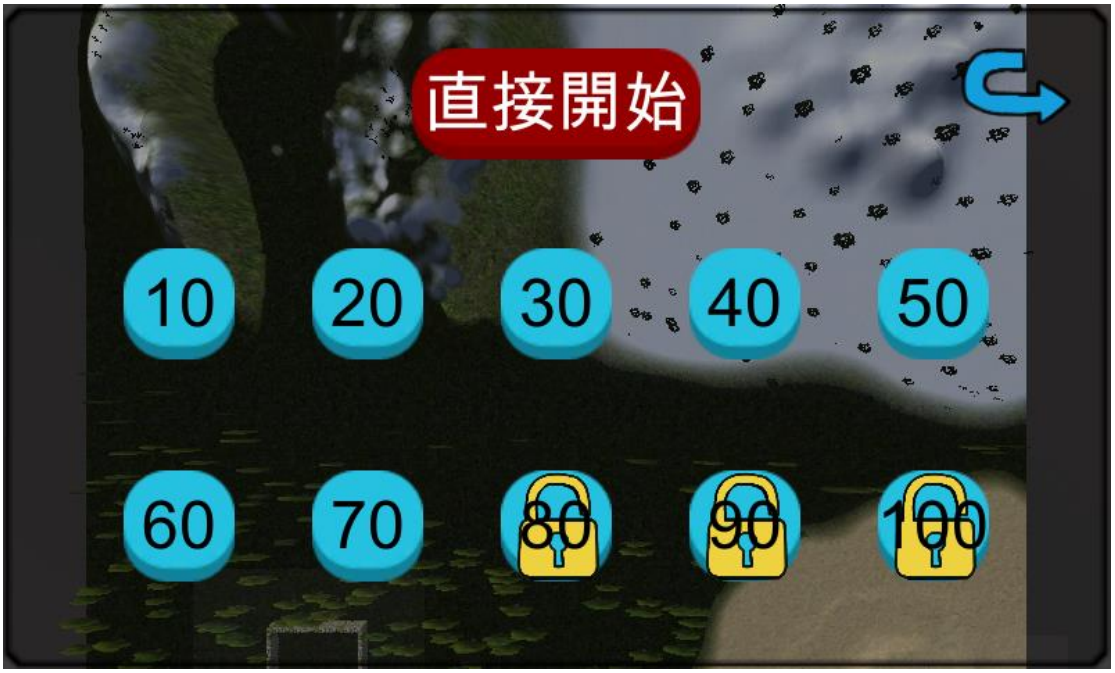
地圖選單

精緻 3D 模型地圖



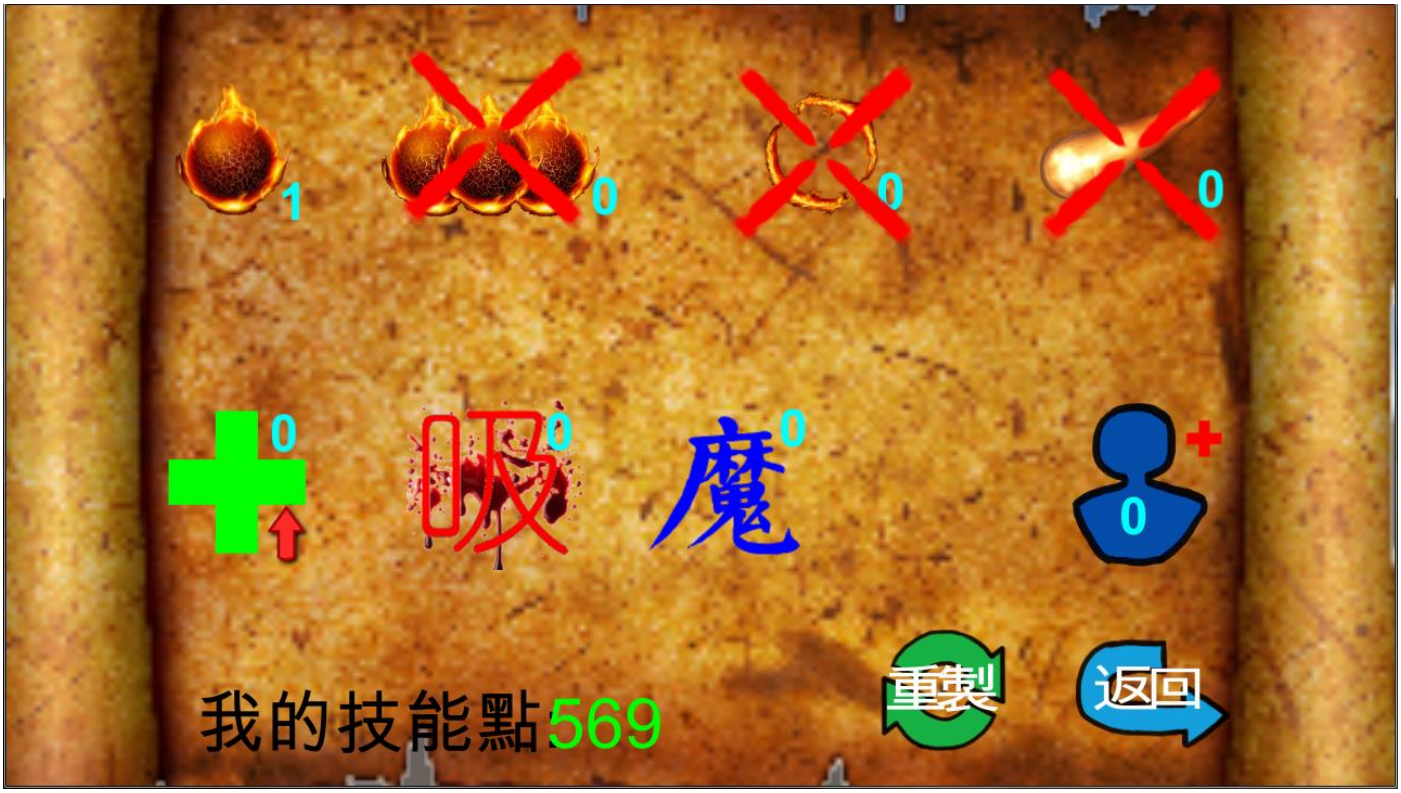
關卡選單

點擊圈選部分的圍牆會出現關卡選單



有鎖頭的關卡表示尚未達到此等級的分數

技能選單



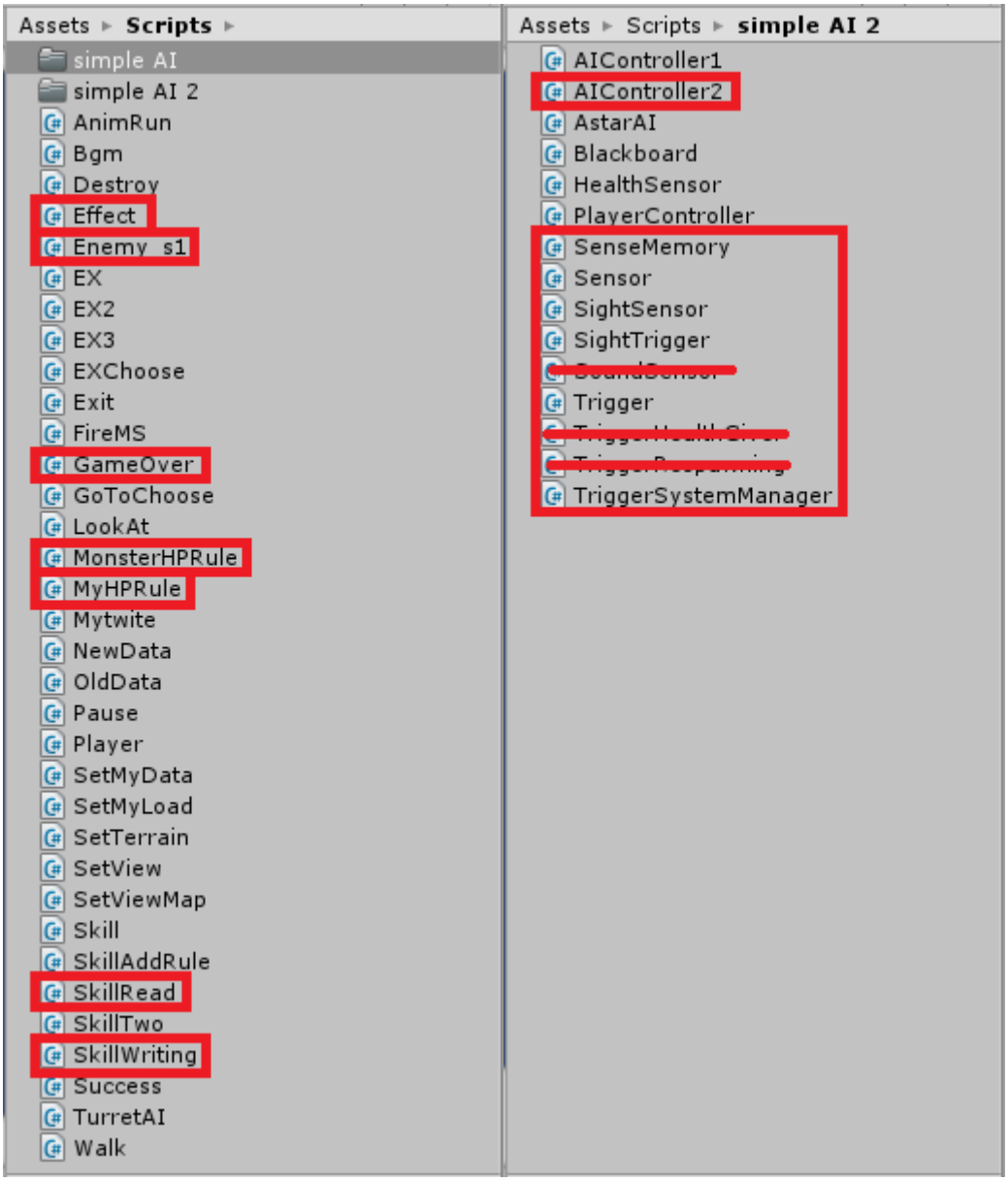
1. 初始點數 10 點給玩家自由分配
2. 除基本攻擊外還有三招強力技能可以學習
3. 如果點滿所有技能方可突破上限，使角色更加強大
4. 玩家如果點錯技能可以使用重新分配點數來初始化

普通攻擊　分散火球　無敵火圈

流星火雨　血量上升　攻擊吸血

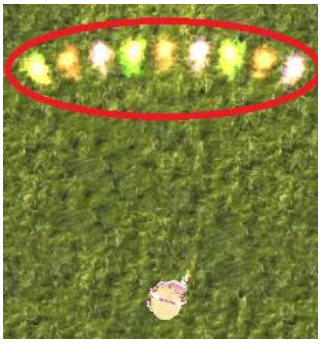
魔　回魔速度　突破上限

第五章 專案內容


Assets ▸ Scripts ▸
simple AI
simple AI 2
AnimRun
Bgm
Destroy
Effect
Enemy_s1
EX
EX2
EX3
EXChoose
Exit
FireMS
GameOver
GoToChoose
LookAt
MonsterHPRule
MyHPRule
Mytwite
NewData
OldData
Pause
Player
SetMyData
SetMyLoad
SetTerrain
SetView
SetViewMap
Skill
SkillAddRule
SkillRead
SkillTwo
SkillWriting
Success
TurretAI
Walk
Assets ▸ Scripts ▸ simple AI 2
AIController1
AIController2
AstarAI
Blackboard
HealthSensor
PlayerController
SenseMemory
Sensor
SightSensor
SightTrigger
Trigger
TriggerSystemManager

Effect.cs

技能特效



Enemy_s1.cs

隨機生成怪物

GameOver.cs

計算所獲得分數

MonsterHPRule.cs

怪物血量計算

MyHPRule.cs

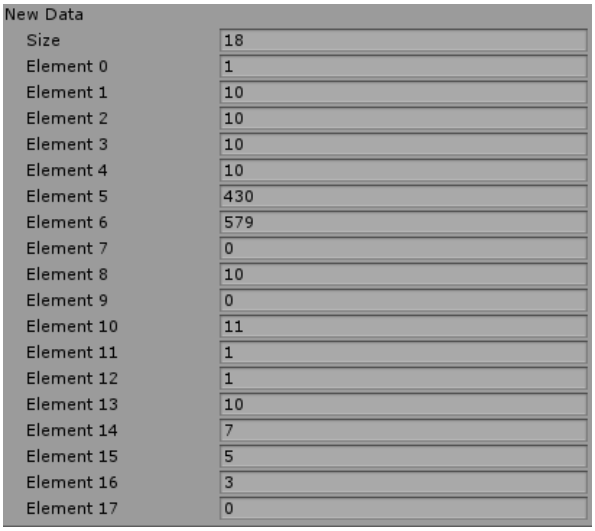
玩家血量計算

SkillRead.cs

讀取技能(所有)資料

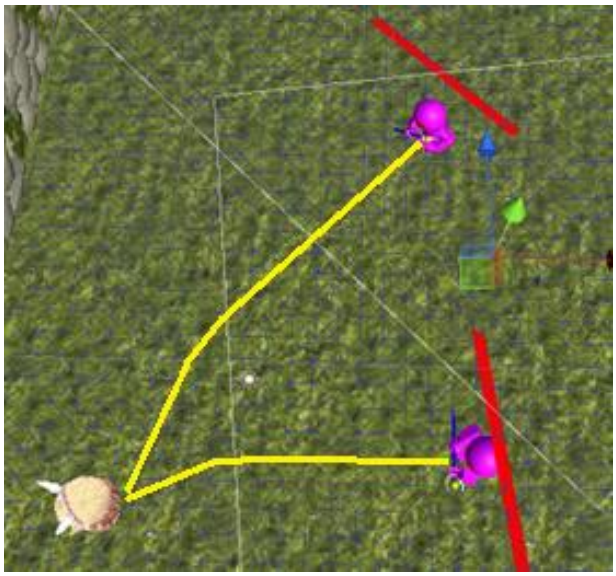
SkillWriting.cs

寫入技能(所有)資料



AIControll2.cs

怪物尋路/攻擊程式 使用 A*演算法(附錄)



SenseMemory.cs

紀錄感測器

SightSensor.cs

視覺感測器

SightTrigger.cs

視覺觸發器

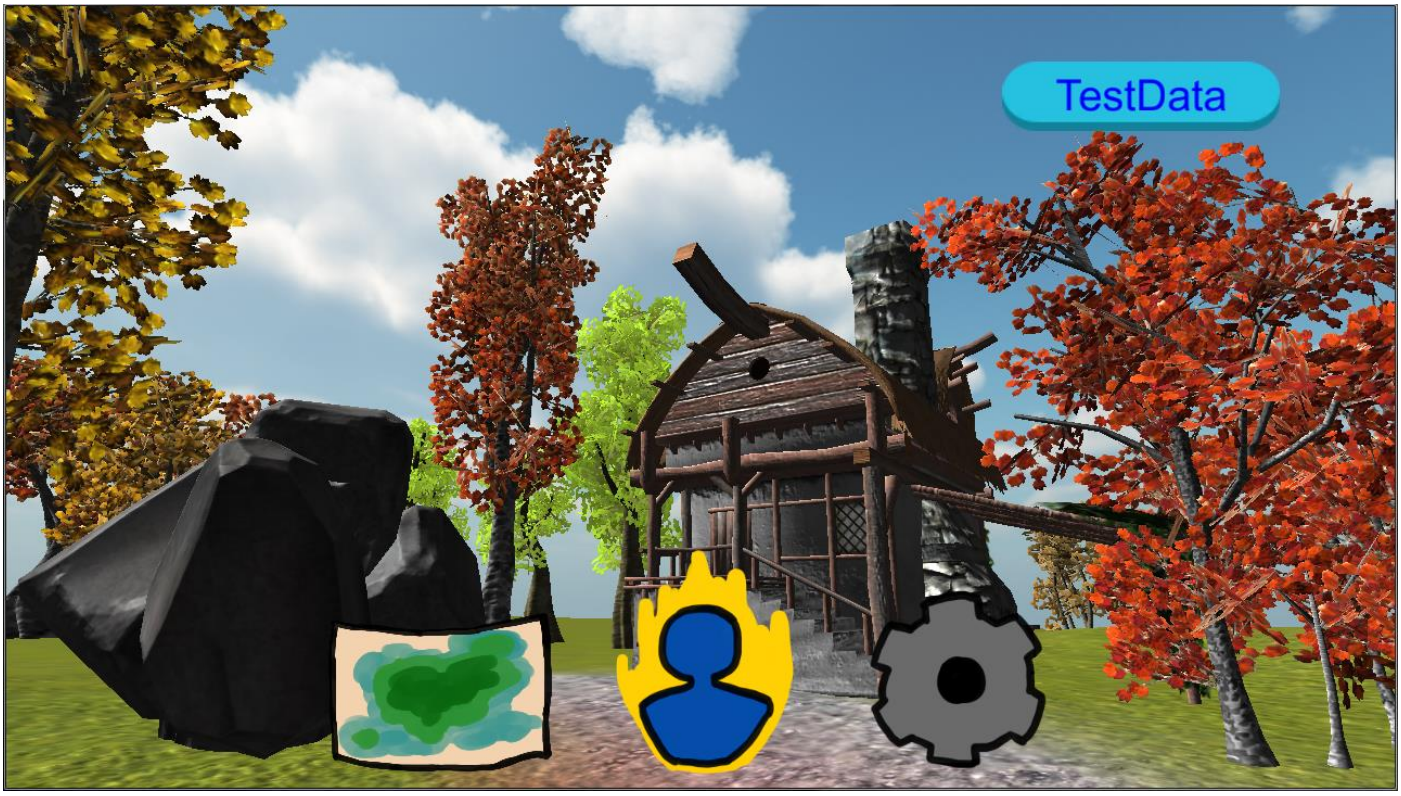
TriggerSystemManager.cs

觸發器管理系統

第六章 遊戲畫面

1. 開頭畫面

2. 主選單畫面



3.1 設定畫面

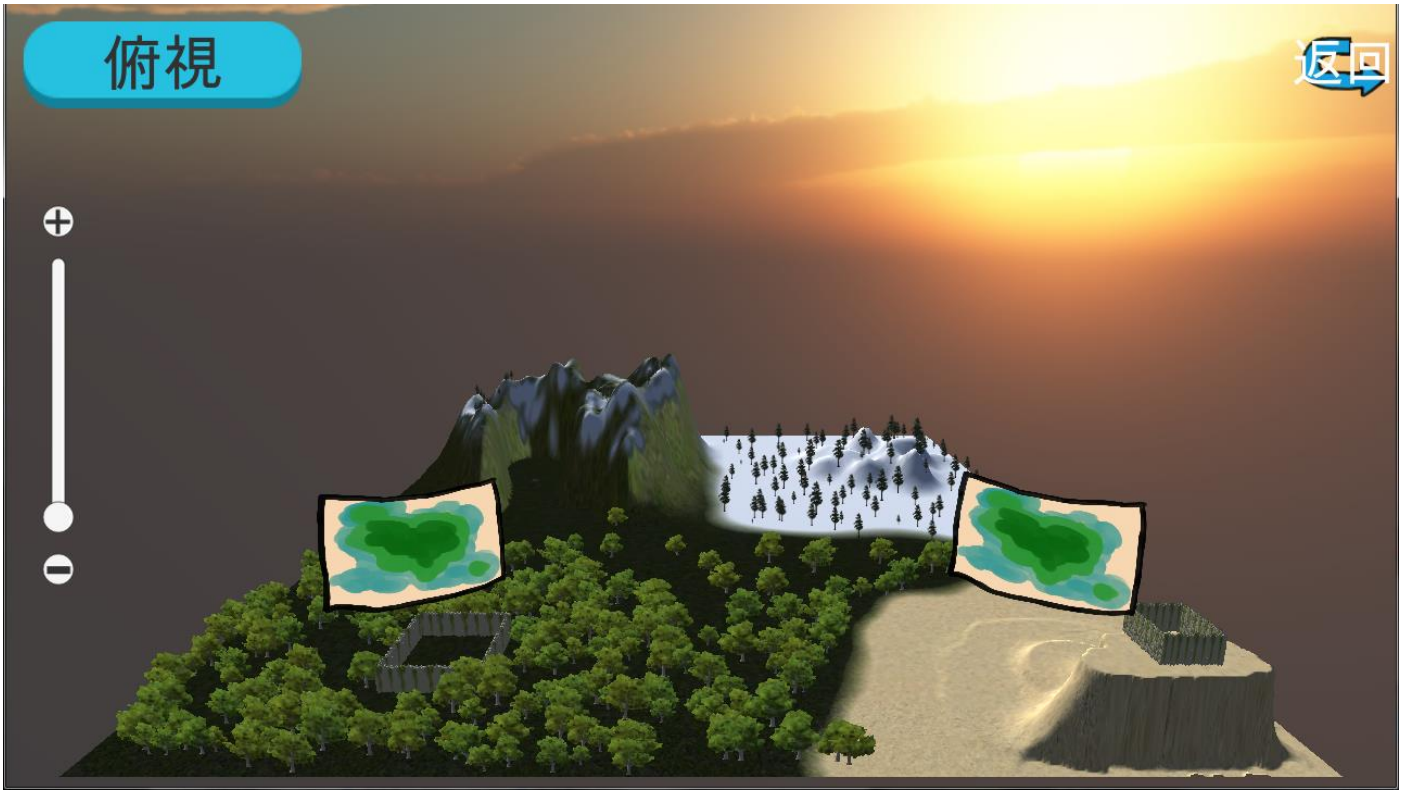
3.2 教學畫面

3.2.1 技能選單教學

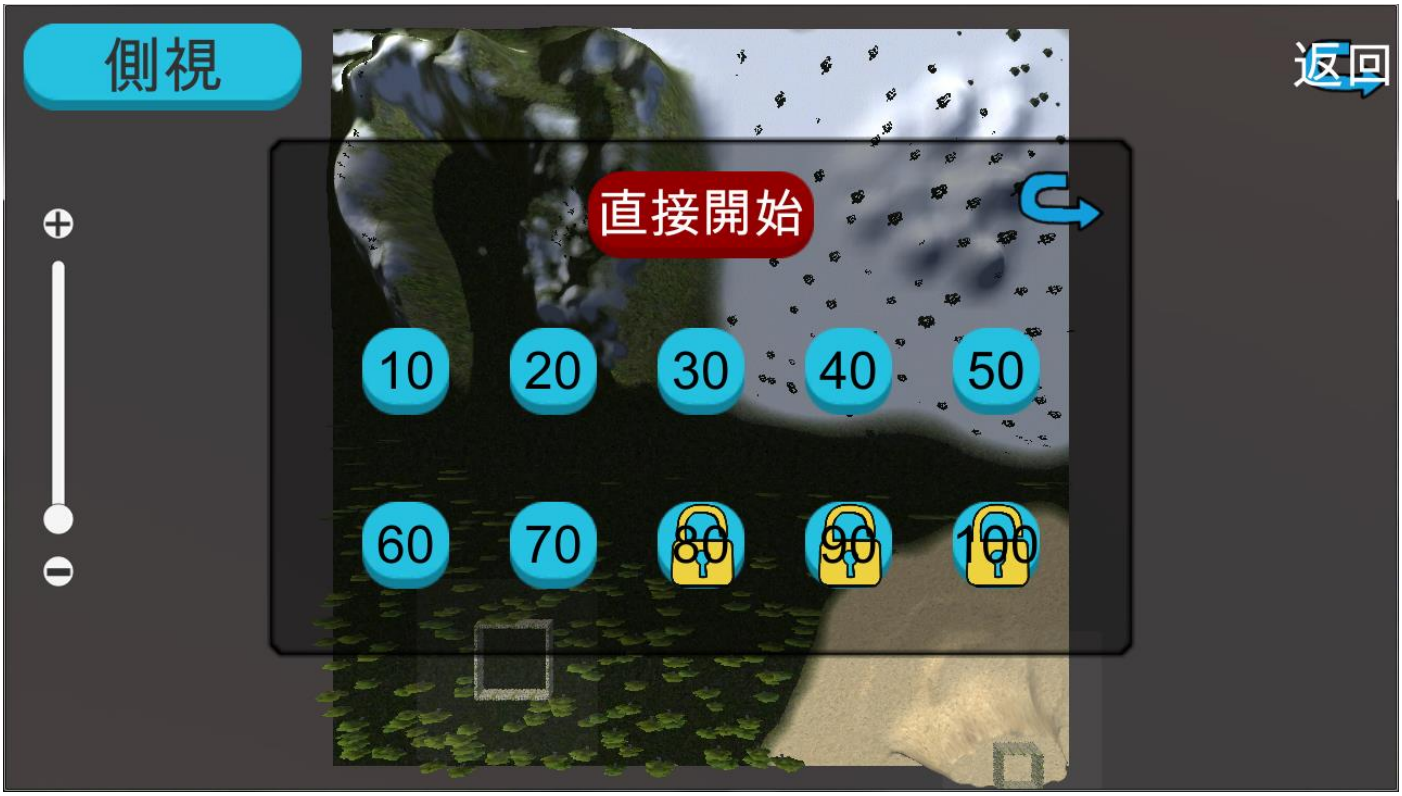
3.2.2 主畫面教學

3.2.3 地圖畫面教學

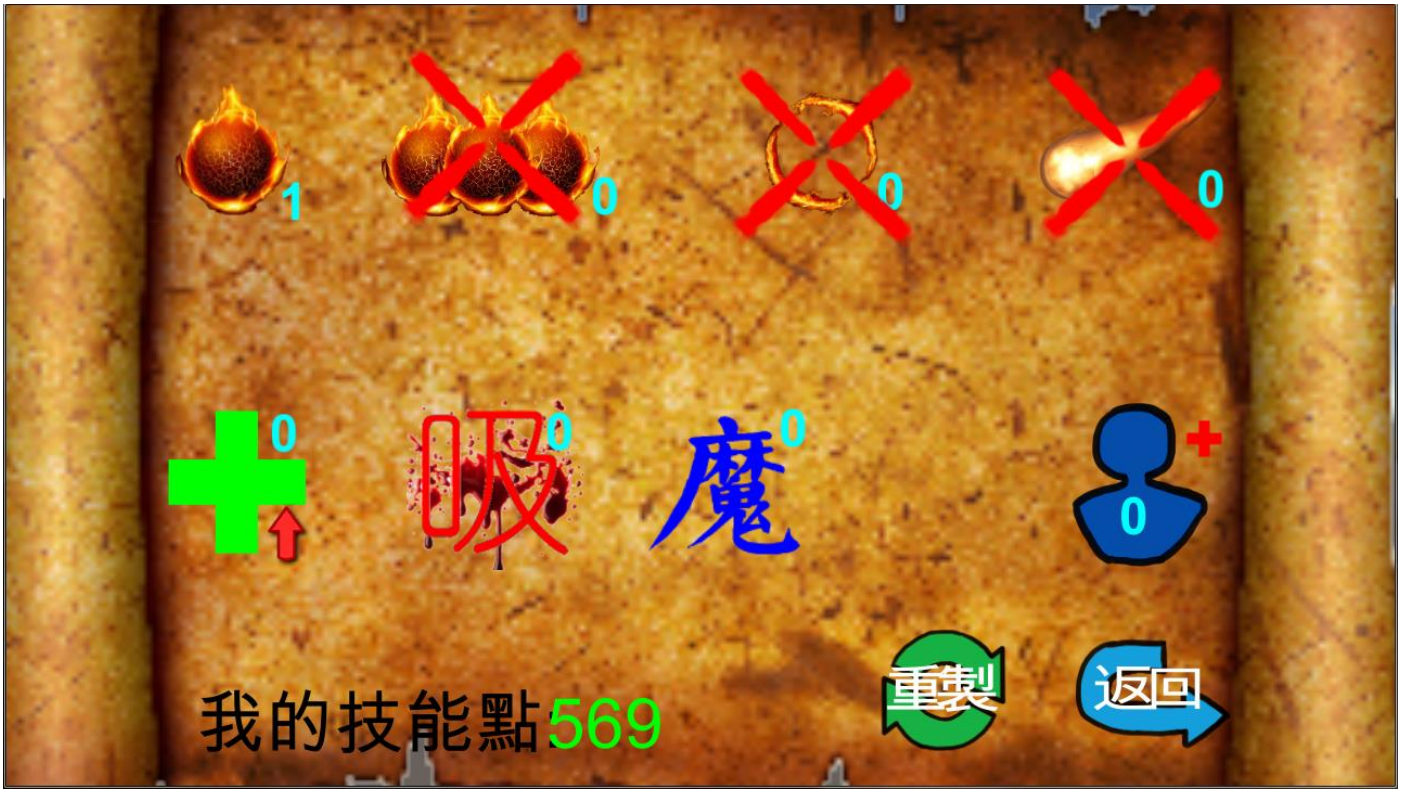
4.1 地圖畫面



4.2 關卡選單

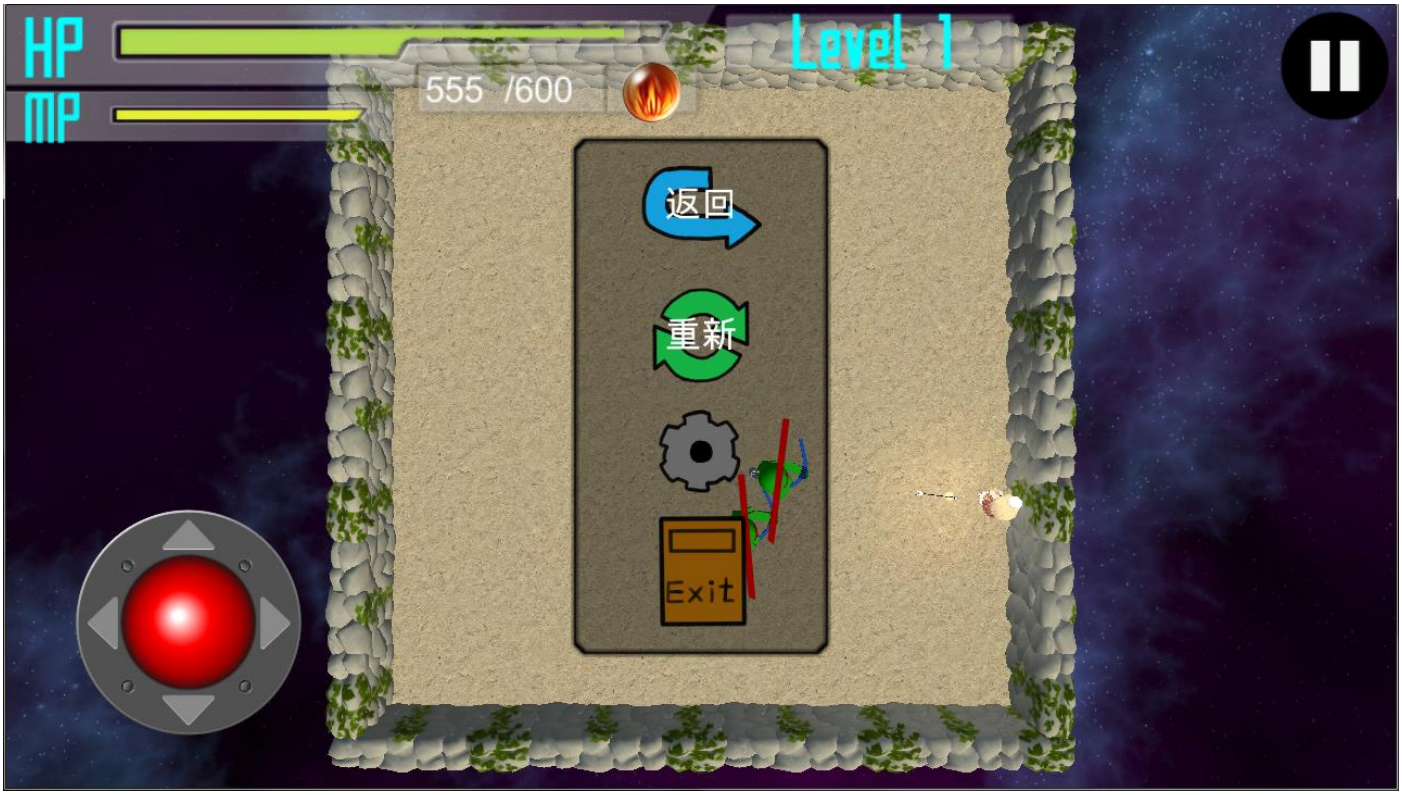


5. 技能畫面



6.1 戰鬥畫面

6.2 暫停選單



6.3 遊戲中設定選單

6.4 遊戲結算畫面

第七章 未來展望

現在我們所做出來的成品對於 Android 系統版本稍舊手機是有點吃力的，而近期出的手機會順暢不少，所以我們現在所遇到的問題就是想讓遊戲在手機中更加順暢，這不僅考驗了我們寫程式的能力，也考驗了模型的製作者，因為在遊戲中，模型是很吃資源的，所以除了克服程式碼的同時也要克服模型質量。

再來是怪物的模型跟遊戲的玩法，這兩項基本上是現在大多數遊戲所要增加的目標，因為一個長久的遊戲除了增加關卡、怪獸外，也要增加一些玩法，而玩法的話是想要增加主角角色（職業）或者增加小魔王戰，這樣的話老玩家不會走，新玩家一直來，達到推陳出新的效果。

最後則是最為重要的手勢這一塊，由於現在我們做出來的跟想像的有一點差距，本來是要邊移動邊畫手勢，可是對於我們目前的技術來說，這個部分我們只能讓他做分離，也就是動搖桿時不能畫手勢，畫手勢後可以立刻接移動，雖然沒有十分得流利順暢，不過對我們來說卻是一個十分重大的突破，因為至少可以用手勢來施放招數，而不是單調的點點手指、按按按鈕，當然未來的我們想要再突破一次，我們想要讓手機螢幕做切割動作，讓搖桿跟手勢區分成兩塊，這樣就可以實現我們最後的目標。

第八章 結論

經過這次的學習讓我們了解到製作一款好玩的遊戲是需要大量的人力與時間，也讓我們知道製作者的辛苦，當然其中的團隊合作是最重要的一環，像我們就有一些小衝突，不過最後還是被我們給化解，化解完當然是做出現在所呈現出來的成品。

雖然是完成了成品，但是也讓我們了解到自己的不足，尤其是關於許多插件上的了解，或者是程式碼上的應用，還有對於ＵＮＩＴＹ這套開發工具的認知，對我們來說都覺得有些不足，尤其是插件這一塊，有些部分都只知道這要怎麼用而不知道它的原理是什麼，這算是在這件專題上的一大遺憾吧！

第九章 附錄

9.1 程式碼

Effect.cs

```
public class Effect : MonoBehaviour
{
    private SkillRead skillReader;//宣告程式碼
    public GameObject Reader;//宣告放此程式碼的物件
    public Transform FireBall;//宣告普通攻擊位置Right
    public GameObject[] FireBallObj;//宣告普通攻擊特效Right
    public Transform FireBig;//宣告普通攻擊位置Right
    public GameObject FireBigObj;//宣告普通攻擊特效Right
    public GameObject FireWall;//宣告火牆
    float mytimer = 0.0f;//我的計時器
    bool FireWallStart = false;//宣告火牆是否開始

    void Start ()
    {
        //讀取FireMS程式碼在AttackRange物件上的資料
        skillReader = Reader.GetComponent<SkillRead>();
        FireWall.SetActive(false);
        FireWallStart = false;//宣告火牆是否開始
        mytimer = 0.0f;//我的計時器
    }

    public void FireBallEvent(int number)//呼叫特效
    {
        Instantiate(FireBallObj[number], FireBall.transform.position, FireBall.
transform.rotation);//呼叫NUMBER號特效
    }

    void FireBigEvent()//呼叫特效
    {
        Instantiate(FireBigObj, FireBig.transform.position, FireBig.transform.r
otation);//呼叫NUMBER號特效
    }
```

```
     void FireEvent()//呼叫特效
    {
        FireWallStart = true;//開始
        FireWall.SetActive(true);//顯示
    }

    void Update ()
    {
        if (FireWallStart)//如果開始
        {
            Mytimer();
        }
    }

    void Mytimer()
    {
        if (mytimer >= (float.Parse(skillReader.newData[3])+float.Parse(skillRe
ader.newData[12])))//如果超過
        {
            FireWallStart = false;//不開始
            mytimer = 0;//歸零
            FireWall.SetActive(false);//不顯示
        }
        else
        {
            mytimer += Time.deltaTime;//加一個單位時間
        }
    }
}
```

Enemy_s1.cs

```
public class Enemy_s1 : MonoBehaviour
{
    private float x, y=0, z;
    public GameObject enemy;
    public GameObject enemyTemp;
    public Transform startPosition;
    public int monsterNum = 0;//怪獸目前數量
    public int MaxMons = 0;//怪獸最大量

    void Update ()
    {
        //當怪物小於最大數量，自動增加到最大值
        if (monsterNum < MaxMons)
        {
            Create ();
            monsterNum++;
        }
    }

    public void Create()
    {
        //隨機座標
        x = Random.Range (5, 45);
        z = Random.Range (5, 45);
        startPosition.transform.position = new Vector3(x,y,z);
        enemyTemp = Instantiate(enemy, startPosition.transform.position, Quater
nion.identity) as GameObject;
    }
}
```

GameOver.cs

```
public class GameOver : MonoBehaviour
{
    private GameOver gameover;//宣告程式碼
    private MyHPRule myHPRule;//宣告程式碼
    public GameObject player;//宣告此程式碼放置物件
    private SkillRead skillRead;//宣告程式碼
    public GameObject Reader;//宣告放此程式碼的物件
    private SkillWriting skillWriting;//宣告程式碼
    public GameObject Writer;//宣告放此程式碼的物件
    public GameObject PlayerGameoverMenu;//顯示死亡菜單
    public Text ScoreNumber;//宣告分數的 TEXT
    public Text MyLevel;//宣告等級
    public int myScore = 0;//宣告分數
    public bool IsGameOver = false;//是否死亡
    public int Level = 0;//宣告等級
    int addScore = 0;//加一次點數

    void Start ()
    {
        Level = 0;
        addScore = 0;//歸零
        //讀取 SkillRead 程式碼在 Reader 物件上的資料
        skillRead = Reader.GetComponent<SkillRead>();
        //讀取 SkillRead 程式碼在 Reader 物件上的資料
        skillWriting = Writer.GetComponent<SkillWriting>();
        gameover = GetComponent<GameOver>();//讀取 GameOver 程式碼上的資料
        //讀取 MyHPRule 程式碼在 player 物件上的資料
        myHPRule = player.GetComponent<MyHPRule>();
        PlayerGameoverMenu.SetActive(false);//不顯示菜單
        IsGameOver = false;
    }
```

```
void Update ()
    {
        MyLevel.text = "Level " + skillRead.newData [10];//顯示現在等級(從一開始)
        ScoreNumber.text = myScore.ToString();
        Debug.Log(Level);
        if (Level >= 10) //如果殺了10 個怪
        {
            skillRead.newData [10] = (int.Parse(skillRead.newData[10])+1).ToStr
ing();
            Level = 0;//重新計算
        }
        if (myHPRule.NowHP <= 0)
        {
            skillWriting.Write ();
            IsGameOver = true;//已死亡
            if (addScore == 0) //加一次
            {
                skillRead.newData [5] = (int.Parse(skillRead.newData[5])+(mySco
re / 100)).ToString();
                skillRead.newData [6] = (int.Parse(skillRead.newData[6])+(mySco
re / 100)).ToString();
                skillRead.newData [11] = ((int.Parse (skillRead.newData [10]) /
 10)).ToString ();
                SaveLevel ();
                addScore++;
                skillWriting.Write ();
            }
            PlayerGameoverMenu.SetActive(true);//顯示菜單
        }

    }
```

```
void SaveLevel()
    {
        if(skillRead.newData[17]=="0")//如果是關卡1
        {
            //如果現在的等級比之前的還要好
            if (int.Parse (skillRead.newData [11]) > int.Parse (skillRead.newDa
ta [13]))
            {
                skillRead.newData[13] = (int.Parse(skillRead.newData[11])).ToSt
ring();//存起來
            }
        }
        else if(skillRead.newData[17]=="1")//如果是關卡2
        {
            if (int.Parse (skillRead.newData [11]) > int.Parse (skillRead.newDa
ta [14]))
            {
                skillRead.newData[14] = (int.Parse(skillRead.newData[11])).ToSt
ring();
            }
        }
        else if(skillRead.newData[17]=="2")//如果是關卡3
        {
            if (int.Parse (skillRead.newData [11]) > int.Parse (skillRead.newDa
ta [15]))
            {
                skillRead.newData[15] = (int.Parse(skillRead.newData[11])).ToSt
ring();
            }
        }
        else if(skillRead.newData[17]=="3")//如果是關卡4
```

```
        {
            if (int.Parse (skillRead.newData [11]) > int.Parse (skillRead.newDa
ta [16]))
            {
                skillRead.newData[16] = (int.Parse(skillRead.newData[11])).ToSt
ring();
            }
        }
    }
}
```

MonsterHPRule.cs

```
public class MonsterHPRule : MonoBehaviour
{
    public GameObject Monster;//宣告怪物
    public Image MonsterHP;//宣告怪物的HP
    public float MaxHP;//宣告怪物最大血量
    public float NowHP;//宣告怪物現在血量
    public int KillScore;//殺死此怪物所獲得的分數
    private GameOver gameover;//宣告程式碼
    public GameObject GameOverMenu;//宣告放此程式碼的物件
    private MyHPRule myHPRule;//宣告程式碼
    public GameObject PQPlayer;//宣告放此程式碼的物件
    private SkillRead skillRead;//宣告程式碼
    public GameObject Reader;//宣告放此程式碼的物件
    private Enemy_s1 enemy_s1;//宣告程式碼
    public GameObject Enemy;//宣告物件

    int myVampire = 0;//我的吸血
    int i=0;

    void Start ()
    {
        i = 0;
       //讀取Enemy_s1程式碼在Enemy物件上的資料
        enemy_s1 = Enemy.GetComponent<Enemy_s1>();
       //讀取GameOver程式碼在GameOverMenu物件上的資料
        gameover = GameOverMenu.GetComponent<GameOver>();
```

```
        //讀取MyHPRule 程式碼在PQPlayer 物件上的資料
        myHPRule = PQPlayer.GetComponent<MyHPRule>();
        //讀取SkillRead 程式碼在Reader 物件上的資料
        skillRead = Reader.GetComponent<SkillRead>();
    }









    void Update ()
    {
        if (i == 0)
        {
            SetmyVampire();//設定基本資料
            i++;
        }
        MonsterHP.fillAmount = NowHP / MaxHP;//顯示怪物血量
        if (NowHP <= 0.0f)
        {
            enemy_s1.monsterNum--;//減一
            Destroy(Monster,3.0f);//刪除
            Monster.SetActive(false);//隱藏
            gameover.Level = gameover.Level+1;//死亡後殺敵數加一
            gameover.myScore += KillScore;//分數++
        }
    }
    void OnCollisionEnter(Collision other)//當碰到碰撞器時
    {
        Debug.Log(other.gameObject.name);
        //跟魔法做碰撞(普攻)
        if (other.gameObject.name == "PlayerFireboltCollider")
        {
            NowHP = NowHP - int.Parse (skillRead.newData [1]) * 10 - int.Parse
(skillRead.newData [12]) * 100;//普攻攻擊係數
            if (myHPRule.NowHP < myHPRule.MaxHP)//如果現在血量小於最大血量
            {
```

```
                if ((myVampire + myHPRule.NowHP) > myHPRule.MaxHP)//如果吸血過多
                {
                    myHPRule.NowHP = myHPRule.MaxHP;//血會變成最大值
                }
                else
                {
                    myHPRule.NowHP += myVampire;//直接加血
                }
            }
        }
        else if (other.gameObject.name == "Meteor(Clone)")//跟大絕做碰撞
        {
            NowHP = NowHP - int.Parse (skillRead.newData [4]) * 15 - int.Parse
(skillRead.newData [12]) * 100;
            if (myHPRule.NowHP < myHPRule.MaxHP)//如果現在血量小於最大血量
            {
                if ((myVampire + myHPRule.NowHP) > myHPRule.MaxHP)//如果吸血過多
                {
                    myHPRule.NowHP = myHPRule.MaxHP;//血會變成最大值
                }
                else
                {
                    myHPRule.NowHP += myVampire;//直接加血
                }
            }
        }
        else if (other.gameObject.name == "FireboltCollider")//一技
        {
            NowHP = NowHP - int.Parse (skillRead.newData [2]) * 10 - int.Parse
(skillRead.newData [12]) * 100;
            if (myHPRule.NowHP < myHPRule.MaxHP)//如果現在血量小於最大血量
            {
                if ((myVampire + myHPRule.NowHP) > myHPRule.MaxHP)//如果吸血過多
                {
                    myHPRule.NowHP = myHPRule.MaxHP;//血會變成最大值
                }
                else
                {
                    myHPRule.NowHP += myVampire;//直接加血
                }
```

```
            }
        }
    }
    void SetmyVampire()
    {
        KillScore = 10 * int.Parse (skillRead.newData [10]);
        myVampire = int.Parse(skillRead.newData[7]) * 10;//吸血
        MaxHP = 50 + int.Parse (skillRead.newData [10]) * 50;//等級
        NowHP = MaxHP;
    }
}
```

MyHPRule.CS

```
public class MyHPRule : MonoBehaviour
{
    public Image PlayerHP;//宣告玩家的HP
    public Text PlayerMaxHP;//宣告現在玩家的血量
    public Text PlayerNowHP;//宣告現在玩家的血量
    public GameObject PlayerSubHP;//宣告現在玩家所扣的血量
    public float MaxHP;//宣告玩家最大血量
    public float NowHP;//宣告玩家現在血量
    public GameObject MyFireWall;//宣告我的火牆
    private SkillRead skillRead;//宣告程式碼
    public GameObject Reader;//宣告放此程式碼的物件
    float mytimer = 0.0f;//我的計時器
    void Start()
    {
        //讀取SkillRead程式碼在Reader物件上的資料
        skillRead = Reader.GetComponent<SkillRead>();
        Time.timeScale = 1.0f;//遊戲開始
        PlayerSubHP.SetActive(false);//隱藏扣血
        //初始玩家血量顯示
        PlayerNowHP.text = NowHP.ToString();
    }
```

```
    void Update()
    {
        PlayerMaxHP.text = "/" + MaxHP.ToString();
        PlayerHP.fillAmount = NowHP / MaxHP;//顯示玩家血條
        PlayerNowHP.text = NowHP.ToString();//顯示玩家血量
        if (NowHP <= 0.0f)//如果血量等於0
        {
            NowHP = 0.0f;//讓血量不會是負的
            Time.timeScale = 0.0f;//整個遊戲暫停
        }
    }
    void OnCollisionEnter(Collision other)//當碰到碰撞器時
    {
        Debug.Log(other.gameObject.name);
        //跟魔法做碰撞
        if ((other.gameObject.name == "FireballCollider") && (MyFireWall.active
 == false))
        {
            NowHP = NowHP - (int.Parse (skillRead.newData [10]) * 5.0f);}}}
```

SkillRead.cs

```
public class SkillRead : MonoBehaviour
{
    protected FileInfo MyData = null;//我的資料
    protected StreamReader MyDataReader = null;//我的資料讀取者
    protected string text = " "; // assigned to allow first line to be read bel
ow
    public String[] newData =new String[18];

    void Start ()
    {
        MyData = new FileInfo(Application.persistentDataPath + "/Data.txt");
        MyDataReader = MyData.OpenText();
        ReadText();
    }
    //讀取資料到面板上
    void ReadText()//讀取資料到面板上
    {
        for (int i = 0; i < 18; i++)
        {
```

```
            //讀取每一行資料
            text = MyDataReader.ReadLine();
            newData[i] = text;
        }
        //關閉
        MyDataReader.Close();
    }
}
```

SkillWriting.cs

```
public class SkillWriting : MonoBehaviour
{
    private SkillRead skillReader;//宣告程式碼
    public GameObject Reader;//宣告放此程式碼的物件

    FileStream MyData = null;//我的資料
    StreamWriter MyDataPan = null;//我的筆
    int i = 0;

    void Start ()
    {
        skillReader = Reader.GetComponent<SkillRead>();//讀取 FireMS 程式碼在
AttackRange 物件上的資料
    }
    //存檔
    public void Write()
```

```
    {
        //讀取資料
        MyData = new FileStream(Application.persistentDataPath + "/Data.txt", F
ileMode.OpenOrCreate);
        //設定一枝筆在資料上
        MyDataPan = new StreamWriter(MyData);
        for (i = 0; i < 18; i++)
        {
            //寫入
            MyDataPan.WriteLine(skillReader.newData[i]);
        }
        MyDataPan.Close();
    }
}
```

AIControll2.cs

```
public class AIController2 : AIPath
{
    float mytimer = 0.0f;//我的計時器
    public AnimatorStateInfo bs;//把現在所撥放的動畫轉成數字
    //宣告 Idle 的動畫名稱,順便轉成變成數字
    static int idle = Animator.StringToHash("Base Layer.Idle");
    //宣告攻擊動作
    static int attack = Animator.StringToHash("Base Layer.Attack1");
    bool clickStart = false;//是否普功
    public Animator Monster;//宣告人物動畫
    public int health;
    public float arriveDistance = 1.0f;
    //目標點預設物
    public GameObject targetPrefab;
```

```
    //可以停止追逐開始射擊的距離
    public float shootingDistance = 10.0f;
    //從射擊狀態重新轉換到追逐狀態的距離
    private Animation anim;
    //黑板物件
    private Blackboard bb;
    //目前路點索引
    private int wayPointIndex = 0;
    //最近感測玩到玩家的位置
    private Vector3 personalLastSighting;
    //上次玩家的位置
    private Vector3 previousSighting;
    //路點的陣列
    private Vector3[] wayPoints;
    private Vector3 memoryPos;
    //記憶物件
    private SenseMemory memory;
    private float timer = 0;
    float stopDistance = 0.6f;
    float waitTime = 6.0f;
    private FSMState state;




    public enum FSMState
    {
        Chasing,        //追逐狀態
        Shooting,        //射擊狀態
        Avoid,            //迴避狀態
    }

    void Start ()
    {
        mytimer = 4.0f;
        anim = GetComponent<Animation>();
        //獲得黑板物件
        bb = GameObject.FindGameObjectWithTag("Blackboard").GetComponent<Blackb
oard>();
```

```
    personalLastSighting = bb.resetPosition;
    previousSighting = bb.resetPosition;
    //獲得記憶物件
    memory = GetComponent<SenseMemory>();

    memoryPos = new Vector3(0,0,0);
    target = targetPrefab.transform;

    state = FSMState.Chasing;
    //儲存所有路點到一陣列中
    //target.position = wayPoints[0];

    base.Start();
}


public override void Update ()
{
    //記住現在所執行的動畫
    this.transform.LookAt (targetPrefab.transform.position);
    bs = Monster.GetCurrentAnimatorStateInfo(0);
    //如過玩家位置發生變化，更新
    if (bb.playerLastPosition != bb.resetPosition)
        memoryPos = bb.playerLastPosition;

    if (bb.playerLastPosition != previousSighting)
    {
        personalLastSighting = bb.playerLastPosition;
    }
    switch (state)
    {
        case FSMState.Chasing:
            Chasing();
            break;
        case FSMState.Shooting:
            timer = 0;
            Shooting();
            break;
        case FSMState.Avoid:
            timer = 0;
```

```
                Avoid();
                break;
        }
        previousSighting = bb.playerLastPosition;
    }
    void Shooting()
    {
        state = FSMState.Shooting;
        //如果玩家位置更新，可以開始追逐
        if (!CanShoot () || ((personalLastSighting != previousSighting) && Vect
or3.Distance (transform.position, personalLastSighting) > shootingDistance))
        {
            state = FSMState.Chasing;
            clickStart = false;
            Monster.SetBool ("Run", true);
        }
        else
        {
            MyClick();//普功
            Mytimer();
            Monster.SetBool ("Run", false);
        }
    }
    bool CanShoot()
    {
        RaycastHit hit;
        Vector3 rayDirection = personalLastSighting - transform.position;
        rayDirection.y = 0;
        if(Vector3.Distance (transform.position, target.transform.position) > s
hootingDistance)
            return false;
        else
            return true;
    }
    void Chasing()
    {
        state = FSMState.Chasing;
        timer += Time.deltaTime;
        if (Vector3.Distance(transform.position, target.position) > shootingDis
tance)
```

```
        {
            if (Vector3.Distance(transform.position, target.position) < 3)
            {
                if (wayPointIndex == wayPoints.Length - 1)
                {
                    wayPointIndex = 0;
                    target.position = wayPoints[wayPointIndex];
                }
                else
                {
                    wayPointIndex++;
                    target.position = wayPoints[wayPointIndex];
                }
            }
            base.Update();
            Monster.SetBool ("Run", true);
        }
        else
        {
            if (CanShoot())
            {
                state = FSMState.Shooting;
            }
            else
            {
                if (Vector3.Distance(transform.position, target.position) > sto
pDistance)
                {
                    base.Update();
                    Monster.SetBool ("Run", true);
                }
                else if (timer < waitTime)
                {
                    transform.Rotate(Vector3.up *30* Time.deltaTime, Space.Worl
d);
                    Monster.SetBool ("Run", false);
                }
                else
                {
                    state = FSMState.Chasing;
```

```
                }
            }
        }
    }
    void MyClick()//普功
    {
        if ((clickStart) && (bs.nameHash == idle))
        {
            //Debug.Log ("123546879");
            Monster.SetBool ("Skill", true);//就放動畫
            clickStart = false;
        }
        else if (bs.nameHash == attack)//如果現在是attack 的動畫
        {
            if (!Monster.IsInTransition(0))//如果此動畫(attack)正在播放
            {
                Monster.SetBool("Skill", false);//就不放動畫
            }
        }
    }
    void Mytimer()
    {
        if (mytimer >= 5.0f)//如果超過
        {
            clickStart = true;
            mytimer = 0;//歸零
        }
        else
        {
            mytimer += Time.deltaTime;//加一個單位時間
        }
    }
}
```

SenseMemory.cs

```
public class SenseMemory : MonoBehaviour
{
    //已經在列表中?
    private bool alreadyInList = false;
    //記憶留存時間
```

```
public float memoryTime = 4.0f;
//記憶列表
public List<MemoryItem> memoryList = new List<MemoryItem>();
//此時需要從記憶列表清單中刪除的項
private List<MemoryItem> removeList = new List<MemoryItem>();
//在記憶列表中尋找玩家資訊
public bool FindInList()
{
    foreach (MemoryItem mi in memoryList)
        if (mi.g.tag == "Player")
            return true;

    return false;
}
//在記憶列表中增加一個項
public void AddToList(GameObject g, float type)
{
    alreadyInList = false;
    //如果該項已經在列表中，那麼更新最後的感測時間等資訊
    foreach (MemoryItem mi in memoryList)
    {
        if (g == mi.g)
        {
            alreadyInList = true;
            mi.lastMemoryTime = Time.time;
            mi.memoryTimeLeft = memoryTime;
            if (type > mi.sensorType)
                mi.sensorType = type;
            break;
        }
    }
    //如果不再列表中，新增項並加入列表
    if (!alreadyInList)
    {
       MemoryItem newItem = new MemoryItem(g, Time.time, memoryTime, type);
        memoryList.Add(newItem);
    }
}

void Update ()
```

```
    {
        removeList.Clear();
        //檢查所有項，找到那些逾時需要 忘記 的項，刪除
        foreach (MemoryItem mi in memoryList)
        {
            mi.memoryTimeLeft -= Time.deltaTime;
            if (mi.memoryTimeLeft < 0)
            {
                //memoryList.Remove(mi);
                removeList.Add(mi);
            }
            else
            {
                //對沒被刪除的項，畫出一條線，表示仍然在記憶體中
                if (mi.g != null)
                    Debug.DrawLine(transform.position, mi.g.transform.position,
 Color.blue);
            }
        }
        foreach (MemoryItem m in removeList)
        {
            memoryList.Remove(m);
        }
    }
}

public class MemoryItem
{
    //感測到的遊戲物件
    public GameObject g;
    //最近的感測時間
    public float lastMemoryTime;
    //還能留在記憶體中的時間
    public float memoryTimeLeft;
    //透過哪種方式感測到的該物件，視覺為 1，聽覺為 0.66
    public float sensorType;

    public MemoryItem(GameObject objectToAdd, float time, float timeLeft,float
type)
    {
```

```
        g = objectToAdd;
        lastMemoryTime = time;
        memoryTimeLeft = timeLeft;
        sensorType = type;
    }
}
```

SightTrigger.cs

```
public class SightTrigger : Trigger
{

    public override void Try(Sensor sensor)
    {
        //如果感測器能感覺到這個觸發器，那麼向感測器發出通知，感測體做出對應的決策或行動
        if (isTouchingTrigger(sensor))
        {
            sensor.Notify(this);
        }
    }
    //判斷感測器是否能被感測到這個觸發器
    protected override bool isTouchingTrigger(Sensor sensor)
    {
        GameObject g = sensor.gameObject;
        //如果這個感測器能夠感測視覺資訊
        if (sensor.sensorType == Sensor.SensorType.sight)
        {
            RaycastHit hit;
            Vector3 rayDirection = transform.position - g.transform.position;
            rayDirection.y = 0;
            //判斷感測體的向前方向與物體所在方向的夾角，是否在視域範圍內
            if ((Vector3.Angle(rayDirection, g.transform.forward)) < (sensor as SightSensor).fieldOfView)
            {
                //在視線距離內是否存在其他帳拽遮擋，如果沒有障礙物，則傳回 true
                if (Physics.Raycast(g.transform.position + new Vector3(0,1,0), rayDirection, out hit, (sensor as SightSensor).viewDistance))
                {
                    if (hit.collider.gameObject == this.gameObject)
                    {
```

```
                        return true;
                    }
                }
            }
        }
        return false;
    }
    //更新觸發器的內部資訊，由於帶有視覺觸發器的AI角色可能是運動的，因此要不停更新這個觸發器的位置
    public override void Updateme()
    {
        position = transform.position;
    }

    void Start ()
    {
        //呼叫基礎類別的Start()函數，如果去掉這個敘述，那麼基礎類別的Start()不會被呼叫
        base.Start();
        //向管理員註冊這個觸發器，管理員會把他加入目前觸發器列表中
        manager.RegisterTrigger(this);
    }
}
```

TriggerSystemManager.cs

```
public class TriggerSystemManager : MonoBehaviour
{
    //初始化目前感測器列表
    List<Sensor> currentSensors = new List<Sensor>();
    //初始化目前觸發器列表
    List<Trigger> currentTriggers = new List<Trigger>();
    //紀錄目前時刻需要被移除的感測器，例如感測體死亡，需要移除感測器時
    List<Sensor> sensorsToRemove;
    //紀錄目前時刻需要被移除的觸發器，例如觸發器已過期時
    List<Trigger> triggersToRemove;

    void Start ()
    {
        sensorsToRemove = new List<Sensor>();
        triggersToRemove = new List<Trigger>();
```

```
    }


    private void UpdateTriggers()
    {
        //對於目前觸發器列表中的每個觸發器 t
        foreach (Trigger t in currentTriggers)
        {
            //如果 t 需要被移除
            if (t.toBeRemoved)
            {
                //將 t 加入需要移除的觸發器清單中(這是由於不能在 foreach 中直接移除，否則會出錯)
                triggersToRemove.Add(t);
            }
            else
            {
                //更新觸發器內部資訊
                t.Updateme();
            }
        }
        //對於需要移除的觸發器清單中的每個發器 t，從目前觸發器列鰾中移除 t
        foreach (Trigger t in triggersToRemove)
            currentTriggers.Remove(t);
    }

    private void TryTriggers()
    {
        //對於目前感測器列表中的每個感應器 s
        foreach (Sensor s in currentSensors)
        {
            //如果 s 所對應的感測體還會有(沒有因死亡而被銷毀)
            if (s.gameObject != null)
            {
                //對於目前感測器列表中的每個感應器 t
                foreach (Trigger t in currentTriggers)
                {
                    //檢測 s 是否在 t 的作用範圍內，並做出對應的回應
                    t.Try(s);
                }
```

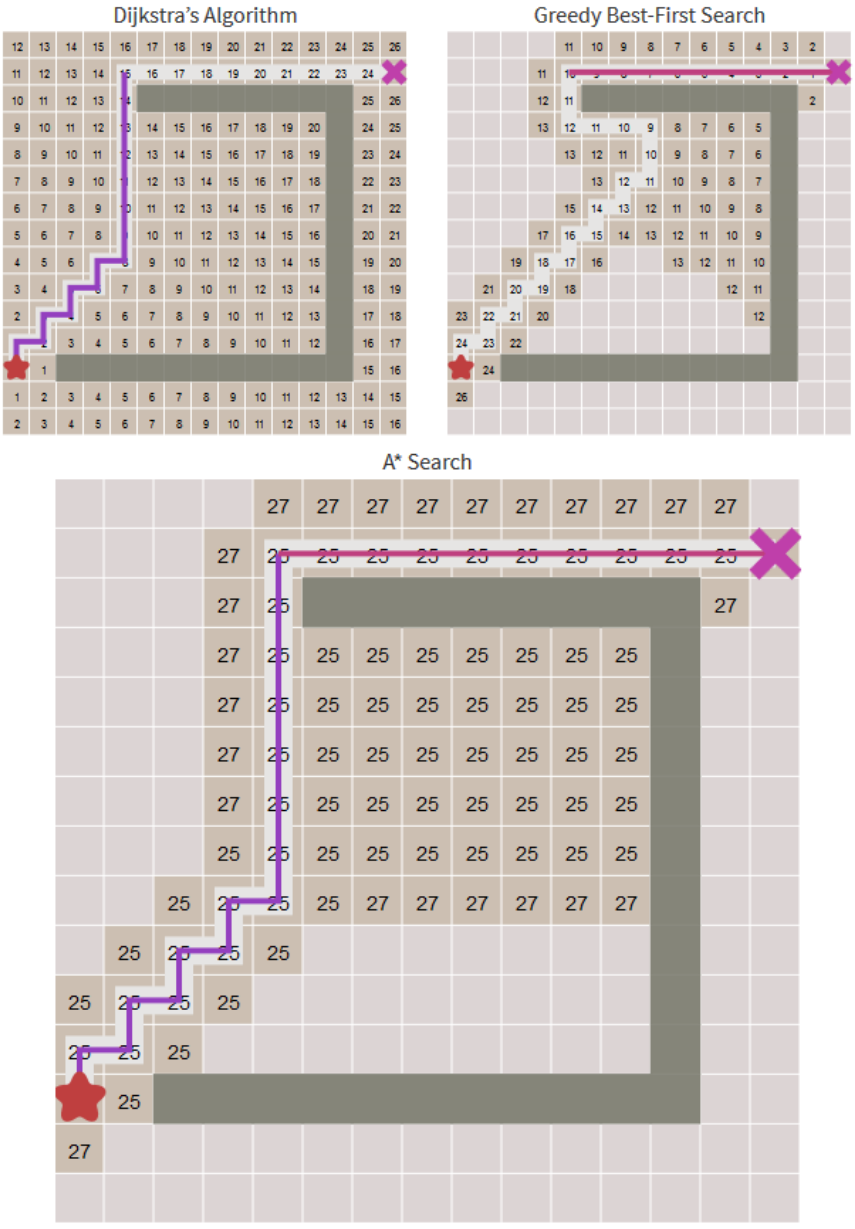
```
            }
            else
            {
                //將感應器 s 加入到需要移除的感測器列表中
                sensorsToRemove.Add(s);
            }
        }
        //對於需要移除的觸發器清單中的每個發器 s，從目前觸發器列鰾中移除 s
        foreach (Sensor s in sensorsToRemove)
            currentSensors.Remove(s);
    }

    void Update ()   //Tick is also OK.
    {
        //更新所有觸發器內部狀態
        UpdateTriggers();
        //反覆運算所有感測器和觸發器，做出對應的行為
        TryTriggers();
    }
    //用於註冊觸發器
    public void RegisterTrigger(Trigger t)
    {
        print ("registering trigger:" + t.name);
        //將參數觸發器 t 加入到目前觸發器列表中
        currentTriggers.Add(t);
    }

    public void RegisterSensor(Sensor s)
    {
        print ("registering sensor:" + s.name + s.sensorType);
        //將參數感應器 s 加入到目前觸發器列表中
        currentSensors.Add(s);
    }
}
```

9.2 A* 演算法

A*（A-Star）演算法是在 Game 中通常用來解決最短路徑(Shortest Path)問題的一種演算法，相對於另一個知名的 Dijkstra(戴克斯特拉)演算法來說，Dijkstra 演算法雖然可以保證找到一條最短的路徑，但不如 A* 演算法這樣簡捷快速，同時，Dijkstra 的搜尋深度在某些情形下也容易顯得不適用，A* 演算法便是為了這些情形而出現的，可以算是 Dijkstra 演算法的一種改良。



從以上的比較圖可以看出，A* 演算法的搜尋深度雖然不如 Dijkstra 演算法，但其結果仍然是很令人滿意的，因為 A* 演算法採用了一套特殊的啟發式評價(Heuristic Estimate)公式將許多明顯為壞的路徑排除考慮, 進而快速計算出一條滿意的路徑。

公式如下:

f(n) = g(n) + h(n)

g(n): 從啟始點到目前節點的距離

h(n): 預測目前節點到結束點的距離(A*演算法的主要評價公式)

f(n): 目前結點的評價分數

其中, h(n) 主導著 A* 演算法的表現方式. 有以下幾種情形:

1. h(n)=0: A* 演算法這時等同 Dijkstra 演算法，並且保證能找到最短路徑

2. h(n)<目前節點到結束點的距離: A* 演算法保證找到最短路徑， h(n)越小，搜尋深度越深

3. h(n)=目前節點到結束點的距離: A* 演算法僅會尋找最佳路徑，並且能快速找到結果

4. h(n)>目前節點到結束點的距離: 不保證能找到最短路徑, 但計算比較快

5. h(n)與 g(n)高度相關: A* 演算法此時成為 BFS (Best-First Search)