评委一评分，签名及备注	队号： 10376	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

题目：基于 Monte-Carlo 随机模拟算法的 2048 游戏 AI

摘要

2048 是近期网络上流行的一款规则简单的益智游戏。它容易上手，但是想要获胜也很不容易。本文分析 2048 的游戏过程并完成了求胜 AI 的设计。

首先，本文建立了游戏的**博弈模型**，分析了其双方的决策集合，游戏的博弈树和决策分支。 本文从理论上分析了良好的决策在博弈树上确定的路径应使得“在此决策前提下，之后进行随机决策，最后能到达 2048”这一事件的概率不断增大。

本文基于游戏本身性质，引入一个简单的**估价指标**——盘面数值之和，来衡量“从给定游戏局面出发，到达 2048”这一事件的概率。本文在几个特殊局面下评估了这指标的一些倾向性。

其后，本文利用 **Monte-Carlo 随机模拟**算法，在某个决策分支点，对四个方向移动得到的新局面多次独立模拟一定步数的随机操作，得到一定步数后游戏格局估价指标的期望值的估计；比较其大小，由此确定此决策分支点应做出的决策。本文利用 matlab 编程实现 AI，并基于不同的取样值、模拟步数的胜率指标对其进行了评价。

最后，本文分析了 4*4 盘面和 N*N 盘面理论上能够得到的最大数值，对于 N*N 盘面，这个数值是2^{N^2+1}。

关键字：博弈模型，估价指标，Monte-Carlo 随机模拟

基于 Monte-Carlo 随机模拟算法的 2048 游戏 AI

1. 问题重述

2048 是近期风靡网络的一款游戏，它的规则很简单：在 4*4 方格盘中有数字方块，他们均为 2 的正整数次幂。每次控制所有数字方块向同一方向移动，两个相同数字的方块碰撞时会合并为他们的和，每次操作之后在空白处随机生成一个 2 或者 4。得到数字 2048 则胜利，格子都被填满且相邻格子都不同——即无法向任何方向移动，则失败。

针对游戏 2048 建立数学模型，并使用完成游戏所需移动次数和获胜概率两个指标来评估算法的表现。同时，讨论 4*4 方格盘下，得到 2048 之后，继续进行游戏所可能达到的最大数值，并将其推广至 N*N 方格盘。

2. 假设与符号说明

2.1 假设

- 2.1.1）2048 游戏原作者做出的设定、规则和胜负判定在每次操作中都被玩家和系统遵从。主要包括以下三点：
  - 玩家每次操作向一个方向移动数字方块。
  - 系统移动方块穿过空格直到它碰到其他方块为止。
  - 系统每次移动中只能合并一次方块，不能连续合并。

  举例来说，列[0，2，2，4]在一次右移操作后变成[0，0，4，4]，而不是[0，0，0，8]；形成后者需要两次右移操作。
- 2.1.2）玩家每次操作后，系统生成新数字方块的位置随机取。在玩家第一次操作前，系统首先在空白方格盘上随机生成一个 2 或 4。系统每次生成 4 的概率为 10%，生成 2 的概率为 90%。（依据 2048 源代码）
- 2.1.3）忽略伪随机数种子及其内在局限性、规律性对计算机随机模拟程度的影响。即当本文中提到任何“随机”概念时，均指不可预测的真正意义上的随机。

2.2 符号说明

表 1-参变量符号说明

名称	含义
$\boldsymbol{A_k}$	第 k+1 次操作前的格局，详见问题分析部分。
$\boldsymbol{A_F}$	游戏结束时的最终格局。
$\boldsymbol{f_{\rightarrow}(A_k)}$	第 k+1 次滑动操作，为向右滑动。相似地其他方向参见下标。详见问题分析部分。
$\boldsymbol{R(A_k^d)}$	第 k+1 次滑动操作后生成随机数字方格的生成操作，详见问题分析。
$\boldsymbol{n}$	进行 Monte-Carlo 模拟的样本容量。
$\boldsymbol{t}$	进行 Monte-Carlo 模拟的运行深度。
$\boldsymbol{Ev(A_k)}$	对格局A_k的估价函数。

（续表 1）

$\max(A_F)$	游戏结束时格局中存在的最大数。
$step(A_F)$	游戏结束所耗费的用户滑动操作次数。
$time(A_F)$	一局游戏耗费的现实时间，用来评价我们算法下计算机的思考时间。
N	游戏的阶，指方格盘的行列数。
$supa_N$	N 阶游戏能拼出的数的最大理论值。

3. 问题分析

3.1 问题的抽象表述分析：

我们引入几个概念来抽象地表述一局游戏。

- 3.1.1）**格局 - A_k**：表示第 k+1 次操作前的方格盘情况。当 k=0 时，A_0表示游戏的初始情况。对于4×4的游戏来说，A_i是一个四行四列的矩阵；为了我们讲述和计算机运算的便利，令A_k中的值为实际方格盘中的数值取以 2 为底的对数。且规定空格在A_k中对应的值为零。

  若用A_k^{real}表示实际方格盘矩阵：

$$A_k(i,j) = \begin{cases} \log_2 A_k^{real}(i,j), where\ A_k^{real}(i,j)\ has\ a\ value. \\ 0, \qquad\qquad where\ A_k^{real}(i,j)\ is\ NAN. \end{cases}$$

举例来说，对于如图所示的真实格局：A_{k0}^{real}，我们有格局：

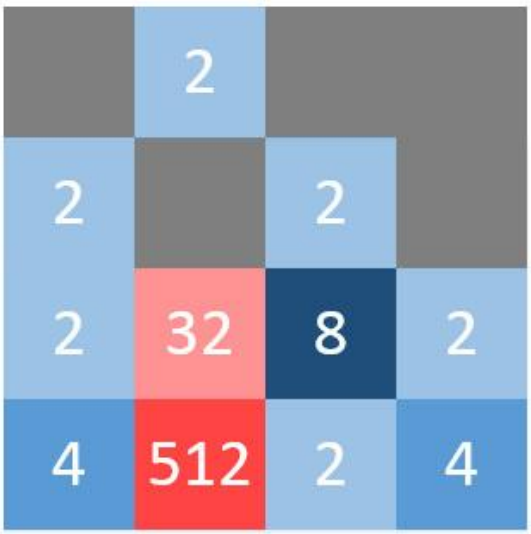


图 1 - 一个真实格局的例子

$$A_{k_0} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 1 & 5 & 3 & 1 \\ 2 & 9 & 1 & 2 \end{bmatrix}$$

- 3.1.2）**滑动操作** – $f_{\uparrow}(A_k), f_{\downarrow}(A_k), f_{\leftarrow}(A_k), f_{\rightarrow}(A_k)$：表示第 k+1 次操作。四种滑动操作代表的滑动方向由下标可以直观地看出。操作都是映射，第 k+1 次滑动操作映射处理的量为格局矩阵A_k，输出的量为格局矩阵A_k^d。

- 3.1.3）**生成操作** – $R(A_k^d)$：表示第 k+1 次操作后系统生成二和四的过程。相似地，它也为映射，输出的量为格局矩阵A_{k+1}。

  以上两种操作构成了游戏的主要部分，仍然对于格局$A_{k_0}^{real}$：

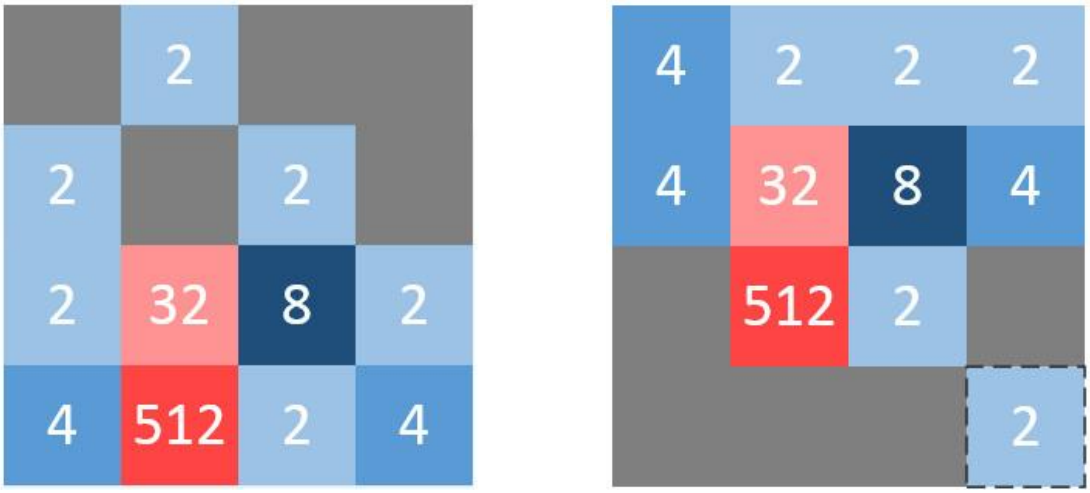


图 2 – 矩阵操作例子

$$A_{k_0+1} = R\left(f_{\uparrow}(A_{k_0})\right) = \begin{bmatrix} 2 & 1 & 1 & 1 \\ 2 & 5 & 3 & 2 \\ 0 & 9 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3.2 双人博弈分析

我们发现，游戏具有如下特征：1）计算机和玩家按一定顺序采取行动。2）计算机和玩家每一次行动所面对的选择的集合是有限集合。3）游戏在一定条件满足时终止。

基于上述特征，我们将游戏归于一个双人动态博弈问题[1]。每一回合玩家和系统分别做出一个决策；给定当前格局A_k或A_k^d双方的决策集合分别为：

$$\Omega_{player} = \{ f_{\uparrow}(A_k), f_{\downarrow}(A_k), f_{\leftarrow}(A_k), f_{\rightarrow}(A_k) \}$$

$$\Omega_{computer} = \begin{cases} \{R(A_k^d)\} & , \min(A_k^d) = 0 \\ \emptyset & , \min(A_k^d) > 0 \end{cases}$$

以图一的格局为例，以

$$A_{k_0}=\begin{bmatrix}0 & 1 & 0 & 0\\ 1 & 0 & 1 & 0\\ 1 & 5 & 3 & 1\\ 2 & 9 & 1 & 2\end{bmatrix}$$

为上端节点，所产生的深度为 2 的局部博弈树为：

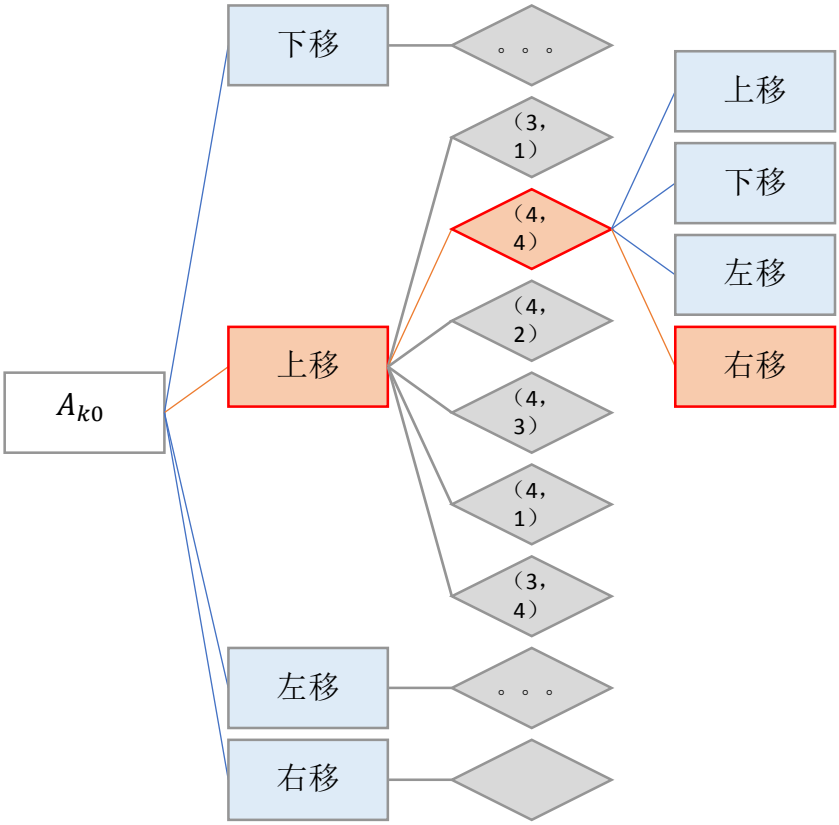


图 3 - 游戏决策树

要找到最优解，可以基于博弈内部设定来构造评价一个格局好坏的估价函数。在每个决策分支点处向下进行一定深度的搜索，遍及深度内所有的可能情况；在上述决策分支点选择估价函数取值最大的解。

考虑到优秀的估价函数构造的难度；搜索深度增大时庞大的解的空间；计算机一方决策的随机性，本文中我们将不采用这种传统的算法来解决此问题。

4. 模型建立

4.1 制胜决策的概率模型[2]

为了得到 2048，我们需要在决策树的分支点处做出“正确”的决策。我们基于 Monte-Carlo 随机模拟来实现决策。

仍然考虑图一的格局A_{k0}以及以它为起始点生成的决策树图三，假设我们从此时开始抛硬币来随机决定执行的每一步滑动操作，则在所有可能的结束状态中，总归有能够达到 2048 的路线。这一概率值是确定的，记为p_{Ak0}。

记事件E_0为“通过随机决策，从格局A_{k0}到达含有 2048 的最终格局”，有：

$$P(E_{k_0}) = p_{Ak0},$$

$$(decisions\ are\ randomly\ made)$$

考虑决策树图三。现在(在随机决策的前提下)，我们只关心第一个分支点，将四种滑动方向的决策分别记作事件：

$$S_{\uparrow} = \{slide\ up\ at\ A_{k0}\}$$
$$S_{\downarrow} = \{slide\ down\ at\ A_{k0}\}$$
$$S_{\leftarrow} = \{slide\ left\ at\ A_{k0}\}$$
$$S_{\rightarrow} = \{slide\ right\ at\ A_{k0}\}$$

则，由全概率公式：

$$P(E_{k_0}) = P(S_{\uparrow})P(E_{k_0}|S_{\uparrow}) + P(S_{\downarrow})P(E_{k_0}|S_{\downarrow}) + P(S_{\rightarrow})P(E_{k_0}|S_{\rightarrow}) + P(S_{\leftarrow})P(E_{k_0}|S_{\leftarrow})$$

$$P(E_{k_0}) = \frac{1}{4} \times \left[P(E_{k_0}|S_{\uparrow}) + P(E_{k_0}|S_{\downarrow}) + P(E_{k_0}|S_{\rightarrow}) + P(E_{k_0}|S_{\leftarrow})\right]$$
$$\leq \max\{P(E_0|S_{\uparrow}), P(E_0|S_{\downarrow}), P(E_0|S_{\rightarrow}), P(E_0|S_{\leftarrow})\}$$

因此，我们用$P(E_0|S_*)$来评价S_*的好坏，一旦我们求得$\max\{P(E_0|S_*)\}$对应的选择，我们即采取这选择到达下一决策分支点，使得此点到达 2048 的概率变为$\max\{P(E_0|S_*)\}$。在下一分支点$A_{k_0}^d$处，记事件$E_{k_0}^d$为“通过随机决策，从格局$A_{k_0}^d$到达含有 2048 的最终格局”，有：

$$P(E_{k_0}^d) = \max\{P(E_{k_0}|S_*)\} \geq P(E_{k_0})$$

虽然我们无力左右计算机的$R(A_k^d)$操作，但我们在每一分支点按上述方法做决策：

$$\forall k, \quad S = S_*,$$
$$s.t \ : \ P(E_0|S_*) = \max\{P(E_0|S_{\uparrow}), P(E_0|S_{\downarrow}), P(E_0|S_{\rightarrow}), P(E_0|S_{\leftarrow})\}$$

在$A_{k+1} = R(A_k^d)$操作后，若有：

$$P(E_{k+1}) \geq P(E_k^d)$$

则：

$$P(E_{k+1}) \geq P(E_k^d) = \max\{P(E_k|S_*)\} \geq P(E_k)$$

考虑到：

$$P(E_{k+1}^d) = \max\{P(E_{k+1}|S_*)\} \geq P(E_{k+1})$$

$$\Rightarrow P(E_{k+1}^d) \geq P(E_k^d)$$

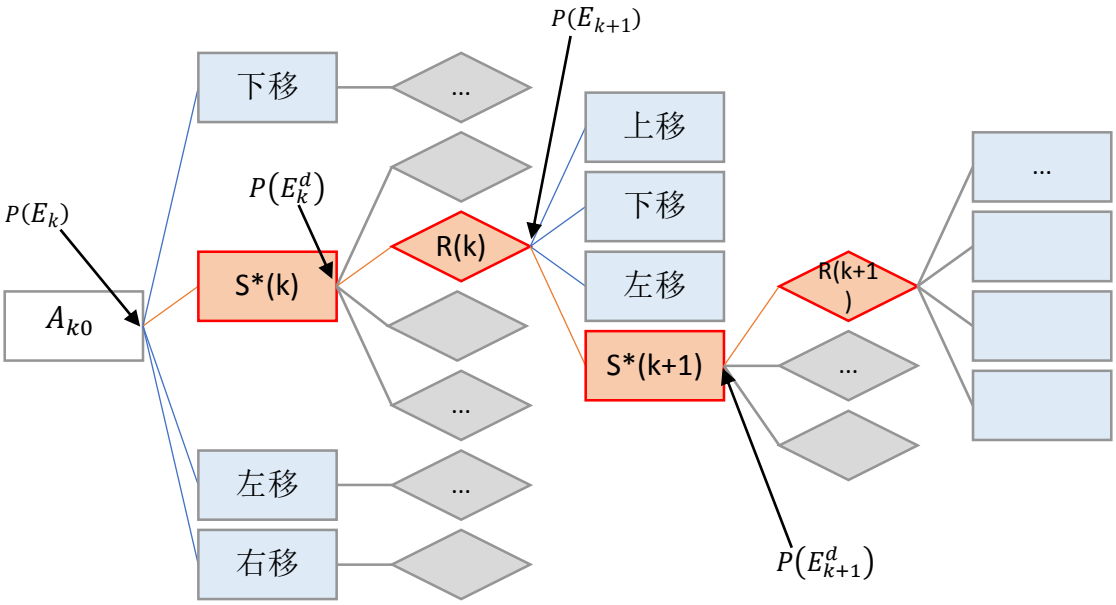


图 4 - “得到 2048”概率增长决策模型

这样一来，我们的决策就使得“在A_k^d格局处，通过随机决策最终到达含有 2048 的最终格局”这一事件的概率随着 k 的增大而增大。最终真正地到达含有 2048 的最终格局。

4.2 确定条件概率

Monte-Carlo 随机模拟方法本质上就是通过做一系列独立试验，用频率值来估计目标概率值，我们将会模拟随机游戏试验来确定最优决策。而我们模型现在的关键目标就是在对局中的任意决策分支点，确定四种滑动操作决策的$P(E_k|S_*) = p$，条件概率大小。[3]

我们在此分支点选择S_*的情况下，做 n 次重复独立的随机操作游戏试验，记μ为到达含有 2048 的最终格局的游戏试验次数。

而由 Bernoulli 大数定律：

$$\lim_{n\to\infty}\left(\left|\frac{\mu}{n} - p\right| < \varepsilon\right) = 1$$

这就是我们使用 Monte-Carlo 随机模拟方法对概率值作统计学估计的理论依据。

4.3 确定评估指标

显然地，进行随机滑动决策的游戏，达到 2048 终局的概率极低，大多数终局都是失败的 。如果样本数太小，事件发生过少或根本不发生，无法估计概率值。因此需要很庞大的样本数才能得出有一定置信度的概率估计结果。

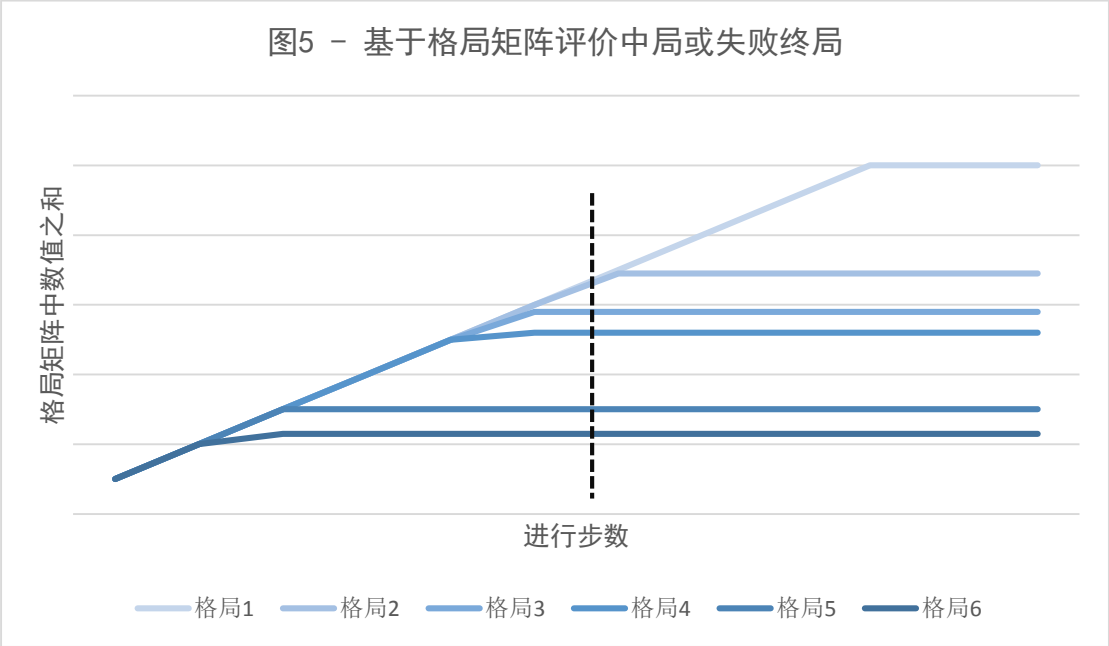
考虑到算法运行的效率，我们不可能取过多样本，相应地，我们不会使每一局游戏都运行结束。因此，十分自然地，我们引入一个简单有效的评估指标来反应每一个失败的终局或进行中的中局与 2048 终局的接近程度，代替概率量。

我们首先介绍游戏的几个重要性质规律来引入这个指标。

- 1）单增性
  格局矩阵A_k内所有数值的和在游戏进行中不断累积增加。
- 2）一致性
  两局不同的游戏，如若都保持不失败直到给定的步数t_0，当t_0足够大以消除随机生成 0（无空格情形），2 或 4 的影响之后，它们格局矩阵A_k内所有数值的和一致，即：较高的和对应较多的步数，即较长的不败时间。
- 3）二元性
  若不设定合成数值的上界，游戏只有两个状态：进行中或失败。

因此，我们使用格局矩阵A_k内所有数值的和来衡量一个失败的终局或进行中的中局与含有 2048 终局的接近程度。引入估价函数$Ev(A_k)$：

$$Ev(A_k) = \sum_{i=1}^{4}\sum_{j=1}^{4} A_k(i,j)$$



如图所示，步数计数器持续工作。起初游戏进行中时，指标持续增长。当游戏失败时，曲线转为水平，停止增长。在给定步数（虚线）处，

$$Ev(A_{k_1}) = Ev(A_{k_2}) > Ev(A_{k_3}) > Ev(A_{k_4}) > Ev(A_{k_5}) > Ev(A_{k_6})$$

4.4 算法综述

综合以上内容，对我们的求解算法作以下综述：

- 我们依据制胜决策的概率模型，对于在每一个决策分支点处，我们求解“在A_k格局处，在选择S_*滑动操作的条件下，通过随机操作最终到达含有 2048 的最终格局”这一事件的概率来选择较优操作。
- 我们按照 Monte-Carlo 随机模拟方法，在选择S_*滑动操作的条件下，模拟多次独立的随机操作游戏，基于结果评估S_*操作。

- 我们依据游戏的性质规律，引入估价函数$Ev(A_k)$,判断每一次独立随机操作游戏在给定步数处的中局或失败终局状态与含有 2048 终局的接近程度。由此提高程序运行效率。

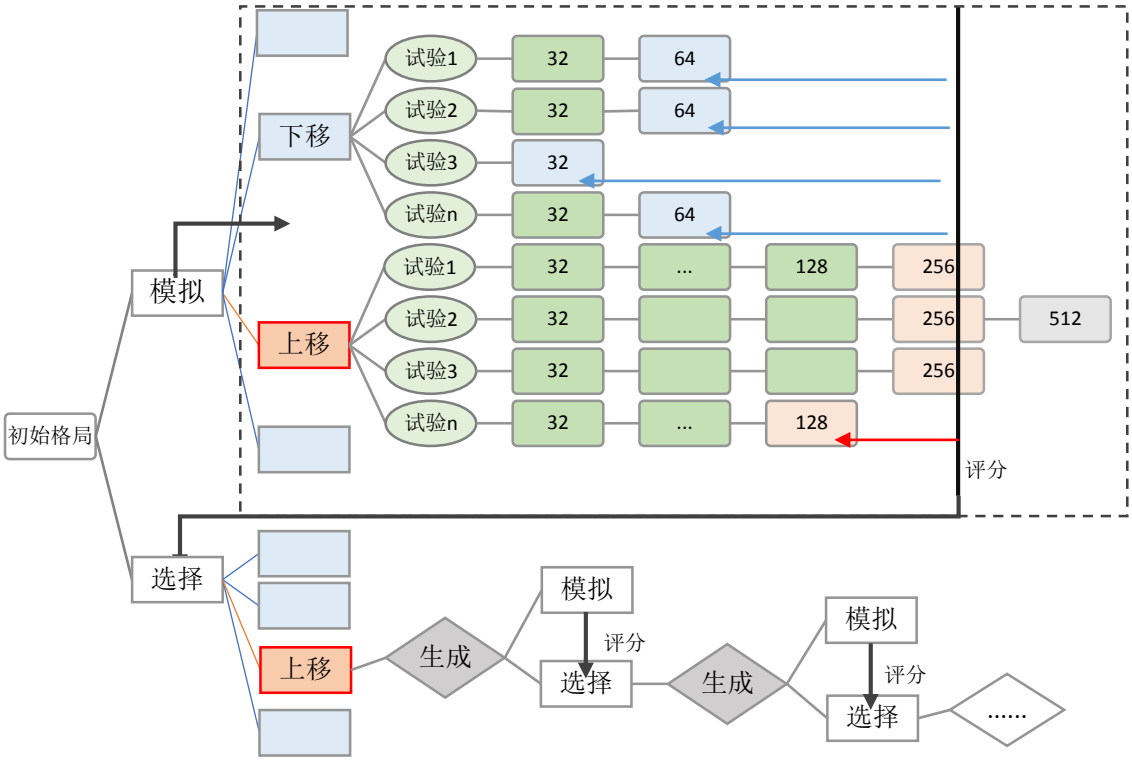


图 6 - 算法运行模式图

5. 模型的求解

5.1 AI 运行结果

我们使用 Matlab 实现模型（代码见附录 2，程序包见附件），如图为其中一个得出 2048 终局的示例和另一个在最大值为 1024 时失败的示例。

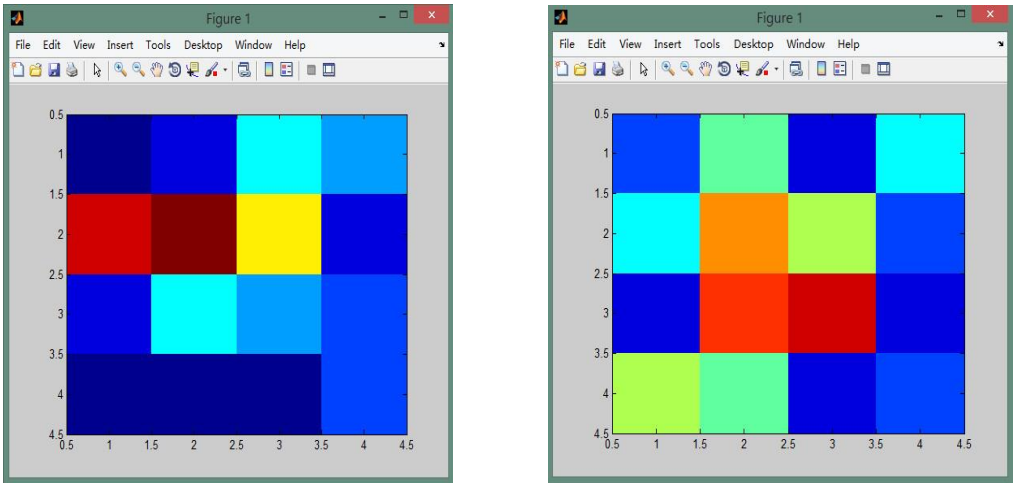


图 7 - Matlab 游戏模拟，成功|失败，n=120，t=60

n 和 t 是模拟 AI 的两个核心参数。n 表示样本容量，即进行随机游戏的局数。t 表示模拟深度，即每一局模拟随机决策游戏向后进行的步数，我们还会在下文进一步讨论 n 和 t 对于模型整体表现的影响。

这两个图像使用 imagesc()绘图函数做出矩阵的颜色图，设定为 gcf 颜色模式，按照光谱颜色顺序偏红色为较大值，偏蓝色为较小值。左侧图中（2，2）深红格子即为 2048。两幅图像对应的格局为：

$$A_{left}=\begin{bmatrix}0&1&4&3\\10&11&7&1\\1&4&3&2\\0&0&0&2\end{bmatrix}\ ;\ A_{left}^{real}=\begin{bmatrix}0&2&16&8\\1024&2048&128&2\\2&16&8&4\\0&0&0&4\end{bmatrix}$$

$$A_{left}=\begin{bmatrix}2&5&1&4\\4&8&6&2\\1&9&10&1\\6&5&1&2\end{bmatrix}\ ;\ A_{right}^{real}=\begin{bmatrix}4&32&2&16\\16&256&64&4\\2&512&1024&2\\64&32&2&4\end{bmatrix}$$

附录 1 中收纳了 n=120，t=60 情况下 24 次测试的图像。

5.2 估价指标的表现

基于模型 4.3 的讨论，我们取了盘面中所有数值的和作为估价指标。

$$Ev(A_k)=\sum_{i=1}^{4}\sum_{j=1}^{4}A_k(i,j)$$

对于滑动操作S_*，进行 n 局随机模拟游戏，运行得到 n 个失败的终局或进行中的中局，则十分自然地，我们对S_*的评分为这些终局或中局评分的算术平均：

$$Ev(S_*)=\frac{1}{n}\times\sum_{p=1}^{n}\sum_{i=1}^{4}\sum_{j=1}^{4}A_{Fp}(i,j)$$

我们通过几个实例来衡量这估价指标在不同情况下的表现，这里我们取样本容量为 100，模拟深度为 50。

5.2.1 平滑倾向

我们的估价指标对于使得操作后格局趋向较好“平滑性”的决策。这里，平滑性指相邻格子中数值的接近程度，显然相邻格子中数值越接近，对合并越有利。我们通过特别构造的格局来使估价函数显示其平滑倾向。

如图 8（下页）所示的格局：

$$A_{test1}=\begin{bmatrix}0&2&0&0\\2&3&0&0\\3&1&2&0\\4&4&3&2\end{bmatrix}$$

容易看出，对于格局A_{test1}，$f_{\uparrow}(A_{test1})$将会在矩阵第一行形成四个连续的 4，在第二行形成三个连续的八。与单纯追求合成较大的 32 的$f_{\rightarrow}(A_{test1})$相比，是倾

向于增强新格局平滑性的操作。我们观察 AI 对于“向上”与“向右”两个操作的估价。

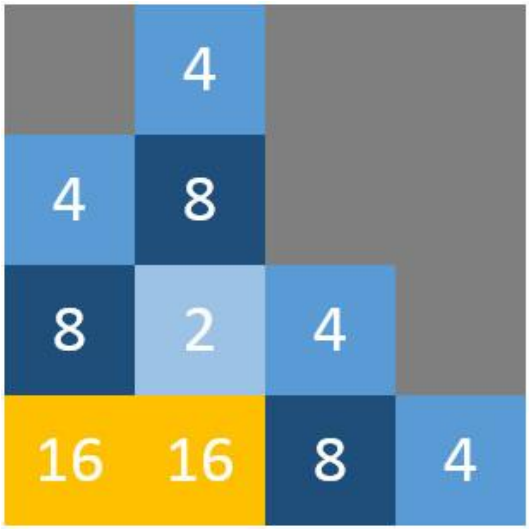


图 8 - 平滑倾向揭示格局，平滑倾向性操作为向上，对照组为向右

我们取 n=100，t=50，作 100 次估价试验，结果如下。图中横坐标为模拟次数，纵坐标为 AI 依照估值指标判断出的平滑倾向操作（向上）较向右操作的优越程度。

$$y = Ev(f_{\uparrow}(A_{test1})) - Ev(f_{\rightarrow}(A_{test1}))$$

AI 有 60%以上的概率认为$f_{\uparrow}(A_{test1})$优于$f_{\rightarrow}(A_{test1})$。我们说，在这个格局下，AI 表现出了平滑倾向性。

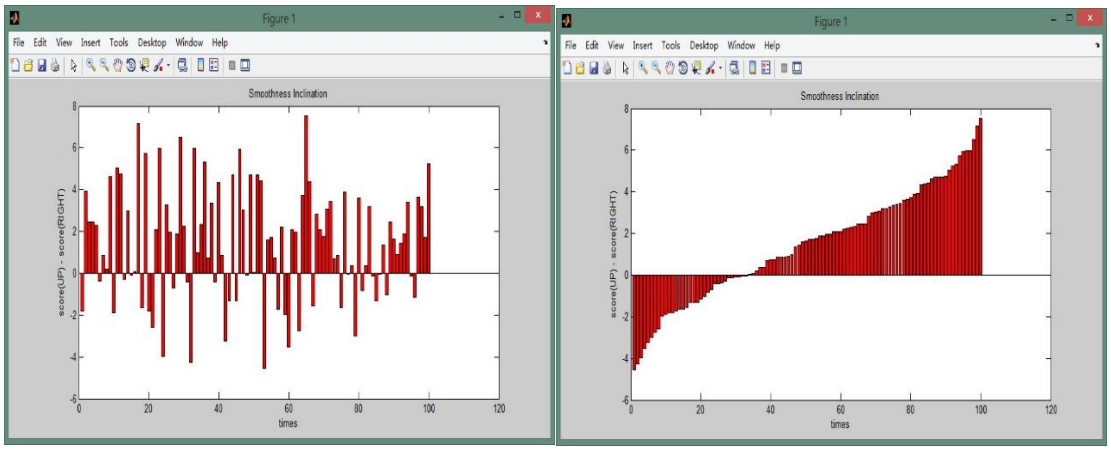


图 9 - 平滑倾向性模拟

5.2.2 单调倾向

我们的估价指标对于使得操作后格局趋向较好“单调性”的决策。这里，单调性数值按单调减少或增加顺序排成相连的一列。显然这种性质在数值较大时，可以引起递进式的合并，对得到更大的值有利。我们通过特别构造的格局来使估价函数显示其单调倾向。

如图 8（下页）所示的格局：

$$A_{test2} = \begin{bmatrix} 0 & 0 & 10 & 0 \\ 0 & 9 & 2 & 0 \\ 8 & 3 & 3 & 0 \\ 7 & 4 & 4 & 0 \end{bmatrix}$$

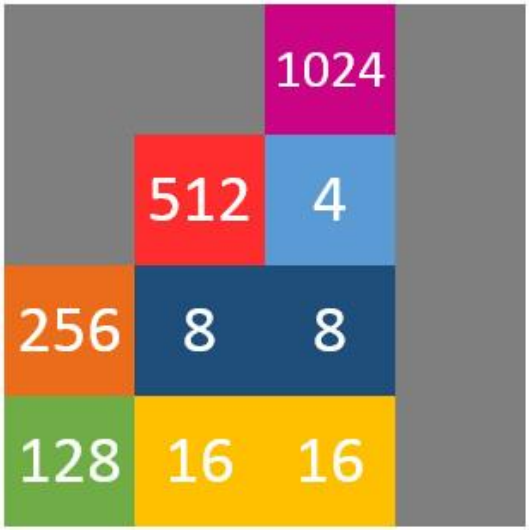


图 10 - 单调倾向揭示格局，单调倾向性操作为向左，对照组为向右

我们取 n=100，t=50，作 100 次估价试验，结果如下。图中横坐标为模拟次数，纵坐标为 AI 依照估值指标判断出的单调倾向操作（向左）较向右操作的优越程度。

$$y = Ev(f_{\leftarrow}(A_{test2})) - Ev(f_{\rightarrow}(A_{test2}))$$

在模拟中，AI 有 80%左右的概率认为$f_{\leftarrow}(A_{test2})$优于$f_{\rightarrow}(A_{test2})$。我们说，在这个格局下，AI 表现出了单调倾向性。

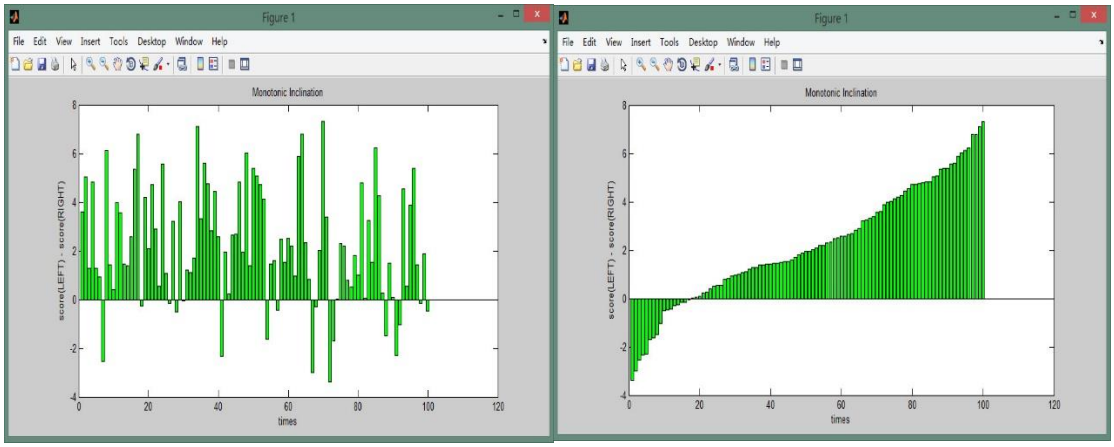
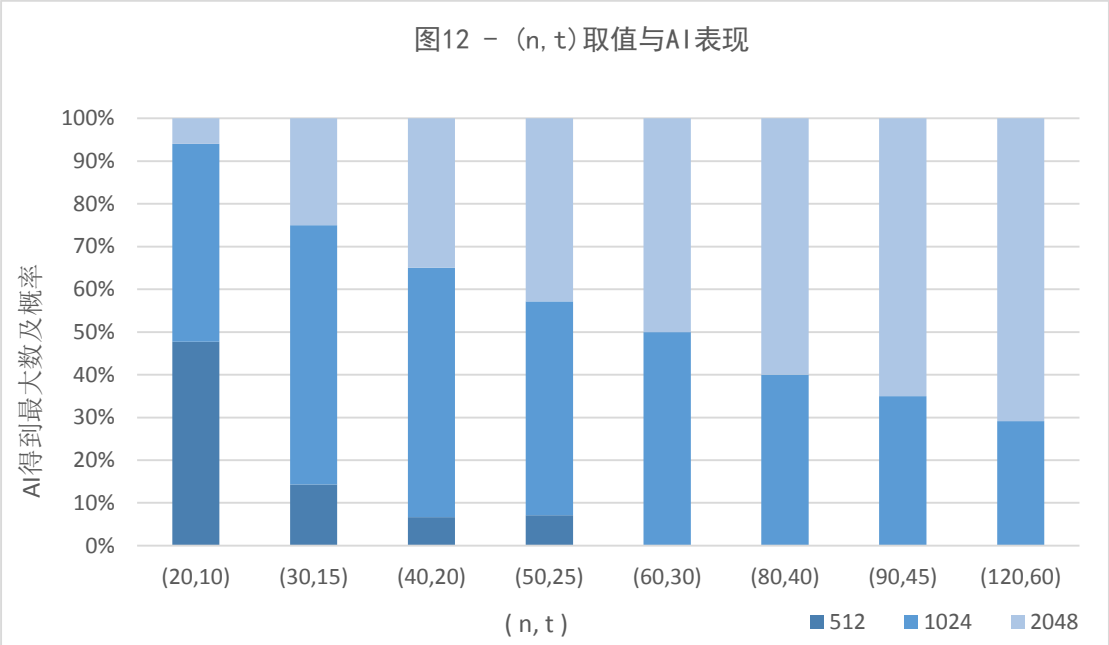


图 11 - 单调倾向性模拟

5.3 AI 的表现

Monte-Carlo 随机模拟 AI 的表现与两个指标密切相关，样本容量 n 与模拟深度 t。

当样本容量较小时，对滑动操作的评分不具有代表性；当模拟深度较浅时，多数随机模拟局还未失败，评分也不可靠；当模拟深度过深时，大多数随机模拟局都已经失败，系统执行无意义的循环，影响运行的效率。基于这种考虑，我们分别在不同的（n，t）组合下，使用获胜概率和移动次数衡量 AI 的表现。



（n，t）	（20，10）	（30，15）	（40，20）	（50，25）	（60，30）	（80，40）	（90，45）	（120，60）
测试次数	67	56	60	42	26	20	20	24

随着（n，t）的一起增大，游戏胜率增大，但增速减缓。同时 AI 每一步的思考时间变长。当（n，t）为（90，45）时，在我们的测试计算机上，一局游戏直胜利（拼出 2048）约需要 11 到 14 分钟。

此外我们也统计了步数指标，计算出不同（n，t）下的平均步数值。

（n，t）	（20，10）	（30，15）	（40，20）	（50，25）	（60，30）	（80，40）	（90，45）	（120，60）
测试次数	67	56	60	42	26	20	20	24
平均步数	430	565	707	798	845	886	934	987

5.4 N 阶游戏能实现的最大数的求解

我们首先考虑 4 阶的情况。我们将证明，4 阶游戏不可能无限制地进行下去。其理论上能够实现的最大数为：

$$supa_N = 2^{17}$$

要拼出2^{17}，盘面上需要同时存在两个2^{16}。而一个2^{16}又需要两个2^{15}，因此，在拼出两个2^{16}之前，必定有一个时刻，盘面中同时存在一个2^{16}和两个2^{15}；同理，在这个时刻之前，必定有一个时刻，盘面中同时存在一个2^{16}，一个2^{15}，两个2^{14}. 依此类推。直到要求盘面内数值个数最多的时刻，最好的情况是以下格局：

$$A_F = \begin{bmatrix} 16 & 15 & 14 & 13 \\ 9 & 10 & 11 & 12 \\ 8 & 7 & 6 & 5 \\ 0 & 2 & 3 & 4 \end{bmatrix}$$

此时，盘面内数值个数的最大值最小，为 15

如果下一步随机生成操作生成 2（在A_F中对应的指数为 1），则游戏失败，若生成 4（对应的指数为 2），则2^{17}能够被合成。

对于任意大于2^{17}的数，盘面内数值个数的最大值的最小值大于等于 16，注意到当盘面内数值个数的最大值取最小时，每一个值都不相等。因此该格局不能进行被任何操作。必将失败。

不难将这一思考推广到 N 阶游戏，在 N 阶游戏中，理论上可能实现的最大数为：

$$supa_N = 2^{N^2+1}$$

在这之前，盘面内必须存在的数值个数的最大值取最小，为N^2-1，这些数值彼此不相等，且必须一个个相连，沿蛇形排布成线型。（此矩阵设 N 为偶数）

$$A_F^{real} = \begin{bmatrix} 2^{N^2} & 2^{N^2-1} & \cdots & 2^{N^2-N+1} \\ \vdots & \ddots & \ddots & \vdots \\ 0 & 4 & \cdots & 2^N \end{bmatrix}$$

6. 模型的评估

6.1 模型重述

我们依据概率模型，对于在每一个决策分支点处，我们求解“在A_k格局处，在选择S_*滑动操作的条件下，通过随机操作最终到达含有 2048 的最终格局”这一事件的概率来确定较优操作。按照 Monte-Carlo 随机模拟方法，对概率量进行统计学估计。引入估价函数$Ev(A_k)$,判断每一次独立随机操作游戏在给定步数处的中局或失败终局状态与含有 2048 终局的接近程度，作为达到 2048 频率（概率）的有效替代。利用 matlab 编程，实现了一定抽样数量和一定模拟深度条件下的模型求解。在 n=120，t=60 的强度下，得到了 70%的胜率。

6.2 模型的优点

- 直截性：Monte-Carlo 方法基于概率模型。与传统博弈树算法不同，不需综合考虑各种评价局势好坏的因素；不需调整各种权重，构造复杂、精细的估价函数。
- 运算线性：随机决策的模拟实现基于伪随机数生成器，同时我们构造的估价指标仅仅是局面矩阵中所有数值的求和，并不涉及非线性运算；在 matlab 中运行起来相对较为高效。
- 简洁性：Monte-Carlo 随机模拟的实现并不复杂，代码量少，需要控制的参数仅有 n、t 两个，十分简洁。

- 普适性：本模型可以广泛应用与各种博弈游戏。比如象棋、围棋、五子棋等。[4]事实上，目前基于 Monte-Carlo 随机模拟的围棋 AI 的核心主体部分是对于棋盘上的每一个着点向后进行随机落子模拟。以这种思想辅以其他剪枝、学习等助于提高效率的调整后，能够高效低运行，战胜多数其他 AI。此外，世界上最强的国际象棋 AI，IBM“深蓝”就是基于 Monte-Carlo 算法。
- 精度——效率可控性：任何算法都面临着精度和效率之间的权衡。本模型的精度对应其对格局判断的准确性，进而对应胜率。效率则对应思考时间。我们可以调整（n，t）来直接、简单地调节这种权衡。
- 精度分段：由于精度调节的方便，我们可以把整个游戏分段设定不同的精度值。我们写的程序就是基于这种方法。当盘面内数值较小，空格较多的前期，采用低精度首先迅速达到一个 1024。之后加强精度，放慢思考来达到 2048。这和人类玩家玩这个游戏时的做法十分类似：在前期，人类玩家往往不会细细思考；在后期，他则会向后看许多步，精心规划移动路线。这种分段处理进一步提升了程序运行效率。

6.3 模型的缺点

- 概率模型粗糙性：随机模拟模型不精细。比起确定的、考虑各种评价因素的估价函数，我们对局面的评价基于随机的重复试验。对于同一个局面，多因素估价函数给出的分数是确定的，我们模拟给出的分数每次都不同。这种波动有时掩盖了细小的局势优劣差别，随着累积，导致胜率受到影响。
- 设备局限性：我们在现有设备上运行（120，60）强度的程序已经较为缓慢，进一步提高精度对效率的牺牲较大。或许程序仍然需要进一步的优化。

7. 参考文献

［1］上海交通大学数学系 武爱文；冯卫国；卫淑芝；熊德文，《概率论与数理统计》，上海：上海交通大学出版社，2011

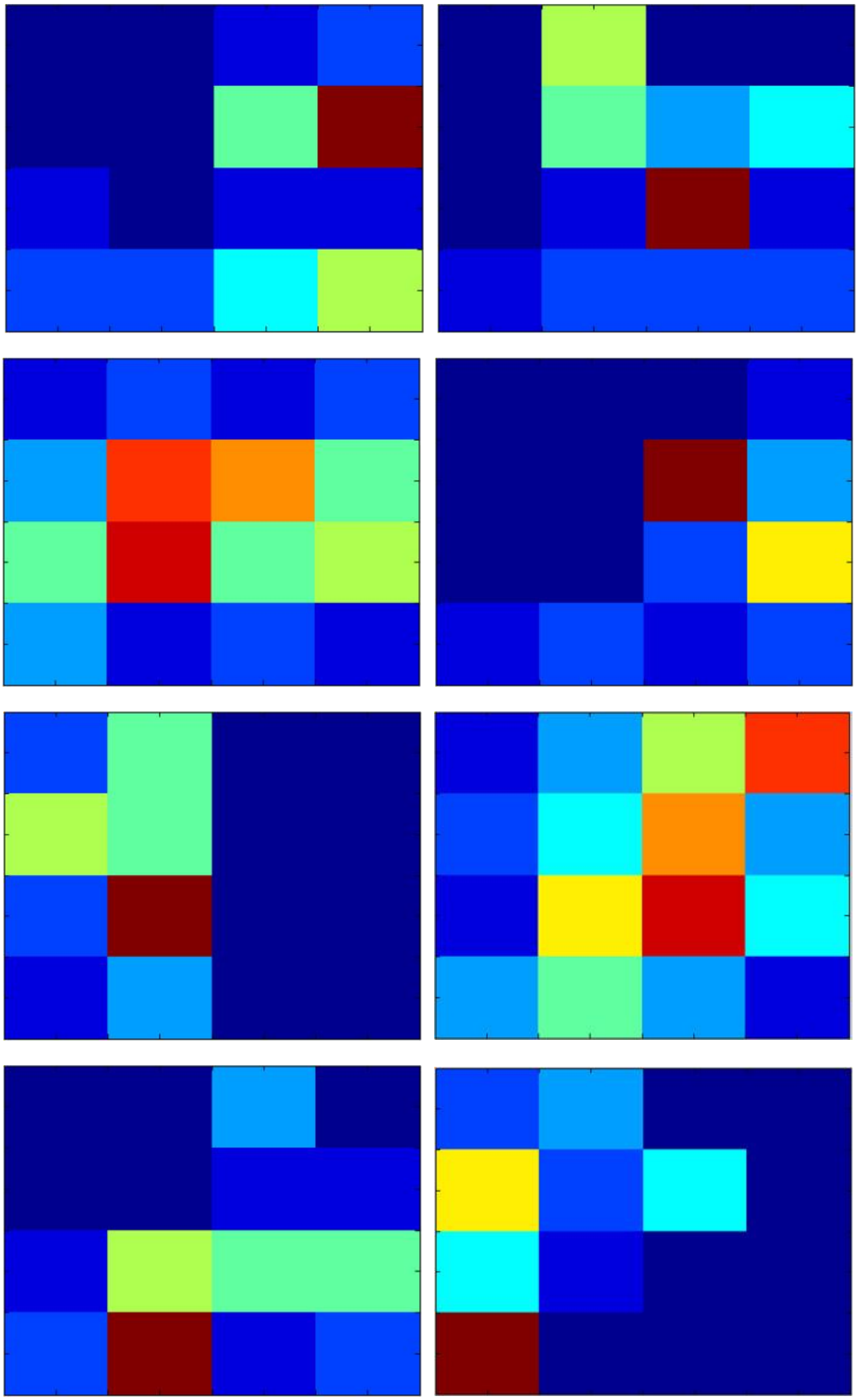
[2] Drew Fudenburg，*Game Theory*，Massachusetts Institute of Technology：UNKNOWN，1991

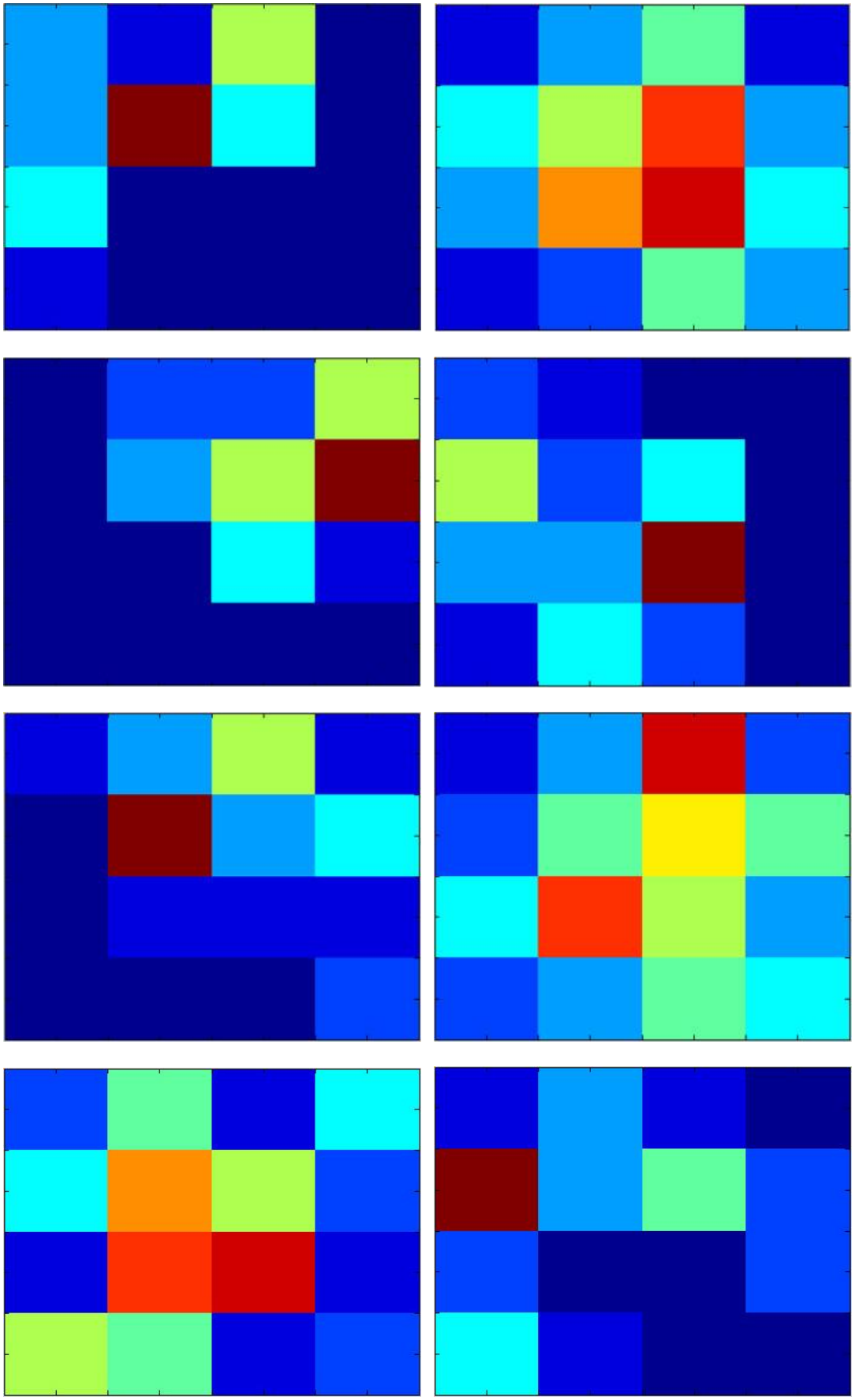
[3] Wikipedia. Monte Carlo method. http://en.wikipedia.org/wiki/Monte_Carlo_method. 2014/5/27

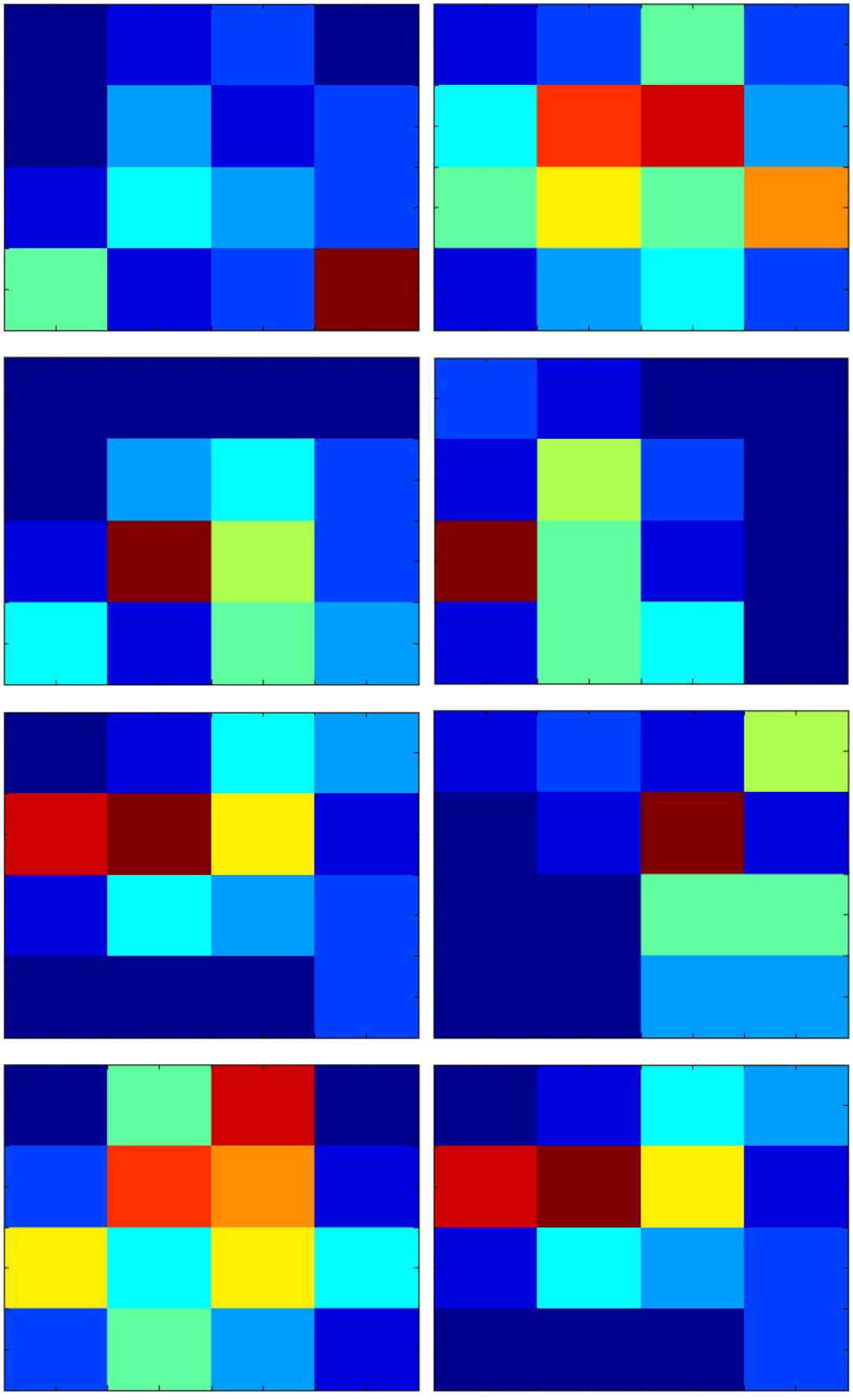
［4］Sylvain Gelly, Joanna Jongwane。蒙特卡洛方法在计算机围棋中的应用[J]。程序员，2008（12）。84 - 87。2008

[5] Dingyu Xue ; YangGuan Chen，*Solving Applied Mathematical Problems with MATLAB®*，United States：CRC Press，2008

附录 – 1：20 次 AI 运行结果，n=120，t=60







附录 - 2：Matlab 程序

```
%--------------第一部分：游戏主体--------------

%--------1-1 向上滑动函数，（其他三个方向类似，不再列出）--------
%---处理顺序：移动前合成——移动——移动后判别是否已经合成——若无，进行移动后合成
function [anew] = slideup(a)
%--------子函数一，控制数字沿移动方向穿过一个空格--------
function [a] = movestep(a,j)
    for i2 = 1:3
        if a(i2,j)==0 && a(i2+1,j)~=0
           a(i2,j) = a(i2+1,j);
           a(i2+1,j)=0;
        end
    end
end
%--------子函数二，控制移动后合成--------
function [a] = secondCollide(a,c,j)
    for i3 = 1:3
        if a(i3,j)==a(i3+1,j) && a(i3+1,j)~=0 && c(j)==0
           a(i3,j) = a(i3,j)+1;
           a(i3+1,j)=0;
           c(j) = 1;
        end
    end
end

%--------主体--------
c = [0,0,0,0];
    for j = 1:4
        for i = 1:3
            if a(i,j)==a(i+1,j) && a(i+1,j)~=0
               a(i,j) = a(i,j)+1;
               a(i+1,j)=0;
               c(j) = 1;
            end
        end %---以上控制移动前合成
        a=movestep(a,j);
        a=movestep(a,j);
        a=movestep(a,j);
        a=secondCollide(a,c,j);
    end
    anew = a;
end
```

```
%--------1-2 随机移动函数--------
function [anew] = randMove(a)
          rdmove2 = ceil(4*rand(1,1));
          move2=rdmove2(1,1);

    if move2==1
       anew = slideup(a);
    elseif move2==2
       anew = slidedown(a);
    elseif move2==3
       anew = sliderright(a);
    elseif move2==4
       anew = slideleft(a);
    end
end

%--------1-3 随机生成函数--------
function [anew] = randDrop(a)
    zeroPosition = find(a==0);
    dropind = ceil(length(zeroPosition)*rand(1,1));
    dropPosition = zeroPosition(dropind);
    rddrop1 = ceil(4*rand(1,1));
    if rddrop1 == 1
      a(dropPosition)=a(dropPosition)+2;
    else
      a(dropPosition)=a(dropPosition)+1;
    end
    anew = a;
end

function [highscore] = montecarloSIMU(a)

%------------第二部分，蒙特卡洛模拟------------
%--------子函数一，随机移动--------
function [anew] = rdmove(a)
    rdmove2 = ceil(4*rand(1,1));
    move2 = rdmove2(1,1);
    if move2==1
       anew = slideup(a);
    elseif move2==2
       anew = slidedown(a);
```

```
    elseif move2==3
        anew = slideright(a);
    elseif move2==4
        anew = slideleft(a);
    else
        anew = slidedown(a);
    end
end

%--------子函数二，随机生成--------
function [anew] = rdDrop(a)
    zeroPosition = find(a==0);
    dropind = ceil(length(zeroPosition)*rand(1,1));
    dropPosition = zeroPosition(dropind);
    rddrop1 = ceil(10*rand(1,1));
    if rddrop1 == 1
      a(dropPosition)=a(dropPosition)+2;
    else
      a(dropPosition)=a(dropPosition)+1;
    end
    anew = a;
end


%-----------主体-----------
a2 = 2.*ones(4,4);
ainit = a;


if max(max(a)) <= 9 %---游戏前期（1024之前），精度设定较低
    score1 = zeros(1,50);
    for n =1:50      %---样本数

        a = ainit;
        for t = 1:25  %---模拟步数
            if min(min(a))==0
                a = rdDrop(a);
                a = rdmove(a);
            elseif min(min(a))>0
                a = rdmove(a);
            end
            if min(min(a)) > 0 & slideleft(a)==a & slideright(a)==a &
slideup(a)==a & slidedown(a)==a ,break
            end;
```

```
        end

        score1(n) = sum(sum(2.^a));

    end
    highscore = sum(score1)  %---评分指标

elseif max(max(a)) > 9  %---游戏中后期（1024之后），精度设定较高
    score2 = zeros(1,90);
    for n =1:90          %---样本数
        a = ainit;
        for t = 1:45     %---模拟步数
            if min(min(a))==0
                a = rdDrop(a);
                a = rdmove(a);
            elseif min(min(a))>0
                a = rdmove(a);
            end
            if min(min(a)) > 0 & slideleft(a)==a & slideright(a)==a &
slideup(a)==a & slidedown(a)==a ,break
            end
        end
        score2(n)=sum(sum(2.^a));
    end
    highscore = sum(score2); %---评分指标


end
end
```

%----------第三部分，决策控制函数（主函数）----------

```
function [anew] = animation2048_Montecarlo()
%--------sub function1--------
    function [anew] = animation_choice(a)
          hsU = montecarloSIMU(slideup(a));
          hsD = montecarloSIMU(slidedown(a));
          hsR = montecarloSIMU(slideright(a));
          hsL = montecarloSIMU(slideleft(a));
          hs = [hsU,hsD,hsR,hsL];
          choice = find(hs==max(hs));
          if choice == 1
              a = slideup(a);
          elseif choice == 2
```

```
            a = slidedown(a);
        elseif choice == 3
            a = slideright(a);
        elseif choice == 4
            a = slideleft(a);
        end
        anew = a;
    end

%--------main--------
    clc,clear
    step = 0;
    a = zeros(4,4);  %---游戏格局矩阵
    for k = 1:3000   %---预设游戏最大步数
        if min(min(a))==0
           step = step + 1;
           a = randDrop(a);
           a = animation_choice(a);
           imagesc(a)
           set(gca,'Clim',[0,11])
           pause(0.01)
        elseif min(min(a))>0
           step = step + 1;
           if a ~= animation_choice(a)
              a = animation_choice(a);
           else
              a = randMove(a);
           end
           imagesc(a)
           set(gca,'Clim',[0,11])
           pause(0.01)
        end
        if min(min(a)) > 0 & slideleft(a)==a & slideright(a)==a &
slideup(a)==a & slidedown(a)==a ,break
        end
        anew = a;
    end
end
```