

# 关于炉石兄弟的目录和文件介绍



wjh 收录于 [Hearthbuddy](#)

□ 2020-03-13 □ 2024-02-11 □ 约 24400 字 □ 预计阅读 49 分钟

## # 更新日志

- 2024-02-11: 调整格式，为博客更变做准备
- 2020-08-01: 修正一些描述错误，以及更新对于SIM中的弃牌函数理解的添加
- 2020-06-07: 修正一些错误
- 2020-05-23: 修正一些错误
- 2020-03-23: 更新\*\*MiniSimulator.cs(ai目录)\*\*的内容
- 2020-03-21: 修正并更新**Minion.cs(ai目录)**的内容，感谢魔王的骑士和**不正不歪 0\_0!**
- 2020-03-20: 更新\*\*Minion.cs(ai目录)\*\*的内容，之后更新会更慢一些，先更新一些关于修复旧版兄弟的内容
- 2020-03-16: 更新\*\*Hrtprozis.cs(ai目录)\*\*的内容
- 2020-03-15: 更新\*\*CardDefs.xml(data目录)\*\*的内容
- 2020-03-14: 更新\*\*\_settings.txt、\_combo.txt(behavior目录)、SIM事件和常用函数(card目录)和CardDB.cs、Handmanager.cs(ai目录)\*\*的内容
- 2020-03-13: 更新\*\*Ai.cs(ai目录)和\_mulligan.txt(behavior目录)\*\*的内容
- 2020-03-12: 创建此页面，目的是为了引导大家对兄弟的代码有所理解。如果存在技术上的问题，请留言回复，也可以留言鼓励一下哦~

## # 目录

### | 目录介绍

首先进入炉石兄弟的根目录，可以看到以下文件夹，以下文件值得我们关注，我会对其中代码——说明。

- Plugins (插件相关文件)
- Routines (策略相关文件)
- DefaultRoutine
  - Silverfish
    - ai
      - action.cs
      - ActionNormalizer.cs
      - Ai.cs

- Behavior.cs
- BehaviorControl.cs
- BehaviorRush.cs
- BehaviourMana.cs
- BoardTester.cs
- CardDB.cs
- CardDB\_cardIDEnum.cs
- CardDB\_cardName.cs
- CardDB\_getSimCard.cs
- ComboBreaker.cs
- Deckmanager.cs
- EnemyTurnSimulator.cs
- Handmanager.cs
- Helpfunctions.cs
- Hrtprozis.cs
- Minion.cs
- MiniSimulator.cs
- MiniSimulatorNextTurn.cs
- Movegenerator.cs
- Mulligan.cs
- PenaltyManager.cs
- PenTemplate.cs
- Playfield.cs
- Probabilitymaker.cs
- Questmanager.cs
- RulesEngine.cs
- Settings.cs
- SimTemplate.cs
- TAGGS.cs
- Weapon.cs
- behavior
  - control
    - Behavior控场模式.cs
  - rush
    - Behavior怼脸模式.cs
- cards
- data
  - CardDefs.xml
- penalties

- UltimateLogs
- DefaultRoutine.cs
- DefaultRoutineSettings.cs
- SettingsGui.xaml
- silverfish\_HB.cs
- Hearthbuddy.exe (炉石兄弟主程序)

## | ai目录

### | Action.cs(回合操作)

```

□ C#
1 public enum actionEnum
2 {
3     endturn = 0,
4     playcard,
5     attackWithHero,
6     useHeroPower,
7     attackWithMinion
8 }
9 public class Action
10 {
11     public actionEnum actionType;
12     public Handmanager.Handcard card;
13     //public int cardEntitiy;
14     public int place; // = target where card/minion is placed
15     public Minion own;
16     public Minion target;
17     public int druidchoice; // 1 left card, 2 right card
18     public int penalty;
19     public int turn = -1;
20     public int prevHpOwn = -1;
21     public int prevHpTarget = -1;
22 }

```

这些是从文件中摘抄出来的一些重要内容，我分别来介绍。

### | actionEnum 回合状态枚举

首先是actionEnum枚举类型，在这个枚举类型中包括着不同的对战方式。

```

□ C#
1 public enum actionEnum
2 {
3     endturn = 0, // 结束此回合
4     playcard, // 出一张牌，出的牌在card中

```

```

5      attackWithHero, // 英雄进行攻击, target作为目标, own通常是我方英雄
6      useHeroPower,
7      // 使用英雄技能, target作为目标
8      // (英雄技能无攻击目标的英雄则为null, 如Mage)
9      attackWithMinion // 随从进行攻击, target作为目标, own是用于攻击的随从
10 }

```

## | Action 回合类

接下来对每个成员进行介绍

```

□ C#
1  public class Action
2  {
3      actionEnum actionType; // 用于记录操作的类型。
4      Handmanager.Handcard card;
5      // 记录出的牌, 在attackWithHero和attackWithMinion操作中为null
6      int place;
7      // 用于记录位置, 如actionType为playcard时就会有内容,
8      // 比如出一个随从就会标记出放置的位置 (最左边为0),
9      // 如果是出一张法术牌的话, 具体内容没有研究, 应该是有赋值的
10
11     Minion own;
12     Minion target;
13     // 这两个成员, 顾名思义一个是 我方随从 (英雄), 另一个是攻击目标。
14
15     int druidchoice;
16     // 用于抉择牌选择
17     // 如果是未修改过的兄弟, 抉择牌的编号分别是 (0: 中间 1: 左边 2: 右边)
18     int penalty;
19     // 惩罚值, 对于这个操作给出多少的惩罚
20     // 值越大越不推荐这样下, 如果值为负数则是非常推荐。
21     int turn = -1;
22     // 记录的应该是此回合中的第几步操作, 初始值为-1 (不确定)
23     int prevHpOwn = -1; // 不确定
24     int prevHpTarget = -1; // 不确定
25 }

```

## | Ai.cs(AI调用和调试)

~~Ai.cs中的内容比较复杂, 涉及到相关的算法知识, 如果只是萌新可以考虑跳过这一块。~~ Ai.cs中的一些成员属性就不在这里阐述了, 之后在\_setting.txt文件的设置中说明, 直接来看**Ai.cs**中最核心的函数**doallmoves**

```

□ C#
1  private void doallmoves(bool test, bool isLethalCheck)
2  {
3      //set maxwide to the value for the first-turn-sim.

```

```
4      foreach (EnemyTurnSimulator ets in enemyTurnSim)
5      {
6          ets.setMaxwide(true);
7      }
8      foreach (EnemyTurnSimulator ets in enemySecondTurnSim)
9      {
10         ets.setMaxwide(true);
11     }
12
13     //if (isLethalCheck) this.posmoves[0].enemySecretList.Clear();
14     this.posmoves[0].isLethalCheck = isLethalCheck;
15     this.mainTurnSimulator.doallmoves(this.posmoves[0]);
16
17     bestplay = this.mainTurnSimulator.bestboard;
18     float bestval = this.mainTurnSimulator.bestmoveValue;
19
20     help.loggonoff(true);
21     help.logg("-----");
22     if (bestplay.ruleWeight != 0) help.logg("ruleWeight " + bestplay.ruleWeight *
23     if (settings.printRules > 0)
24     {
25         String[] rulesStr = bestplay.rulesUsed.Split('@');
26         foreach (string rs in rulesStr)
27         {
28             if (rs == "") continue;
29             help.logg("rule: " + rs);
30         }
31     }
32     help.logg("value of best board " + bestval);
33
34     this.bestActions.Clear();
35     this.bestmove = null;
36     ActionNormalizer an = new ActionNormalizer();
37     //an.checkLostActions(bestplay, isLethalCheck);
38     if (settings.adjustActions > 0) an.adjustActions(bestplay, isLethalCheck);
39     foreach (Action a in bestplay.playactions)
40     {
41         this.bestActions.Add(new Action(a));
42         a.print();
43     }
44
45     if (this.bestActions.Count >= 1)
46     {
47         this.bestmove = this.bestActions[0];
48         this.bestActions.RemoveAt(0);
49     }
50     this.bestmoveValue = bestval;
51
52     if (bestmove != null && bestmove.actionType != actionEnum.endturn) // save the
53     {
```

```
54         this.nextMoveGuess = new Playfield();
55
56         this.nextMoveGuess.doAction(bestmove);
57     }
58     else
59     {
60         nextMoveGuess.mana = -100;
61     }
62
63     if (isLethalCheck)
64     {
65         this.lethalMissing = bestplay.enemyHero.armor + bestplay.enemyHero.Hp;//RR
66         help.logg("missing dmg to lethal " + this.lethalMissing);
67     }
68 }
```

## | doallmoves 参数解析

接下来，我将详细解析函数的每个步骤，首先从它接收的两个参数开始：

□ C#

```
1 private void doallmoves(bool test, bool isLethalCheck)
```

- 第一个参数 `test`，主要用于调试，当设置为`true`时，启用调试模式。
- 第二个参数 `isLethalCheck`，用于执行**斩杀检查**。我的代码跟踪发现，当此参数为`true`时，函数主要执行可能导致对手直接被击败的操作模拟，以判断是否存在斩杀可能。如果无法实现斩杀，将重新以 `isLethalCheck = false` 进行计算。这也解释了为何有时会发生误判斩杀的情况，解决此问题的详细方法会在 `PenalityManager.cs` 中提出。

## | Maxwide 是什么？

□ C#

```
1 //set maxwide to the value for the first-turn-sim.
2 foreach (EnemyTurnSimulator ets in enemyTurnSim)
3 {
4     ets.setMaxwide(true);
5 }
6 foreach (EnemyTurnSimulator ets in enemySecondTurnSim)
7 {
8     ets.setMaxwide(true);
9 }
```

对于上述代码，有些读者可能会感到困惑：**Maxwide**是什么？**enemyTurnSim**和**enemySecondTurnSim**又分别是什么？我的理解是，这段代码负责为接下来的状态搜索操作进行**初始化**。其中，“**Wide**”指的是搜索过程中的广度，“**Deep**”则指的是深度。

这里需要一些树相关的知识，如果对**树的概念**有所理解的可以跳过这部分内容。

## 搜索树

如图展示，**A节点** 被定义为该树的**根节点(root)**。位于树深度为1的层级（**深度 = 1**）的 **B、C、D、E、F、G节点**源自**A节点**，因此我们将**A节点**视为这些节点的父节点。在搜索树的语境下，这表示这些节点是通过**A节点**状态的转换而来的，在深度为1的这个层级上，层宽度为6（**广度 = 6**）。

在炉石兄弟里：

- **根节点 (A节点)** 对应于不采取任何行动的场景，可以理解为对手回合结束后的游戏状态。
- **深度 (Deep)** 用于表示操作的序号，例如在一个回合中依次执行了3项操作。
- **广度 (Wide)** 表示在给定深度下，可供选择的操作数量。

以一个具体例子说明，以下三项操作对应着从 **A** 到 **E**，再到 **J**，最后到 **P** 的路径：

1. 使用我的随从A攻击对方英雄（达到E节点，深度 = 1）。
2. 使用英雄技能（达到J节点，深度 = 2）。
3. 结束回合（达到P节点，深度 = 3）。

炉石兄弟将在每一步操作结束时计算并比较各路径的评分，选择评分更高的路径进行操作。

在掌握了前述知识之后，我们对于**MaxWide**的概念会更加易于理解。

设想这样一个场景：我们手中有十张可出的牌，而场上双方均已满员。在这种情况下，我们能采取的操作策略数量是何等庞大？毫无疑问，这个数字非常巨大。因此，在此时，可选范围（**Wide**）实在是过于广泛，且对于每个选择点，我们还需进一步探索其后续路径，这种增长速度是指数级的。

为了应对这种情况，引入了**MaxWide**概念，通过限制搜索的最大广度来缩减搜索范围，从而降低搜索所需的时间并防止程序超时。然而，需要注意的是，在缩小搜索范围的过程中，遗漏某些最优解是难以避免的。

因此，根据系统配置的不同，调整**MaxWide**的值能够在提升AI智能与缩短搜索时间之间找到平衡点。配置较高时，增加**MaxWide**能有效提升AI的表现；相反，配置较低时，减少**MaxWide**则有助于减少搜索时间，防止出牌超时。

## | 斩杀检测状态转移并转移doallmoves调用

□ C#

```
1 this.posmoves[0].isLethalCheck = isLethalCheck;  
2 this.mainTurnSimulator.doallmoves(this.posmoves[0]);
```

这段代码将 `isLethalCheck` 参数传递给了 `Playfield` 对象。那么，什么是 `Playfield`？暂时可以将其理解为游戏中的**场景数据**，包括当前的游戏状态、玩家的手牌、场上的随从等信息。对于 `Playfield` 的详细解释和运用，我们会在后续内容中深入探讨。

接下来，代码调用了 `mainTurnSimulator.doallmoves` 函数。这个函数的调用标志着搜索和模拟运算的开始，其详细的逻辑和实现可以在 `MiniSimulator.cs` 文件中找到详细的解析。

## 剩余部分内容

之后的内容并不是核心重点，因此将以注释的形式进行简要说明。

□ C#

```
1 bestplay = this.mainTurnSimulator.bestboard; // 上面函数计算之后的最优场面
2 float bestval = this.mainTurnSimulator.bestmoveValue; // 上面最优场面的评分
3
4 // 输出与rule相关信息
5 help.loggonoff(true);
6 help.logg("-----");
7 if (bestplay.ruleWeight != 0) help.logg("ruleWeight " + bestplay.ruleWeight * -1);
8 if (settings.printRules > 0)
9 {
10     String[] rulesStr = bestplay.rulesUsed.Split('@');
11     foreach (string rs in rulesStr)
12     {
13         if (rs == "") continue;
14         help.logg("rule: " + rs);
15     }
16 }
17 // 输出与rule相关信息结束
18 help.logg("value of best board " + bestval); // 输出最优场面评分
19
20 this.bestActions.Clear(); // 清除上次回合计算的操作方法
21 this.bestmove = null; // 清除上次回合计算的操作方法
22
23 // 接下来两行是对操作重排使得其先打出AOE伤害，在_setting.txt中设定，默认关闭，实际使用效果
24 ActionNormalizer an = new ActionNormalizer();
25 if (settings.adjustActions > 0) an.adjustActions(bestplay, isLethalCheck);
26
27
28
29 foreach (Action a in bestplay.playactions)
30 {
31     this.bestActions.Add(new Action(a));
32     a.print();
33 }
34 // 输出场面信息
35 if (this.bestActions.Count >= 1)
36 {
37     this.bestmove = this.bestActions[0];
38     this.bestActions.RemoveAt(0);
39 }
40 this.bestmoveValue = bestval;
41
42 // 如果不是回合结束操作，则模拟一下下一个操作
43 if (bestmove != null && bestmove.actionType != actionEnum.endturn) // save the guess
44 {
45     this.nextMoveGuess = new Playfield();
```



```

46     this.nextMoveGuess.doAction(bestmove);
47 }
48 else
49 {
50     //如果是回合结束操作，下一个操作则不进行（不进行的方法是让法力值-100，使其无操作可做）
51     nextMoveGuess.mana = -100;
52 }
53
54 //斩杀信息输出，输出斩杀所需的伤害
55 if (isLethalCheck)
56 {
57     this.lethalMissing = bestplay.enemyHero.armor + bestplay.enemyHero.Hp;//RR
58     help.logg("missing dmg to lethal " + this.lethalMissing);
59 }
60
61 这个文件中另一个重要的函数与调试AI息息相关。这里我直接在注释中写出我的一些理解
62 public List<double> autoTester(bool printstuff, string data = "", int mode = 0) //
63 {
64     //游戏信息（场面）还原，
65     List<double> retval = new List<double>();
66     double calcTime = 0;
67     help.logg("simulating board ");
68
69     BoardTester bt = new BoardTester(data);
70     if (!bt.dataareaded) return retval;
71     hp.printHero();
72     hp.printOwnMinions();
73     hp.printEnemyMinions();
74     hm.printcards();
75
76
77     //开始计算
78     posmoves.Clear();
79     Playfield pMain = new Playfield();
80     pMain.print = printstuff;
81     posmoves.Add(pMain);
82     //输出场面信息
83     foreach (Playfield p in this.posmoves)
84     {
85         p.printBoard();
86     }
87     help.logg("ownminionscount " + posmoves[0].ownMinions.Count);
88     help.logg("owncardscount " + posmoves[0].owncards.Count);
89
90     foreach (var item in this.posmoves[0].owncards)
91     {
92         help.logg("card " + item.card.name + " is playable :" + item.canplayCard(p
93     }
94     help.logg("ability " + posmoves[0].ownHeroAblility.card.name + " is playable :
95     //输出场面信息结束

```

```
96     DateTime strt = DateTime.Now;
97     // 斩杀判定
98     if (mode == 0 || mode == 1)
99     {
100         // 可以看到这个函数正是我们上面所讲的，这里第二个参数传入了true
101         // 则说明这里这个分支则是兄弟进行扎煞判定的地方
102         doallmoves(false, true);
103         calcTime = (DateTime.Now - strt).TotalSeconds;
104         help.logg("calculated " + calcTime);
105         retval.Add(calcTime);
106     }
107     // 这里要补充一下，如果兄弟判定对面血量为负数(已斩杀)，则会将场面评分提升到10000以上
108     // 而这里的bestmoveValue > 5000，则说明场面已经大概可以斩杀了。
109     // 但是为什么是五千不是一万？
110     // 这个问题的原因在于兄弟是会对当前回合和我方下一回合进行模拟的，而默认的权重分配是0.
111     // 如果这个回合斩杀了，兄弟可能就不会对下个回合进行模拟，所以如果这个回合评分为10000
112     // 以上只是个人猜测，有少量的代码可证明，知道原因的可以在下面评论哦
113     if (Settings.Instance.berserkIfCanFinishNextTour > 0 && bestmoveValue > 5000)
114     {
115
116     }
117     else if (bestmoveValue < 10000)
118     {
119         // 到了这里，就是正常的模拟了，会正常模拟所有可能的操作方法。
120         // 如果是调试，应该进入这里的**doallmoves(false, false);**函数进行调试。
121         // normal
122         if (mode == 0 || mode == 2)
123         {
124             posmoves.Clear();
125             pMain = new Playfield();
126             pMain.print = printstuff;
127             posmoves.Add(pMain);
128             strt = DateTime.Now;
129             doallmoves(false, false);
130             calcTime = (DateTime.Now - strt).TotalSeconds;
131             help.logg("calculated " + calcTime);
132             retval.Add(calcTime);
133         }
134     }
135
136     if (printstuff)
137     {
138         this.mainTurnSimulator.printPosmoves();
139         simulateWholeTurn();
140         help.logg("calculated " + calcTime);
141     }
142
143     return retval;
144 }
```

这个文件的内容终于写完了，比我想象中的要快一些。对于搜索树的介绍是我一时兴起，写完之后才发现这个**AI.cs**文件并不是AI计算的主要文件，那就充当对于理解\*\*MiniSimulator.cs(AI功能)\*\*的铺垫吧。

## | CardDB.cs(卡牌数据)

### | cardtype 解析

□ C#

```
1 public enum cardtype
2 {
3     NONE, //未知
4     MOB = 4, //随从
5     SPELL = 5, //法术
6     WEAPON = 7, //武器
7     HEROPWR = 10, //英雄技能
8     ENCHANTMENT = 6, //增幅（例如：变形术，救赎，力量的代价，自然之力的附加效果）
9     HERO = 3, //英雄卡
10 }
```

### | ErrorType 解析(部分)

□ C#

```
1 public enum ErrorType2
2 {
3     REQ_MINION_TARGET = 1, 随从目标
4     REQ_FRIENDLY_TARGET = 2, 友方目标
5     REQ_ENEMY_TARGET = 3, 敌方目标
6     REQ_DAMAGED_TARGET = 4, 损伤目标
7     REQ_MAX_SECRETS = 5, 最大奥秘
8     REQ_FROZEN_TARGET = 6, 冻结目标
9     REQ_CHARGE_TARGET = 7, 冲锋目标
10    REQ_TARGET_MAX_ATTACK = 8, 最大攻击力目标 有param=
11    REQ_NONSELF_TARGET = 9, 非自己目标
12    REQ_TARGET_WITH_RACE = 10, 种族目标 有param=
13    REQ_TARGET_TO_PLAY = 11, 需要目标
14    REQ_NUM_MINION_SLOTS = 12, 随从数目插槽 有param=
15    REQ_WEAPON_EQIPPED = 13, 武器装备, 需要武器
16    REQ_HERO_TARGET = 17, 英雄目标
17    REQ_TARGET_IF_AVAILABLE = 22, 有目标如果用（抉择星辰降落，巫医）
18    REQ_MINIMUM_ENEMY_MINIONS = 23, 最少的敌方随从 有param=
19    REQ_TARGET_FOR_COMBO = 24, 连击有目标
20    REQ_HERO_OR_MINION_TARGET = 33, 英雄或随从目标
21    REQ_TARGET_MIN_ATTACK = 41, 最小攻击力, 有param=
22    REQ_MINIMUM_TOTAL_MINIONS = 45, 需要最少随从数目, 有param=
23    REQ_MUST_TARGET_TAUNTER = 46, 目标必须是嘲讽
24    REQ_UNDAMAGED_TARGET = 47, 目标必须是未受伤的
```

```

25     REQ_TARGET_IF_AVAILABLE_AND_DRAGON_IN_HAND = 51, 有龙牌在手才有目标
26     REQ_LEGENDARY_TARGET = 52, 传说目标
27     REQ_FRIENDLY_MINION_DIED_THIS_TURN = 53, 需要一个死亡的友方随从在当前回合死亡
28     REQ_FRIENDLY_MINION_DIED_THIS_GAME = 54, 需要一个死亡的友方随从
29 }

```

## | Card 成员解析

□ C#

```

1 public class Card
2 {
3     public cardName name = cardName.unknown; // 名称
4     public int race = 0; // 种族
5     // 14 鱼人 15 恶魔 17 机械 18 元素 20 野兽 21 图腾 23 海盗 24 龙
6     public int rarity = 0; // 稀有度
7     // 1 普通白 2 基础无 3 稀有蓝 4 史诗紫 5 传说橙
8     public int cost = 0; // 费用
9     public int Class = 0; // 职业
10    // 2 德鲁伊 3 猎人 4 法师 5 圣骑士 6 牧师 7 潜行者 8 萨满 9 术士 10 战士 11 梦境牌 12 中立
11    public cardtype type = CardDB.cardtype.NONE; // 类别
12    public int Attack = 0; // 攻击力
13    public int Health = 0; // 血量
14    public int Durability = 0; // 耐久值
15    public bool tank = false; // 嘲讽
16    public bool Silence = false; // 沉默
17    public bool choice = false; // 抉择
18    public bool windfury = false; // 风怒
19    public bool poisonous = false; // 剧毒
20    public bool lifesteal = false; // 吸血
21    public bool reborn = false; // 复生
22    public bool deathrattle = false; // 亡语
23    public bool battlecry = false; // 战吼
24    public bool discover = false; // 发现
25    public bool oneTurnEffect = false;
26    public bool Enrage = false; // 愤怒
27    public bool Aura = false; // 光环
28    public bool Combo = false; // 连击
29    public int overload = 0; // 超载
30    public bool untouchable = false; // 不可被攻击
31    public bool Stealth = false; // 潜行
32    public bool Freeze = false; // 冰冻
33    public bool Shield = false; // 圣盾
34    public bool Charge = false; // 突袭
35    public bool Modular = false; // 磁力
36    public bool Rush = false; // 冲锋
37    public bool Secret = false; // 奥秘
38    public bool Quest = false; // 任务
39    public bool Morph = false; // 变形
40    public bool Spellpower = false; // 法术伤害

```

```
41     public bool Inspire = false; //激励
42 }
```

## | 添加新卡机制、修复卡牌特效修改点

修改手牌费用识别，比如海巨人

□ C#

```
1 public int getManaCost(Playfield p, int currentcost)
2 public int calculateManaCost(Playfield p)
```

修改攻击目标，比如矮人神射手

□ C#

```
1 public List<Minion> getTargetsForCard(Playfield p, bool isLethalCheck, bool own)
2 public List<Minion> getTargetsForHeroPower(Playfield p, bool own)
```

## | BUG 修正

### | BUG1：未读取到卡牌名称

修改位置在原来的 switch (enumID) 的 case 185: 处

□ C#

```
1 while ((index1 = s.IndexOf("<enUS>")) == -1)
2 {
3     s = reader.ReadLine();
4 }
5 index1 += 6;
6 index2 = s.IndexOf("</enUS>", index1);
7 temp = s.Substring(index1, index2 - index1);
8 sb.Clear();
9 sb.Append(temp);
10 sb.Replace("&lt;", "");
11 sb.Replace("&gt;", "");
12 sb.Replace("/&gt;", "");
13 sb.Replace("'", "");
14 sb.Replace(" ", "");
15 sb.Replace(":", "");
16 sb.Replace(".", "");
17 sb.Replace("!", "");
18 sb.Replace("?", "");
19 sb.Replace("-", "");
20 sb.Replace("_", "");
21 sb.Replace(",", "");
22 sb.Replace("(", "");
23 sb.Replace(")", "");
24 sb.Replace("/", "");
25 sb.Replace("\\", "");
```

```

26 sb.Replace(".", "");
27 c.name = this.cardNamestringToEnum(sb.ToString().ToLower());
28 if(c.name == CardDB.cardName.unknown)
29 {
30     // try chinese
31
32     while ((index1 = s.IndexOf("<zhCN>")) == -1)
33     {
34         s = reader.ReadLine();
35     }
36     index1 += 6;
37     index2 = s.IndexOf("</zhCN>", index1);
38     temp = s.Substring(index1, index2 - index1);
39     sb.Clear();
40     sb.Append(temp);
41     sb.Replace("&lt;", "");
42     sb.Replace("&bgt;", "");
43     sb.Replace("/&bgt;", "");
44     sb.Replace("'", "");
45     sb.Replace(" ", "");
46     sb.Replace(":", "");
47     sb.Replace(".", "");
48     sb.Replace("!", "");
49     sb.Replace("?", "");
50     sb.Replace("-", "");
51     sb.Replace("_", "");
52     sb.Replace(",", "");
53     sb.Replace("(", "");
54     sb.Replace(")", "");
55     sb.Replace("/", "");
56     sb.Replace(".", "");
57     sb.Replace("\\", "");
58     c.name = this.cardNamestringToEnum(sb.ToString().ToLower());
59 }

```

## | BUG2: 奥秘的playrequires未写入

最近发现论坛上有不少朋友在讨论关于奥秘法中如何裸出“男仆”和“云雾王子”的策略问题。问题核心在于 CardDB.cs 文件未能正确加载“男仆”和“云雾王子”触发所需的最少奥秘数（至少1个奥秘）的 playrequires 条件，导致系统误认为当前奥秘数量（默认大于0个）已满足条件，从而错误触发“云雾王子”的6点伤害效果。

解决这一问题的方法是，在 CardDB.cs 文件的 switch (reqID) 部分加入相应的代码行。

□ C#

```
1 case 59: c.needControlaSecret = param; continue;
```

## | Handmanager.cs(手牌管理)

C#

```

1 public class Handcard
2 {
3     public int position = 0; //手牌的位置
4     public int entity = -1; //炉石传说内部entity编号
5     public int manacost = 1000; //花费费用，但获取卡牌费用要用getManaCost(Playfield p
6     public int addattack = 0; //增加的攻击力，如风驰电掣的SIM中就使其 +1
7     public int addHp = 0; //增加的血量
8     public CardDB.Card card; //卡牌，指向CardDB.cs
9     public Minion target; //目标
10    public int elemPoweredUp = 0; //上回合是否使用元素牌
11    public int extraParam2 = 0; //扩展参数2，可以用来记录一些此卡需要的特殊数据
12    public bool extraParam3 = false; //扩展参数3
13    //读取卡牌法力值
14    public int getManaCost(Playfield p)
15    {
16        return this.card.getManaCost(p, this.manacost);
17    }
18    //判定卡牌是否能够使用
19    public bool canplayCard(Playfield p, bool own)
20    {
21        return this.card.canplayCard(p, this.manacost, own);
22    }
23 }

```

Handmanager.cs内容不多，所以介绍很简短。

## | Hrtprozis.cs(对局信息)

Hrtprozis是兄弟中非常重要的一个类，记录着从兄弟内部数据中获取到的各种信息

C#

```

1 public class Hrtprozis
2 {
3     public int pId = 0; //唯一id
4     public int attackFaceHp = 15; //打脸血量
5     public int ownHeroFatigue = 0; //我方疲劳
6     public int ownDeckSize = 30; //我方牌库数量
7     public int enemyDeckSize = 30; //敌方牌库数量
8     public int enemyHeroFatigue = 0; //敌方疲劳
9     public int gTurn = 0; //第几回合
10    public int gTurnStep = 0; //第几个操作
11
12    public int ownHeroEntity = -1; //我方英雄Entity
13    public int enemyHeroEntity = -1; //敌方英雄Entity
14    public DateTime roundstart = DateTime.Now; //回合开始时间
15    public int currentMana = 0; //当前法力值
16
17    public int heroHp = 30, enemyHp = 30; //我方英雄血量，敌方英雄血量

```



```
18 public int heroAtk = 0, enemyAtk = 0; //我方英雄攻击力, 敌方英雄攻击力
19 public int heroDefence = 0, enemyDefence = 0; //我方英雄护甲, 敌方英雄护甲
20 public bool ownheroisread = false; //我方英雄是否可以攻击
21 public int ownHeroNumAttacksThisTurn = 0; //我方英雄此回合攻击了几次
22 public bool ownHeroWindfury = false; //我方英雄是否风怒
23 public bool herofrozen = false; //我方英雄是否冻结
24 public bool enemyfrozen = false; //敌方英雄是否冻结
25
26 public List<CardDB.cardIDEnum> ownSecretList = new List<CardDB.cardIDEnum>();
27 public int enemySecretCount = 0; //敌方奥秘数量
28 public Dictionary<int, CardDB.cardIDEnum> DiscoverCards = new Dictionary<int,
29 public Dictionary<CardDB.cardIDEnum, int> turnDeck = new Dictionary<CardDB.cam
30 private Dictionary<int, CardDB.cardIDEnum> deckCardForCost = new Dictionary<in
31 public bool noDuplicates = false; //牌库无重复（宇宙）
32
33 private int numTauntCards = -1; //牌库有几张嘲讽卡
34 private int numDivineShieldCards = -1; //牌库有几张圣盾卡
35 private int numLifestealCards = -1; //牌库有几张吸血卡
36 private int numWindfuryCards = -1; //牌库有几张风怒卡
37
38 public bool setGameRule = false; //设置Rule
39
40 public HeroEnum heroname = HeroEnum.None, enemyHeroname = HeroEnum.None; //我方
41 public string heronameingame = "", enemyHeronameingame = ""; //我方英雄名称, 敌
42 public TAG_CLASS ownHeroStartClass = TAG_CLASS.INVALID; //我方英雄职业
43 public TAG_CLASS enemyHeroStartClass = TAG_CLASS.INVALID; //敌方英雄职业
44 public CardDB.Card heroAbility; //我方英雄技能
45 public bool ownAbilityisReady = false; //我方英雄技能是否准备完成
46 public int ownHeroPowerCost = 2; //我方英雄技能费用
47 public CardDB.Card enemyAbility; //敌方英雄技能
48 public int enemyHeroPowerCost = 2; //敌方英雄技能费用
49 public int numOptionsPlayedThisTurn = 0; //此回合中的操作数
50 public int numMinionsPlayedThisTurn = 0; //此回合中的随从数
51 public CardDB.cardIDEnum OwnLastDiedMinion = CardDB.cardIDEnum.None; //我方最后
52
53 public int cardsPlayedThisTurn = 0; //此回合中的出牌数, 可以用于判定连击
54 public int ueberladung = 0; //过载水晶
55 public int lockedMana = 0; //锁定水晶
56 public int ownMaxMana = 0; //我方最大法力值
57 public int enemyMaxMana = 0; //敌方最大法力值
58
59 public Minion ownHero = new Minion(); //我方英雄
60 public Minion enemyHero = new Minion(); //敌方英雄
61 public Weapon ownWeapon = new Weapon(); //我方武器
62 public Weapon enemyWeapon = new Weapon(); //敌方武器
63 public List<Minion> ownMinions = new List<Minion>(); //我方随从
64 public List<Minion> enemyMinions = new List<Minion>(); //敌方随从
65 public Dictionary<int, IDEnumOwner> LurkersDB = new Dictionary<int, IDEnumOwne
66
67 public int anzOgOwnCThunHpBonus = 0; //克苏恩血量
```



```

68     public int anzOgOwnCThunAngrBonus = 0; //克苏恩攻击力
69     public int anzOgOwnCThunTaunt = 0; //克苏恩嘲讽
70     public int anzOwnJadeGolem = 0; //我方魔像计数器
71     public int anzEnemyJadeGolem = 0; //敌方魔像计数器
72     public int OwnInvoke = 0; //我方祈求数
73     public int EnemyInvoke = 0; //敌方祈求数
74     public int ownCrystalCore = 0; //魔王的骑士：水晶核心，任务贼的任务奖励，在本局对战
75     public bool ownMinionsInDeckCost0 = false; //我方牌库中是否有0费随从
76     public int anzOwnElementalsThisTurn = 0; //在此回合中使用元素牌
77     public int anzOwnElementalsLastTurn = 0; //上一回合使用元素牌
78     public int ownElementalsHaveLifesteal = 0; //我方元素牌具有吸血
79     private int ownPlayerController = 0; //我方玩家控制？
80
81     public PenaltyManager penman; //惩罚管理
82     public Settings settings; //设置管理
83     Helpfunctions help; //调试输出
84     CardDB cdb; //卡牌数据
85 }

```

接下来是介绍这个类中一些中能用到的函数

□ C#

```

1  //英雄ID转到名称
2  public string heroIDtoName(string s)
3
4  //英雄名称转到枚举，留牌文件就是用这个函数进行转换的，由此可见圣骑士的名称用pala和paladin
5  public HeroEnum heroNametoEnum(string s)
6  {
7      switch (s)
8      {
9          case "all": return HeroEnum.all;
10         case "druid": return HeroEnum.druid;
11         case "hunter": return HeroEnum.hunter;
12         case "mage": return HeroEnum.mage;
13         case "pala": return HeroEnum.pala;
14         case "paladin": return HeroEnum.pala;
15         case "priest": return HeroEnum.priest;
16         case "shaman": return HeroEnum.shaman;
17         case "thief": return HeroEnum.thief;
18         case "rogue": return HeroEnum.thief;
19         case "maievshadowson": return HeroEnum.thief;
20         case "warlock": return HeroEnum.warlock;
21         case "warrior": return HeroEnum.warrior;
22         case "lordjaraxxus": return HeroEnum.lordjaraxxus;
23         case "ragnarosthefirelord": return HeroEnum.ragnarosthefirelord;
24         default: return HeroEnum.None;
25     }
26 }
27
28 //英雄枚举转到种类

```

```

29 public TAG_CLASS heroEnumtoTagClass(HeroEnum he)
30
31 //英雄种类转到枚举
32 public HeroEnum heroTAG_CLASSstringToEnum(string s)
33
34 //读取牌库中X费的卡牌
35 public CardDB.cardIDEnum getDeckCardsForCost(int cost)
36
37 //读取牌库中指定特性的卡牌(GAME_TAGS.TAUNT,GAME_TAGS.DIVINE_SHIELD,GAME_TAGS.LIFESTE
38 public int numDeckCardsByTag(GAME_TAGS tag)
39
40 //输出当前场面上的信息到日志
41 public void printHero()
42
43 //输出我方随从信息到日志
44 public void printOwnMinions()
45
46 //输出敌方随从信息到日志
47 public void printEnemyMinions()
48
49 //输出我方牌库信息(兄弟的牌库读取时好时坏)
50 public void printOwnDeck()
51 [/collapse]
52
53 [collapse title="Minion.cs(随从信息)"]
54 public class Minion
55 {
56     public int anzGotDmg = 0; //受到伤害次数
57     public int GotDmgValue = 0; //受到伤害总和
58     public int anzGotHealed = 0; //受到治疗次数
59     public int GotHealedValue = 0; //受到治疗总和
60     public bool gotInspire = false; //得到激励
61     public bool isHero = false; //是否为英雄
62     public bool own; //是否为己方
63     public int pID = 0; //PID
64     public CardDB.cardName name = CardDB.cardName.unknown; //随从名称
65     public TAG_CLASS cardClass = TAG_CLASS.INVALID; //随从归属职业
66     public int synergy = 0; //职业契合度,即各种族(机械鱼人恶魔野兽)与职业相关性
67     public Handmanager.Handcard handcard; //手牌信息,如果Minion中储存的内容不够的话,
68     public int entitiyID = -1; //实例ID
69     public int zonepos = 0; //随从放置位置
70     public CardDB.Card deathrattle2; //亡语2号,比如其他随从的亡语技能转移
71     public bool playedThisTurn = false; //在这回合使用
72     public bool playedPrevTurn = false; //在上回合使用
73     public int numAttacksThisTurn = 0; //这回合攻击了几次
74     public bool immunewhileAttacking = false; //攻击时免疫
75     public bool allreadyAttacked = false; //已经攻击过
76     public bool shadowmadness = false; //暗影狂乱,直到回合结束,获得一个攻击力小于:
77     public bool destroyOnOwnTurnStart = false; //我方回合开始被消灭
78     public bool destroyOnEnemyTurnStart = false; //敌方回合开始被消灭

```

```

79  public bool destroyOnOwnTurnEnd = false; //我方回合结束被消灭
80  public bool destroyOnEnemyTurnEnd = false; //敌方回合结束被消灭
81  public bool changeOwnerOnTurnStart = false; //回合开始时变更所有权，如暗影狂乱结界
82  public bool conceal = false; //隐藏（直到你的下个回合，使所有友方随从获得潜行）
83  public int ancestralspirit = 0; //先祖之魂，使一个随从获得“亡语：再次召唤该随从。
84  public int desperatestand = 0; //殊死一搏，使一个随从获得“亡语：回到战场，并具有1/
85  public int souloftheforest = 0; //丛林之魂，使你的所有随从获得“亡语：召唤一个2/2的
86  public int stegodon = 0; //剑龙
87  public int livingspores = 0; //活性孢子 亡语：召唤两个1/1的植物。
88  public int explorershut = 0; //探险帽 使一个随从获得 + 1/+1，亡语：将一个探险帽置
89  public int returnToHand = 0; //回到手牌
90  public int infest = 0; //寄生感染 使你的所有随从获得 “亡语：随机将一张野兽牌置入你
91
92  public int ownBlessingOfWisdom = 0; //我方智慧祝福
93  public int enemyBlessingOfWisdom = 0; //敌方智慧祝福
94  public int ownPowerWordGlory = 0; //我方真言术：耀
95  public int enemyPowerWordGlory = 0; //敌方真言术：耀
96  public int spellpower = 0; //法术强度
97
98  public bool cantBeTargetedBySpellsOrHeroPowers = false; //无法成为法术或英雄技能
99  public bool cantAttackHeroes = false; //无法攻击英雄
100 public bool cantAttack = false; //无法攻击
101
102 public int Hp = 0; //当前血量
103 public int maxHp = 0; //最大血量
104 public int armor = 0; //护甲值（英雄）
105
106 public int Angr = 0; //攻击力
107 public int AdjacentAngr = 0; //相邻buff攻击力加成
108 public int tempAttack = 0; //一回合攻击力加成
109 public int justBuffed = 0; //?
110
111 public bool Ready = false; //攻击准备就绪
112
113 public bool taunt = false; //嘲讽
114 public bool wounded = false; //受伤
115
116 public bool divineshield = false; //圣盾
117 public bool windfury = false; //风怒
118 public bool frozen = false; //冻结
119 public bool stealth = false; //隐身
120 public bool immune = false; //免疫
121 public bool untouchable = false; //无法被攻击
122 public bool exhausted = false; //无法攻击?
123 public bool lifesteal = false; //吸血
124 public bool modular = false; //磁力
125 public int charge = 0; //冲锋
126 public int rush = 0; //突袭
127 public int hChoice = 0; //?
128 public bool poisonous = false; //剧毒

```

```

129     public bool cantLowerHPbelowONE = false; //血量无法低于1
130
131     public bool silenced = false; //沉默
132     public bool playedFromHand = false; //从手牌中打出
133     public bool extraParam = false; //扩展参数1
134     public int extraParam2 = 0; //扩展参数2
135 }
136
137 下面还有几个常用的函数
138 //受到伤害
139 public void getDamageOrHeal(int dmg, Playfield p, bool isMinionAttack, bool dontCa
140 //随从死亡
141 public void minionDied(Playfield p)
142 //更新状态
143 public void updateReadyness()
144 //被沉默
145 public void becomeSilence(Playfield p)
146 //攻击之后的状态
147 public Minion GetTargetForMinionWithSurvival(Playfield p, bool own)
148 //特殊随从的特效添加位置
149 public void loadEnchantments(List<miniEnch> enchants, int ownPlayerControler)

```

## | MiniSimulator.cs(AI功能)

### | 核心函数 doallmoves

为了更好地理解，建议将本部分内容与**Ai.cs**文件的学习相结合。本节将重点介绍核心函数 `doallmoves`。为了简化学习过程，我已经省略了部分非关键代码。建议使用Visual Studio进行单步调试，以便更深入地理解代码逻辑。

□ C#

```

1 //val是value的简称
2 public float doallmoves(Playfield playf)
3 {
4     print = playf.print; //确定是否输出场面，记录在print中
5     this.isLethalCheck = playf.isLethalCheck; //是否为伤害检查
6     enoughCalculations = false; //计算是否足够，深度大于最大深度，计算场面数大于最大宽
7     botBase = Ai.Instance.botBase; //策略文件类
8     this.addToPosmoves(playf); //将当前场面加入到状态队列
9     bool havedonesomething = true; //是否无步骤可出，例如：法力值不够出任何牌，随从全部
10    List<Playfield> temp = new List<Playfield>(); //这一回合的状态队列
11    int deep = 0; //深度
12    bestoldval = -20000000; //最小的val，用于标记是否已经计算过场面val
13    while (havedonesomething)
14    {
15        temp.AddRange(this.posmoves);
16        havedonesomething = false;
17        //startEnemyTurnSimThread 函数非常重要，AI的核心内容，看下面的介绍

```

```

18     if (print) startEnemyTurnSimThread(temp, 0, temp.Count);
19     else
20     {
21         //似乎是启用多线程的形式计算
22         Parallel.ForEach(Partitioner.Create(0, temp.Count),
23             range =>
24             {
25                 startEnemyTurnSimThread(temp, range.Item1, range.Item2);
26             });
27     }
28
29     foreach (Playfield p in temp)
30     {
31         if (this.totalboards > 0) this.calculated += p.nextPlayfields.Count; ,
32         if (this.calculated <= this.totalboards) //如果宽度小于设定值则继续计算
33         {
34             this.posmoves.AddRange(p.nextPlayfields); //将操作后的PlayField进入
35         }
36
37         float pVal = botBase.getPlayfieldValue(p);
38         //计算场面的val, 如果是调试的话跟可以进去看看, 不在调试中跟进去的是模版文件
39         //里面开头的第一句话便是 if (p.value >= -2000000) return p.value;
40         //和上面的bestoldval = -20000000;对应来看就能理解
41         //意思就是防止多次运算, 这个场面的val如果大于-2000000(已经计算过val)则直接
42
43         if (pVal > bestoldval)
44         {
45             //如果新计算的val大于以前记录的最大val, 则更新最大的val, 并且记录场面信
46             bestoldval = pVal;
47             bestold = p;
48             bestoldDuplications.Clear();
49         }
50         else if (pVal == bestoldval) bestoldDuplications.Add(p); //如果val相等则
51     }
52
53     if (isLethalCheck && bestoldval >= 10000) this.posmoves.Clear();
54     //如果bestval大于等于10000则意味着兄弟计算出当前场面可以斩杀
55     //对应的策略中的代码是 if (p.enemyHero.Hp <= 0) retval = 10000;
56
57     if (this.posmoves.Count > 0) havedonesomething = true; //如果还有其他状态可
58
59     cuttingpossibilities(isLethalCheck); //去除重复的PlayField, 下面会讲解
60     deep++;
61     //下面两行就是判断计算的数量有没有大于设定的值, 如果大于则不进行运算了
62     if (this.calculated > this.totalboards) enoughCalculations = true;
63     if (deep >= this.maxdeep) enoughCalculations = true;
64 }
65
66 if (this.dirtyTwoTurnSim > 0 && !twoturnfields.Contains(bestold)) twoturnfield
67 if (!isLethalCheck && bestoldval < 10000) doDirtyTwoTurnsim();

```

```

68
69     if (posmoves.Count >= 1)
70     {
71         //把所有的场面根据val进行排序，将高分的排在前面
72         posmoves.Sort((a, b) => botBase.getPlayfieldValue(b).CompareTo(botBase.get
73         Playfield bestplay = posmoves[0];
74         float bestval = botBase.getPlayfieldValue(bestplay);
75         int pcount = posmoves.Count;
76         for (int i = 1; i < pcount; i++)
77         {
78             float val = botBase.getPlayfieldValue(posmoves[i]);
79             if (bestval > val) break;
80             if (posmoves[i].cardsPlayedThisTurn > bestplay.cardsPlayedThisTurn) co
81             else if (posmoves[i].cardsPlayedThisTurn == bestplay.cardsPlayedThisTurn
82             {
83                 if (bestplay.optionsPlayedThisTurn > posmoves[i].optionsPlayedThis
84                 else if (bestplay.optionsPlayedThisTurn == posmoves[i].optionsPlay
85             }
86             bestplay = posmoves[i];
87             bestval = val;
88         }
89         this.bestmove = bestplay.getNextAction();
90         this.bestmoveValue = bestval;
91         this.bestboard = new Playfield(bestplay);
92         this.bestboard.guessingHeroHP = bestplay.guessingHeroHP;
93         this.bestboard.value = bestplay.value;
94         this.bestboard.hashCode = bestplay.hashCode;
95         bestoldDuplicates.Clear();
96         //正常退出从这里返回
97         return bestval;
98     }
99     this.bestmove = null;
100    this.bestmoveValue = -100000;
101    this.bestboard = playf;
102    //这里是异常退出返回的值
103    return -10000;
104 }
105 //对于这个函数名称我不是特别的理解，似乎应该是ownTurnSimThread
106 private void startEnemyTurnSimThread(List<Playfield> source, int startIndex, int e
107 {
108     int berserk = Settings.Instance.berserkIfCanFinishNextTour;
109     int printRules = Settings.Instance.printRules;
110     for (int i = startIndex; i < endIndex; i++)
111     {
112         Playfield p = source[i];
113         if (p.complete || p.ownHero.Hp <= 0) { }
114         else if (!enoughCalculations)
115         {
116             //从这里进入调用getMoveList函数，这个函数返回的是兄弟下一步所有可以做的操作
117             //如果调试到这里，输出actions，发现没有你想要的Action就可以跟进去看看

```



```

118 //一般来说，法力值不够出的牌会直接省略，如果正常操作能够出的牌没出，
119 //则说明，惩罚值大于500，操作直接被省略，这是我调试的经验
120 //建议直接跟进去看寻找原因，可以加深理解
121 List<Action> actions = movegen.getMoveList(p, usePenaltyManager, useC
122
123 if (printRules > 0) p.endTurnState = new Playfield(p);
124 //读取到actions后接下来对每个步骤进行模拟
125 //从而得到操作之后的场面并且计算val值
126 foreach (Action a in actions)
127 {
128     Playfield pf = new Playfield(p);
129     pf.doAction(a);
130     pf.evaluatePenalty += - pf.ruleWeight + RulesEngine.Instance.getf
131     if (pf.ownHero.Hp > 0 && pf.evaluatePenalty < 500) p.nextPlayfie
132 }
133 }
134
135 if (this.isLethalCheck)
136 {
137     //可以斩杀
138     if (berserk > 0)
139     {
140         //结束回合，接下来模拟敌方下一回合的操作
141         p.endTurn();
142         if (p.enemyHero.Hp > 0)
143         {
144             bool needETS = true;
145             //如果对面没有嘲讽且我方随从全部可以攻击则进行不模拟
146             if (p.anzEnemyTaunt < 1) foreach (Minion m in p.ownMinions) {
147                 else
148                 {
149                     if (p.anzOwnTaunt < 1) foreach (Minion m in p.ownMinions)
150                 }
151                 //从这里进入模拟敌方下一回合的操作
152                 if (needETS) Ai.Instance.enemyTurnSim[threadnumber].simulateEr
153             }
154         }
155
156         p.complete = true;
157     }
158 }
159 else
160 {
161     p.endTurn();
162     //这里和上面的区别就在于不进行判定是否需要模拟敌方的下个回合和一些判定
163     if (p.enemyHero.Hp > 0)
164     {
165         Ai.Instance.enemyTurnSim[threadnumber].simulateEnemyTurn(p, this.
166         //如果val <= -10000 再进入判断
167         if (p.value <= -10000)

```

```

168         {
169             bool secondChance = false;
170             foreach (Action a in p.playactions)
171             {
172                 if (a.actionType == actionEnum.playcard)
173                 {
174                     //这里判定一下是否出了战吼相关的牌，如果有的话 val += 1500
175                     if (pen.cardDrawBattleCryDatabase.ContainsKey(a.card.c
176                 }
177             }
178             if (secondChance) p.value += 1500;
179         }
180     }
181     p.complete = true;
182 }
183 //计算一下p的val值
184 botBase.getPlayfieldValue(p);
185 }
186 }

```

## 调试经验

- 未执行可行的斩杀操作
  1. 检查敌方是否设置了奥秘，并确认是否启用了防奥秘机制。若已启用，可能由于担心触发如“寒冰屏障”等效果，故未进行攻击以破解奥秘。
  2. 若敌方无奥秘，考虑在 `PenalityManager.cs` 文件的相关数据库中添加适当的数据项。
- 未采取更佳的操作方案，直接结束回合
  1. 在面对极其不利的局面时，AI可能会选择直接跳过回合，认定胜利无望。
  2. 当场上随从众多或手牌数量大时，AI在计算过程中耗时较长，最终可能由于计算量超出设定的 `MaxWide` 和 `MaxDeep` 限制而终止计算。建议在 `_setting.txt`（位于 `behavior/_setting.txt`）文件中调整相关参数值。
  3. 通过跟踪 `List<Action> actions = movegen.getMoveList(p, usePenalityManager, useCuttingTargets, true);` 这一行代码进行调试。
- 攻击己方随从或为敌方随从提供增益
  1. 这种行为的出现通常是因为AI计算得出的评分高于原有状态。例如，若设置奥秘的评分过高，AI可能会使用技能攻击如“疯狂的科学家”以产生奥秘。出现这种策略的原因可能是策略编写时对惩罚机制的应用不当。建议通过调试来识别和解决这个问题，并与大家分享解决方案，共同学习进步。

## PenalityManager.cs(惩罚管理)

待更新



## | Playfield.cs(场面信息)

待更新

## | Questmanager.cs(任务管理)

兄弟目前的任务管理不支持支线任务，在我更新之后我会教大家怎么修改兄弟来支持支线任务

## | behavior目录

### | \_mulligan.txt (留牌配置)

此内容是对贴吧用户：**wayne0036** 所发的帖子内容进行的整理。

**\_mulligan.txt**是放在策略目录下的文件，如果文件夹下不存在此文件则需要自己创建一个。

如文件位置：`Routines\DefaultRoutine\Silverfish\behavior\策略名称\_mulligan.txt`

文件格式：`卡牌ID;己方职业;对方职业;留牌策略;/条件`

卡牌ID：

卡牌ID	含义
GAME_005	幸运币
CFM_066	暗金教侍从
ULD_239	火焰结界

以上只是随便找了几张卡牌举例，关于其他的卡牌信息可以在[FBIGAME炉石数据库](#)查询。

职业信息：

职业简称	含义
druid	德鲁伊
hunter	猎人
mage	法师
pala	圣骑士
priest	牧师
thief	盗贼
shaman	萨满
warlock	术士
warrior	战士

职业简称	含义
demonhunter	恶魔猎手
None	任意职业

留牌策略:

留牌策略	含义
Hold:1	留一张
Hold:2	留两张
Discard:2	一张不留

条件：仅当你的手牌中包含特定的卡牌（请填写具体的ID）时，才会执行留牌操作（其中，**GAME\_005** 代表游戏中的幸运币，该条件可用于判断玩家是否是后手）。

注意：

1. 留牌配置文件必须采用UTF-8编码格式，以避免读取错误。如果遇到读取问题，检查文件的编码设置可能会有所帮助。

2. 费用低于三的卡牌将默认被保留，而费用高于三的卡牌则会默认被舍弃。这意味着，只有通过编写留牌策略，费用高于三的卡牌才能被保留；同理，费用低于三的卡牌若要舍弃，也需编写相应的留牌策略。

以上信息旨在提供一些基础知识，帮助大家更好地理解留牌文件的相关概念。如果你对以上内容仍有疑问，我推荐使用[紫火大佬](#)开发的\*\*[HearthBuddy留牌编辑器](#)\*\*，该工具可以方便地帮助你编写和调整留牌策略。

下面贴出奥秘法的留牌策略方便学习理解

Code

```
1 //奥秘法
2 //暗金教侍从
3 CFM_066;mage;None;Hold:1;/
4 CFM_066;mage;None;Hold:2;/GAME_005/LOOT_101/EX1_287/UNG_020/FP1_018/ULD_239
5 //肯瑞托法师
6 EX1_612;mage;None;Hold:1;/LOOT_101/EX1_287/UNG_020/FP1_018/ULD_239
7 //法术法制
8 EX1_287;mage;None;Hold:1;/CFM_066/EX1_612
9 EX1_287;mage;None;Discard:2;/
10 //寒冰护体
11 EX1_289;mage;None;Hold:1;/CFM_066/EX1_612
12 EX1_289;mage;None;Discard:2;/
13 //爆炸符文
14 LOOT_101;mage;None;Hold:1;/CFM_066/EX1_612
15 LOOT_101;mage;None;Discard:2;/
```

```
16 //寒冰屏障
17 EX1_295;mage;None;Discard:2;/
18 //火焰结界
19 ULD_239;mage;hunter;Hold:1;/CFM_066/EX1_612
20 ULD_239;mage;shaman;Hold:1;/CFM_066/EX1_612
21 ULD_239;mage;pala;Hold:1;/CFM_066/EX1_612
22 ULD_239;mage;warrior;Discard:2;/
23 ULD_239;mage;mage;Discard:2;/
24 ULD_239;mage;priest;Discard:2;/
25 ULD_239;mage;thief;Hold:1;/CFM_066/EX1_612
26 ULD_239;mage;warlock;Hold:1;/CFM_066/EX1_612
27 ULD_239;mage;druid;Discard:2;/
28 //复制
29 FP1_018;mage;None;Hold:1;/EX1_612
30 FP1_018;mage;None;Discard:2;/
31 //远古谜团
32 ULD_726;mage;None;Discard:2;/CFM_066
33 //麦迪文的男仆
34 KAR_092;mage;None;Hold:1;/FP1_004/
35 //老旧的火把
36 LOE_002;mage;None;Discard:2;/
37 //对空奥术法师
38 ULD_240;mage;None;Discard:2;/
39 ULD_240;mage;warlock;Hold:1;/ULD_726
40 ULD_240;mage;thief;Hold:1;/ULD_726
41 ULD_240;mage;pala;Hold:1;/ULD_726
42 ULD_240;mage;shaman;Hold:1;/ULD_726
43 ULD_240;mage;hunter;Hold:1;/ULD_726
44 //奥术增幅体
45 YOD_008;mage;None;Discard:2;/
```

这一块到这里就结束啦，如有疑问请评论。

## | **\_settings.txt (AI配置)**

**\_settings.txt**是放在策略目录下的文件，如果文件夹下不存在此文件则需要自己创建一个。文件位置：`Routines\DefaultRoutine\Silverfish\behavior\策略名称\_settings.txt`

先看一下文件内容

### □ Code

```
1 // If you want to set your own settings, rename this file to _settings.txt and place it in the
2 // folder: Routines\DefaultRoutine\Silverfish\behavior\策略名称\_settings.txt
3 enfacehp = 15; // hp of enemy when your hero is allowed to attack the enemy face
4 // weaponOnlyAttackMobsUntilEnfacehp - If your opponent has more HP than enfacehp,
5 // 0 - don't attack face until enfacehp (except weapons with 1 Attack)
6 // 1 - don't attack face until enfacehp if weapon's durability = 1 (if durability
7 // 2 - don't attack face until enfacehp (any weapon)
8 // 3 - don't attack face until enfacehp if weapon's durability = 1 (if durability
9 // 4 - don't attack face until enfacehp (except: you have any* weapon generating cards)
```

```
10 // 5 - don't attack face until enfacehp (except: you have any* weapon generating c
11 weaponOnlyAttackMobsUntilEnfacehp = 0;
12
13 maxwide = 3000; // number of boards which are taken to the next deep-lvl
14 playaround = false; // play around some enemys aoe-spells
15 // these two parameters are value between 0 and 100 (0 <= Your_Value <= 100)
16 playaroundprob = 50; // probability where the enemy NOT plays the aoe-spell: 10
17 playaroundprob2 = 80; // probability where the enemy plays the aoe-spell, and you
18
19 twotsamount = 0; // number of boards where the second AI step is simulated
20 enemyTurnMaxWide = 40; // max number of enemy boards calculated in enemys-first-tu
21 enemyTurnMaxWideSecondStep = 200; // max number of enemy boards calculated in enem
22
23 nextTurnDeep = 10; //maximum actions in your second turn
24 nextTurnMaxWide = 20; //maximum best boards for calculation at each step in the se
25 nextTurnTotalBoards = 200; //maximum boards calculated in second turn simulation
26 berserkIfCanFinishNextTour = 1; // 0 - off(default), 1 - if there is any chance to
27
28 alpha = 50; // weight of the second turn in calculation (0<= alpha <= 100)
29 useSecretsPlayAround = true; // playing arround enemys secrets
30 placement = 0; // 0 - minions are interleaved by value (..low value - hi value..)
31
32 ImprovedCalculations = 1; // 0 - disabled(for old PCs), 1 - enabled
33 adjustActions = 0; // test!! - reorder actions after calculations: 0 - as calculat
34 printRules = 1; //0 - off, 1 - on
```

接下来对每个配置进行说明

参数名	含义
enfacehp	如果对面英雄的血量低于设定的值，就优先打脸。比如这里设定的值是15
weaponOnlyAttackMobsUntilEnfacehp	这里有六个设定值，前置条件都是对面血量大于enfacehp 0： 除了一攻武器，其他的不打脸 1： 一攻武器和武器耐久大于1时打脸 2： 任何武器都不打脸 3： 武器耐久度大于1时打脸 4： 有其他武器在手的时候打脸（升级 算作一攻武器） 5： 有其他一攻以上的武器在手或者手上的和装备着的都是一攻武器的时候攻击（升级算作一攻武器）
maxwide	最大的搜索宽度，详情见Ai.cs解析
playaround	设置值为true之后会防御AOE技能

参数名	含义
playaroundprob	设置范围 (0 - 100) 表示对手有多少几率会不使用AOE法术 0: 总是使用 100: 一定不会使用
playaroundprob2	设置范围 (0 - 100) 表示对手有多少几率使用AOE法术后你的随从会存活 0: 不会存活 , 100: 总是会存活
twotsamount	模拟多少回合之后的操作 比如设置为1, 就会模拟你一回合之后的情况
enemyTurnMaxWide	兄弟不仅会模拟自己的操作, 也会模拟对面的操作, 类似于maxwide 要求大于 enemyTurnMaxWideSecondStep
enemyTurnMaxWideSecondStep	在敌人回合的第二次操作中, 最多计算多少步 要求小于enemyTurnMaxWide
nextTurnDeep	下一回合的深度, 详情见Ai.cs解析
nextTurnMaxWide	下一回合的最大宽度, 详情见Ai.cs解析
nextTurnTotalBoards	下一回合最多的场面记录数
berserklfCanFinishNextTour	填入1时, 如果当前场面全部打脸, 下一回合就可以斩杀, 这一回合就会全部打脸 填入0则不开启
alpha	设置范围 (0 - 100) 在Ai.cs中提到过的分配权重, 分配给当前回合模拟和下一回合模拟的权重 比如这里填0, 则就是下一回合的信息不计入计算
useSecretsPlayAround	开启之后可以防奥秘, 无脑抢血的卡组建议关掉
placement	随从放置位置设定 为0时随从放置会根据其价值交错开来 (低价值 - 高价值 - 低价值) 为1时高价值的在两侧, 低价值的在中间

参数名	含义
ImprovedCalculations	提高计算能力 老电脑填0，新电脑填1
adjustActions	对操作重排 填1时优先使用AOE攻击 重排函数在： ActionNormalizer.cs中
printRules	填1：输出规则 填0：不输出

| **\_combo.txt (连招配置)**

**\_combo.txt**是放在策略目录下的文件，如果文件夹下不存在此文件则需要自己创建一个。如文件位置：`Routines\\DefaultRoutine\\Silverfish\\behavior\\策略名称\\_combo.txt`

文件格式： 卡牌1ID,卡牌1惩罚;卡牌2ID,卡牌2惩罚;[附加参数]

附加参数：

参数名	含义
mana	Combo所需的法力值，兄弟会自动修正
bonus	优先级，越高优先级越大
nxttrn	是否为两回合Combo
bonusfirst	第一张卡的优先级
bonussecond	第二张卡的优先级

参数的添加方法一般是：**参数名:参数值** 且参数之间用 **冒号** 隔开，**==**只有 `nxttrn` 不需要冒号和参数值 **==**

职业信息：

职业简称	含义
druid	德鲁伊
hunter	猎人
mage	法师
pala	圣骑士
priest	牧师
thief	盗贼
shaman	萨满

职业简称	含义
warlock	术士
warrior	战士
demonhunter	恶魔猎手
不填hero参数	任意职业

给出一些例子便于理解：

□ Code

```
1 //先使用水晶学再使用风驰电掣，构成一套Combo
2 BOT_909,0;CFM_305,0;mana:2;bonus:100;hero:pala;
3 //本回合使用暴走旋风，下回合使用奥术暴龙，构成一套Combo
4 DAL_742,0;TRL_311,0;mana:14;nxttrn;bonus:100;
```

这一块到这里就结束啦，如果有疑问请评论。

## | `_rules.txt` (规则引擎)

规则引擎的编写相当复杂，先等我琢磨透了再来写吧~

## | Behavior控场模式.cs

对自带策略的解说

## | Behavior怼脸模式.cs

对自带策略的解说

## | card目录

**SIM事件和常用函数整理** 首先要学会在头部引用三个库文件，这样可以方便后期的整理与修改

□ C#

```
1 using System;
2 using System.Collections.Generic;
3 using System.Text;
```

## | 奥秘触发事件(onSecretPlay)

onSecretPlay有三张不同参数的写法

□ C#

```
1 public virtual void onSecretPlay(Playfield p, bool ownplay, Minion attacker, Minio
2 public virtual void onSecretPlay(Playfield p, bool ownplay, Minion target, int num
```

```
3 public virtual void onSecretPlay(Playfield p, bool ownplay, int number)
```

我分别给出例子

## 自动防御矩阵

□ C#

```
1 //<b>奥秘: </b>当你的随从受到攻击时, 使其获得<b>圣盾</b>。
2 public override void onSecretPlay(Playfield p, bool ownplay, Minion attacker, Minion target)
3 {
4     number = 0;
5     if (ownplay)
6     {
7         if (p.ownMinions.Count >= 1)
8         {
9             if (p.ownMinions[p.ownMinions.Count - 1].name == CardDB.cardName.defensive
10             {
11                 target.divineshield = true;
12             }
13         }
14     }
15     else
16     {
17         if (p.enemyMinions.Count >= 1)
18         {
19             if (p.enemyMinions[p.enemyMinions.Count - 1].name == CardDB.cardName.defensive
20             {
21                 target.divineshield = true;
22             }
23         }
24     }
25 }
```

## 爆炸符文

□ C#

```
1 //<b>奥秘: </b>在你的对手使用一张随从牌后, 对该随从造成$6点伤害, 超过其生命值上限的伤害将
2 public override void onSecretPlay(Playfield p, bool ownplay, Minion target, int number)
3 {
4     int dmg = (ownplay) ? p.getSpellDamageDamage(6) : p.getEnemySpellDamageDamage(6);
5     if (target != null)
6     {
7         if (target.Hp < dmg)
8             p.minionGetDamageOrHeal((ownplay) ? p.enemyHero : p.ownHero, dmg - target.Hp);
9         p.minionGetDamageOrHeal(target, dmg);
10    }
11 }
```

## 火焰结界



 C#

```
1 //<b>奥秘: </b>在一个随从攻击你的英雄后, 对所有敌方随从造成$3点伤害。
2 public override void onSecretPlay(Playfield p, bool ownplay, int number)
3 {
4     int dmg = (ownplay) ? p.getSpellDamageDamage(3) : p.getEnemySpellDamageDamage(
5     p.allMinionOfASideGetDamage(!ownplay, dmg);
6 }
```

## | 使用法术牌事件(onCardPlay)

### 奥术智慧

 C#

```
1 //抽两张牌。
2 public override void onCardPlay(Playfield p, bool ownplay, Minion target, int choi
3 {
4     p.drawACard(CardDB.cardIDEnum.None, ownplay);
5     p.drawACard(CardDB.cardIDEnum.None, ownplay);
6 }
```

## | 弃牌事件(onCardDicscard)

### 玛克扎尔的小鬼

 C#

```
1 //每当你弃掉一张牌时, 抽一张牌。
2 public override bool onCardDicscard(Playfield p, Handmanager.Handcard hc, Minion o
3 {
4     if (own == null) return false;
5     if (checkBonus) return false;
6     p.drawACard(CardDB.cardIDEnum.None, own.own);
7     return false;
8 }
```

### 镀银魔像

 C#

```
1 //如果你弃掉了这张随从牌, 则会召唤它。
2 public override bool onCardDicscard(Playfield p, Handmanager.Handcard hc, Minion o
3 {
4     if (checkBonus) return true;
5     if (own != null) return false;
6
7     bool ownplay = true;
8     List<Minion> temp = (ownplay) ? p.ownMinions : p.enemyMinions;
9     p.callKid(hc.card, temp.Count, ownplay, false);
10    Minion m = temp[temp.Count - 1];
```

```

11     if (m.name == hc.card.name && m.playedThisTurn)
12     {
13         m.entitiyID = hc.entity;
14         m.Angr += hc.addattack;
15         m.Hp += hc.addHp;
16     }
17     return true;
18 }

```

稍微研究了一下弃牌函数。一般来说这个函数返回true的时候降低弃牌的惩罚。如果函数返回了false就不会降低，所以增益效果强的卡牌应当返回true。checkBonus是用来检测卡牌是否有弃牌效果的，也就是检测并不模拟SIM效果。own指的是弃掉随从卡的时候使用，或者场上随从相应的弃牌效果

## | 战吼事件(getBattlecryEffect)

注意：炉石兄弟存在一个问题，武器牌的战吼并不能写在这个函数中，而是要写在法术牌的onCardPlay事件中，这可能是一个bug。所以，用到这个事件的只能是随从的SIM

### 雏龙巨婴

□ C#

```

1  //<b>战吼：</b>抽一张牌。
2  public override void getBattlecryEffect(Playfield p, Minion own, Minion target, in
3  {
4      p.drawACard(CardDB.cardName.unknown, own.own);
5  }

```

## | 光环特效(onAuraStarts & onAuraEnds)

onAuraStarts和onAuraEnds一般都是成对出现的 **战歌指挥官**

□ C#

```

1  //你的具有冲锋的随从获得+1攻击力。
2  public override void onAuraStarts(Playfield p, Minion own)
3  {
4      if (own.own)
5      {
6          foreach (Minion m in p.ownMinions)
7          {
8              if (m.charge > 0) p.minionGetBuffed(m, 1, 0);
9          }
10     }
11     else
12     {
13         foreach (Minion m in p.enemyMinions)
14         {
15             if (m.charge > 0) p.minionGetBuffed(m, 1, 0);

```

```

16     }
17 }
18
19 }
20
21 public override void onAuraEnds(Playfield p, Minion own)
22 {
23     if (own.own)
24     {
25         foreach (Minion m in p.ownMinions)
26         {
27             if (m.charge > 0) p.minionGetBuffed(m, -1, 0);
28         }
29     }
30     else
31     {
32         foreach (Minion m in p.enemyMinions)
33         {
34             if (m.charge > 0) p.minionGetBuffed(m, -1, 0);
35         }
36     }
37 }

```

## | 激怒特效(onEnrageStart & onEnrageStop)

**onEnrageStart**和**onEnrageStop**一般都是成对出现的 **牛头人战士**

□ C#

```

1 //受伤时具有+3攻击力。
2 public override void onEnrageStart(Playfield p, Minion m)
3 {
4     m.Angr += 3;
5 }
6
7 public override void onEnrageStop(Playfield p, Minion m)
8 {
9     m.Angr -= 3;
10 }

```

## | 治疗事件(onAMinionGotHealedTrigger & onAHeroGotHealedTrigger & onACharGotHealed)

**北郡牧师**

□ C#

```

1 //每当一个随从获得治疗时，抽一张牌。
2 public override void onAMinionGotHealedTrigger(Playfield p, Minion triggerEffectMi
3 {
4     for (int i = 0; i < minionsGotHealed; i++)

```

```

5      {
6          p.drawACard(CardDB.cardIDEnum.None, triggerEffectMinion.own);
7      }
8  }

```

## 黑色卫士

□ C#

```

1  //每当你的英雄获得治疗时，便随机对一个敌方随从造成等量的伤害。
2  public override void onAHeroGotHealedTrigger(Playfield p, Minion triggerEffectMinion)
3  {
4      int dmg = ownerOfHeroGotHealed;
5      Minion target = null;
6      if (triggerEffectMinion.own) target = p.getEnemyCharTargetForRandomSingleDamage;
7      else target = p.searchRandomMinion(p.ownMinions, searchmode.searchHighestAttack);
8      if (target != null) p.minionGetDamageOrHeal(target, dmg);
9  }

```

## 圣光护卫者

□ C#

```

1  //每当一个角色获得治疗，便获得+2攻击力。
2  public override void onACharGotHealed(Playfield p, Minion triggerEffectMinion, int amount)
3  {
4      p.minionGetBuffed(triggerEffectMinion, 2 * charsGotHealed, 0);
5  }

```

## | 回合事件(onTurnStartTrigger & onTurnEndsTrigger)

### 常用

□ C#

```

1  if (turnStartOfOwner == triggerEffectMinion.own)

```

来判定是否在随从拥有者的回合开始或结束

### 末日预言者

□ C#

```

1  //在你的回合开始时，消灭所有随从。
2  public override void onTurnStartTrigger(Playfield p, Minion triggerEffectMinion, bool isYourTurn)
3  {
4      if (turnStartOfOwner == triggerEffectMinion.own)
5      {
6          foreach (Minion m in p.ownMinions)
7          {
8              if (m.entitiyID == triggerEffectMinion.entitiyID) continue;
9              if (m.playedThisTurn || m.playedPrevTurn)
10             {

```

```

11         if (PenaltyManager.Instance.ownSummonFromDeathrattle.ContainsKey(p))
12             p.evaluatePenalty += (m.Hp + m.Angr) * 6;
13     }
14 }
15 p.allMinionsGetDestroyed();
16 }
17 }

```

## 迦顿男爵

□ C#

```

1 //在你的回合结束时，对所有其他角色造成2点伤害。
2 public override void onTurnEndsTrigger(Playfield p, Minion triggerEffectMinion, bool isYourTurn)
3 {
4     if (turnEndOfOwner == triggerEffectMinion.own)
5     {
6         p.allCharsGetDamage(2, triggerEffectMinion.entitiyID);
7     }
8 }

```

## | 伤害事件(onMinionGotDmgTrigger)

### 苦痛侍僧

□ C#

```

1 //每当该随从受到伤害，抽一张牌。
2 public override void onMinionGotDmgTrigger(Playfield p, Minion m, int anzOwnMinion)
3 {
4     if (m.anzGotDmg > 0)
5     {
6         int tmp = m.anzGotDmg;
7         m.anzGotDmg = 0;
8         for (int i = 0; i < tmp; i++)
9         {
10             p.drawACard(CardDB.cardIDEnum.None, m.own);
11         }
12     }
13 }

```

## | 死亡事件(onMinionDiedTrigger)

注意：死亡事件不等同于亡语事件，亡语事件请使用**onDeathrattle**

这里就要搬出一个教科书式的官方SIM 每召唤一个鱼人就获得+1攻击力，每有一个鱼人死亡就-1攻击力 **老瞎眼**

□ C#

```
1 //冲锋，在战场上每有一个其他鱼人便获得+1攻击力。
2 public override void getBattlecryEffect(Playfield p, Minion own, Minion target, in
3 {
4     foreach (Minion m in p.ownMinions)
5     {
6         if (m.handcard.card.race == 14)
7         {
8             if (m.entitiyID != own.entitiyID) p.minionGetBuffed(own, 1, 0);
9         }
10    }
11
12    foreach (Minion m in p.enemyMinions)
13    {
14        if (m.handcard.card.race == 14)
15        {
16            if (m.entitiyID != own.entitiyID) p.minionGetBuffed(own, 1, 0);
17        }
18    }
19 }
20
21 public override void onMinionIsSummoned(Playfield p, Minion triggerEffectMinion, M
22 {
23     if (summonedMinion.handcard.card.race == 14)
24     {
25         p.minionGetBuffed(triggerEffectMinion, 1, 0);
26     }
27 }
28
29 public override void onMinionDiedTrigger(Playfield p, Minion m, Minion diedMinion)
30 {
31     int diedMinions = p.tempTrigger.ownMurlocDied + p.tempTrigger.enemyMurlocDied;
32     if (diedMinions == 0) return;
33     int residual = (p.pID == m.pID) ? diedMinions - m.extraParam2 : diedMinions;
34     m.pID = p.pID;
35     m.extraParam2 = diedMinions;
36     if (residual >= 1)
37     {
38         p.minionGetBuffed(m, -1 * residual, 0);
39     }
40 }
```

## | 召唤事件(onMinionIsSummoned & onMinionWasSummoned)

**onMinionIsSummoned**和**onMinionWasSummoned**的区别在于前者是随从准备被召唤了，但是还没有存在于场面上。后者是随从已经被召唤到场面上了。下面**大胖**的这个例子就只能用**onMinionWasSummoned**，原因是它需要对所召唤的随从进行buff，而用**onMinionIsSummoned**并不能在**p.ownMinions**中读取到需要增幅的随从 **公正之剑**

```

1 //在你召唤一个随从后，使其获得+1/+1，这把武器失去1点耐久度。
2 public override void onCardPlay(Playfield p, bool ownplay, Minion target, int choi
3 {
4     p.equipWeapon(card,ownplay);
5 }
6
7 public override void onMinionIsSummoned(Playfield p, Minion triggerEffectMinion, M
8 {
9     if (triggerEffectMinion.own == summonedMinion.own )
10    {
11        p.minionGetBuffed(summonedMinion, 1, 1);
12        p.lowerWeaponDurability(1, triggerEffectMinion.own);
13    }
14 }

```

## 大胖

□ C#

```

1 //每当你使用一张攻击力为1的随从牌，便使该牌所召唤的随从获得+2/+2。
2 public override void onMinionWasSummoned(Playfield p, Minion m, Minion summonedMin
3 {
4     if (summonedMinion.playedFromHand && summonedMinion.Angr == 1 && m.own == summ
5     {
6         p.minionGetBuffed(summonedMinion, 2, 2);
7     }
8 }

```

## | 亡语事件(onDeathrattle)

### 麻风侏儒

□ C#

```

1 //亡语：对敌方英雄造成2点伤害。
2 public override void onDeathrattle(Playfield p, Minion m)
3 {
4     p.minionGetDamageOrHeal(m.own ? p.enemyHero : p.ownHero, 2);
5 }

```

## | 出牌事件(onCardIsGoingToBePlayed & onCardWasPlayed)

**onCardIsGoingToBePlayed**和**onCardWasPlayed**的区别在于前者是牌准备打出的时候的事件，而后者是牌已经被打出的时候的事件。**onCardIsGoingToBePlayed**分为两个不同参数的版本

□ C#

```

1 public virtual void onCardIsGoingToBePlayed(Playfield p, Handmanager.Handcard hc,
2 public virtual void onCardIsGoingToBePlayed(Playfield p, Handmanager.Handcard hc,

```

两者的区别比较大，简单的来说，前者写的一般是你场上随从发生的事件，而后者一般写的是你手牌里发生的事件，话不多说，来看例子吧

## 任务达人

□ C#

```
1 //每当你使用一张牌时，便获得+1/+1。
2 public override void onCardIsGoingToBePlayed(Playfield p, Handmanager.Handcard hc,
3 {
4     if (triggerEffectMinion.own == wasOwnCard)
5     {
6         p.minionGetBuffed(triggerEffectMinion, 1, 1);
7     }
8 }
```

## 黑金大亨

□ C#

```
1 //每当你召唤一个具有战吼的随从时，便使这张牌（在你手牌中时）获得+1/+1。
2 public override void onCardIsGoingToBePlayed(Playfield p, Handmanager.Handcard hc,
3 {
4     if (hc.card.battlecry && hc.card.type == CardDB.cardtype.MOB)
5     {
6         hc.addattack++;
7         hc.addHp++;
8     }
9 }
```

**onCardWasPlayed**目前还没有合适的例子，这两者的区别类似于召唤事件的两个函数

## | 激励事件(onInspire)

### 达拉然铁骑士

□ C#

```
1 //激励：获得法术伤害+1。
2 public override void onInspire(Playfield p, Minion m, bool own)
3 {
4     if (m.own == own)
5     {
6         m.spellpower++;
7         if (m.own) p.spellpower++;
8         else p.enemyspellpower++;
9     }
10 }
11
12 public override void onAuraEnds(Playfield p, Minion m)
13 {
14     if (m.own) p.spellpower -= m.spellpower;
```



```

15     else p.enemyspellpower -= m.spellpower;
16 }

```

## | 失去圣盾事件(onMinionLosesDivineShield)

### 浴火者伯瓦尔

□ C#

```

1 //圣盾在一个友方随从失去圣盾后，获得+2攻击力。
2 public override void onMinionLosesDivineShield(Playfield p, Minion m, int num)
3 {
4     p.minionGetBuffed(m, 2 * num, 0);
5 }

```

## | 发现牌的权值(getDiscoverScore)

注意：这个SIM函数是我后期加上去的，在原版兄弟中并没有这个函数，添加的方法请见贴吧，后期我也会在博客里发出来

**public virtual int getDiscoverScore(Playfield p)** 返回增加的权值，权值越大越容易被选择。**红龙女王阿莱克丝塔萨**

□ C#

```

1 //红龙女王是一张非常超模的卡，9费8-8还送两张0费龙卡，基本上宇宙体系看到就必选，但不是宇宙1
2 //于是我们可以在这个函数中对牌库中是否为宇宙体系进行判断，从而来驱使兄弟是否选择这张牌。
3 public override int getDiscoverScore(Playfield p)
4 {
5     if (p.prozis.noDuplicates) return 500;
6     else return 0;
7 }

```

以下所有函数都是在Playfield.cs中，在SIM中使用时请用**p.xxxx**

计算我方/敌方英雄英雄技能伤害

□ C#

```

1 public int getHeroPowerDamage(int dmg)
2 public int getEnemyHeroPowerDamage(int dmg)

```

计算我方/敌方法术伤害

□ C#

```

1 public int getSpellDamageDamage(int dmg)
2 public int getEnemySpellDamageDamage(int dmg)
3 //常用写法
4 int dmg = (ownplay) ? p.getSpellDamageDamage(3) : p.getEnemySpellDamageDamage(3);

```

计算我方/敌方法术治疗效果

□ C#

```
1 public int getSpellHeal(int heal)
2 public int getEnemySpellHeal(int heal)
```

计算我方/敌方随从治疗效果

□ C#

```
1 public int getMinionHeal(int heal)
2 public int getEnemyMinionHeal(int heal)
```

使用法术吸血

□ C#

```
1 public void applySpellLifesteal(int heal, bool own)
```

随从攻击随从（第三个参数：背叛效果）

□ C#

```
1 public void minionAttacksMinion(Minion attacker, Minion defender, bool dontcount =
```

用武器攻击(英雄攻击)

□ C#

```
1 public void attackWithWeapon(Minion hero, Minion target, int penalty)
```

我方/敌方出牌

□ C#

```
1 public void playACard(Handmanager.Handcard hc, Minion target, int position, int ch
2 public void enemyplaysACard(CardDB.Card c, Minion target, int position, int choice
```

降低武器耐久

□ C#

```
1 public void lowerWeaponDurability(int value, bool own)
```

获得获取清除所有Buff

□ C#

```
1 public void minionGetOrEraseAllAreaBufs(Minion m, bool get)
```

装备武器

□ C#

```
1 public void equipWeapon(CardDB.Card c, bool own)
```

召唤随从

□ C#

```
1 public void callKid(CardDB.Card c, int zonepos, bool own, bool spawnKid = true, bo
```

## 设置随从信息

### □ C#

```
1 //冻结一个随从
2 public void minionGetFrozen(Minion target)
3 //沉默一个随从
4 public void minionGetSilenced(Minion m)
5 //消灭一个随从
6 public void minionGetDestroyed(Minion m)
7 //设置随从控制方
8 public void minionGetControlled(Minion m, bool newOwner, bool canAttack, bool force)
9 //设置随从磁力
10 public void Magnetic(Minion mOwn)
11 //设置随从风怒
12 public void minionGetWindfury(Minion m)
13 //设置随从冲锋
14 public void minionGetCharge(Minion m)
15 //设置随从突袭
16 public void minionGetRush(Minion m)
17 //设置随从丢失突袭
18 public void minionLostCharge(Minion m)
19 //设置随从丢失圣盾
20 public void minionLosesDivineShield(Minion m)
21 //设置所有随从被消灭
22 public void allMinionsGetDestroyed()
```

## Buff相关 (攻击力, 血量, 护甲)

### □ C#

```
1 //获得一回合Buff (下回合消失)
2 public void minionGetTempBuff(Minion m, int tempAttack, int tempHp)
3 //设置一方全部随从获得Buff
4 public void allMinionOfASideGetBuffed(bool own, int attackbuff, int hpbuff)
5 //设置随从获得Buff
6 public void minionGetBuffed(Minion m, int attackbuff, int hpbuff)
7 //设置随从获得相邻buff
8 public void minionGetAdjacentBuff(Minion m, int angr, int vert)
9 //设置克苏恩获得Buff
10 public void cthunGetBuffed(int attackbuff, int hpbuff, int tauntbuff)
11 //设置攻击力为X
12 public void minionSetAngrToX(Minion m, int newAngr)
13 //设置血量为X
14 public void minionSetLifetoX(Minion m, int newHp)
15 //设置英雄获得护甲
16 public void minionGetArmor(Minion m, int armor)
17 //设置随从的攻击力为血量
18 public void minionSetAngrToHP(Minion m)
```

```

19 //交换随从的攻击力和血量
20 public void minionSwapAngrAndHP(Minion m)

```

受到伤害或者治疗：

□ C#

```

1 //对随从受到伤害或者治疗
2 public void minionGetDamageOrHeal(Minion m, int dmgOrHeal, bool dontDmgLoss = false)
3 //对一方全部随从受到伤害或者治疗
4 public void allMinionOfASideGetDamage(bool own, int damages, bool frozen = false)
5 //对一方全部成员（随从和英雄）受到伤害或者治疗
6 public void allCharsOfASideGetDamage(bool own, int damages)
7 //对一方全部成员（随从和英雄）受到【随机】伤害或者治疗（造成的伤害随机分配给所有敌人）
8 public void allCharsOfASideGetRandomDamage(bool ownSide, int times)
9 //对全部成员（随从和英雄）受到伤害或者治疗（除exceptID外，传入m.entitiyID）
10 public void allCharsGetDamage(int damages, int exceptID = -1)
11 //对全部随从受到伤害或者治疗（除exceptID外，传入m.entitiyID）
12 public void allMinionsGetDamage(int damages, int exceptID = -1)

```

弃牌（num：弃掉的牌数）

□ C#

```

1 public void discardCards(int num, bool own)

```

设置一个新的英雄技能，英雄卡用

□ C#

```

1 public void setNewHeroPower(CardDB.cardIDEnum newHeroPower, bool own)

```

摸一张牌

□ C#

```

1 public void drawACard(CardDB.cardName ss, bool own, bool nopen = false)
2 public void drawACard(CardDB.cardIDEnum ss, bool own, bool nopen = false)

```

移除卡牌

□ C#

```

1 public void removeCard(Handmanager.Handcard hcc)

```

将随从移回手牌（第三个参数：费用变化，降费填负数）

□ C#

```

1 public void minionReturnToHand(Minion m, bool own, int manachange)
2 //示例：冰冻陷阱
3 //奥秘：当一个敌方随从攻击时，将其移回拥有者的手牌，并且法力值消耗增加（2）点。
4 public override void onSecretPlay(Playfield p, bool ownplay, Minion target, int num)
5 {
6     p.minionReturnToHand(target, !ownplay, 2);

```

```
7     target.Hp = -100;
8 }
```

## 随从移回到牌库

□ C#

```
1 public void minionReturnToDeck(Minion m, bool own)
```

## 使随从变形成为另一个（例如：将所有随从转为传说随从）

□ C#

```
1 public void minionTransform(Minion m, CardDB.Card c)
```

## 搜索随从

□ C#

```
1 //随机搜索一个敌方目标造成单次伤害（参数二标记是否只为随从）
2 public Minion getEnemyCharTargetForRandomSingleDamage(int damage, bool onlyMinions
3 //搜索随从(searchmode:搜索模式)
4 public Minion searchRandomMinion(List<Minion> minions, searchmode mode)
```

## 得到一张随机（限定法力值）的卡牌（例如：退化）

□ C#

```
1 public CardDB.Card getRandomCardForManaMinion(int manaCost)
```

## 常用的一些成员

□ C#

```
1 //超载
2 p.ueberladung
3
4 //法术伤害
5 p.spellpower
6 p.enemyspellpower
7
8 //我方/敌方英雄
9 p.ownHero
10 p.enemyHero
11
12 //我方/敌方随从
13 p.ownMinions
14 p.enemyMinions
15 //待补充...
```

## SIM模版

□ C#

```
1 //召唤一个随从
2 CardDB.Card kid = CardDB.Instance.getCardDataFromID(CardDB.cardIDEnum.UNG_201t); /
```

```

3  int pos = ownplay ? p.ownMinions.Count : p.enemyMinions.Count;
4  p.callKid(kid, pos, own.own);
5
6  //卡牌数量
7  int cardsCount = (own.own) ? p.enemyAnzCards : p.owncards.Count;
8
9  //随从信息
10 List<Minion> temp = (own.own) ? p.enemyMinions : p.ownMinions;
11
12 //判定某张卡的种族
13 if((TAG_RACE)m.handcard.card.race == TAG_RACE.DRAGON)
14
15 //抓一张随机的卡
16 p.drawACard(CardDB.cardName.unknown, own.own);
17
18 //判定手牌有龙牌时做...
19 foreach (Handmanager.Handcard hc in p.owncards)
20 {
21     if ((TAG_RACE)hc.card.race == TAG_RACE.DRAGON)
22     {
23         //所做的事件
24         break;
25     }
26 }
27
28 //随从的法术强度增加模版 (+1)
29 public override void onAuraStarts(Playfield p, Minion m)
30 {
31     if (m.own) p.spellpower += 1;
32     else p.enemyspellpower += 1;
33 }
34
35 public override void onAuraEnds(Playfield p, Minion m)
36 {
37     if (m.own) p.spellpower -= 1;
38     else p.enemyspellpower -= 1;
39 }

```

## SIM书写常见问题 buff和debuff特效优化

### □ Code

```

1 //写在这个函数，速度更快一些
2 public void updateAdjacentBufs(bool own)中
3
4 使用**target**参数时
5 //一定要在使用前先判定不为null
6 if(target != null)
7 使用搜索随从后
8 //使用getEnemyCharTargetForRandomSingleDamage或searchRandomMinion之后，一定要验证返回

```

```
9 //判定不为null之后再调用
10 if(m != null)
```

## | data目录

### | CardDefs.xml(卡牌数据)

XML文件的版本的卡牌数据，是很容易理解的，只要看Tag的name属性就能够理解对应的意义，所以在这里就不进行讲解了。这里要讲的是的PlayRequirement的填写和意义  
PlayRequirement的添加位置：一张卡的\*\*\*\*前面。

### | 目标只能是随从

例如：沟渠潜伏者 - 战吼：消灭一个随从。亡语：再次召唤被消灭的随从。

#### □ Code

```
1 //REQ_MINION_TARGET
2 <PlayRequirement param="" reqID="1"/>
```

### | 目标只能是友方

例如：小型法术蓝宝石 - 选择一个友方随从，召唤一个它的复制

#### □ Code

```
1 //REQ_FRIENDLY_TARGET
2 <PlayRequirement param="" reqID="2"/>
```

### | 目标只能是敌方

例如：战槌挑战者 - ;战吼：选择一个敌方随从。与其战斗至死！

#### □ Code

```
1 //REQ_ENEMY_TARGET
2 <PlayRequirement param="" reqID="3"/>
```

### | 目标只能是受伤的随从

例如：斩杀 - 消灭一个受伤的敌方随从。

#### □ Code

```
1 //REQ_DAMAGED_TARGET
2 <PlayRequirement param="" reqID="4"/>
```

### | 目标只能是被冻结的随从

例如：冰爆 - 消灭一个被冻结的随从。



#### □ Code

```
1 //REQ_FROZEN_TARGET
2 <PlayRequirement param="" reqID="6"/>
```

## | 目标攻击力要求小于

例如：暗言术：痛 - 消灭一个攻击力小于或等于3的随从。

#### □ Code

```
1 //REQ_TARGET_MAX_ATTACK
2 <PlayRequirement param="[攻击力]" reqID="8"/>
```

## | 目标是除了自身英雄以外的目标

例如：破法者 - 战吼：沉默一个随从。

#### □ Code

```
1 //REQ_NONSELF_TARGET
2 <PlayRequirement param="" reqID="9"/>
```

## | 目标只能是指定种族的

例如：牺牲契约 - 牺牲一个恶魔，为你的英雄恢复#5点生命值。

#### □ Code

```
1 //REQ_TARGET_WITH_RACE
2 <PlayRequirement param="[填种族数字]" reqID="10"/>
```

## | 需要有目标

#### □ Code

```
1 //REQ_TARGET_TO_PLAY
2 //注意，有部分随从是**只能以随从为目标**的，这类随从可能是可以空场下场的，记得把这类随从标
3 <PlayRequirement param="" reqID="11"/>
```

## | 场上可以放的随从数目

例如：召唤守卫 - 召唤两个2/4的守卫。

#### □ Code

```
1 //REQ_NUM_MINION_SLOTS
2 //注意，如果是要召唤两个随从的，数目也是填1，因为两个随从只会出现一个，但是卡牌是可以使用的
3 <PlayRequirement param="[填数目]" reqID="12"/>
```

## | 需要武器才能使用

例如：吸血药膏 - 使你的武器获得吸血。

#### □ Code

```
1 //REQ_WEAPON_EQUIPPED
2 <PlayRequirement param="" reqID="13"/>
```

## | 目标只能是英雄

例如：阿莱克丝塔萨 - 战吼：将一方英雄的剩余生命值变为15。

#### □ Code

```
1 //REQ_HERO_TARGET
2 <PlayRequirement param="" reqID="17"/>
```

## | 无目标时也可以使用

例如：无面腐蚀者

#### □ Code

```
1 //REQ_TARGET_IF_AVAILABLE
2 <PlayRequirement param="" reqID="22"/>
```

## | 敌方随从最少需要X个才能使用

例如：关门放狗 - 战场上每有一个敌方随从，便召唤一个1/1并具有冲锋的猎犬。

#### □ Code

```
1 //REQ_MINIMUM_ENEMY_MINIONS
2 //注意，像是关门放狗这张牌数目就是填1
3 <PlayRequirement param="[填数目]" reqID="23"/>
```

## | 连击时有目标

例如：幽暗城勇士 - 连击：造成1点伤害。

#### □ Code

```
1 //REQ_TARGET_FOR_COMBO
2 <PlayRequirement param="" reqID="24"/>
```

## | 要求目标攻击力大于

例如：暗言术：灭 - 消灭一个攻击力大于或等于5的随从。

#### □ Code

```
1 //REQ_TARGET_MIN_ATTACK
2 <PlayRequirement param="[攻击力]" reqID="41"/>
```

## | 对面场上的全部随从最少需要X个

例如：弹射之刃 - 随机对一个随从造成\$1点伤害。重复此效果，直到某个随从死亡。

### □ Code

```
1 //REQ_MINIMUM_TOTAL_MINIONS
2 <PlayRequirement param="[填数目]" reqID="45"/>
```

## | 目标必须是嘲讽随从

例如：破盾者 - 战吼：沉默一个具有嘲讽的敌方随从。

### □ Code

```
1 //REQ_MUST_TARGET_TAUNTER
2 <PlayRequirement param="" reqID="46"/>
```

## | 目标只能是未受伤的随从

例如：暗影打击 - 对一个未受伤的角色造成\$5点伤害。

### □ Code

```
1 //REQ_UNDAMAGED_TARGET
2 <PlayRequirement param="" reqID="47"/>
```

## | 英雄技能

例如：召唤伙伴 - 英雄技能：随机召唤一个动物伙伴。

### □ Code

```
1 //REQ_MINION_OR_ENEMY_HERO
2 <PlayRequirement param="" reqID="50"/>
```

## | 手牌中有龙牌的话则有目标

例如：看台喷火龙 - 战吼：如果你的手牌中有龙牌，则对一个敌方随从造成7点伤害。

### □ Code

```
1 //REQ_TARGET_IF_AVAILABLE_AND_DRAGON_IN_HAND
2 <PlayRequirement param="" reqID="51"/>
```

## | 目标只能是传说随从

例如：盖斯 - 消灭一个传说随从

### □ Code

```
1 //REQ_LEGENDARY_TARGET
2 <PlayRequirement param="" reqID="52"/>
```

## | 敌方有武器时可以使用

例如：回响之锣 - 摧毁你对手的武器。

### □ Code

```
1 //REQ_ENEMY_WEAPON_EQUIPPED
2 <PlayRequirement param="" reqID="55"/>
```

## | 我方随从最少要X个可以使用

例如：穿刺者戈莫克 - 战吼：如果你拥有至少四个其他随从，则造成4点伤害。

### □ Code

```
1 //REQ_TARGET_IF_AVAILABLE_AND_MINIMUM_FRIENDLY_MINIONS
2 <PlayRequirement param="" reqID="56"/>
```

## | 目标只能是亡语随从

例如：哈霍兰公主 - 战吼：触发一个友方随从的亡语。

### □ Code

```
1 //REQ_TARGET_WITH_DEATHRATTLE
2 <PlayRequirement param="" reqID="58"/>
```

## | 最少需要X个奥秘才有目标

例如：麦迪文的男仆 - 战吼：如果你控制一个奥秘则造成3点伤害。

### □ Code

```
1 //REQ_TARGET_IF_AVAILABLE_AND_MINIMUM_FRIENDLY_SECRETS
2 <PlayRequirement param="[奥秘个数]" reqID="59"/>
```

## | 目标只能是潜行随从

例如：暗影大师 - 战吼：使一个潜行的随从获得+2/+2。

### □ Code

```
1 //REQ_STEALTHED_TARGET
2 <PlayRequirement param="" reqID="62"/>
```

## | 场上可以放一个随从且小于10个水晶：

### □ Code

```
1 //REQ_MINION_SLOT_OR_MANA_CRYSTAL_SLOT
2 <PlayRequirement param="" reqID="63"/>
```

## | 如果上个回合打过元素牌则有目标

例如：火焰使者 - 战吼：如果你在上个回合使用过元素牌，则造成5点伤害。

### □ Code

```
1 //REQ_TARGET_IF_AVAILABLE_AND_ELEMENTAL_PLAYED_LAST_TURN
2 <PlayRequirement param="" reqID="65"/>
```

## | 必须先使用其他卡牌

例如：暗影映像 - 每当你使用一张牌，变形成为该卡牌的复制。

### □ Code

```
1 //REQ_MUST_PLAY_OTHER_CARD_FIRST
2 <PlayRequirement param="" reqID="69"/>
```

## | 手牌未满

例如：合成僵尸兽 - 英雄技能：制造一个自定义的僵尸兽。

### □ Code

```
1 //REQ_HAND_NOT_FULL
2 //常用于英雄技能可以发现卡牌至手牌上的操作
3 <PlayRequirement param="" reqID="70"/>
```

这就是兄弟支持的大部分的**PlayRequirement**了（除了两个不常用的）

大家学会了如何增加卡牌信息之后，希望可以尝试添加一下新卡的数据。如果有人做好了，可以尝试着在评论区分享一下。

最后这里教大家一个快速搜索出需要设定条件的卡牌的方法。[“手牌中有龙牌”的全部卡牌](#)

## 附带：云雾王子的基础数据

### □ Code

```
1 <Entity CardID="ULD_293" ID="54493" version="2">
2   <MasterPower>00000012-be43-4d77-8780-0d77d38da392</MasterPower>
3   <Tag enumID="185" name="CARDNAME" type="LocString">
4     <deDE>Wolkenprinz</deDE>
5     <enUS>Cloud Prince</enUS>
6     <esES>Príncipe de las Nubes</esES>
7     <esMX>Príncipe de las nubes</esMX>
8     <frFR>Prince-nuage</frFR>
9     <itIT>Principe delle Nubi</itIT>
10    <jaJP>雲の公子</jaJP>
11    <koKR>구름 왕자</koKR>
```

```

12      <plPL>Książę chmur</plPL>
13      <ptBR>Príncipe das Nuvens</ptBR>
14      <ruRU>Принц облаков</ruRU>
15      <thTH>เจ้าชายเมฆ</thTH>
16      <zhCN>云雾王子</zhCN>
17      <zhTW>雲霧親王</zhTW>
18  </Tag>
19  <Tag enumID="184" name="CARDTEXT" type="LocString">
20      <deDE>[x]&lt;b>Kampfschrei:&lt;/b>;
21  Verursacht 6 Schaden,
22  wenn Ihr ein &lt;b>Geheimnis&lt;/b>;
23  kontrolliert.</deDE>
24      <enUS>&lt;b>Battlecry:&lt;/b> If you control a &lt;b>Secret&lt;/b>;
25      <esES>[x]&lt;b>Grito de batalla:&lt;/b>;
26  Si controlas un &lt;b>secreto&lt;/b>;,
27  inflige 6 p. de daño.</esES>
28      <esMX>&lt;b>Grito de batalla:&lt;/b> si controlas un &lt;b>Secret
29      <frFR>&lt;b>Cri de guerre :&lt;/b> si vous contrôlez un &lt;b>Sec
30      <itIT>[x]&lt;b>Grido di Battaglia:&lt;/b>;
31  infligge 6 danni se
32  controlla un &lt;b>Segreto&lt;/b>.</itIT>
33      <jaJP>[x]&lt;b>雄叫び:&lt;/b>;
34  自分の&lt;b>秘策&lt;/b>;が準備
35  されている場合
36  _____6ダメージを与える。_</jaJP>
37      <koKR>[x]&lt;b>전투의 함성:&lt;/b>;
38  내 전장에 &lt;b>비밀&lt;/b>;이 있으면,
39  피해를 6 줍니다.</koKR>
40      <plPL>&lt;b>Okrzyk bojowy:&lt;/b>;
41  Zadaj 6 pkt. obrażeń, jeśli kontrolujesz &lt;b>Sekret&lt;/b>.</plPL>
42      <ptBR>&lt;b>Grito de Guerra:&lt;/b> Se você controlar um &lt;b>Se
43      <ruRU>&lt;b>Боевой ключ:&lt;/b> если у вас есть активный &lt;b>се
44  [x]наносит 6 ед. урона.</ruRU>
45      <thTH>&lt;b>คำรามสู้ศึก:&lt;/b> ถ้าคุณมี &lt;b>ก๊ับดัก&lt;/b>; ในสนาม_
46      <zhCN>&lt;b>战吼:&lt;/b>;
47  如果你控制一个&lt;b>奥秘&lt;/b>;，则造成6点伤害.</zhCN>
48      <zhTW>&lt;b>戰吼:&lt;/b>; 若你場上有
49  &lt;b>秘密&lt;/b>;，造成6點傷害</zhTW>
50  </Tag>
51  <Tag enumID="351" name="FLAVORTEXT" type="LocString">
52      <deDE>Ein gar luftiger Zeitgenosse, der sich aber reinhängt, dass die Funl
53      <enUS>"In West Cloudidelphia, born and raised; flinging lightning for
54      <esES>Al oeste en Nubedelfia crecía y vivía, sin hacerle mucho caso a los
55      <esMX>"En Nimbusdelfia yo nací y crecí; con rayos que lanzar, fue una
56      <frFR>Le nouveau prince-nuage, maintenant avec un fourrage à la vanille p
57      <itIT>"Fulminando i miei nemici sono cresciuto, me la sono spassata,
58      <jaJP>「くもプリ」と呼ばれ親しまれている彼は、その秘密めいたイケボでファンに6ダッ
59      <koKR>내 비밀이 하늘에 닿아 올려 구름도 나를 듣기까지, 마음에 들 때까지.</koKR>
60      <plPL>Całe życie buja w obłokach.</plPL>
61      <ptBR>Isso é que é vida de príncipe: pernas pra cima e a cabeça, só nas nu

```

```

62      <ruRU>Вумаем в облаках по долгу службы.</ruRU>
63      <thTH>เกิดและโดนบ่ทองฟ้า เอาแต่ลอยไปลอยมาทั้งวัน</thTH>
64      <zhCN>在那西方云雾之乡
65      我就在那地方成长
66      每天都要到处浪荡
67      还要放个闪电听响</zhCN>
68      <zhTW>那個雲～那個霧啊～</zhTW>
69      </Tag>
70      <Tag enumID="325" name="TARGETING_ARROW_TEXT" type="LocString">
71          <deDE>Verursacht 6 Schaden.</deDE>
72          <enUS>Deal 6 damage.</enUS>
73          <esES>Inflige 6 p. de daño.</esES>
74          <esMX>Inflige 6 de daño.</esMX>
75          <frFR>Inflige 6_points de dégâts.</frFR>
76          <itIT>Infligge 6 danni.</itIT>
77          <jaJP>6ダメージを与える.</jaJP>
78          <koKR>피해 6</koKR>
79          <plPL>Zadaj 6 pkt. obrażeń.</plPL>
80          <ptBR>Cause 6 de dano.</ptBR>
81          <ruRU>Нанести 6 ед. урона.</ruRU>
82          <thTH>สร้างความเสียหาย_6_แต้ม</thTH>
83          <zhCN>造成6点伤害.</zhCN>
84          <zhTW>造成6點傷害</zhTW>
85      </Tag>
86      <Tag enumID="342" name="ARTISTNAME" type="String">Anton Zemskov</Tag>
87      <Tag enumID="45" name="HEALTH" type="Int" value="4"/>
88      <Tag enumID="47" name="ATK" type="Int" value="4"/>
89      <Tag enumID="48" name="COST" type="Int" value="5"/>
90      <Tag enumID="183" name="CARD_SET" type="Int" value="1158"/>
91      <Tag enumID="199" name="CLASS" type="Int" value="4"/>
92      <Tag enumID="200" name="CARDRACE" type="Int" value="18"/>
93      <Tag enumID="202" name="CARDTYPE" type="Int" value="4"/>
94      <Tag enumID="203" name="RARITY" type="Int" value="1"/>
95      <Tag enumID="218" name="BATTLECRY" type="Int" value="1"/>
96      <Tag enumID="321" name="COLLECTIBLE" type="Int" value="1"/>
97      <ReferencedTag enumID="219" name="SECRET" type="Int" value="1"/>
98      <Power definition="cabafd80-a4fc-474a-9b91-b9f36fe14d7d"/>
99      <Power definition="00000012-be43-4d77-8780-0d77d38da392">
100          <PlayRequirement param="1" reqID="59"/>
101      </Power>
102 </Entity>

```

它这里用到的就是 <PlayRequirement param="1" reqID="59"/>

注意：如果没有 <Power definition="xxxxx"> 这一行没有的话也没有关系，兄弟不读取此信息。

□ Code

```
1 <Tag enumID="342" name="ARTISTNAME" type="String">Anton Zemskov</Tag>
2 <Tag enumID="45" name="HEALTH" type="Int" value="4"/>
3 <Tag enumID="47" name="ATK" type="Int" value="4"/>
4 <Tag enumID="48" name="COST" type="Int" value="5"/>
5 <Tag enumID="183" name="CARD_SET" type="Int" value="1158"/>
6 <Tag enumID="199" name="CLASS" type="Int" value="4"/>
7 <Tag enumID="200" name="CARDRACE" type="Int" value="18"/>
8 <Tag enumID="202" name="CARDTYPE" type="Int" value="4"/>
9 <Tag enumID="203" name="RARITY" type="Int" value="1"/>
10 <Tag enumID="218" name="BATTLECRY" type="Int" value="1"/>
11 <Tag enumID="321" name="COLLECTIBLE" type="Int" value="1"/>
12 <ReferencedTag enumID="219" name="SECRET" type="Int" value="1"/>
13 <PlayRequirement param="1" reqID="59"/>
```

就像是这样也是可以的！

更新于 2024-02-11

CC BY-NC-SA 4.0