# Turning Around and Around: Motion Planning through Thick and Thin Turnstiles

Aster Greenblatt* Oscar Hernandez† Robert A. Hearn‡ Yichao Hou§ Hiro Ito¶ Minwoo Kang‖
Aaron Williams** Andrew Winslow††

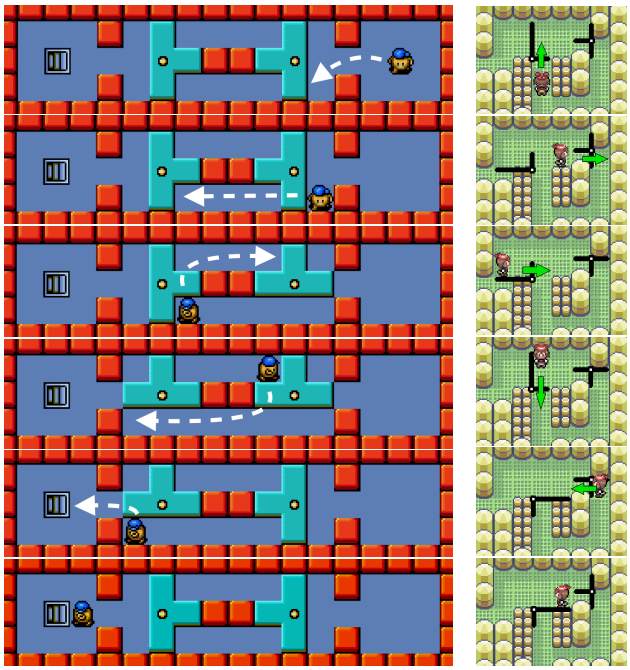
## Abstract

We examine the computational complexity of turnstile puzzles, which are grid-based tour puzzles with walls and turnstiles. A turnstile is a wall that can be rotated in 90° increments, either clockwise or counter-clockwise, around a central pivot when pushed by the player's token. In a 'thick' turnstile, the pivot and arms occupy cells of the grid, whereas in a 'thin' turnstile the pivot and arms occupy grid points and lines, respectively. We prove that reaching an exit is PSPACE-hard, even when restricted to just one of the following turnstiles types: , , , , , or . This establishes PSPACE-hardness for a dozen video games spanning several decades, including Kwirk (1989), Pokemon Ruby (2002), and Super Mario Odyssey (2017). Our hardness results are obtained by applying the motion planning framework in Jayson Lynch's PhD thesis *A framework for proving the computational intractability of motion planning problems* [MIT, 2020]. We also show that the decision problem can be solved in polynomial-time when restricted to or . We also formulate new open problems, and provide a survey of puzzles and games using turnstiles, which have also been called rotating doors, revolving gates, and spinning blocks.

## 1 Introduction

This article explores the familiar territory of grid-based motion planning, but with a few twists. In Section 1.1 we discuss the mechanism that we investigate, and in Section 1.2 we specify the tool that we will use.

*Division of Science, Mathematics, and Computing, Bard College at Simon's Rock, asterj.greenblatt@gmail.com
†Department of Mathematics & Statistics, University of Alaska Fairbanks, oihernandez@alaska.edu
‡bob@hearn.to
§yhou17@simons-rock.edu
¶School of Informatics and Engineering, University of Electro-Communications, itohiro@uec.ac.jp
‖Electrical Engineering and Computer Science, University of California, Berkeley, minwoo_kang@berkeley.edu
**Department of Computer Science, Williams College, aaron.williams@williams.edu
††andrewwinslow@gmail.com



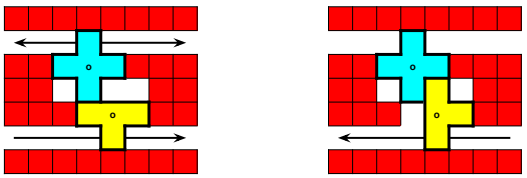
(a) Level 1. (b) Fortree City.

Figure 1: Solutions from (a) *Kwirk*, and (b) *Pokemon Ruby*.

### 1.1 Pushing, Pulling, Sliding, . . . and Rotating

There have been numerous studies on the computational complexity of grid-based puzzles and games that use pushing, pulling, or sliding as a core mechanism, with early contributions involving pushing by Fryers and Greene [11], Dor and Zwick [10], and Culberson [5]. More recently, in the *Games in Particular* section of Hearn and Demaine's *Games, Puzzles, and Computation*, 7 of the 10 puzzles use some form of pushing or sliding, including generalizations of Dad's Puzzle and Sokoban [15]. Research in this area is still active, with Barr et al. using a side-view and normal gravity [4], and Ani et al. [1] on block pulling being recent examples.

We consider a push-based rotation mechanism. Turnstiles have appeared in physical puzzles and video games for at least four decades, with two examples in Figure 1. Our main result is that single-player turnstile puzzles are PSPACE-hard, even when restricted to any of the following shapes: , , , , , or .



(a) State 1 with traversals. (b) State 2 with traversals.

Figure 2: One of the first gadgets that we designed has two 'tunnels' (black arrows) and two states. (a) The top tunnel is traversable in both directions since the turnstile can spin freely, and the bottom tunnel is traversable only from left-to-right since the turnstile can only spin counterclockwise. (b) The top tunnel is not traversable in either direction since the turnstile is blocked, and the bottom tunnel is traversable only from right-to-left since the turnstile can only spin clockwise. It is not possible to move between the two tunnels, and traversing the bottom tunnel always causes the state to change.

### 1.2 Motion Planning Framework

Jayson Lynch's PhD thesis [16] facilitates "proof by picture" hardness results. For example, Figure 2 shows one of the first constructions developed by the authors (see Section 5.2). Although it appears to do "something" non-trivial, we were unable to integrate it into a standard hardness reduction. Miraculously, the framework views this as a "non-crossing toggle lock (NTL)" gadget, and it suffices as the key component of a PSPACE-hardness proof. The framework[1] was developed and applied across the following publications.

- A simplified framework was introduced in *Computational Complexity of Motion Planning of a Robot through Simple Gadgets* at FUN 2018 by Demaine, Grosof, Lynch, and Rudoy [7]. Its focus is on two state gadgets and a single agent.
- A generalization appears in *Toward a General Complexity Theory of Motion Planning: Characterizing Which Gadgets Make Games Hard* by Demaine, Hendrickson, and Lynch at ITCS 2020 [9].
- The framework has been applied in *Trains, Games, and Complexity: 0/1/2-Player Motion Planning through Input/Output Gadgets* by Ani, Demaine, Hendrickson, and Lynch [3], and also in [2].

As indicated by the third item, the general framework is quite broad. In many ways, it promises to be an agent-based analog to the well-known constraint logic framework developed by Hearn and Demaine [14, 15]. In this paper, we are only concerned with 1-player puzzles, so the simplified setting of [7] still holds particular value.

[1] Appendix A includes an auxiliary series of images that shows how Figure 1a can be modeled using the framework.

### 1.3 Outline

In Section 2, we discuss rotation mechanisms, with a focus on turnstile puzzles. In Sections 3–4, we introduce concepts from the motion planning framework, and show how they can be applied to turnstile puzzles. Section 5 concludes with a summary, and open problems. Throughout the paper, we include additional information with an eye towards facilitating new research.

## 2 Rotation Mechanisms

Readers are likely familiar with several pushing mechanisms found in the literature, many of which can be categorized using the `Push[Push]-1/k/*-[X]` classification, as discussed in Demaine, Demaine, Hoffmann, and O'Rourke [6] (also see [8]). Some of these variations have been inspired by video games, with *Sokoban* (1981) and *Pengo* (1982) being two prominent examples. In Sokoban, the player controls a warehouse worker who can push a single box one grid cell at a time, and the worker moves with the box. In Pengo, the player controls a penguin who can push a single ice block, but in this case, the block slides along the ice as far as possible, and the penguin remains stationary.

Similarly, there are many rotation mechanisms found in video games, but fewer of them have been studied in the literature. We discuss five such mechanisms in Section 2.1. Then in Section 2.2 we define our decision problem, and prove that two special cases can be solved efficiently. Section 2.3 provides a survey of games and puzzles that have used the turnstile mechanism.

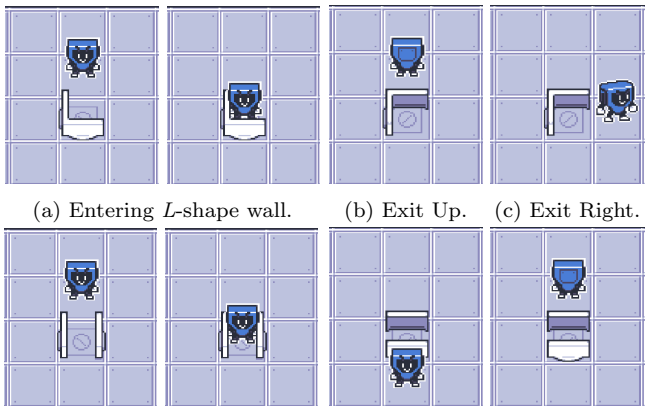
### 2.1 Specific Mechanisms

Here we discuss five different rotation mechanisms found in video games. The first mechanism was previously studied using the motion planning framework [7]. To the best of our knowledge, the remaining mechanisms have not previously been considered.

#### 2.1.1 $k$-Spinners

*The Legend of Zelda: Oracle of Seasons* and *Oracle of Ages* were released for Nintendo's Game Boy Color in 2001. Both games include a mechanism that is referred to as a 4-*spinner* in [7]. The mechanism can be embedded in a 3-by-3 grid of cells, with a pivot in the center, walls in each corner, and open chambers in the remaining four cells. The player interacts with the mechanism by entering one the chambers. Once inside, the 4-spinner turns 90°, and then stops, so that the player must exit the chamber. The direction of the turn depends on its state. In its blue state, the spinner turns counterclockwise, while in its red state, it turns clockwise. After the spinner has rotated, it changes state.

Figure 3: 4-Spinner. Link enters the chamber on the left. The spinner is blue, so it rotates 90°counterclockwise, and Link must exit downward. Once Link exits, the spinner turns red, indicating that its next rotation is 90°clockwise.



(a) Entering $L$-shape wall. (b) Exit Up. (c) Exit Right.

(d) Entering =-shape wall. (e) Exit Down. (f) Exit Up.

Figure 4: Rotating walls. (a) Circuit Dude enters an $L$-shape wall from an open side, then (b)–(c) can exit through either open side, and upon exiting, the walls rotate 90°clockwise in-place. (d)–(f) The =-shape wall behaves similarly. Note: Rotating walls appeared earlier in Bobby Carrot.

(a) Entering swivel (b) Exit ↑ (c) Exit → (d) Exit ↓ (e) Exit ←

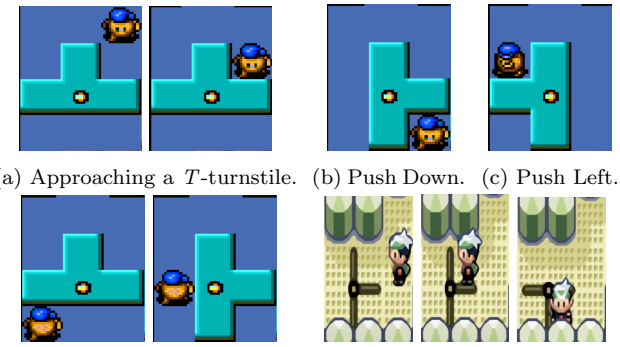
Figure 5: Swivel Door. (a) Melinda enters the swivel through an open side, then (b)–(c) exiting through an open side leaves the swivel unchanged, or (d)–(e) exiting through a wall causes the swivel to rotate 90°so that side opens.

Figure 3 illustrates a 4-spinner from the Moonlit Grotto in the Oracle of Ages. A *k-spinner* generalizes a 4-spinner by having $k$ chambers.

**Theorem 1 ([7])** *Motion planning is PSPACE-complete when restricted to $k$-spinners, for all $k \geq 4$.*

### 2.1.2 Rotating Walls

*Bobby Carrots* is a series of early mobile puzzle games. The first game was implemented in Java (J2ME) and was available for various handsets in 2004, including Nokia phones running Symbian. The most recent game,



(a) Approaching a $T$-turnstile. (b) Push Down. (c) Push Left.

(d) Rotate with no extra move. (e) Turning a thin $T$-turnstile.

Figure 6: Turnstiles. (a) Kwirk approaches an internal corner of a thick turnstile, then (b)–(c) turns it clockwise or counterclockwise with a single push, with the trailing arm moving Kwirk one extra cell. (d) Pushing the turnstile with a trailing arm does not cause the extra move. (f) Thin turnstiles behave similarly, but without the extra move.

*Bobby Carrot 5: Forever*, was released on the Nintendo Wii and iPhone[2]) and elsewhere in 2011. The series includes a both $L$-shaped and =-shaped *rotating walls*. These mechanisms occupy one grid cell, and a pair of *thin walls* occupy two of the four gridlines inside its perimeter. The player can enter or exit the cell through either open side, and the cell rotates 90°once the player exits it. The same mechanism has been used in other games, including *Circuit Dude*, which launched on the Arduboy in 2016, followed by iOS / Android / Steam / Switch. Figure 4 illustrates the mechanism.

### 2.1.3 Swivel Doors

*Chip's Challenge* was released for the Atari Lynx in 1989, and was popularized in *Microsoft Entertainment Pack 4* for Windows 3.1 in 1992. Its sequel, *Chip's Challenge 2* (CC2), was developed in 1999, but was not available to the public until its 2015 release on Steam. The sequel added new mechanisms, including the *swivel door*. The swivel door is visually identical to $L$-shaped rotating walls, as they both involve two consecutive thin walls around a single grid cell. Furthermore, the player enters the mechanisms in the same way (i.e. through one of the two openings). However, Chip and Melinda can exit a swivel door in any of the four cardinal directions. If they exit through either of the two openings, then the swivel door does not change. Otherwise, if they exit through one of the walls, then the cell rotates 90°. More specifically, the rotation is chosen so that the exited side of the mechanism becomes open. In other words, the mechanism behaves as if the walls are slid out of the way upon exit. Figure 5 provides an illustration.

[2]Due to Apple's iconoclastic update policies, this game, and so many other classics, are no longer available on their platforms.

### 2.1.4 Thick Turnstiles

*Puzzle Boy* (🇯🇵 1989) was developed for the Game Boy by Atlus, and localized as *Kwirk* (🇺🇸 1990). The game uses *thick turnstiles*, which have a central pivot that occupies a single cell, and arms that occupy at most one cell radiating outward in each direction, producing the , , , , and (and their rotations). Each mechanism can rotate 90°around its pivot, either clockwise or counterclockwise, when the player pushes one of its arms. If a turnstile can rotate, then the player will also move in the direction of the push. Furthermore, if an arm is following behind the player, and would occupy their cell after the rotation, then the player is pushed one extra cell by the trailing arm. A turnstile cannot be rotated if a wall, or another turnstile, occupies a cell that the arm would need to rotate through. Figure 6 shows a thick $T$-shaped turnstile.

### 2.1.5 Thin Turnstiles

*Lady Bug* is a maze chase game released in arcades in 1981 by Universal, and later ported to home consoles. The game features *thin turnstiles*, which have a central pivot that occupies a grid point, and arms that radiate out along single grid lines, in the shape (or its rotation ). In *Pokemon Ruby and Sapphire*, the mechanism was generalized to include additional shapes. The mechanism behaves similarly to thick turnstiles. However, the player is never pushed an extra cell by a trailing arm, since the arms occupy grid lines instead of cells. Figure 6 shows a thin $T$-shaped turnstile.

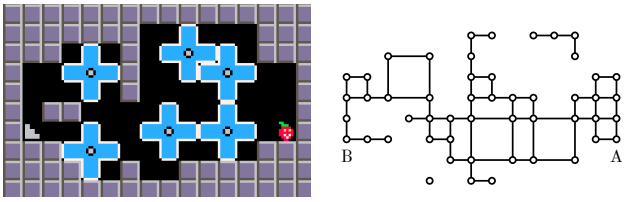
When considering multiple thin turnstiles, it is natural to wonder if *overlapping arms* are allowed. In other words, can the arms of two turnstiles occupy the same grid line?[3] This question is not answered by the Pokemon games, since the turnstiles are placed in ways that prevent this possibility. We take the position that overlapping arms are never allowed. Thus, a thin turnstile cannot be given a particular rotation if that rotation would result in an overlap. For the same reason, we do not mix thin and thick turnstiles in the same puzzles.

## 2.2 Turnstile Decision Problem

The Turnstile decision problem takes as input a *level* $L$, which is a grid in which the cells, points, and lines form walls and well-formed turnstiles (either thick or thin), with an *initial position* and *exit position*. If the player can move from the initial position to the exit by a sequence of moves, then the output is yes. The proof of Proposition 2 appears in Appendix B.

**Proposition 2** Turnstile *is in PSPACE.*

[3]This is not a concern with rotating walls or swivel doors, since their thin obstacles are internal to a specific grid cell.




(a) A level with turnstiles. (b) Graph corresponding to (a).

Figure 7: When restricted to , the Turnstile decision problem can be solved using undirected s-t connectivity.

We will prove that Turnstile is PSPACE-hard, and hence PSPACE-complete. In fact, we'll see that Figure 2 guarantees this. Therefore, we'll focus on determining whether Turnstile remains PSPACE-hard (and PSPACE-complete) when the turnstile types are restricted. Let Turnstile$_S$ denote the restriction of Turnstile to levels that only use turnstiles from the set $S$. For example, we now prove that the decision problem can be solved efficiently when restricted to or . Figure 7 illustrates the proof of Proposition 3.

**Proposition 3** Turnstile$_S$ *for* $S = \{$ $\}$ *is decidable in polynomial-time.*

**Proof.** We'll show that the decision problem can be transformed into an undirected s-t connectivity problem in polynomial-time.

First observe that the turnstile is unique amongst the thick turnstiles in that it has only one orientation. In other words, rotating a turnstile does not change which cells are occupied. Therefore, when $S = \{$ $\}$, we can partition the turnstiles into two classes: those that can always rotate, and those that can never rotate. Furthermore, it is easy to identify those in the latter category, since they are precisely those in which one of the four cells that are diagonally-adjacent to its pivot is occupied by a wall or the arm of another turnstile.

We create a graph associated with the level as follows. Begin with a grid graph with points associated with cells of the grid that are either empty, along with the initial the exit positions. Now we consider each turnstile that can rotate. Given such a turnstile whose pivot occupies position $(i, j)$ of the grid, we add edges of the form $(a, b)$ where $a \in \{(i+1, j+1), (i-1, j-1)\}$ and $b \in \{(i-1, j+1), (i+1, j-1)\}$. In other words, we add a square that connects the empty cells that are diagonally-adjacent to the pivot. The creation of this graph can be done in polynomial-time.

Finally, we check if the start and exit positions are connected, which can be done in deterministic log-space. Hence, the overall algorithm takes polynomial-time. □

The following proposition can be proven similarly.

**Proposition 4** Turnstile$_S$ *for* $S = \{$ $\}$ *is decidable in deterministic log-space.*

### 2.3 History

Tables 1–2 list puzzles and games that use thick or thin turnstiles, either as a primary or secondary mechanism, respectively. Our results apply to all of the entries, except for *Lady Bug* and *Drelbs*, which are action games rather than puzzle games, and the physical puzzle *Turnstiles* which is a multi-player game.

## 3 Gadgets in General

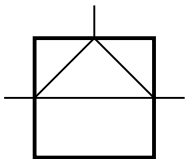
We begin our presentation of the motion planning framework by considering gadgets and their properties. Our presentation focuses on sets and functions, and we use the graphical styles from both the initial framework [7] and the generalized framework [9]. We begin by introducing the simplest gadgets: hallways and 1-toggles.

### 3.1 Branching Hallways

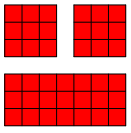
A *branching hallway* simply connects three locations. Since there is no gravity in TURNSTILE, it is easy to implement these gadgets.

**Proposition 5** *Branching hallways can be implemented in* TURNSTILE *without turnstiles.*

**Proof.** The branching hallway gadget from [7] is shown below on the left, with an implementation on the right.



Branching hallway.



Implementation in TURNSTILE.

□

### 3.2 1-Toggle

A *1-toggle* connects two locations, A and B, via a dynamic path known as a *tunnel*. In state 1, it possible to travel from A to B, while in state 2, it is possible to travel from B to A. Furthermore, the gadget toggles its state whenever one of these *traversals* in completed. The 1-toggle is illustrated in several ways in Figure 8.

- Figure 8a is the graphical style from [7], in which a single diagram represents two states. In this type of diagram, a directed arrow represents a *toggle*, and traversing it causes its direction to switch.
- Figure 8b is the graphical style from [9], in which both states are shown. One can think of the rounded rectangles as being drawn on top of each other, since the left side of both rectangles refer to the same location, as do the right sides of both

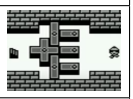
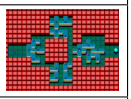
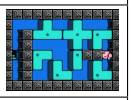
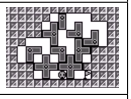
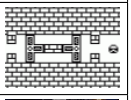
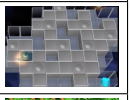
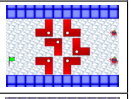
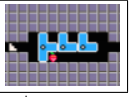
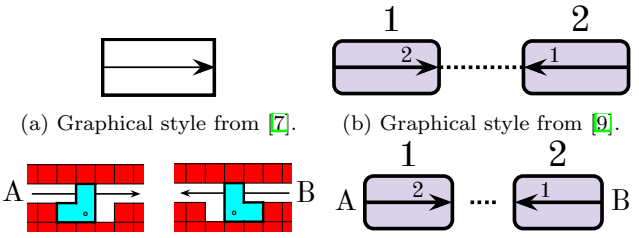
| Release | Title | Developer | Platform(s) | Screenshot |
|---|---|---|---|---|
| 1989 JP<br>1990 US | Puzzle Boy<br>Kwirk | Atlus | Game Boy |  |
| 1989 JP | Maze-kun<br>(Mr. Maze) | Telenet Japan | NEC PC-8800, MSX 2 |  |
| 1990 JP | Puzzle Boys | Atlus | Famicom Disk System |  |
| 1991 JP | Puzzle Boy | Telenet Japan | NEC PC Engine |  |
| 1991 JP<br>1992 US | Puzzle Boy 2<br>Amazing Tater | Atlus | Game Boy |  |
| 1998 | Kwirk† | Ludvig Strigeus | TI-89 / TI-90 Calculator |  |
| 2003 | Shin Megami Tensei: Nocture | Atlus | PlayStation 2 |  |
| 2003 | Devil Children: Puzzle de Call! | Atlus | Game Boy Advance |  |
| 2008 | Puzzle Boy Flash† | Blawars | Web browser |  |
| 2013 | Kwirk Free†<br>Kwirk 2† | Galiksoft | Android / Amazon App Store |  |
| 2020 | Turner† | Pico Beast | Pico-8 |  |

Table 1: Video games using thick turnstiles. †clone

| Release | Title | Developer | Platform(s) | Screenshot |
|---|---|---|---|---|
| 1981 | Lady Bug | Universal | Arcade Intellivision Colecovision |  |
| 1983 | Lady Tut | Programe | Apple ][ Commodore 64 |  |
| 1983 | Drelbs | Synapse Software | Atari 8-bit Apple ][ Commodore 64 |  |
| 2002 JP<br>2003 US | Pokemon Ruby and Sapphire | Game Freak | Game Boy Advance |  |
| 2004 JP<br>2005 US | Pokemon Emerald | Game Freak | Game Boy Advance |  |
| 2012 | Turnstile | ThinkFun | Physical puzzle |  |
| 2014 | Pokemon Omega Ruby and Alpha Sapphire | Game Freak | Nintendo 3DS |  |
| 2017 | Super Mario Odyssey | Nintendo | Nintendo Switch |  |

Table 2: Puzzles and games using thin turnstiles.




(a) Graphical style from [7]. (b) Graphical style from [9].

(c) Implementation in Kwirk. Left: State 1. Right: State 2. (d) Graphical style from [9] with location labels.

Figure 8: A 1-Toggle gadget in several graphical styles, and implemented with one . In state 1, the gadget's only traversal is $A \to B$ (left to right). In state 2, the gadget's only traversal is $B \to A$ (right to left). The state changes if and only if a traversal occurs.

rectangles. In this style, the arrows are labeled with the state number that the gadget changes to after that particular traversal. The dotted line illustrates that the tunnel is reversible (i.e. once a traversal is complete, it can be done in reverse, and the gadget will return to its previous state).

- Figure 8c illustrates how a 1-toggle can be implemented with a single turnstile. The two states can again be interpreted as being drawn on top of each other, and in this case we have explicitly labeled the left location as A, and the right location as B.
- Figure 8d provides our slight modification of the graphical style from [9], which is the result of student feedback from the second-last author's course on the theory of computation. Specifically, we add explicit location labels, and we shorten the dotted line between the states. The first modification makes it easier to discuss the gadget, while the second is avoid a common misunderstanding. When an A to B traversal is conducted in state 1, the gadget toggles to state 2, and the agent ends in location B. The agent does not magically transport back to location A, as some have interpreted the dotted line to indicate.

### 3.3 Definition of a Gadget

Now we formally define a gadget. A *gadget* is a triple $g = (n, L, T)$ whose components are defined below, where $[x]$ denotes $\{1, 2, \ldots, x\}$.

- $n$ is the finite number of *states*.
- $L$ is a finite ground set of $m$ *locations*.
- $T \subseteq [n] \times L \times [n] \times L$ is a set of *traversals*. An individual *traversal* is a 4-tuple $(s_1, \ell_1, s_2, \ell_2) \in T$, where $s_1$ is the *current state*, $\ell_1$ is the *entry location*, $s_2$ is *next state*, and $\ell_2$ is the *exit location*.

The reader may wonder why the definition includes a set of locations, rather than the number of locations. The reason is that sets allow for gadgets to share, or not share, locations. For example, we could have two instances of a 1-toggle, the first with states $L_1 = \{A, B\}$, and the second with states $L_2 = \{B, C\}$.

In later sections, we will be considering systems of gadgets, and how they are connected to each other. In this context, planarity is a consideration, and we need to know the cyclic order of the locations around a gadget. This leads to the following definition.

A *planar gadget* is a pair $p = (g, \pi)$ where

- $g = (n, L, T)$ is a gadget.
- $\pi$ is a cyclic order of the gadget's locations. (In other words, $\pi$ is a necklace over $L$.)

## 4 Gadget Types

At the end of Section 3, we formally defined gadgets. Now we define several specific types of (planar) gadgets, and show how to implement them with turnstiles.

As a simple example, we can formally define a *1-toggle* as $g_{1T} = (n, L, T)$, with $n = 2$ states, two locations $L = \{A, B\}$, and two transitions

$$T = \{(1, A, 2, B), (2, B, 1, A)\}.$$

There are no planar variations of this gadget, since at least four locations are required to have multiple cyclic orders of the location set.

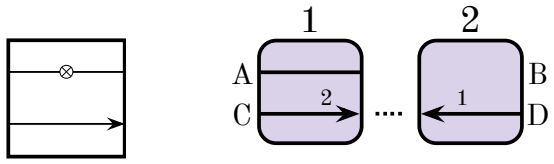
When implementing a gadget type, it is important to think of the type as a template, rather than a single gadget. More specifically, an *implementation* of a gadget must have the property that separate copies are *independent* in the sense that they do not share any state. For example, if a video game has a button that acts globally (i.e. opens or toggles all doors), then it cannot be used in an implementation. This is will not be an issue in Turnstile since each turnstile has its own orientation. For example, if we make two copies of the 1-toggle implementation in Figure 8c, then they will act independently of each other.

### 4.1 Noncrossing Toggle Lock (NTL)

A *toggle-lock (TL)* is a gadget $g_{TL} = (n, L, T)$ with $n = 2$ states, four locations $L = \{A, B, C, D\}$, and the following set of four transitions $T$:

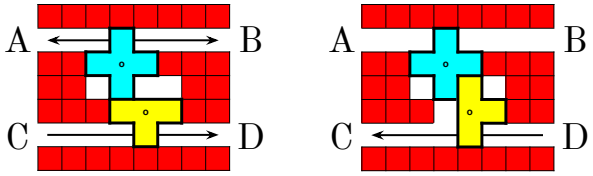
$$\{(1, A, 1, B), (1, B, 1, A), (1, C, 2, D), (2, D, 1, C\}. \quad (1)$$

In other words, a toggle lock has two tunnels, A-B and C-D. The first tunnel is traversable in both directions in state 1, and is not traversable in state 2, while the second tunnel is always traversable in one-direction.




(a) NTL drawing from [7]. (b) NTL drawing from [9].

Figure 9: Noncrossing toggle lock (NTL) gadget.




(a) State 1 of NTL. (b) State 2 of NTL.

Figure 10: Implementing a noncrossing toggle-lock.

Traversing the second tunnel toggles the gadget's state, which changes the traversability of both tunnels. The first tunnel will be referred to as a *lock* in [7].

We are particularly interested in the one of the planar toggle-locks.

- A *noncrossing toggle-lock (NTL)* is a pair $p = (g_{TL}, \pi)$ with $\pi_n = ABDC$.

The $\pi_n$ order implies that the two tunnels do not cross each other. The NTL gadget is illustrated in Figure 9.

The first non-trivial gadget that the authors constructed happened to be a noncrossing toggle-lock, and Figure 2 is reproduced in Figure 10. Fortuitously, the simplified motion planning framework proves that the ability to implement NTL gadgets (and branching hallways) is sufficient for establishing PSPACE-hardness. One detail that we mention

**Theorem 6** Turnstile *is PSPACE-complete.*

**Proof.** The decision problem is in PSPACE by Proposition 2, and branching hallway can be implemented in Turnstile by Proposition 5. Figure 2 implements a noncrossing toggle-lock. Therefore, the result follows from Corollary 5.2 of [7]. □

### 4.2 Crossing 2-Toggle (C2T)

A *2-toggle (2T)* is a gadget $g_{2T} = (n, L, T)$ with $n = 2$ states, four locations $L = \{A, B, C, D\}$, and the following set of four transitions:

$$T = \{(1, A, 2, B), (1, C, 2, D), (2, B, 1, A), (2, D, 1, C\}.$$

In other words, a 2-toggle has two tunnels, A-B and C-D, which are always one-directional, and traversing either either tunnel toggles the direction of both tunnels.

We are particularly interested in the one of the planar 2-toggles.

- A *crossing 2-toggle (C2T)* is a pair $p = (g_{2T}, \pi_c)$ with $\pi_c = ADBC$.

The order $\pi_c$ implies that the two cross each other. The C2T gadget is illustrated in Figure 11.

Although we did not have a name for it at the time, the second non-trivial gadget that the authors constructed was a crossing 2-toggle using a pair of turnstiles. Later we added thin turnstiles to our investigation, and it was possible to mimic the construction with thick turnstiles using a pair of turnstiles. In both cases, the turnstiles are positioned so that they each have two possible orientations, and a single enclosed cell between them. During a traversal, the player rotates one of the turnstiles to enter the enclosed cell, then must either retreat, or turn the other turnstile to continue along the direction that the entered from. Figures 12 and 13 illustrate the two constructions. Amazingly, the simplified motion planning framework again proves that these images are central to establishing PSPACE-hardness.

**Theorem 7** Turnstile *is PSPACE-complete when restricted to thick T-shaped or thin T-shaped turnstiles. That is,* Turnstile$_S$ *is PSPACE-complete when* $S = \{$ $\}$ *or* $S = \{$ $\}$.

**Proof.** The decision problem is in PSPACE by Proposition 2, and branching hallway can be implemented in Turnstile by Proposition 5. Figures 12 and 13 implement a crossing 2-toggle $S = \{$ $\}$ and $S = \{$ $\}$, respectively. Therefore, the result follows from Corollary 5.2 of [7]. □

### 4.3 Locking 2-Toggles (L2T)

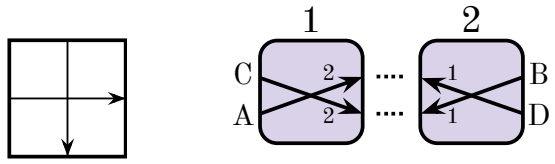
A *locking 2-toggle (L2T)* is a gadget $g_{L2T} = (n, L, T)$ with $n = 3$ states, four locations $L = \{A, B, C, D\}$, and the following set of four transitions:

$$T = \{(1, B, 3, A), (2, D, 3, C), (3, A, 1, B), (3, C, 2, D\}.$$

In other words, a locking 2-toggle has two tunnels, A-B and C-D, which are always traversable in at most one direction. In state 3, both tunnels are traversable, and traversing the first tunnel changes the gadget to state 1, while traversing the second tunnel changes the gadget to state 2. In state 1, only the first tunnel is traversable (in the opposite direction), while only the second tunnel is traversable (in the opposite direction) in state 2.

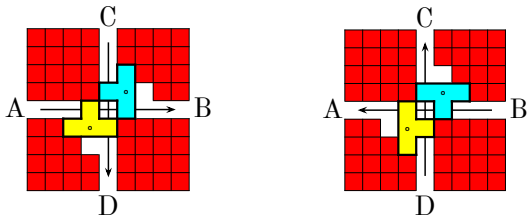
We consider two planar locking 2-toggles.

- A *parallel locking 2-toggle (PL2T)* is a pair $p = (g_{L2T}, \pi_p)$ with $\pi_p = ACDB$.



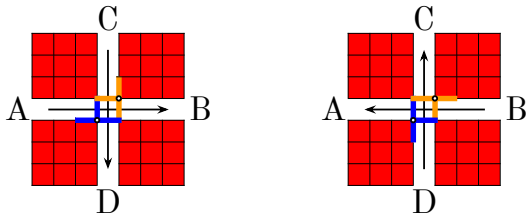

(a) C2T drawing from [7]. (b) C2T drawing from [9].

Figure 11: Crossing 2-toggle (C2T) gadget drawing..




(a) State 1 of C2T (b) State 2 of C2T

Figure 12: Crossing 2-toggle using .




(a) State 1 of C2T (b) State 2 of C2T

Figure 13: Crossing 2-toggle using .

- An *antiparallel locking 2-toggle (AL2T)* is a pair $p = (g_{L2T}, \pi_a)$ with $\pi_a = ADCB$.

Both orders implies that the two tunnels do cross each. In the first case, the traversal directions in state 3 are the same, while in the second case, the traversal directions in state 3 are opposite. The PL2T and AL2T gadgets are illustrated in Figure 14 and 16, respectively. Since the L2T gadget has three states, the simplified graphical style in [7] cannot be used.

We were able to implement the PL2T gadget in one way, and the AL2T gadget in three ways. In each case, the general idea is to place two 1-toggles together in such a way that they interfere with the other 1-toggle in one or the two states. Figures 15 illustrates our construction of PL2T, and Figures 17, 18, and 19 illustrate our constructions of AL2T. The generalized motion planning framework again proves that these images are central to establishing PSPACE-hardness.

**Theorem 8** TURNSTILE *is PSPACE-complete when restricted to thick or thin L-shaped or* 1*-shaped turnstiles. That is,* TURNSTILE$_S$ *is PSPACE-complete when* $S = \{$ $\}$ *or* $S = \{$ $\}$ *or* $S = \{$ $\}$ *or* $S = \{$ $\}$.

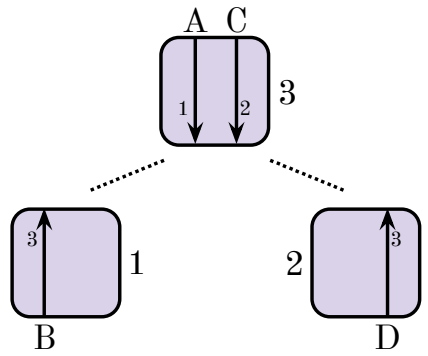
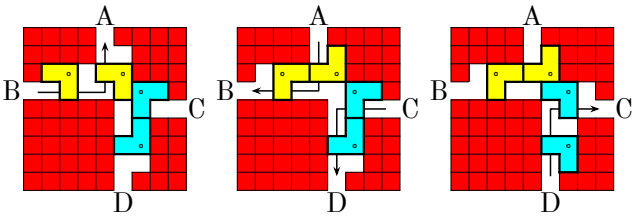



Figure 14: Parallel locking 2-toggle (PL2T) gadget drawing.




(a) State 1 of PL2T (b) State 3 of PL2T (c) State 2 of PL2T

Figure 15: Parallel locking 2-toggle using .

**Proof.** The decision problem is in PSPACE by Proposition 2. Figures 15, 18, 17, and 19, implement locking 2-toggles with $S = \{$ $\}$ or $S = \{$ $\}$ or $S = \{$ $\}$ or $S = \{$ $\}$, respectively. Therefore, the result follows from Theorem 10 of [9]. □

## 5 Summary and Future Work

We have shown that motion planning through turnstiles is PSPACE-complete for three shapes of turnstiles, regardless of whether they are thick or thin. However, the decision problem is solvable in polynomial-time when restricted to +-shaped turnstiles that are thick or thin. The unresolved singleton case is $i$-shaped turnstiles (i.e. and ); the pairings of +-shaped and $i$-shaped is also open. These complexity results are summarized in Figure 20 for thick turnstiles, and the same results hold for thin turnstiles.

### 5.1 Future Work

One can classify the turnstiles that we have considered in this article as *short turnstiles*, in the sense that the arms have length one. Considering long turnstiles is a natural next step, and these mechanisms would better model the Pokemon series of games (see Figure 1b).

Rotating walls and swivel doors from Section 2 also provide open problems. Another variation involves *ratchet rotation*, in which turnstiles can only turn one direction (i.e. clockwise or counterclockwise).

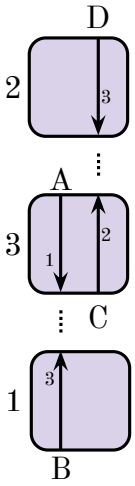
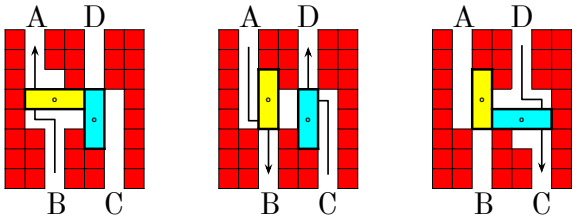
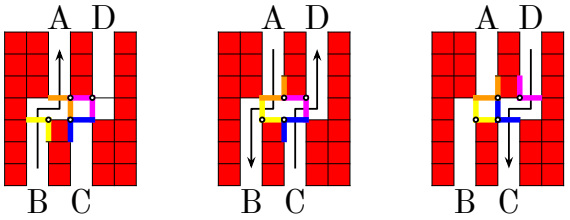



Figure 16: Antiparallel locking 2-toggle (AL2T) gadget.



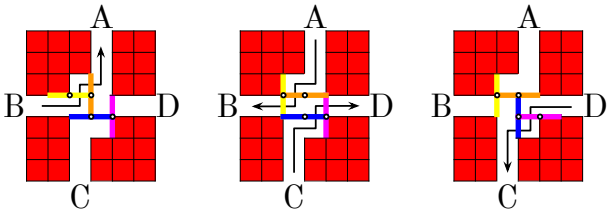

(a) State 1 of AL2T (b) State 3 of AL2T (c) State 2 of AL2T

Figure 17: Antiparallel locking 2-toggle using .




(a) State 1 of AL2T (b) State 3 of AL2T (c) State 2 of AL2T

Figure 18: Antiparallel locking 2-toggle using .




(a) State 1 of AL2T (b) State 3 of AL2T (c) State 2 of AL2T

Figure 19: Antiparallel locking 2-toggle using .

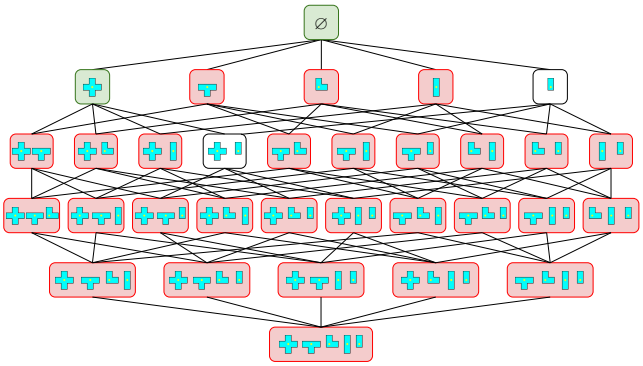



Figure 20: A summary of computational complextiy results for the TURNSTILE$_S$ decision problem for all $S \subseteq \{$ , , , , $\}$. The subsets in red nodes are PSPACE-complete, and the subsets in green nodes are in P. The two subsets in white nodes, $\{$ $\}$ and $\{$ , $\}$, remain open.

## 5.2 Acknowledgements

The initial work on this project was conducted at the *32nd Bellairs Winter Workshop on Computational Geometry* in early 2017. We thank participant Mikhail Rudoy for pointing us towards the motion planning framework, which was still in its early stages of development at the time. We also thank Erik Demaine and Jayson Lynch for several helpful discussions.

## References

[1] J. Ani, S. Asif, E. D. Demaine, Y. Diomidov, D. H. Hendrickson, J. Lynch, S. Scheffler, and A. Suhl. Pspace-completeness of pulling blocks to reach a goal. *J. Inf. Process.*, 28:929–941, 2020.

[2] J. Ani, J. Bosboom, E. D. Demaine, Y. Diomidov, D. H. Hendrickson, and J. Lynch. Walking through doors is hard, even without staircases: Proving pspace-hardness via planar assemblies of door gadgets. In M. Farach-Colton, G. Prencipe, and R. Uehara, editors, *10th International Conference on Fun with Algorithms, FUN 2021, May 30 to June 1, 2021, Favignana Island, Sicily, Italy*, volume 157 of *LIPIcs*, pages 3:1–3:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[3] J. Ani, E. D. Demaine, D. H. Hendrickson, and J. Lynch. Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. *CoRR*, abs/2005.03192, 2020.

[4] A. Barr, C. Chang, and A. Williams. Block Dude puzzles are NP-hard (and the rugs really tie the reductions together). In *Proceedings of the 33rd Canadian Conference on Computational Geometry, Dalhousie University, Halifax, Canada, August 10-12, 2021*, 2021.

[5] J. Culberson. Sokoban is PSPACE-complete. In *In Proceedings of the 1st International Conference on Fun with Algorithm*, pages 65–76, 1998.

[6] E. Demaine, M. Demaine, M. Hoffmann, and J. O'Rourke. Pushing blocks is hard. *Computational Geometry*, 26:21–36, 2003.

[7] E. D. Demaine, I. Grosof, J. Lynch, and M. Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In H. Ito, S. Leonardi, L. Pagli, and G. Prencipe, editors, *9th International Conference on Fun with Algorithms, FUN 2018, June 13-15, 2018, La Maddalena, Italy*, volume 100 of *LIPIcs*, pages 18:1–18:21. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

[8] E. D. Demaine, R. A. Hearn, and M. Hoffmann. Push-2-f is PSPACE-complete. In *Proceedings of the 14th Canadian Conference on Computational Geometry, University of Lethbridge, Alberta, Canada, August 12-14, 2002*, pages 31–35, 2002.

[9] E. D. Demaine, D. H. Hendrickson, and J. Lynch. Toward a general complexity theory of motion planning: Characterizing which gadgets make games hard. In T. Vidick, editor, *11th Innovations in Theoretical Computer Science Conference, ITCS 2020, January 12-14, 2020, Seattle, Washington, USA*, volume 151 of *LIPIcs*, pages 62:1–62:42. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[10] D. Dor and U. Zwick. Sokoban and other motion planning problems. *Computational Geometry*, 13(4):215 – 228, 1999.

[11] M. Fryers and M. T. Greene. Sokoban, 1995.

[12] A. Greenblatt, J. Kopinsky, B. North, M. Tyrrell, and A. Williams. MazezaM levels with exponentially long solutions. In *20th Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2017)*, pages 109–110, 2017.

[13] A. Greenblatt and A. Williams. MazezaM – puzzle game. https://community.arduboy.com/t/mazezam-puzzle-game/3723/25.

[14] R. A. Hearn and E. D. Demaine. The nondeterministic constraint logic model of computation: Reductions and applications. In P. Widmayer, F. T. Ruiz, R. M. Bueno, M. Hennessy, S. J. Eidenbenz, and R. Conejo, editors, *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, volume 2380 of *Lecture Notes in Computer Science*, pages 401–413. Springer, 2002.

[15] R. A. Hearn and E. D. Demaine. *Games, Puzzles, and Computation*. A K Peters/CRC Press, 1st edition, 2009.

[16] J. R. Lynch. *A framework for proving the computational intractability of motion planning problems*. PhD thesis, MIT, 9 2020. https://dspace.mit.edu/handle/1721.1/129205.

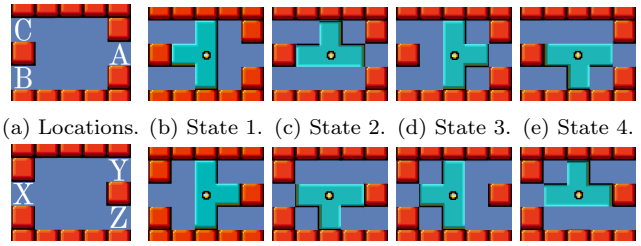
[17] B. North. Simpler exponential MazezaM level family. https://bennorth.github.io/simpler-exponential-mazezam/index.html.


## Appendix

We conclude with several notes, which may be helpful for some readers.

## A Modeling Level 1

Figures 21–23 shows how Level 1 in Kwirk can be modeled using gadgets and the motion planning framework. Note that the two gadgets are simply rotations of each other.

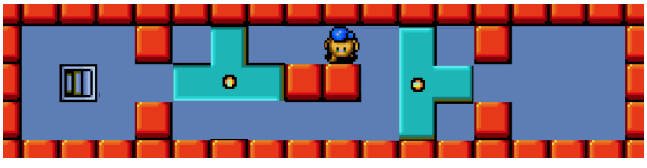



(a) Locations. (b) State 1. (c) State 2. (d) State 3. (e) State 4.

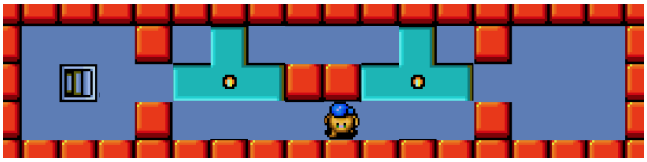
(f) Locations. (g) State 1. (h) State 2. (i) State 3. (j) State 4.

Figure 21: (a)–(e) Gadget $g_1 = (n_1, L_1, T_1)$ with $n_1 = 4$ states, location set $L_1 = \{A, B, C\}$, and traversal set $T_1 = \{(1, A, 2, B), (1, A, 3, B), (1, A, 3, C), (1, A, 4, C), \ldots\}$. (f)–(j) Gadget $g_2 = (4, L_2, T_2)$ with $L_2 = \{X, Y, Z\}$ and $T_2 = \{(1, X, 2, Y), (1, X, 3, Y), (1, X, 3, Z), (1, X, 4, Z), \ldots\}$.


Figure 22: Modeling Level 1 from Figure 1a using the system of gadgets $S = (G, C)$, with gadget set $G = \{g_1, g_2\}$, and connections $C = \{\{B, Z\}, \{C, Y\}\}$. Note that gadget $g_1$ is on the right.



(a) System state $(a, s_1, s_2)$ with agent location $a = C$, and gadget states $s_1 = 3$ and $s_2 = 4$.



(b) System state $(a, s_1, s_2)$ with agent location $a = B$, and gadget states $s_1 = 2$ and $s_2 = 4$.

Figure 23: (a)–(b) A traversal move in gadget $g_1$ in system $S$. The traversal is $(3, C, 2, B) \in T_1$, and the agent can move right, down, down, right, left, left, left to make this traversal.

## B Membership in PSPACE

The proof of Proposition 2 uses standard techniques and appears below.

**Proof.** We will prove that TURNSTILE is in NPSPACE, which establishes the result by Savitch's theorem. For simplicity, our argument only considers levels with thick turnstiles; thin turnstiles can be handled by considering grid lines as well as grid cells.

Consider an instance of the problem TURNSTILE($L$) in which $L$ has a total of $n$ grid cells. The size of the input is then $O(n)$ since each cell can be occupied by a small number of different game elements, hence, each cell contributes a small number of bits to the level's encoding.

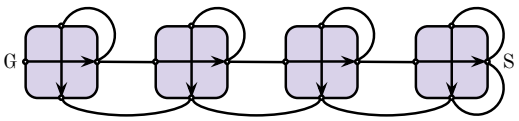
Our algorithm starts in the initial state of the level, and then non-deterministically makes moves (i.e. up, down, left, right) until the player reaches the exit position and the algorithm returns yes, or we exceed a pre-determined maximum number of moves. The maximum number of moves that we allow is chosen to be at least the number of different states that the level can have, since if there is a solution, then there is a solution that does not repeat any states. The state of a level consists of the position of the player's token, and the orientation of each turnstile. Each turnstile can have at most 4 orientations, and the number of turnstiles is at most $n$. Thus, the number of states is at most $n \cdot 4^n$. To implement our algorithm, we need to store the current state of the level, and the move counter. The state of the level requires $O(n)$-bits, and the move counter requires $\log n \cdot 4^n = 2n \log n$ bits. Therefore, our algorithm uses $O(n \log n)$ nondeterministic space. □

## C Explicit Exponential Level Construction

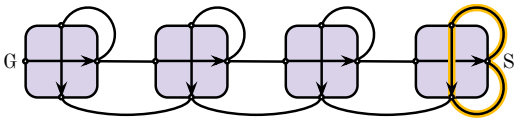
There are two distinct benefits to constructing simple levels that require an exponential number of moves to solve.

1. It illustrates that the simplest certificate (i.e. the sequence of moves) is not sufficient for establishing membership in NP. For example, see [12, 17] for the puzzle game *MazezaM*.
2. Game developers may wish to include such a level in their game. For example, this is true for MazezaM on the Arduboy [13].

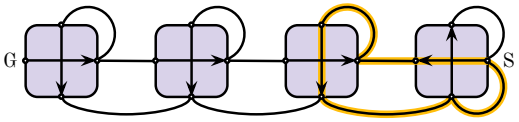
For these reasons, a lexicographic 4-bit binary counter using crossing 2-toggles (C2T) is given in Figure 25, along with an implementation in Kwirk using turnstiles.



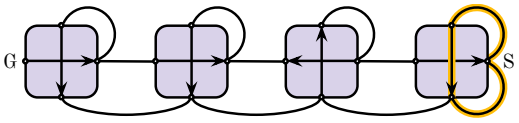

(a) A 4-bit counter in its initial state $b_4b_3b_2b_1 = 0000$.



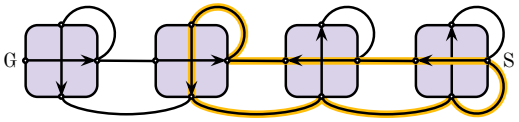

(b) State 0000. The highlighted loop complements $b_1$.



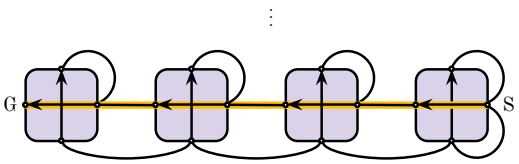

(c) State 0001. The highlighted loop complements $b_2b_1$.




(d) State 0010. The highlighted loop complements $b_1$.

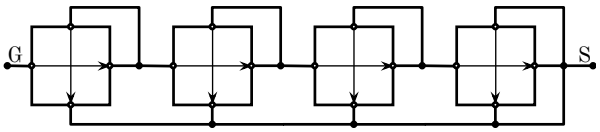



(e) State 0011. The highlighted loop complements $b_3b_2b_1$.

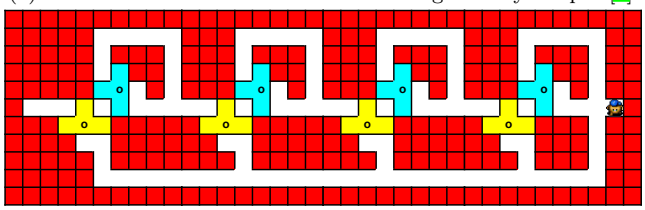



(f) State 1111. Reaching the goal via the highlighted path.

Figure 24: Binary counting with motion planning gadgets. (a) A 4-bit counter using crossing 2-toggles. (b)–(f) Each image begins with the player at the start location $S$, and the counter state is a binary string $b_4b_3b_2b_1$ based on the state of each gadget. The player must traverse states $0000, 0001, \ldots, 1111$ to reach the goal location $G$.




(a) The 4-bit counter drawn with branching hallways as per [7].



(b) A 4-bit counter in Kwirk.

Figure 25: (a) The 4-bit counter from Figure 24 implemented in Kwirk using the crossing 2-toggle with $S = \{$ $\}$. It follows the same design, but with (a) branching hallways, as in the style of [7].