arXiv:2302.01145v1 [cs.CC] 2 Feb 2023

这个游戏不会自己分析自己

Aviv Adler* Hayashi Ani* Lily Chung* MichaelCoulombe* ErikD. Demaine*Jenny Diomidova* Dylan Hendrickson* Jayson Lynch*

摘要

我们分析了益智电子游戏这个游戏不会自己加载，在这款游戏中，玩家需要将三种不同颜色的数据包，从给定的数据源路由到对应颜色的给定数据汇点。给定数据源、数据汇点和一些预先放置的箭头方块后，我们证明了该博弈属于 Σ_2^P；当数据源周期相同时，问题属于 NP；对于三种颜色以及六个周期相同且需要玩家输入的数据源，问题是 NP完全的；即使不需要玩家输入，对于两种颜色且有许多周期不同的数据源，模拟该博弈也同时是 NP-hard 和 coNP 困难。另一方面，我们刻画了三个数据汇点的哪些位置能够通过放置箭头方块，实现完美的 布局，从而无论数据源放置在哪里，都能保证正确路由；这实际上解决了通常玩法下该博弈的大多数实例。

1 引言

这个游戏不会自己加载 (TGINGTLI) [Ost15]是一款由 Roger “atiaxi” Ostrander 于 2015年 为 Loading Screen Jam 制作的免费游戏。Loading Screen Jam 是一场在 itch.io 上举办的游戏创作活动，本作在 46 款参赛作品中总排名第 7。这场游戏创作活动旨在庆祝美国专利 5,718,632 [Hay98], 到期；该专利涵盖了在电子游戏加载画面中加入小游戏的行为。秉持这一理念，TGINGTLI 是一款以玩家帮助游戏加载自身的三种不同资源——存档数据、游戏玩法和音乐，分别以红色、绿色和蓝色表示——为主题的实时益智游戏。玩家需要在网格单元格上放置箭头，将进入网格的数据引导至对应的汇点单元格。图1展示了一次游戏过程示例。

我们将 TGINGTLI 形式化如下。给定一个 $m \times n$ 网格，其中每个单位方格单元格要么为空，要么包含一个数据汇点，要么包含一个指向四个基点方向之一的箭头。（在实际实现的游戏中，$m = n = 12$ 且初始时不会放置箭头。）每个数据汇点和箭头都有一种颜色（资源）：红色、绿色或蓝色；网格中每种颜色恰好有一个数据汇点。在在线版本（实际实现的版本）中，源会在游戏过程中不断出现；而在此处讨论的离线版本中，所有源都预先已知。请注意，网格的一条外边缘上可能有多个不同颜色的源。最后，有一条加载条，其起始值为整数 k_0，目标值为整数 k^*。

在博弈过程中，每个源点会定期产生其对应颜色的数据包，并以恒定速度进入网格。如果数据包进入了与其颜色相同的箭头所在单元格，就会沿箭头所指的方向转向。（其他颜色的箭头会被忽略。）如果数据包到达了与其颜色相同的汇点，就会消失，加载条增加一个单位的数据。如果数据包到达了颜色不匹配的汇点，或完全离开网格，就会消失，加载条减少一个单位的数据，这称为受到伤害。数据包也可能沿着箭头循环移动，因而无限期地留在网格中；这不会使加载条增加或减少

*计算机科学与人工智能实验室，麻省理工学院，美国马萨诸塞州剑桥市 02139，{adlera,joshuaa,lkdc,mcoulomb,edemaine,diomidova,dylanhen,jaysonl}@mit.edu

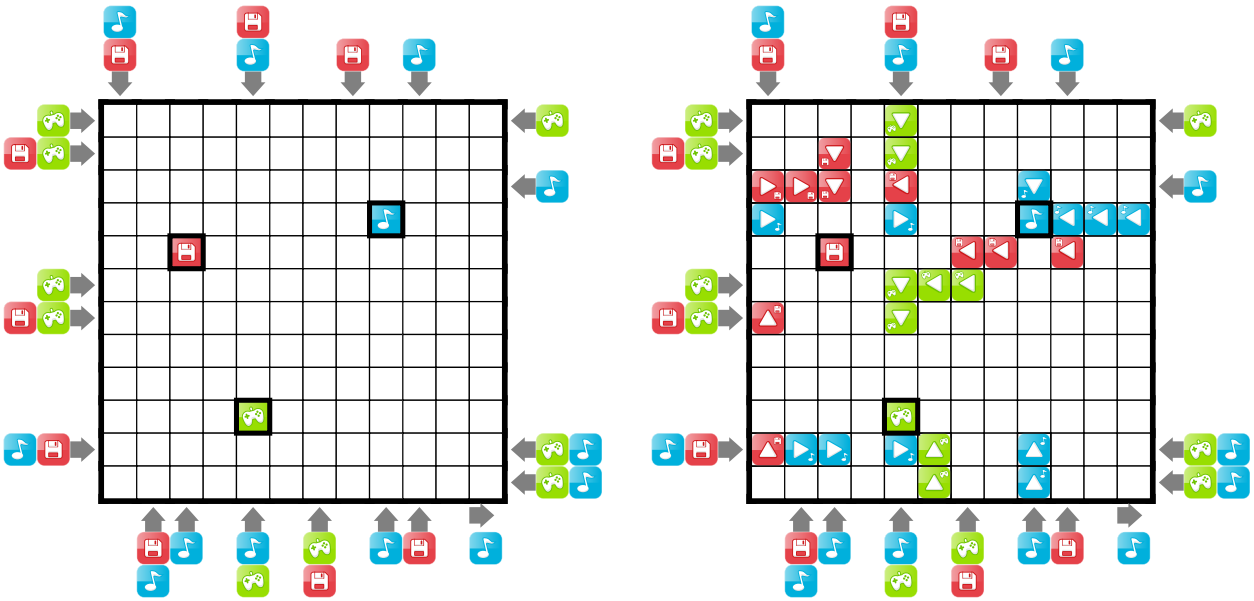


图1：左：TGINGTLI中现实世界第16关的（最终）输入。右：成功通关，将每个数据包都路由到对应的汇点。

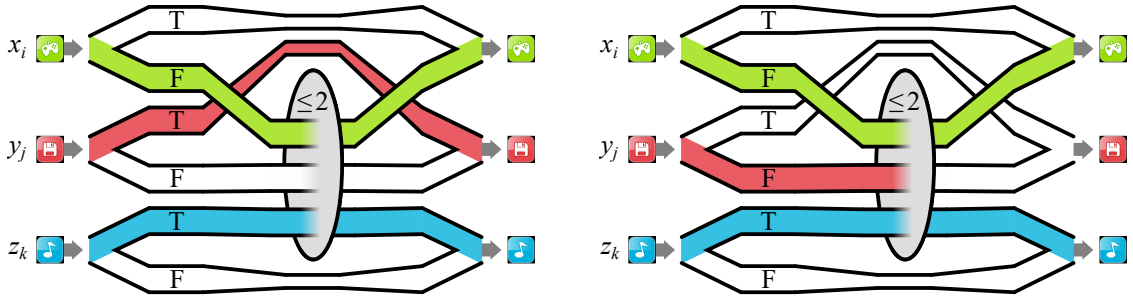
。玩家可以随时在一个空单元格中永久放置一个箭头，箭头可以是任意颜色，并指向四个方向中的任意一个。如果加载条达到目标数值 k^*，玩家就获胜；但如果加载条降至零以下，玩家就会失败。

在第 2 节中，我们证明 TGINGTLI 判定问题是 NP困难的：给定网格的描述（包括源、汇点和预先放置的箭头），玩家能否放置箭头并获胜？即使只有六个源和三种颜色，这种归约仍然成立；它引入了一个新问题——***3DSAT***，其中变量分为三种不同的颜色，每个子句都混合了三种颜色的变量。在第3节中，我们引入了关于源的周期行为的更详细模型，并说明周期各不相同的多个源会使博弈的胜利问题同时具有 NP 和 coNP 困难性，即使没有玩家输入（只需模拟博弈）。从积极的一面来看，我们证明这些问题属于 Σ_2^P；当所有源周期都相同时（如我们第一个 NP困难性证明中的情形），问题属于 NP，因此这种情形实际上是 NP完全的。

在第4节中，我们考察实际实现的游戏中关卡的初始状态：网格上已放置汇点，但没有预先放置箭头。我们完整刻画了何时存在箭头的完美布局，使得无论数据包的源点位于何处，都能保证它们被路由到正确的汇点。特别是，这一结果为实际实现的游戏中大多数汇点布局提供了获胜策略。值得注意的是，由于该解法不依赖源点的位置，因此适用于在线设置。

2 三种颜色和六个源点的 NP困难性

我们首先通过从一个新问题归约，证明 TGINGTLI 是 NP-hard。这个问题称为 三维***SAT***（***3DSAT***），其定义类比于三维匹配（3DM）。3DSAT 是 3SAT 的一种变体：除了提供一个 3CNF公式作为输入外，还要为该 CNF公式中的每个变量指定三种颜色（红色、绿色或蓝色）之一，并要求 CNF公式中的子句都是三色子句，即每个子句恰好包含一个（可能取反的）各色变量。



(a) 满足的子句　　(b) 不满足的子句

图 2：我们 NP困难性归约的示意图。

引理2.1。 *3DSAT* 是 *NP*完全的。

证明。 我们通过将一个 3CNF公式 F 转换为一个三维 CNF公式 F'，把 3SAT 归约到 3DSAT。对于 F 中的每个变量 x，我们在 F' 中创建三个变量 $x^{(1)}, x^{(2)}, x^{(3)}$（它们预期是 x 的三个不同颜色的相等副本），并向 F' 添加六个子句以强制满足 $x^{(1)} = x^{(2)} = x^{(3)}$:

$$\left.\begin{aligned} \neg x^{(1)} \vee x^{(2)} \vee x^{(3)} &\iff (x^{(1)} \to x^{(2)}) \vee x^{(3)} \\ \neg x^{(1)} \vee x^{(2)} \vee \neg x^{(3)} &\iff (x^{(1)} \to x^{(2)}) \vee \neg x^{(3)} \end{aligned}\right\} \iff x^{(1)} \to x^{(2)}$$

$$\left.\begin{aligned} x^{(1)} \vee \neg x^{(2)} \vee x^{(3)} &\iff (x^{(2)} \to x^{(3)}) \vee x^{(1)} \\ \neg x^{(1)} \vee \neg x^{(2)} \vee x^{(3)} &\iff (x^{(2)} \to x^{(3)}) \vee \neg x^{(1)} \end{aligned}\right\} \iff x^{(2)} \to x^{(3)}$$

$$\left.\begin{aligned} x^{(1)} \vee x^{(2)} \vee \neg x^{(3)} &\iff (x^{(3)} \to x^{(1)}) \vee x^{(2)} \\ x^{(1)} \vee \neg x^{(2)} \vee \neg x^{(3)} &\iff (x^{(3)} \to x^{(1)}) \vee \neg x^{(2)} \end{aligned}\right\} \iff x^{(3)} \to x^{(1)}$$

因此，左侧的子句等价于蕴含环 $x^{(1)} \implies x^{(2)} \implies x^{(3)} \implies x^{(1)}$，而它又等价于 $x^{(1)} = x^{(2)} = x^{(3)}$。

对于 F 中每个子句 c，若第一个文字使用变量 x，第二个文字使用变量 y，第三个文字使用变量 z，我们就在 F' 中创建一个对应的子句 c'，使用 $x^{(1)}$、 $y^{(2)}$和 $z^{(3)}$ （其否定形式与 c 中的相同）。因此， F' 中的所有子句（包括上文的变量重复子句）都在第 i个文字中使用形如 $x^{(i)}$ 的变量，其中 $i \in \{1, 2, 3\}$，所以我们可以相应地对这些变量进行 3 着色。 □

定理 2.2。 即使只有三种颜色，且六个源点的周期相同，*TGINGTLI* 仍是 *NP-hard*。

证明。 我们的归约来自 3DSAT。图 2 给出了一个高层次的示意：每个变量构件都有两条可能的路径，供对应颜色的数据包流通过；每个子句构件最多允许两种颜色的数据包成功通过。当至少一个变量使子句成立时，子句构件会让其他变量成功通过；否则，至少有一个数据包流会进入一个环。同一种颜色的变量会串接起来，以重复使用该颜色的数据包流。

具体来说，游戏棋盘上的大多数单元格都会预先填好，只留少数几个空单元格（在图中用问号表示），供玩家填写。

对于每种着色，例如红色，我们都在构造的左边缘放置一个红色源构件。然后，对于依次出现的每个红色变量 x，我们在红色数据流的末端放置一个图 4 所示的变量构件。为防止数据包进入循环，玩家必须选择将数据流向上或向下发送，从而使其沿两条向右的路径之一前进，

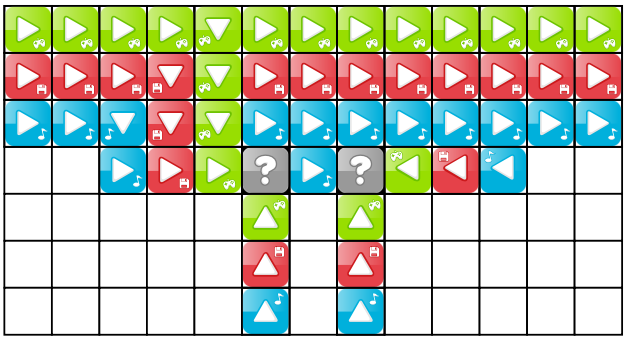


图 3：子句构件。最多两股代表假文字的数据流可以通过该构件（在“?”单元格中放置向上的箭头），而不进入环。放置任何其他方向的箭头也会使数据流进入环。

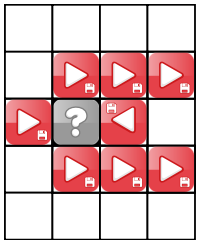


图 4：变量构件让玩家能够将数据包从源点路由到两条文字路径中的一条。

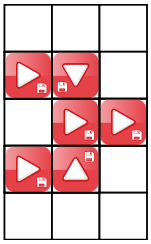


图 5：合并构件将两条文字路径重新合并为一条。

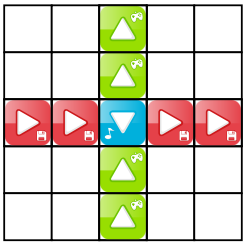


图 6：用于连接颜色相同或不同的两条文字路径的交叉构件。（中心单元格的颜色与两条路径都不同。）

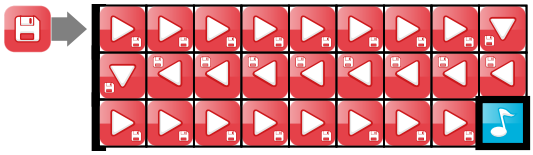


图 7：伤害构件会在所需的启动延迟后，以单位速率造成伤害。

分别表示文字 x 和 x。数据包流经过的路径被视为假；空路径则被视为 真。

然后，我们对每条文字路径进行路由，使其依次经过包含该文字的每个子句。图 6 展示了一个用于实现这种路由的交叉构件。（注意，即使两条线颜色相同，它仍然有效。）

图 3 展示了子句构件，它最多允许两条数据包流通过。如果三个文字都为假，那么至少有一条数据流必须被置入一个环中。另一方面，如果子句得到满足，那么承载数据的文字路径就可以将数据传递给下一个子句，依此类推。请注意，对于三种颜色，经过子句的绕行路径长度都相同。

红色变量 x的所有用途结束后，我们将与 x 和x 对应的两条红色文字路径延长至长度相同，然后使用图 5 中的合并构件将它们重新合并。接着，我们将这条红色路径连接到下一个红色变量的变量构件，如此继续。最后，在所有红色变量之后，我们将红色数据流连接到一个红色汇点。

我们将红色、绿色和蓝色数据流延长至相同长度 ℓ。如果玩家成功满足所有子句，那么在初始延迟 ℓ 后，加载条将增加 3 个单位（每种颜色 1 个单位）。我们设置参数，使玩家在这种情况下获胜：$k^* - k_0 = 3$。否则，加载速度至多为 2。为了确保玩家在这种情况下落败，我们添加图 7 中的 3 个伤害构件，每个构件都在初始延迟 $\ell + 1$ 后以每周期 1 点的速度造成伤害。因此，每周期的净值为 -1，所以即使 k_0 很大，玩家最终也会落败。 □

这一 NP 困难性结果并不需要对源及其发送数据包的方式作出非常具体的建模。要判断该问题是否属于 NP，我们需要一个更具体的源模型，下一节将对此进行讨论。

3 Σ_2^P 中的成员资格与源周期性带来的困难性

本节考察不同源点发送数据包时周期可能不同所带来的影响。具体而言，我们将证明，通过精心设定周期，再结合博弈长度无界这一条件，判定 TGINGTLI 的结果具有 NP-hard 和 coNP困难性，即使玩家不进行任何操作也是如此。反过来，即使允许玩家输入，我们也证明该问题属于 Σ_2^P。

3.1 模型与问题

更准确地说，我们将每个源点 s 建模为以自身周期 p_s向网格发送其着色对应的数据包，并设定一个较短的预热时间 w_s，在此期间，源点可以发送一种更具体的数据包模式。实际实现的 TGINGTLI 中，每个源点的预热行为是：创建后先等待 5 秒再发送第一个数据包，随后间隔 2 秒、1.999 秒、1.998 秒依次发送数据包，直到达到固定的 0.5 秒周期。从技术上讲，这相当于一个持续 1881.25 秒、发送 1500 个数据包的预热阶段，之后周期为 0.5 秒。

在模拟问题中，给定网格的初始状态，以及一份带时间戳的事件列表，其中记录了每个源点在预热阶段发射数据包的时间、每个源点开始周期行为的时间，以及玩家放置每个箭头的时间。我们假设时间戳以二进制编码，但为了让问题相对简单，周期和预热时间以一进制编码。问题要求预测玩家能否获胜；也就是说，在发生失败之前，加载条能否到达 k^*。

在博弈问题中，我们并未给出玩家的箭头放置情况。如果允许使用非确定性算法，那么博弈问题可归约为模拟问题：只需猜测我们会在何处、何时放置哪些箭头即可。

解决模拟问题的一种自然方法，是从初始状态开始模拟博弈，直到依次到达各个事件。具体来说，给定一个游戏状态（包含汇点、具有不同周期和偏移量的源点、已放置的箭头，以及各位置飞行中的数据包数量的网格）和一个未来时间戳 t，我们希望确定时间 t时的游戏状态。借助这一计算，我们可以通过“跳过”事件之间的时间，快速求出博弈的未来状态。在一个 $m \times n$ 网格上，共有 $O(mn)$个事件，因此我们可以通过模拟事件之间多项式数量的阶段，确定任意时间 t 时的游戏状态。

这项计算很容易完成。给定时间 t，我们可以分别除以每个源点的周期和每个箭头循环的周期，从而确定每个源点产生多少个数据包，以及箭头会将它们路由到何处——是送至影响负载的汇点、离开网格、困在环中，还是在环外飞行中——然后汇总这些影响，得出新的负载量以及各位置的数据包数量。

然而，仅能计算未来状态并不足以解决模拟和博弈问题，因为在某个中间时刻，负载量可能会降至0以下，或者达到目标量 k^*。尽管如此，这足以证明这些问题属于 Σ_2^P：只需猜测一个胜利时刻，并验证之前不存在失败时刻：

引理 3.1。 模拟问题和游戏问题都属于 Σ_2^P。

证明。 如果存在一个时间点，此时玩家获胜，且所有更早的时间点都没有失败，那么玩家就获胜。为解决模拟问题，非确定性地猜测获胜时间，并通过计算该时间点的状态来验证此时是否获胜。然后使用一次 coNP 查询检查该时间点之前是否没有失败；这同样利用了能够快速计算任意单个时间戳对应状态的能力。

为了解决游戏问题，我们首先猜测箭头放置的具体细节，然后像之前一样解决由此得到的模拟问题。
□

一种较简单的情况是，所有源在预热后都具有相同的周期，这与实际游戏中的实现方式一致。定理 2.2 证明了这一版本的博弈是 NP-hard，现在我们可以证明它是 NP完全：

引理 3.2。 如果所有源在经过多项式数量的时间步 t_p后都具有相同的多项式长度周期，那么模拟问题属于 *P*，游戏问题属于 *NP*。

证明。 在这种情况下，我们可以通过显式模拟超过所有数据包路径长度的时间，检查事件之间每个阶段是否会胜利或失败；此时加载条的值会以各源共有的周期循环。（箭头循环的周期可能不同，但不会影响加载条，因此检查胜负时无需考虑。）我们跳过每个阶段，并在此过程中检查是否胜利或失败。如果游戏持续到最后一个事件之后，我们就测量一个周期内净得分变化的符号。如果为正，玩家最终会胜利；如果为负，玩家最终会失败；如果为零，游戏将永远继续下去。
□

在本节余下部分，我们考虑这样一种情况：每个源点都可以被赋予任意整数周期，并且该周期不会随时间变化。

3.2 周期和阈值问题

源周期各不相同时，难点在于博弈整体的周期行为可能具有极大的（指数级）周期。

我们可以通过周期和阈值问题对这一难点建模，定义如下。给定一个函数 $f(x) = \sum_i g_i(x)$，其中每个 g_i 都有一进制整数周期 T_i 和一进制最大绝对值 M_i。此外，还给定一个一进制整数 $\tau > 0$ 和一个二进制整数时间 x^*。目标是确定是否存在一个整数 x 属于 $[0, x^*)$，使得 $f(x) \geq \tau$。（直观地说，达到 τ 就意味着获胜。）

定理 3.3。 周期和阈值问题是 *NP*完全的，即使在以下限制条件下也是如此：

1. 每个 $|g_i|$ 都是单热函数，即在其周期内，除恰好一个 x 之外，处处都有 $g_i(x) = 0$，而在该位置有 $g_i(x) = \pm 1$。*2.* 给定一个一进制整数 $\lambda < 0$，满足对于所有 $0 \leq x < x^*$，都有 $f(x) > \lambda$，且 $f(x^*) \leq \lambda$。（直观而言，降至 λ 对应于失败。）

证明。 首先，该问题属于 NP：我们可以猜测 $x \in [0, x^*)$，然后在多项式时间内评估 $f(x) \leq c$。

对于 NP困难性，我们从 3SAT 进行归约。我们将每个变量 v_i 映射为除 2 以外的第 i个素数 p_i。利用中国剩余定理，我们可以用单个整数 $0 \leq x < \prod_i p_i$ 表示一个布尔赋值 ϕ：当 ϕ 将 v_i 设为真时，$x \equiv 1 \bmod p_i$；当ϕ 将 v_i 设为假时，$x \equiv 0 \bmod p_i$。（此映射不会使用 x 模 p_i的其他取值。具体而言，它会将$x \equiv -1 \bmod p_i$ 留空，因为 $p_i \geq 3$。）

接下来，我们将每个子句（例如 $C = (v_i \vee v_j \vee \neg v_k)$）映射为函数

$$g_C(x) = \max\{[x \equiv 1 \mod p_i], [x \equiv 1 \mod p_j], [x \equiv 0 \mod p_k]\},$$

也就是说，正文字检查 $x \equiv 1$，否定文字检查 $x \equiv 0$。当且仅当 x 对应一个满足 C的布尔赋值时，此函数的值恰好为 1。此函数的周期为$p_i p_j p_k$，其一进制值有多项式上界。令 τ 为子句数，则存在一个值x，使得总和为 τ，当且仅当该 3SAT公式存在满足赋值。（若令τ 更小，我们就可以从最大3SAT归约。）

为实现性质1，我们将每个 g_C 函数拆分为多项式数量的单热函数（数量不超过周期）。事实上，每个子句只需七个函数，分别对应该子句的一种满足赋值。

为实现性质2，对于每个素数 p_i，我们加入函数 $h_i(x) = -[x \equiv -1 \text{ 模 } p_i]$。此函数仅在 x 的未使用值上取 -1，这些值不对应任何赋值 ϕ，因此不会影响上述论证。令 $-\lambda$ 为素数（变量）的数量，并令 $x^* = \prod_i p_i - 1$，则有 $\sum_i h_i(x^*) = \lambda$，因为对于所有 i，$h_i(x^*) \equiv -1 \text{ 模 } p_i$；而对于所有$0 \le x < x^*$，$\sum_i h_i(x) > \lambda$。所有用到的值 x 都小于 x^*。

总之，$f(x)$ 是所构造函数之和，我们得到了所需的性质。 □

3.3 TwoColors 的模拟问题的困难性

我们可以利用周期和阈值问题的困难性，证明模拟 TGINGTLI 是困难的，即使没有玩家输入也如此。

定理 3.4。 模拟 *TGINGTLI* 并确定玩家是否获胜是 *NP-hard* 的，即使只有两种颜色也是如此。

证明。 我们从定理 3.3 证明为 NP完全的周期和阈值问题进行归约。

对于每个具有单热值 $g_i(x_i) = 1$ 和周期 T_i 的函数 g_i，我们创建一个蓝色源 b_i 和一个红色源 r_i，二者具有相同的发射周期 T_i，并将来自这些源的红色数据包和蓝色数据包传送至蓝色汇点。通过调整路径长度和/或源的预热时间，我们安排红色数据包在每个蓝色数据包之后一个时间单位到达；蓝色数据包发生在满足 $\equiv x_i \bmod T_i$ 的时刻。因此，在时刻 x_i，对加载条数值的净影响为 $+1$，但在时刻 $x_i + 1$ 会回到 0。类似地，对于每个具有单热值 $g_i(x_i) = -1$ 的函数 g_i，我们执行相同的构造，但交换红色和蓝色的角色。

设定 $k_0 = -\lambda - 1 \ge 0$ 时，当函数 g_i 的总和降至 λ 时，加载条恰好会变为负值（玩家也恰好会输）。设定 $k^* = k_0 + \tau$ 时，当函数 g_i 的总和升至 τ 时，加载条恰好会达到 k^*（玩家也恰好会赢）。 □

这个 NP困难性证明所依赖的博弈特性与第 2 节中的证明截然不同：它不使用玩家输入，而是依赖不同源点的周期有所变化（但以一进制表示时数值较小）。更有意思的是，我们还可以证明同一问题是 coNP 困难：

定理 3.5。 模拟 *TGINGTLI* 并判定玩家是否胜利是 *coNP* 困难的，即使只有两种着色。

证明。 我们从周期和阈值问题的补问题进行归约；根据定理 3.3，该问题是 coNP-完全的。补问题的目标是确定是否不存在 整数 $x \text{in } [0, x^*)$ 使得 $f(x) \ge \tau$。核心思路是将所有值取反，从而互换获胜与失败的角色。

对于每个函数 g_i，我们按照定理 3.4 中相同的方式构造两个源点，并将它们连接到一个汇点，但将值取反：如果 $g_i(x_i) = \pm 1$，那么我们设计数据包，使其在时间 x_i 产生 ∓ 1 的净效果，在其他时间则为 0。

设定 $k_0 = \tau - 1$ 时，只有当函数 g_i 的和升至 τ 时，加载条才会降至负值（玩家也会输），也就是说，周期和阈值问题的答案为“是”。设定 $k^* = k_0 - \lambda$ 时，只有当函数 g_i 的和降至 λ 时，加载条才会达到 k^*（玩家也会获胜），也就是说，周期和阈值问题的答案为“否”。

□

4 完美布局的刻画

假设给定一个棋盘，除三个数据汇点的位置外，其他位置均为空。是否可以放置箭头，使所有可能的输入数据包都被路由到正确的汇点？我们将这种箭头配置称为完美布局。特别是，这种布局无论数据源如何，都能确保胜利。本节将全面刻画哪些棋盘和汇点位置允许形成完美布局。我们的一些结果适用于一般数量 c 的颜色，但完整的刻画依赖于 $c = 3$。

4.1 颜色而非箭头

我们首先证明，无须考虑箭头的方向，只需考虑它们在网格中的颜色和位置。

设 B 是一个棋盘，其汇点位置已指定，并设 ∂B 是 B 边界上的边集。假设我们已将颜色分配给 B 的单元格，且这些颜色与汇点的颜色一致；设 C_i 是着成 i 色的网格单元格集合。当且仅当两个单元格，或一个单元格与一条边界边 $e \in \partial B$ 位于同一行或同一列，且它们之间没有颜色不同于 i 的汇点时，我们称 C_i 中的这两个单元格，或 C_i 中的单元格与边界边彼此可见。设 G_i 是一个图，其顶点集为 $C_i \cup \partial B$；当一对顶点彼此可见时，它们之间就有一条边。

引理 4.1。设 B 是一个汇点位置已指定的棋盘。那么，B 存在完美布局，当且仅当可以为网格中其余单元格选择颜色，使得对于每种颜色 i，图 G_i 都是连通的。

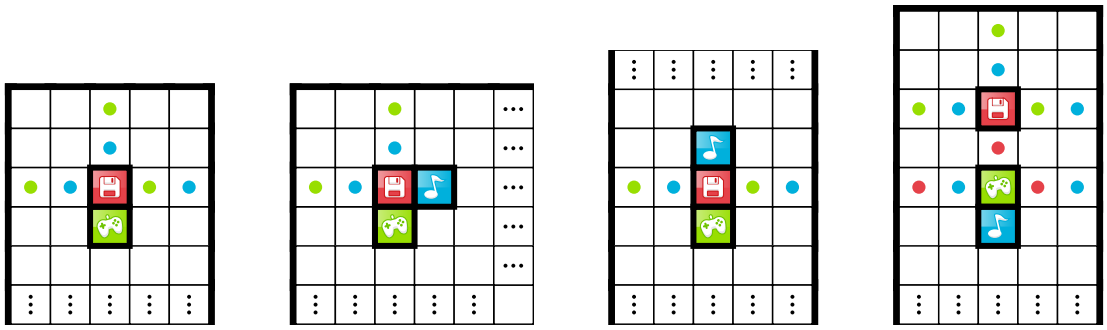
证明。($\Longrightarrow$) 不失一般性，假设完美布局所含箭头数尽可能少。用完美布局中汇点和箭头的颜色给棋盘上的单元格着色。（如果完美布局中某个单元格为空，就将其着成与相邻单元格相同的颜色；这不会影响连通性。）固定一种颜色 i。每条边界边都通过颜色为 i 的数据包从该边进入时所经过的路径，连通到颜色为 i 的汇点。（特别是，这条路径不能经过其他颜色的汇点。）根据完美布局中箭头数最少这一条件，每个颜色为 i 的箭头都包含在这样一条路径中。因此 G_i 是连通的。

($\Longleftarrow$) 我们将用一个同色箭头替换每个单元格，从而构成一个完美布局。具体来说，对于每种颜色 i，选取一棵以颜色为 i的汇点为根的 G_i 生成树，并将树中从子节点指向父节点的方向设为箭头方向。根据连通性，从边界边进入的任何数据包都会沿树向上移动至根节点，从而被路由到正确的汇点。

□

4.2 不可能的棋盘

接下来，我们证明某些棋盘无法实现完美布局。首先，我们讨论汇点与边界或彼此之间距离过近的棋盘。然后，我们给出关于棋盘尺寸的基于面积的约束。



(a) (b) (c) (d)

图 8：没有完美布局的汇点配置。圆点表示被迫采用特定颜色的箭头（每行或每列内的颜色可进行置换）。

引理 4.2。 如果汇点与网格边界之间的某一行或某一列中，空白单元格少于 $c-1$ 个，则不存在完美布局。

证明。 完美布局必须阻止其他 $c-1$ 颜色的数据包从此边界进入并抵达该汇点；这需要为 $c-1$ 箭头留出足够空间。 □

引理 4.3. 对于 $c=3$，如果满足以下任一条件，棋盘就没有完美布局（如图 *8* 所示）：

（*a*）一个数据汇点距三个边界均为两个单元格，并且与另一个汇点相邻；（*b*）一个数据汇点距两个相邻边界均为两个单元格，并且与另外两个汇点相邻；（*c*）一个数据汇点距两个相对边界均为两个单元格，并且与另外两个汇点相邻；或者（*d*）一个数据汇点距三个边界均为两个单元格，并且与一对相邻汇点之间隔着一个空白单元格。

证明。 不失一般性地，假设在每种情况下，最先提到的汇点都是红色。

情况 (a)、(b) 和 (c)：红色汇点与边界之间的单元格对（图中以圆点标出）必须分别放置绿色箭头和蓝色箭头，以确保这些数据包不会到达红色汇点。因此，与红色汇点处于同一行或同一列的位置都没有空位可放置红色箭头，所以来自其他行或列的红色数据包无法到达红色汇点。

情况 (d)：红色汇点与边界之间的单元格对（图中以绿色和蓝色圆点标出）必须分别放置绿色箭头和蓝色箭头，以确保这些数据包不会与红色汇点相撞。因此，红色汇点与另一对汇点之间的空白方格必须放置一支指向红色汇点的红色箭头，以便来自其他行和列的数据包能够到达红色汇点。根据对称性，不妨假设离红色汇点最近的汇点是绿色汇点。与其他情况一样，绿色汇点与边界之间的单元格对必须放置红色箭头和蓝色箭头。因此，没有绿色箭头可用于将来自其他行或列的绿色数据包引导至绿色汇点。

□

现在我们来证明，存在完美布局的棋盘尺寸必须满足的一项约束。

引理 4.4。 设 c 为颜色数。假设一个棋盘上存在完美布局，其中 m和 n 分别表示行数和列数，而 p 和 q 分别表示至少包含一个汇点的行数和列数。那么

$$c(m+n)+(c-2)(p+q)\leq mn-c. \tag{4.1}$$

证明。 每个 $m-p$ 未占用的行都必须包含 c 个垂直箭头，以便将各着色的数据包从该行引出。每个 p 已占用的行都必须在最左侧汇点的左边包含 $c-1$ 个垂直箭头，以便将来自左边界边、着色错误的数据包从该汇点引开；同理，最右侧汇点的右边也必须有 $c-1$ 个垂直箭头。因此，总共需要 $c(m-p)+2(c-1)p=cm+(c-2)p$ 个垂直箭头。根据同样的论证，必须有 $cn+(c-2)q$ 个水平箭头，因此箭头总数为 $c(m+n)+(c-2)(p+q)$。可供放置箭头的单元格有 $mn-c$ 个，由此得证。 □

除插入空行或空列、旋转、镜像和重新着色外， $c=3$ 汇点彼此之间可能存在六种不同的配置，如图 9 所示。我们根据汇点的这种配置（C、I、J、L、Y 或 /）定义棋盘的类型 。



图 9：三个汇点在旋转、镜像、重新着色以及移除空行后可能出现的六种配置。

棋盘类型决定 p 和 q 的值，因此也决定最小棋盘尺寸，具体如下。定义棋盘的尺寸为至少 $m\times n$，是指它至少有 m 行和 n 列，或至少有 n 行和 m 列。

引理 *4.5. </style}*若要存在满足 $c=3$的完美布局，必须满足：

- 类型为 *Y* 或 / 的棋盘尺寸至少为 7×8。
- 类型为 *C* 或 *J* 的棋盘尺寸至少为 7×8 或 6×9。
- 类型为 *L* 的棋盘尺寸至少为 7×7 或 6×9。
- 型棋盘的尺寸至少为 7×7、 6×9 或 5×11。

证明。 这些界限由引理 4.4 以及引理 4.2 中关于汇点必须能够放置在距离边界至少两个单元格处的要求共同推出。 □

4.3 构造完美布局

在本节中，我们将完成对具有完美布局的 $c=3$棋盘的刻画。我们证明，引理 4.2、4.3 和 4.5 是构造完美布局的唯一障碍：

定理 *4.6. </style}*棋盘上有 $c=3$ 个汇点，当且仅当以下条件全部成立时，才存在完美布局：

1. 所有汇点与边界的距离至少为两个单元格（引理 *4.2*）。*2.* 棋盘不包含图 *8* 中的四种无解构型中的任何一种（引理 *4.3*）。*3.* 棋盘符合引理 *4.5* 中的尺寸限制。

如果棋盘的尺寸属于引理 4.5 为其类型定义的极小尺寸之一，我们就称其为极小棋盘。证明定理 4.6 的策略是将问题归约为有限个极小棋盘，然后通过计算机对其进行验证。为此，我们将从非极小棋盘中移除空行和空列，以减小其尺寸；我们将证明，在此过程中始终可以保持上述条件不变。

引理 4.7。 满足定理 *4.6* 三个条件的所有极小棋盘都具有完美布局。

证明。 证明方法是对所有此类极小棋盘进行穷举计算机搜索。我们编写了一个 Python 程序，用于生成所有可能的棋盘模式，将每个完美布局问题归约为可满足性模理论（SMT）问题，然后使用 Z3 [Mic] 求解。搜索结果见附录 A。■

如果 B_0和 B_1是棋盘，那么我们定义 $\boldsymbol{B_0 \lessdot B_1}$表示： B_0可以通过从 B_1中移除单个空行或空列得到。

引理 4.8。 如果 $B_0 \lessdot B_1$ 和 B_0 有完美布局，那么 B_1 也有完美布局。

证明。 由对称性，不妨考虑 B_1 增加了一行的情形。根据引理 4.1，只需证明我们可以为新行的单元格着色，同时保持每种颜色的连通性。为此，我们复制相邻行中各单元格（包括汇点）的颜色。由原着色的连通性可知，所得着色也是连通的。

□

引理 4.9。 设 B_1 是一个非极小棋盘，且满足定理 *4.6* 的三个条件。那么存在一个棋盘 B_0，它也满足全部三个条件，并且 $B_0 \lessdot B_1$。

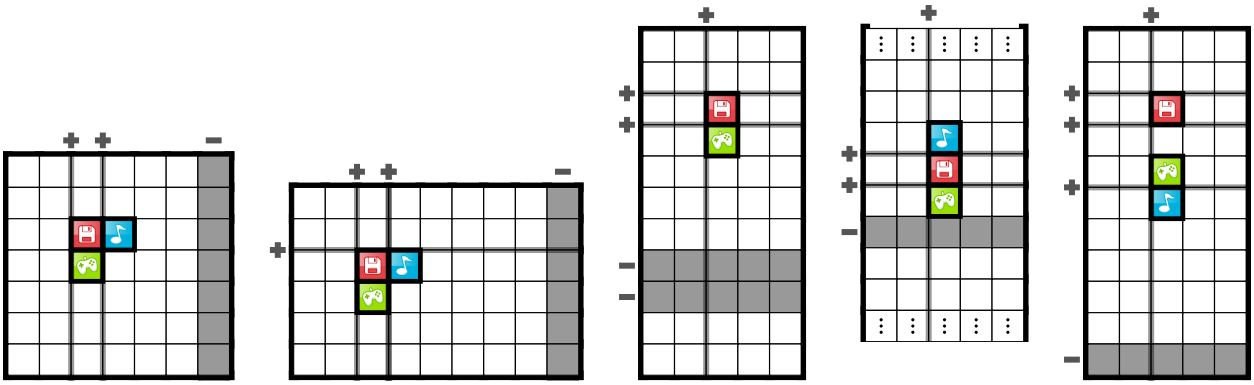
证明。 由对称性，不妨设 B_1 的行数 m 不是最小的。通过从 B_1 中移除一行（该行既不属于前两行或后两行，也不包含汇点），我们得到一个棋盘 B_0'，它满足条件 (1) 和 (3)，且 $B_0' \lessdot B_1$。如果 B_0' 也满足条件 (2)，那么结论成立；因此，我们可以假设它不满足条件 (2)。

那么，B_0' 必定包含四种无解构型之一，而 B_1 则是通过插入一条空行或空列来消除该无解构型得到的。图 10 展示了 B_0' 的所有可能情况，以及可插入行或列的位置，从而得到 B_1 的相应可能情况。（B_0' 可能还包含图中未显示的其他空行和空列，但这不会影响证明。）对于每种可能情况，图 10 还标出了另一条可从 B_1 中删除的行或列；删除后可得到 $B_0 \lessdot B_1$，其中 B_0 满足全部三个条件。

□

定理 *4.6* 的证明。 由引理 4.2、4.3 和 4.5 可知，具有完美布局的所有棋盘都必须满足该定理的三个性质。我们通过对棋盘大小进行归纳，证明这些性质也构成充分条件。作为基础情形，根据引理 4.7，该结论对极小棋盘成立。对于非极小棋盘 B_1，引理 4.9 表明，存在一个更小的棋盘 B_0，它满足全部三个条件，并且 $B_0 \lessdot B_1$。根据归纳假设，B_0 具有完美布局。引理 4.8 表明，B_1 也具有完美布局。

□



(a) L, 7×7 (b) L, 6×9 (c) I, 5×11 (d) I, 5×11 (e) I, 5×11

图 10：满足条件 (1) 和 (3) 但不满足条件 (2) 的所有棋盘，忽略旋转、镜像和重新着色。可以在标有“+”的任意位置插入一行或一列空行或空列，从而得到满足全部三个条件的棋盘。随后移除标有“–”的行或列，仍可保留这些条件。在情况 (c) 中，移除一行不包含蓝色汇点的行。在情况 (d) 中，⋮表示零行或多行。

5 个开放问题

主要的复杂度开放问题是 TGINGTLI 是否为 Σ_2^P-完全问题。鉴于我们关于 NP 和 coNP困难性的结果，我们怀疑这一点确实成立。

我们也可以针对限制更多的博弈版本提出复杂度问题。例如，如果棋盘的行数是常数，会怎样?

在刻画完美布局时，我们看到许多引理可以推广到不同数量的颜色。进一步探索这个博弈，并尝试刻画使用三种以上颜色的完美布局，或许会很有意思。

一个相关问题是：哪些棋盘和汇点配置允许存在一种无损布局，使得任何从边界进入的数据包要么到达颜色正确的汇点，要么陷入无限循环。这种布局可以避免失败；在实际实现的博弈中，这种布局实际上会赢得博弈（因为只要没有足够的空间放置新的源点，玩家就会获胜）。我们能否像刻画完美布局那样刻画这类布局?

完美布局和无损布局能够应对任何可能的源点。然而，对于那些不存在完美布局或无损布局的棋盘，如果能有一种算法判断给定的一组源点或数据包序列是否仍可通过放置箭头而赢得该棋盘上的博弈，那就很有用。由于棋盘起初除了汇点之外是空的，我们的困难性结果并不适用。

谜题通常以具有唯一解为佳。因此，自然会引出这样的问题：它是否具有 ASP 困难性，以及解的计数是否为 #P 困难。

致谢

本研究起源于麻省理工学院 Erik Demaine 在2019年春季讲授的《算法下界：趣味困难性证明》（6.892）课程中的开放问题求解活动。我们感谢该课程的其他参与者在相关讨论中提供帮助，并营造了鼓舞人心的氛围。特别感谢 Quanquan C. Liu 对早期成果所作的有益讨论和贡献。

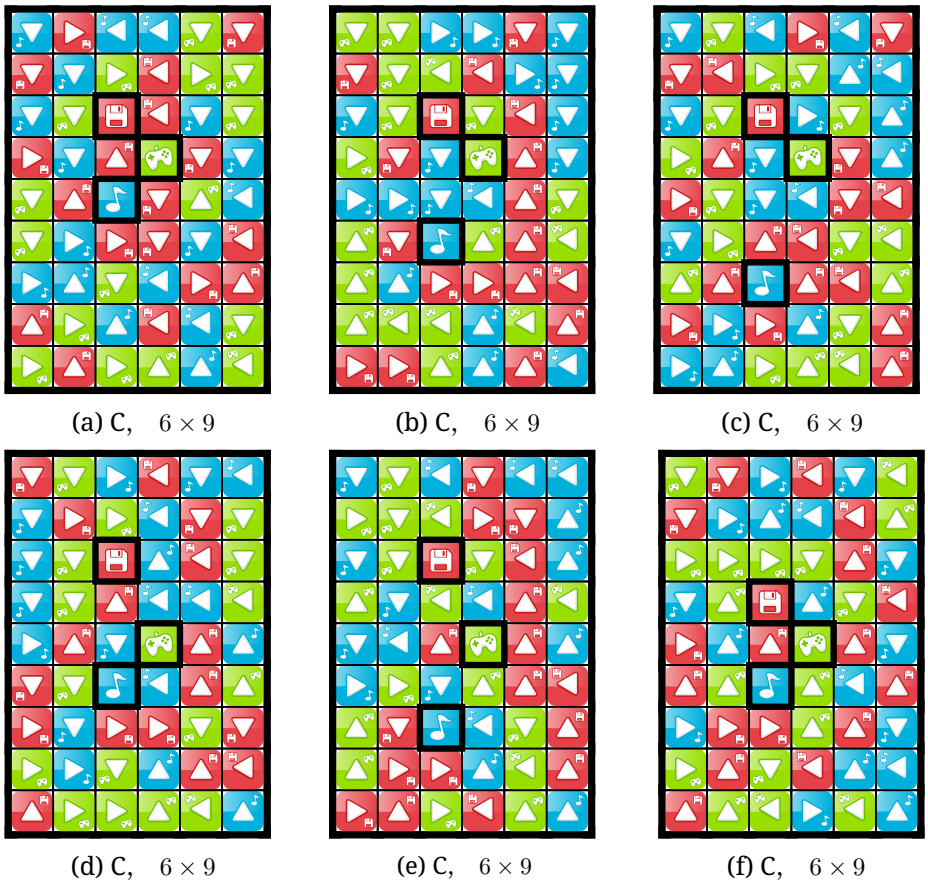
本文的大部分图示均使用 SVGTiler [https://github.com/edemaine/svgtiler]绘制。图标（与博弈相匹配）由 looneybits 制作，并以公共领域许可发布 [https:// opengameart.org/content/gui-buttons-vol1]。

参考资料

[Hay98] 林洋一。《记录介质、游戏程序代码加载方法及游戏机》。美国专利 5,718,632，授权日期：1998 年 2 月 17 日。[Mic] 微软研究院。Z3 定理证明器。https://github.com/Z3Prover/z3。[Ost15] Roger “atiaxi” Ostrander。《这个游戏不会自己加载》，2015年12月。https: // atiaxi.itch.io/this-game-is-not-going-to-load-itself。

自动求解器给出的完美布局

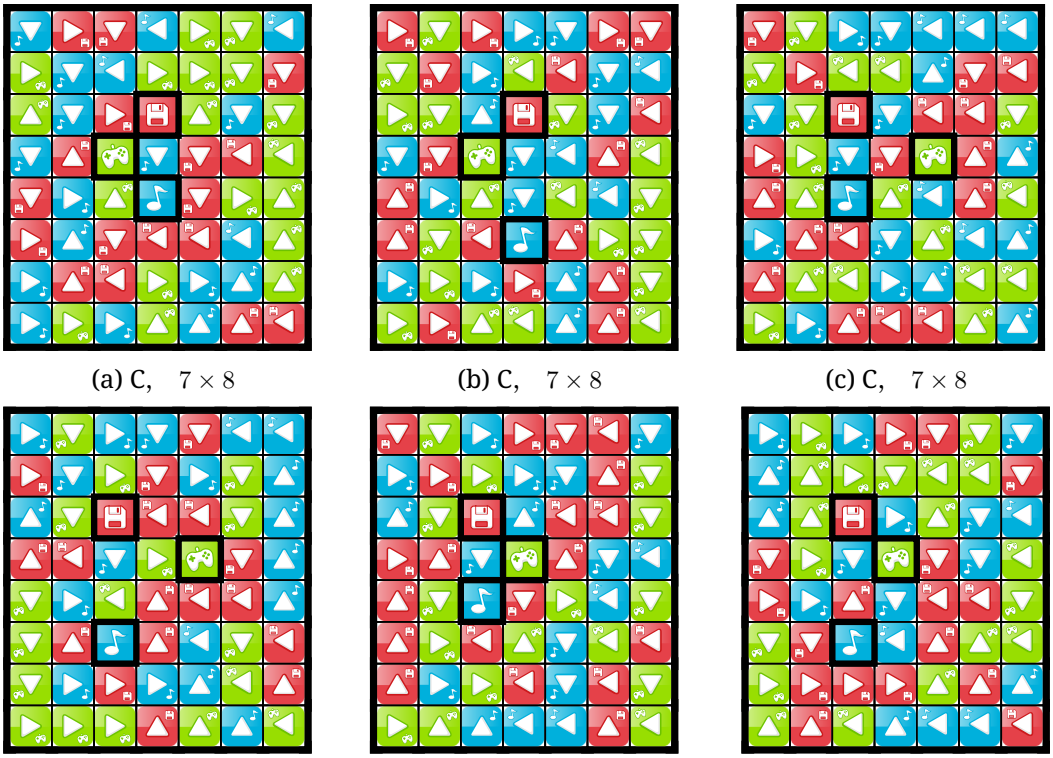
以下 13 幅图展示了自动求解器找到的所有情况。图 11、12 和 13 对应 C 型图案中的汇点。图 14、15 和 16 对应 I 型图案中的汇点。图 17、18 和 19 对应 J 型图案中的汇点。图 20 和 21 对应 L 型图案中的汇点。图 22 对应 Y 型图案中的汇点。最后，图 23 对应 / 型图案中的汇点。



(a) C, 6×9 (b) C, 6×9 (c) C, 6×9

(d) C, 6×9 (e) C, 6×9 (f) C, 6×9

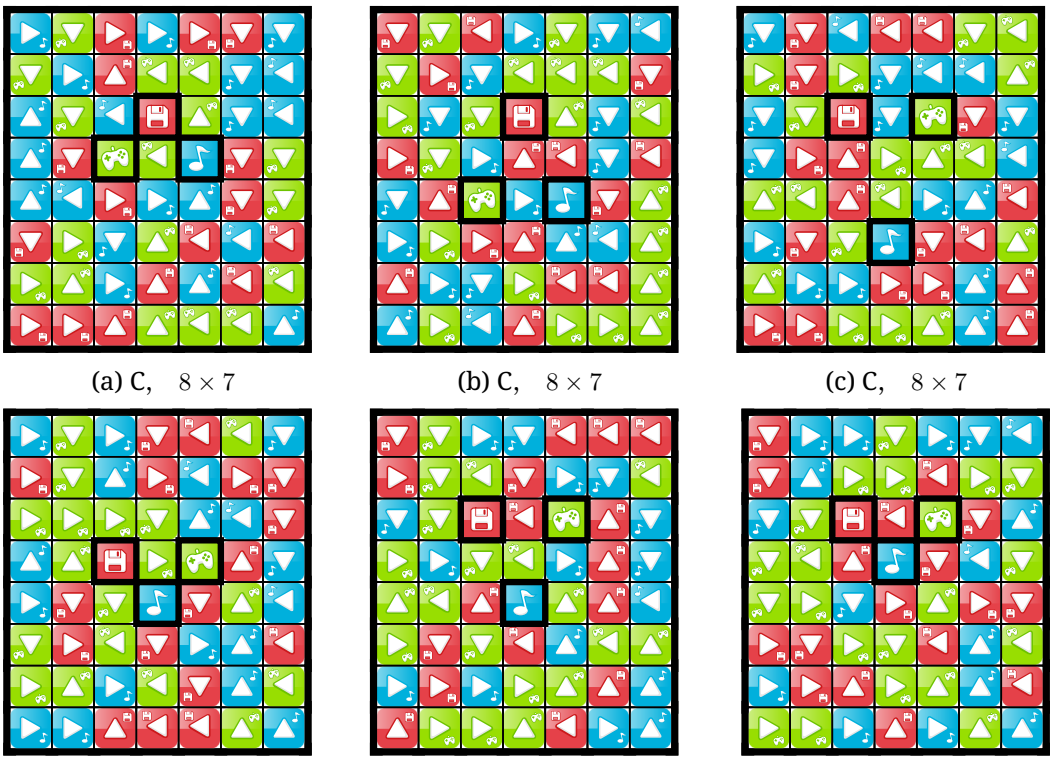
图1 1：自动求解器针对尺寸为 6×9 的 C 型图案中汇点的求解结果 .



(a) C, 7×8 (b) C, 7×8 (c) C, 7×8

(d) C, 7×8 (e) C, 7×8 (f) C, 7×8

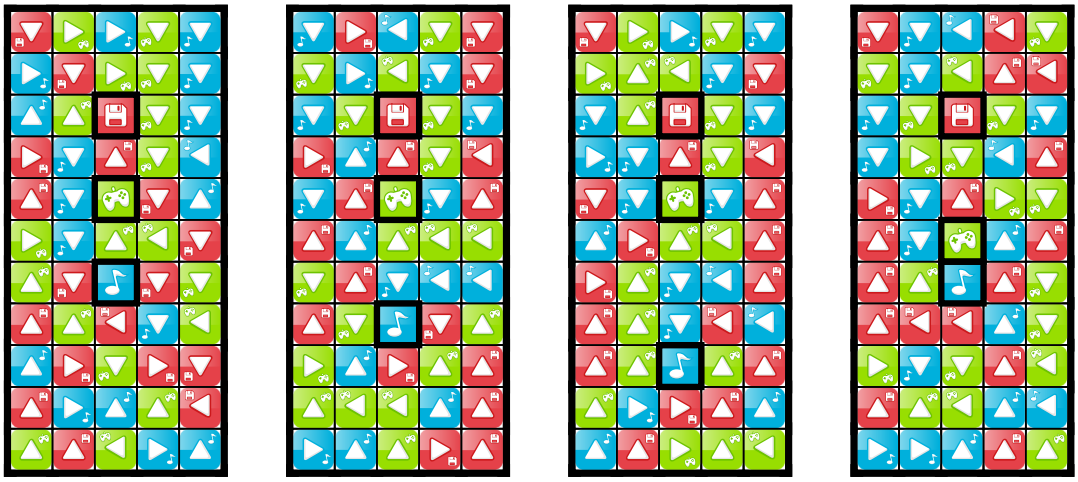
图12：自动求解器针对尺寸为 7×8 的 C 型图案中汇点的解。



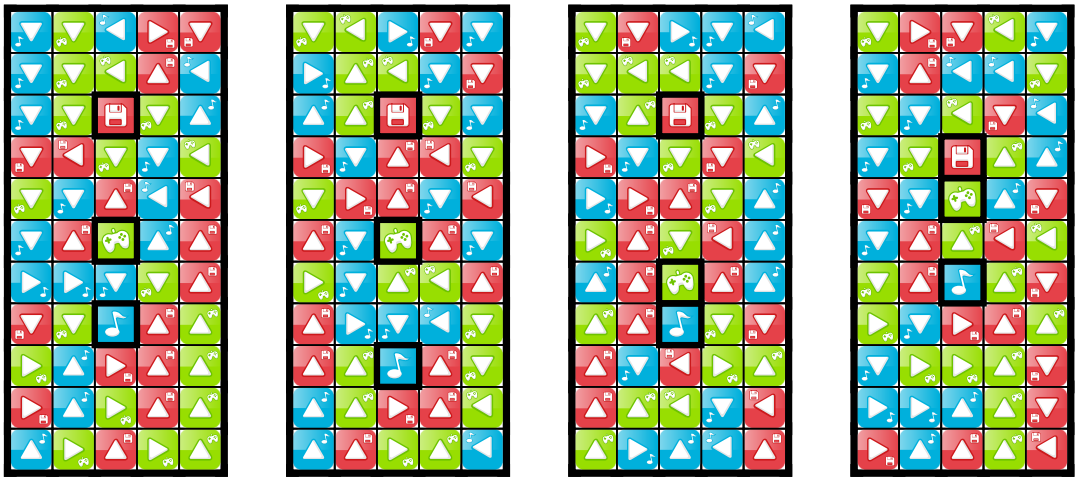
(a) C, 8×7 (b) C, 8×7 (c) C, 8×7

(d) C, 8×7 (e) C, 8×7 (f) C, 8×7

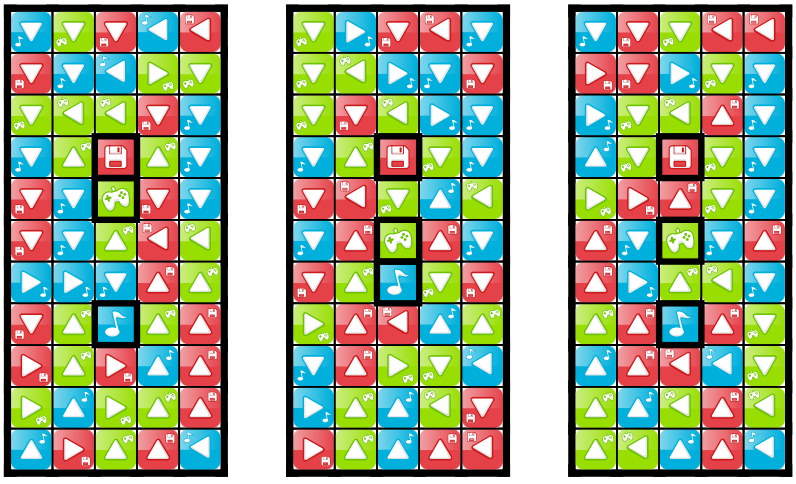
图1 3：尺寸为 $8 \times$ 的 C 型图案中汇点的自动求解器解 7.



(a) 我， 5×11 (b) 我， 5×11 (c) 我， 5×11 (d) 我， 5×11

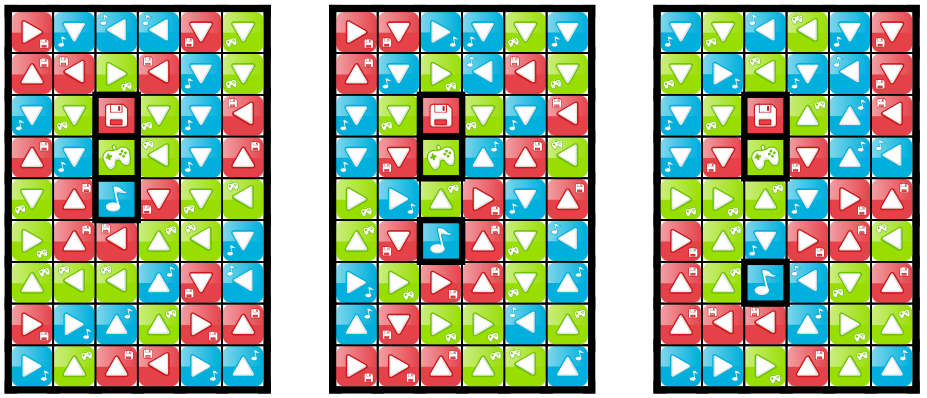


(e) 我， 5×11 (f) 我， 5×11 (g) 我， 5×11 (h) 我， 5×11



(i) 我， 5×11 (j) 我， 5×11 (k) 我， 5×11

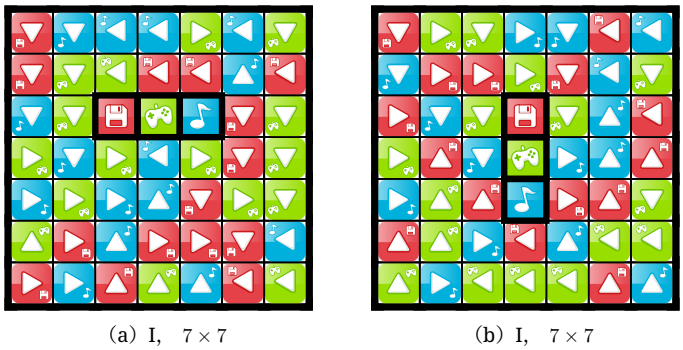
图1　　4：自动求解器针对尺寸为 5×11 的 I 型图案中汇点的解　　.



(a) 我， 6×9 (b) 我， 6×9 (c) 我， 6×9

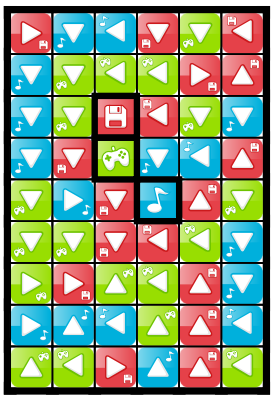
(d) 我， 6×9 (e) 我， 6×9 (f) 我， 6×9

图1 5：自动求解器对尺寸为 6×9 的 I 型图案中汇点的求解方案 .

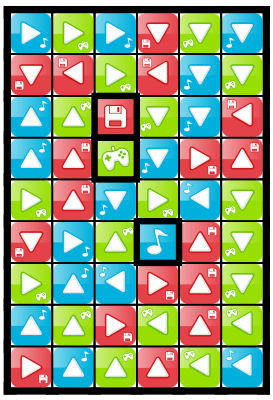


（a）I， 7×7 （b）I， 7×7

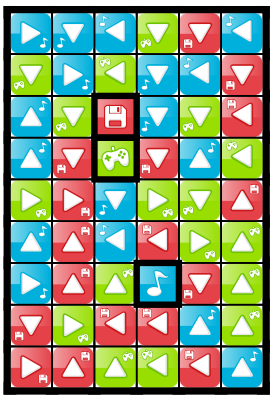
图16：自动求解器对尺寸为 7×7 的 I 型图案中汇点的求解方案。



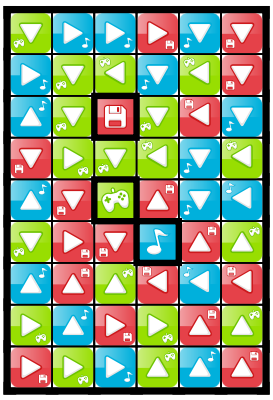
(a) J, 6×9



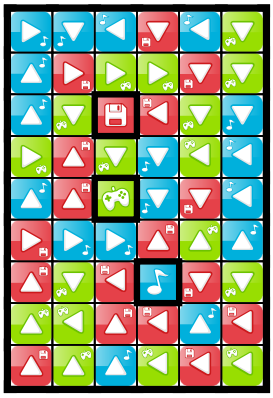
(b) J, 6×9



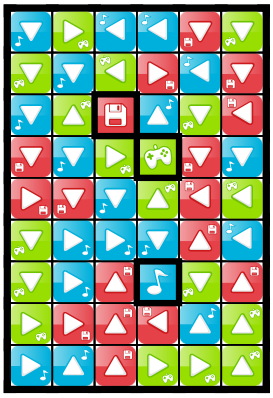
(c) J, 6×9



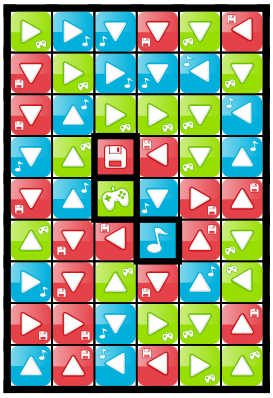
(d) J, 6×9



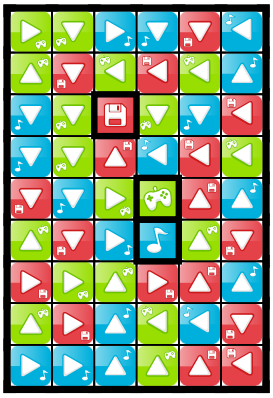
(e) J, 6×9



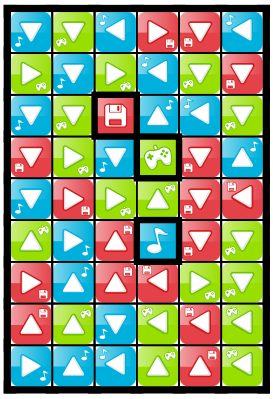
(f) J, 6×9



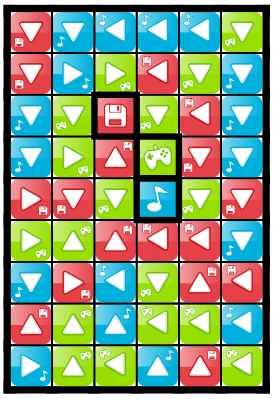
(g) J, 6×9



(h) J, 6×9

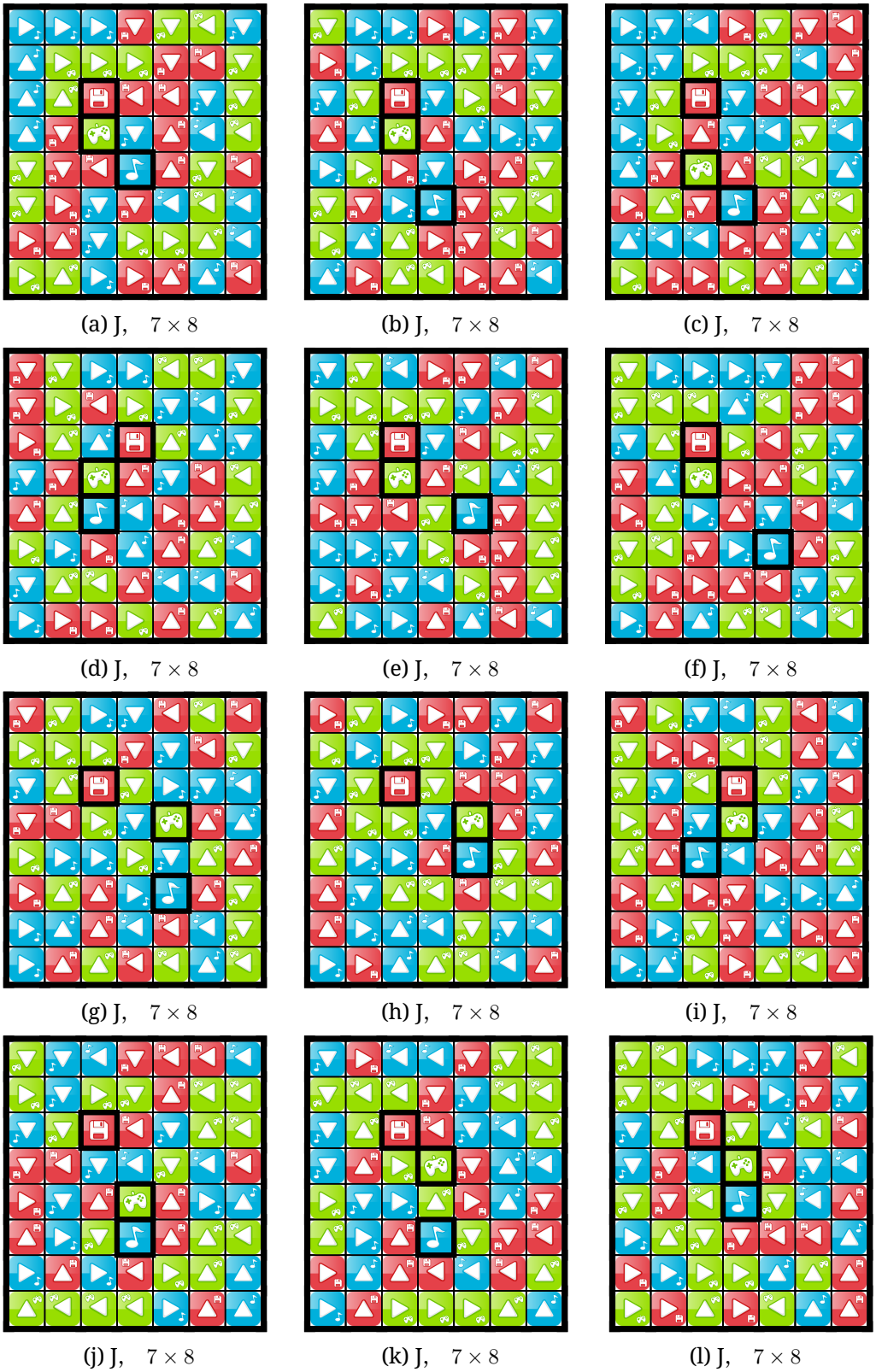


(i) J, 6×9



(j) J, 6×9

图17：自动求解器对尺寸为 6×9 的 J 型图案中汇点的求解结果。



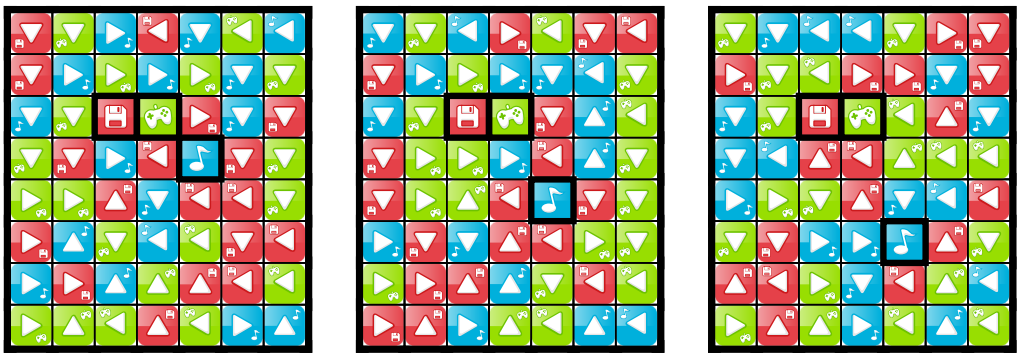
(a) J, 7×8 (b) J, 7×8 (c) J, 7×8

(d) J, 7×8 (e) J, 7×8 (f) J, 7×8

(g) J, 7×8 (h) J, 7×8 (i) J, 7×8

(j) J, 7×8 (k) J, 7×8 (l) J, 7×8

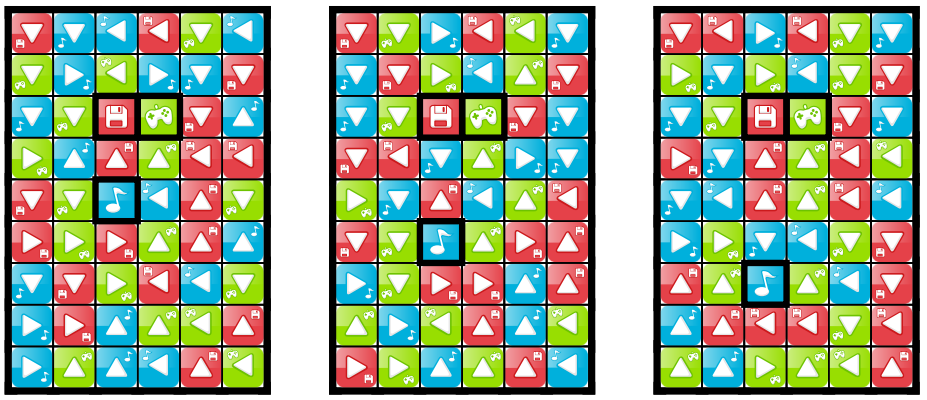
图18：自动求解器为尺寸为 7×8 的 J 型图案中的汇点找到的解。



(a) J, 8×7 (b) J, 8×7 (c) J, 8×7

(d) J, 8×7 (e) J, 8×7 (f) J, 8×7

图1 9：自动求解器对尺寸为 $8 \times$ 的 J 型图案中汇点的求解结果 7.

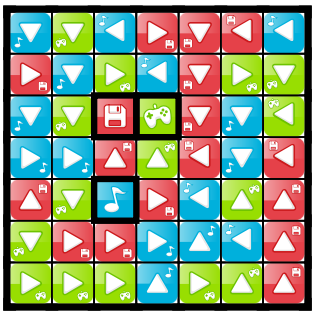


(a) L, 6×9 (b) L, 6×9 (c) L, 6×9

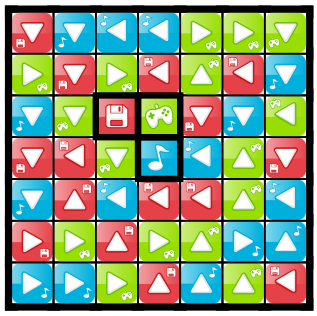
(d) L, 6×9 (e) L, 6×9 (f) L, 6×9

(g) L, 6×9 (h) L, 6×9 (i) L, 6×9

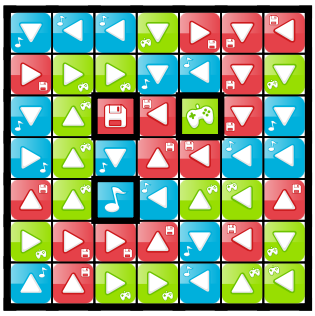
图 2 0：自动求解器对尺寸为 $6 \times$ 的 L 型图案中汇点的求解结果 9.



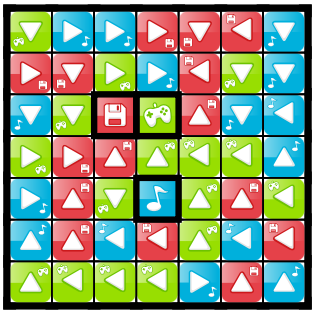
(a) L, 7×7



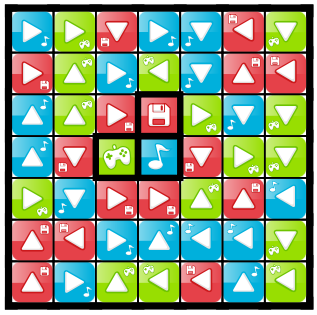
(b) L, 7×7



(c) L, 7×7

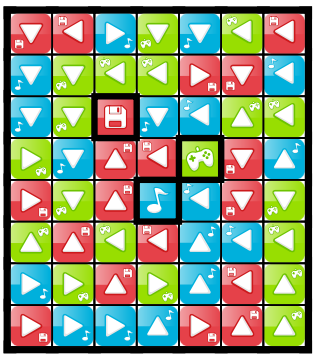


(d) L, 7×7

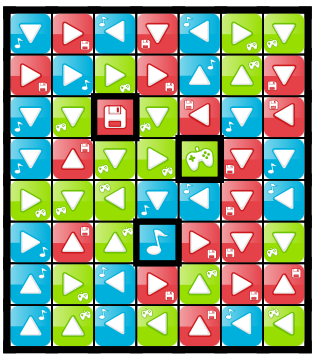


(e) L, 7×7

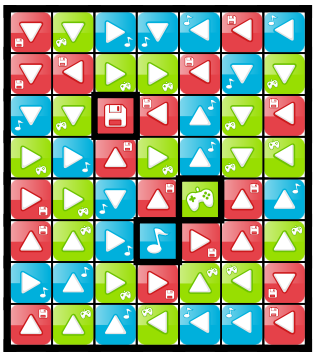
图 21：自动求解器针对尺寸为 7×7 的 L 型图案中的汇点给出的解。



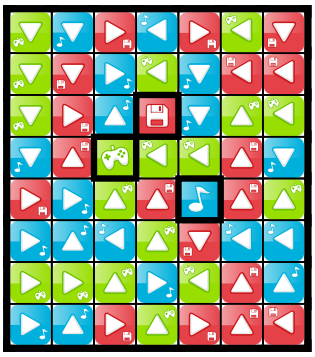
(a) Y, 7×8



(b) Y, 7×8

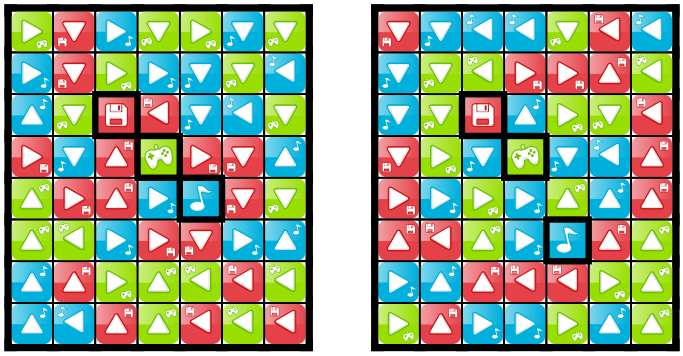


(c) Y, 7×8



(d) Y, 7×8

图 22：自动求解器针对尺寸为 7×8 的 Y 型图案中汇点给出的解。



(a) /， 7×8 (b) /， 7×8

图 23：自动求解器对尺寸为 7×8 的 / 模式中汇点的求解结果 .