





3D游戏设计实验报告（二）

游戏介绍

游戏名为《Walk To Green》,是一款基于九宫格地图设计的解谜游戏（虽然解密的程度很轻）。在游戏中玩家需要扮演一个红色方块，通过观察环境，推断出游戏机制，在合适的时机选择路径到达终点。

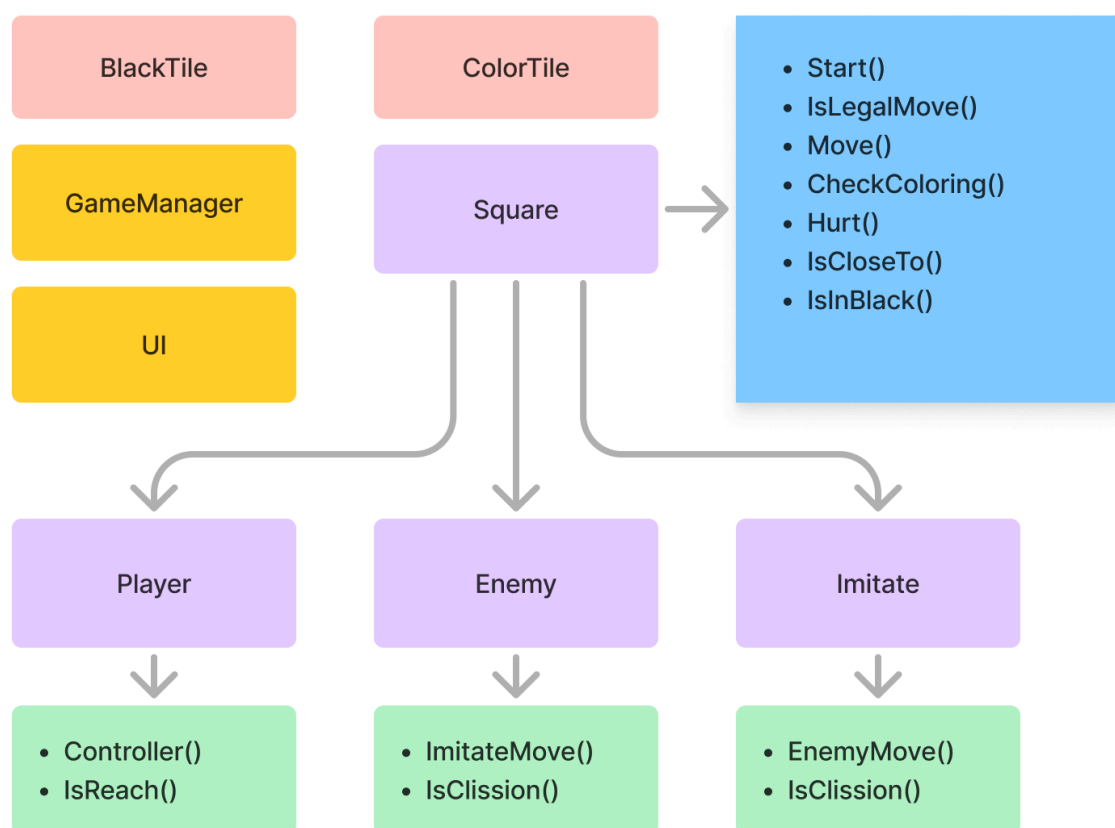
[游戏演示视频链接](#)

游戏构成

Square	Describe	Tile	Describe
	玩家(Player)可控制该方块进行移动		黑色地砖(BlackTile)会阻挡方块移动
	敌人(Enemy)按照既定轨迹周期性运动		终点(Terminal)代表当前关卡的结束
	模仿块(Imitate)会重复玩家的移动轨迹		染色地砖(ColorTile)会被经过的方块染色

除开本游戏的九宫格地图外，其余的游戏元素共有六种，分别为玩家、敌人、模仿快、黑色地砖、终点以及染色地砖，它们组合分布在小小的九宫格中形成各式各样的关卡。

游戏实现



游戏的编码实现部分的模块如上图所示，其中又可以根据各类的不同职能划分为三块：

Square类及其子类

Square基类中主要实现了Player、Enemy、Imitate类中的共同功能，比如移动与受伤。而Player类中实现的移动是基于键盘控制的，Enemy类中实现的移动是基于动作序列的，Imitate类的移动则是仿照Player对象的。同时在Player类中需对终点的到达进行判断，Enemy与Imitate类则需判断与玩家的碰撞。

Tile型类

Tile型类有BlackTile与ColorTile两种，这两类由于比较被动，所以实现的功能主要是为其他类提供改变自身状态的方法。例如ColorTile类中的ChangeColor()。

GameManager类及UI类

GameManager类重要负责游戏系统状态的切换，例如当当前场景的ColorTile都被涂成Player的颜色时，游戏中的BlackTile解锁。同时当玩家通关后，不同关卡对应的场景的切换也由GameManager类控制。UI类则是控制场景中的UI显示。

Square类的实现

说是实现Square类，实际上游戏场景中并无绑定Script(Square.cs)的精灵，但是由于Player、Enemy、Imitate类在移动、受伤等情况需要遵守相同的规则，所以就通过类的继承的方式，通过在基类Square中实现上述共通的方法，增强代码的复用性。

如上述游戏代码结构图所示，我们在Square类中分别实现了IsLegalMove()、Move()等方法，并在Start()方法中对类的变量进行初始化等。而对于子类中需要特化的方法则分别在子类中实现。

```
protected IEnumerator Move(string direction)
{
    Vector3 movement = new Vector3(0,0,0);
    if(direction == "up") movement = new Vector3(0,1,0);
    if(direction == "down") movement = new Vector3(0,-1,0);
    if(direction == "left") movement = new Vector3(-1,0,0);
    if(direction == "right") movement = new Vector3(1,0,0);

    Vector3 startPosition = transform.position;
    Vector3 targetPosition = startPosition + movement;

    float timeElapsed = 0f;
    IsMove = true;
    while (timeElapsed < MoveTime && !IsFlash)
    {
        timeElapsed += Time.deltaTime;
        float t = timeElapsed / MoveTime;
        transform.position = Vector3.Lerp(startPosition,
targetPosition, t);
        yield return null;
    }

    if(!IsFlash) transform.position = targetPosition;
    IsMove = false;
}
```

GameManager类及UI类的实现

其中GameManager类与UI类放在一起说明是因为它们具有相同的特点：它们不负责控制某个对象的行为逻辑，而是对游戏全局进行控制，并且这一点不随着场景变换而改变。这个特点意味着这种控制类的实例有且仅有1个，所以使用单例模式构造该类。

```
public static GameManager Instance { get; private set; }

private void Awake()
{
    if (Instance != null && Instance != this){
        Destroy(gameObject);
        return;
    }

    Instance = this;
    DontDestroyOnLoad(gameObject);
    SceneManager.sceneLoaded += OnSceneLoaded;
}
```

游戏中的MVC模式

MVC 模式代表 Model-View-Controller（模型-视图-控制器）模式。这种模式用于应用程序的分层开发。在本游戏中，游戏的视图通过游戏中的精灵、场景、及摄像机及光源的控制来呈现。而游戏中的数据则主要由精灵的组件（Component）来存储，也就是模型。而控制则由附着在精灵上的控制器来完成。

MVC的使用可以为游戏制作带来很多好处，例如增强了游戏的可维护性、可扩展性和性能。将数据和逻辑分开，使得代码结构更加清晰，便于理解和维护。开发者可以独立于逻辑更改数据或反之亦然。同时当游戏中出现问题时，能够更容易地定位问题。开发者可以专注于逻辑或数据的特定部分，而不必在混合代码中进行大量的查找。