三国杀游戏平衡性与记分规则合理性分析报告

摘要

就一个游戏而言，对于参与者，需要研究不同的策略去达到胜利，而对于游戏设计者，则需要研究这个游戏的平衡性与记分规则的合理性，并不断去调整它们。

在本文中，我们将站在游戏设计者的角度研究最近较为流行的三国杀游戏的平衡性与记分规则合理性，通过简化三国杀游戏，建立一个用随机过程描述的游戏模型，对游戏参与者进行简单的讨论之后，我们给每个参与者的策略进行了较为合理的假设，在这个假设下，我们发现这个随机过程是时齐的马尔科夫有限链。

经过简化模型与玩家策略假设之后，我们给出了游戏平衡性与记分规则合理性的度量，进一步，我们利用概率转移算法，计算出玩家数是4人时的游戏平衡性与记分规则合理性。

我们发现，在 4 个人的时候，游戏是极其不平衡的，反贼有很大的概率死亡。对于更多玩家数的情况，概率转移模型已经不能胜任，于是我们采用了蒙特卡洛算法进行大量的随机试验，计算出了 5-10 人时各种身分分配方案下的平衡性与合理性，我们发现，在人数适中的情况下（确切地说是 6-8 人），游戏平衡性最佳，记分也非常合理，随着人数进一步增加，主公的优势过于明显，导致平衡性丧失，但是记分规则弥补了这个问题，使得各个身份的得分期望相差不多。

在计算平衡性与合理性的时候，我们还利用概率转移的结果验证了由大数定理保证的蒙特卡洛算法的有效性。

在本文的开始和最后，我们都试着讨论了研究三国杀游戏对现实生活的意义。

关键词 三国杀，动态博弈，概率转移算法，蒙特卡洛算法，游戏平衡性，记分规则合理性

目　　录

§1 介绍

1.1 背景：动态博弈

在现实生活中，无时无刻不在发生动态博弈：不同的玩家和联盟有不同的应对措施。比较熟悉的例子如：在商业上，公司的股东，员工，管理者，竞争对手。在学校里，教授，学生，行政人员，后勤人员。这样的例子甚至可以追溯到古时候，主公，忠臣，反贼，内奸。

这些博弈都有一个共同的特征：每个身在其中的人都属于一个或多个联盟，他们之间有不同的利益关系。

1.2 简介：三国杀游戏

三国杀是近来在大学生中比较流行的一个休闲纸牌游戏，它将杀人游戏和三国故事结合起来，玩家扮演刘备、孙权、曹操，或者关羽、张飞、赵云等三国著名人物，进行一场斗智斗勇的游戏。游戏开始，每一位玩家得到一张随机分发的身份牌。如果你扮演的是主公，必须马上亮出你的身份牌。你的主公身份会招致反贼的群起而攻，但你身边的忠臣们会不遗余力地保护你，同时也要小心内奸潜伏在你的周围。反贼的任务则是刺杀主公，一旦主公死亡，反贼就会胜利。忠臣的任务则是保护主公，只要主公成功地活到了最后，即消灭了所有的反贼和内奸，正义的一方就会获得胜利。内奸则希望借正义之手消灭所有的反贼，借反贼之手消灭主公身边的忠臣，内奸的任务最难完成，他要消灭除了自己之外的所有人。明确了你的游戏任务之后，就可以选择角色，这一步将决定你在三国中扮演哪一个历史人物，每一个人物都有特殊的技能属性，这些属性跟人物的真实性格是相关的，不同的技能属性可能给你的游戏带来不同的帮助。

1.3 动机：游戏趣味性

对于一个游戏而言，其趣味性在于结果的未知性，在三国杀游戏中，倘若游戏设计得不平衡，在分配身份和角色之后，就确定了胜负和相应的得分谁多谁少，那就会导致整个游戏的趣味性丧失。

在社会中也有这样的要求，比如，在赌场里，赌局不能一边倒；在学校里，两个老师教同一个课程，难度和给分不能相差太大，等等。

针对一个游戏而言，不同身份胜负的概率不应该相差太悬殊，在这个胜负概率较为平衡的基础上，还要给出一个记分的方法，也就是奖惩的措施，给赢面稍小的相对多奖励一些。这些平衡与合理的设计在以上社会现象中也同样存在。

综上，研究游戏平衡性与记分规则合理性不仅对三国杀游戏本身具有意义，对一些社会现象也具有同样的普遍意义。

1.4 问题：游戏平衡性与记分规则合理性

在这篇论文中，我们考虑这样的问题：

- 在三国杀游戏中，不同人数下，已经有不同身份个数的分配方案（例如：5 玩家，其中主公，忠臣，内奸各 1 个，反贼 2 个），在这些方案下，游戏的胜负平衡性应如何度量，在这种度量下，判断游戏的平衡性是否好。。
- 对已有的记分方案进行评估，给出记分规则合理性的度量，并判断记分规则是否合理。。

§2 简化的三国杀游戏

由于原有的三国杀模型过于复杂，且许多规则并不影响总体的游戏平衡性判定，所以我们研究以下三国杀游戏的简化模型。

2.1 不同的联盟

- 一共有若干个玩家，这些人围坐成一圈，按出牌顺序方向（逆时针）依次标号 $1, 2, \ldots, N$，标号为 1 的玩家是主公。
- N 名玩家中有 p 名忠臣，他们的编号是 $A_1, A_2, \ldots, A_p$。
- N 名玩家中有 q 名反贼，他们的编号是 $B_1, B_2, \ldots, B_q$。
- N 名玩家中有 r 名内奸，他们的编号是 $C_1, C_2, \ldots, C_r$。

在游戏一开始，会告诉每个玩家他们各自的身份。
每个人的初始体力值和最大体力值均是 4 。

注 2.1 与传统的三国杀游戏不同，在简化模型中没有角色这个概念，也就是说，没有特殊技能这个概念。而且在考虑到传统三国杀游戏中最大体力值是 3 的角色都拥有较强的能力，所以在简化模型中统一认为最大体力值是 4 。

2.2 回合

在简化模型中，牌堆认为是无限的序列，每种牌以一定的概率出现。
在开始游戏之前，每名玩家从牌堆摸取同样张数的游戏牌，作为起始手牌。
游戏是从主公开始，每人有一个游戏回合，逆时针方向按回合轮流进行。每回合分以下三个阶段组成，直到游戏结束。

摸牌阶段 从牌堆摸两张牌。

出牌阶段 可以出任意张牌，加强自己或攻击他人，只需遵照以下两条限制：每回合仅限用杀攻击一次；玩家摆在面前的牌（包括自己的装备）不能有两张同名的。

弃牌阶段 不想再出牌或不能再出牌时就进入弃牌阶段，此时检查自己的手牌（拿在手上的牌）数，是否超过自己当前体力值，弃掉手里多于当前体力值的手牌。

记在回合 t 结束后，主公，忠臣，反贼，内奸分别剩余 $O(t), P(t), Q(t), R(t)$ 人。

注 2.2 简化模型与传统的三国杀游戏中存在如下区别：在简化模型中，牌堆认为是无限的序列，这导致在传统的三国杀游戏中摸牌阶段中可能出现的“牌堆被摸完”的情况在简化模型中不存在。注意，由于在简化模型中不存在角色的概念和延时类锦囊，所以没有传统游戏中的回合开始阶段与回合结束阶段。

2.3 死亡判定

任何时候，如果某角色体力值减少为 0，则角色死亡。

死亡的角色手牌和该角色面前已装备的装备牌弃到弃牌堆。角色死亡的玩家亮出身份牌，并退出游戏。此后需要计算距离时，忽略已死亡的角色。

注 2.3 在简化模型中“任何人杀死反贼可获得摸三张牌的奖励”这条规则不存在。另外，传统的三国杀游戏中所规定的“如果主公误杀了忠臣，需要弃掉所有的手牌和已装备的牌。”在简化模型中不存在。

2.4 胜利条件

胜利条件是由拿到的身份牌决定的：

- 主公和忠臣的胜利条件是消灭所有反贼和内奸，即存在时刻 t 使得 $Q(t) = R(t) = 0$。
- 反贼则只需杀死主公就能获得胜利，即存在时刻 t 使得 $O(t) = 0$。
- 内奸则只有消灭除了自己之外的所有人才能取得胜利（包括别的内奸），即存在时刻 t 使得 $O(t) = P(t) = Q(t) = 0$ 且 $R(t) = 1$ 。

如果角色死亡导致任意一方的胜利条件达成，则游戏立即结束，也就是说，在使得上面三个等式成立的最小的时刻 t 游戏结束。

2.5 玩家距离

无论顺时针还是逆时针，取最短路径，即为两名玩家间的距离值。

对于一个局面 S（其中包括了所有人的信息），记 $d(S, i, j)$ 表示在这个局面下玩家 i 和玩家 j 的距离。

注 2.4 在简化模型中不存在“装备马可以改变玩家间的距离”这种说法，因为根本没有装备马。

2.6 游戏牌功能

游戏牌包括杀、闪、桃，及锦囊牌和装备牌。除装备牌外，游戏牌使用后均需弃掉。

2.6.1 普通牌

杀 出牌阶段使用，攻击一名在攻击距离内（除自己外）的玩家，若攻击成功，被攻击的玩家减 1 点体力。未装备武器时，玩家杀的攻击距离为 1。装备武器后，杀的攻击距离为武器的攻击距离。出现杀的概率是 30/71。

闪 在别的玩家对你出杀可以打出闪，闪避一次杀的攻击。牌堆出现闪的概率是 15/71。

桃 出牌阶段使用，为自己加一点体力。任何时候体力值不得超过体力上限。桃也可以在任何时候需要扣减最后一点体力时，对自己使用，抵消一点体力伤害。牌堆出现桃的概率是 8/71。

注 2.5 简化模型中削弱了桃的作用，即不能在其他玩家扣减最后一点体力的时候使用。

2.6.2 锦囊牌

过河拆桥 出牌阶段对除自己外任意一名玩家使用，随机抽对方一张手牌，或选择一张已装备的装备牌，并将之弃掉。牌堆出现过河拆桥的概率是 6/71 。

顺手牵羊 出牌阶段对距离为 1 的一名玩家使用，随机抽对方一张手牌，或选择一张已装备的装备牌，将之收入自己的手牌，顺手牵羊的距离与武器无关。牌堆出现顺手牵羊的概率是 5/71 。

注 2.6 简化模型中忽略传统三国杀中所有总数量小于 5 的牌。

2.6.3 装备牌

简化模型中只有一种装备牌。

武器 攻击距离是 3 。牌堆出现武器的概率是 7/71 。

注 2.7 简化模型除了忽略了所有马，防具和诸葛连弩外，并将剩下的 7 个武器的攻击距离加权平均计算得到平均攻击距离约为 3 。

2.7 简化模型梗概

相对于传统的三国杀游戏，简化模型有如下特性

1. 牌堆是无限的，且每种牌以给定概率出现。

2. 没有角色牌，所有玩家都除身份外处于相同水平，没有特殊技能以及一切与角色有关的概念。

3. 锦囊牌仅保留两类，即过河拆桥和顺手牵羊。

4. 桃不能用于对其他玩家的急救。

5. 当主公误杀忠臣或者任何人杀死反贼时，没有奖励和惩罚措施。

§3 随机过程模型

简化模型中的牌堆可以认为是一串独立同分布的随即变量 $\{X(i)\}$ 。每个随机变量的分布是根据简化模型中每类牌出现的概率决定的。这些随机变量的取值集合 $Card$ 就是所有牌的种类

简化模型的每个状态可通过一个随机过程 $\{S(t)\}$ 描述。其中 $S(t)$ 是一个随机向量，表示 t 时刻的状态，其分量包括

- 该时刻应该摸牌的玩家的编号 $N(t)$。
- 每个玩家剩余的体力值，记 $H(t,i)$ 表示此时第 i 个玩家的剩余体力值。
- 每个玩家手中剩余 4 张牌（可能少于 4 张牌）的情况，记四维向量 $V(t,i)$ 表示此时第 i 个玩家的剩余体力值。
- 每个玩家是否装备了装备牌，记 $W(t,i)$ 表示此时第 i 个玩家的是否装备武器，0 为未装备，1 为装备。

容易发现由简化模型的性质， t 时刻，标号是 $N(t)$ 人从牌堆拿的牌恰好是 $X(2t-1), X(2t)$ 。

我们称 S 的取值集合 $State$ 的元素为局面，包含摸牌玩家编号，每个玩家剩余体力值等等信息）。

对于这个随机过程，如果每个玩家的策略（非混合策略）给定且策略是仅根据当前状态决定的，在这种情况下，编号为 i 的人的策略会诱导一个从 $State \times Card^2$ 到 $State$ 的映射 $p(i)$ ，表示在面对一个局面的时候，当玩家 i 按照他的策略采取行动，等回合结束的局面。

而对于混合策略的情况，编号为 i 的人的策略也会诱导一个从 $State$ 上的分布与 $Card$ 上的分布的平方的笛卡尔积到 $State$ 上的分布的映射 $p(i)(\cdot)$ ，这个映射可以如下决定整个随机过程：

$$S(t+1) = p(N(t))(S(t), X(2t-1), X(2t)) \tag{3.1}$$

注意，初始条件 $S(1)$ 中分量满足：$P(N(1)=1)=1, P(H(t,i)=4)=1$ ，且

$$P(V(1,i)=(C_1,C_2,C_3,C_4))=P(C=C_1)P(C=C_2)P(C=C_3)P(C=C_4) \tag{3.2}$$

在“每个玩家的策略给定且策略是仅根据当前状态决定的”的假定下，注意到 $X(t)$ 是独立同分布的，可知这个随机过程是一个马尔科夫链。如果再假定“策略与时间 t 无关”，则这个马尔科夫链是时齐的。

在后面的文章中我们都作这样的假定，即每个玩家的策略给定的，策略是仅根据当前状态决定的且与时间无关。由于可能的状态数是有限的（因为玩家数量是有限的，最大血量是有限的，手牌可能的情况也是有限的）所以这个马尔科夫链是有限链，从而还有对应的概率转移矩阵。

基于以上随机过程模型，我们计算在不同假设下，即对于不同的 $p(i)$ 的概率转移矩阵，从而确定整个随机过程，以及随机过程中的一些统计量。

注 3.1 这样假设的合理性在于对于三国杀中的大多数玩家，他们不会注意之前的情况，而只注重当前的状态。

§4 简化模型的进一步假设

在本节中，我们首先对扮演不同身份玩家的决策作出一个较为合理的假设。

4.1 初始情况假设

为了进一步简化，我们假设在开始阶段，每个人手上都有两张闪，没有其他牌（包括锦囊，装备）。

4.2 决策假设

为了分析记分规则的合理性，我们需要对每个玩家的策略作出一定的假设。每个玩家在他的回合摸好两张牌之后，分两个步骤作出决策：选择对象，执行操作。

4.2.1 选择对象

下面是不同身份对应的不同对象选择策略：

反贼 如果所有忠臣都死了，在最后一个忠臣死的时候，所有反贼同时表明自己的身份俗称跳反，也就是说，所有人都知道反贼的身份当且仅当忠臣的人数为 0，即 $P(t)=0$。在忠臣没有全部死亡的情况下，反贼随机从能攻击到的活着的对象中（除了主公）等概率地随机选择对象。否则，反贼始终攻击主公，如果主公不在攻击范围内，则等概率地选择攻击范围内的活着的内奸。

主公 在反贼没有标明身份的情况下，随机从能攻击到的所有对象中等概率地选择一个对象。否则，随机从能攻击到的反贼中等概率地选择一个对象。

忠臣 忠臣始终随机从能攻击到的活着的对象中（显然，除了主公）等概率地随机选择，通过反贼的策略可知，不存在忠臣得知反贼身份的情况。

内奸 始终随机从能攻击到的活着的对象中（除了主公）选择体力值最大的，如果有体力值相等的，则等概率选择其中一个。如果只有主公存活，则攻击主公。

注 4.1 所谓对象就是接下来执行操作的时候需要攻击的对象，通过不同联盟之间的利益关系，以上对象的选择是有一定合理性的。注意，在简化模型中，装备牌只有攻击距离为 3 的“武器”，没有“马”的装备，所以不存在无法选择对象的情况，除了一种情况，就是反贼在表明身份之后，攻击范围内的所有人都是反贼。

4.2.2 执行操作

在确定了对象，或者以一个概率分布确定了一个对象之后，考虑该时刻玩家在拿上两张牌之后对这个对象执行的操作。

在出牌阶段，玩家首先将手上的所有装备牌装备到身上，由于装备只有“武器”，并且互相之间没有任何区别，所以如果已经有武器，则直接弃掉装备牌。此时，手上只剩余杀，闪，桃，过河拆桥与顺手牵羊。这些按照下面的策略对已经选择的对象执行。注意，以下的策略是按顺序执行的。

顺手牵羊 如果已选择的对象有装备，则对装备使用顺手牵羊，否则，等概率地选择对相的一张手牌（如果有的话），将其顺手牵羊。

过河拆桥 如果已选择的对象有装备，则对装备使用过河拆桥，否则，等概率地选择对相的一张手牌（如果有的话），将其过河拆桥。

杀 对已选择的对象使用。

闪 不操作。

桃 如果自己的体力值没有达到最大值，则对自己使用。

被攻击的玩家出闪，如果都没有，就扣血。

弃牌阶段，玩家弃掉所有手头的杀，桃，过河拆桥，顺手牵羊，仅保留闪。

注 4.2 根据弃牌规则，玩家没有可能出桃来抵消伤害。根据被攻击的玩家必定出闪可知，不存在血比闪的数量少的情况。

4.3 对假设策略的进一步说明

通过不同身份的目的，我们逐一说明各个身份的策略假设是具有合理性的。

主公 由于主公在开始时期不知道剩下人的身份，所以就随机攻击，当反贼暴露身份之后，很自然，主公应该针对反贼进行攻击。

忠臣 忠臣的策略就是不攻击主公，由于不知道其他人的身份，所以只能随机攻击，如果有反贼暴露，则应该攻击反贼。

反贼 反贼在忠臣灭亡的情况下，应开始攻击主公，尽快结束战斗。

内奸 内奸需要消灭所有其他人，所以它需要不断削弱周围的人，但是在开始的时候他不能攻击主公，直到只剩下他和主攻。

我们所假定的策略都是根据以上的几条制定的，所以具有一定的合理性。

4.4 初始状态分布

由假定的策略确定了一个齐时的马尔科夫有限链，由此可以通过状态与转移概率矩阵的不断迭代，得到各种中止状态的概率。

还是通过随机向量 $S(t) = (N(t), H(t), V(t), W(t))$ 记录每个状态在 t 时刻的分布。

注意，由于手上所留下的牌只可能是闪，这里 V 是一个有 n 各分量的向量，这里 n 是人数。

由前面的假定，初始状态 $S(1)$ 的分布是满足

$$P(S(1) = (1, (4, 4, \ldots, 4), (2, 2, \ldots, 2), (0, 0, \ldots, 0))) = 1 \tag{4.1}$$

注 4.3 结合初始情况和对玩家策略的假定，我们可以知道在使用顺手牵羊或者过河拆桥的时候，如果要对对象的手牌进行随机选择，则选择的结果一定是闪。

§5 平衡性与合理性的度量

平衡性体现在胜负结果之间差别的大小，我们通过差方来度量平衡性 Bal，定义如下

$$Bal = (P_0 - \bar{P})^2 + (P_1 - \bar{P})^2 + (P_2 - \bar{P})^2 + (P_3 - \bar{P})^2 \tag{5.1}$$

其中，P_0, P_1, P_2, P_3 表示主公，每个忠臣，每个反贼，每个内奸最终存活的概率，$\bar{P}$ 是这四个值的平均。

注 5.1 对于一个纸牌游戏，大家主要的目的就是娱乐，而每个人无论胜负，都希望自己能活到最后一刻，否则提前死亡，就只能看着别人进行游戏。所以这里采用每个身份最终存活得概率来度量游戏的平衡性。

合理性体现在得分期望之间的差别大小，同样通过差方来度量合理性 Rat，定义如下

$$Rat = (E_0 - \bar{E})^2 + (E_1 - \bar{E})^2 + (E_2 - \bar{E})^2 + (E_3 - \bar{E})^2 \tag{5.2}$$

其中，E_0, E_1, E_2, E_3 表示主公，忠臣，反贼，内奸的得分期望，$\bar{E}$ 表示四个得分期望的平均值。

注 5.2 对于记分，需要要求每个身份的得分期望不要相差太大，所以我们采用这样公式来度量记分合理性。

§6 概率转移算法

下面介绍迭代算法的细节。

6.1 算法框架

对于分布 $S(t)$，我们可以通过以上的策略完全确定 $S(t+1)$，下面用一个例子来说明。

例 6.1 对于一个四个人的游戏，其中编号为 $1, 2, 3, 4$ 的分别是主公，内奸，忠臣，反贼。如图 1 所是这是一个状态 $S(t)$ 中的一个局面，假设其出现的概率是 $1/2$。

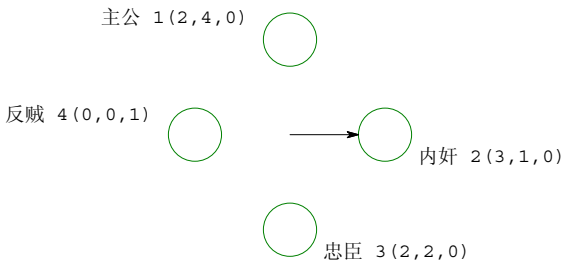

图 1: 状态 1

其中主公，内奸，忠臣，反贼的剩余体力值分别是 $2, 3, 2, 0$，他们手中闪的数量分别是 $4, 1, 2, 0$，该时刻轮到内奸。

假设内奸先后摸到一张杀和一张桃，这种情况出现的概率是 $\frac{30}{71} \times \frac{8}{71}$。根据之前的策略，内奸首先对自己使用桃，随后以 1 的概率攻击忠臣，此时忠臣出闪。

于是“先后摸到一张杀和一张桃”的条件下，对下一时刻的局面（如图 2 所示）有 $\frac{1}{2} \times \frac{30}{71} \times \frac{8}{71}$ 的概率的贡献。

对所有 t 时刻的可能的局面和两张牌不同的情况进行这样的计算就可以得到 $t+1$ 时刻的状态（也就是局面的概率分布）。

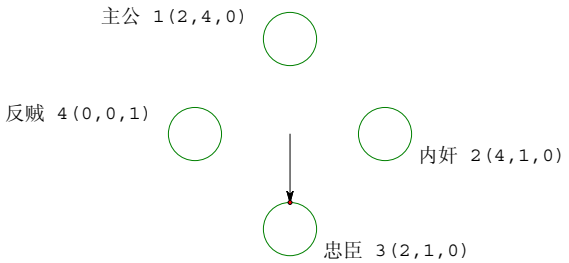

图 2: 状态 2

6.2 迭代中止条件

在时刻 t 的时候，当所有非中止状态（非中止状态就是尚未有某一方已经获胜）的概率都小于一个固定的常数的时候，我们就停止迭代。这里我们取这个常数是 $1/(2^{31}-1)$，其实为了实现的方便，以上算法都是将所有概率乘以一个常数 $2^{31}-1$ 之后进行迭代计算的。

6.3 计算结果分析

根据以上算法，我们设计了迭代的程序，代码见附录 A。

当玩家总数 $n=4$ 的时候，最终结果如表1所示，此时除中止局面外，其他局面的概率都是一个较小的量。表1中第一行的 0 代表“主公”，1 代表“忠臣”，2 代表“反贼”，3 代表“内奸”。从而 0123 代表逆时针分别坐“主公，忠臣，反贼，内奸”，其他以此类推。第一列中0代表死亡，1 代表存活，从左向右依次是主公，忠臣，反贼，内奸的生死状态。例如 0001 表示主攻死亡，忠臣死亡，反贼死亡，内奸存活。

	0123	0132	0213	0231	0312	0123	平均
迭代步骤	348	283	349	282	337	331	322
1 0 0 0	3.30%	45.75%	4.50%	29.16%	11.41%	6.00%	16.69%
1 1 0 0	26.74%	37.17%	5.67%	62.29%	6.52%	60.05%	33.07%
0 0 1 0	0.18%	9.21%	0.51%	3.28%	3.14%	0.21%	2.75%
0 0 0 1	69.78%	7.87%	89.32%	5.27%	78.91%	33.73%	47.48%
0 0 1 1	0.00%	0.00%	0.01%	0.00%	0.02%	0.00%	0.00%

表 1: 不同初始座位的终结状态概率及其平均

传统三国杀游戏中的计分规则如下：

主公获胜 主公得 10 分，每存活一个忠臣加5分；忠臣存活得 10 分，死亡得 7 分；反贼得 0 分；

内奸得 0 分，除非忠臣已死，内奸可以得 15 分。

反贼获胜 主公，忠臣得0分；反贼存活得 12 分，死亡得9分；内奸存活得 5 分减去存活反贼数，否则内奸可以得 0 分。

内奸获胜 主公得 3 分，内奸得 35 分，其他人不得分。

通过这些最终状态的概率分布，可以计算出各个身份的存活概率，如表2所示。

身份	主公	忠臣	反贼	内奸
存活概率	49.76%	33.07%	2.76%	47.49%

表 2: 不同身份的存活概率

通过表1的数据，还可以计算出每种初始座位下，不同身份的得分期望，如表3所示。

身份	0123	0132	0213	0231	0312	0123	平均
主公	6.43	10.39	3.98	12.42	4.49	10.62	8.05
忠臣	2.90	6.92	0.88	8.27	1.45	6.43	4.48
反贼	0.02	1.10	0.06	0.39	0.38	0.03	0.33
内奸	24.92	9.62	31.94	6.22	29.33	12.71	19.12

表 3: 不同初始座位的不同身份的得分期望与平均

此时，平衡性 $Bal = 14.05\%$ ，合理性 $Rat = 194.93$ 。

通过以上分析，在 4 个人的游戏中，通过概率转移算法，我们精确求出了各个角色的期望，结果表明游戏记分的平衡性在 4 个人的时候是非常不好的，对内奸非常有利，但是对反贼非常不利，这不利于游戏的趣味性。

§7 概率转移算法的优越性与局限性

首先，我们计算状态的总个数，注意到，一个状态就是一个局面，由当前出牌的人，每个人的剩余血量，每个人手中牌的数量，是否装备武器组成。再结合前面的一系列假定，还可以知道每个人手中牌的数量就是这个人手中闪的数量。

定理 7.1 对于一个玩家数量为 n 时，可能到达的状态不超过 $n \times 30^n$ 种。

证明 当前出牌的人可能性最多是 n，对于血量为 h 的人，根据注 4.2 他手中的牌（显然只有闪）的数量可能是 $0, 1, ..., h$ 中的一种，于是血量与手中牌的可能性总数是 $1+2+3+4+5 = 15$ 种，最后，是否有装备的可能性是 2 种。由乘法原理知，可以达到的状态不超过 $n \times 30^n$ 种。

□

对于通常使用的通过矩阵乘法迭代来计算马尔科夫过程，注意到，此时，矩阵的规模是状态数的平方，而由定理7.1的结论可以知道，矩阵的规模将非常大，从而会带来极高的时间复杂度。而之前提到的概率转移算法，其优越性在于充分利用了概率模型中转移概率矩阵的稀疏性，从原本的矩阵乘法来进行迭代的算法，转变为一个以概率来模拟过程从而实现迭代的算法，极大提高了运算效率。

虽然概率转移算法能够精确地计算出达到中止状态的概率，但是根据定理7.1的结论，对于 5 个玩家的情况，需要存储 5×30^5 个状态，数量级是10^8，一般计算机内存无法承受，另一方面，计算时间和状态数也成正比，且有一个较大的系数，容易看出在玩家数大于 6 的时候，时间复杂度早已无法承受，基于以上原因，我们需要研究新的算法，使之对较大的 n 同样能得到比较好的结果。

§8 蒙特卡洛算法

对于一般的 n 个玩家，我们考虑采取蒙特卡洛算法，即进行大量随机试验，导出中止状态的概率分布。

8.1 算法框架

蒙特卡洛算法的框架较为简单，在游戏开始前，随机将 n 个玩家的位置排好，每个人发放 2 张闪，从主攻开始摸牌，摸得两张牌也随机生成，每种牌的出现概率由之前的假定确定，并根据之前假设的策略去模拟接下来的每一步，直到达到一个中止状态。

根据大数定理，理论上，大量随机试验的到的中止状态的概率分布会收敛到由马尔科夫链计算出的中止状态概率分布，从而蒙特卡洛算法在理论上是有效的。

8.2 计算结果分析

根据以上算法，我们设计了随机模拟的程序，代码见附录 B。

对于不同的游戏人数和身份分配方案，我们进行 10^6 次试验，对不同的中止状态进行计数。由此可以计算出每个身份的存活概率，得分期望。具体数据见附录C中的表格。

通过各个身份存活概率和得分期望，我们可以计算出不同身份分配方案下的游戏平衡性和合理性，如表4所示。

游戏人数	主公	忠臣	反贼	内奸	Bal	Rat
4	1	1	1	1	13.35%	173.77
5	1	1	2	1	4.66%	84.28
6	1	1	3	1	3.78%	25.84
6′	1	1	2	2	2.42%	36.90

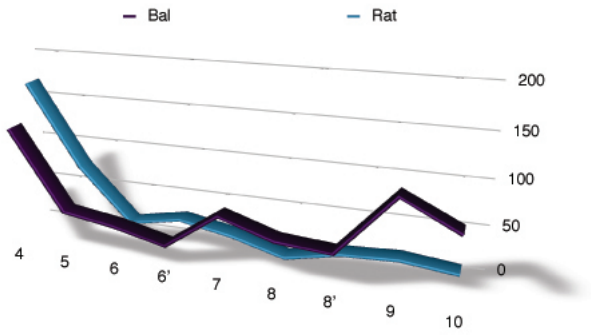

图 3: 平衡性与合理性

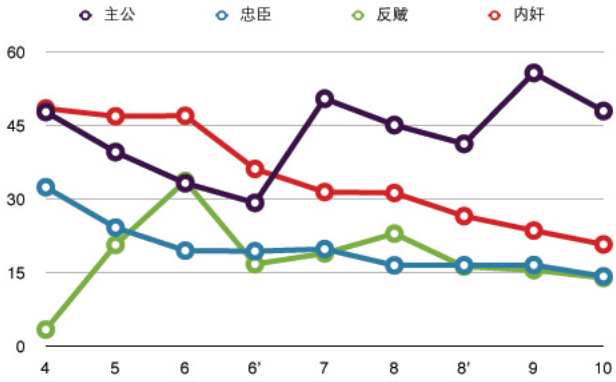

图 4: 存活概率

7	1	2	3	1	6.48%	26.09
8	1	2	4	1	4.58%	9.74
8´	1	2	3	2	4.16%	22.16
9	1	3	4	1	10.75%	25.03
10	1	3	4	2	7.82%	18.58

表 4: 不同身份分配方案的平衡性与合理性

如图3所示，我们可以看出平衡性和合理性随身份分配方案的变化而变化。
通过附录C的数据，我们可以跟踪不同角色的存活概率，如图4所示。
同样，我们可以跟踪不同角色的得分期望，如图5所示。
根据这些数据，我们进行如下分析：

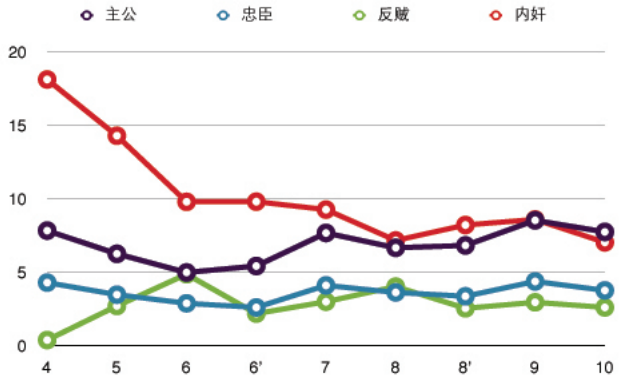

图 5: 得分期望

1. 在玩家数是 6 个，分配方案是 1 主公 1 忠臣 2 反贼 2 内奸的时候，游戏的平衡性最好，此时各个身份获胜的概率相差最小，游戏可玩性最强。
2. 在人数较少的情况下，玩家数是 4 个的游戏最不公平，反贼有很高的概率输掉比赛，于是游戏便在剩下三个人之间进行。
3. 人数较多的情况下，玩家数是 9 个，分配方案是 1 主公 3 忠臣 4 反贼 1 内奸的时候，游戏平衡性不佳，主公此时优势过于明显。对于玩家数是 10 个人的身份分配方案，也有同样的问题存在。
4. 虽然对于较多的人数游戏的平衡性不佳，但是考虑到记分，我们可以发现，即使在主公占有明显又是的情况下，但是各个身份的得分期望仍然相差不多。
5. 主公的存活概率随人数的增加而增加。内奸的存活概率始终高于忠臣和反贼。
6. 随着人数的增加，各个身份的得分期望越来越趋近。

§9 蒙特卡洛算法的有效性检验

之前的两个算法，即转移概率算法与蒙特卡洛算法，都能够给出玩家数是 4 时的结果，在此我们进行比较，对应数据如表5所示。

算法名称	主公	忠臣	反贼	内奸	Bal	Rat
概率转移	49.76%, 8.05	33.07%, 4.48	2.76%, 0.33	47.49%, 19.12	14.05%	194.93
蒙特卡洛	47.89%, 7.87	32.54%, 4.33	3.51%, 0.42	48.60%, 18.16	13.35%	173.77

表 5: 算法比较

其中，每个身份下的两个数字分别是存活概率和得分期望。可以看出两个算法得到的结果相近，这在实际上再次验证了蒙特卡洛算法的有效性，即其准确程度和概率转移算法相近。

§10 总结

通过以上分析比较我们可以得到如下结论：

1. 主公存活概率在 6 个玩家，在1忠臣2反贼2内奸的情况下，此时，整个游戏也最具平衡性，娱乐性较强。
2. 忠臣在存活概率比较平稳，随人数增加有缓慢的降低。
3. 反贼在人数多的情况下，存活概率与忠臣趋于一致，在6玩家1主公1忠臣3反贼1内奸的情况下存活概率最高。
4. 内奸在人数多的情况下存活的概率越来越小，在7人以下的情况下，还是具有一定优势的。
5. 在人中等情况下（即 $6-8$ 人），平衡性较好，记分规则也比较合理，游戏的有趣性和可玩性非常高。
6. 在人数较少的情况下，特别是 4 个人的时候，反贼劣势太明显，有戏可玩性较低。
7. 在人数较多的情况下，虽然主攻的优势过于明显，但是记分规则有效弥补了这个问题，游戏有趣性中等。

§11 讨论与开放问题

现实生活中的很多问题都与三国杀的更一般的情况有一个对应，游戏的平衡性对应于现实生活中的一个活动（或者是比赛）能否吸引他人来参加，而游戏记分规则的合理性对应于这个活动是否对参赛者公平。

在三国杀中，胜负条件决定了各方的策略，而这里的胜负条件是较为简单的，研究更加复杂的胜负条件，对现实生活的复杂情况更有意义。对于这种复杂情况，通过以上的算法去计算几乎是不可能的，对于一些特殊的复杂的情况，是否存在一个简单的游戏平衡性与记分合理性的度量，使得计算这些量非常的简单高效，这个问题有待日后进一步解决。

A 概率转移算法代码

源代码如下，采用 GCC 4.0 编译。

```
#include <stdio.h>
#include <stdlib.h>

#define n (4)
#define fn (15*15*15*15)
#define tn (2*2*2*2)
#define in (12+12*15+12*15*15+12*15*15*15)
#define mint (2147483647)

int s[n][fn][tn], t[n][fn][tn];

int multiply(int a, int b, int c) {
  if (mint / b > a) return (a * b) / c;
  else return (a / c) * b;
}

int min(int a, int b) {
  return a < b ? a : b;
}

int main() {
  const int ch[16] = {0, 1, 1, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4, 4};
  const int cv[16] = {0, 0, 1, 0, 1, 2, 0, 1, 2, 3, 0, 1, 2, 3, 4};
  const int cpc[6] = {15, 8, 7, 5, 6, 30};

  int flag, i, j, k, l, m, o, M, N, nOb, dp, ti, tj, tk, step;
  int id[n], H[n], V[n], W[n], R[4], Ob[n], tH[n], tV[n], tW[n], C[6];
  int ss[16];

  id[0] = 0; id[1] = 3; id[2] = 2; id[3] = 1;

  step = 0;

  memset(s, 0, sizeof(int)*n*fn*tn);
```

```
s[0][in][0] = mint;

do
{
  step++;
  memset(t, 0, sizeof(int)*n*fn*tn);
  flag = 0;
  for (i = 0; i < n; i++)
    for (j = 0; j < fn; j++)
      for (k = 0; k < tn; k++)
      if (s[i][j][k])
      {
        if (s[i][j][k] < 0) printf("Error!");
        N = i;
        R[0] = R[1] = R[2] = R[3] = 0;
        tj = j;
        for (l = 0; l < n; l++)
        {
          H[l] = tj % 15;
          tj /= 15;
        }
        tk = k;
        for (l = 0; l < n; l++)
        {
          V[l] = cv[H[l]];
          H[l] = ch[H[l]];
          W[l] = tk % 2;
          tk /= 2;
          if (H[l]) R[id[l]]++;
        }
        if (R[2] + R[3] == 0 || R[0] == 0 || R[0] + R[1] + R[2] == 0)
        {
          t[i][j][k] += s[i][j][k];
        }
        else
        {
          if (s[i][j][k] > flag) flag = s[i][j][k];
```

```
for (l = 0; l < n; l++) Ob[l] = 0;
if (W[N]) l = m = 3; else l = m = 1;
o = N;
while (l > 0)
{
  o = (o + 1) % n;
  if (o == N) break;
  if (H[o])
  {
    l--;
    Ob[o] = 1;
  }
}
o = N;
while (m > 0)
{
  o = (o + n - 1) % n;
  if (o == N) break;
  if (H[o])
  {
    m--;
    Ob[o] = 1;
  }
}
switch (id[N])
{
  case 0:
    if (!R[1]) for (l = 0; l < n; l++)
      if (Ob[l] && id[l] != 2) Ob[l] = 0;
    break;
  case 1:
    Ob[0] = 0;
    break;
  case 2:
    if (R[1]) Ob[0] = 0;
    else for (l = 1; l < n; l++) Ob[l] = 0;
    break;
```

```
  case 3:
    if (R[1] + R[2] || R[3] > 1)
    {
      Ob[0] = 0;
      m = 0;
      for (l = 1; l < n; l++)
        if (Ob[l] && H[l] > m) m = H[l];
      for (l = 1; l < n; l++)
        if (Ob[l] && H[l] < m) Ob[l] = 0;
    }
}
nOb = 0;
for (l = 0; l < n; l++) if (Ob[l]) nOb++;
for (l = 0; l < 36; l++)
{
  memset(C, 0, sizeof(int)*6);
  C[l%6]++;
  C[l/6]++;
  dp = multiply(multiply(s[i][j][k], cpc[l%6], 71)
       , cpc[l/6], 71);
  if (nOb)
  {
    dp /= nOb;
    for (M = 0; M < n; M++) if (Ob[M])
    {
      memcpy(tH, H, sizeof(int)*n);
      memcpy(tV, V, sizeof(int)*n);
      memcpy(tW, W, sizeof(int)*n);
      while (C[3])
      {
        if (tW[M])
        {
          tW[M] = 0;
          tW[N] = 1;
        }
        else if (tV[M])
        {
```

```
        tV[M]--;
        tV[N]++;
      }
      C[3]--;
    }
    while (C[4])
    {
      if (tW[M]) tW[M] = 0;
      else if (tV[M]) tV[M]--;
      C[4]--;
    }
    if (C[5])
    {
      if (tV[M]) tV[M]--;
      else tH[M]--;
    }
    tH[N] = min(tH[N] + C[1], 4);
    tV[N] += C[0];
    tW[N] = min(tW[N] + C[2], 1);
    tV[N] = min(tV[N], tH[N]);
    ti = (N + 1) % n;
    while (!H[ti] && ti != N) ti = (ti + 1) % n;
    for (m = 0; m < n; m++)
      tH[m] = tH[m] * (tH[m] + 1) / 2 + tV[m];
    tj = 0;
    tk = 0;
    for (m = n - 1; m >= 0; m--)
    {
      tj = tj * 15 + tH[m];
      tk = tk * 2 + tW[m];
    }
    t[ti][tj][tk] += dp;
  }
}
else
{
  memcpy(tH, H, sizeof(int)*n);
```

```
              memcpy(tV, V, sizeof(int)*n);
              memcpy(tW, W, sizeof(int)*n);
              tH[N] = min(tH[N] + C[1], 4);
              tV[N] += C[0];
              tW[N] = min(tW[N] + C[2], 1);
              tV[N] = min(tV[N], tH[N]);
              ti = (N + 1) % n;
              while (!H[ti] && ti != N) ti = (ti + 1) % n;
              for (m = 0; m < n; m++)
                tH[m] = tH[m] * (tH[m] + 1) / 2 + tV[m];
              tj = 0;
              tk = 0;
              for (m = n - 1; m >= 0; m--)
              {
                tj = tj * 15 + tH[m];
                tk = tk * 2 + tW[m];
              }
              t[ti][tj][tk] += dp;
            }
          }
        }
      }
  memcpy(s, t, sizeof(int)*n*fn*tn);
  printf("%d\n",flag);
} while (flag);

printf("%d\n", step);

for (i = 0; i < 16; i++) ss[i] = 0;

for (i = 0; i < n; i++)
  for (j = 0; j < fn; j++)
    for (k = 0; k < tn; k++)
    if (s[i][j][k])
    {
      N = i;
      R[0] = R[1] = R[2] = R[3] = 0;
```

```
        tj = j;
        for (l = 0; l < n; l++)
        {
          H[l] = tj % 15;
          tj /= 15;
        }
        tk = k;
        for (l = 0; l < n; l++)
        {
          V[l] = cv[H[l]];
          H[l] = ch[H[l]];
          W[l] = tk % 2;
          tk /= 2;
          if (H[l]) R[id[l]]++;
        }
        ss[R[0]+R[1]*2+R[2]*4+R[3]*8] += s[i][j][k];
      }

  for (i = 0; i < 16; i++) printf("%d :%d\n", i, ss[i]);

  return 0;
}
```

B 蒙特卡洛算法代码

源代码如下，采用 GCC 4.0 编译。

```
#include <stdio.h>
#include <stdlib.h>

#define n (10) #define NT 1000000

int Stat[5][5][5][5];

int min(int a, int b) {
  return a < b ? a : b;
}
```

```
int main() {
  int i, j, k, l, m, o, nOb, p;
  int nt = NT;
  int N, M, H[n], V[n], W[n], Ob[n];
  int R[4], C[6];
  int id[n];
  int idn[n] = {0, 1, 1, 1, 2, 2, 2, 2, 3, 3};

  int idm[4] = {1, 3, 4, 2};
  double vr[4], vs[4];

  int card[71] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
          1, 1, 1, 1, 1, 1, 1, 1,
          2, 2, 2, 2, 2, 2, 2,
          3, 3, 3, 3, 3,
          4, 4, 4, 4, 4, 4,
          5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5,
          5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5};

  srand(0);

  while (nt > 0)
  {
    memset(id, 0, sizeof(int)*n);

    id[0] = 0;
    i = 1;
    while (i < n)
    {
      j = rand() % n;
      if (!id[j] && j)
      {
        id[j] = idn[i];
        i++;
      }
    }
```

```
N = 0;

for (i = 0; i < n; i++)
{
  H[i] = 4;
  V[i] = 2;
  W[i] = 0;
}

while (1)
{
  R[0] = R[1] = R[2] = R[3] = 0;
  for (l = 0; l < n; l++) if (H[l]) R[id[l]]++;
  if (R[2] + R[3] == 0 || R[0] == 0 ||
      (R[0] + R[1] + R[2] == 0 && R[3] == 1)) break;

  memset(Ob, 0, sizeof(int)*n);
  if (W[N]) l = m = 3; else l = m = 1;
  o = N;
  while (l > 0)
  {
    o = (o + 1) % n;
    if (o == N) break;
    if (H[o])
    {
      l--;
      Ob[o] = 1;
    }
  }
  o = N;
  while (m > 0)
  {
    o = (o + n - 1) % n;
    if (o == N) break;
    if (H[o])
    {
```

```
    m--;
    Ob[o] = 1;
  }
}
switch (id[N])
{
  case 0:
    if (!R[1]) for (l = 0; l < n; l++)
      if (Ob[l] && id[l] != 2) Ob[l] = 0;
    break;
  case 1:
    Ob[0] = 0;
    break;
  case 2:
    if (R[1]) Ob[0] = 0;
    else for (l = 1; l < n; l++) Ob[l] = 0;
    break;
  case 3:
    if (R[1] + R[2] || R[3] > 1)
    {
      Ob[0] = 0;
      m = 0;
      for (l = 1; l < n; l++) if (Ob[l] && H[l] > m) m = H[l];
      for (l = 1; l < n; l++) if (Ob[l] && H[l] < m) Ob[l] = 0;
    }
}
nOb = 0;
for (l = 0; l < n; l++) if (Ob[l]) nOb++;

memset(C, 0, sizeof(int)*6);
C[card[rand()%71]]++;
C[card[rand()%71]]++;

if (nOb)
{
  do M = rand()%n; while (!Ob[M]);
  while (C[3])
```

```
  {
    if (W[M])
    {
      W[M] = 0;
      W[N] = 1;
    }
    else if (V[M])
    {
      V[M]--;
      V[N]++;
    }
    C[3]--;
  }
  while (C[4])
  {
    if (W[M]) W[M] = 0;
    else if (V[M]) V[M]--;
    C[4]--;
  }
  if (C[5])
  {
    if (V[M]) V[M]--;
    else H[M]--;
  }
  H[N] = min(H[N] + C[1], 4);
  V[N] = min(V[N] + C[0], H[N]);
  W[N] = min(W[N] + C[2], 1);
}
else
{
  H[N] = min(H[N] + C[1], 4);
  V[N] = min(V[N] + C[0], H[N]);
  W[N] = min(W[N] + C[2], 1);
}
M = N;
do M = (M + 1) % n; while (M != N && !H[M]);
N = M;
```

```
  }
  Stat[R[0]][R[1]][R[2]][R[3]]++;
  nt--;
  if (nt % (NT / 10) == 0) printf("%d\n", nt / (NT / 10));
}

printf("%d & %d & %d & %d & %d \\\\\n",
idm[0], idm[1], idm[2], idm[3], n);

memset(vr, 0, sizeof(int) * 4);
memset(vs, 0, sizeof(int) * 4);

for (i = 0; i < 5; i++)
  for (j = 0; j < 5; j++)
    for (k = 0; k < 5; k++)
      for (l = 0; l < 5; l++)
        if (p = Stat[i][j][k][l])
        {
          printf("%d & %d & %d & %d & %d \\\\\n", i, j, k, l, p);
          vr[0] += i * p;
          vr[1] += j * p;
          vr[2] += k * p;
          vr[3] += l * p;
          if (i + j + k == 0 && l == 1)
          {
            vs[0] += 3 * p;
            vs[3] += 35 / idm[3] * P;
          }
          else if (k + l == 0)
          {
            vs[0] += (10 + 5 * j) * p;
            vs[1] += (7 + 3. * j / idm[1]) * p;
            if (j + k + l == 0) vs[3] += 15. / id[3] * p;
          }
          else if (i == 0)
          {
            vs[2] += (9 + 3. * k / idm[2]) * p;
```

```
            vs[3] += (5. - k) * l / idm[3] * p;
          }
        }

  vr[0] /= (NT * idm[0]);
  vr[1] /= (NT * idm[1]);
  vr[2] /= (NT * idm[2]);
  vr[3] /= (NT * idm[3]);

  vs[0] /= NT;
  vs[1] /= NT;
  vs[2] /= NT;
  vs[3] /= NT;

  printf("%f\t%f\t%f\t%f\n", vr[0], vr[1], vr[2], vr[3]);
  printf("%f\t%f\t%f\t%f\n", vs[0], vs[1], vs[2], vs[3]);

  return 0;
}
```

C 原始数据

当游戏人数是 4 人，主公 1 人，忠臣 1 人，反贼 1 人，内奸 1 人时，有

主公	忠臣	反贼	内奸	次数
1	1	1	1	4
0	0	0	1	485928
0	0	1	0	35032
0	0	1	1	101
1	0	0	0	153542
1	1	0	0	325397

表 6: 4: 1 1 1 1

对应的存活概率和得分期望是

	主公	忠臣	反贼	内奸
存活概率	47.89%	32.54%	3.51%	48.60%

得分期望	7.87	4.33	0.42	18.16

表 7: 4: 1 1 1 1

当游戏人数是 5 人，主公 1 人，忠臣 1 人，反贼 2 人，内奸 1 人时，有

主公	忠臣	反贼	内奸	次数
0	0	0	1	367333
0	0	1	0	42848
0	0	1	1	12477
0	0	2	0	89974
0	0	2	1	90239
1	0	0	0	154155
1	1	0	0	242974

表 8: 5: 1 1 2 1

对应的存活概率和得分期望是

	主公	忠臣	反贼	内奸
存活概率	39.71%	24.30%	20.79%	47.00%
得分期望	6.29	3.51	2.74	14.33

表 9: 5: 1 1 2 1

当游戏人数是 6 人，主公 1 人，忠臣 1 人，反贼 3 人，内奸 1 人时，有

主公	忠臣	反贼	内奸	次数
0	0	0	1	234980
0	0	1	0	39799
0	0	1	1	9622
0	0	2	0	101698
0	0	2	1	79516
0	0	3	0	53681
0	0	3	1	147022
1	0	0	0	137557
1	1	0	0	196125

表 10: 6: 1 1 3 1

对应的存活概率和得分期望是

	主公	忠臣	反贼	内奸
存活概率	33.37%	19.61%	33.80%	47.11%
得分期望	5.02	2.92	4.90	9.83

表 11: 6: 1 1 3 1

当游戏人数是 6 人，主公 1 人，忠臣 1 人，反贼 2 人，内奸 2 人时，有

主公	忠臣	反贼	内奸	次数
0	0	0	1	515781
0	0	1	0	24245
0	0	1	1	7016
0	0	1	2	12886
0	0	2	0	35698
0	0	2	1	44828
0	0	2	2	66038
1	0	0	0	98561
1	1	0	0	194947

表 12: 6: 1 1 2 2

对应的存活概率和得分期望是

	主公	忠臣	反贼	内奸
存活概率	29.35%	19.49%	16.86%	36.27%
得分期望	5.46	2.64	2.22	9.84

表 13: 6: 1 1 2 2

当游戏人数是 7 人，主公 1 人，忠臣 2 人，反贼 3 人，内奸 1 人时，有

主公	忠臣	反贼	内奸	次数
0	0	0	1	220690
0	0	1	0	49643
0	0	1	1	5620
0	0	2	0	95569
0	0	2	1	43068
0	0	3	0	33358

0	0	3	1	46173
1	0	0	0	177501
1	1	0	0	258277
1	2	0	0	70101

表 14: 7: 1 2 3 1

对应的存活概率和得分期望是

	主公	忠臣	反贼	内奸
存活概率	50.59%	19.92%	19.04%	31.56%
得分期望	7.71	4.14	3.03	9.30

表 15: 7: 1 2 3 1

当游戏人数是 8 人，主公 1 人，忠臣 2 人，反贼 4 人，内奸 1 人时，有

主公	忠臣	反贼	内奸	次数
0	0	0	1	173067
0	0	1	0	49173
0	0	1	1	5297
0	0	2	0	109005
0	0	2	1	40022
0	0	3	0	56464
0	0	3	1	57576
0	0	4	0	19452
0	0	4	1	37710
1	0	0	0	169920
1	1	0	0	232631
1	2	0	0	49683

表 16: 8: 1 2 4 1

对应的存活概率和得分期望是

	主公	忠臣	反贼	内奸
存活概率	45.22%	16.60%	23.08%	31.37%
得分期望	6.70	3.66	4.06	7.20

表 17: 8: 1 2 4 1

当游戏人数是 8 人，主公 1 人，忠臣 2 人，反贼 3 人，内奸 2 人时，有

主公	忠臣	反贼	内奸	次数
0	0	0	1	353958
0	0	1	0	35081
0	0	1	1	5460
0	0	1	2	4236
0	0	2	0	56133
0	0	2	1	37625
0	0	2	2	22319
0	0	3	0	15093
0	0	3	1	29437
0	0	3	2	26752
1	0	0	0	133149
1	1	0	0	228874
1	2	0	0	51883

表 18: 8: 1 2 3 2

对应的存活概率和得分期望是

	主公	忠臣	反贼	内奸
存活概率	41.39%	16.63%	16.36%	26.65%
得分期望	6.86	3.40	2.58	8.25

表 19: 8: 1 2 3 2

当游戏人数是 9 人，主公 1 人，忠臣 3 人，反贼 4 人，内奸 1 人时，有

主公	忠臣	反贼	内奸	次数
0	0	0	1	161296
0	0	1	0	52932
0	0	1	1	3615
0	0	2	0	99409
0	0	2	1	27315
0	0	3	0	41506
0	0	3	1	30808
0	0	4	0	10691
0	0	4	1	13979

1	0	0	0	187841
1	1	0	0	259768
1	2	0	0	92128
1	3	0	0	18712

表 20: 9: 1 3 4 1

对应的存活概率和得分期望是

	主公	忠臣	反贼	内奸
存活概率	55.84%	16.67%	15.64%	23.70%
得分期望	8.57	4.41	2.99	8.63

表 21: 9: 1 3 4 1

当游戏人数是 10 人，主公 1 人，忠臣 3 人，反贼 4 人，内奸 2 人时，有

主公	忠臣	反贼	内奸	次数
0	0	0	1	271987
0	0	1	0	40431
0	0	1	1	4331
0	0	1	2	2169
0	0	2	0	68218
0	0	2	1	30765
0	0	2	2	10919
0	0	3	0	23678
0	0	3	1	29057
0	0	3	2	14347
0	0	4	0	4936
0	0	4	1	10058
0	0	4	2	8193
1	0	0	0	151980
1	1	0	0	240364
1	2	0	0	75525
1	3	0	0	13042

表 22: 10: 1 3 4 2

对应的存活概率和得分期望是

	主公	忠臣	反贼	内奸
存活概率	48.09%	14.35%	14.02%	20.87%
得分期望	7.78	3.80	2.64	7.07

表 23: 6: 1 1 3 1

参考文献

[1] Mark Barverman, Omid Etesami, Elchanan Mossel，*Mafia: A Theoretical Study of Players and Coalitions in a Partial Infomation Environment*，The Annals of Applied Probability，2008.

[2] 张波, 张景肖，应用随机过程，清华大学出版社, Springer，2004.