2014

Game: 2048

PAIR PROJECT

ZHAOFAN QIU & QING LI

MSRA | BJX #2

一、 时间安排

本次课程的 Pair-Project(结对编程)的项目定题为程序完成 2048 游戏，基本要求为利用程序来进行 2048 的游戏，取得尽量高的分数，编写时间为 2 周，为了充分利用这两周时间，进行时间安排如下：

时间	事宜	内容
9.16~9.19	游戏调研	调研 2048 游戏的基本策略，人工总结 2048 游戏方法。
9.20 ~ 9.23	算法调研	调研程序进行 2048 游戏的算法，以互联网检索为基础，总结常见算法。
9.24 ~ 9.27	算法编写	选取调研中较好的算法进行编写。
9.28~9.30	性能提升	提升算法性能并撰写报告。

其中调研花费的时间较长，原因如下：

1、 2048AI 程序编写较为简单，主要的难度应该在于算法的设计和选取，所以需要大量的时间进行调研。

2、 平时还有部分 mentor 布置的任务需要完成，故需要将课程代码时间集中快速完成，以免两者互相拖慢进度。

3、 调研过程中同时完成代码框架设计和接口定义的工作，为后期代码编写过程和最后代码的可读性将会带来极大的便利。

最终在实现过程中基本按照以上时间安排进行。

在多方比较之下，选择了启发式搜索作为最终的算法，由于该问题规模较小，使用启发式搜索可以获得相当理想的性能。

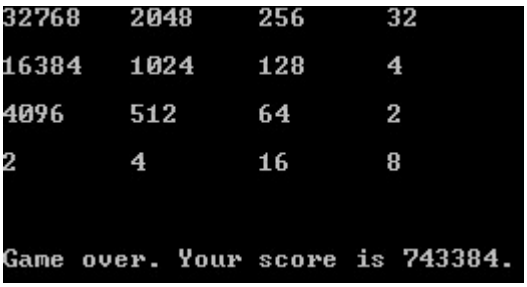
由于进行一次游戏测试的时间较长，无法进行大量测试，连续测试十次得到结果如下：

步数	最大块	得分
14736	16384	385360
14364	16384	377716
14875	16384	387172
22268	32768	629736
14887	16384	387240
13083	16384	343152
22332	32768	629992
14910	16384	387260
14039	16384	367472
22270	32768	629764
(average)16776.4	21299.2	452486.4

目前测试最好的效果如下：

最大块	最高分
32768	743384

最终局面如下：



(可以看出上述测试界面为自行编写的命令行版界面，可以极大的提升速度，否则，使用网页版进行一次完整的至少需要一天。。。故以下所有的测试均为使用命令行版测试的结果)

二、 课题调研&算法尝试

由于本次课题的目标为游戏 AI 设计，所以调研分为两部分，第一部分是人工进行游戏的策略调研，第二部分是程序实现 AI 的算法调研。由于游戏策略较多而且繁杂，在此处主要针对 AI 算法的调研。调研并尝试编写程序的方法如下：

1、随机移动

使用随机策略进行 10 局游戏，得到无 AI 的随机移动性能如下：

步数	最大块	得分
128	128	1204
78	64	572
94	64	704
261	256	3204
163	128	1656
136	128	1324
56	32	308
124	16	128
135	128	1332
84	64	704
(average)125.9	100.8	1113.6

可以看出，随机移动平均在 125 步左右就会 game over，较差的情况甚至 56 步就 game over。同样最大的块也只能到 256，平均分则在 1113.6 分。效果十分不好，但是可以作为接下来改进算法的出发点。

2、随机 Monte Carlo 方法

使用蒙特卡洛方法解博弈问题（2048 类似博弈，但又不完全相同）的基本方法思想是，如果当前局面很有利的话，后面抛弃 AI 使用简单规则继续游戏也可以取得很好的得分，所以考虑使用以下步骤设计 AI：

1、 从当前局面开始；

2、 选择某一个方向进行下一步；

3、 从新局面开始 N 局以随机移动为策略的游戏，并将这 N 局的平均分作为选择该方向走的评分。

4、 从四个方向中选择蒙特卡洛评分最大的一个。

蒙特卡洛方法思想简单，不需要任何人工规则的干预，且极具可扩展性，需要提高性能，只需要增加机器性能并增加 N 即可。但是这样对于硬件计算能力的要求较高，随机式的蒙特卡洛难以实现接近实时的判断下一步的方向，也难以进行测试。

3、人工评分

与随机蒙特卡洛方法相比，人工评分速度非常快，但是由于人工规则的局限性，性能并比不上蒙特卡洛方法。具体的规则有：

1、 越多空格评分越高；

2、 大的方块在边缘评分高；

3、 每行每列应具有单调性；

4、 可合并方块越多分越高。

通过调整参数综合以上四个规则可以得到一种移动策略，性能如下：

步数	最大块	得分
193	128	1904
329	256	4000
369	256	4464
363	256	4420
162	128	1548
568	512	7948
285	256	3356
192	128	2056
495	512	6852
332	256	4040

(average)328.8	268.8	4058.8

可以看出，通过这些简单的规则进行移动，效果已经获得了很大的提升，平均分已经从随机的 1113.6 提升到了 4058.8，最大方块也已经出现了 512。但是人工评分的效果还并不理想，因为这样的评分并不存在任何的前瞻性，或者说启发性，无法看到这一步以后的东西。

4、启发式 Monte Carlo 方法

很自然地，结合人工评分作为启发函数，结合蒙特卡洛方法，可以很快得到具有前瞻性的移动策略，而且在实现上十分简单，基本步骤如下：

1、 从当前局面开始；

2、 选择某一个方向进行下一步；

3、 从新局面开始 N 局以人工评分为策略的游戏，并将这 N 局的平均分作为选择该方向走的评分。

4、 从四个方向中选择蒙特卡洛评分最大的一个。

与随机蒙特卡洛相比，启发式的蒙特卡洛将之前的随机策略替换为使用人工评分为随机采样的策略，这样和之前相比，蒙特卡洛的采样更加具有方向性，而人工的评分也会更具前瞻性。

同样蒙特卡洛最大的影响因素是运行效率，所以为了实现该算法设计一系列运行效率提升的手段，包括预处理、位操作、并行化等，在具体实现中会具体讲述。

采用上述策略同样进行测试，得到效果如下：

步数	最大块	得分
1793	2048	34720
1649	2048	27884
3498	4096	76632
2612	4096	57040
1379	2048	26440
1891	2048	36304
1723	2048	33088
1034	1024	17456
1832	2048	35592
1876	2048	36180
(average)1928.7	2355.2	38133.6

可以看到，使用带有启发式的蒙特卡洛方法后，平均得分一下从4058.8跃升至38133.6，最大的方块也可以达到4096，而且以极大的概率达到2048，性能已经比较理想，如果运算速度可以上升，提升蒙特卡洛随机的次数，理论上可以继续提升算法性能。

5、启发式搜索

启发式搜索是针对该问题最有效且最难以实现的算法，相比蒙特卡洛的完全随机评分，启发式搜索近可能地去穷尽接下来可能发生的情况，并依据我们可以预测的未来的可能性中选择一个对自己最有利的情况。

想要写出理想的启发式搜索，设计启发式函数设计、剪枝规则设计、得分传递方式设计、参数细节设计等多个方面，算法设计十分复杂，但是如果以上设计都理想地完成，在效果上启发式搜索目前很难被其他算法超越。在这几个方面，选择方案如下：

1. 启发式函数设计:启发式函数仍使用上述定义的人工评分标准，即空格数、极大边缘、单调性、可合并性。
2. 剪枝规则：发生概率，减去发生概率较小的枝。
3. 得分传递方式：移动步骤取最大方向，随机生成方块步骤选取期望（均值）。

值得一提的是，在调研得到的各种算法中，启发式函数和剪枝规则并没有什么大的不同，但是得分传递方式却又很大区别。存在两种选取方式：

1. 期望式：电脑随机的因素考虑得分的期望。
2. 博弈式：电脑随机的因素考虑得到得分最少的情况。

很明显，我们选用的得分传递方式是期望式的，理由是期望是的得分传递方式最符合我们对随机事件的理解。但是如果采用博弈式，假设电脑随机过程存在智能，可以选择对玩家最不利的随机方式，会使得算法过于保守，很多情况下过于担心小概率事件的出现。同样博弈式也存在好处，即可以使用 alpha-beta 剪枝，极大的加快算法效率。权衡之下，我们仍选择期望式，在实际测试中期望式也的确比博弈式性能更优。

启发式搜索效果如下：

步数	最大块	得分
14736	16384	385360
14364	16384	377716
14875	16384	387172
22268	32768	629736
14887	16384	387240
13083	16384	343152
22332	32768	629992
14910	16384	387260
14039	16384	367472
22270	32768	629764
(average)16776.4	21299.2	452486.4

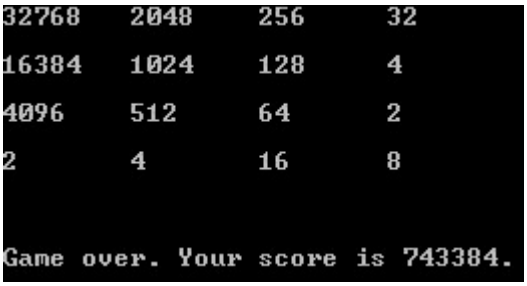
可以看到使用启发式搜索之后，性能有了极大的提升，10 局的平均得分达到了 452468.4 分，最大的方块达到了 32768（2 的 15 次方）已经达到了目前使用的数据表示方式的上限（2~2 的 15 次方），理论上已经很难再进行提升。

但是这种算法目前在速度上比较慢，如果需要进一步提高，只能提升算法执行效率，得分上已经有所提升。

目前测试最好的效果如下：

最大块	最高分
32768	743384

最终局面如下：



三、具体实现

根据老师对于项目的要求，以及调研的结果，决定使用 C#实现启发式搜索算法（其他算法其实也实现了），并将模块设计分为以下几个部分：

1、 局面处理部分；

2、 模拟运行部分；

3、 启发式函数部分；

4、 启发式搜索部分；

其中局面处理和模拟运行部分主要是为了加速对于局面的处理，提高检索过程中局面变化的速度和测试的速度；而启发式函数和启发式搜索则负责对下一步进行预测。

1、局面处理部分

为了对局面加速进行处理，首先对局面进行压缩存储。

数据格式：每一个局面为有序的 16 个无符号数，假设其范围为 0~32768，0 表示没有方块，如此一来，可以使用一个 4 位无符号整型 0x0 ~ 0xF 表示一个方块，其中 0x0 表示没有方块，0x1 ~ 0xF 则表示 2 的次方数。很自然的 16 个数的局面，就可以表示为一个 64 位无符号整数，即 UInt64。使用一个 UInt64 表达一个局面，在当前 64 位系统中可以极大的加快数据传

递和处理的速率，并藉由位运算可以进一步提升处理效率。同时每一行（列）可以使用 UInt16 来表示。

预处理：有以下几个内容需要在预处理阶段进行，以提升算法速度。

1. 某一行左（右）移后的结果；
2. 某一列上（下）移后的结果；
3. 某一行（列）的系统得分；
4. 某一行（列）的启发式评分。

有了以上四点预处理的结果，在进行局面移动和评分时，只需要查表即可得出结果。

局面处理部分提供的主要接口如下：

- 局面初始化
- 局面移动
- 指定位置插入
- 随机插入
- 局面系统得分

2、模拟运行部分

模拟运行部分使用局面处理部分的接口，模拟游戏运行的过程，过程中，留出接口调用启发式搜索的返回函数。由于局面处理部分性能较高，而且自己编写的模拟运行程序基于命令行不存在动画，故运行效率远高于网页版，便于进行算法测试。

3、启发函数部分

启发式搜索使用的启发函数与上述算法中的人工评分并无区别，基于空格数、极大边缘、单调性、可合并性。

4、启发式搜索部分

启发式搜索实现的要点在于：

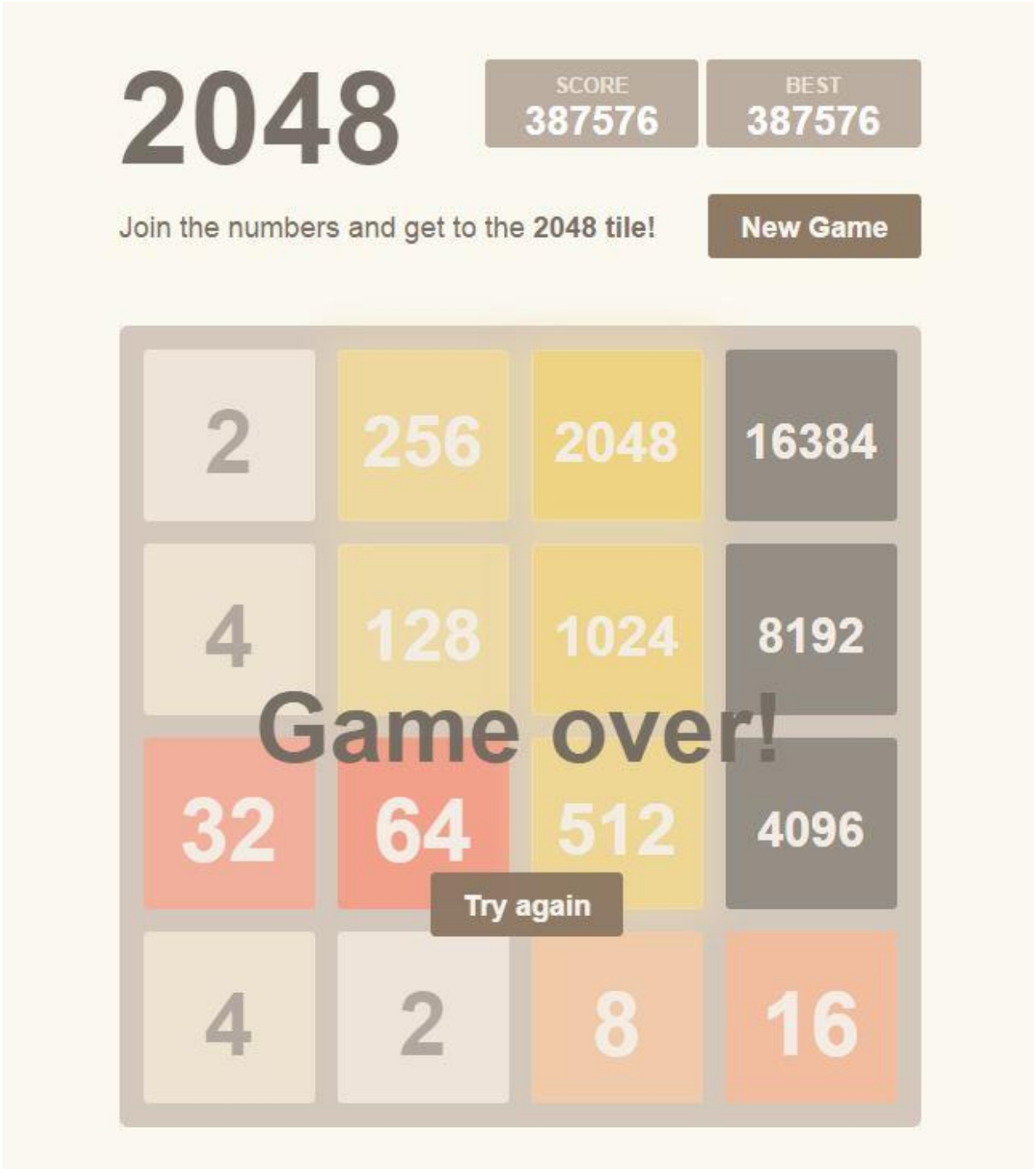
1. 减去发生概率较小的分支。
2. 并行计算四个方向的搜索。
3. 建立查找表，记录每次搜索过程中搜索过的局面，直接得到上次搜索的结果。
4. 使用期望最大化的得分传递方式。

四、最终测试结果展示

算法测试结果

步数	最大块	得分
14736	16384	385360
14364	16384	377716
14875	16384	387172
22268	32768	629736
14887	16384	387240
13083	16384	343152
22332	32768	629992
14910	16384	387260
14039	16384	367472
22270	32768	629764
(average)16776.4	21299.2	452486.4

使用网页版运行一次结果展示


2048
SCORE
387576
BEST
387576
Join the numbers and get to the 2048 tile!
New Game
2
256
2048
16384
4
128
1024
8192
Game over!
32
64
512
4096
Try again
4
2
8
16