评委一评分，签名及备注	队号： 20035	评委三评分，签名及备注
评委二评分，签名及备注	选题： A	评委四评分，签名及备注

2048 游戏玩法

摘要

继游戏“FlappyBird”下架后，一款名为“2048”的游戏受到了大众的追捧，很多网友称其玩法看似简单，但真正玩起来着实不易。为了解决这一让众多网友头疼的“学霸”游戏中存在的相关问题，建立了以下模型。

针对问题一，从最根本的组合递推方式入手，通过建立树状图得到了 2048 的基本步骤，根据 *mini-max* 算法原理，并结合玩法步骤，建立了该游戏的通用模型。在计算方格移动次数时，考虑到最后一个数字出现的随机性，采用倒推的方式，得到了完成 2048 的步数。最后，根据 *Alpha—beta* 剪枝算法，对随机游戏者，搜集其实验数据进行分析，建立游戏者达到最大数字与所用步数以及成功率的数学模型，并与上述通用模型做对比分析，从而验证该通用模型是有效的。

针对问题二，运用递归算法和数学归纳法建立最大值模型，讨论在最后值 M 不同取值时最大数 Q 的值，得到 Q 的最大值为2^{17}。假设最大值可以达到2^{18}，通过计算，假设不成立，所以“2048”游戏能达到的最大值为2^{17}。运用所建的最大值模型，找到了递推规律，在$N \times N$的模式中，假设所能达到的最大值为2^{N^2+1}，通过验证，假设成立，因此最大值为2^{N^2+1}。

关键字： *Mini-max* **算法** *Alpha—beta* **剪枝算法** **递归算法**

“2048” 游戏玩法

1. 问题重述

“2048”是一款益智游戏，其游戏规则：每次控制所有方块向同一个方向运动，两个相同数字的方块撞在一起后合并成为的和，每次操作之后会在空白的方格出随机产生数字2或4，最终得到一个“2048”的方块就算通关。如果16个格子全部填满并且相邻的格子内的数字都不相同，则无法移动格子，最终游戏结束。

通过建立模型，解决以下问题：

1、 如何达到2048，给出一个通用模型，并采用完成游戏所需移动次数和成功概率来验证模型的有效性；

2、得到2048后，游戏还可以继续玩，那么最大能达到的数值是多少？如果方格扩展到$N \times N$个，能达到的最大数是多少？

2. 问题分析

对于问题一，很多网友自称“一旦玩上它就根本停不下来”，基于这点，通过搜集网友的试验次数，计算发现，随意的玩法很难达到2048，因此需要建立一个通用的模型。由于随意移动方块时，若两个相同数所组合的数越大，越难组合到一起，通过实验发现，这个游戏的基本思想是递归生成，这样固定住了最大数值，再通过递推累加的方法得到新的最大数。根据这一想法通过 *mini-max* 算法建立模型，并且对于所给出的模型，假设了移动的次数。验证该模型的有效性，玩该游戏的人中随机抽取100人进行试验，计算成功率，因此，通过多次试验得出结论。验证移动次数与模型有效性时，建立移动次数与每次游戏结束时得到最大数的散点图，得到回归方程，当组合出2048时需要多少移动次数，是否与假设相似，如与假设相似，则验证了模型的有效性。

对于问题二，根据递归生成的模型，按照问题一给出的通用模型，将最大数放在一个角，其余成蛇型一次递减，方格中倒数第二个数为4且最后一个数系统生成4时，方格才可以移动，游戏继续进行，通过递推累加得到最后一位数，而剩下的15个方格中按照递推累加法，得到的最后一位数即最大的数字，易得第一次计算的数为最大，因此得到了“2048”的最大组合值，可以得出此游戏并不是可以无尽的玩下去。对于$N \times N$的方格，由数学归纳法，总结出$N \times N$方格的规律模型，得到问题的结论。

3. 模型假设

1. 假设每次移动都为有效移动；
2. 假设每次组合只能组合一对；
3. 假设系统随机出现数的位置与试验结果无关；
4. 假设游戏所用步数与玩家自身的主观因素无关。

4. 符号说明

M	最后一个系统随机出的数
Q	方格中最大生成数
H	结束时组合的最大数
P_1	结束时最大值出现的数字的次数
Y	移动次数
P_2	移动步数出现的次数
G	游戏在理想模型下所能达到的所有有限集
k	后继函数
m	游戏状态
Alpha	代替问题一中的2048
N	每组成功数
y	每次游戏结束时得到的最大数
x	达到的最大数
z	所用平均步数
i	次幂数
n	格子数

5. 模型建立及求解

5.1 问题一

5.1.1 背景

“2048”是最近推出的一款游戏，其游戏规则如下：

- 开始时方格内随机出现两个数字，仅可能为 2 或 4；
- 玩家可以选择上下左右四个方向，若棋盘内的数字出现位移或合并，视为有效移动；
- 玩家选择的方向上若有相同的数字则合并，每次有效移动可以同时合并，但不可以连续合并；
- 合并所得的所有新生成数字想加即为该步的有效得分；

- 玩家选择的方向行或列前方有空格则出现位移；
- 每有效移动一步，棋盘的空位(无数字处)随机出现一个数字(依然可能为 2 或 4)；
- 棋盘被数字填满，无法进行有效移动，判负，游戏结束；
- 棋盘上出现 2048，判胜。

通过搜集数据[1]，该网友一共玩了 4246 次，每次结束时出现最大值的次数（如表 1）和结束时移动方块出现的次数（如表 2）：

表 1 出现最大值的次数

H	2^3	2^4	2^5	2^6	2^7	2^8	2^9
P_1	1	7	218	1412	2173	433	2

表 2 移动方块的次数与出现的次数

Y	0~50	50~100	100~150	150~200	200~250
P_2	2	362	1458	1556	551
Y	250~300	300~350	350~400	400~450	450~500
P_2	247	64	4	1	1

根据搜集数据得出，表 1 中最小值为 8，最大值能达到 1024，经计算数学期望约为 101；在表 2 中最小值是 41，最大值是 468，中位值 161，数学期望为 163。

因此，随意的玩法要到达 2048 比较难，而且用的步骤多，下面建立通用玩法，增大达到 2048 的成功率并减少移动次数。

5.1.2 *Mini-max* 算法模型建立

Mini-max 算法（如图 1）是一个回归的算法，这意味着它毕竟搜索进行到游戏树的的叶子节点，取得游戏的结局值，再将这些数值按 *Mini-max* 过程后向传递上去，直到获得所有内部节点的 *Mini-max* 值[2]。

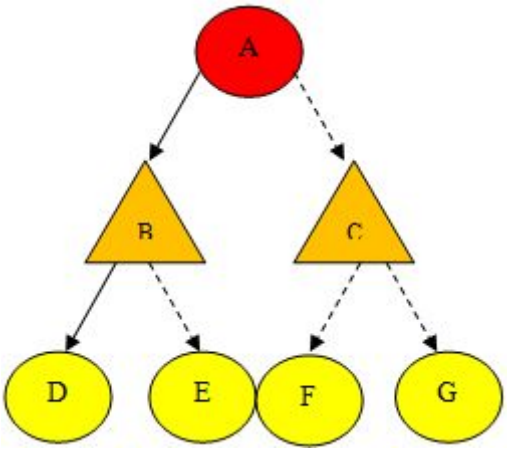

图 1　*Mini-max* 算法树状图

为建立达到 2048 的通用模型，作如下定义：

一个区间赋值游戏为一个四元方程组（*G*，*m*，*k*，*f*），其中 *G* 为游戏在理想模型下所能达到的所有有限集，$m_1 \in G$是游戏的初始状态，*k* 为后继函数，$f: G \to [0,1]$是评价函数。$M' \in s(m)$表示状态m'可由状态 *m*

经过合理的步骤完成，通常建立的游戏模型应同时满足以下两个条件：

(1)不存在来回循环的状态。

(2)当游戏结束时，$ev(m) \in [0,1]$

其中，0 和 1 分别代表游戏运行者完成后即结束和完成后仍继续闯关下去的情况，游戏结局为 0 时，成为 minimizer，结局为 1 时称为 maximizer，0 和 1 之间的数值代表中间结果，即完不成的目标的情况。$Ev(m) = max$代表完成后仍继续闯关下去的情况，$ev(m) = min$代表完成后即结束的情况。递归定义函数：$ev_\alpha: G \to [0,1]$如下：

$$ev_\alpha(m) = \begin{cases} ev(m) & ev(m) \in [0,1] \\ \max_{m' \in s(m')} ev_\alpha(m') & ev(m) = max \\ \min_{m' \in s(m')} ev_\alpha(m') & ev(m) = min \end{cases}$$

由上式可知，max 的值等于运行结果的最大值，min 的值等于运行结果的最小值，故得到 *mini-max* 算法如下：

If　$ev(m) \in [0,1]$　return　$ev(m)$

Else　if　$ev(m) = max$　return　$\max_{m' \in s(m')} ev_\alpha(m')$

Else　if　$ev(m) = min$　return　$\min_{m' \in s(m')} ev_\alpha(m')$

5.1.3 *Alpha—beta* 剪枝算法

单一的 *Mini-max* 算法得到通用模型比较复杂，由于所要构造数字的数目较大，因此需要的步骤较多，限制了其移动次数。

利用 *Alpha—beta* 剪枝算法的工作原理（如图 2），从最大数字开始，详述使

用 *Alpha—beta* 的每一步骤：

(1)最大数字被Alpha代替，最小数字为 2。

(2)画出以Alpha/2的数字为节点。

(3)构造第三层的节点，同上。
(4)直到构造的节点数为 2 为止。
(5)剩余的结构图按照上述四个步骤完成。
对于问题一，按照 *Alpha—beta* 剪枝算法示意图步骤画出 2048 的构成图。

即想合成2048，先考虑 1024 所在的树状图，要形成 1024，首先要合成两个 512 而且要保证这两个数字相邻。同理合成每个 512 都需要两个相邻的 256，依此类推，直到分支到最小数字 2 为止（如图 2），这样就可以快速的计算出所需步骤数。

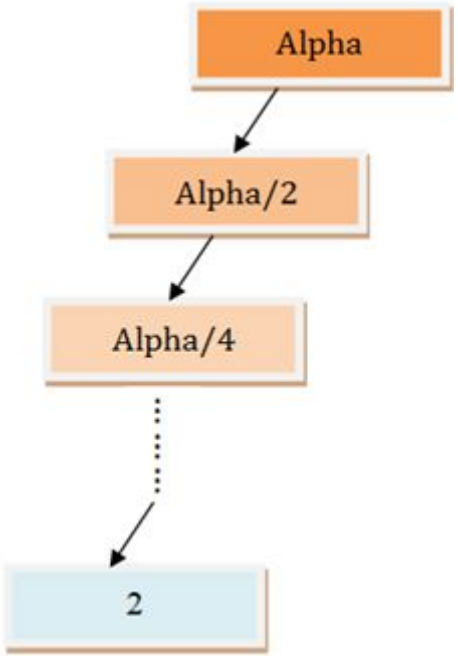

图 2　*Alpha—beta* 剪枝算法示意图

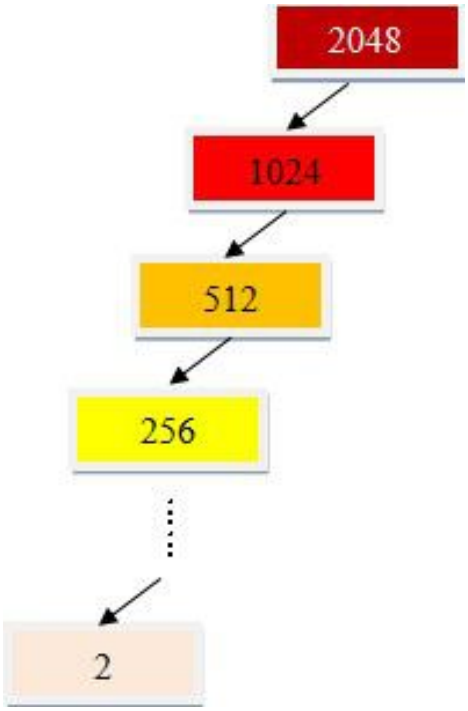

图 3　2048 推导图

当构造出上图后按数字大小依次(从高往低)将 1024, 512,256,128 四个数排在边缘，这四个数不动，当数字越来越大，就向角落那个大数字堆叠过去，这样可以尽量的减少格子的使用率。需要注意的是：如果将数字定在右下角，只能进行向右或者向下移动，才能确保大数不移动，因此首先要做的就是将大数排成一行或一列，将四个格子填满，顺序依然是由大到小向外排列。当一行或者一列补满后，就可以获得向左、向下、向右移动。如果数字越来越多，需要做的是将数字蛇形排列，将第一行的尾数对接在第二行的首元，这样就可以保证格子最节省，而且也最安全的将数字叠高。如图 4 所示就是完美排列的情况，合并成 2048 的最完美的最后一步。

2			
4	4		2
8	16	32	64
1024	512	256	128

图 4 合成 2048

格局评价:分数高的，“好”格局是容易走向成功的关键；反之，分数低的，由于“坏”格局引起的。

单调性:单调性是指方块中的数字从上到下，从左到右均遵循从递增或递减。一般来说，越单调的格局越好，如图 5 所示：

8	32	64	512
4	8	16	256
2	4	8	32
		4	8

图 5 单调格局

空格数:空格越少对玩家来说越不利，因为挪动数字的空间越少，可供玩家选择的机会就越少。

5.1.3 方块移动次数模型建立

游戏结束即出现 2048 时，假设是随机出现的数字都是 2，并且每次能和成且只能合成一个数 让它依次生成：

4、8、16、32、64、128、256、512、1024、2048

- 第一步生成 4 需要移动 1 次；（2 个 2 组成）
- 第二步生成 8 需要移动 2 次；（另一个 4 由 2 个 2 分别相加）
- 第三步生成 16 需要移动 4 次；（另一个 8 由 4 个 2 分别相加）
- 第四步生成 32 需要移动 8 次；（另一个 16 由 8 个 2 分别相加）
- 第五步生成 64 需要移动 16 次；（另一个 32 由 16 个 2 分别相加）
- 第六步生成 128 需要移动 32 次；（另一个 64 由 32 个 2 分别相加）
- 第七步生成 256 需要移动 64 次；（另一个 128 由 64 个 2 分别相加）
- 第八步生成 512 需要移动 128 次；（另一个 256 由128个 2 分别相加）
- 第九步生成 1024 需要移动 256 次；（另一个 512 由 256 个 2 分别相加）
- 第十步生成 2048 需要移动 512 次；（另一个 1024 由 512 个 2 分别相加）

所以，总共需要移动的次数为：

$$1+2+4+8+16+32+64+128+256+512=1023(次);$$

当随机出现的数字都为数字 4 时，并且每次能和成且只能合成一个数 让它依次生成：

8、16、32、64、128、256、512、1024、2048

- 第一步生成 8 需要 1 步；（2 个 4 组成）
- 第二步生成 16 需要 2 步；（另一个 8 由 2 个 4 组成）
- 第三步生成 32 需要 4 步；（另一个 16 由 4 个 4 组成）
- 第四步生成 64 需要 8 步；（另一个 32 由 8 个 4 组成）
- 第五步生成 128 需要 16 步；（另一个 64 由 16 个 4 组成）
- 第六步生成 256 需要 32 步；（另一个 128 由 32 个 4 组成）
- 第七步生成 512 需要 64 步；（另一个 256 由 64 个 4 组成）
- 第八步生成 1024 需要 128 步；（另一个 512 由 128 个 4 组成）
- 第九步生成 2048 需要 256 步；（另一个 1048 由 256 个 4 组成）

所以，总共需要移动的次数为：

$$1+2+4+8+16+32+64+128+256=511\text{（次）}$$

当 2 或 4 都出现时，不能确定出现的概率。但是经过搜索该游戏的源代码（见附录[1]），可以看出出现 2 的概率为 90%，出现 4 的概率为 10%。

- 产生 2048 需要移动 1 次（2 个 1024 相加）
- 产生 2 个 1024 需要移动 2 次（4 个 512 分别相加）
- 产生 4 个 512 需要移动 4 次（8 个 256 分别相加）
- 产生 8 个 256 需要移动 8 次（16 个 128 分别相加）
- 产生 16 个 128 需要移动 16 次（32 个 64 分别相加）
- 产生 32 个 64 需要移动 32 次（64 个 32 分别相加）
- 产生 64 个 32 需要移动 64 次（128 个 16 分别相加）
- 产生 128 个 16 需要移动 128 次（256 个 8 分别相加）
- 产生 256 个 8 需要移动 256 次（512 个 4 分别相加）
- 产生 512 个 4 需要移动 461 次（922 个 2 和 51 个 4 分别相加）

所以，总共需要移动的次数为：

$$1+2+4+8+16+32+64+128+256+461=972\text{（次）}$$

该游戏的模型空间大。比如，假设每步出现的都是 4，那么至少也需要 511 步才能出现 2048，每步至少有两种选择，所以要总共能供选择的数目超过2^{512}。

类似于象棋，围棋一样，基本上不可能有算法能保证 100%必胜。但是如果运用正确的方法策略，将会得到较高的获胜率。

5.1.4 **模型检验**:

成功率检验：

用建立的通用模型玩该游戏，猜想理论上依照这种方法成功率为 100%，为了验证模型的有效性及成功率是否接近 100%，在没有达到 2048 的人群中随机抽取 100 人，每人玩 1 次，则将得到 100 个数据，并记录他们每次游戏结束时得到的最大数（如图 6，详见附录[2]）。

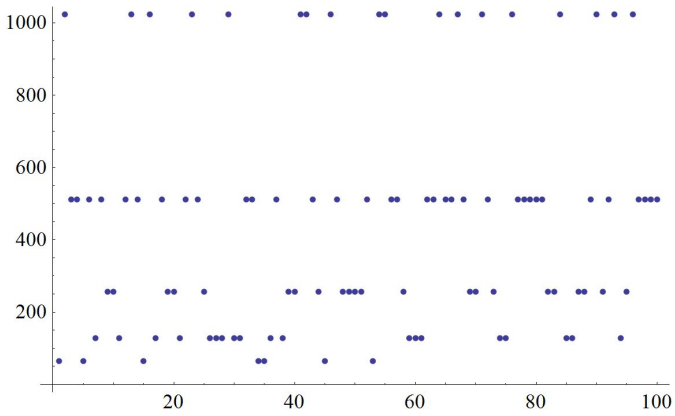

图 6　没有达到 2048 的记录

如图 6 所示，进行试验的人群中是没有找到规律，随意移动，游戏结束时达到的最大数很分散，并且没有达到 2048。为了验证模型的有效性，让这 100 个人按照给出的模型再进行一次，得到的数据，如图 7 所示：

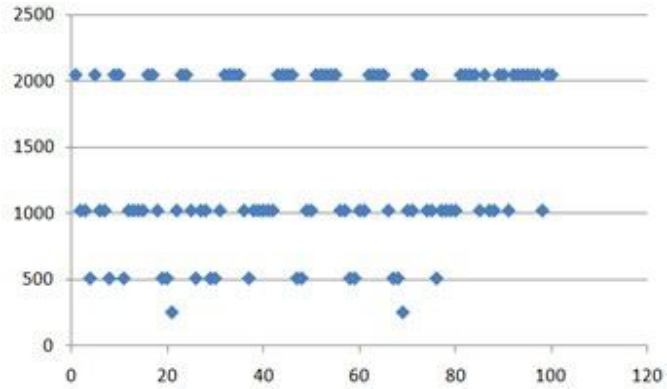

图 7　试验一次记录

由图 7 可以看出，最大数为 2048，最小数为 256，众数为 2048，其次，出现最多的数是 1024，成功率为 43%，尽管达到 2048 的人数已经多于得到其他数的人数，而且成功率得到了提升，但效果不明显，分析其原因是第一次试验该模型，试验的人群对模型并不熟悉，但是可以看出成绩已经优于试用模型之前。为了更好的展示建立模型的有效性，对随机抽取的 100 个人进行第二、第三次试验（如图 8 图 9）

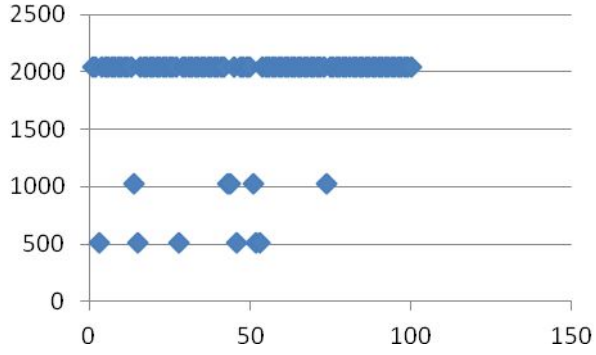

图 8　试验第二次记录

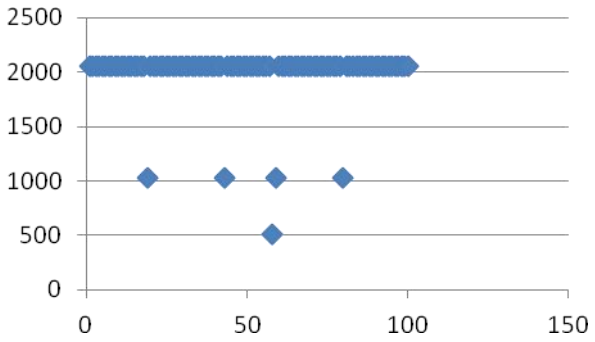

图 9 试验第三次记录

由图 8、图 9 可以看出，后两次的成绩明显高于第一次，第三次试验时成功率 95%，但是仍有一些误差，原因是出现新方块的位置不定，而且人的操作使移动的方向也具有不定性，但是多次训练可以看出，对模型越了解，越容易形成固定的移动模式，成功率越高。

方块移动次数检验：

为验证模型有效性，随机抽出 100 名游戏者玩“2048”游戏，每人各玩 5 次，记录他们所能达到的最大数字以及所用的步数的平均数，结果如表 3 所示：

表 3 到达的最大数与平均步数

x	64	128	256	512	1024
z	89	124	289	358	673

利用 *mathematics* 求其回归方程，并绘制散点图。得到达到最大数字和所用平均步数的回归方程为：$z = 72.4167 + 0.59018x$

模型散点图为(附录[3])：

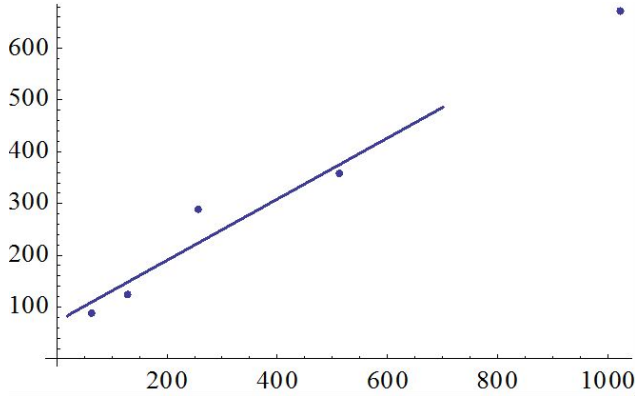

图 10 步数与最大数回归

当数字达到 2048 时， 带入上式回归方程，得到所用步数为 1281,而建立的通用模型当出现的都是 2 时，所用步数 1023（出现 4 或 2、4 都出现时，所用步数更少）小于实验所得数据 1281。这是由于游戏者自身因素的影响，如无效步数等导致了实验数据偏大。

综上所述：通过建立达到最大数和所用平均步数的关系模型验证之前所建立

的通用模型有效。

5.2 问题二

5.2.1 递归生成最大值

“2048”游戏的基本思想是递归生成，将最大的数2^n放在右下角，按照最大的数成S型依次排列。在图 11 中最大数为2^n，即“2048”游戏所能到达的最大数为2^{n+1}。

M	2^{n-14}	2^{n-13}	2^{n-12}
2^{n-8}	2^{n-9}	2^{n-10}	2^{n-11}
2^{n-7}	2^{n-6}	2^{n-5}	2^{n-4}
2^n	2^{n-1}	2^{n-2}	2^{n-3}

图 11　递归生成 2048

由于最后一个数 M 随机生成，分情况讨论：

1. 当$2^{n-14}=4$时，$M=2$停止游戏；$M=4$时有一致的方块可以进行组合，游戏继续，即所有的方块都可按照蛇型依次组合，则最终递推累加得：

$$Q=2^n+2^{n-1}+2^{n-2}+\cdots+2^{n-14}+M=\sum_{i=n-14}^{n}2^i+M$$

最终是两个2^n组合得到2^{n+1}，因此最大的组合数Q为2^{n+1}。

$$2^{n-14}=4 \qquad (1)$$

由（1）式可知此时$n=16$，即$2^{n+1}=2^{17}$，右下角2^n处变为2^{17}，则剩下的 15 个方格中最大的数为2^{n-1}。当2^{n-14} 取4且 M 为 4 时游戏可以继续进行，此时$2^{n-1}=2^{16}$，不可以再进行组合，因此最大能到达的数为2^{17}。

2. 当$2^{n-14}=2$时，M = 4游戏结束，M = 2游戏继续，最大组合出的数依然是2^{n+1}，此时：

$$2^{n-14}=2 \qquad (2)$$

由（2）式可知此时$n=15$，即$2^{n+1}=2^{16}$，此时最大可以达到的数为2^{16}。

3. 再假设2^{18}也能够合成，那么其上一步一定由两个2^{17}合成， 又假设其中一个2^{17}已合成，那么另一个2^{17}的上一步一定是由两个2^{16}合成。假设其中一个2^{16}已合成，另一个2^{16}的上一步一定由两个2^{15}合成……依此类推，但由于“2048”是由$4\times4=16$个方格组成，即空位只有 16 个。依此类推，方格中倒数第二个的空位就会变成2^3，而最后一个数 M 的最大值为 4，但此时已没有多余的空位来再生成 4，所以没有凑够两个 4 来合成2^3，所以也就没有两个2^3合成2^4……依此类推，所以没有两个2^{17}生成2^{18}。但如果最大值为2^{17}就有空位再生成2^2，令2^2凑够两个2^3……依此类推，所以能够合成2^{17}，而不能合成2^{18}。故最大能达到的数值为2^{17}（如图 12）。

16	32	4096	8192
8	64	2048	16384
4	128	1024	32768
4	256	512	65536

图 12　递推生成 131072

综上所述，“2048”游戏不可以无止境的玩下去，在4×4的情况下，将“2048”

模拟成图 11 式样，其中，已知左上角的格必须是 4，才能保证所得到的数值最大，所以由（1）式得到$n = 16$，在这种情况下是能够全合并的，因此最大可以合并得到2^{17}，由于出现的数（2 或 4）和出现的位置是随机的，该模型提供的是理想上能达到的最大值。

5.2.2 归纳假设$N \times N$

当方格扩展到$N \times N$个方格时，为了计算其最大能达到的数值， 采用分析假设的数学思想。

当格子数为1×1时，由于随机出现的数为 4 或 2，要达到最大值，则出现 4，故最大值为2^2，如表 4：

表 4

4

当格子数为2×2时，,左上角出现 2 时无法移动，游戏终止；当出现 4 时，继续移动，如表 5：

表 5

4	4
16	8

即达到的最大值为$2^5 = 32$

当格子数达到3×3时左上角出现 2 时无法移动，游戏终止；当出现 4 时，游戏继续，如表 6：

表 6

4	4	8
16	32	64
512	256	128

所以达到的最大值为$2^{10} = 1024$。

当格子数达到4×4时，已经求出其最大能出现的值为2^{17}。

所以假设当格子数达到$N\times N$时,如表 7。

表 7 递归表

4	4	…	…	2^{N}
…	…	…	…	…
…	…	…	…	…
…	…	…	…	…
2^{N^2}	2^{N^2-1}	…	…	2^{N^2-N+1}

假设2^{N^2+1}能够合成，那么其上一步一定由两个2^{N^2}合成， 又假设其中一个2^{N^2}已合成，那么另一个2^{N^2}的上一步一定是由两个2^{N^2-1}合成。假设其中一个2^{N^2-1}已合成，另一个2^{N^2-1}的上一步一定……依此类推，假设由$N\times N$个方格组成，即空位有N^2个，此时左上角第二个数为2^2，所以还有空位生成2^2，2^2可以生成 2 个，来生成2^3，以此类推，所以能够生成2^{N^2+1}，即假设成立。

6. 模型评价

6.1 模型优点

1. 仅用递归思想就将数据进行处理得出结论，结构清晰，操作简单，可读性强，而且容易用数学归纳法来证明算法的正确性；

2. 所建立的模型具有很强的数学理论基础，增加了结论的可信度；

3. 排除了一次合并多个数字和一次不能合并数字的客观因素；

4. 能客观的反应游戏结果的真实性和可行性，防止主观偏差；

5. 根据建立的模型完成目标数字 2048 所用的步骤相对较少，思路清晰，规律性强且运用此模型能够组成更大的数字。

6.2 模型缺点

1.在实际操作中，游戏情况千变万化，而本文只考虑了几种特殊的情况，与实际结果产生误差；

2.不能将所有影响游戏结果的因素都纳入计算，使之产生偏差。

6.3 模型推广

为解决本题，采用递归算法和 *mini-max* 算法，通过建立模型对数据进行处理得出结论。

该模型可以推广到九连环、象棋等问题的研究上，如利用本文所建立的模型来解决九连环的解法问题，利用递归算法可以得出套上 *n* 连环所需步数为:

$$f(n)=\frac{[2^{n+1}-\frac{1}{2}(-1)^n-15]}{3}$$

6.4 模型优化

此模型建立时将一次能合并多个数和无效合并的因素排除，使得模型得出的结论存在偏差。故方案应该将这两个因素也考虑进去，并作出相应的模型，同时也要考虑每次数字出现的位置。

参考文献

[1]http://www.guokr.com/post/577816/

[2]梁彬,基于概率的混合 Mini-max 搜索算法,中山大学,硕士学位论文,2006.

[3]杨爱民,刘春凤,曲、屈静国,大学数学实验,北京:科学出版社,2012.

[4]姜启源,数学模型（第二版）,北京:高等教育出版社,1993.

[5]蔡锁章，数学建模原理与方法,北京:海洋出版社，2000.

[6]Richard O.Duda,Peter E.Hart,David G.Stork,模式分类,机械工业出版社,中信出版社,2011.

[7]唐焕文,贺明峰,数学模型引论（第二版）,北京:高等教育出版社,2002.

[8]孙博文,分形算法与程序设计与 Visual C++实现,北京:科学出版社，2004.

[9]姜昕,言穆江,象棋入门一月通，上海文化出版社,2005,15.

[10]孙伟,马绍汉,分布式博弈树搜索算法,计算机学报,第 18 卷,第 1 期.

[11]肖齐英,王正志,博弈树搜索与静态估计值函数,计算机应用研究,第 4 期 1997,74-76.

[12]H.J .Berliner,Some Necessary Conditions for a Master Chess Program,Procs.3rd Int.Joint Conf.on Art.Intell,(Menlo Park:SRI),Stanfoet,1993,77-85.

[13]王晓东,计算机算法设计与分析,北京:电子工业出版社,2001.
[14]王迤远,刘云霞,递归模拟过程的实现,周口师范高等专科学报,2000,17(5):78~80.
[15]王永燕,康健,一种直观的递归模拟方法,济南大学学报,2001,15(3):240~242.
[16]齐治昌,谭庆平,宁洪,软件工程(第二版),北京:高等教育出版社,2004:231~340.

附录

附录[1]

```
/*2048*/
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<time.h>

int code[4][4]={0,0,0,0,
                0,0,0,0,
                0,0,0,0,
                0,0,0,0};/*游戏中的 16 个格子*/
int temp[5];/*中间变量*/
int move=0;/*移动次数*/
int score=0;/*分数*/

void print(void)/*显示游戏界面*/
```

```
{
    int i,j;

    clrscr();/*清屏*/
    printf("2048\n");
    printf("W--UP  A--LEFT  S--DOWN  D--RIGHT  0--EXIT\n");
    printf("Score:%d Move:%d\n",score,move);
    printf("Made by Yanjisheng\n");
    printf("|------------------|\n");/*显示横向分隔线*/
    for(i=0;i<=3;i++)
    {
        for(j=0;j<=3;j++)
        {
            if(code[i][j]==0)
            {
                printf("|       ");/*0 显示空格*/
            }
            else
            {
                printf("|%4d",code[i][j]);/*显示数字和分隔线*/
            }
        }
        printf("|\n|------------------|\n");/*显示横向分隔线*/
    }
}

int add(void)/*对中间变量数组进行处理*/
{
    int i;
    int t=0;
    int change=0;/*判断数组是否有改变，0 不变，1 变化*/

    do
    {
        for(i=0;i<=3;i++)
        {
            if(temp[i]==0)
            {
                if(temp[i]!=temp[i+1])
                    change=1;/*当一个 0 后面不是 0 时数组改变*/
                temp[i]=temp[i+1];
                temp[i+1]=0;
            }
        }/*去掉中间的 0*/
```

```
            t++;
        }while(t<=3);/*重复多次*/
        for(i=1;i<=3;i++)
        {
            if(temp[i]==temp[i-1])
            {

                if(temp[i]!=0)
                {
                    change=1;/*当两个非零相同的数相加时数组改变*/
                    score=score+temp[i];/*加分*/
                }
                temp[i-1]=temp[i-1]*2;
                temp[i]=0;
            }
        }/*把两个相邻的相同的数加起来*/
        do
        {
            for(i=0;i<=3;i++)
            {
                if(temp[i]==0)
                {
                    temp[i]=temp[i+1];
                    temp[i+1]=0;
                }
            }/*去掉中间的 0*/
            t++;
        }while(t<=3);/*重复多次*/
        return change;
}

int main(void)
{
        int gameover=0;/*判断游戏是否结束，1 结束，0 继续*/
        int i,j;
        int change=1;/*判断格子中的数是否改变，0 不变*/
        char input;

        srand((unsigned)time(NULL));/*设置随机数的起点*/
        while(gameover==0)
        {
            if(change>=1)/*仅当数发生改变时添加新数*/
            {
                do
```

```
        {
            i=((unsigned)rand())%4;
            j=((unsigned)rand())%4;
        }while(code[i][j]!=0);
        if(((unsigned)rand())%4==0)
        {
            code[i][j]=4;
        }
        else
        {
            code[i][j]=2;/*随机选一个空格填上 2 或 4*/
        }
        move++;/*增加次数*/
    }
    print();/*显示*/
    input=getch();/*输入方向*/

    change=0;
    switch(input)
    {
        case '0':/*退出*/
            printf("Are you sure to exit?(y/n)");
            input=getchar();
            if(input=='y'||input=='Y')
                exit(0);
            break;

        case 'W':
        case 'w':/*上*/
            for(j=0;j<=3;j++)
            {
                for(i=0;i<=3;i++)
                {
                    temp[i]=code[i][j];/*把一列数移到中间变量*/
                }
                temp[4]=0;
                change=change+add();
                for(i=0;i<=3;i++)
                {
                    code[i][j]=temp[i];/*把处理好的中间变量移回来*/
                }
            }
            break;
```

```
case 'A':
case 'a':/*左*/
    for(i=0;i<=3;i++)
    {
        for(j=0;j<=3;j++)
        {
            temp[j]=code[i][j];/*把一行数移到中间变量*/
        }
        temp[4]=0;
        change=change+add();
        for(j=0;j<=3;j++)
        {
            code[i][j]=temp[j];/*把处理好的中间变量移回来*/
        }
    }

    break;

case 'S':
case 's':/*下*/
    for(j=0;j<=3;j++)
    {
        for(i=0;i<=3;i++)
        {
            temp[i]=code[3-i][j];/*把一列数移到中间变量*/
        }
        temp[4]=0;
        change=change+add();
        for(i=0;i<=3;i++)
        {
            code[3-i][j]=temp[i];/*把处理好的中间变量移回来*/
        }
    }
    break;

case 'D':
case 'd':/*右*/
    for(i=0;i<=3;i++)
    {
        for(j=0;j<=3;j++)
        {
            temp[j]=code[i][3-j];/*把一行数移到中间变量*/
        }
        temp[4]=0;
```

```
                    change=change+add();
                    for(j=0;j<=3;j++)
                    {
                        code[i][3-j]=temp[j];/*把处理好的中间变量移回来*/
                    }
                }
                break;
            }
            gameover=1;
            for(i=0;i<=3;i++)
                for(j=0;j<=3;j++)
                    if(code[i][j]==0)
                        gameover=0;/*所有格子都填满则游戏结束*/

        }
        printf("Game over!\n");
        getch();
        return 0;
}
```

附录[2]

模型前试验	1	2	3	4	5	6	7
y	64	1024	512	512	64	512	128
	8	9	10	11	12	13	14
	1024	1024	512	512	512	256	512
	15	16	17	18	19	20	21
	64	1024	128	512	256	256	128
	22	23	24	25	26	27	28
	512	1024	512	256	128	128	128
	29	30	31	32	33	34	35
	1024	128	128	512	512	64	64
	36	37	38	39	40	41	42
	128	512	128	256	256	1024	1024
	43	44	45	46	47	48	49
	512	256	64	1024	512	256	256
	50	51	52	53	54	55	56
	256	256	521	64	1024	1025	512
	57	58	59	60	61	62	63
	512	256	128	128	128	512	512
	64	65	66	67	68	69	70
	1024	512	512	1024	512	256	256
	71	72	73	74	75	76	77
	1024	512	256	128	128	1024	512

	78	79	80	81	82	83	84
	512	512	512	512	256	256	1024
	85	86	87	88	89	90	91
	128	128	256	256	512	1024	256
	92	93	94	95	96	97	98
	512	1024	128	256	1024	512	512
	99	100					
	512	512					

第一次试验	1	2	3	4	5	6	7
y	2048	1024	1024	512	2048	1024	1024
	8	9	10	11	12	13	14
	512	2048	2048	512	1024	1024	1024
	15	16	17	18	19	20	21
	1024	2048	2048	1024	512	512	256
	22	23	24	25	26	27	28
	1024	2048	2048	1024	512	1024	1024
	29	30	31	32	33	34	35
	512	512	1024	2048	2048	2048	2048
	36	37	38	39	40	41	42
	1024	512	1024	1024	1024	1024	1024
	43	44	45	46	47	48	49
	2048	2048	2048	2048	512	512	1024
	50	51	52	53	54	55	56
	1024	2048	2048	2048	2048	2048	1024
	57	58	59	60	61	62	63
	1024	512	512	1024	1024	2048	2048
	64	65	66	67	68	69	70
	2048	2048	1024	512	512	256	1024
	71	72	73	74	75	76	77
	1024	2048	2048	1024	1024	512	1024
	78	79	80	81	82	83	84
	1024	1024	1024	2048	2048	2048	2048
	85	86	87	88	89	90	91
	1024	2048	1024	1024	2048	2048	1024
	92	93	94	95	96	97	98
	2048	2048	2048	2048	2048	2048	1024
	99	100					
	2048	2048					

第三次试验	1	2	3	4	5	6	7

y	2048	2048	2048	2048	2048	2048	2048
	8	9	10	11	12	13	14
	2048	2048	2048	2048	2048	2048	2048
	15	16	17	18	19	20	21
	2048	2048	2048	2048	1024	2048	2048
	22	23	24	25	26	27	28
	2048	2048	2048	2048	2048	2048	2048
	29	30	31	32	33	34	35
	2048	2048	2048	2048	2048	2048	2048
	36	37	38	39	40	41	42
	2048	2048	2048	2048	2048	2048	2048
	43	44	45	46	47	48	49
	1024	2048	2048	2048	2048	2048	2048
	50	51	52	53	54	55	56
	2048	2048	2048	2048	2048	2048	2048
	57	58	59	60	61	62	63
	2048	512	1024	2048	2048	2048	2048
	64	65	66	67	68	69	70
	2048	2048	2048	2048	2048	2048	2048
	71	72	73	74	75	76	77
	2048	2048	2048	2048	2048	2048	2048
	78	79	80	81	82	83	84
	2048	2048	1024	2048	2048	2048	2048
	85	86	87	88	89	90	91
	2048	2048	2048	2048	2048	2048	2048
	92	93	94	95	96	97	98
	2048	2048	2048	2048	2048	2048	2048
	99	100					
	2048	2048					

Appendix [3]

```
F = Fit[data, {1, x}, x]
72.4167 + 0.59018 x
pd = ListPlot[data, DisplayFunction -> Identity];
fd = Plot[F, {x, 20, 700}, DisplayFunction -> Identity];
Show[pd, fd, DisplayFunction -> $DisplayFunction]
```