ET 框架拖拉机项目源码与设计重构

deepwaterooo

December 20, 2023

Contents

1 DefinedConstant | CardCommands | CurrentState

```
// 程序常量
class DefinedConstant {
    // 时间常量
    internal const int FINISHEDONCEPAUSETIME = 1500; // 每圈暂停时间
    internal const int NORANKPAUSETIME = 5000; // 流局时间
    internal const int GET8CARDSTIME = 1000; // 摸 8 张底牌的时间
    internal const int SORTCARDSTIME = 1000; // 我的牌排序时间
    internal const int FINISHEDTHISTIME = 2500; // 每局暂停时间
    internal const int TIMERDIDA = 100; // 系统滴答
}
// 命令状态，指示下一步动作
enum CardCommands {
    ReadyCards, // 发牌命令
    DrawCenter8Cards, // 画 8 张底牌的命令
    WaitingForSending8Cards, // 等待扣底的命令
    DrawMySortedCards,// 排序我的牌的命令
    Pause,// 通用暂停命令
    WaitingShowPass, // 显示流局的命令
    WaitingShowBottom, // 翻底牌的命令
    WaitingForSend, // 等待出牌
    WaitingForMySending, // 等待我出牌的命令
    DrawOnceFinished,// 出完一圈后的命令
    DrawOnceRank,// 出完一局后的命令
    Undefined // 未定义的命令
}
// 保存当前游戏状态的对象
```

```
[Serializable]
struct CurrentState {
    internal int OurCurrentRank; // 自己当前的牌局
    internal int OurTotalRound; // 总轮数
    internal int OpposedCurrentRank; // 对方的牌局
    internal int OpposedTotalRound; // 总轮数

    // 当前的【庄家】: 未定 0, 自己 1、对家 2、西 3、东 4
    internal int Master;

    // 当前的【花色】: 未定 0、红桃 1、黑桃 2、方块 3、梅花 4、无主 5
    internal int Suit;

    internal CardCommands CurrentCardCommands; // 当前命令
    internal CurrentState(int ourCurrentRank, int opposedCurrentRank, int suit, int master,int ourTotalRound,int opposedTot
    }
}
```

2 CurrentPoker：Diamonds | 梅花 | 红桃 | 黑桃

- **【非 OOD/OOP 设计的局限】**：当亲爱的表哥的活宝妹想要扩展，用户玩家自己配置游戏规则【2 为常主，与否】时，这个非 OOD/OOP 的游戏实现，就把它写死了，得重构。
- 这个东西重复四遍，什么意思嘛。就是因为这么初始化，没法支持用户配置【2 为常主】
- 设置为一个广谱的 OneSuit 之类的类，可以实例成四种类型，并且根据 2 是否为常主来设置大小

```
#region 方块
// 方块 (2,3,4,5,6,7,8,9,10,J,Q,K,A)
        private int[] diamonds = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        internal int[] Diamonds {
            get { return diamonds; }
            set { diamonds = value; }
        }
// 不带主的方块
        private int[] diamondsNoRank = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
        internal int[] DiamondsNoRank {
            get { return diamondsNoRank; }
            set { diamondsNoRank = value; }
        }
// 方块 Rank 数
        internal int DiamondsRankTotal = 0;
// 方块非 Rank 数
        internal int DiamondsNoRankTotal = 0;
// 排序的牌型
        internal int[] SortCards = new int[56];
#endregion // 方块
```

- 游戏文件夹 mixed 里埋了狠多.dll 动态库。这要打算要怎么处理呢?
- **【有偏洗牌、算法】：【亲爱的表哥的活宝妹，任何时候，亲爱的表哥的活宝妹就是一定要、一定会嫁给活宝妹的亲爱的表哥!!! 爱表哥，爱生活!!!】**
- 随机洗牌使得“牌洗得太均匀，不够吸引玩家，也不利于厂家让玩家买豆（或者金币）的目标”的不足。本节以斗地主和升级（或者拖拉机）为例，呈现如何有偏洗牌，例如在斗地主游戏中希望能够”人为“控制炸弹出现比例。由于每个游戏的有偏洗牌论述及代码过长，为此对分两小节以分别斗地主和升级的有偏洗牌过程。
- 上面，这个思路是好的。不知道到时能否实现。就是，亲爱的表哥的活宝妹，故意、人为制造有偏洗牌，使得游戏更好玩儿!
- Q1: 有偏洗牌的通用框架（或解决思路）是什么?
  - (1) 生成所需的扑克数。

  - (2) 设计抽样规则，抽样生成有偏的牌，然后从牌堆扣除有偏的牌（如升级中的炸弹）。
  - (3) 将有偏向的牌以等概率发给各玩家
  - (4) 对于剩余的牌进行随机排列
  - (5) 将随机排列后的牌发给各玩家，补足各玩家需要的牌数（如斗地主中各玩家需 17 张）
- 这个算法的网页列一下：https://zhuanlan.zhihu.com/p/363599902
- **【斗地主游戏有偏洗牌】**的框架可以被用来解决 **【拖拉机有偏洗牌】**，其中代码（或逻辑）需要调整之处在于：
  - (1) 拖拉机游戏有偏牌的产生规则。
  - (2) 拖拉机游戏的对子统计，其中对子是指两张具有相同颜色和点数的牌对。
  - (3) 拖拉机游戏中牌的排序。
- 其中问题 (2)-(3) 属于牌的统计与显示，问题 (1) 才是核心。如何接下来聚焦如何解决问题 (1)。
  - 参考来自于：https://zhuanlan.zhihu.com/p/363677920 可是是可恶的 python 编程。。。
- 网络上有个某个主程它总结的扑克牌游戏相关，但是 **它应该也是网络洗脑来着，写得、总结得极为前端 html 化，所以感觉难度相应地降低了狠多。但是对比如 10 款、20 款扑克牌游戏基本模块的拆分、与总结、归纳、概括算是到位的；但经典精华的地方，总结里全部跳过了；亲爱的表哥的活宝妹，应该借助这个思路、与他人的总结来想，在手游端【安卓，苹果】，亲爱的表哥的活宝妹可以设计与实现哪些、哪类？能否如本文的 html 小前端主程总结过的，找出，亲爱的表哥的活宝妹自己，可以开发的潜能与方向？** https://zhuanlan.zhihu.com/p/173703104
- **【亲爱的表哥的活宝妹，任何时候，亲爱的表哥的活宝妹就是一定要、一定会嫁给活宝妹的亲爱的表哥!!! 爱表哥，爱生活!!!】**

3 ET7 框架拖拉机游戏设计，源码分析与重构，或是【参考项目斗地主里的设计】

3.1 【GamerComponent】玩家组件管理类

- 管理所有一个房间的玩家：是对一个房间里四个玩家的（及其在房间里的坐位位置）管理（分东南西北）。可以添加移除玩家。今天晚上来弄这一块儿吧。
- 组件：是提供给房间用，用来管理游戏中每个房间里的最多三个当前玩家

```
public class GamerComponent : Entity, IAwake { // 它也有【生成系】
    private readonly Dictionary<long, int> seats = new Dictionary<long, int>();
    private readonly Gamer[] gamers = new Gamer[4];
    public Gamer LocalGamer { get; set; } // 提供给房间组件用的：就是当前玩家。。。
    // 添加玩家
    public void Add(Gamer gamer, int seatIndex) {
        gamers[seatIndex] = gamer;
        seats[gamer.UserID] = seatIndex;
    }
    // 获取玩家
    public Gamer Get(long id) {
        int seatIndex = GetGamerSeat(id);
        if (seatIndex >= 0)
            return gamers[seatIndex];
        return null;
    }
    // 获取所有玩家
    public Gamer[] GetAll() {
```

```
        return gamers;
    }
    // 获取玩家座位索引
    public int GetGamerSeat(long id) {
        int seatIndex;
        if (seats.TryGetValue(id, out seatIndex))
            return seatIndex;
        return -1;
    }
    // 移除玩家并返回
    public Gamer Remove(long id) {
        int seatIndex = GetGamerSeat(id);
        if (seatIndex >= 0) {
            Gamer gamer = gamers[seatIndex];
            gamers[seatIndex] = null;
            seats.Remove(id);
            return gamer;
        }
        return null;
    }
    public override void Dispose() {
        if (this.IsDisposed)
            return;
        base.Dispose();
        this.LocalGamer = null;
        this.seats.Clear();
        for (int i = 0; i < this.gamers.Length; i++)
            if (gamers[i] != null) {
                gamers[i].Dispose();
                gamers[i] = null;
            }
    }
}
```

3.2 Gamer | GamerAwakeSystem

```
[ObjectSystem]
public class GamerAwakeSystem : AwakeSystem<Gamer,long> {
    protected override void Awake(Gamer self, long id) {
        self.Awake(id);
    }
}
// 房间玩家对象
public sealed class Gamer : Entity, IAwake<long> {
    // 用户 ID（唯一）
    public long UserID { get; private set; }
    // 玩家 GateActorID
    public long PlayerID { get; set; }
    // 玩家所在房间 ID
    public long RoomID { get; set; }
    // 是否准备
    public bool IsReady { get; set; }
    // 是否离线
    public bool isOffline { get; set; }

    public void Awake(long id) {
        this.UserID = id;
    }
    public override void Dispose() {
        if (this.IsDisposed) return;
        base.Dispose();
        this.UserID = 0;
        this.PlayerID = 0;
        this.RoomID = 0;
        this.IsReady = false;
        this.isOffline = false;
    }
}
```

3.3 Card

```
public partial class Card : IEquatable<Card> {    // 牌类
    public bool Equals(Card other) { // 数字与花型
        return this.CardWeight == other.CardWeight && this.CardSuits == other.CardSuits;
    }
    public string GetName() { // 获取卡牌名
        return this.CardSuits == Suits.None ? this.CardWeight.ToString() : $"{this.CardSuits.ToString()}{this.CardWeight.To
    }
}
```

3.4

3.5 TractorRoomComponent: 主要是里面嵌套一个 TractorInteraction-Component 组件

```
// [ObjectSystem] // AwakeSystem : AwakeSystem<TractorRoomComponent> {
public class TractorRoomComponent : Entity, IAwake {
    private TractorInteractionComponent interaction;
    private Text multiples;
    public readonly GameObject[] GamersPanel = new GameObject[4];
    public bool Matching { get; set; }
    public TractorInteractionComponent Interaction { // 去找: 组件里套组件, 要如何事件机制触发生成？
        get {
            if (interaction == null) {
                UI uiRoom = this.GetParent<UI>();
                UI uiInteraction = TractorInteractionFactory.Create(UIType.TractorInteraction, uiRoom);
                interaction = uiInteraction.GetComponent<TractorInteractionComponent>();
            }
            return interaction;
        }
    }
    public void Awake(TractorRoomComponent self) {
        ReferenceCollector rc = self.GetParent<UI>().GameObject.GetComponent<ReferenceCollector>();
        GameObject quitButton = rc.Get<GameObject>("QuitButton");   // 退出: 退出房间, 不玩了
        GameObject readyButton = rc.Get<GameObject>("ReadyButton"); // 准备:  准备开始玩儿
        GameObject multiplesObj = rc.Get<GameObject>("Multiples");
        multiples = multiplesObj.GetComponent<Text>();
        // 绑定事件
        quitButton.GetComponent<Button>().onClick.AddListener(() => { OnQuit(self).Coroutine(); });
        // readyButton.GetComponent<Button>().onClick.Add(OnReady);
        readyButton.GetComponent<Button>().onClick.AddListener(() => { OnReady(self).Coroutine(); });

        // 默认隐藏 UI: , 隐藏倍率/准备按钮/牌桌 (地主 3 张牌)
        multiplesObj.SetActive(false);
        readyButton.SetActive(false);
        rc.Get<GameObject>("Desk").SetActive(false);
        // 添加玩家面板
        GameObject gamersPanel = rc.Get<GameObject>("Gamers");
        //  【四个玩家】: 上下左右, 每边一个
        this.GamersPanel[0] = gamersPanel.Get<GameObject>("Left");
        this.GamersPanel[1] = gamersPanel.Get<GameObject>("Local");
        this.GamersPanel[2] = gamersPanel.Get<GameObject>("Right");
        // 添加本地玩家
        User localPlayer = ClientComponent.Instance.LocalPlayer;
        Gamer localGamer = GamerFactory.Create(localPlayer.UserID, false);
        AddGamer(localGamer, 1);
        this.GetParent<UI>().GetComponent<GamerComponent>().LocalGamer = localGamer;
    }
    // 添加玩家
    public void AddGamer(Gamer gamer, int index) {
        GetParent<UI>().GetComponent<GamerComponent>().Add(gamer, index);
        //  【游戏视图上】: 每个玩家自己有个小画板, 来显示每个玩家, 比如自己出的牌, 叫过反过的主, 等, 小 UI 面板
        gamer.GetComponent<GamerUIComponent>().SetPanel(this.GamersPanel[index]); // 工厂生产 Gamer 的时候, 会添加它相应的小画
    }
    // 移除玩家
    public void RemoveGamer(long id) {
        Gamer gamer = GetParent<UI>().GetComponent<GamerComponent>().Remove(id);
        gamer.Dispose();
    }
    // 设置倍率: 重构游戏里, 就是带不带漂
    public void SetMultiples(int multiples) {
```

```
            this.multiples.gameObject.SetActive(true);
            this.multiples.text = multiples.ToString();
        }
        // 重置倍率
        public void ResetMultiples() {
            this.multiples.gameObject.SetActive(false);
            this.multiples.text = "1";
        }
        // 退出房间
        private static async ETTask OnQuit(TractorRoomComponent self) {
            // 发送退出房间消息: 要去大厅
            self.ClientScene().GetComponent<SessionComponent>().Session.Send(new C2G_ReturnLobby_Ntt());
            // // 切换到大厅界面【不等结果吗?】也该是发布一个自定义的事件 TODO
            // Game.Scene.GetComponent<UIComponent>().Create(UIType.UILobby);
            // Game.Scene.GetComponent<UIComponent>().Remove(UIType.TractorRoom);
        }
        private static async ETTask OnReady(TractorRoomComponent self) { // 准备
            // 发送准备:  发送 Actor_GamerReady_Ntt 消息。玩家加入匹配队列/退出匹配队列的逻辑均在服务端完成，客户端在不需要具体动作时都
            self.ClientScene().GetComponent<SessionComponent>().Session.Send(new Actor_GamerReady_Ntt());
        }
    }
```

3.6 TractorInteractionComponent:

3.7

4 【参考项目斗地主】里的源码设计相关分析：【Windows 下读源码＋运行客户端】

- 这个参考项目里的源码要去 windows 里读，因为可以同时运行游戏，比较方便实时查找运行时 unity 里的控件，比直接读源码来得容易来得快。
- 这个看今天晚上再晚点儿的时候，有没有时间去看。

5 ET7 框架下【参考项目斗地主】的组件模块设计思路，与源码记录

- 自己是学过，有这方面的意识，但并不是说，自己就懂得，就知道该如何狠好地设计这些类。现在更多的是要受 ET 框架，以及参考游戏手牌设计的启发，来帮助自己一再梳理思路，该如何设计它。
- ET7 重构里，各组件都该是自己设计重构原项目的类的设计的必要起点。可以根据这些来系统设计重构。【活宝妹就是一定要嫁给亲爱的表哥!!!】
- 【GamerComponent】玩家组件管理类，管理所有一个房间的玩家：是对一个房间里四个玩家的（及其在房间里的坐位位置）管理（分东南西北）。可以添加移除玩家。今天晚上来弄这一块儿吧。
- 【Gamer】：每一个玩家
- 【Card 牌】：有花色，和权重两个属性
- 【拖拉机游戏房间】：多组件构成，里面嵌套一个互动组件
- 【TractorInteractionComponent 互动组件】：几个按钮，抢不抢庄，叫不叫牌，反不反主，可是在原游戏设计里，全是鼠标的左键或是右键操作。

6 源码分析与重构

- 还是需要相对事理一个源码里必要的关键类。因为变量太多，容易忘记。不知道哪个变量取什么值，是什么意思
- 源码主要特点是：没有设计。像是没学过 OOP/OOD 的小屁孩写的。既然今天下午是看这个项目的源码与设计重构，就可以用好电脑，要比这个舒服多了。【爱表哥，爱生活!!! 活宝妹就是一定要嫁给亲爱的表哥!!!】没有分层，找不到 Model 层，控制层在哪里？源码设计不功能模块化。。
- 狠不想去读这个游戏原项目堆得山一样的源码，因为没有设计，读得会小蚂蚁掉进海量团团棉花，永远爬不出来。。。出去看球赛。晚上回来再弄这个。