

# Klotski

## 一、成员

Name	SID
李乐平	12112627
张天舒	12111046
黄景娟	12112847

## 二、使用

运行主程序: 先编辑配置传入实参 `terminal` 或者 `gui`, 再从 `src` 下找到 `Main` 运行

运行随机生成可解棋盘: 从 `src` 下找到 `GenerateRandom` 运行

## 三、算法

### 1. 思路

初始化棋盘, 找到空格搜索上下左右, 将可以走的棋盘存入优先队列, 每次从优先队列中选出离目标值最近的棋盘继续搜索, 直到找到目标结束, 若优先队列为空则找不到答案。

距离目标值采用 Manhattan Distance:  $distance = |i - x_{goal}| + |j - y_{goal}|$

### 2. 结构

`Main` 主窗口进入程序

`Core` 找解的核心代码

`Board` 棋盘类

`Step` 走法类

`KlostkiFrame` GUI框架

`Grid` GUI网格类

`GenerateRandom` 随机生成可解棋盘

`DFS` 深搜

`BFS` 广搜

### 3. Search伪代码

```
1  开始:  
2    输入  
3    处理绑定类型，将值与绑定类型建立映射  
4    建立优先队列，优先队列的评估函数为当前盘面所有值距离其归位的曼哈顿距离（水平距离的绝对值加竖直距  
离的绝对值）之和，并作适当优化  
5    将初始盘面压入优先队列  
6    将初始盘面压入散列表  
7    开始搜索:  
8      如果超时:  
9        结束搜索  
10     返回空  
11     输出No  
12     结束  
13     如果优先队列中元素数量过多:  
14       剪枝  
15     如果优先队列不为空:  
16       取优先队列顶端的盘面  
17       如果已经复原:  
18         返回移动次序  
19         输出Yes  
20         输出移动次序  
21         结束  
22       遍历盘面，找到空点  
23       向空点四周移动，根据四周格子中值对应的绑定类型判断能否移动  
24       如果能移动:  
25         建立移动后新的盘面  
26         添加移动次序  
27         如果散列表中不存在当前盘面且步数未超过设定的上限:  
28           将新盘面压入散列表  
29           将新盘面压入优先队列  
30     否则:  
31       结束搜索  
32       返回空  
33       输出No  
34       结束
```

### 4. 测试

#### (1) 测试1

Sample Input 1

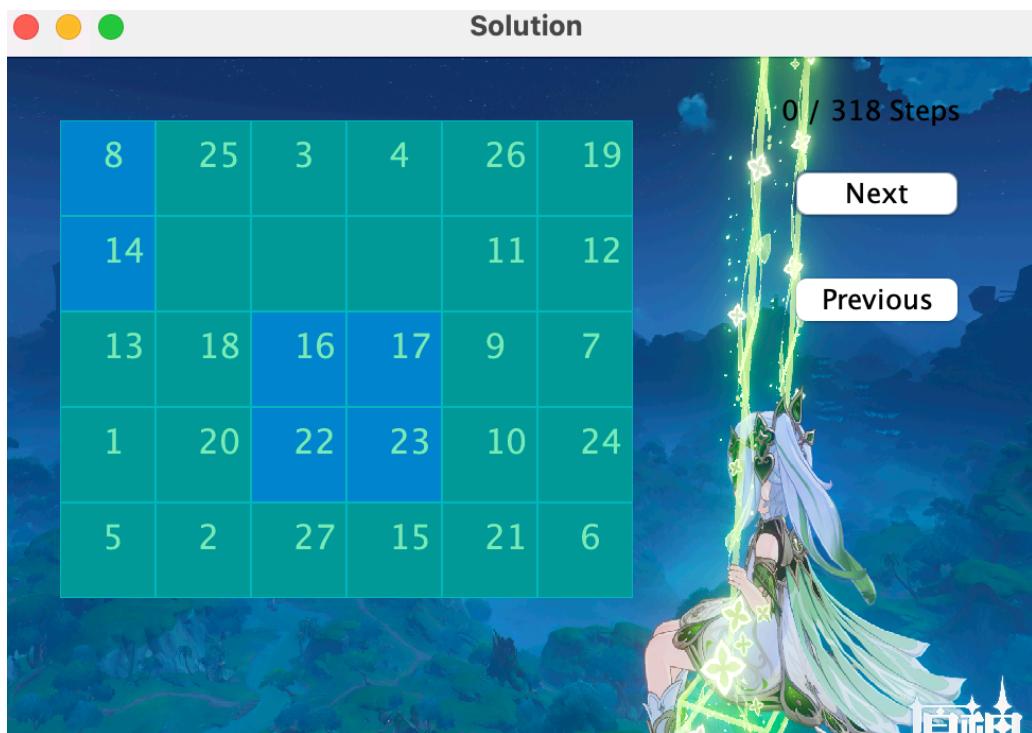
```
1 5 6
2 8 25 3 4 26 19
3 14 0 0 0 11 12
4 13 18 16 17 9 7
5 1 20 22 23 10 24
6 5 2 27 15 21 6
7 2
8 8 2*1
9 16 2*2
```

### Sample Output 1

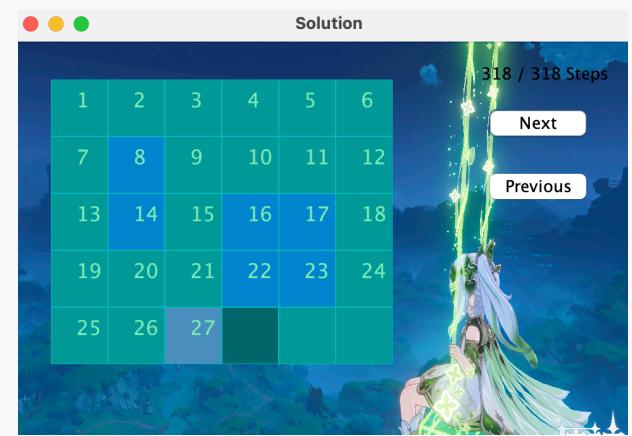
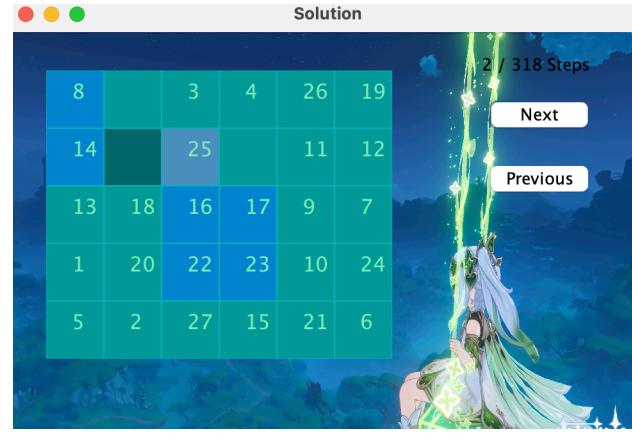
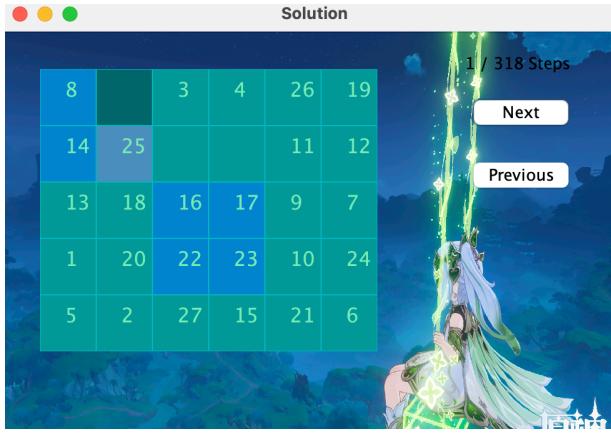
```
1 Yes
2 318
3 25 D
4 25 R
5 8 R
6 ...
7 27 L
8 26 L
9 27 L
```

### GUI 1

初始化



点击Next走棋



## (2) 测试2

### Sample Input 2

```
1 | 6 5
2 | 1 2 3 4 5 6
3 | 7 8 9 10 11 12
4 | 13 14 15 16 17 18
5 | 19 20 21 22 23 0
6 | 24 25 26 27 28 0
7 |
8 | 22 2*2
```

### Sample Outout 2

```
1 | No
```

### (3) 测试3

#### Sample Input 3

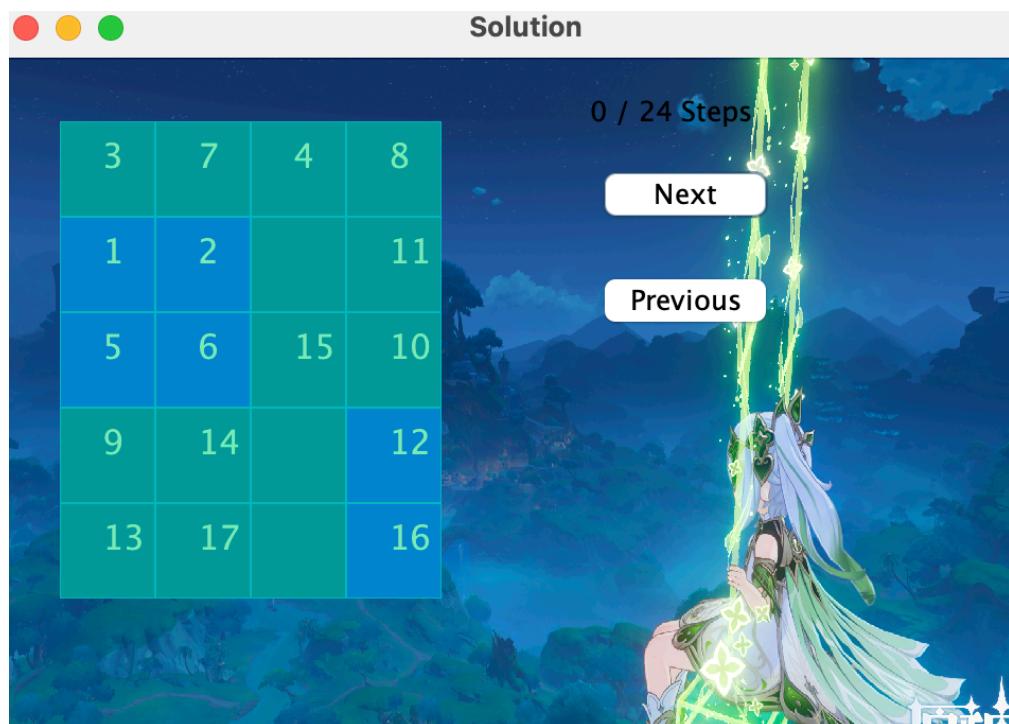
```
1 5 4
2 3 7 4 8
3 1 2 0 11
4 5 6 15 10
5 9 14 0 12
6 13 17 0 16
7 2
8 12 2*1
9 1 2*2
```

#### Sample Output 3

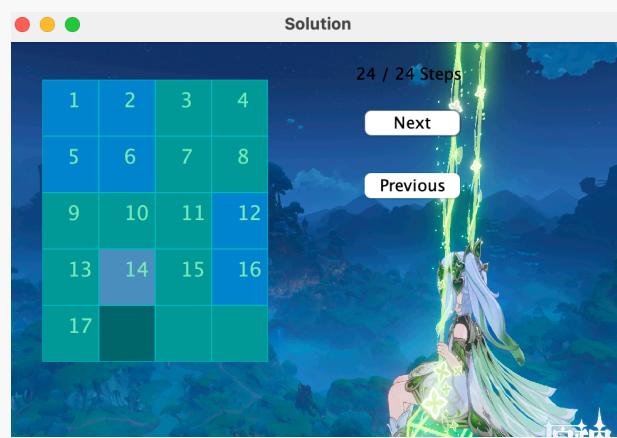
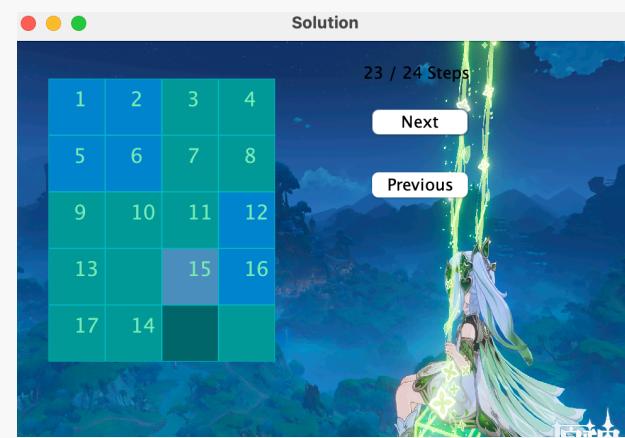
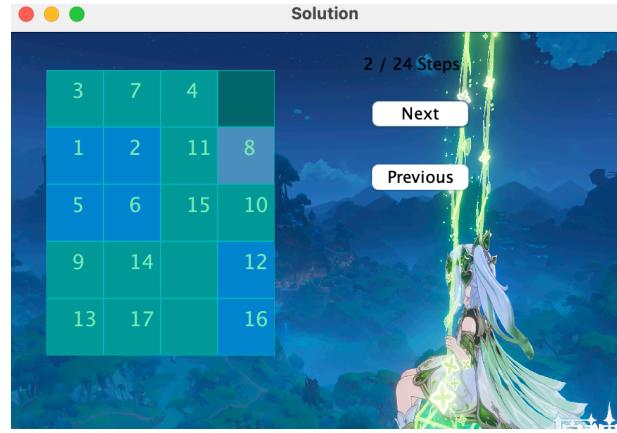
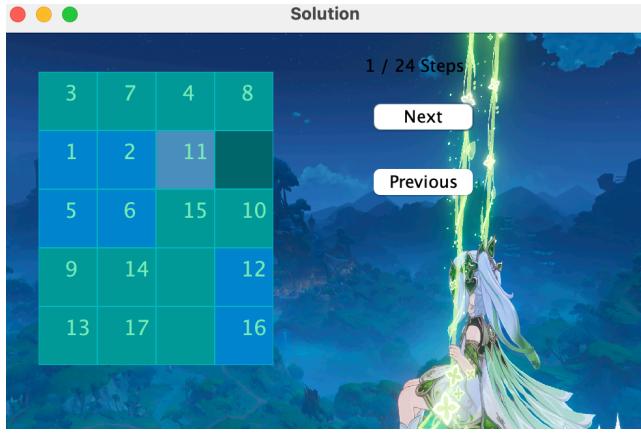
```
1 Yes
2 24
3 11 L
4 8 D
5 ...
6 15 U
7 14 U
```

### GUI 3

#### 初始化



点击**Next**走棋



#### (4) 测试4

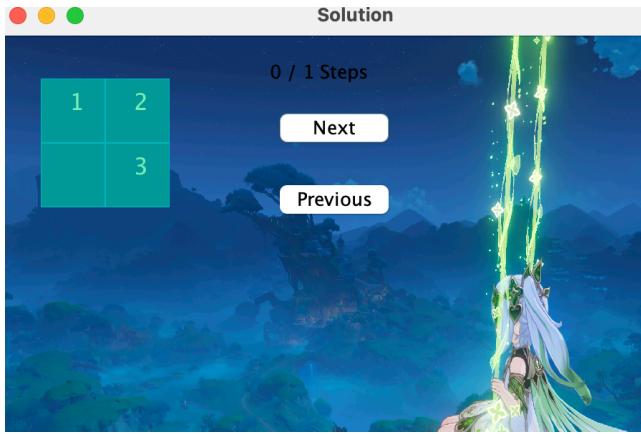
##### Sample Input 4

1	2	2
2	1	2
3	0	3
4	0	

##### Sample Output 4

1	Yes
2	1
3	3 L

##### GUI 4



## (5) 测试5

### Sample Input 5

1	2 2
2	2 1
3	0 3
4	0

### Sample Output 5

1	No
---	----

## (6) 测试6

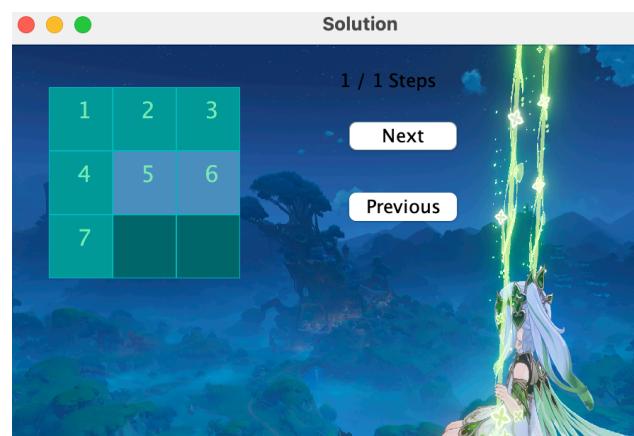
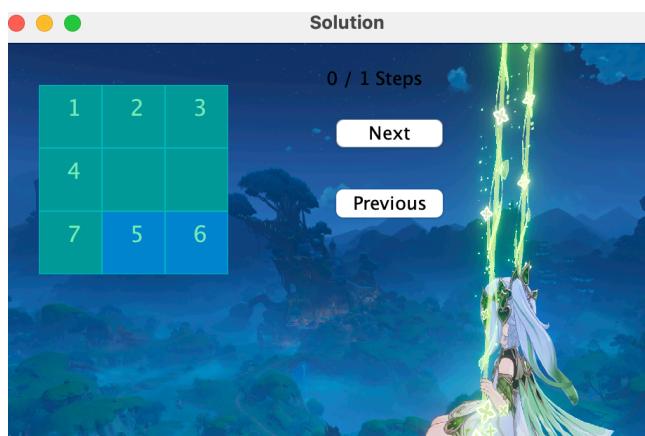
### Sample Input 6

1	3 3
2	1 2 3
3	4 0 0
4	7 5 6
5	1
6	5 1*2

### Sample Output 6

1	Yes
2	1
3	5 U

## GUI 6



## (7) 测试7

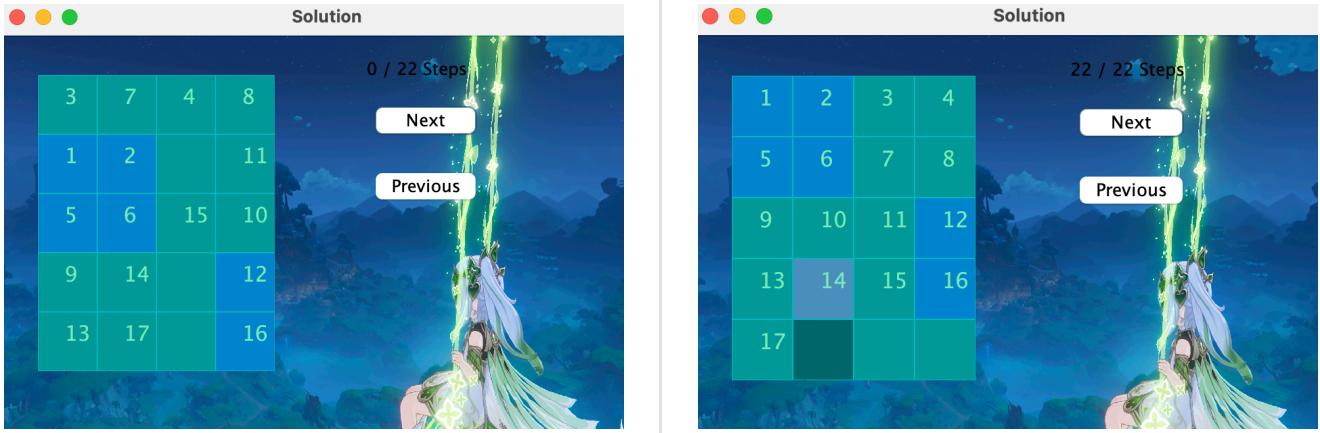
### Sample Input 7

```
1 5 4
2 3 7 4 8
3 1 2 0 11
4 5 6 15 10
5 9 14 0 12
6 13 17 0 16
7 2
8 12 2*1
9 1 2*2
```

### Sample Output 7

```
1 Yes
2 22
3 11 L
4 15 D
5 10 L
6 ...
7 15 U
8 10 U
9 14 U
```

## GUI 7



## 四、Bonus

### 1. 随机可解棋盘

#### (1) 生成

随机生成  $3 \times 3 - 6 \times 6$  大小(可以是  $3 \times 4$  等长方形)、有 0-3 个绑定( $1 \times 2$ 、 $2 \times 1$ 、 $2 \times 2$ ) 的棋盘,

命令行输出可解棋盘(格式按照 input), GUI 展示复原过程。

#### (2) 运行

从 src 下找到 GenerateRandom 运行

#### (3) 原理

先按照大小生成最终棋盘, 找到所有可走步数并随机选择其中一步进行移动, 如此循环 `random.nextInt(800) + 20` 次, 按照此种办法逆着走回去即可得到最终棋盘, 因此棋盘一定可解。

#### (4) 伪代码

```

1 | 生成随机盘面:
2 | 开始:
3 |   生成随机长宽 (h, l) ∈ [3, 6] × [3, 6]
4 |   生成随机数字个数 (n ∈ [h - 3, h - 1])
5 |   生成有序盘面
6 |   生成随机步数 (s ∈ [20, 819])
7 |   进行随机合法绑定 (绑定次数 ≤ 3)
8 |   遍历棋盘 s 次, 每次找到所有可走步数并随机选择其中一步进行移动, 并压入散列表防止走回头路
9 |   将步数数组颠倒, 返回最终盘面并格式化输出

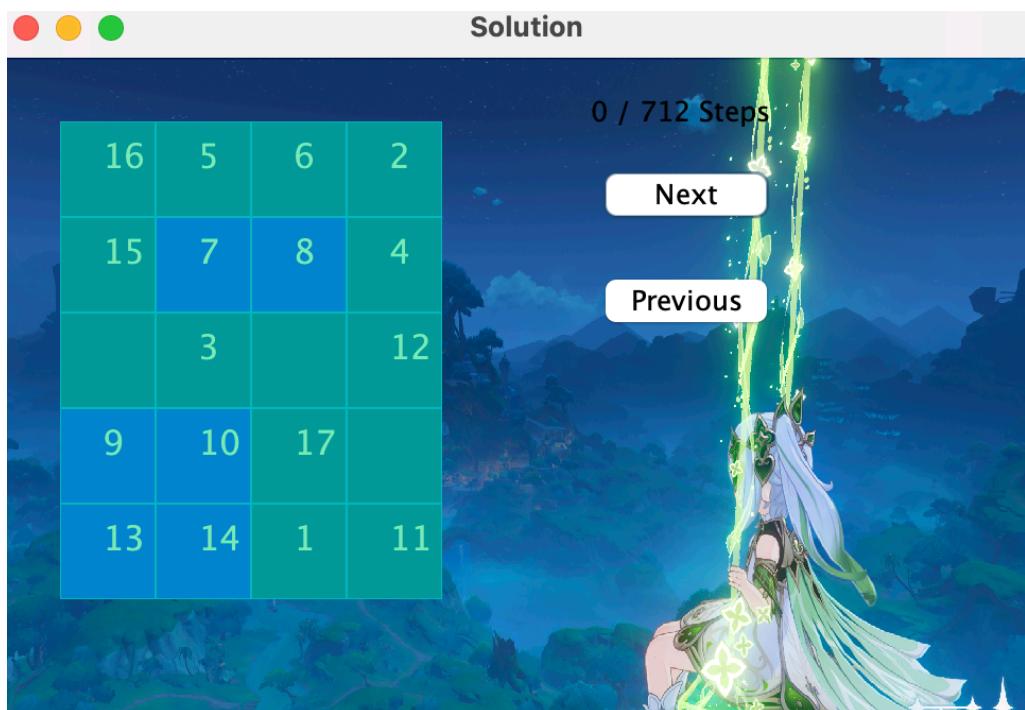
```

#### (5) 测试

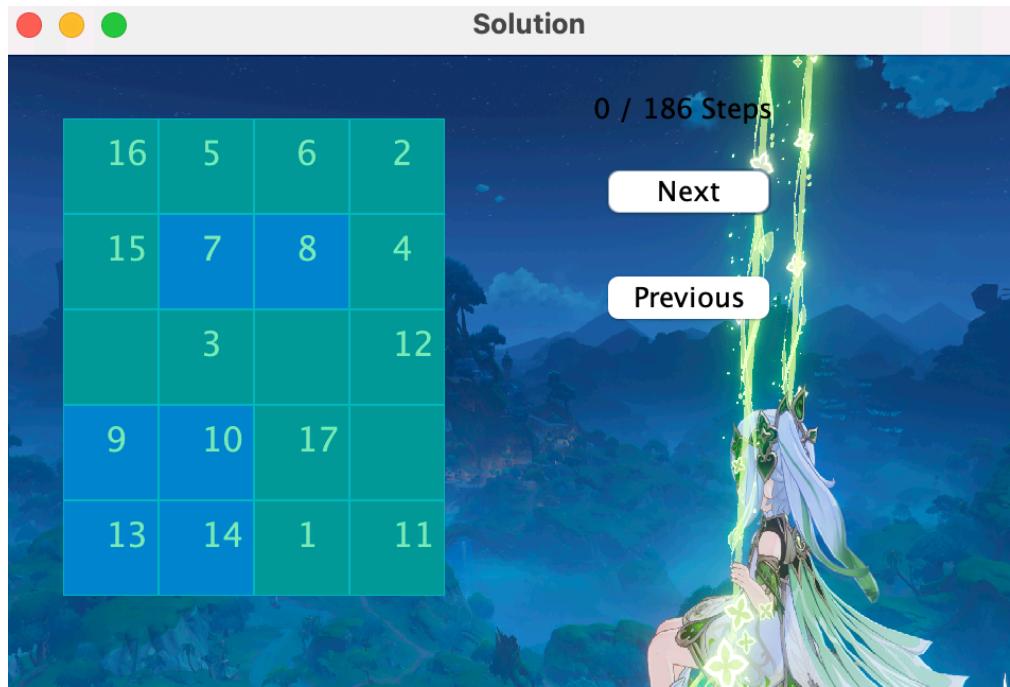
## Output

```
1 5 4
2 16 5 6 2
3 15 7 8 4
4 0 3 0 12
5 9 10 17 0
6 13 14 1 11
7 2
8 7 1*2
9 9 2*2
```

## GUI



手动将 Output 输入到 Main 解出来



## 2. 对比算法

### (1) 说明

1. 我们的算法采用简化版 A\* 算法，采用优先队列的数据结构，以曼哈顿距离为选择下一步的棋局的优先级，结合搜索算法，单向从棋局开始搜索到结束。
2. 将数据结构进行更改，分别改成栈和队列，即可得到朴素深搜和朴素广搜，可以实现小棋局的求解。
3. 将下一步棋局优先级更改，采用欧式距离与曼哈顿距离之和，即可得到 A\* 算法。
4. 同时，提出采用双向搜索，从起点和终点两端同时搜索，两棋局局面相遇即可得到走法，此方法可以减小搜索时间，但是占用更多空间。

### (2) 表格对比

算法	核心算法	朴素深搜	朴素广搜	A*算法	双向广搜
数据结构	优先队列	栈	队列	优先队列	队列
下一步优先级	曼哈顿距离	任意	任意	欧式+曼哈顿距离	任意
特点	快	慢	中	快	快
可以解决	6*6	3*3	3*3	*	*

### (3) 前三个算法对比 3\*3 效果

由图可得，朴素深搜搜索次数多，更具有盲目性。

Input

1	3	3
2	4	0
3	1	8
4	6	5
5	0	

## GUI

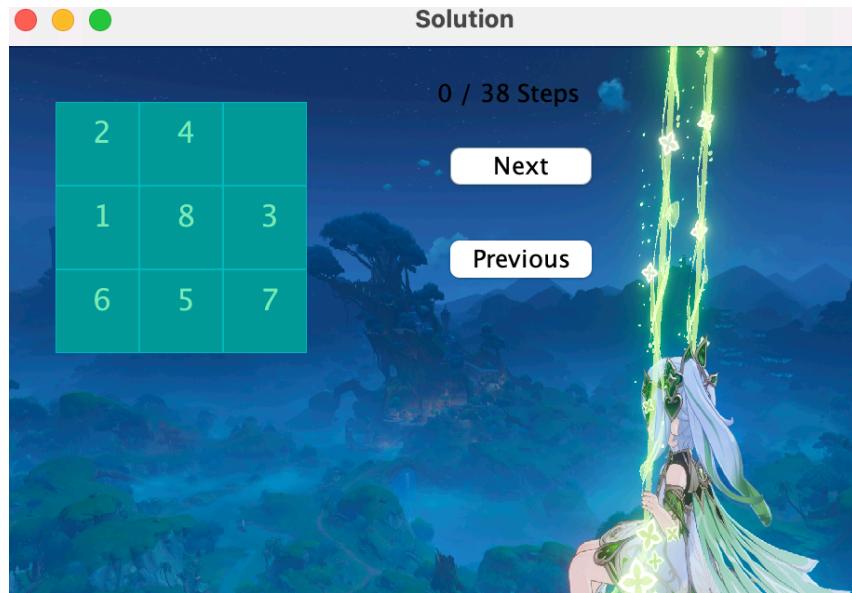


Figure 1 核心算法

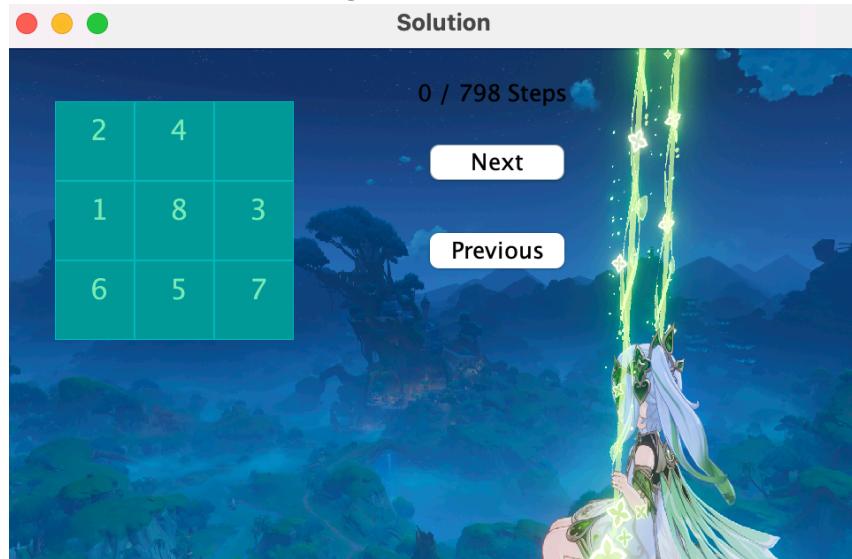


Figure 2 朴素深搜

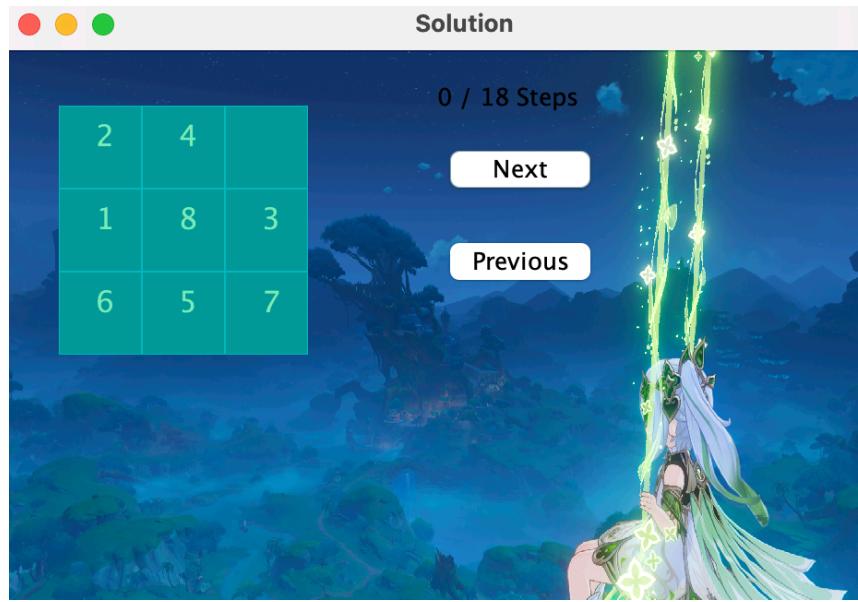


Figure 3 朴素广搜