南京邮电大学

算法与数据结构设计报告

（ 2023 / 2024 学年 第 一 学期）

题　　目： 2048 游戏及其 AI 算法

专　　业	理工科强化班
学 生 姓 名	陶陆天
班 级 学 号	Q21010219
指 导 教 师	陈兴国
指 导 单 位	数据科学与工程系
日　　期	2023.10.30-2022.11.10

支撑指标点	评价准则	计分（每项10分）
课程目标1：文献调研与资料收集能力，问题发现、研究、分析与解决能力（20分）	1、能够掌握算法与数据结构设计的相关基础知识，并能够针对求解的工程问题，收集资料进行合理的分析与设计	
	2、通过调研，能够选择合适的程序设计语言与编程开发平台，对求解的工程问题进行编程实现	
课程目标2：通过课程设计，培养学生综合应用算法和数据结构等知识解决工程问题的实践能力（20分）	3、能够给出数据结构和算法的设计描述，给出关键算法的流程图或伪代码，并给出各算法之间的结构关系描述	
	4、具备一定的人机交互设计意识，人机交互设计合理、友好，操作简便	
课程目标3：培养解决工程问题的开发工具运用能力，能够利用程序设计软件或系统对问题求解进行模拟和实现，能够设计测试数据验证问题解决方法的正确性，并能够对问题解决方法的性能和效率进行分析（40分）	5、具备一定的算法与数据结构设计分析能力，能够完成课题要求的各项任务和指标	
	6、能够结合计算机软硬件资源，合理选用算法、数据结构、数据存储方式等技术手段，对求解的工程问题进行有效建模和求解	
	7、具备一定自学能力与探索创新意识，能够充分利用教科书及其资源（如网络等）自学新知识与新技能	
	8、掌握调试方法与工具，对程序开发过程中出现的问题进行分析、跟踪与调试，并能够进行充分测试	
课程目标4：选择同类课题的学生能够通过讨论和交流解决课程设计中的难题，能在实验报告中准确阐述课程设计的内容，能够清晰陈述观点和回答问题（20分）	9、能够正确、完整地回答指导教师关于课题的问询，反映其对课题内容，以及相关的工程基础知识具有较好的理解和掌握	
	10、具备一定的语言表达能力与文字处理能力，能够结合复杂工程问题撰写报告，报告内容和实验数据详实，格式规范	
算法与数据结构设计能力测评总分		

指导教师： 陈兴国　　2023 年 11 月 12 日

备注：

具体课题题目

一、课题内容和要求

本课题目标系统“2048 运行系统”的功能框架图如图 1 所示。

图 1 2048 运行系统

（1）该系统分为两种模式，一种是单人运行模式；另外一种是 AI 自动运行模式。

（2）对于 AI 自行运行模式，我们在运行前先设置深度优先搜索的深度和广度。在搜索过程中，我们对每一次的结果进行评估打分，之后返回当前状态下的最佳方向并且向此处移动。我们利用蒙特卡洛算法结合启发式算法对路径进行评估。

（3）支持根据需求调整出现 2 和 4 的概率。

（4）支持在游戏结束后直接重新开始。

（5）个人分工：

1024*2 小组 （陶陆天，陈子祥，王颢楷）	分工
陶陆天	game_function.py 主要函数（以程序主体为主，包括移动函数，合并函数，随机放置函数等，详见下） 使用单个函数进行移动和合并操作，让修改代码更加方便 提出 vis 数组以给合并单元格使用特效 参与 AI 算法和启发式算法讨论 参与启发式算法中评估函数的撰写（包括平滑性，空格数量，合并优先等方面）

二、数据结构说明

```
board = np.zeros(NUMBER_OF_SQUARES, dtype="int")        # 棋盘为 4*4 的矩阵
NEW_TILE_DISTRIBUTION = np.array([2, 2, 2, 2, 2, 2, 2, 2, 2, 4])     #2,4 出现的概率
                                                                      数组
class Display(Frame):                    # 程序主体结构：
    def __init__(self):
        Frame.__init__(self)
        self.master.title('2048')
        self.master.bind("<Key>", self.key_press)
        self.bind('<Button>', self.key_press)
        self.mainloop()
gamegrid = Display()
```

三、算法设计

1 初始化游戏界面算法

创建了一个 4*4 的 board，用来存放每个 2048 每个格子的数字，并且随机生成了两个数字以完成初始化。

代码如下：

```
def initialize_game():
    board = np.zeros(NUMBER_OF_SQUARES, dtype="int")
    # vis = np.zeros((CELL_COUNT, CELL_COUNT), dtype="int")
    initial_twos = np.random.default_rng().choice(NUMBER_OF_SQUARES, 2,
replace=False)  # 在棋盘中选择两个不同的随机数
    board[initial_twos] = 2
    board = board.reshape((CELL_COUNT, CELL_COUNT))  # 按照行优先顺序进行
重构
    # ndarray.reshape 是 Numpy 数组对象的一个方法，用于重新构造数组的形状。该
方法的语法如下：
    # ndarray.reshape(shape, order='C')
```

order：可选参数。用于指定元素存储的顺序，可以是’C’，‘F’，或 ‘A’。默认为 ‘C’。

其中，'C’按行优先顺序存储元素，'F’按列优先顺序存储元素，'A’表示让 Numpy 自动使用最优的方式。

return board　#返回初始化后的 board 数组

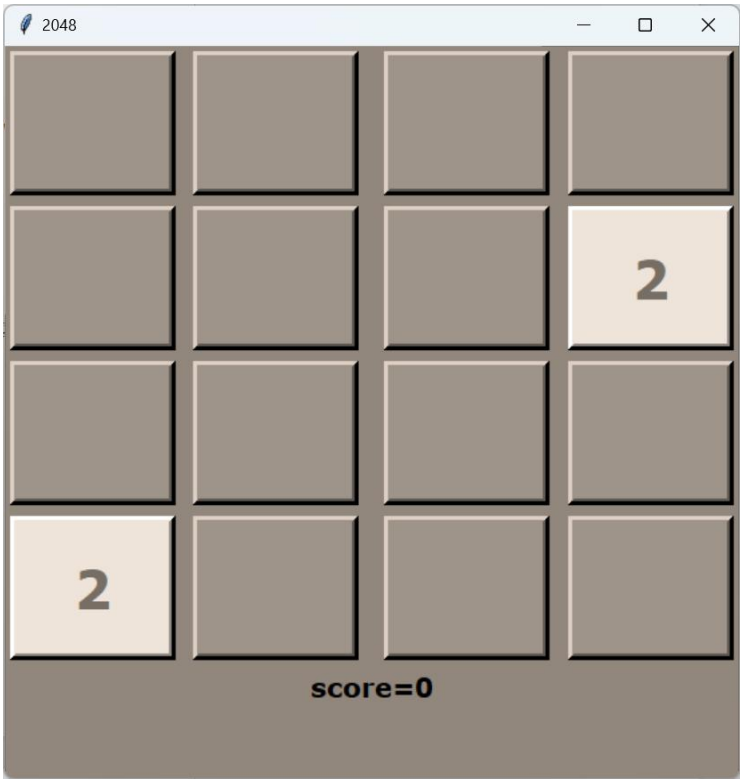

图 2 初始化后界面

2 移动完随机放置 2or4 算法

在每一步移动完之后，从 NEW_TILE_DISTRIBUTION（即 2，4 出现概率数组）中随机挑选一个数字随机生成在一个空白的格子中，并返回生成后的数组。

代码如下：

```
def add_new_tile(board):  # 移动完随机放置 2or4 的方法
    tile_value=NEW_TILE_DISTRIBUTION[np.random.randint(0,
len(NEW_TILE_DISTRIBUTION))]
    # 从该列表随机抽取一个数，个数为权重.2 与 4 的个数占权重为 9：1
    tile_row_options, tile_col_options = np.nonzero(np.logical_not(board))
    # np.logical_not(board)对数字版取反，非零为 False，零为 True
    # np.nonzero(np.logical_not(board))返回 True 的行列索引，传递至 tile_row_options,
```

```
tile_col_options
        tile_loc = np.random.randint(0, len(tile_row_options))   # 随机找个位置放置
tile_value（2or4）
        board[tile_row_options[tile_loc], tile_col_options[tile_loc]] = tile_value   # 放置
    return board
```

3 将数组中非零数字向右移动并不进行合并的算法

为了简化函数数量，我们将向上向下向右和向左移动统一修改为对 board 数组分别进行向右旋转 90°，向左旋转 90°，不旋转和向右旋转 180°之后执行此函数，之后再旋转回去的操作，如此便能将 4 个函数统一成一个，方便书写和修改。

该函数中只会将 board 中的数字向右移动到最右边，不会进行合并操作，合并操作将在下面的 merge_elements(board)函数中被进行。

代码如下：

```
def push_board_right(board):   # 向右移动的方法（不进行合并）
    new = np.zeros((CELL_COUNT, CELL_COUNT), dtype="int")
    #   重新生成一个 4*4 的数组用来存放移动后的 board
    done = False
    #   能否进行移动
    for row in range(CELL_COUNT):
        count = CELL_COUNT - 1
        for col in range(CELL_COUNT - 1, -1, -1):
        # 从 CELL_COUNT - 1 开始迭代，直到-1（不包括-1），步长为-1
            if board[row][col] != 0: #   这个格子上有数字
                new[row][count] = board[row][col]
                if col != count: #   可以移动
                    done = True
                count -= 1
    return new, done
    #返回新 board 和能否进行移动的 done
```

函数功能样例如下：

2		2		操作后 ——>			2	2
	4		8				4	8
16								16
	8	8					8	8

4 将数组中相邻的相同的非零数字向右合并的算法

在执行完成 push_board_right(board)之后，执行该函数，以达到将相邻的相同数字（即移动过程中相遇的相同数字）合并的效果，但此函数合并后的数字有可能造成出现中间的空格（详见下示例），故此函数执行完成后应该再度执行上个 push_board_right(board)函数使得空格消除，并在该两函数执行完成之后执行旋转操作使得 board 变成下一个我们所需的状态。

代码如下：

```
def merge_elements(board):    # 合并相同的格子
    score = 0
    done = False
    vis = np.zeros((CELL_COUNT, CELL_COUNT), dtype="int")    # 格子是否被合并
    for row in range(CELL_COUNT):
        for col in range(CELL_COUNT - 1, 0, -1):    # 从 CELL_COUNT - 1 开始迭代，直到 0（不包括 0），步长为-1
            if board[row][col] == board[row][col - 1] and board[row][col] != 0:
                # 找到了像同行的相邻的相同数字，即可被合并的数字
                board[row][col] *= 2 # 数字合并
                score += board[row][col] * 4
                board[row][col - 1] = 0 # 上个数字变成 0
                vis[row][col] = 2    # 2 是合并
                done = True
            elif board[row][col] != board[row][col - 1] and board[row][col] != 0:
                vis[row][col] = 1    # 有数字且没合并
    for row in range(CELL_COUNT):
        for col in range(CELL_COUNT - 1, 0, -1):
```

```
                if board[row][col] == 0:
                    score += 2 ^ 8 + 1
            return board, done, score, vis   # 传出参数
            # 返回执行完成操作后的 board，能否合并的 done，得分 score，和被合并的格子数组 vis
```

函数功能样例如下：

2	2	2	2	操作后——>		4		4
		4	8				4	8
			16					16
		8	8					16

5 向上移动的算法

在玩家输入 w 字符或是 AI 使用蒙特卡洛算法结合启发式算法评估向上移动方案的得分时进入此函数，使用先旋转(board = np.rot90(board, -1))后移动（push_board_right(board)）再合并(merge_elements(board))然后再移动（push_board_right(board)）以消除空白格子的方法移动并合并相同格子，最后旋转回来(board = np.rot90(board))并记录被合并的格子（vis 数组），把合并后的 board，能否成功移动 move_made，移动后的得分 score，被合并的格子 vis 传回到原本的代码中来进行覆盖（玩家移动）或是进行判断比较（AI 判断或是移动）。

代码如下：

```
def move_up(board):   # 向上移动的方法（包含合并）
    board = np.rot90(board, -1)   # 顺时针旋转 90°把向上变成向右
    board, has_pushed = push_board_right(board)
    #   push_board_right(board)函数能够将 board 中存放的非 0 数字向右移动，等待被合并
    board, has_merged, score, vis = merge_elements(board)
    #     merge_elements(board)函数能够将 board 中存放的相邻的相同数字进行合并，并且返回合并后的数组 board，是否产生合并 has_merged，得分 score 以及被合并的格子数组 vis。
    board, _ = push_board_right(board)   # 将合并后的格子转移到最右
    vis, _ = push_board_right(vis)    # 将表达每个格子是否发生合并的 vis 数组也转移
```

到最右，即 vis 跟着 board 一起动

```
board = np.rot90(board)  # 逆时针旋转 90°变回向上
vis = np.rot90(vis)  # vis 也同样跟着 board 一起动
move_made = has_pushed or has_merged  # 判断是否移动或者合并
return board, move_made, score, vis
```

返回向上移动后的 board，能否移动的 move_made，移动后的得分 score，判断合并数组 vis

函数功能样例如下：

2	8			操作后 ——>	4	8	16	16
2			8		4	4		
2	4		8					
2		16						

函数功能实例如下图：

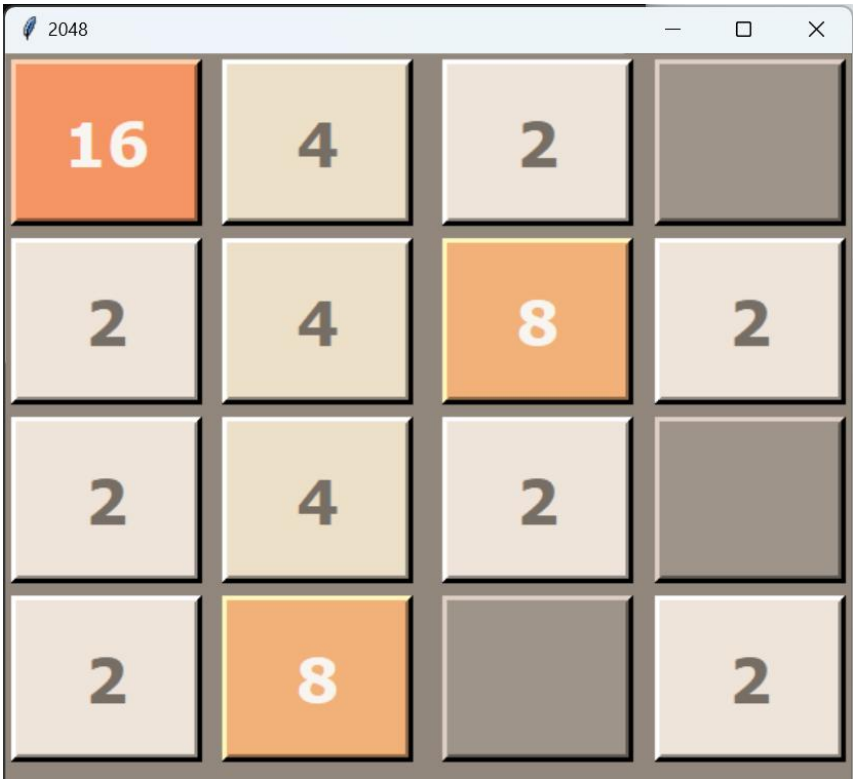

图 3 向上移动前

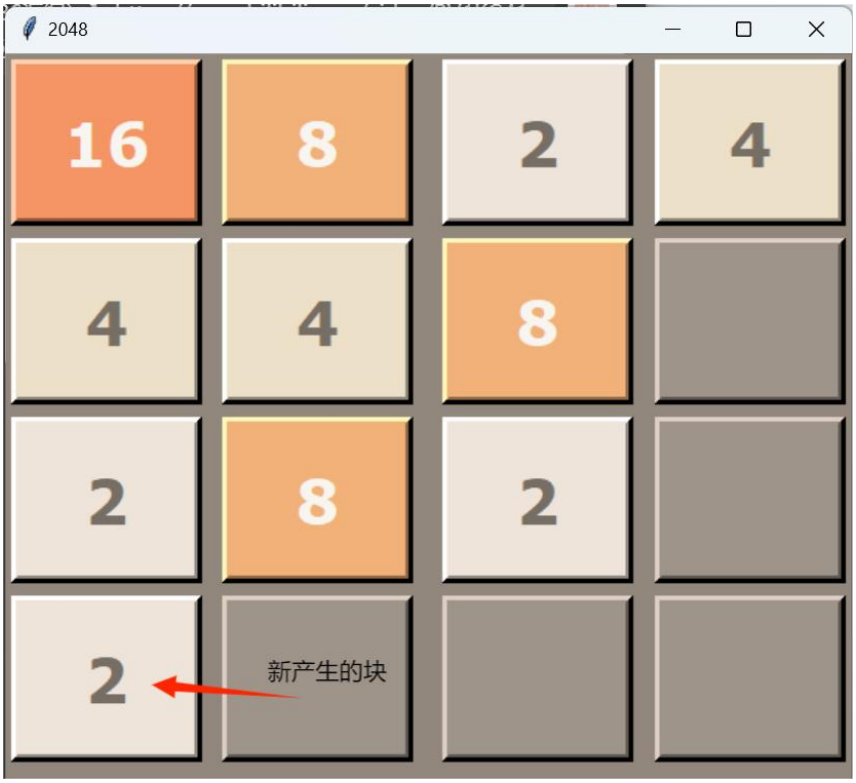

图 4 向上移动后

6 向下移动的算法

在玩家输入 s 字符或是 AI 使用蒙特卡洛算法结合启发式算法评估向上移动方案的得分时进入此函数，使用先旋转(board = np.rot90(board))后移动（push_board_right(board)）再合并(merge_elements(board))然后再移动（push_board_right(board)）以消除空白格子的方法移动并合并相同格子，最后旋转回来(board = np.rot90(board, -1))并记录被合并的格子（vis 数组），把合并后的 board，能否成功移动 move_made，移动后的得分 score，被合并的格子 vis 传回到原本的代码中来进行覆盖（玩家移动）或是进行判断比较（AI 判断或是移动）。

代码如下：

```
def move_down(board):  # 向下移动的方法（包含合并）
    board = np.rot90(board)  # 逆时针旋转 90°把向下变成向右
    board, has_pushed = push_board_right(board)
    #  push_board_right(board)函数能够将 board 中存放的非 0 数字向右移动，等待被合并
    board, has_merged, score, vis = merge_elements(board)
    #  merge_elements(board)函数能够将 board 中存放的相邻的相同数字进行合并，并且返回合并后的数组 board，是否产生合并 has_merged，得分 score 以及被合并
```

的格子数组 vis。

```
board, _ = push_board_right(board)  # 将合并后的格子转移到最右
vis, _ = push_board_right(vis) # 将表达每个格子是否发生合并的 vis 数组也转移
                                到最右，即 vis 跟着 board 一起动
board = np.rot90(board, -1)  # 顺时针旋转 90°
vis = np.rot90(vis, -1)     # vis 一起动
move_made = has_pushed or has_merged  # 判断是否移动或者合并
return board, move_made, score, vis
    # 返回向下移动后的 board，能否移动的 move_made，移动后的得分 score，
    判断合并数组 vis
```

函数功能样例如下：

2	8			操作后 ——>				
2			8					
2	4		8		4	8		
2		16			4	4	16	16

函数功能实例如下图：

图 5 向下移动前

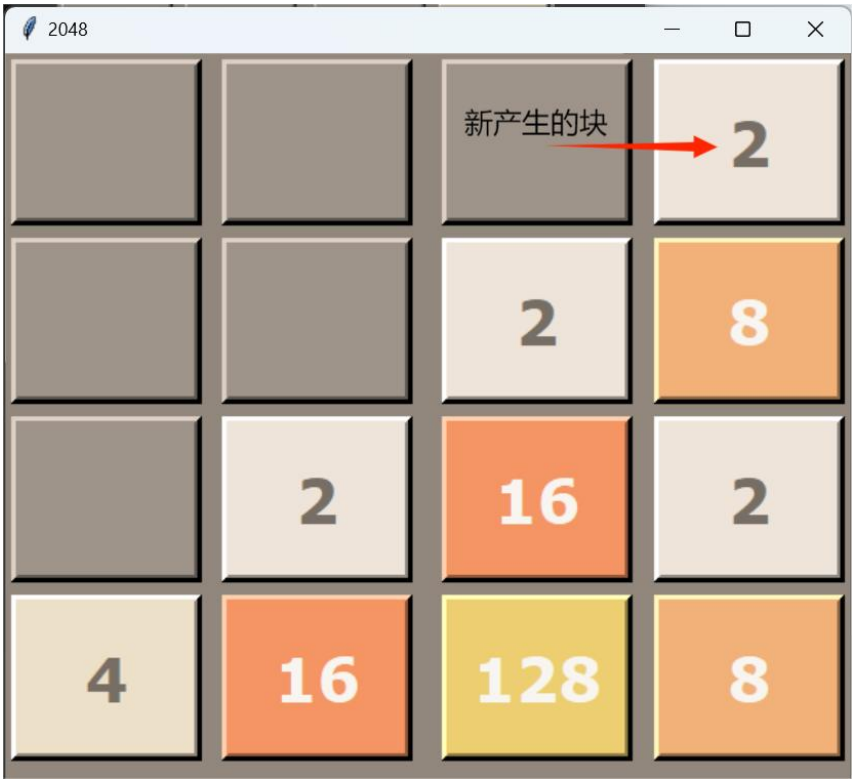

图 6 向下移动后

7 向左移动的算法

在玩家输入 a 字符或是 AI 使用蒙特卡洛算法结合启发式算法评估向上移动方案的得分时进入此函数，使用先旋转(board = np.rot90(board, 2))后移动（push_board_right(board)）再合并(merge_elements(board))然后再移动（push_board_right(board)）以消除空白格子的方法移动并合并相同格子，最后旋转回来(board = np.rot90(board, -2))并记录被合并的格子（vis 数组），把合并后的 board，能否成功移动 move_made，移动后的得分 score，被合并的格子 vis 传回到原本的代码中来进行覆盖（玩家移动）或是进行判断比较（AI 判断或是移动）。

代码如下：

```
def move_left(board):    # 向左移动的方法（包含合并）
    board = np.rot90(board, 2)    # 顺时针旋转 180°
    board, has_pushed = push_board_right(board)
    #    push_board_right(board)函数能够将 board 中存放的非 0 数字向右移动，等待被合并
    board, has_merged, score, vis = merge_elements(board)
    #     merge_elements(board)函数能够将 board 中存放的相邻的相同数字进行合并，并且返回合并后的数组 board，是否产生合并 has_merged，得分 score 以及被合
```

并的格子数组 vis。

```
board, _ = push_board_right(board)  # 将合并后的格子转移到最右
vis, _ = push_board_right(vis)  # 将表达每个格子是否发生合并的 vis 数组也转移
                                到最右，即 vis 跟着 board 一起动
board = np.rot90(board, -2)  # 顺时针旋转 180°
vis = np.rot90(vis, -2) # vis 一起旋转
move_made = has_pushed or has_merged  # 判断是否移动或者合并
 return board, move_made, score, vis
    # 返回向左移动后的 board，能否移动的 move_made，移动后的得分 score，
    判断合并数组 vis
```

函数功能样例如下：

2	2	2	2	操作后 ——>	4	4		
8		8			16			
16			16		32			
2	2		4		4	4		

函数功能实例如下图：

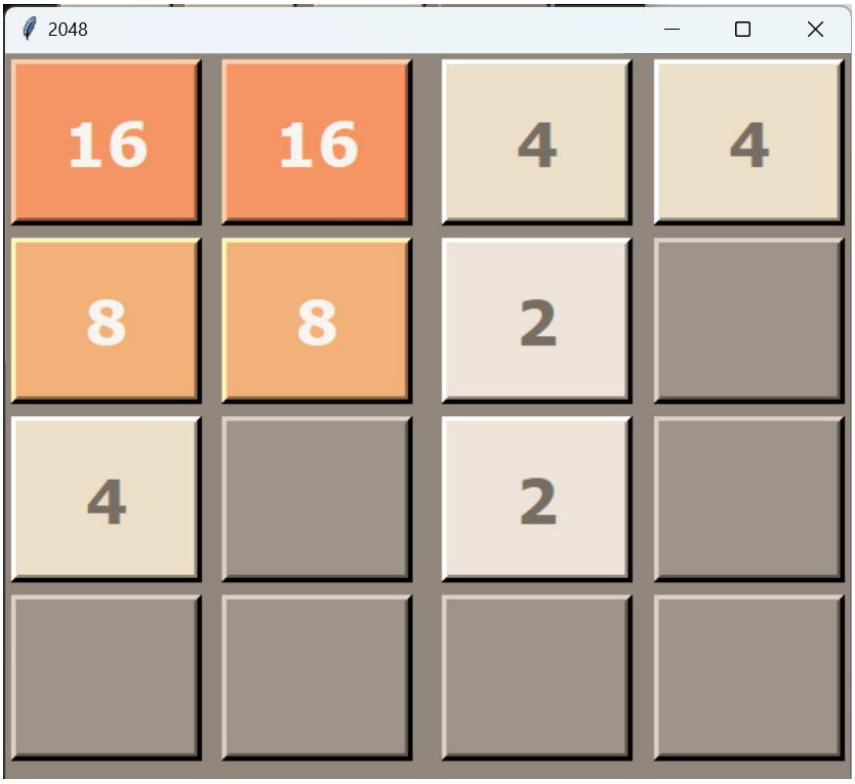

图 7 向左移动前

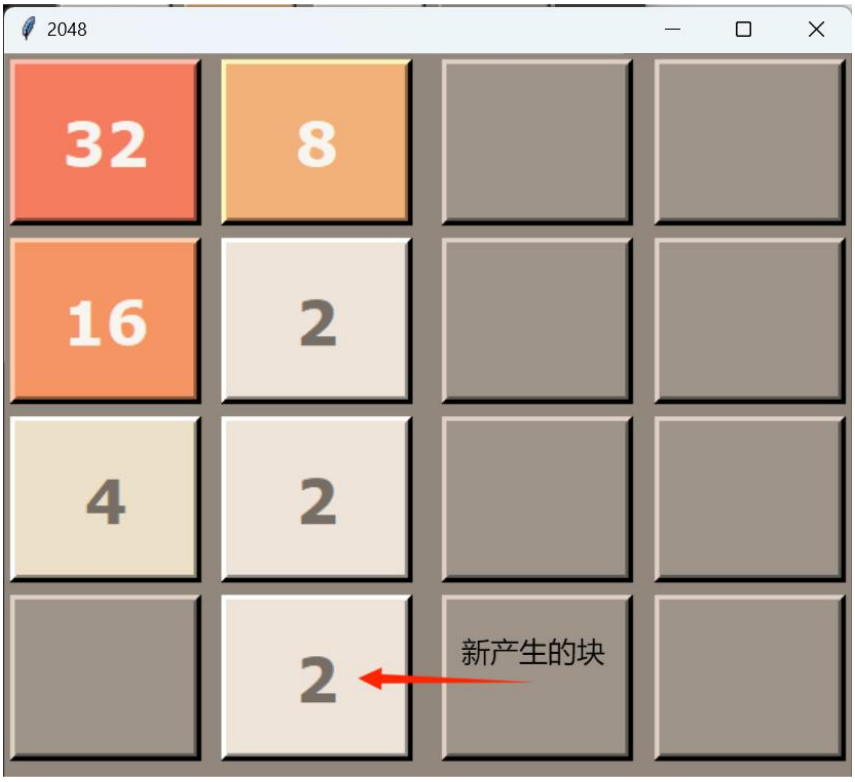

图 8 向左移动后

8 向右移动的算法

在玩家输入 d 字符或是 AI 使用蒙特卡洛算法结合启发式算法评估向上移动方案的得分时进入此函数，使用先移动（push_board_right(board)）再合并(merge_elements(board))然后再移动（push_board_right(board)）以消除空白格子的方法移动并合并相同格子并记录被合并的格子（vis 数组），把合并后的 board，能否成功移动 move_made，移动后的得分 score，被合并的格子 vis 传回到原本的代码中来进行覆盖（玩家移动）或是进行判断比较（AI 判断或是移动）。

代码如下：

```
def move_right(board):    # 向右移动的方法（包含合并）
    board, has_pushed = push_board_right(board)
    #   push_board_right(board)函数能够将 board 中存放的非 0 数字向右移动，等待被合并
    board, has_merged, score, vis = merge_elements(board)
    #    merge_elements(board)函数能够将 board 中存放的相邻的相同数字进行合并，并且返回合并后的数组 board，是否产生合并 has_merged，得分 score 以及被合并的格子数组 vis。
```

board, _ = push_board_right(board)　# 将合并后的格子转移到最右

vis, _ = push_board_right(vis)　# 将表达每个格子是否发生合并的 vis 数组也转移到最右，即 vis 跟着 board 一起动

move_made = has_pushed or has_merged　# 判断是否移动或者合并

return board, move_made, score, vis

返回向右移动后的 board，能否移动的 move_made，移动后的得分 score，判断合并数组 vis

函数功能样例如下：

2	2	2	2	操作后 ——>			4	4
8		8						16
16			16					32
2	2		4				4	4

函数功能实例如下图：

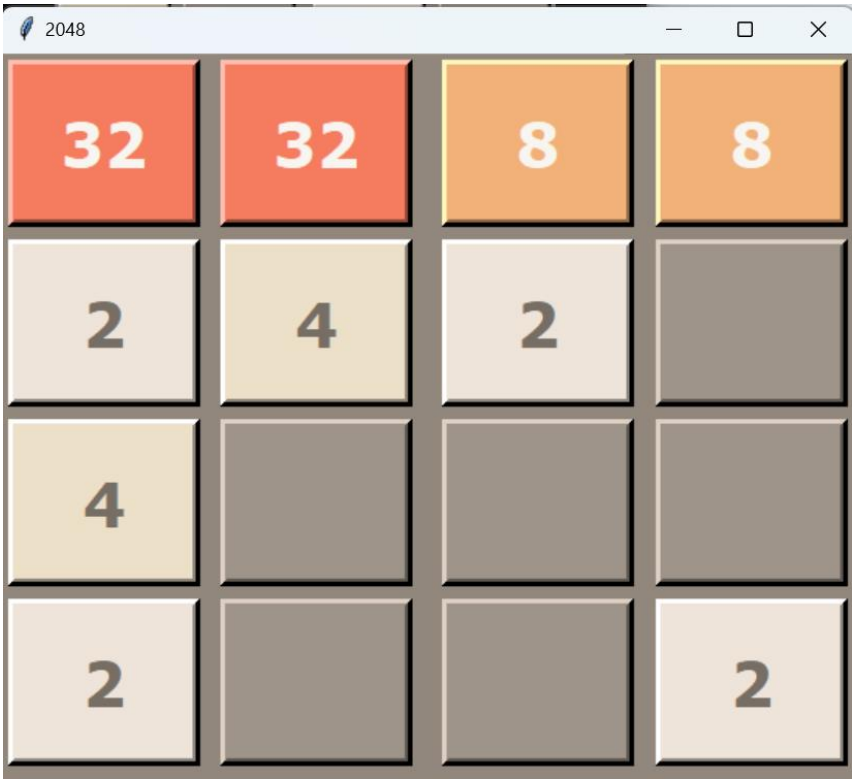

图 9　向右移动前

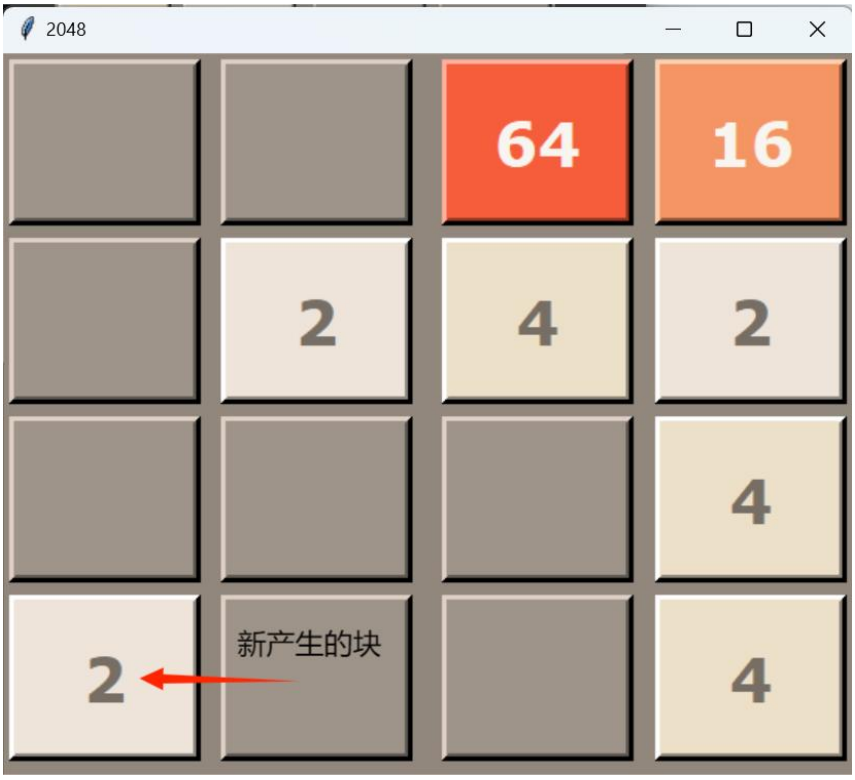

图 10 向右移动后

8 AI 随机执行向上或向下向左向右并记录结果的算法

轮到 AI 移动时，AI 对于固定深度和固定广度进行随机搜索，每次搜索时调用该函数，并返回执行操作后的新状态 board，是否能够执行(在 game_ai.py 中为 game_valid 变量，此处为返回的第二个变量 True 或者 False)，得分 score，是否被合并 vis 数组。

代码如下：

```
def random_move(board):
    move_made = False #  能否移动
    move_order = [move_right, move_up, move_down, move_left] #  四个方向的函数名称数组
    while not move_made and len(move_order) > 0:   #  最多循环四次，若有成功的时候则跳出循环
        move_index = np.random.randint(0, len(move_order))
        move = move_order[move_index]     #    随机一个方向
        board, move_made, score, vis = move(board)    #  向右上下左随机方向移动
        if move_made:
            return board, True, score, vis    #能够移动则返回移动后的各项值
```

```
            move_order.pop(move_index)   # 删除该方向
        return board, False, score, vis
        #四个方向都无法执行，则返回失败
```

四、测试数据及其结果分析

仅能在使用按键 s 时完成向下移动的操作：

图 11 向下移动前

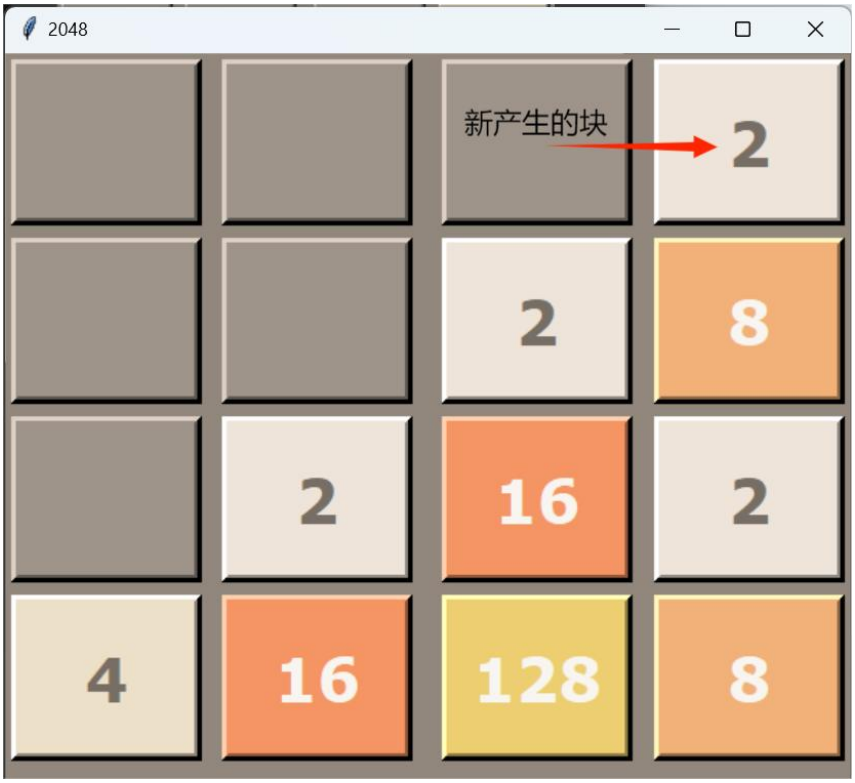

图 12 向下移动后

仅能在使用按键 w 时完成向下移动的操作：

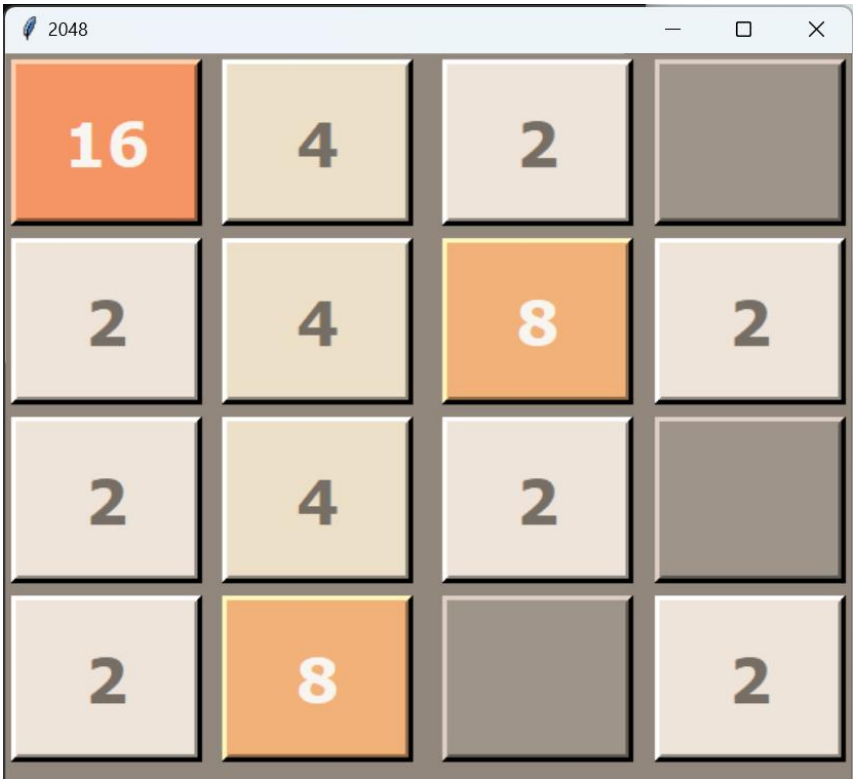

图 13 向上移动前

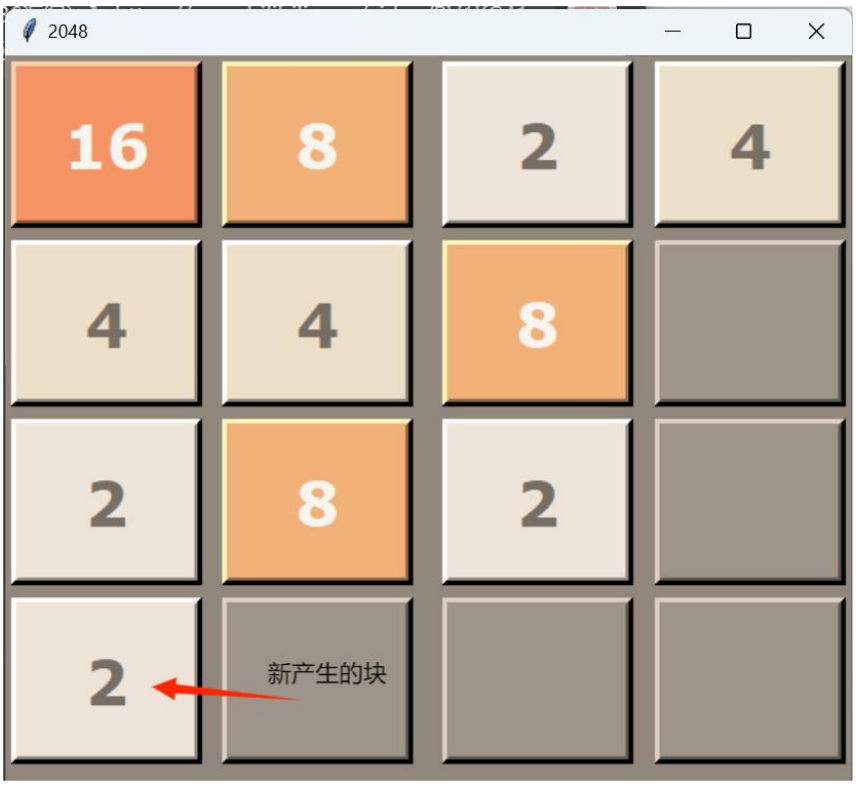

图 14 向上移动后

仅能在使用按键 a 时完成向下移动的操作：

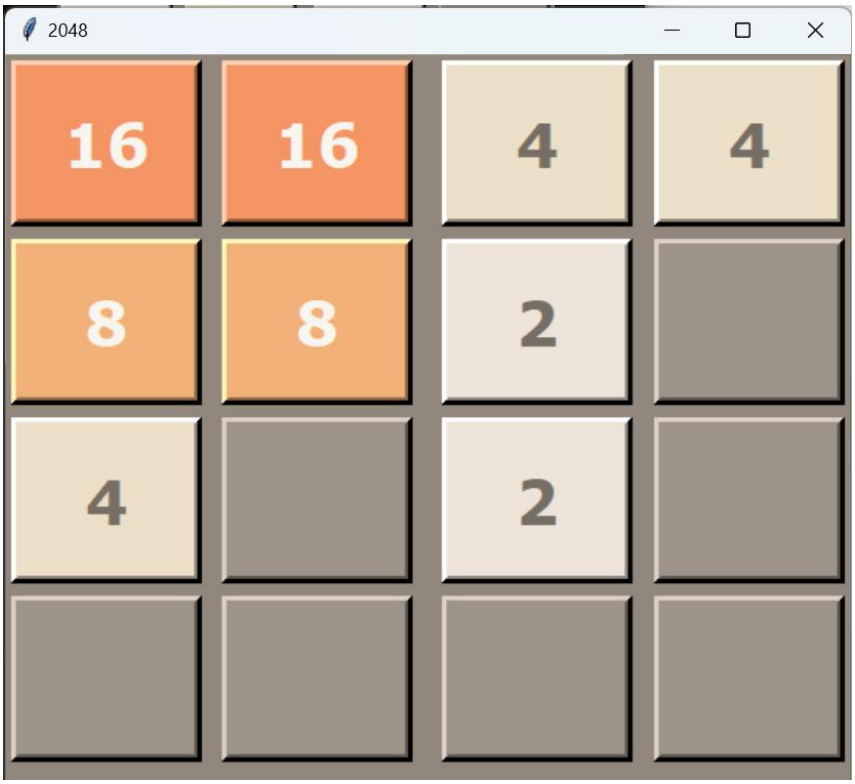

图 15 向左移动前

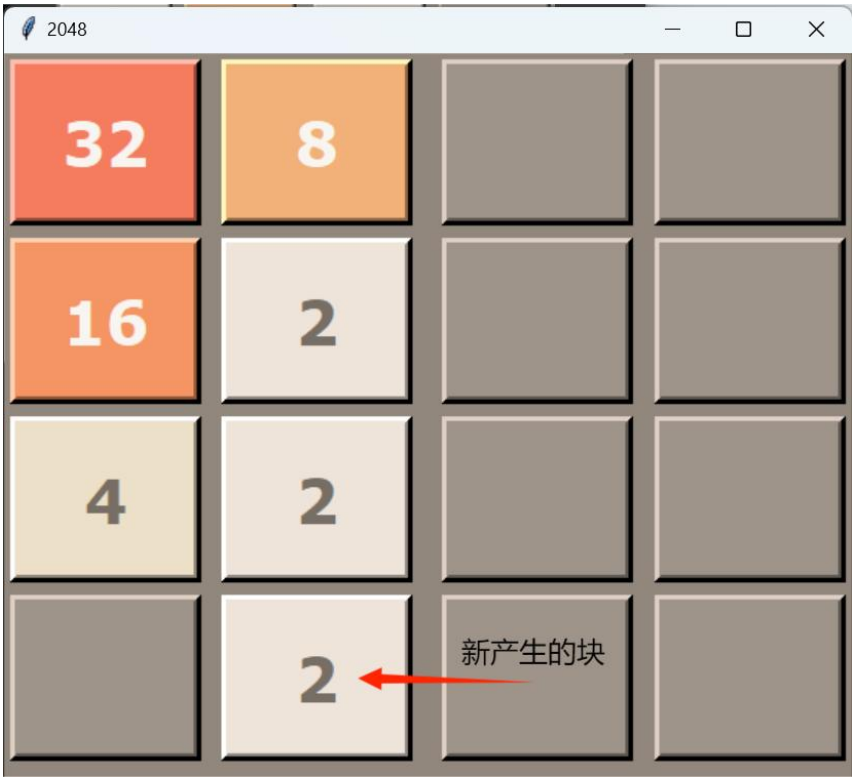

图 16 向左移动后

仅能在使用按键 d 时完成向下移动的操作：

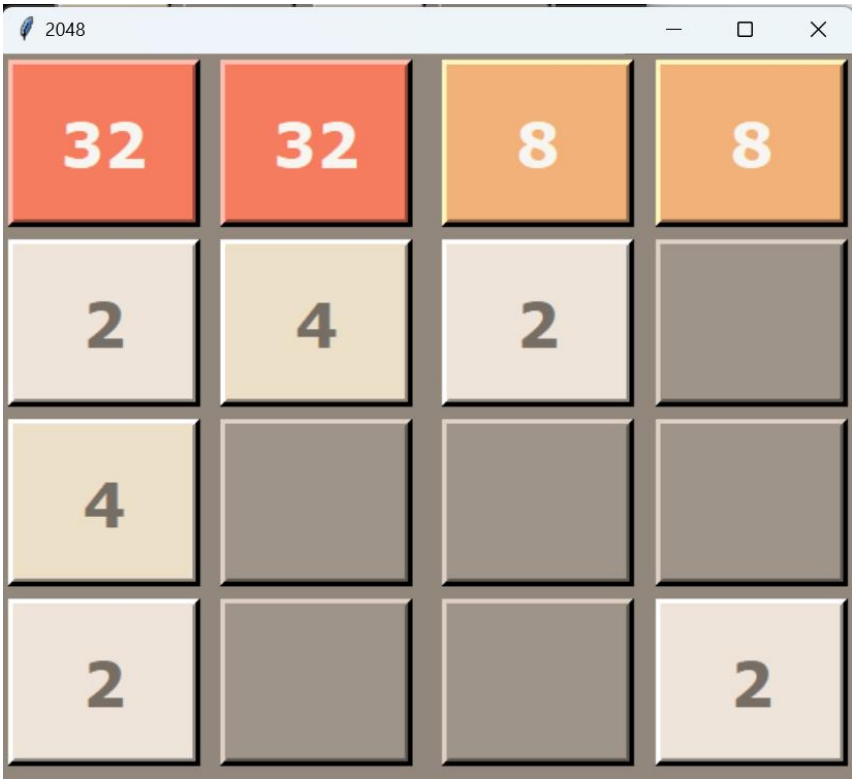

图 17 向右移动前

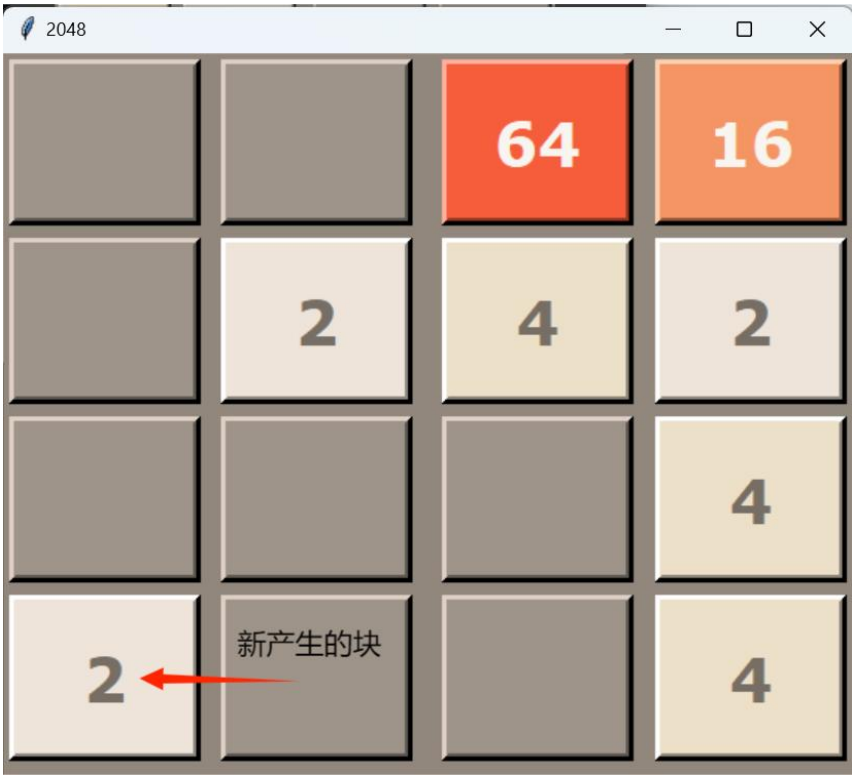

图 18 向右移动后

五、算法设计和程序调试过程中的问题

问题 1：

发现无法在 function 文件中使用函数来让合并出现特效。

解决方法：

引入 vis 数组，记录下合并的数字的确切位置，传出 vis 到 display 文件中，在执行绘画界面时使用 vis 来绘制合并特效。

问题 2：

在调用向上向下向左向右移动的函数时，发现系统报错无法正常移动。

解决方法：

发现是由于传出的函数变量在接收时出错，使用“_”来过滤掉剩余的传出参数即可正常运行。

问题 3：

在完成合并后，vis 数组数据出现问题导致游戏中显示合并的特效的格子出错。

解决方法：

在仔细研究后，发现多个问题：

1 由于 board 是在旋转后进行的 push 操作，后面还旋转回去了，而 vis 数组没有进行

旋转操作。

解法：所以 vis 数组也要一并旋转回去以保持 board 对应的 vis 数组不出错。

2 由于 vis 数组一开始使用的时 bool 型变量，只有两个数值，在使用 push_board_right(board)函数来和board数组保持对应时若使用0代表board的0，1代表board 产生合并，那么 board 中有数字却未发生合并的数字无法表示。

解法：将 vis 类型改为 int，0 代表没有数值，1 代表有数值没合并，2 代表合并后的块。

六、展望

1 使用 pygame 绘制更加优秀的移动动画，合并动画，产生新块的动画。

2 借鉴网上的启发式算法优化自己的启发式算法。

3 编写一个脚本自动运行来寻找最高分。

七、课程设计总结

优化搜索的剪枝策略：

1 使用动态广度与深度：不再每个方向上给与相同的广度，而是在能够容易得分的方向上给与更多广度和深度；不容易得分或者是容易死路的方向上给与更少广度与深度。

2 使用记忆化搜索，记录一些运行过程中容易决策出错的局面，当局面相同时直接给出结果。

启发式的优化：

1 借鉴网上更加优秀的作品的启发式思想，增加启发式的条件，优化不同条件的权重，不断测试找到最好的权重比。

2 记忆化启发式，找到一些启发式较难比较的局面，当局面相同时直接给出结果。

总结：

这次实践课程很好的锻炼了我们的实际代码能力，我也学会了一门新的语言，学会了新的算法。最终我们的代码近乎达到了 100%的 2048 概率。