

基于游戏《轨道连结》的状态机研究

2024年8月12日

作者: NnWinter冬
联系方式(主要): t.me/NnWinter
联系方式(次要): github.com/NnWinter
特别鸣谢: Zhao (♥)
轨道连结游戏制作组 (不错的游戏)
b站 10707223 (视频作者)

本文是基于对B站视频【程序员究极疑问——这个程序为什么能跑？】
<https://www.bilibili.com/video/BV1GC4y1J7VH> 中游戏《轨道连结》特定解法的研究，主要是对游戏中的轨道系统进行分析。

本文内容主要是关于如何将游戏中的轨道系统转化为状态机，以及其编程实现。

本文为娱乐性质，不具有实际应用价值，内容也并不严谨，请多加注意。

创作声明：
本文所有内容皆为原创，无引用的外部参考。开发过程已上传到云端备份，如有版权争端请联系作者。
本文开源协议为 GPL v3，转载时请注明出处。

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

== 目录 ==

Copyright (C) 2024 NnWinter	1. 节点分析	--- 2
This program is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.	2. 整合子系统	--- 3
	3. 变轨处理	--- 4
This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.	4. 状态机	--- 6
	5. 状态机代码	--- 7
You should have received a copy of the GNU General Public License along with this program. If not, see .	6. 状态机运行映射效果图	--- 10

1.节点分析

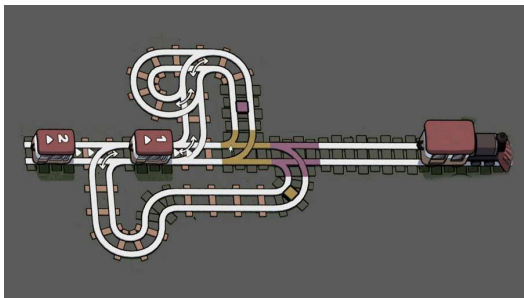
岔路和切换的地方一共有8个节点，分别以数字标出



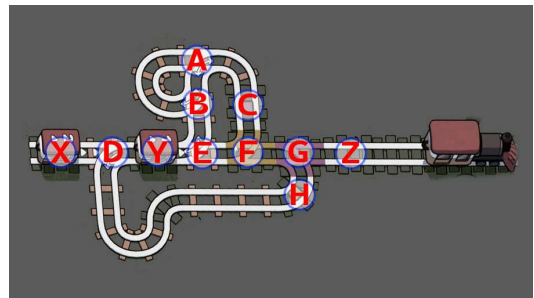
[图1] 视频截图



[图2] 标注图



[图3] 简化图



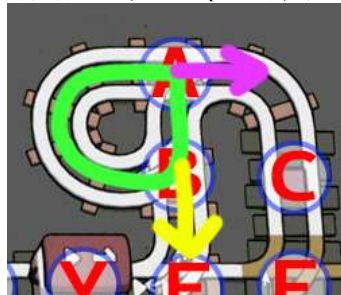
[图4] 节点图

将节点的数字与字母对应以便于规划路径图

- (1) = A • (5) = E • 车2 = X
- (2) = B • (6) = F • 车1 = Y
- (3) = C • (7) = G • 出口 = Z
- (4) = D • (8) = H

分析节点之间的关系

以 A 和 B (1 和 2) 之间的关系为例，可以看到这里是一个环形结构(绿色)并具有两个出口(紫色/黄色)



[图5] 子系统

在这个子系统中，输入和输出是确定的：

1. 如果从紫色部分进来，会绕一圈从紫色部分出去
2. 如果从黄色部分进来，会直接向上右转从紫色部分出去

并且可以算出，从紫色方向进入A会用4步回到A：A -> B -> 空 -> 空 -> A

同理，从黄色方向进入B会用1步到达A：B -> A

所以根据这个原理，我们可以推断出：任何白色轨道构成的子系统的输入输出必然是固定的。

2.整合子系统

根据上文可以有推论：

每个子系统的输入输出是确定的；将多个这样的子系统进行连接时，输入输出也是确定的。

还以[图5]为例，如果从B进入这个系统，一定会在1步后到达A；

$B \rightarrow A$

而从A出来后，还会继续走到C；

$A \rightarrow \text{空} \rightarrow C$

反之，如果从C出发，会先到达A，途径B，回到C。

$C \rightarrow \text{空} \rightarrow A \rightarrow B \rightarrow \text{空} \rightarrow \text{空} \rightarrow A \rightarrow \text{空} \rightarrow C$

因此可以通过这种方式 将所有的白色轨道合并为一个子系统。

在不考虑车厢的前提下，可以看出实际有用的节点有且仅有 4 个，分别是 X(入口)、F(可变轨道-橙)、G(可变轨道-紫) 和 Z(出口)。

首选挑选其中一个节点作为起点，看看都分别有哪些输入和输出，这里可以选挑选 X 作为起点。

可以看到，X 只有一个输入就是当它本身作为起点时(没有其它情况可以到达)，而输出也只有一个确定的路线就是会到达 F。

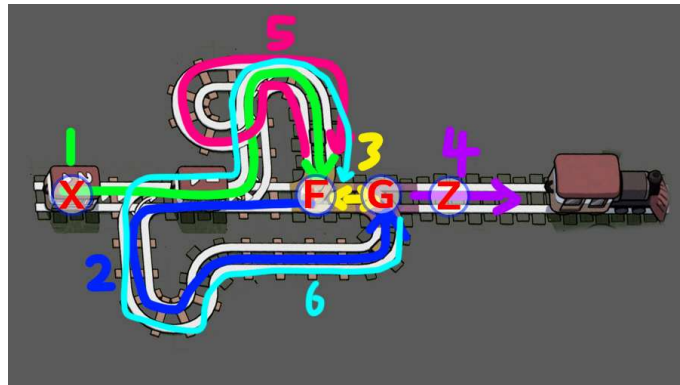
具体路线经过： $X \rightarrow D \rightarrow Y \rightarrow E \rightarrow B \rightarrow A \rightarrow \text{空} \rightarrow C \rightarrow F$

中间的过程不重要

最后只需要知道，从 X 出发，经过 8 步后一定会到达 F。

因此可以写作： $X \text{--} 8 \text{--} F$

根据上述原理，可以统计出所有节点之间的关系：



[图6] 节点关系图

1. $X \text{--} 8 \text{--} F$ (绿色)
2. $F \text{--} 1 \text{--} G$ (黄色)
3. $F \text{--} 10 \text{--} F$ (洋红)
4. $F \text{--} 11 \text{--} G$ (蓝色)
5. $G \text{--} 1 \text{--} F$ (黄色)
6. $G \text{--} 15 \text{--} F$ (青色)
7. $G \text{--} 1 \text{--} Z$ (紫色)

注：数字编号与图中并不一致

3.变轨处理

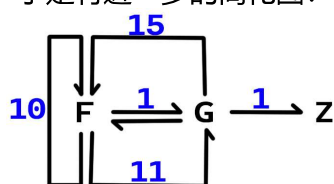
已知 只有 2 个轨道的状态会发生改变，且每个轨道只有 2 种状态。(F, G)
因此 全局的轨道布局只有 4 种状态。(FG, FG', F'G, F'G')

要重点关注的部分显然还是 F 和 G 的变化。
而通过观察可以看出，在最开始时，两个车厢其实一前一后经过了 F (绿色路线[1])

因此 可以省略掉从 X 必然会到 F 的部分，
相当于两个车厢分别在6步后从 F 出发的。
所以 X 节点是冗余的，可以排除掉。

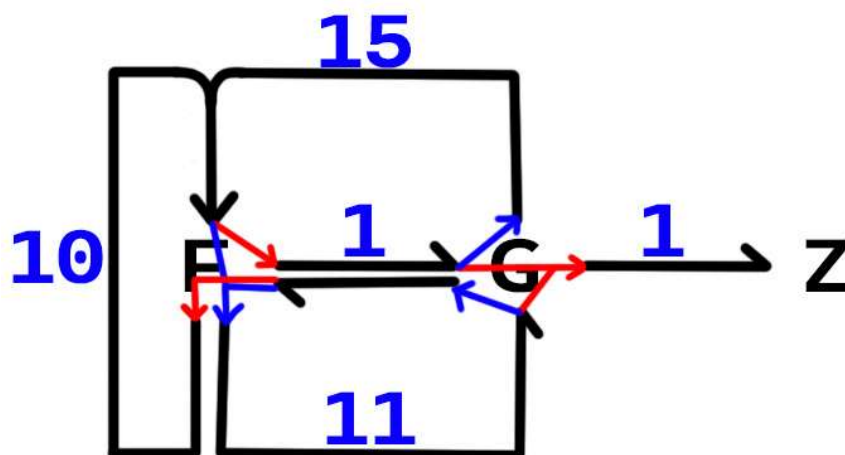
此时只剩下了 F, G 和 Z 这三个节点。
而根据之前的节点关系，F 只能到达 F 或 G，而无法达到 Z。
反之，Z 只能从 G 来。
因此 G 是连接 F 和 Z 的中间节点。(显而易见)

于是有进一步的简化图：



[图7] FGZ节点关系图

以视频中[图1]的轨道初始状态为 \overline{F} 和 \overline{G} ，切换后分别为 \overline{F} 和 \overline{G} 。
将蓝色作为初始状态，红色作为变轨状态进行区分 (根据 [图3] 和 [图7] 推理而来)：



[图8] FGZ节点变轨关系图

[图8]可以被拆分为 4 种不同的路径，而每一种路径的结构是已知的，因此可以先将每种路径进行展开，变成一维的状态机路线。

介于 F 是起点，所以可以从这里开始。

在 \overline{F} 的状态下，由于 F 只有 1 个出口，所以不需要进行拆分，但 \overline{F} 时会有 2 个入口 2 个出口，因此还是要将不同入口分为 a 和 b。

定义为：

从上方进入 F 为 Fa ，从右侧进入 F 为 Fb 。

从左侧进入 G 为 Ga ，从下侧进入 G 为 Gb 。

-- 路线计算 --

路线[1] 假设从 F 上方进入蓝色，所以是 \overline{Fa} ，而 \overline{G} 则需要取决于抵达方向，则路线为：

$\overline{Fa} -- 11 -> Gb$ (这里 Gb 是因为从下边进入 G 的)

而此时已知进入的是 \overline{Gb} ，所以一定会走路线：

$\overline{Gb} -- 1 -> Fb$ (这里 Fb 是因为从 G 回来是从右侧进入的)

只要回到了 \overline{F} 就可以停了，原因在后边。

接下来就是拼接成一条完整的路线：

$\overline{Fa} -- 11 -> Gb -- 1 -> Fb$

对于 $\overline{Fb} + \overline{G}$ 是不需要计算的，因为只要是从 \overline{F} 进入的，一定会往下走 $\overline{Fb} -- 11 -> Gb$

路线[2] $\overline{Fa} + \overline{G}$ 或 $\overline{Fb} + \overline{G}$:

$\overline{Fa} -- 11 -> Gb + \overline{Gb} -- 1 -> Z$
 $= \overline{Fa} -- 11 -> Gb -- 1 -> Z$

路线[3] $\overline{Fa} + \overline{G}$:

$\overline{Fa} -- 1 -> Ga + \overline{Ga} -- 15 -> Fa$
 $= \overline{Fa} -- 1 -> Ga -- 15 -> Fa$

路线[4] $\overline{Fb} + \overline{G}$:

$= \overline{Fb} -- 10 -> Fa$ 不继续因为上边已有 \overline{Fa} : $\overline{Fb} -- 10 -> \overline{Fa} -- 1 -> \overline{Ga} -- 15 -> \overline{Fa}$

路线[5] $\overline{Fa} + \overline{G}$:

$\overline{Fa} -- 1 -> Ga + \overline{Ga} -- 1 -> Z$
 $= \overline{Fa} -- 1 -> Ga -- 1 -> Z$

$\overline{Fb} + \overline{G}$ 跳过，同路线[4]一样会回到 \overline{Fa} 。

至此拿到了用于制作状态机的所有路线

1. $\overline{Fa} -- 11 -> Gb -- 1 -> Fb$
2. $\overline{Fa} -- 11 -> Gb -- 1 -> Z$
3. $\overline{Fa} -- 1 -> Ga -- 15 -> Fa$
4. $\overline{Fb} -- 10 -> Fa$
5. $\overline{Fa} -- 1 -> Ga -- 1 -> Z$

4.状态机

但快速处理的方法就是把节点关系唯一化，并不需要像上篇那样大费周章去计算。这样就不会有重复的部分了：

$$\boxed{Fa -- 11 \rightarrow Gb -- 1 \rightarrow Fb} + \boxed{Fa -- 11 \rightarrow Gb -- 1 \rightarrow Z} + \boxed{Fa -- 1 \rightarrow Ga -- 15 \rightarrow Fa} + \boxed{Fb -- 10 \rightarrow Fa} + \boxed{Fa -- 1 \rightarrow Ga -- 1 \rightarrow Z}$$

- | | | |
|--|---------|--|
| <ul style="list-style-type: none">• $Fa -- 11 \rightarrow Gb$• $Gb -- 1 \rightarrow Fb$• $Gb -- 1 \rightarrow Z$• $Fa -- 1 \rightarrow Ga$• $Ga -- 15 \rightarrow Fa$• $Fb -- 10 \rightarrow Fa$• $Ga -- 1 \rightarrow Z$ | 排序一下 -> | <ul style="list-style-type: none">• $Fa -- 11 \rightarrow Gb$• $Fa -- 1 \rightarrow Ga$• $Fb -- 10 \rightarrow Fa$• $Ga -- 15 \rightarrow Fa$• $Ga -- 1 \rightarrow Z$• $Gb -- 1 \rightarrow Fb$• $Gb -- 1 \rightarrow Z$ |
|--|---------|--|

还可以再清理一下数据， Ga 和 Gb 进入 Fa 的路线是一样的，所以可以合并为一个。

- | | | |
|--|--------------------------|--|
| <ul style="list-style-type: none">• $Fa -- 11 \rightarrow Gb$• $Fa -- 1 \rightarrow Ga$• $Fb -- 10 \rightarrow Fa$• $Ga -- 15 \rightarrow Fa$• $G -- 1 \rightarrow Z$• $Gb -- 1 \rightarrow Fb$ | 把颜色换成标识 ->
r=红色, b=蓝色 | <ul style="list-style-type: none">• $Fa_b -- 11 \rightarrow Gb$• $Fa_r -- 1 \rightarrow Ga$• $Fb_r -- 10 \rightarrow Fa$• $Ga_b -- 15 \rightarrow Fa$• $G_r -- 1 \rightarrow Z$• $Gb_b -- 1 \rightarrow Fb$ |
|--|--------------------------|--|

接下来就可以把这个逻辑和数据写入状态机了

首先要确定的是这些节点和填充节点的位置，为了方便计算就直接把所有的节点按序放入数组
每一个要填充的长度就是 [步长+1]，步长中涵盖了目标节点，而 + 1 则是涵盖自己本身
通过这种方式来算出我们需要的状态机内存的长度和节点位置

总长度 = $12 + 2 + 11 + 16 + 2 + 2 = 45$

索引分别在 0, 12, 14, 25, 41, 43

接下来是很重要的 **修改状态触发器**，需要结合 图[3] 来推算出哪些路径会改变铁轨的状态
图[6] 的 紫色路径 中能看出 紫色 的触发点与 图[8] 中左侧长度为 10 的路径有关
进入 F 节点是从上边，也就是对应了 Fa ，所以 $Fa(F$ 的 a 方向)。

所以紫色触发器分别在：

$23 = 25 - 2 \leq "3.F -- 10 \rightarrow F(洋红)" = "Fb -- 10 \rightarrow Fa"$

$39 = 41 - 2 \leq "6.G -- 15 \rightarrow F(青色)" = "Ga -- 15 \rightarrow Fa"$

黄触发器 - 可以看到6号的青色线路是最长的15格，因此比较容易确定。从 G 出发且长度为 15，那就是 G 的上方向那条线路。

黄触发器分别在：

$10 = 12 - 2 \leq "4.F -- 11 \rightarrow G(蓝色)" = "Fa -- 11 \rightarrow Gb"$

$27 = 25 + 2 \leq "6.G -- 15 \rightarrow F(青色)" = "Ga -- 15 \rightarrow Fa"$

到这里就可以把所有数据填入状态机了。

虽然状态机的45个空间已经能放下所有数据，但我们还希望存储两个可变铁轨的颜色信息，因此额外需要 2 个空间来存储这些信息。

5.状态机代码

创建一个数组作为状态机的状态

已知只需要 45 个空间就能实现状态机

```
using System.IO; using System.Drawing;
// 创建一个长度为 47 的空间并填充 空格
var 状态机 = Enumerable.Repeat(" ", 47).ToArray();
// 将对应的节点跳进去 起始 和 终点
状态机[0] = "Fab"; 状态机[12-1] = "Gb";
状态机[12] = "Far"; 状态机[14-1] = "Ga";
状态机[14] = "Fbr"; 状态机[25-1] = "Fa";
状态机[25] = "Gab"; 状态机[41-1] = "Fa";
状态机[41] = "Gr"; 状态机[43-1] = "Z";
状态机[43] = "Gbb"; 状态机[45-1] = "Fb";
// 触发器也跳进去
状态机[23] = "pur"; 状态机[39] = "pur";
状态机[15] = "pur";
状态机[10] = "yel"; 状态机[27] = "yel";
// 车厢加上轨道颜色信息
状态机[45] = ""; 状态机[46] = ""; var 车厢1 = 0; var 车厢2 = 0;
var 初始化数据 = () =>{
    状态机[45] = "blu"; // 45 是紫色轨道
    状态机[46] = "blu"; // 46 是黄色轨道
    车厢1 = 36; 车厢2 = 34;
};
// 显示状态机的内容
初始化数据();
Console.WriteLine(string.Join('|',状态机));
```

179 0.0s csharp - C# Script Code

... Fab| | | | | | | | | | yel| Gb|Far| Ga|Fbr|pur| | | | | | | | | | pur| Fa|Gab| |yel| | | | | | | | | | pur| Fa| Gr| Z|Gbb| Fb|blu|blu

```
// 做一个能把车厢显示在状态机上的函数
var 显示状态机数据 = (bool 含车厢 = true) => {
    var cp = 状态机.Clone() as string[];
    if(车厢1 >= 0){ cp[车厢1] = "[1]"; }
    if(车厢2 >= 0){ cp[车厢2] = "[2]"; }
    Console.WriteLine(string.Join("|",cp));
};
显示状态机数据();
// 可以看到位置是正确的, 在 3 步后 车厢1 会到达紫色触发器
```

180 0.0s csharp - C# Script Code

... Fab| | | | | | | | | | yel| Gb|Far| Ga|Fbr|pur| | | | | | | | | | pur| Fa|Gab| |yel| | | | | | | | | | [2]| [1]| | | | pur| Fa| Gr| Z|Gbb| Fb|blu|blu

[图9] 状态机数据创建

可以看到下边的白色文本输出就是状态机的数据。

这个状态机并不会检查两个车厢是否碰撞，因为篇幅和工作量原因不进行展开。但如果想比较，除了直接比较在状态机上的位置是否一致，还需要在重合的轨道上添加数字。每个重叠的轨道使用同一个数字，这样当两个不同车厢被单步执行时，后执行的车厢可以判断另一个车厢的轨道和自己所在的轨道数字是否相同。如果相同则表示轨道重合相撞。

现在要确定两个车厢的初始位置，因为状态机里并不包含 [图4] 中 车厢2 所在的 X 节点的位置。但是已知起始路线是确定从 X 和 Y 出发到 F，可以找一个与这条路线重合的路径并将它们放入其中。

合适的位置是可以是两个车厢分别前进2格后，车厢2 在节点 B，同时 车厢1 在节点 Y。这两个位置前往 F 的路线是可以在状态机里找到的，可以看到与 "**G -- 15 -> F**(青色)" 重合此时 车厢1 距离 Fa 还有 4 格远，因此很容易找到对应的状态机位置，是 41[下一段起点] - 1[得到上一段终点] - 4 = 36
所以 车厢1 在索引 36 处，车厢2 在 34 处。

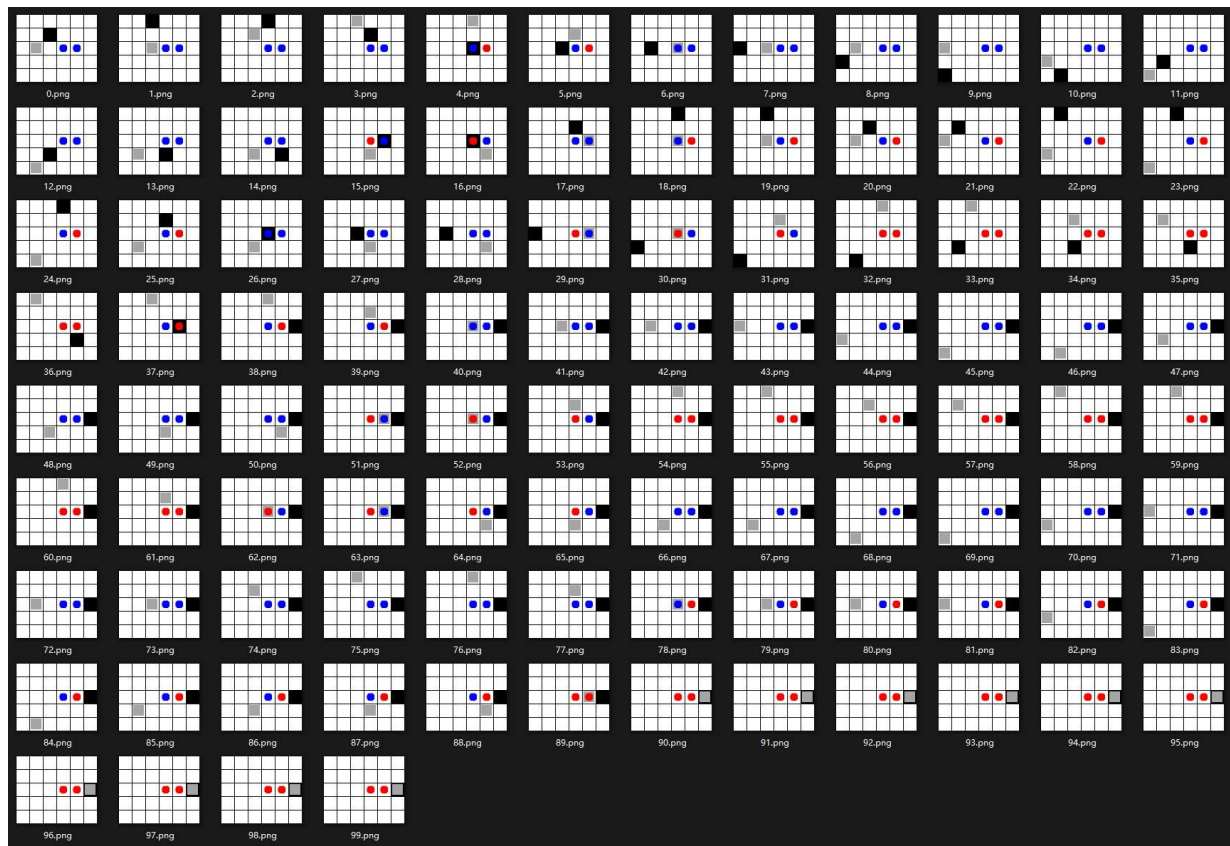
由于数据段并没有做去重，就会导致状态机当前节点和和跳转节点的重复。

比如在 [1] 后边过 pur (紫色触发器) 后的 Fa 时会对跳转进行判断，但是跳转的目标地点则是数据开头的 Fab (因为最开始数据都是蓝色，所以会进到b分支)。如果直接让地址设置为 Fab，就会导致 Fa 和 Fab 都分别执行了一次，但问题在于这两个是同一个地址，于是就会导致出现：
Ga_b -- 15 -> Fa -> Fa_b -- 11 -> Gb...

因此跳转地址应进行 + 1，这样就会变成 **Ga_b -- 15 -> Fa(b) -- 11 -> Gb...**

具体代码见下一页

6.状态机运行映射效果图



[图10] 状态机运行映射

映射关系对照表

```
var mapping = (int i) =>{
    var p = new Point(-1, -1);
    switch(i){
        case -1: p = new Point(5, 2); break;
        case 42: p = new Point(5, 2); break;
        // 蓝色
        case 1: p = new Point(2, 2); break;
        case 2: p = new Point(1, 2); break;
        case 3: p = new Point(0, 2); break;
        case 4: p = new Point(0, 1); break;
        case 5: p = new Point(0, 0); break;
        case 6: p = new Point(1, 0); break;
        case 7: p = new Point(1, 1); break;
        case 8: p = new Point(2, 1); break;
        case 9: p = new Point(3, 1); break;
        case 10: p = new Point(4, 1); break;
        case 11: p = new Point(4, 2); break;
        case 12: p = new Point(3, 2); break;
        case 13: p = new Point(4, 2); break;

        // 品红
        case 15: p = new Point(3, 3); break;
        case 16: p = new Point(3, 4); break;
        case 17: p = new Point(2, 4); break;
        case 18: p = new Point(2, 3); break;
        case 19: p = new Point(1, 3); break;
        case 20: p = new Point(1, 4); break;
        case 21: p = new Point(2, 4); break;
        case 22: p = new Point(3, 4); break;
        case 23: p = new Point(3, 3); break;
        case 24: p = new Point(3, 2); break;

        // 青色
        case 26: p = new Point(4, 1); break;
        case 27: p = new Point(3, 1); break;
        case 28: p = new Point(2, 1); break;
        case 29: p = new Point(1, 1); break;
        case 30: p = new Point(1, 0); break;
        case 31: p = new Point(0, 0); break;
        case 32: p = new Point(0, 1); break;
        case 33: p = new Point(0, 2); break;
        case 34: p = new Point(1, 2); break;
        case 35: p = new Point(2, 2); break;
        case 36: p = new Point(2, 3); break;
        case 37: p = new Point(2, 4); break;
        case 38: p = new Point(3, 4); break;
        case 39: p = new Point(3, 3); break;
        case 40: p = new Point(3, 2); break;

        // 黄色
        case 44: p = new Point(3, 2); break;
    }
    p.Y = 4 - p.Y;
    return p;
};
```

因极端女权运动影响，原文章消失不见。此版本为重新上传的修改版 - 2025/08/06

页1 - 修改了联系方式避免骚扰

页8~9 - 微调数据变化行距

页10 - 映射关系对照表移动到第11页

页11 - 新页面