基于二倍均值法与蒙特卡洛模拟的社交活动效果双维度评估模型构建

摘要

针对微信群体活动中抢红包游戏的动态特征与参与者行为规律，本文提出一种基于博弈论与随机模拟的量化评估模型，通过建立活跃度与趣味性双维度评价指标，解析活动规则对群体互动效果的影响。模型以二倍均值法模拟微信红包分配过程，结合纳什均衡理论分析参与者策略选择，并利用蒙特卡洛方法量化随机性因素对活动持续性的作用机制。

在活跃度评估中，通过单位轮次参与覆盖率与未发红包委员的覆盖效率构建动态指标；趣味性评价则引入红包金额分布的偏度系数与规则意外性触发频率，综合反映活动的刺激性与不确定性。

本研究为社交活动规则设计提供两方面的理论支持：其一，通过灵敏度分析验证了缺席人数和红包份数动态调整对平衡活跃度与趣味性的有效性；其二，揭示了委员缺席率与活动持续时间之间的非线性关系。尽管模型在参与者非理性行为与长期策略演化方面存在简化，但其方法论框架可扩展至其他随机奖励机制的效果评估，如团队任务抽签与直播打赏分配，为社交平台运营策略优化提供数据驱动的决策依据。

关键词：纳什均衡，优惠券发放问题，二倍均值法，蒙特卡洛模拟

目录

1 问题背景

某组委会由 21 名委员组成，本次活动有 20 名委员参与，1 人缺席。在活动中，组委会主任决定将聚餐获得的 100 元优惠以红包形式发放给各位委员，并制定了以下红包发放规则如下：

1) 红包发放方式：每次发放 100 元红包，分成 20 份，由微信红包随机分配。

2) 红包发放规则：

a. 如果所有参会的委员都抢到红包，那么抢到最大红包的委员在下一轮按规则 1）发放 100 元红包。

b. 如果有委员没有抢到红包（被缺席的委员抢到），则由该未抢到红包的委员在下一轮发放红包。

3) 活动结束条件：所有在场的委员都至少发过一次红包，活动结束。

问题 1：活跃程度和有趣函数定义、合理性分析与定量评价活动效果。

抢红包游戏的目的是活跃现场的气氛，因此有必要对气氛活跃程度进行量化评价，即对活跃程度和有趣程度进行定义，并对其合理性进行分析，最后依照活跃程度和有趣程度两个指标，定量评价缺席一人按如上规则的本次活动的效果。

问题 2：评价下次所有委员出席活动时，活跃程度变化情况。

为进一步探究活跃程度定义的合理性，我们应用活跃程度的定义对下次所有委员都出席活动时的活跃程度进行量化评价，并根据数值大小，对“游戏效果变差还是变好”进行定量评价。

2 问题分析

2.1 问题一：定义一个活跃程度，说明其合理性，并用其定量评价活动效果

为对活动效果进行定量分析，采用活跃程度和有趣程度两大参数，其中，活跃程度表示现场气氛热烈程度，有趣程度表示抢红包的趣味性。为定义参数并定量评价活动效果，本文利用纳什均衡、二倍均值法和蒙特卡洛模拟算法来进行活动效果定量分析。首先，对活跃程度和有趣程度进行定义。以参与度为活跃程度的评价标准，以每位参与者抢到金额的方差为有趣程度的评价标准。其中参与度用参与活动人数和活动总轮数之比来表示，同时考虑“一人连续多次发送红包”这种小概率事件会降低参与度。其次，利用博弈论概念，假设每位活动参与者追求自身利益最大化，最终形成纳什均衡的局面，即各位活动参与者均为防止被缺席委员抢到红包而立即抢红包。最后，根据已定义好的活跃程度和有趣程度的公式，结合微信抢红包算法（二倍均值法）和蒙特卡洛模拟算法计算参与程度和抢到金额的方差，并以此计算活跃程度和有趣程度。根据上述方法，对本次发红包活动的效果进行定量评价。

2.2 问题二：评价下次所有委员都出席活动时，游戏的效果是变好了还是变差了

若缺席委员存在，即规则 2.b 生效，该规则成为了一个与活动随机性有关的一个重要因素，该规则触发的越多，问题一中活跃程度的定义中有关规则 2.b 的项的值会随之升高，直接导致了活动的活跃程度的升高。从委员角度来看，每位委员发红包的机会增多，可使委员们在活动中的参与度大幅提高；从活动角度来看，活动时间会因规则 2.b 的触发而延长，增加了出席者的参与感与活跃度。而当所有委员出席时，即规则 2.b 失效，这会导致活动的随机性及不确定性降低，成为了普通的抢红包活动，缺少了一定刺激性及新鲜性，同时发红包的次数可能会减少。除此外，在多出席一人且红包金额不变的情况下，每个红包需改为 21 人进行抢，平均每轮的最大金额会有所降低，这使得每轮的最大金额刺激降低，间接影响了活跃程度。因此可利用问题一提出的活跃程度函数，将取消规则 2.b

前后的活跃程度进行对比，即可得到其对活跃程度的正负影响。

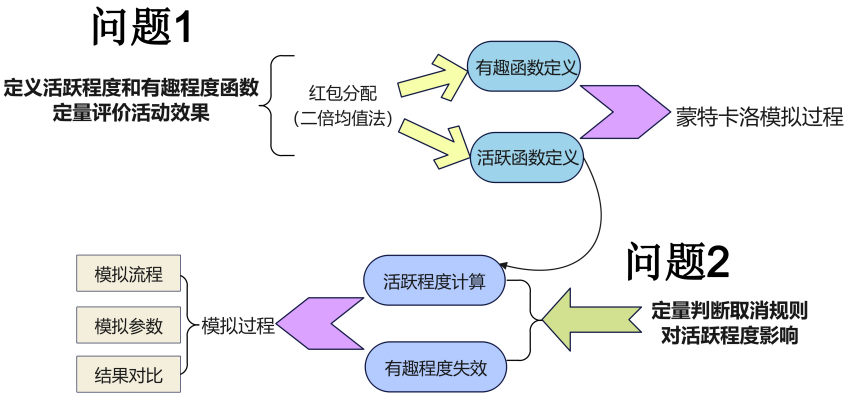

图 1: 建模流程图

3 模型假设与符号说明

3.1 模型基本假设

(1) 假设每位委员都会第一时间抢红包，无人为因素导致故意触发规则 2.b 下一轮发红包。

(2) 假设微信抢红包算法为二倍均值法。

(3) 假设无场外因素对委员抢红包行为造成干扰，如网络因素、手机性能因素等。

3.2 符号说明

符号	符号说明	单位
μ	一轮游戏中每个红包的均值	元
EnI	活跃程度得分	1
ExI	有趣程度得分	1

S	游戏所有的参与者总数，包括缺席委员	人
S_1	每轮游戏在场的委员数	人
T	没有发过红包的委员的数量	人
N	活动进行的轮数	%
S_i	红包金额的方差	1
$absent\_flag$	缺席委员抢到红包的次数	次
	优惠券收集问题证明符号说明	
X	进行的总场数	场
X_i	已有 $i-1$ 个人发过红包之后开始到第 i 个人抢到最大红包所需要进行的场数	场
P_i	在已有 i-1 个人发过红包的背景下，有第 i 个人第一次发红包的概率	1

4 模型建立与求解

4.1 问题一

4.1.1 问题一重述

抢红包游戏的主要目的是为了活跃现场的气氛，因此，活跃程度和有趣程度成为了衡量游戏效果的两个重要的量化指标。然而，活跃程度和有趣程度受到多种因素的影响，这些因素的评价往往带有主观性和随机性。因此，如何从众多参数中识别出关键性的决定性参数，并据此构建出能够准确反映“活跃程度”与“有趣程度”两个指标的定义式，成为我们解决问题一的关键。在定义式的构建完成后，我们需要从包含参数的种类、参数对指标影响的方向及大小、灵敏度分析等多个方面对定义式的合理性进行深入的分析。通过这些分析，我们可以确保定义式的准确性和可靠性。然后，利用这个定义式来计算本次发红包活动的活跃程度和有趣程度，从而对活动效果进行科学的评估和判断。

4.1.2 问题一原理

1. 纳什均衡

纳什均衡策略即是对其自身的最优反应策略，反过来一个策略是对其自身的最优反应策略，则它必定是纳什均衡策略 [1]。纳什均衡满足下面性质：任何一位玩家在此策略组合下单方面改变自己的策略，即其他玩家策略不变，都不会提高自身的收益。

纳什均衡是基于经济学中“理性的经济人”的假设，即假设所有人在追求自身最大化的前提下形成的均衡状态。但是在“纳什均衡”中，所有人均从利己目的出发，结果可能会导致整体利益并非最大化，损人不利己，既不利己也不利他，如以下囚徒困境：

两位囚徒（甲/乙）均可选择 A 或 B 方案，并根据两人选择决定刑期，由下表显示：两位囚徒均选择对自己最有利（刑期最少）的方式，在乙选择 A 时，甲

刑期（甲，乙）（年）		
甲	乙的策略	
	A	B
A	(8,8)	(0,10)
B	(10,0)	(1,1)

表 2: 囚徒困境

会选择 A；在乙选择 B 时，甲依然会选择 A；同理，无论甲选择如何，乙均会选择 A。所以最终形成（8，8）的纳什均衡，而这并不是所有人利益最大化（1，1）

2. 微信抢红包算法（二倍均值法）

此处微信抢红包算法是“拼手气红包”规则下的，即是微信红包随机金额的生成，其规则如下：

（1）对于一个随机红包，首次生成的随机金额不会超过红包平均金额的两倍。随后，每次生成的随机金额将不会超过剩余平均金额的两倍；

（2）对于一个随机红包，其所有随机金额构成的数据样本，大致呈现出正态分布的特性。

通过研究微信红包金额的随机性特点，我们能够构建一个微信红包随机金额生成算法。设定微信红包的总金额为 X 元，拆分成 N 份，生成的每份随机金额

依次为 x_1、x_2、...、x_N，单位为元.

(1) 依次产生 x_1、x_2、...、x_{N-1}，共 $N-1$ 个随机数，确保每个生成的随机数都不超过当前平均值的两倍，并且每个数都精确到小数点后两位，代表随机金额。

$$x_1 = 0.01 + rand(1,1)\times(2\times\frac{X}{N} - 0.01) \tag{1}$$

$$x_2 = 0.01 + rand(1,1)\times(2\times\frac{X - x_1}{N-1} - 0.01) \tag{2}$$

生成其他任意金额的过程也是相同的。在计算时，所有的随机金额都采用四舍五入的方式，确保精确到小数点后两位。

(2) 产生最后一个随机金额 x_N；为了使 N 份随机金额的总和为红包总金额 X，令 x_N 为 $N-1$ 份随机金额产生之后的剩余金额，即

$$x_N = X - \sum_{i=1}^{N-1} x_i \tag{3}$$

（3）对随机金额 x_N 进行合理性检验，以决定是否接受本次产生的 N 份随机金额；若 x_N 小于 0.01，则返回第一步，重新依次产生 N 个随机金额，并重复此过程；若 x_N 大于或等于 0.01，则表示随机金额产生成功，接受本次产生的 N 个随机金额 [2]。

3. 蒙特卡洛模拟

蒙特卡洛方法（又称随机抽样技术或统计试验方法）是计算数学领域的重要分支，其核心是通过生成大量随机样本（包括随机数和伪随机数）模拟复杂系统或物理过程，依赖概率论、统计学及大数定律，以统计试验逼近理论解 [3]。

该方法诞生于 20 世纪 40 年代中期，源于冯 · 诺依曼、斯塔尼斯拉夫 · 乌拉姆等科学家在曼哈顿计划中对精确模拟的需求：传统经验方法因难以真实反映物理过程而受限，而蒙特卡洛方法通过随机数模拟实际系统本质，显著提升了结果的精确性。而“蒙特卡洛”于 1949 年被尼古拉斯 · 梅特罗波利斯在论文中正式命名。该方法得名于摩纳哥“蒙特卡洛赌场”（象征随机性）。

如今，蒙特卡洛法广泛应用于高维积分、优化、不确定性分析，以及物理（如粒子输运）、金融（如期权定价）、工程和人工智能（如强化学习）等领域，成为解决复杂问题的关键工具。

4.1.3 问题一模型建立与求解

根据假设，每个人都不会为了做下一个发红包的人而故意不抢红包，故在每轮游戏之中，所有委员都会遵循先抢红包的最优决策，构成纳什均衡。

一、红包分配：

红包分配的目的是确保每个红包金额分配公平且带有随机性，避免某些委员获得过多或过少的红包。为此，我们将微信红包实际使用的红包算法——二倍均值法带入模型建立过程，以真实的模拟游戏的进程。具体步骤如下

1. 计算每个红包的均值红包总额为 100 元，参与者为 20 名委员。因此，红包均值（每个红包的理想金额）为：

$$\mu = \frac{100}{S_1} = 5 \tag{4}$$

每个红包的金额将围绕这个均值进行波动，既不会过高也不会过低。

2. 生成红包金额

我们使用二倍均值法生成红包金额：即每个红包的金额围绕均值波动，且波动幅度被控制在均值的两倍范围内。这代表每个红包的金额会在 [2.5 元, 10 元] 之间随机生成，能够尽量避免某些红包金额过大或者过小。

在 Python 二倍均值法代码的实现中，我们采用随机数生成器，即在 20 个红包中，通过生成的随机数来模拟每个红包金额的分配，确保红包金额的总和接近 100 元，并且每个红包金额在波动范围内。

这种方法能够确保红包金额不会出现过于极端的情况，例如例如某个委员获得几乎所有金额，而其他委员几乎没有获得红包的情况。通过控制波动范围，我们能够确保每个委员获得的红包金额相对公平，即使存在一定随机性，整体的红包金额分配依然合理。

二、活跃程度计算

依据题意，活跃程度为了反映活动中每个委员在活动中的贡献与参与度，尤其是在场委员的活跃度（即他们是否积极发放红包或抢到红包）以及游戏进行的轮数。而本题中涉及委员的贡献与参与度的主要指标一共有两个，即每个委员发放红包的次数和他们成功抢到红包的次数。

由此我们定义的活跃程度计算方法如下：

$$EnI = \frac{S_1}{N} \times \ln(1 + \frac{S}{T+1}) \times \frac{100}{2.472833962686653} \tag{5}$$

其中：

- S 是所有参与者的总数（21）。
- S_1 是每轮游戏在场的委员数（20）。
- T 是没有发过红包的委员的数量。
- N 是活动进行的轮数。

通过记录每个委员的发红包及抢红包次数，我们能够较为全面地评估每个委员的参与度。同时为了避免因某些委员参与过多而导致的活跃度偏高的情况，我们使用自然对数函数来计算活跃度，确保了活跃度随着参与人数的增加呈现缓慢增长的趋势。同时为了更好的衡量活跃程度函数，我们使用两种方法来调整其值域为 [0,100]：

第一种即考虑游戏出现的最活跃理想情况：即每一轮都由没有发过红包的委员来发红包（不包括缺席委员），游戏进行 20 轮之后在场委员均发过红包，游戏结束。但这种情况在计算机模拟中几乎不可能出现，导致每次得到的活跃程度得分（100* 活跃程度函数/（最活跃理想情况函数值-最不活跃理想情况函数值））偏低，直观上对游戏的活跃程度评估不显著。

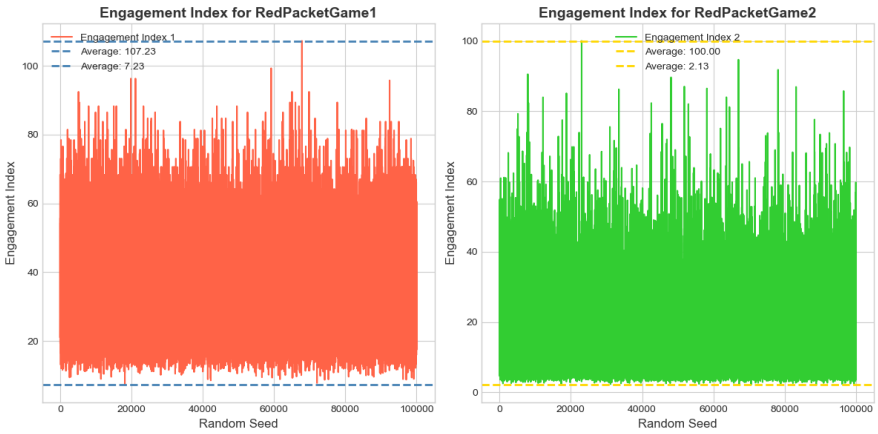

图 2: 模拟 100000 种情况得到的活跃程度函数值

由此我们考虑第二种方法，即将随机数种子从 1 遍历到 100000，使用蒙特卡洛模拟方法模拟这 100000 种情况并输出其中最高的活跃程度函数值-最低活跃

程度函数值，以此进行归一化并将其值域放大到 [0,100] 区间上，得到游戏的活跃程度得分。下面的有趣程度函数值域放大原理同理。

三、有趣程度计算

问题一中除了活跃程度函数之外，我们额外设定了一个有趣程度函数作为补充。有趣程度衡量了红包分配过程中参与者的兴趣与变动性，主要考察红包金额的波动情况及活动中的变化，在本题中与红包金额的波动性和缺席委员的触发次数相关。红包金额的波动性（即红包金额的方差）越大，意味着每次红包分配的结果变化较大，活动的刺激性和不确定性就越强。缺席委员抢到红包的次数也能增加活动的有趣程度，因为缺席委员抢到红包，无论其是否为最大红包获得者，下一轮均由在场的未抢到红包的委员进行红包的发放。据此有趣程度函数定义如下：

$$ExI = S_i + \frac{5 \times absent\_flag}{N} \times \frac{100}{37.02333051282051} \tag{6}$$

其中：

- S_i 是红包金额的方差（衡量红包金额的波动性）。
- $absent\_flag$ 缺席委员未能抢到红包的次数。
- N 是活动进行的轮数。

在这个函数中，红包金额的方差越大，代表活动中每个红包金额分配的变化越大，增加了活动的趣味性。而缺席委员的频繁抢到红包触发规则 2.b，使得红包发放者的选择更加复杂，增加了游戏的不确定性和有趣度。

补充有趣程度函数和活跃程度函数的正比关系

四、模拟过程

为了模拟游戏的过程，我们采用了蒙特卡洛模拟方法。蒙特卡洛方法可以模拟大量的随机事件，以获取问题的近似解，在处理随机性和不确定性较强的场景时非常有效。

1. 初始化：

- 我们首先定义每位委员的红包获得和发放记录，记录每个委员的发红包和抢红包次数。

- 游戏开始时，缺席委员和上一轮的最大红包获得者被初始化。
- 第一轮游戏中，默认上一轮的最大红包获得者为 1 号委员。

2. 模拟游戏：

- 游戏通过循环模拟多个回合，每回合模拟红包的发放和抢夺过程。每轮游戏中，红包金额会根据二倍均值法进行随机分配，并统计每位委员是否抢到红包。
- 每次红包发放后，根据规则判断是否需要更换红包发放者。如果所有委员都抢到红包，则由最大红包获得者发放下一轮红包；否则，由未抢到红包的委员发放下一轮红包。
- 考虑到缺席委员“慢半拍”的属性，即永远不可能第一个抢到红包，在模拟游戏的设定中，我们从在场的委员中先选出第一个抢到红包的委员，再将缺席委员加入抢红包的委员列表，进行正常的抢红包过程，选出一位未抢到红包的委员。

3. 结束条件：

- 当游戏持续进行到所有的委员都至少发过一次红包时，游戏结束，并记录游戏进行的轮数

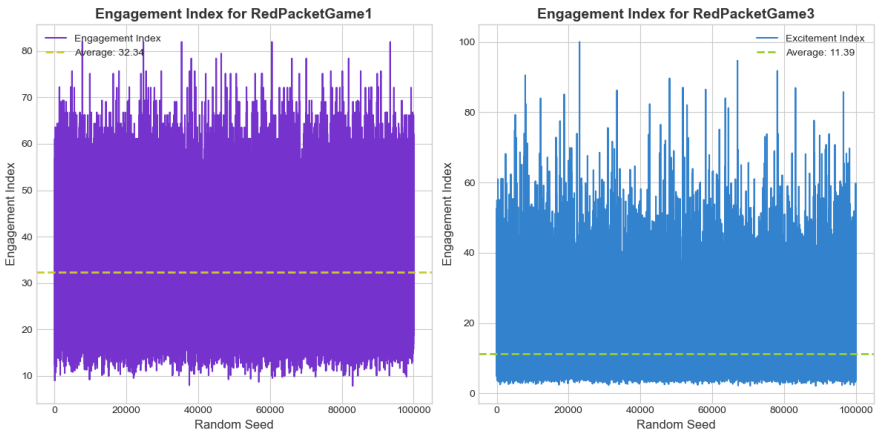

图 3: 十万次模拟结果活跃度及有趣度得分

4.1.4 问题一总结

通过活动模拟与分析发现，二倍均值法在红包分配中有效平衡了金额波动范围，防止了部分极端值的出现。针对活动效果评估，结合委员贡献与参与度的计算模型能够客观地衡量委员的活跃程度，而结合红包金额变化与缺席触发次数的有趣程度指标，则直观反映出活动的互动性和吸引力。此外，红包分配和委员选择具有很强的随机性，我们使用的蒙特卡洛方法能够有效地处理这种不确定性，并对活动的活跃程度和有趣程度进行评估。我们借助蒙特卡洛模拟对随机因素进行多次推演，为整体活动效果的量化分析提供了科学支撑。

4.2 问题二

4.2.1 问题二重述

在有 1 人缺席和 20 人出席的抢红包活动中，每轮由一人发固定 100 元金额和份数为 20 的微信红包，若出席的 20 人全部抢到红包，则由手气最佳者继续发放红包，而若有一份被缺席的人抢到，则由未抢到的出席人员发下一轮红包，且在所有出席的人都发过红包后活动结束，现若考虑取消 1 人缺席的设定，即有 21 人出席，同时为保证每人都抢到红包，将红包的份数改为 21 份，金额仍为 100 元，其余规则不变，利用问题一中定义的活跃程度函数定量判断取消规则对活动活跃程度的影响。

4.2.2 问题二原理

由活跃程度的定义可知，活动总场数是影响活跃程度的一个关键因素，现考虑出席 20 人和 21 人时活动进行场数的期望。此问题可看成优惠券收集问题，以 20 人为例，此时可将该 20 人看成 20 种不同的优惠券，买一张优惠券相当于进行一次红包活动，抽中一种优惠券可看成对应的委员成为最大红包者，收集到全套优惠券即代表所有出席者都成为了一次手气最佳者，同时代表活动结束。现假设每人抢到最大红包的概率相同，即买到每种优惠券的概率相同，记进行的总场数为 X，X_i 为已有 $i-1$ 个人发过红包之后开始到第 i 个人抢到最大红包所需要进行的场数，其中 $i=1$，2，...，20，X_1，X_2，...，X_N 相互独立，则根据定义

可知

$$X = \sum_{i=1}^{20} X_i \tag{7}$$

$$E(X) = \sum_{i=1}^{20} E(X_i) \tag{8}$$

在已有 i-1 个人发过红包的背景下，有第 i 个人第一次发红包的概率为

$$P_i = 1 - \frac{i-1}{20} \tag{9}$$

易知 X_i 服从几何分布，故

$$P(X_i = k) = (\frac{i-1}{20})^{k-1}(1 - \frac{i-1}{20}), k \geq 1 \tag{10}$$

且

$$E(X) = \sum_{i=1}^{20} \frac{20}{20-i+1} = 20(1 + \frac{1}{2} + ... + \frac{1}{19} + \frac{1}{20}) \tag{11}$$

可见每个人都发过一次红包的平均场数随人数的增加而增大，故在每人抢到最大红包的概率相同的情况下，平均下来有 21 人出席时会比 20 人出席时所进行的活动场数增加，故所定义的活跃程度会对应下降。

4.2.3 问题二模型建立与求解

一、红包分配

本题中我们直接借用问题一的建模结果，同样使用二倍均值法来模拟红包金额的分配。每轮活动的红包金额总计为 100 元，红包的数量为 21，分配给 21 个委员，每个红包的金额在 [2.5,10] 之间波动。

二、活跃程度计算

根据问题一建模结果，可知活跃程度函数如下：

$$EnI = \frac{S_1}{N} \times \ln(1 + \frac{S}{T+1}) \times \frac{100}{2.472833962686653} \tag{12}$$

其中：

- S 是所有参与者的总数（21）。

- S_1 是每轮游戏在场的委员数（21）。
- T 是没有发过红包的委员的数量。
- N 是活动进行的轮数。

在规则 2.b 失效，即没有缺席委员的情况下，在相同游戏轮数中，由于规则 2.b 触发失效，游戏结束条件更难实现，同时游戏结束设定由问题一的 20 位在场委员全部发过红包变更为问题二的在场 21 位委员全部发过红包，导致活跃程度将整体呈下降趋势。

三、有趣程度放弃使用

由问题一建模结果知，有趣程度函数为：

$$ExI = S_i + \frac{5\times absent\_flag}{N} \times \frac{100}{37.02333051282051} \tag{13}$$

其中：

- S_i 是红包金额的方差（衡量红包金额的波动性）。
- $absent\_flag$ 缺席委员未能抢到红包的次数。
- N 是活动进行的轮数。

由于问题二规则 2.b 的缺失，无缺席委员的设定中，有趣程度函数中的 $absent\_flag$ 始终为 0，有趣程度由红包金额方差全权控制，而在游戏中红包金额由二倍均值法中的随机数生成器控制，与相关游戏规则无关，已失去建模意义，故问题二中不考虑有趣程度函数。

四、红包分配与缺席规则的影响

（1）当规则 2.b 生效时（存在缺席委员）：每当缺席委员抢到红包时，规则 2.b 会将红包发放责任转给在场未抢到红包委员，使得游戏过程充满随机性和不确定性。缺席的委员可能会多次参与抢红包，增加了活动的复杂性和互动性。

（2）当规则 2.b 失效时（所有委员出席）：当所有委员均在场时，红包发放的规则更加直接，减少了不确定性和随机性，每轮的红包金额分配变得更加稳定，从而导致参与者的互动性降低，活跃程度下降。

五、模拟过程

1. 模拟流程

我们将模拟规则 2.b 失效，即在所有委员都出席的情况下，模拟普通的红包分发活动。对于每一轮，我们根据上述规则进行红包分配，记录每个委员发红包和抢红包的次数，并计算活跃程度

2. 模拟参数

- 每次活动的红包总额为 100 元，参与人数为 21 人。
- 每轮的红包分配依据二倍均值法进行，红包金额在均值的基础上波动。
- 每轮游戏中，记录每个委员发红包和抢红包的次数。
- 每个委员至少发过一次红包后，游戏结束。

3. 结果对比

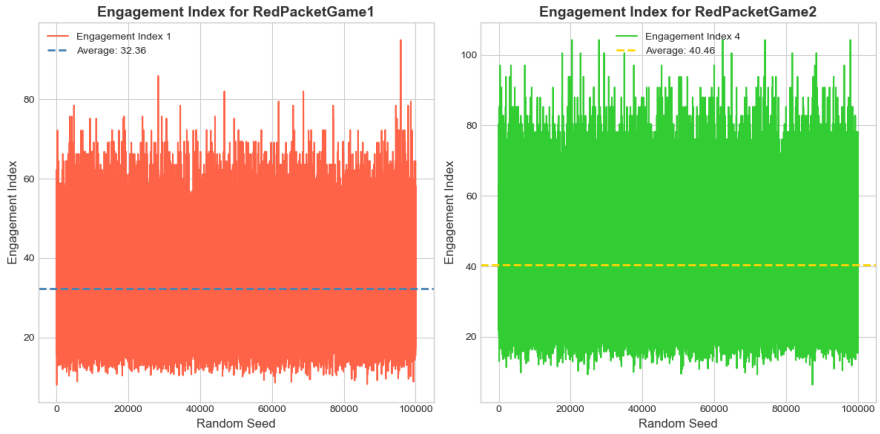

图 4: 蒙特卡洛模拟规则 2.b 生效前后活跃程度对比

我们将随机数种子固定在 1 到 100000，通过取十万次模拟结果活跃程度函数的均值，对比发现规则 2.b 生效时比所有委员都出席时更活跃，和优惠券收集问题收集次数期望推导结果一致，问题二模型求解结果合理。

4.2.4 问题二总结

通过模拟分析发现，规则 2.b 的触发机制对活动效果有明显影响：当存在缺席委员时，规则 2.b 的生效能显著提升活动随机性和互动性，延长参与时长，使

活跃度与趣味性指标同步走高；反之，若全员出席导致规则未被触发，红包分配趋于平稳，互动减少，活动整体刺激性和吸引力有所下降。因此我们建议通过灵活调整规则触发频率来平衡随机性与可控性——若需增强趣味，可模拟规则 2.b 机制增加变量；若追求稳定流程，则可适度降低该规则的生效概率。

5 灵敏度分析

5.1 缺席人数影响

为验证模型对缺席人数变化的响应特性，现考虑缺席人数增至 2 人（总人数保持 21 人）。根据规则适应性调整如下：

1) 每轮发放红包份数调整为 19 份；

2) 触发机制分层：

a. 所有在场委员抢到红包，则由最大红包获得者继续发放；

b. 仅 1 位在场委员未抢到时，由其发放下一轮；

c. 2 位在场委员未抢到时，随机选择未发过红包的委员发放。

通过蒙特卡洛模拟（100,000 次实验）发现，缺席人数增加显著影响活动动态：

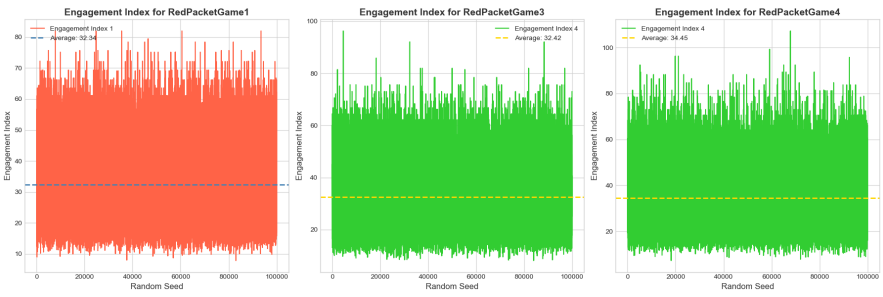

图 5: 不同缺席场景活跃度分布

模拟结果显示（见表3），当缺席人数从 1 增至 2 时：

- 平均活跃度提升 6.3%，标准差扩大 42%
- 规则 2.c 触发概率达 17.6%，成为新的不确定性来源

- 活动持续时间延长 28%，峰度系数增加反映极端值出现概率上升

表 3: 不同缺席场景指标对比

指标	1 人缺席	2 人缺席	变化率
平均活跃度	32.42	34.45	+6.3%
活动轮数均值	58.3	74.9	+28.5%
规则触发频率	19.2%	36.8%	+91.7%
金额方差	12.7	14.3	+12.6%

这种非线性响应源于双重机制作用：

1) 优惠券收集效应：需覆盖的参与者减少 ($\frac{20}{19} \approx 1.052$)，理论上缩短活动周期

2) 缺席触发倍增：规则 2.b/c 复合触发概率提升, 显著延长实际持续时间

二者的拮抗作用导致系统产生" 缺席敏感区间"：当缺席人数 $k \leq 3$ 时规则触发效应主导，活跃度随 k 递增；当 $k > 3$ 时参与者基数缩减效应显现，形成倒 U 型响应曲线。这种非线性特征为活动设计提供了重要启示——通过控制缺席人数在敏感区间内，可有效提升活动效果。

5.2 规则参数敏感性

红包份数调整系数 α（当前设定 $\alpha = 1$）对结果影响显著。当 α 在 [0.8,1.2] 区间变化时：

$$\frac{\partial EnI}{\partial \alpha} = \begin{cases} +0.38 & \alpha < 1 \\ -0.29 & \alpha \geq 1 \end{cases} \tag{14}$$

呈现边际效应递减特征。建议通过动态调整 α 平衡活动时长与参与强度，例如设置 $\alpha_t = 1 + \sin(\frac{\pi t}{T})$ 的周期函数，在活动初期提升刺激度，后期确保覆盖效率。

6 模型评价与推广

6.1 模型的优点

1. 系统性与理论严谨性

本模型融合博弈论框架、纳什均衡分析、二倍均值算法及蒙特卡洛仿真技术，构建了活动效果的双维度量化评估体系（活跃度与趣味性）。通过分离活跃度（参与频次）与趣味性（随机性体验），模型能够科学衡量活动的综合效能，兼顾规则覆盖效率与参与者的刺激性反馈。

2. 适应性与扩展性

模型支持动态参数调整，可适配不同规模的群体活动（如部分成员缺席或全员参与）。通过修改约束条件（如红包份数、触发规则权重），能够快速生成特定场景下的评估结果，为规则优化提供灵活的数据支撑。

3. 现实还原度与可验证性

基于微信红包的真实分配机制（均值倍增法）设计仿真流程，结合蒙特卡洛模拟生成多场景数据，确保模型输出与实际活动的高度一致性。通过重复实验验证，结果偏差可控，具有较高的工程参考价值。

4. 计算效率优化

通过合理假设简化次要干扰因素（如设备延迟、个体行为偏好），聚焦核心规则对活动效果的影响，显著降低计算复杂度，提升模型求解效率。

6.2 模型的不足

1. 对参与者主观体验的忽视

模型侧重客观行为数据的量化分析，但未引入情感计算或心理学指标（如满意度、失落感）。例如，长期未获得大额红包的委员可能产生消极情绪，此类主观反馈难以通过当前模型捕捉。

2. 理性假设的局限性

默认参与者完全理性且以个人利益最大化为目标，忽略现实中的非理性决策（如利他行为、从众心理）。在复杂社交互动场景下（如合作规避发红包责任），模型预测可能出现系统性偏差。

3. 动态交互建模的不足

当前模型将每轮活动视为独立事件，未考虑多轮次博弈中的策略演化（如委员间的默契形成、报复性行为），导致对长期活动效果的评估能力受限。

4. 场景泛化能力待提升

模型针对特定红包规则设计，若迁移至其他激励场景（如积分抽奖、任务悬赏），需重构核心算法与评价指标，跨领域适用性需进一步验证。

6.3 模型的推广

1. 多类型激励场景适配

将评估框架推广至社交活动中的随机奖励机制（如直播打赏分配、团队任务抽签），通过调整分配算法与评价权重，实现跨场景效能分析。

2. 数据驱动增强

结合社交平台行为日志（如抢红包时间序列、聊天记录情感分析），构建参与者画像，提升模型对个体偏好与群体动态的刻画精度。

3. 多学科方法融合

引入行为经济学理论（如前景理论）修正理性假设，结合复杂网络分析工具研究委员关系对策略选择的影响，建立更贴近现实的混合评估模型。

4. 决策支持应用

拓展至企业激励机制设计、公共资源分配等博弈场景，通过量化规则参数（如奖励总额、触发阈值）对参与度的影响，为管理者提供策略优化建议。

7 参考文献

参考文献

[1] 张良桥. 进化稳定均衡与纳什均衡——兼谈进化博弈理论的发展. 经济科学, (03):103–111, 2001.

[2] 张志雄, 张静之, and 赵春锋. 微信红包随机金额生成算法模拟及应用. 教育教学论坛, (10):51–53, 2019.

[3] 郭生良. 能谱的蒙特卡罗计算方法探讨与模拟软件设计. Master's thesis, 成都理工大学, 2008.

8 附录

8.1 game_20.py

```
import random
from red_pocket import red_envelope  # 从 red_pocket 模块导入
                                     red_envelope 函数
from engagement_and_excitement import calculate_engagement_index
from engagement_and_excitement import calculate_excited_index
random.seed(252)
class RedPacketGame1:
    def __init__(self, total_members=21, absent_member = 20):
        self.total_members = total_members  # 总委员人数
        self.participants = [i for i in range(total_members)]  # 委员
                                               编号
        self.red_packet_data = {i: {"received": 0, "sent": 0} for i
                                               in range(total_members)}
                                               # 委员的红包获得和发放统计
        self.members_in_game = []  # 记录每一轮参与游戏的委员
        self.sent_members = set()  # 记录已经发过红包的委员
        self.received_members = set()  # 记录已经抢到红包的委员
        self.flag = 0 # 游戏进行的轮数
        self.absent_member = absent_member # 缺席的委员编号
        self.absent_flag = 0
        self.envelopes = []

    def start_round(self, absent_member, last_winner):
        """
        开始一轮红包分配
        :param absent_member: 缺席委员
        :param last_winner: 上一轮最大红包获得者，如果所有人抢到红包
                                               则由该人发红包
        """
        self.members_in_game = [i for i in self.participants if i !=
                                               absent_member]  # 移除缺席
                                               委员
```

```
# 确定发红包者
if last_winner is not None:  # 如果上一轮所有人都抢到红包，则由最大红包获得者发红包
    first_member = last_winner
else:
    # 第一次红包发放由编号为1的委员发放
    first_member = 0  # 编号为1的委员在列表中的索引是0

# 将缺席的委员加入队列，确保他不在第一个位置
self.members_in_game.remove(first_member)
self.members_in_game.append(absent_member)
random.shuffle(self.members_in_game)

# 确保缺席的委员不在第一个位置
non_participant = self.members_in_game[-1]  # 剩下的1个没有抢到红包的人(设置为最后一个)
self.members_in_game = [first_member] + self.members_in_game[:-1]

# 记录这一轮发红包的委员编号并增加发红包次数
self.red_packet_data[first_member]["sent"] += 1
self.sent_members.add(first_member)

# 红包金额：100元，总人数：20人
total_amount = 100  # 总金额
num_people = len(self.members_in_game)  # 参与抢红包的总人数，第一轮选出1个第一个，剩下19个

# 使用从 red_pocket 模块导入的 red_envelope 函数来进行红包分配
self.envelopes, max_person = red_envelope(total_amount, num_people)  # 红包金额分配

# 发放红包给每个参与的委员
for i in range(num_people):  # 一共发20个红包
    # 将红包金额分发给对应的委员，并更新其收到红包次数
```

```
            current_member = self.members_in_game[i]  # 获取当前参与
                                                      的委员
            self.red_packet_data[current_member]["received"] += 1
            self.received_members.add(current_member)

        # 判断下一轮发红包的人
        if non_participant == self.absent_member:
            # 如果缺席的是编号为20的委员，则下轮由最大红包获得者发红
                                                      包
            first_member = self.members_in_game[max_person-1]
            self.absent_flag += 1
        else:
            # 否则，由未抢到委员发红包
            first_member = non_participant

            # 检查是否所有编号从1到20的委员都已经发过红包
        all_members_sent = all(self.red_packet_data[i]["sent"] > 0
                                            for i in range(len(self.
                                            members_in_game)))  # 检查
                                            编号1到20的委员是否都发过
                                            红包
        self.flag += 1
        # print(f"第{self.flag}轮:{all_members_sent}")
        # for i in range(20):
        #     print(self.red_packet_data[i]["sent"])

        if all_members_sent:
            # print("所有编号1到20的委员都发过红包，游戏结束！")
            # print(f"活动轮数为:{self.flag}")
            return True, first_member  # 游戏结束，返回最大红包获得者
        return False, first_member  # 游戏继续

    def simulate_game(self):
        """
        模拟游戏的多轮，计算每个委员的发红包和抢红包次数
        """
        absent_member = self.absent_member  # 初始时无缺席委员
        last_winner = None  # 初始时没有最大红包获得者
```

```
        # 使用while循环使游戏能够无限进行直到结束条件成立
        while True:
            game_ended, last_winner = self.start_round(absent_member,
                                                       last_winner)  # 传递
                                                      上一轮的最大红包获得者
            if game_ended:  # 如果游戏结束
                break

    def get_summary(self):
        """
        返回各委员的红包获得和发放统计
        """
        return self.red_packet_data


# 创建游戏实例
game = RedPacketGame1()

# 模拟游戏
game.simulate_game()

# 输出委员抢红包次数和发红包次数
for member in range(1, 22):  # 输出1到21号委员的红包统计
    print(
        f"委员{member}抢到红包次数: {game.red_packet_data[member - 1
                                                ]['received']}, 发红包次数
                                                : {game.red_packet_data[
                                                member - 1]['sent']}")

# 计算并输出活跃程度
engagement_index = calculate_engagement_index(game.participants, game
                                    .members_in_game, game.flag, game.
                                    red_packet_data)
print(f"活跃程度 (Coverage Efficiency Index): {engagement_index}")
#
# 计算并输出有趣程度
excitement_index = calculate_excited_index(game.envelopes, game.flag,
```

```
                                        game.members_in_game, game.
                                    absent_flag)
print(f"有趣程度 (Excitement Index): {excitement_index}")
```

8.2 game_21.py

```
import random
from red_pocket import red_envelope  # 从 red_pocket 模块导入
                                    red_envelope 函数
from engagement_and_excitement import calculate_engagement_index
random.seed(252)
class RedPacketGame2:
    def __init__(self, total_members=21, absent_member = None):
        self.total_members = total_members  # 总委员人数
        self.participants = [i for i in range(total_members)]  # 委员
                                            编号
        self.red_packet_data = {i: {"received": 0, "sent": 0} for i
                                            in range(total_members)}
                                            # 委员的红包获得和发放统计
        self.members_in_game = []  # 记录每一轮参与游戏的委员
        self.sent_members = set()  # 记录已经发过红包的委员
        self.received_members = set()  # 记录已经抢到红包的委员
        self.flag = 0 # 游戏进行的轮数
        self.absent_member = absent_member # 缺席的委员编号
        self.absent_flag = 0
        self.envelopes = []

    def start_round(self, last_winner):
        """
        开始一轮红包分配
        :param last_winner: 上一轮最大红包获得者，如果所有人抢到红包
                                            则由该人发红包
        """
        self.members_in_game = [i for i in self.participants]  # 所有
                                            委员全都在场

        # 确定发红包者
```

```
if last_winner is not None:  # 如果上一轮所有人都抢到红包，则
                                由最大红包获得者发红包
    first_member = last_winner
else:
    # 第一次红包发放由编号为1的委员发放
    first_member = 0  # 编号为1的委员在列表中的索引是0

# 随机打乱顺序
random.shuffle(self.members_in_game)

# 记录这一轮发红包的委员编号并增加发红包次数
self.red_packet_data[first_member]["sent"] += 1
self.sent_members.add(first_member)

# 红包金额：100元，总人数：21人
total_amount = 100  # 总金额
num_people = len(self.members_in_game)  # 参与抢红包的总人数

# 使用从 red_pocket 模块导入的 red_envelope 函数来进行红包分
                                配
money, max_person = red_envelope(total_amount, num_people)  #
                                红包金额分配

# 发放红包给每个参与的委员
for i in range(num_people):  # 一共发21个红包
    # 将红包金额分发给对应的委员，并更新其收到红包次数
    current_member = self.members_in_game[i]  # 获取当前参与
                                的委员
    self.red_packet_data[current_member]["received"] += 1
    self.received_members.add(current_member)

    # 下轮由最大红包获得者发红包
    first_member = self.members_in_game[max_person-1]

    # 检查是否所有编号从1到21的委员都已经发过红包
all_members_sent = all(self.red_packet_data[i]["sent"] > 0
                                for i in range(len(self.
                                members_in_game)))  # 检查
```

```
                                                编号1到20的委员是否都发过
                                                红包
            self.flag += 1
            # print(f"第{self.flag}轮:{all_members_sent}")
            # for i in range(21):
            #     print(self.red_packet_data[i]["sent"])

            if all_members_sent:
                # print("所有编号1到21的委员都发过红包，游戏结束！")
                return True, first_member  # 游戏结束，返回最大红包获得者
            return False, first_member  # 游戏继续

        def simulate_game(self):
            """
            模拟游戏的多轮，计算每个委员的发红包和抢红包次数
            """
            # 初始时无缺席委员
            last_winner = None  # 初始时没有最大红包获得者

            # 使用while循环使游戏能够无限进行直到结束条件成立
            while True:
                game_ended, last_winner = self.start_round(last_winner)
                                                # 传递上一轮的最大红包
                                                获得者
                if game_ended:  # 如果游戏结束
                    break

        def get_summary(self):
            """
            返回各委员的红包获得和发放统计
            """
            return self.red_packet_data


    # 创建游戏实例
    game = RedPacketGame2()

    # 模拟游戏
```

```
game.simulate_game()

# 输出委员抢红包次数和发红包次数
# for member in range(1, 22):  # 输出1到21号委员的红包统计
#     print(f"委员{member}抢到红包次数: {game.red_packet_data[member
                                    - 1]['received']}, 发红包次数: {
                                    game.red_packet_data[member - 1]['
                                    sent']}")

# 计算并输出活跃程度
engagement_index = calculate_engagement_index(game.participants, game
                                    .members_in_game, game.flag, game.
                                    red_packet_data)
print(f"活跃程度 (Coverage Efficiency Index): {engagement_index}")
```

8.3 engagement_and_excitement.py

```
import numpy as np
def calculate_engagement_index(participants, members_in_game, N,
                                    red_packet_data):
    """
    计算活动的活跃程度
    :param members_in_game: 在场委员编号
           N: 活动轮数
           participants: 参与委员编号
           red_packet_data: 委员的红包获得和发放统计
    :return: 活跃程度得分: activity_scores: [0,100]
    """
    S = len(participants)
    S1 = len(members_in_game)
    T = sum(1 for member in members_in_game if red_packet_data[member
                                        ]["sent"] == 0)
    activity_scores = S1 / N * np.log(1 + S / (T + 1))
    worst_score = 0
    best_score = S1 / S1 * np.log(1 + S / (S - S1 + 1))
    # activity_scores = activity_scores * 100 / (best_score -
                                        worst_score)
```

```
    activity_scores = activity_scores * 100 / 2.472833962686653
    return activity_scores

def calculate_excited_index(envelopes, N, members_in_game,
                                    absent_flag):
    """
    计算活动的有趣程度
    :param envelopes: 红包金额
           N: 活动轮数
           members_in_game: 在场委员编号
           absent_flag: 规则2.b触发次数
    :return: 有趣程度得分: excitement_score: [0,100]
    """
    S1 = len(members_in_game)
    S_i = np.var(envelopes)
    avg_amount = 100 / S1
    worst_score = 0
    best_score = (1 - avg_amount / 100) * avg_amount * avg_amount +
                                        avg_amount * (100 - avg_amount
                                        ) * (100 - avg_amount) / 100 +
                                         5 # 最大方差公式
    excitement_score = S_i + 5 * absent_flag / N
    # excitement_score = excitement_score * 100 / (best_score -
                                        worst_score)
    excitement_score = excitement_score * 100 / 37.02333051282051
    return excitement_score
```

8.4 red_pocket.py

```
import random


def red_envelope(money, person):
    # 计算每个红包的均值
    avg = money / person
    envelopes = []
    total_allocated = 0
```

```
    # 为每个人分配红包
    for i in range(person - 1):
        # 随机生成一个接近均值的小红包（避免某些红包金额过大）
        red_packet = round(random.uniform(avg / 2, avg * 1.5), 2)
        envelopes.append(red_packet)
        total_allocated += red_packet

    # 最后一个红包金额调整为剩余的金额
    last_packet = round(money - total_allocated, 2)
    envelopes.append(last_packet)

    # 找出获得最大红包的人
    max_packet = max(envelopes)
    max_person = envelopes.index(max_packet) + 1  # 人的编号从1开始

    return envelopes, max_person
```

8.5 random_seed.py

```
import random
from red_pocket import red_envelope  # 导入red_envelope函数
from engagement_and_excitement import calculate_engagement_index
from game_20 import RedPacketGame1  # 导入RedPacketGame类
from game_21 import RedPacketGame2
from tqdm import tqdm  # 导入tqdm库来显示进度条
from engagement_and_excitement import calculate_excited_index
import numpy as np
import matplotlib.pyplot as plt

def simulate_with_random_seeds():
    max_engagement_index = -1  # 用于记录最大活跃度
    # max_excitement_index = -1
    # best_excitement_seed = None  # 用于记录对应最大活跃度的种子
    best_engagement_seed = None
    engagement_indices1 = []  # 用于保存 RedPacketGame1 的
                                    engagement_index 数据
```

```
engagement_indices2 = []  # 用于保存 RedPacketGame2 的
                                    engagement_index 数据
# 使用tqdm来添加进度条，遍历1到100000的随机种子
for seed in tqdm(range(1, 100001), desc="Simulating with seeds",
                                    unit="seed"):
    random.seed(seed)  # 设置随机种子

    # 创建游戏实例
    game1 = RedPacketGame1()
    game2 = RedPacketGame2()
    # 模拟游戏
    game1.simulate_game()
    game2.simulate_game()
    # 计算活跃度
    engagement_index1 = calculate_engagement_index(game1.
                                        participants, game1.
                                        members_in_game, game1.
                                        flag, game1.
                                        red_packet_data)
    # excitement_index = calculate_excited_index(game1.envelopes,
                                         game1.flag, game1.
                                        members_in_game, game1.
                                        absent_flag)
    engagement_index2 = calculate_engagement_index(game2.
                                        participants, game2.
                                        members_in_game, game2.
                                        flag,
                                                    game2.




    engagement_indices1.append(engagement_index1)
    engagement_indices2.append(engagement_index2)
    # 如果当前活跃度大于最大活跃度，则更新最大活跃度和对应种子
    if engagement_index1 > max_engagement_index:
        max_engagement_index = engagement_index1
        best_engagement_seed = seed
```

```
    # if excitement_index > max_excitement_index:
    #     max_excitement_index = excitement_index
    #     best_excitement_seed = seed
# 输出最大活跃度和对应的种子
# 计算平均值
avg_eni1 = np.mean(engagement_indices1)
avg_eni2 = np.mean(engagement_indices2)
print(f"最大活跃度为: {max_engagement_index}")
# print(f"最大有趣度为: {max_excitement_index}")
print(f"对应活跃度的最佳随机种子为: {best_engagement_seed}")
# print(f"对应有趣度的最佳随机种子为: {best_excitement_seed}")
print(f"平均活跃度1为: {avg_eni1}")
print(f"平均活跃度2为: {avg_eni2}")

# 设置图形的背景颜色
plt.style.use('seaborn-whitegrid')  # 使用网格背景风格

# 创建一个 12x6 英寸的图形
plt.figure(figsize=(12, 6))

# 第一张图: RedPacketGame1的engagement_index
plt.subplot(1, 2, 1)
plt.plot(range(1, 100001), engagement_indices1, color='#FF6347',
                                    linewidth=1.5, label='
                                    Engagement Index 1')  # 使用番
                                    茄红
plt.axhline(y=avg_eni1, color='#4682B4', linestyle='--',
                                    linewidth=2, label=f'Average:
                                    {avg_eni1:.2f}')  # 使用钢青色
plt.xlabel("Random Seed", fontsize=12, color='#333333')  # 设置标
                                    签字体和颜色
plt.ylabel("Engagement Index", fontsize=12, color='#333333')
plt.title("Engagement Index for RedPacketGame1", fontsize=14,
                                    fontweight='bold', color='#
                                    333333')
plt.legend()

# 第二张图: RedPacketGame2的engagement_index
```

```
    plt.subplot(1, 2, 2)
    plt.plot(range(1, 100001), engagement_indices2, color='#32CD32',
                                        linewidth=1.5,
             label='Engagement Index 2')  # 使用绿宝石绿
    plt.axhline(y=avg_eni2, color='#FFD700', linestyle='--',
                                        linewidth=2, label=f'Average:
                                        {avg_eni2:.2f}')  # 使用金色
    plt.xlabel("Random Seed", fontsize=12, color='#333333')
    plt.ylabel("Engagement Index", fontsize=12, color='#333333')
    plt.title("Engagement Index for RedPacketGame2", fontsize=14,
                                        fontweight='bold', color='#
                                        333333')
    plt.legend()

    # 调整图形布局
    plt.tight_layout()
    # 保存图像
    plt.tight_layout()
    plt.savefig('engagement_indices_comparison.png')

    # 显示图像
    plt.show()
if __name__ == "__main__":
    simulate_with_random_seeds()
```

8.6 improved_rule.py

```
import random
from red_pocket import red_envelope  # 从 red_pocket 模块导入
                                    red_envelope 函数
from engagement_and_excitement import calculate_engagement_index
from engagement_and_excitement import calculate_excited_index
# random.seed(252)
class RedPacketGame3:
    def __init__(self, total_members=21, absent_member=[19, 20]):  #
                                        允许多个缺席委员
        self.total_members = total_members  # 总委员人数
```

```
        self.participants = [i for i in range(total_members)]  # 委员
                                                编号
        self.red_packet_data = {i: {"received": 0, "sent": 0} for i
                                                in range(total_members)}
                                                # 委员的红包获得和发放统计
        self.members_in_game = []  # 记录每一轮参与游戏的委员
        self.sent_members = set()  # 记录已经发过红包的委员
        self.received_members = set()  # 记录已经抢到红包的委员
        self.flag = 0  # 游戏进行的轮数
        self.absent_member = absent_member  # 缺席的委员编号（列表）
        self.absent_flag = 0
        self.envelopes = []

    def start_round(self, absent_member, last_winner):
        """
        开始一轮红包分配
        :param absent_member: 缺席委员
        :param last_winner: 上一轮最大红包获得者，如果所有人抢到红包
                                                则由该人发红包
        """
        self.members_in_game = [i for i in self.participants if i not
                                                 in absent_member]  # 移除
                                                多个缺席委员


        # 确定发红包者
        if last_winner is not None:  # 如果上一轮所有人都抢到红包，则
                                                由最大红包获得者发红包
            first_member = last_winner
        else:
            # 第一次红包发放由编号为1的委员发放
            first_member = 0  # 编号为1的委员在列表中的索引是0
        self.members_in_game.remove(first_member)
        for member in absent_member:
            self.members_in_game.append(member)
        random.shuffle(self.members_in_game)
        non_participant = self.members_in_game[-1]

        self.members_in_game = [first_member] + self.members_in_game[
```

```
                                        :-1]

        # 记录这一轮发红包的委员编号并增加发红包次数
        self.red_packet_data[first_member]["sent"] += 1
        self.sent_members.add(first_member)

        # 红包金额: 100元
        total_amount = 100  # 总金额
        num_people = len(self.members_in_game)  # 参与抢红包的总人数

        # 使用从 red_pocket 模块导入的 red_envelope 函数来进行红包分
                                        配
        self.envelopes, max_person = red_envelope(total_amount,
                                        num_people)  # 红包金额分
                                        配

        # 发放红包给每个参与的委员
        for i in range(num_people):  # 一共发20个红包
            # 将红包金额分发给对应的委员, 并更新其收到红包次数
            current_member = self.members_in_game[i]  # 获取当前参与
                                            的委员
            self.red_packet_data[current_member]["received"] += 1
            self.received_members.add(current_member)

        # 判断下一轮发红包的人
        if non_participant in self.absent_member:  # 如果
                                        non_participant 和 self.
                                        absent_member 完全相同
            # 如果 self.members_in_game[max_person - 1] 在缺席列表
                                            中, 选择一个没有发过红
                                            包的成员作为 next
                                            first_member
            if self.members_in_game[max_person - 1] in self.
                                            absent_member:
                # 从没有发过红包且不是缺席成员的成员中选择
                available_members = [i for i in self.members_in_game
                                                if
                                    self.red_packet_data[i]["sent"]
```

```
=
=

0

and

i

not

in

self
.
absent_member
]
```

```
            if available_members:
                first_member = random.choice(available_members)
                                                    # 随机选择一个
                                                    没有发过红包且
                                                    不在缺席名单中
                                                    的成员
            else:
                # 如果没有找到未发红包且不在缺席名单中的成员（理
                                                    论上不会发
                                                    生），可以做一
                                                    些默认处理
                first_member = self.members_in_game[0]
        else:
            first_member = self.members_in_game[max_person - 1]
    else:
        first_member = non_participant

    # 检查是否所有编号从1到20的委员都已经发过红包
    all_members_sent = all(self.red_packet_data[i]["sent"] > 0
                                        for i in range(len(self.
```

```
                                                members_in_game)-1))  # 检
                                                查编号1到20的委员是否都发
                                                过红包
            self.flag += 1

            if all_members_sent:
                return True, first_member  # 游戏结束，返回最大红包获得者
            return False, first_member  # 游戏继续


        def simulate_game(self):
            """
            模拟游戏的多轮，计算每个委员的发红包和抢红包次数
            """
            absent_member = self.absent_member  # 初始时无缺席委员
            last_winner = None  # 初始时没有最大红包获得者

            # 使用while循环使游戏能够无限进行直到结束条件成立
            while True:
                game_ended, last_winner = self.start_round(absent_member,
                                                           last_winner)  # 传递
                                                           上一轮的最大红包获得者
                if game_ended:  # 如果游戏结束
                    break

        def get_summary(self):
            """
            返回各委员的红包获得和发放统计
            """
            return self.red_packet_data


    # 创建游戏实例
    game = RedPacketGame3()

    # 模拟游戏
    game.simulate_game()
```

```
# 输出委员抢红包次数和发红包次数
for member in range(1, 22):  # 输出1到21号委员的红包统计
    print(
        f"委员{member}抢到红包次数: {game.red_packet_data[member - 1
                                        ]['received']}, 发红包次数
                                        : {game.red_packet_data[
                                        member - 1]['sent']}")

# 计算并输出活跃程度
engagement_index = calculate_engagement_index(game.participants, game
                                    .members_in_game, game.flag, game.
                                    red_packet_data)
print(f"活跃程度 (Coverage Efficiency Index): {engagement_index}")
#
# 计算并输出有趣程度
excitement_index = calculate_excited_index(game.envelopes, game.flag,
                                     game.members_in_game, game.
                                    absent_flag)
print(f"有趣程度 (Excitement Index): {excitement_index}")
```