

Unity 版本 2021.1

战棋类游戏

剧情推动。

操作：

鼠标拖动单位移动，松开移动到相应的位置。移动结束后会根据鼠标位置选择角色的朝向（有箭头提示），再次按下鼠标左键可确定角色朝向。

操作尽量只用鼠标进行，避免加入键盘，我觉得这在体验上很关键。

回合逻辑：

每回合分为移动、战斗、特殊三个阶段。

在移动阶段，按单位的速度的顺序进行移动。

敌方单位会在轮到他的时候自动移动。

全单位移动完毕之后进入战斗阶段，如若一个单位的攻击范围内存在可攻击的单位，则自动进行战斗处理。战斗行动也是按照速度顺序执行。

战斗阶段完毕后进入特殊阶段，这个阶段可以启动角色的特殊技能（如再动、攻击技能等）。

关卡设计：

目前为了 Debug 方便，使用一个 BattleGridGen class 自动生成地图。地图数据读取自 LevelDesign 这个 static class，目前只有 plain floor，可考虑加入树林、障碍物等战棋常见场地效果。

也可以考虑直接每个关卡新建一个 scene，或者也可以单纯完善 BattleGridGen 来生成不同关卡的地图，两种方案都可以。

```
public List<GameObject> grids = new List<GameObject>();  
  
public GameObject[,] gridMatrix;//grid matrix
```

如图↑，在 BattleGridGen 中，每一个地图格子都是一个 GameObject，为了方便各种功能的实现，格子会用两种数据结构记录。

一种是单纯的一个 list. 最左上角的格子 grid1, 即在 list 里面是第[0]个。

地图格子同时也用一个三项 Matrix 表示: 第一项表示排 (column), 第二项表示行 (row), 第三项为保留项, 目前皆为 0, 未来或许可以用来实现一些别的功能 (比如可用来表示这个格子是否是障碍物之类的...)

[2][3][0], 即表示第二行第三个格子, 即 list 中第[2*row + 3]个格子。

关卡可分为夜战和日战。

日战将没有战争迷雾, 但敌人依然会有视野范围, 只有视野范围内存在友方单位敌人才会行动 (或许可以有类似哨兵一类的, 向视野外的敌人发出支援信号之类的机制)

夜战则会出现战争迷雾, 在友方单位视野外的地图会被迷雾包裹。

RPG 元素:

主要为角色培养系统, 还有道具\装备系统也可以考虑。

主要角色为明日方舟 A4 预备小队成员, 目前只写了芬, 可根据角色不同加入一些不同的机制。

可以花费资源提升角色的等级、技能等级。

出战前可选择携带什么技能 (从几种里面选一个), 或者如果有装备\道具系统的话, 可以选择携带什么道具\装备进入战斗。

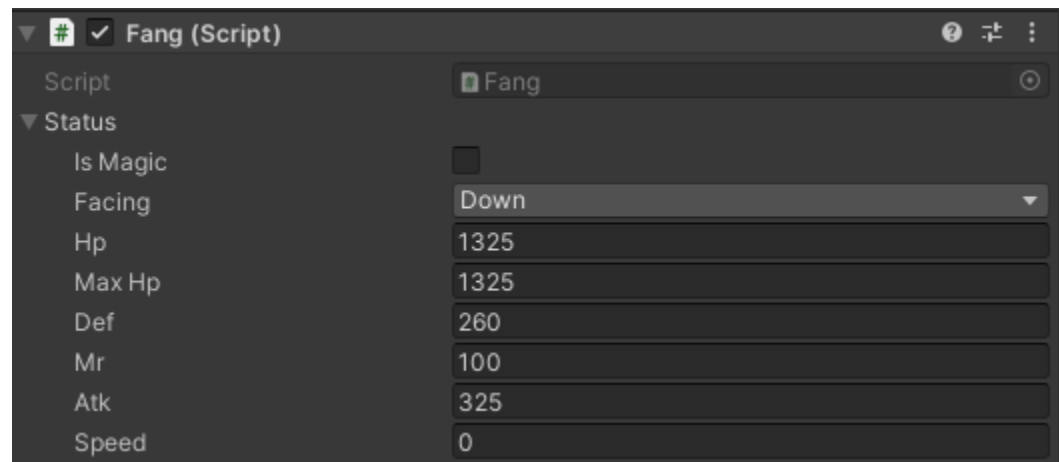
单位逻辑：

所有的单位都继承 Unit class.

每一个 Unit 都会有一个 Status class，用以表示单位的血量攻击等数据。

Facing 为朝向，影响攻击范围的朝向，同时攻击角色背面将会获得伤害加成。

```
public abstract class Unit : MonoBehaviour
{
    public Status Status;
```



此外 status 还会存有一个 List<Effect>，为 buff 或者 debuff.

```
public List<Effect> effects; //for storing buff/debuff.
```

Effect 为 abstract class. 具体的 effect 需要继承 Effect.

有一个改变属性的 Effect，叫做 StatusChangeEffect，可供参考。

```
public class StatusChangeEffect : Effect
{
    int atkChange;
    int defChange;
    int mrChange;
    int maxHpChange;

    3 references
    public override void effect()
    {
        unit.Status.effects.Add(this);
        unit.Status.atk += atkChange;
        unit.Status.def += defChange;
        unit.Status.mr += mrChange;
        unit.Status.maxHp += maxHpChange;
    }

    3 references
    public override void remove()
    {
        unit.Status.atk -= atkChange;
        unit.Status.def -= defChange;
        unit.Status.mr -= mrChange;
        unit.Status.maxHp -= maxHpChange;
        unit.Status.effects.Remove(this);
    }

    2 references
    public StatusChangeEffect(Unit unit, int period = 1, int atkChange = 0, int defChange = 0, int mrChange
    {
        this.unit = unit;
        this.period = period;
        this.atkChange = atkChange;
        this.defChange = defChange;
        this.mrChange = mrChange;
        this.maxHpChange = maxHpChange;
    }
}
```

此外一个 Unit 里面还有：

```
public int atkRange;
public int moveRange;
public List<Skills> skills;
public int cd;
public bool hasMoved = false;
public bool hasAttacked = false;
public bool hasSpecial = false;
public GameObject currentPosition;
public GameObject lastPosition;
public bool isMoving = false;
public bool isTurn = false;
public List<GameObject> viableRoutes; //s
```

Atkrange 为攻击范围。

MoveRange 为可移动范围。

这两个可考虑移进 status 里面，写的时候疏忽了，现在改可能有点麻烦（）

Skills 本来想或许可以带多个技能，就先写成 list 了，应该不碍事。

Cd 为最先可发动的技能的 cd 剩余。如果按计划的只能带一个主动技能，就没差了。也可以考虑移进 status.

剩下的 bool 都是一些 flag，不必特别在意（）

CurrentPostion 和 lastPosition 记录当前单位所在的地图格子，便于悔棋机制之类的。

CurrentPosition 比较重要，目前是通过一个叫 snapToFloor()的一个 method，使这个角色位置锁定在 currentPostion 这个格子上。如果乱动就会 snap 到别的地方。

ViableRoutes 目前写成 public 方便 debug，其用来保存目前角色的移动范围中所有可移动的格子。

```
public event Action<Unit, Unit> onAttackEvent;
public event Action<Unit, Unit> onReceiveDmgEvent;
public event Action onKillEvent;
public event Action onUpdateEvent;
public event Action onMoveEvent;
```

剩下的就是一些 event，如果可以，能用 event 的尽量利用 event 系统来写，避免太过混乱。

如 onAttackEvent，这个 Event 会在每次角色进行攻击的时候触发。

比如如果你想要写一个“攻击附带毒”的效果，可以利用这个 event.

例如：onAttackEvent += ApplyPoison(target, this);

关于 Event system 可以看一下教程之类的。

这个 class 中具体的 method 就不详写了，有比较简略的 comment.

关于继承 Unit:

写角色的时候，基本上直接继承 Unit，override 一些具有角色特别机制的 method 即可。

以角色芬（Fang）举例

```
6 public class Fang : Unit
7 {
8     //deploys character onto board.
9     public override GameObject deploy()
10    {
11        GameObject Fang = Instantiate(gameObject);
12        DefaultUnits.setDefaultFang(Fang);
13        return Fang;
14    }
15
16    @ Unity Message | 2 references
17    protected override void OnMouseOver()...
24
25    @ Unity Message | 2 references
26    protected override void OnMouseExit()...
33
34    @ Unity Message | 2 references
35    protected override void OnMouseDown()
36    {
37        if (!isMoving && (LevelController.levelController.roundStatus.Equals(RoundStatus.move) || LevelController.leve
38        {
39            onUpdateEvent -= snapToFloor;
40            onUpdateEvent += move;
41            isMoving = true;
42            MoveEvent();
43        }
44    }
45
46    //Fang's attack deals more dmg when target's hp is lower than half.
47    public override void attack(GameObject target)
48    {
49        if(target.GetComponent<Unit>().Status.hp < target.GetComponent<Unit>().Status.maxHp / 2)
50        {
51            this.Status.effects.Add(new StatusChangeEffect(this, 1, 150));
52        }
53        base.attack(target);
54    }
55
56    // Start is called before the first frame update
57    @ Unity Message | 4 references
58    protected override void Start()...
```

在继承 unit 之后，大部分都不需要改动。

需要写的只有 Deploy（spawn 该角色在地图上），根据这个照抄就行了。

OnMouseOver 以及 OnMouseExit 这两个 method 为 unity 库的 method，因为 fang 是友军角色，所以我就写了个在鼠标移到它上面的时候，会有高亮 UI 显示（敌人不需要高亮，所以 override 之后留空即可）

我 Override 了 OnMouseDown（即鼠标左键单击后）进行移动的操作，因为 fang 有一个技能是发动后可以在特殊阶段再进行一次移动，与一般 Unit 逻辑不同（一般 Unit 只可在移动阶段进行移动，fang 因为有技能的关系，可以在特殊阶段移动，所以 override 一下，加一个 if 条件。）

这之后我还 override 了 attack.

我给 fang 的一个特殊机制就是斩杀效果，敌人在半血以下会加伤。因为有特殊机制所以 override 了。

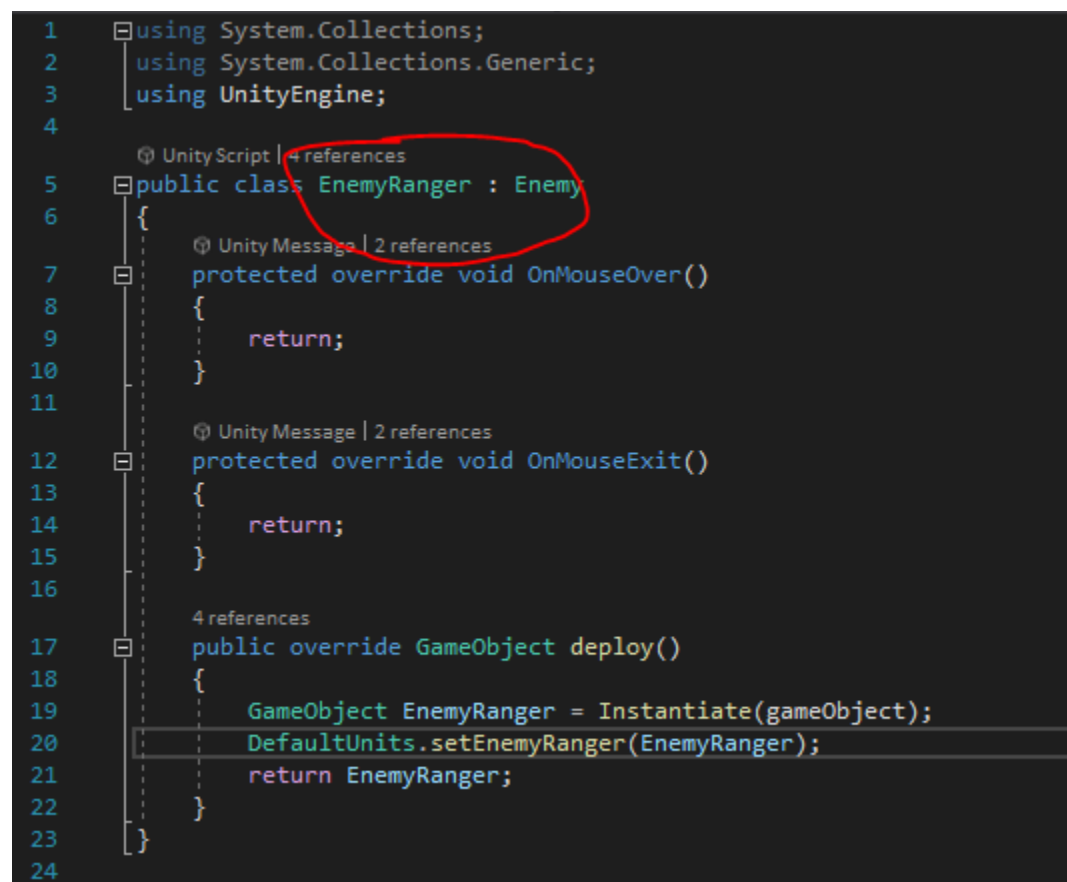
至于 start，其实本来不用 override 的，有一段代码我懒得复制回 unit 里面了，总之 start 直接复制黏贴就行了（）

关于继承结构：

所有友军皆直接继承 Unit.

注意所有敌方则是会多继承一次。

比如 EnemyRanger:



```
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class EnemyRanger : Enemy
6 {
7     protected override void OnMouseOver()
8     {
9         return;
10    }
11
12    protected override void OnMouseExit()
13    {
14        return;
15    }
16
17    public override GameObject deploy()
18    {
19        GameObject EnemyRanger = Instantiate(gameObject);
20        DefaultUnits.setEnemyRanger(EnemyRanger);
21        return EnemyRanger;
22    }
23 }
24
```

它继承 Enemy 这个 class.

而 Enemy 这个 class 则是继承自 Unit.


```

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using System;
5
6  public abstract class Enemy : Unit
7  {
8      public bool isRanged;
9      public int visionRange;
10     protected List<GameObject> vision;
11

```

关于 Enemy class:

```

@ Unity Script | 1 reference
public abstract class Enemy : Unit
{
    public bool isRanged;
    public int visionRange;
    protected List<GameObject> vision;
}

```

IsRanged 这个 bool，目前用来决定敌人 AI 的行动逻辑。比如近战的就会尽量靠近友军，远程的敌人就会尽量远离友军。

Vision range 为视野范围，打算移到 unit 中普遍使用。

Vision 这个 list 用来保存“在视野范围中的格子”。

其余的也没什么，就是行动逻辑之类的和 Unit 的不同。

具体可以看代码，也不详写了。

关于 LevelController:

这个 class 用于执行几乎所有游戏中的行动逻辑，很重要，也很长。

不过应该也没什么需要改的。

主要的变量有：

保存所有存活角色的 list:

```
public List<Unit> aliveUnits = new List<Unit>(); //list for alive units on boards.
```

在攻击阶段自动执行所有可行的战斗：

```
public List<attack> viableAttacks = new List<attack>(); //list for all viable attacks after move phase.
```

保存所有角色的速度：

```
public SortedList<int, Unit> speedList = new SortedList<int, Unit>();
```

保存单位的 prefab，用于 spawn 这些单位：

```
//prefabs for characters.  
public GameObject Fang;  
public GameObject EnemyRanger;
```

请注意 LevelController 内有一个 static levelContoller.

```
public class LevelController : MonoBehaviour  
{  
    public static LevelController levelController; //make this class an unique object in unity.
```

```
private void Awake()  
{  
    levelController = this;  
}
```

在这个脚本 awake 的时候，这个 static levelContoller 便被赋予“自己”。

通过这个办法，把这个不太能用 static 的 class 变成了一个假 static class.

如此一来，在同一个 scene 中，便可以直接 get 这个脚本中的数据。

例：

比如你想写一个核弹，一键秒杀所有单位。

```
void NUKE (){  
    List<Unit> ALLUnits = Levelcontroller.levelController.aliveUnits;  
    foreach(Unit x in ALLUnits){  
        Destroy(x);  
    }  
}
```

如此，可直接调用 levelController 中的数据或者 method.