Walking Through Doors Is Hard, Even Without Staircases: Proving PSPACE-Hardness via Planar Assemblies of Door Gadgets

Joshua Ani
Massachusetts Institute of Technology, Cambridge, MA, USA
joshuaa@mit.edu

Jeffrey Bosboom
Massachusetts Institute of Technology, Cambridge, MA, USA
jbosboom@mit.edu

Erik D. Demaine
Massachusetts Institute of Technology, Cambridge, MA, USA
edemaine@mit.edu

Yenhenii Diomidov
Massachusetts Institute of Technology, Cambridge, MA, USA
diomidov@mit.edu

Dylan Hendrickson
Massachusetts Institute of Technology, Cambridge, MA, USA
dylanhen@mit.edu

Jayson Lynch
Massachusetts Institute of Technology, Cambridge, MA, USA
jaysonl@mit.edu

Abstract

A door gadget has two states and three tunnels that can be traversed by an agent (player, robot, etc.): the "open" and "close" tunnel sets the gadget's state to open and closed, respectively, while the "traverse" tunnel can be traversed if and only if the door is in the open state. We prove that it is PSPACE-complete to decide whether an agent can move from one location to another through a *planar* assembly of such door gadgets, removing the traditional need for crossover gadgets and thereby simplifying past PSPACE-hardness proofs of Lemmings and Nintendo games Super Mario Bros., Legend of Zelda, and Donkey Kong Country. Our result holds in all but one of the possible local planar embedding of the open, close, and traverse tunnels within a door gadget; in the one remaining case, we prove NP-hardness.

We also introduce and analyze a simpler type of door gadget, called the *self-closing* door. This gadget has two states and only two tunnels, similar to the "open" and "traverse" tunnels of doors, except that traversing the traverse tunnel also closes the door. In a variant called the *symmetric* self-closing door, the "open" tunnel can be traversed if and only if the door is closed. We prove that it is PSPACE-complete to decide whether an agent can move from one location to another through a *planar* assembly of either type of self-closing door. Then we apply this framework to prove new PSPACE-hardness results for several 3D Mario games and Sokobond.

2012 ACM Subject Classification Theory of computation → Problems, reductions and completeness

Keywords and phrases gadgets, motion planning, hardness of games

Digital Object Identifier 10.4230/LIPIcs.FUN.2021.3

Related Version https://arxiv.org/abs/2006.01256

Acknowledgements This work was initiated during open problem solving in the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.892) taught by Erik Demaine in Spring 2019. We thank the other participants of that class for related discussions and providing an inspiring atmosphere.

10th International Conference on Fun with Algorithms (FUN 2021).
Editors: Martin Farach-Colton, Giuseppe Prencipe, and Ryuhei Uehara; Article No. 3; pp. 3:1–3:23

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Puzzle video games are rife with doors that block the player's passage when closed/locked. To open such a door, the player often needs to collect the right key or keycard, or to press the right combination of buttons or pressure plates, or to solve some other puzzle. Many of these game features in sufficient generality imply that the video game is NP-hard or PSPACE-hard, according to a series of metatheorems starting at FUN 2010 [7, 9, 10].

An intriguing twist is to use doors as a framework for proving hardness of video games that do not "naturally" have doors, but have some mechanics that suffice to simulate doors via a gadget. The first use of a local "door gadget" was by Viglietta to prove Lemmings PSPACE-complete at FUN 2014 [11]. This door gadget is a portion of a level design containing three directed paths that the player can traverse: a "traverse" path that can be traversed if and only if the door is open, a "close" path that forces the door to close, and an "open" path that allows the player to open the door if desired. Viglietta [11, Metatheorem 3] proved that such a door gadget, together with the ability to wire together door entrance/exit ports according to an arbitrary graph (including crossovers for a 2D game like Lemmings), where the player has the choice of how to traverse the graph, suffice to prove PSPACE-hardness of deciding whether an agent can move from one location to another. At the same FUN, Aloupis et al. [1] used this door framework to prove Legend of Zelda: Link to the Past and Donkey Kong Country 1, 2, and 3 PSPACE-complete [1]. At the next FUN, Demaine et al. [6] used this door framework to prove Super Mario Bros. PSPACE-complete. All of these proofs feature a crossover gadget for wiring paths between door gadgets.

The motion-planning-through-gadgets framework of [4, 5] (initiated at FUN 2018) formalizes the idea of moving one or more agents through a graph of local gadgets, where each gadget has local state and traversal paths whose traversal affects that gadget's state (only). In the 1-player unbounded setting considered here, past work analyzed gadgets that are

1. ***deterministic***, meaning that when an agent enters a gadget at any location, it has a unique exit location and causes a unique state change;
2. ***reversible***, meaning that every such traversal can be immediately undone, both in terms of agent location and gadget state change; and
3. ***k-tunnel***, meaning that the $2k$ entrance/exit locations can be paired up such that, in any state, traversal paths only connected paired locations (in some direction).

Restricted to deterministic reversible k-tunnel gadgets, Demaine et al. [5] characterized which gadget sets make motion planning of an agent from one location to another PSPACE-complete: whenever the gadget set contains a gadget with ***interacting tunnels***, meaning that traversing some traversal path changes (adds or removes) the traversability of some other traversal path (in some direction). Furthermore, they proved the same characterization when the gadgets are connected in a planar graph, obviating the need for a crossover gadget.

Door gadgets naturally fit into this motion-planning-through-gadgets framework. (Indeed, they were one of the inspirations for the framework.) Notably, however, the door gadget used in [1, 6, 11] is neither deterministic (the open path can open the door or not, according to the player's choice) nor reversible (the paths are all directed in fixed directions), so the existing characterization and planarity result do not apply.

In this paper, we develop a specialized motion-planning-through-*doors* framework, completing another subspace of the motion-planning-through-gadgets framework. Our framework applies to a variety of different door gadgets, including the door gadget of [1, 6, 11]. In all cases, a door gadget has two states and three disjoint traversal paths: "traverse", "close", and "open". Each path may be individually ***directed*** (traversable in one direction) or ***undirected***

(traversable in both directions). In addition, the open traversal path may have identical entrance and exit locations, meaning that its traversal changes the door's state but does not move the agent (breaking the k-tunnel assumption). In this way, we can require that traversing the open and close traversal paths force the door's state to open and closed, respectively, but still effectively allow the player to make a choice of whether to open the door (by skipping or including the open traversal, which leaves the agent in the same location either way).

In Section 2, we introduce two more families of door gadgets. A ***self-closing door*** has two states but only two traversal paths: "open" and "self-close". The self-close traversal is possible only in the open state, and it forcibly changes the state to closed. As before, each traversal path can be either directed or undirected; and the open traversal forces the state to open, but we allow the open traversal path to have identical start and end locations, which effectively allows optional opening. A ***symmetric self-closing door*** has two states and two traversal paths: "self-open" and "self-close". The self-open/close traversal is possibly only in the closed/open state, respectively, and it forcibly changes the state to open/closed, respectively. (This definition is fully symmetric between "open" and "close".) Each traversal path can be either directed or undirected, but we no longer allow optional traversal.

In Section 3, we prove that *planar* 1-player motion planning is PSPACE-complete for every door gadget, for every local combinatorial planar embedding of every type door gadget except for one (which we only prove NP-hard). Thus, all that is needed to prove a new game PSPACE-hard is to construct any single supported door gadget, and to show how to connect the door entrances/exits together in a planar graph. In particular, the crossover gadgets previously constructed for Lemmings [11, Figure 2(e)], Legend of Zelda: Link to the Past and Donkey Kong Country 1, 2, and 3 [1, Figures 28 and 20], and Super Mario Bros. [6, Figure 5] are no longer necessary for those PSPACE-hardness proofs: they can now be omitted. (See Section 4 for details.) Our result should therefore make it easier in the future to prove 2D games PSPACE-hard. Because of their reduced conceptual complexity – only two traversal paths, which behave essentially identically for symmetric self-closing doors – we have found it even easier to prove games PSPACE-hard by building self-closing door gadgets.

In the full version of the paper we prove that every door is ***universal***, meaning that any one of them can simulate *all* gadgets in the motion-planning-through-gadgets framework of [4, 5]. This provides the first examples of fully universal gadgets.

In Section 4, we illustrate this approach by proving PSPACE-hardness for one 2D game, Sokobond, and several different 3D Mario games: Super Mario 64, Super Mario 64 DS, Super Mario Sunshine, Super Mario Galaxy, and Captain Toad: Treasure Tracker (and the associated levels in Super Mario 3D World). Additional applications to Super Mario Galaxy 2 and Super Mario 3D Land/World are presented in the full version of the paper. These reductions consist of just one gadget, a symmetric self-closing door, along with easy methods for connecting these gadgets. For the 3D games, the main benefit is the simplicity of the symmetric self-closing door: crossovers are generally easy in the 3D games, though conveniently we still do not need to explicitly build them.

2 Self-Closing Doors

In this section, we introduce different kinds of self-closing doors and show that 1-player motion planning is PSPACE-hard for them.

2.1 Terminology

A ***self-closing door*** is a 2-state gadget that has a tunnel that closes itself when traversed (the *self-closing* tunnel), a tunnel/port that reopens said tunnel (the *opening* tunnel/port), and no other ports. We will talk about two major kinds of self-closing door. A ***normal self-closing door*** is a self-closing door where the open path/tunnel is always open. A ***symmetric self-closing door*** is a self-closing door where the open path/tunnel is a tunnel and also closes itself when traversed. As with doors, these can be ***directed***, ***undirected***, or ***mixed***, and a normal self-closing door can also be ***open-required*** or ***open-optional***. An 'X' on a tunnel indicates that the tunnel closes itself when traversed. A dotted line indicates a closed tunnel and a solid line indicates an open tunnel. For normal self-closing doors, the open path/tunnel will be colored green. Figure 1 shows some self-closing doors.

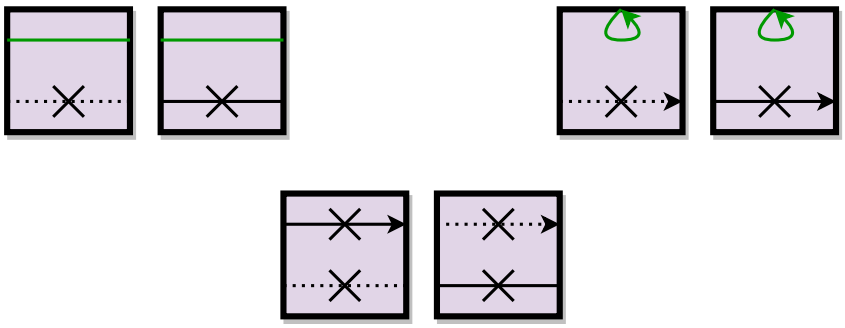

Figure 1 Left: An undirected open-required normal self-closing door. Right: A directed open-optional normal self-closing door. Bottom: A mixed symmetric self-closing door.

2.2 PSPACE-hardness of Self-Closing Doors

In this section we show PSPACE-hardness for 1-player motion planning with any of the self-closing doors. We do so by showing undirected self-closing doors can simulate diodes, and self-closing doors without open-optional tunnels can simulate ones with open-optional tunnels. We then prove the main Theorem 2.3 which gives PSPACE-hardness of the directed, open-optional, normal self-closing door by simulating a directed, open-optional door gadget.

▶ **Lemma 2.1.** *In 1-player motion planning, any normal or symmetric self-closing door can simulate an open-optional self-closing door.*

Proof. In the case of an open-optional normal self-closing door, we are done. In the case of an open-required normal self-closing door, we do the same thing we did for the proof for Theorem 3.6. In the case of a symmetric self-closing door, we pick a tunnel to be the opening tunnel and do what we did for Theorem 3.6. This simulates an open-optional self-closing door. ◀

▶ **Lemma 2.2.** *1-player motion planning with the undirected open-optional normal self-closing door can simulate a directed open-optional normal self-closing door.*

Proof. We can simulate a diode by wiring 2 undirected open-optional normal self-closing doors as shown in Figure 4. The player can enter from the left, open the left self-closing door, traverse it, and do the same for the right self-closing door. The player cannot enter from the right. If the player tries to open the left self-closing door and then leave, the player

still cannot enter from the right. If the player tries to open the right self-closing door and then leave, they will not be able to leave. So this simulates a diode. We can wire a diode to each side the self-closing tunnel to get a directed self-closing tunnel which can be applied to make the undirected self-closing door directed. ◀

▶ **Theorem 2.3.** *1-player motion planning with the directed open-optional normal self-closing door is PSPACE-hard.*

Proof. We can simulate a diode by wiring the opening port to the input end of the self-closing tunnel. The player can open the self-closing tunnel then traverse it, but cannot go the other way because the self-closing tunnel is directed. Then we show that we can duplicate the open port and the self-closing tunnel as in Figure 2. We then actually triplicate the open port and duplicate the self-closing tunnel, and wire them up to simulate the directed open-optional door as shown in Figure 3, for which PSPACE-hardness is known. ◀

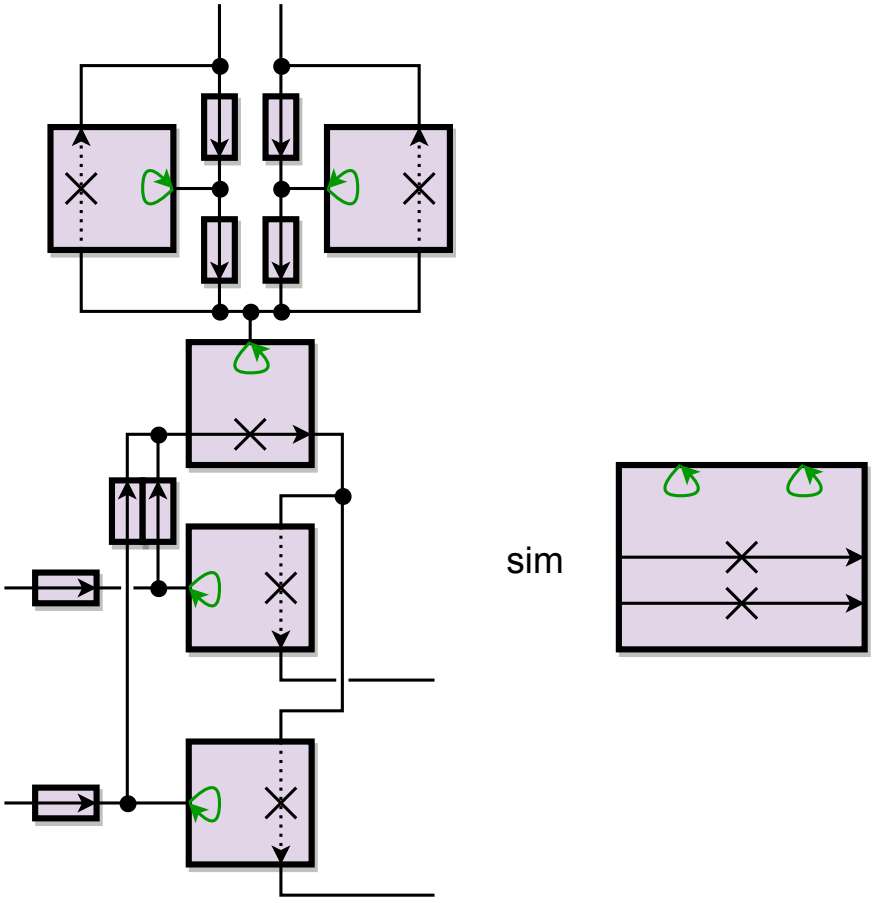

Figure 2 The directed open-optional normal self-closing door can simulate a version of itself with the opening port and the self-closing tunnel duplicated. Note that the opening port duplicator is planar.

Chaining the simulations in Lemmas 2.1 and 2.2 with Theorem 2.3 we obtain PSPACE-hardness for all variations.

▶ **Corollary 2.4.** *1-player motion planning with any normal, symmetric, or open-optional normal self-closing door is PSPACE-hard.*

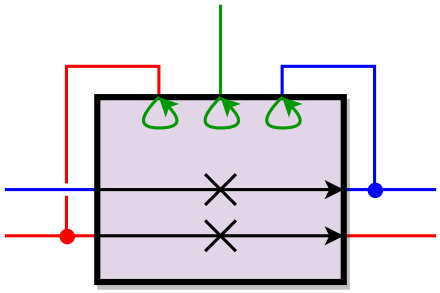

Figure 3 Simulation of the directed open-optional door. Green wires correspond to the opening port; blue wires correspond to the traverse tunnel; and red wires correspond to the closing tunnel. Note that the player has no reason to not open the gadget after traversing the blue wire.

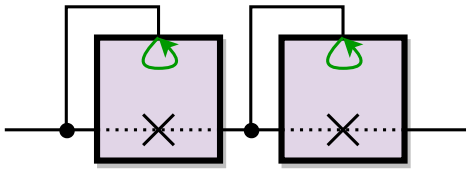

Figure 4 Undirected open-optional normal self-closing door simulating a diode.

3 Planar Doors

In this section, we adapt the door framework of [1, Section 2.2] (a cleaner presentation of the framework from [11]) into the motion-planning-through-gadgets framework. We then improve upon those results by showing most variations on the door gadget remain PSPACE-hard in the planar case. We also show that 1-player planar motion planning with any normal or symmetric self-closing door is PSPACE-hard.

3.1 Terminology

We define a ***door*** to be a gadget with an *opening* port or tunnel, a *traverse* tunnel, and a *closing* tunnel, and each of the tunnels may be directed or undirected. The opening port/tunnel opens the traverse tunnel, and the closing tunnel closes the traverse tunnel. Throughout this paper, the opening port/tunnel will be colored green, the traverse tunnel will be colored blue, and the closing tunnel will be colored red. In addition, a solid traverse tunnel represents an open door, and a dotted traverse tunnel represents a closed door. A ***directed door*** is a door where all tunnels are directed. An ***undirected door*** is a door where all tunnels are undirected. A door that is neither undirected nor directed is a ***mixed door***. An ***open-required door*** is a door with an opening tunnel, and an ***open-optional door*** is one with an opening port. A directed open-required door, an undirected open-required door, and a mixed open-optional door are shown in Figure 5.

In 2D, we care about the arrangement of ports in a gadget. For *planar motion planning* problems we want a *planar* system of gadgets, where the gadgets and connections are drawn in the plane without crossings. Planar gadgets also specify a clockwise ordering of their ports, although we consider rotations and reflections of a gadget to be the same. A single gadget type thus corresponds to multiple planar gadget types, depending on the choice of the order of locations. For a planar system of gadgets, the gadgets are drawn as small diagrams with points on their exterior coorisponding to their ports and connections are drawn as paths connecting the points corresponding to the ports without crossing gadget interiors or other connections.

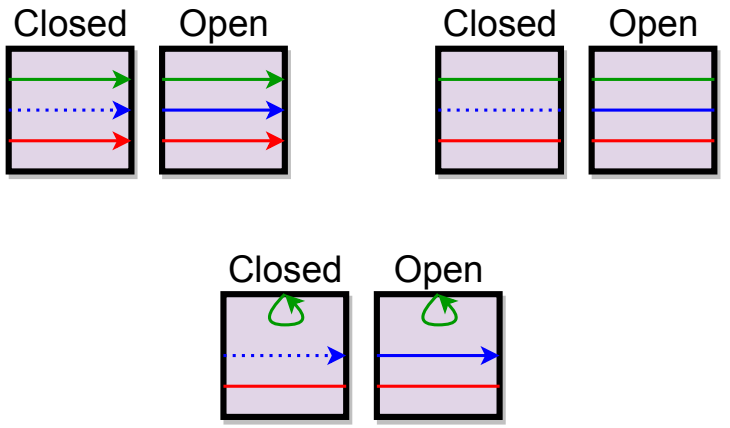

Figure 5 Left: A directed open-required door. Right: An undirected open-required door. Bottom: A mixed open-optional door.

3.2 PSPACE-hardness for Planar Self-Closing Doors

For completeness, we give a proof that the planar directed open-optional normal self-closing door is PSPACE-hard. This result was also given in [2].

▶ **Theorem 3.1.** *1-player planar motion planning with the directed open-optional normal self-closing door is PSPACE-hard.*

Proof. Since Theorem 2.3 shows PSPACE-completeness in the non-planar case, it will suffice to build a crossover gadget. First, we wish to duplicate the opening ports as in the prior proof. We show how to do so in Figure 2. Note that this time we cannot directly duplicate the self-closing tunnel as the construction from Theorem 2.3 uses crossovers. We can also simulate a diode as proven in Theorem 2.3 since the construction is planar. We use these to simulate a pair of self-closing doors where the opening ports alternate which door they open, shown in Figure 6. If the agent enters from port 1 or 4, they will open door E or F, respectively, and then leave. If the agent enters from port 2, they can open doors A, B, and C. Assume they then traverse door B. If they then open door E, they would have to traverse door C, maybe open F, and get stuck. So instead of opening door E, the agent traverses door A, ending up back at port 2 with no change except that door C is open. Entering port 2 or 3 gives the opportunity to open door C without being forced to take a different path, so leaving door C open does not help. So instead of traversing door B, the agent traverses door C. The agent is then forced to go right and can open door F. Then they are forced to traverse door B. If the agent opens door E, they will be stuck, so the agent traverses door A instead and returns to port 2, leaving door F open. Similarly, if the agent enters from port 3, the only useful thing they can do is open port E and return to port 3.

Using this, we then simulate a directed crossover as in Figure 7 which are able to simulate an undirected crossover, removing the planar restriction and reducing this problem to Theorem 2.3. In the simulation of a directed crossover, the agent must open the left tunnel of a gadget and then open both tunnels of the other one, forcing them to cross over, since the only path forward goes through the left tunnels of both gadgets. ◀

▶ **Theorem 3.2.** *1-player planar motion planning with any normal or symmetric self-closing door is PSPACE-hard.*

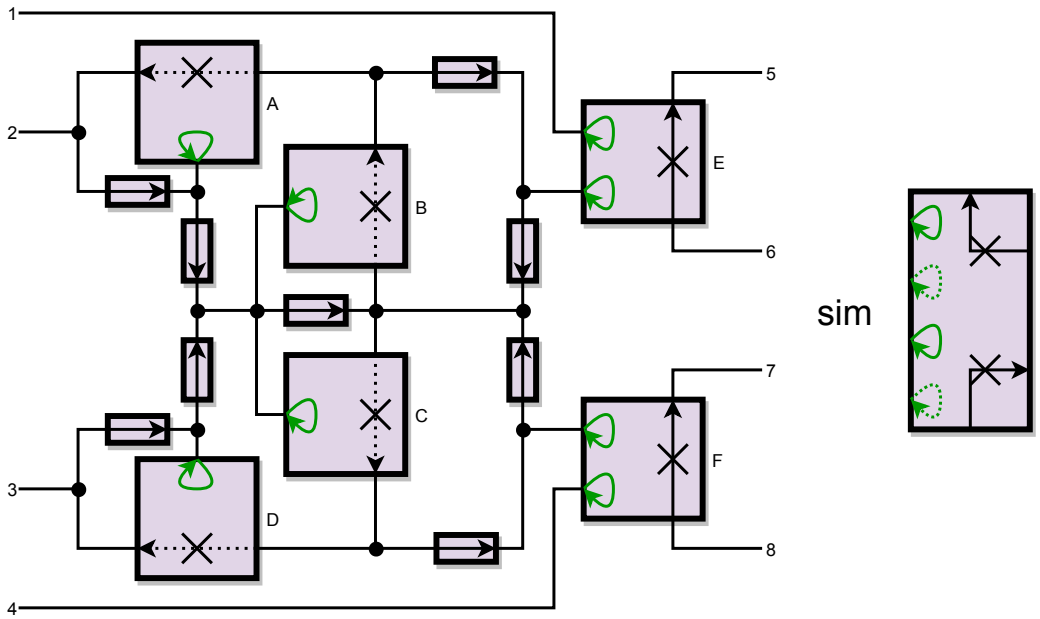

Figure 6 Directed open-optional normal self-closing door simulating the gadget on the right, where solid opening ports control the top self-closing tunnel and dotted opening ports control the bottom self-closing tunnel. The gadgets and external ports are labelled to help with the proof.

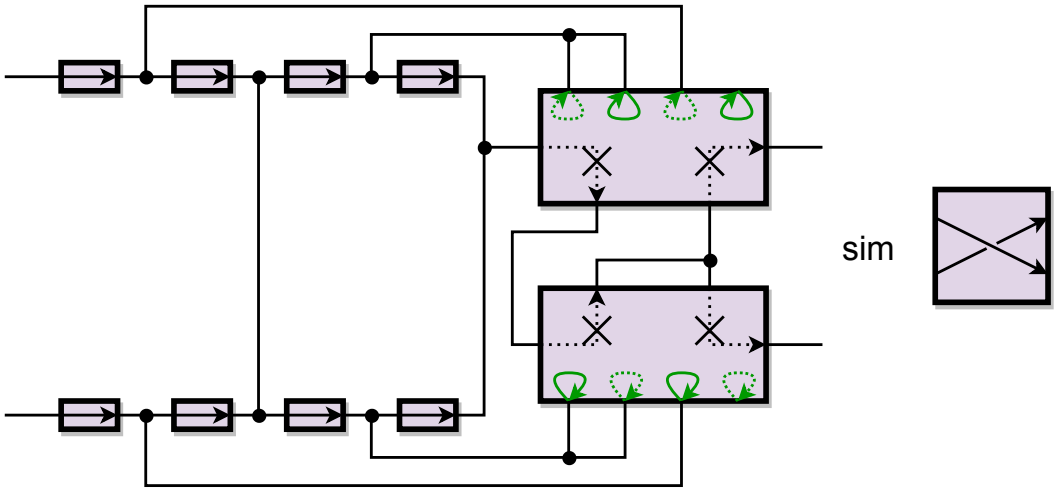

Figure 7 Directed open-optional normal self-closing door simulating a crossover.

Proof. Any normal or symmetric self-closing door can simulate a diode as shown in Figure 8(a–f). Then we can simulate the directed open-optional normal self-closing door as shown in Figure 9(a–d). Finally we apply Theorem 3.1 to show PSPACE-hardness. ◀

3.3 PSPACE-hardness for Planar Doors

We will show that 1-player planar motion planning with almost any door is PSPACE-hard by showing that 1-player planar motion planning with almost any fully directed door is PSPACE-hard and that mixed and undirected doors can planarly simulate at least one of the PSPACE-hard fully directed doors.

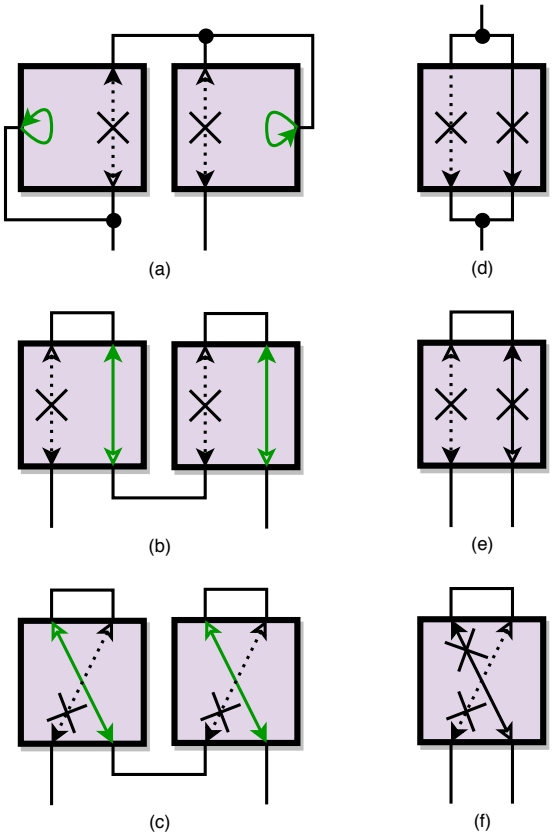

Figure 8 Six types of self-closing doors simulating diodes. Filled-in arrows indicate directions that are required to exist, and outlined arrows indicate optional directions. Case (a) is the same as Figure 4.

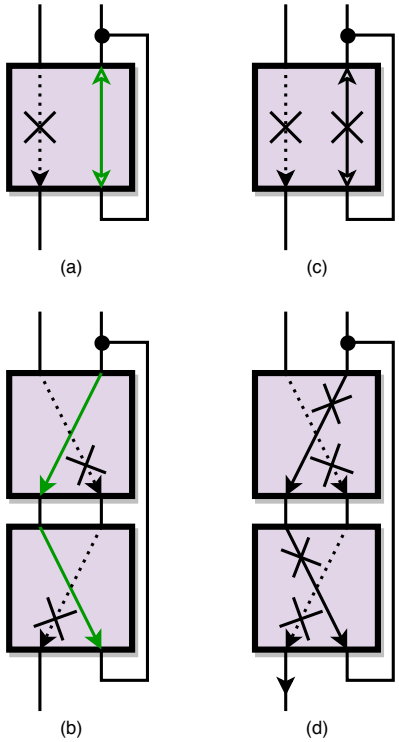

Figure 9 Four types of directed self-closing doors simulating the directed open-optional normal self-closing door. Filled-in arrows indicate directions that are required to exist, and exactly one of the outlined directions must exist.

We first show that mixed and undirected doors can simulate fully directed doors in Lemmas 3.3 and 3.4. Since undirected and partially directed doors can planarly simulate at least one fully directed door, it suffices to prove hardness for all fully directed doors. Next, we show hardness for all fully directed doors with at least one pair of crossing tunnels. We then show we can collapse adjacent opening ports to optional opening ports in Theorem 3.6, this leaves 12 fully directed doors with no crossing tunnels (Figure 10)s. These 12 doors are shown and named in Figure 10. Proofs for 11 of the 12 of these cases are given in Theorem 3.8. Finally, we show NP-hardness for the remaining Case 8: OTtocC door in Theorem 3.11.

▶ **Lemma 3.3.** *Any mixed door can planarly simulate some fully directed door which is not the Case 8: OTtocC door.*

Proof. Consider an arbitrary mixed door M. Since M is mixed, it has a directed tunnel. No tunnel changes its own traversability when crossed, so this tunnel simulates a diode. We wire each undirected tunnel of M through diodes at each end pointing in the same direction. This simulates a directed door. If M is not the door in Case 8: OTtocC, we are done. Otherwise, flip one set of diodes wired through an undirected tunnel of M, simulating a different directed door. ◀

▶ **Lemma 3.4.** *An undirected door can planarly simulate a fully directed door which is not the Case 8: OTtocC door.*

Proof. Consider an arbitrary undirected door U. We wire an external wire to a port of the opening port/tunnel. The player can visit the port, or if it is a tunnel, cross the tunnel both ways, to open the gadget. If the opening port/tunnel was a tunnel, this turns it into a port, making the gadget U'. Consider the order of the ports of the opening port O, the traverse tunnel $\{T_0, T_1\}$, and the closing tunnel $\{C_0, C_1\}$ around the edge of U', and label the ports p_0, p_1, p_2, p_3, p_4. We want to show that a traverse tunnel port is adjacent to a closing tunnel port. Assume not. Without loss of generality, let $p_0 = T_0$. Then $\{p_1, p_4\} = \{T_1, O\}$. But then $\{p_2, p_3\} = \{C_0, C_1\}$, and one of $\{p_2, p_3\}$ must be adjacent to a traverse tunnel port, a contradiction. Since one of the traverse tunnel ports, say T_1, is adjacent to one of the closing tunnel ports, say C_0, we wire T_1 to C_0 without blocking an opening port or opening tunnel port. This simulates a directed open-optional normal self-closing door: The player can open the gadget by going to the opening port (or if it is a tunnel, by going through the tunnel and back). If the gadget is open, the player can go through the traverse tunnel and then the closing tunnel, but cannot go the other way. If the gadget is closed, the player cannot go either way through the traverse-tunnel-closing-tunnel path. ◀

▶ **Theorem 3.5.** *1-player planar motion planning with any directed door with an internal crossing is PSPACE-hard.*

Proof. If the opening tunnel crosses the closing tunnel, then we have a crossover because these tunnels are always open. If the opening tunnel crosses the traverse tunnel, then we start the door open and have a crossover because neither tunnel closes itself or the other. Otherwise, the traverse tunnel crosses the closing tunnel and the opening port/tunnel can

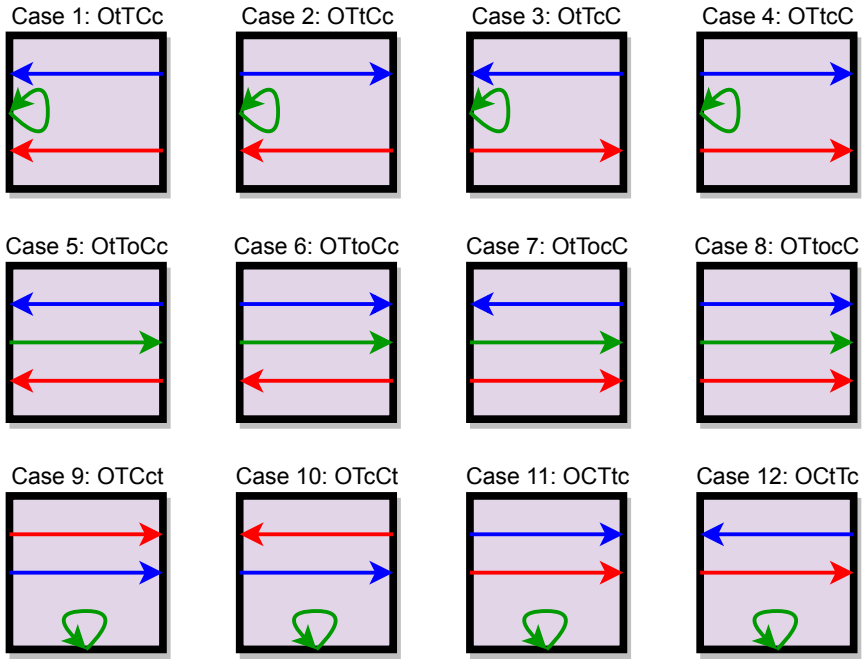

Figure 10 The twelve cases of a planar directed door without internal crossings. Opening tunnels with adjacent ports are merged into opening ports.

simulate an opening port. Then we have four cases, as shown in Figure 11. In cases 1, 2, and 4, we can simulate a crossover by connecting the opening port to either the input of the traverse tunnel or the output of the closing tunnel to ensure that the traverse tunnel is open when we need to use it. (Figure 12).

Case 3, however, is more tricky, as both of these ports are separated from the opening port by other ports. We use 2 copies to provide a path from the input of the traverse tunnel to the opening port without giving access to the close tunnel. The horizontal path of the crossover involves crossing from the left door to the right door, which is allowed as long as the left door is open. To take the vertical path, the player opens the middle door, goes down closing the left door, opens the right door, traverses the middle door, opens the left door (to keep the horizontal path open), and traverses the right door. The player can leave partway through this traversal, but this does nothing useful. So all doors with internal crossings can simulate crossovers, removing the planarity constraint. ◀

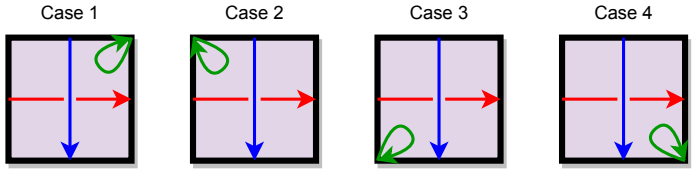

Figure 11 The four cases where the traverse tunnel crosses the closing tunnel but the opening port/tunnel does not cross either and can thus simulate a port.

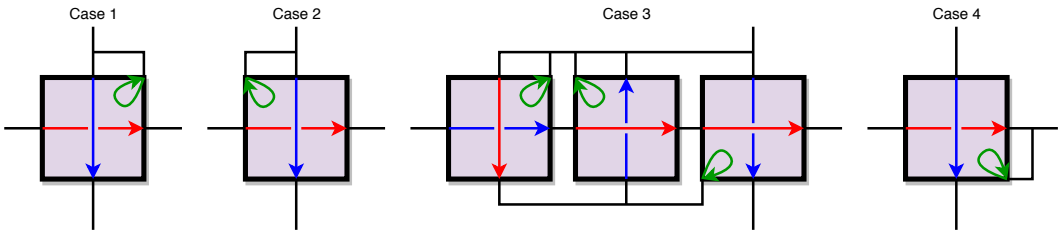

Figure 12 All four cases of the traverse tunnel crossing the closing tunnel can each simulate a crossover.

▶ **Theorem 3.6.** *In 1-player motion planning, any door can simulate its corresponding open-optional door.*

Proof. In case of a door that is not already open-optional, we wire one end of the open tunnel to the other end and wire some point on this loop externally as shown in Figure 13. This turns the open tunnel into an open port. ◀

Before continuing, we prove another gadget, the directed tripwire lock, is PSPACE-complete. Recall that a tripwire lock is a 2-state 2-tunnel gadget with an undirected tunnel that is traversable in exactly 1 state and an undirected tunnel that toggles the state of the gadget [4]. The ***directed tripwire lock*** is similar except that its tunnels are directed.

▶ **Lemma 3.7.** *1-player planar motion planning with the parallel directed tripwire-lock is PSPACE-hard.*

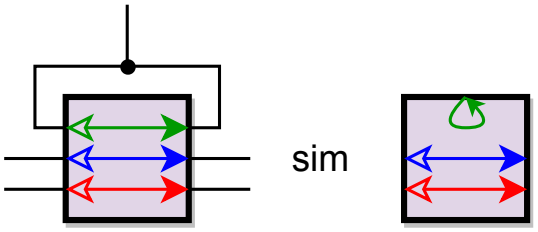

Figure 13 An open-required door simulates its corresponding open-optional door. Outlined arrows indicate optionally allowed traversals.

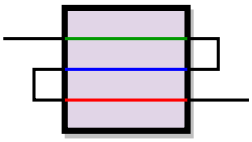

Figure 14 Simulation of a diode with an undirected door.

A proof can be found in the full version of the paper.

For directed doors, there are only the cases without internal crossings left. If the opening port/tunnel is a tunnel and its ports are adjacent, we easily simulate an opening port, reducing the number of cases to consider. There are twelve cases, shown in Figure 10. We name these cases based on the cyclic order of ports, with exits-only having lowercase letters.

▶ **Theorem 3.8.** *1-player planar motion planning with any directed door without internal crossings except the Case 8: OTtocC door is PSPACE-hard.*

Proof. We divide into multiple cases. Note the cases are numbered according to Figure 10, not in the order they are addressed in this proof.

Case 2: OTtCc, Case 10: OTcCt, and Case 12: OCtTc doors. In all these doors the opening port/tunnel is a port, and the traverse tunnel output is adjacent to the closing tunnel input. Thus, we can simulate a directed open-optional self-closing door by wiring the traverse tunnel output to the closing tunnel input and by attaching a wire to the open port, and these wires do not cross each other. Then this reduces to Theorem 3.2.

Case 1: OtTCc door. can simulate the directed version of the tripwire lock, as shown in Figure 15. We will refer to the gadgets numbered left to right. The lock is simply the traverse tunnel on door 1. In the two simulated states we will either have doors 1 and 3 open or door 2 open. If door 2 is open, when traversing the tripwire tunnel we can go through the traverse tunnel allowing us to open doors 1 and 4. With door 4 now open, we can go through its traverse tunnel opening door 3, and then closing door 4 on the way out. This leaves us with doors 1 and 3 open. Going through the tripwire tunnel again closes door 1 but allows us to go through the traverse tunnel of door 3, allowing us to open door 2. Doors 3 and 4 are then closed on the way out. There are states where we could fail to open all of these doors while traversing the close tunnel, but this will leave the gadget with strictly less traversability and thus the agent will never want to take such a path. Thus the Case 1: OtTCc door is PSPACE-complete by Lemma 3.7.

Case 3: OtTcC door. This door can simulate a directed open-optional normal self-closing door (Figure 16). If the agent enters from port O (the opening port), they can open doors 2 and 3. If they then leave, they have accomplished nothing because door 2 was already open, and door 3 can be opened from port O anyway and cannot be traversed from port T_0 or T_1 as we will see later. So they close door 2 instead. Then they can open door 1 and they are forced to traverse door 3. The agent can then reopen door 2 and return to port O. Now all the doors are open. If the agent then enters from port T_0, then they are forced to close door

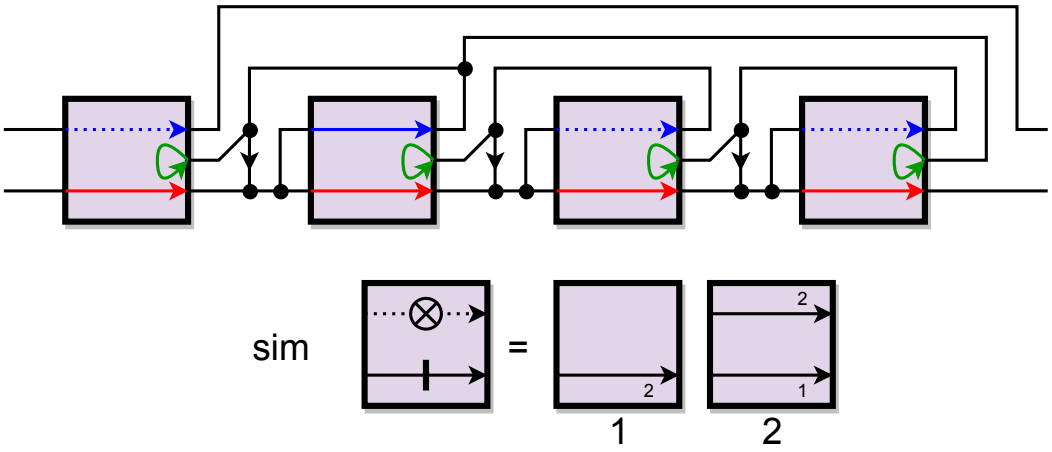

Figure 15 The Case 1: OtTCc door simulates the parallel directed tripwire lock. In addition, the state diagram of the directed tripwire lock. Arrows are drawn directly on wires to represent diodes.

3. They can then open door 1 (useless), and then they are forced to traverse door 2 and close door 1, leading to port T_1. The agent could not have taken this path initially because door 1 was closed, and they cannot take it again without visiting port O because they just closed door 1.

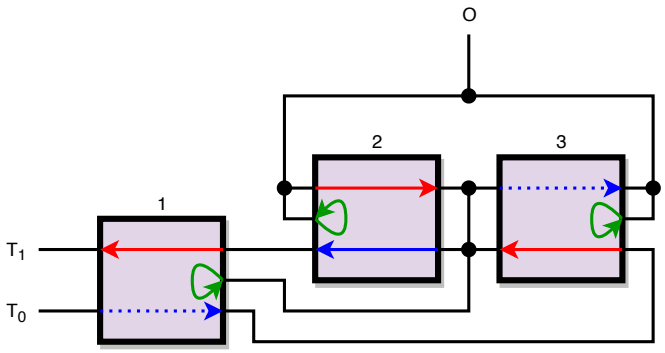

Figure 16 Simulation of a self-closing door with the Case 3: OtTcC door. The simulation starts in the closed state. Ports and gadgets are labelled.

Case 4: OTtcC door. A proof of this case can be found in the full paper.

Case 6: OTtoCc door. This door can simulate a directed open-optional normal self-closing door (Figure 17). If the agent enters from port O, they are forced to close door 3. If the agent then traverses door 2, they are forced to open door 3 and return to port O, accomplishing nothing. So the agent has no other option but to close door 1. If the agent tries to open door 2, they get stuck, so they instead open door 1. Continuing the loop involving door 1 does nothing, so the agent then traverses door 2, opens door 3, and returns to port O. Now door 1 is open. If the agent enters from port T_0, then they are forced to close door 2, traverse door 1, and close door 1. Reopening door 1 puts the agent back into the situation of being forced to close door 1, so the agent instead opens door 2 and traverses door 3 to port T_1.

The agent could not have taken this path initially since door 1 was closed, and they cannot take it again without visiting port O because they closed door 1.

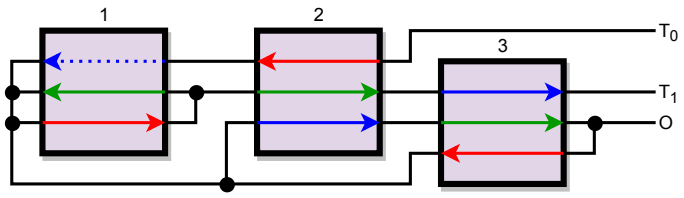

Figure 17 Simulation of a self-closing door with the Case 6: OTtoCc door. The simulation starts in the closed state. Ports and gadgets are labelled.

Case 5: OtToCc door. This door can simulate the Case 6: OTtoCc door, which has been covered, by effectively flipping the traverse tunnel. (Figure 18). Door 1 is the gadget that we flip the traverse tunnel of. If the agent enters from port T_0, they must open door 2, the close door 2. If door 1 is open and the agent then traverses it, they are back to a previous position with nothing changed. Instead, the agent opens door 3. If the agent then closes door 3, they get stuck because door 2 is closed. So they must close door 2 (again) or traverse door 3. These actions lead to the same situation. If the agent opens door 3 (again), they are back to the same situation that occurred after opening door 3 the first time. If door 1 is open, the agent then traverses door 1. Then they must open door 2. Closing door 2 leads to a previous situation, so the agent then traverses door 3. If the agent then traverses door 1 (again), they must open door 2 (again), leading to a previous situation. So they instead open door 3. Closing door 2 and traversing door 3 lead to different previous situations, so the agent then closes door 3, and then is forced to traverse door 2 to port T_1, leaving all the doors unchanged. If door 1 is not open, then the agent is unable to leave.

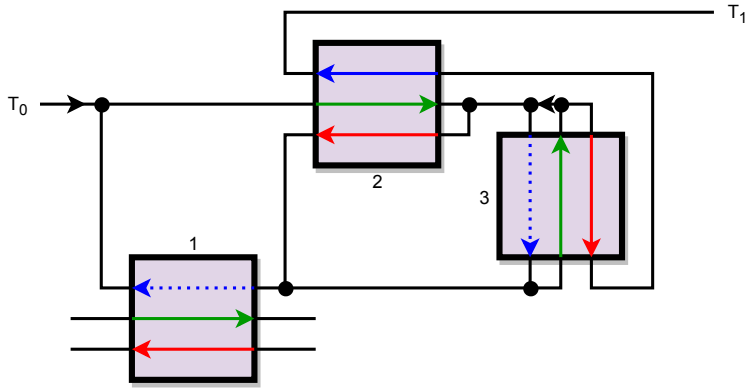

Figure 18 Simulation of the Case 6: OTtoCc with the Case 5: OtToCc door. The traverse tunnel of the leftmost gadget is effectively flipped.

Case 7: OtTocC door. This door can simulate a directed open-optional normal self-closing door (Figure 19). If the agent enters from port O, they must open door 1, then close door 2. If the agent then closes door 3, they get stuck because door 2 is closed. The agent can traverse door 1 and leave via port O, but they can also open and then traverse door 3 and then do the same thing, which is advantageous. So the agent opens and traverses door 3,

then traverses door 1 to port O. Now door 1 is open, door 2 is closed, and door 3 is open. If the agent enters from port T_0, they must close door 1, then open door 2, then traverse door 3. Opening door 3 and then traversing it is a no-op, and door 1 is closed, so the agent closes door 3 and then must traverse door 2 to port T_1. This leaves door 1 closed, door 2 open, and door 3 closed. The agent could not have taken this path initially because door 3 was closed, and cannot take it again without visiting port O first for the same reason.

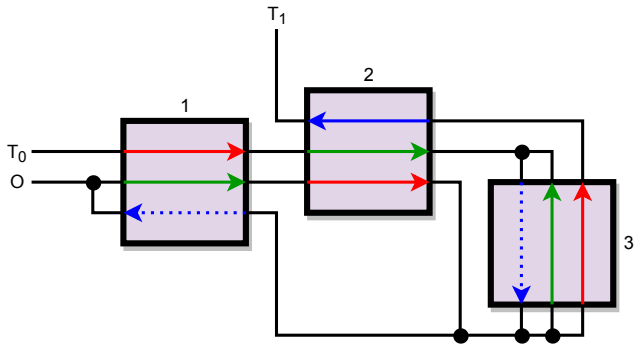

Figure 19 Simulation of a self-closing door with the Case 7: OtTocC door.

Case 9: OTCct door. This door can simulate a directed open-optional normal self-closing door (Figure 20). If the agent enters from port O, they can open door 1 and must close door 2. If the agent later enters from port T_0, then they must traverse door 1. They then can open door 2 (and must, since that is the only way out) and must close door 1. Then the agent traverses door 2 to port T_1. The agent could not have taken this path initially because door 1 was closed, and cannot take the path again without visiting port O first for the same reason.

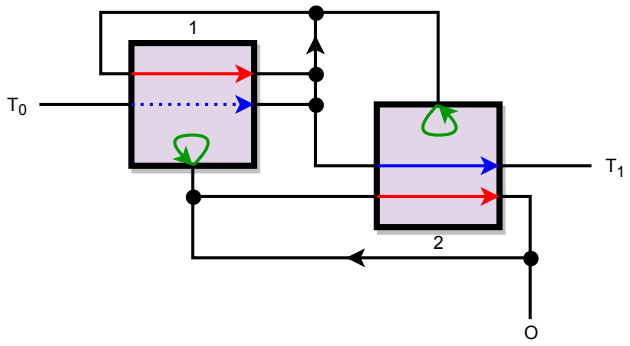

Figure 20 Simulation of a self-closing door with the Case 9: OTCct door.

Case 11: OCTtc door. A proof of this case can be found in the full paper.

This covers all the planar directed doors without internal crossings except the OTtocC door, finishing the proof. ◀

▶ **Theorem 3.9.** *1-player planar motion planning with any door except the door in Case 8: OTtocC is PSPACE-hard.*

Proof. This follows from Theorems 3.5, 3.8, 3.3, and 3.4, as those cover all the cases. ◀

To prove NP-hardness of the last case (Case 8: OTtocC), we first prove NP-hardness of other useful gadgets. A ***NAND*** gadget is a directed 2-tunnel gadget where traversing either tunnel closes both tunnels (preventing all future traversals). There are three planar types of NAND gadgets, named by analogy with 2-toggles [4]: one ***crossing*** type (where the two tunnels cross); and two noncrossing types, ***parallel*** (where the directions are the same) and ***antiparallel*** (where the directions are opposite). The notion of NAND gadgets was introduced in [3], which proved NP-hardness using a combination of parallel and antiparallel NAND gadgets, "one-way" gadgets, "fork" gadgets, and "XOR" gadgets. We prove that NAND gadgets alone suffice:

▶ **Lemma 3.10.** *1-player planar motion planning is NP-hard with either antiparallel NAND gadgets or crossing NAND gadgets.*

Proof. Figures 21 and 22 show that antiparallel NAND gadgets can simulate crossing NAND gadgets and vice versa. Figure 23 shows how crossing NAND gadgets can simulate parallel NAND gadgets. Therefore we can assume the availability of all three planar types of NAND gadgets.

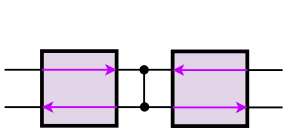

Figure 21 Simulation of crossing NAND gadget by antiparallel NAND gadgets.

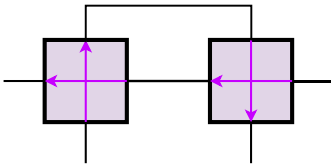

Figure 22 Simulation of antiparallel NAND gadget by crossing NAND gadgets.

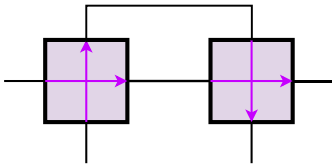

Figure 23 Simulation of parallel NAND gadget by crossing NAND gadgets.

We follow the NP-hardness reduction from Planar 3-Coloring to Push-1-X in [3]. This reduction requires four types of gadgets. Their "NAND gadget" is our parallel and antiparallel (noncrossing) NAND gadgets, which we have. Their "XOR-crossing gadget" is a crossing 2-tunnel gadget that breaks down (in a particular way) if both tunnels get traversed. The reduction guarantees that at most one tunnel in an XOR-crossing gadget will be traversed (because they correspond to different color assignments), so we can replace this gadget with a crossing NAND gadget (which even prevents both tunnels from being traversed). Their "fork gadget" is a one-entrance two-exit gadget such that either traversal closes the other traversal; we can simulate this gadget with a parallel NAND gadget by connecting together the two entrances. Their "one-way gadget" is a gadget that prevents traversal in one direction, but provides no constraint after being traversed in the other direction. Because this gadget is required only to block certain traversals, and each gadget gets visited only once (in particular because the reduction is to Push-1-X where the robot is not permitted to revisit a square), we can replace this gadget with a NAND gadget where one tunnel is not connected to anything. Therefore we have established NP-hardness using only NAND gadgets. ◀

▶ **Theorem 3.11.** *1-player planar motion planning with the door in Case 8: OTtocC is NP-hard.*

Proof. We show how to simulate antiparallel NAND gadgets, which is NP-hard by Lemma 3.10. First, Figure 24 shows how to combine two Case 8: OTtocC doors to build a door-like gadget with an open tunnel and two traverse–close tunnels, where traversing the open tunnel opens both traverse–close tunnels, and traversing either traverse–close tunnel closes the other traverse–close tunnel. Next, Figure 25 shows how to combine two of these gadgets to build an antiparallel NAND gadget. The top tunnel in the top gadget is initially closed, forcing the agent to open it and thus close the bottom tunnel of the bottom gadget, which is possibly only if the bottom tunnel of the bottom gadget was not already traversed. Because the open tunnel of the bottom gadget is not connected to anything, both tunnels of the bottom gadget will remain closed once closed. ◀

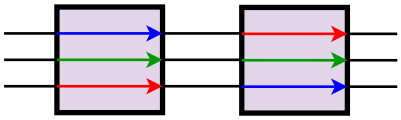

Figure 24 Simulation of parallel double-close door with the Case 8: OTtocC door.

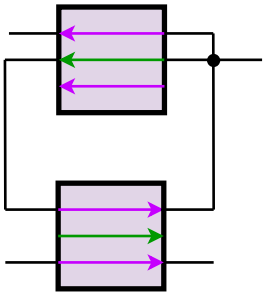

Figure 25 Simulation of an antiparallel NAND gadget with a parallel double-close door.

4 Applications

In this section we use our results about the complexity of door gadgets to prove PSPACE-hardness for seven new video games: Sokobond, and several different 3D Mario games. More applications are in the full paper.

Sokobond is a 2D block pushing game where the blocks are able to fuse into polyominoes. The Mario games considered are all 3D platformers in which the player controls Mario in an attempt to collect resources or reach target locations while avoiding or defeating enemies and environmental hazards. The player's main actions are having Mario jump and walk in an approximately continuous environment. Mario also has health and ways of taking damage which can cause the player to lose the game. More details on the needed additional mechanics are given in the section for each game. Captain Toad: Treasure Tracker is a 3D puzzle platformer and is mechanically similar to Mario except that Toad is unable to jump.

In addition, our planar door results simplify prior uses of a door framework. The Lemmings door [11, Figure 4] has an internal crossing, so Theorem 3.5 applies. The Donkey Kong Country 1, 2, and 3 doors [1, Figures 21–23] are the Case 10: OTcCt door, Case 4: OTtcC door, and internal crossing door, respectively, so Theorems 3.8 and 3.5 applies. The Legend of Zelda: A Link to the Past door [1, Figure 30] has an internal crossing, so Theorem 3.5 applies. The Super Mario Bros. door [6, Figure 6] is is the Case 4: OTtcC door, so Theorem 3.8 applies. Therefore all of the crossover gadgets in these reductions [11, Figure 2(e)], [1, Figure 20], [1, Figure 28], [6, Figure 5] are not in fact needed to prove PSPACE-hardness of these games.

4.1 Sokobond

Sokobond [8] is a 2D block pushing game where the blocks are atoms/molecules. Movement is discrete along a square grid. The player starts as a single atom. Each atom except He has some number of free electrons (H has 1, O has 2, N has 3, C has 4). When two atoms that both have free electrons are adjacent, they both lose a free electron and bond into a molecule. Molecules are rigid, so pushing an atom in a molecule results in the entire molecule moving. Atoms/molecules can also push each other.

Sokobond with He atoms is trivially NP-hard as it includes PUSH-$*$ [3]. We show PSPACE-hardness even without He atoms:

▶ **Theorem 4.1.** *Completing a level in Sokobond with H and O atoms is PSPACE-hard.*

Proof. We reduce from 1-player planar motion planning with a door that is not the Case 8: OTtocC door and use Theorem 3.9.

Let the player start as an H atom trying to reach another H atom. We can simulate a door that is not the Case 8: OTtocC door as shown in Figure 26. To open the door, the player pulls down on the big molecule. The player can go through the traverse tunnel if and only if the molecule is down. When going through the closing tunnel, the player is forced to push up on the molecule, closing the traverse tunnel. The molecule used to simulate a door has no free electrons, so the level can be completed if and only if the player can reach the other H atom. ◀

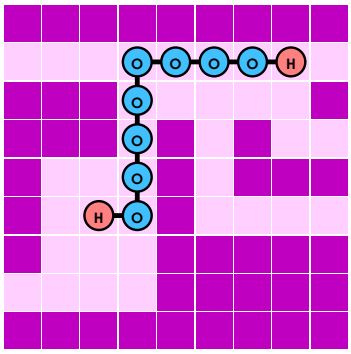

Figure 26 Simulation of a door in Sokobond. The opening port is at the bottom left. The traverse tunnel is undirected and runs between the top left and the top right. The closing tunnel is undirected and runs between the middle right and the bottom right.

4.2 Captain Toad: Treasure Tracker

Captain Toad: Treasure Tracker is a 3D puzzle platformer in the Mario universe, originally appearing as a type of level in Super Mario 3D World, and then released as a stand-alone game on the Wii U and ported to the 3DS and Switch. Notably, Toad can fall but not jump. The game contains rotating platforms controlled by a wheel which Toad must be adjacent to to move. The platforms move in 90° increments. We show PSPACE-hardness by constructing an antiparallel symmetric self-closing door (Theorem 2.4).

▶ **Theorem 4.2.** *Collecting Stars in a Captain Toad: Treasure Tracker is PSPACE-hard assuming no level size limit.*

Proof. Figure 27 gives a top-down view of the construction. There is a U-shaped rotating platform at a height slightly below the high ground and far above the low ground. The U-shaped platform rotates counterclockwise and can be reached from the nearby high ground; however, the gap between the back of the U and the other side is too far for Toad to jump. Further, the dividing wall sits slightly above the rotating platform, preventing Toad from crossing. Toad is able to go onto the U platform from the high ground, activate the gear twice, and jump off of the U platform onto the low ground across the gap. The U platform is now facing the other way, allowing Toad to enter from the high ground on the other side, but preventing other traversals. ◀

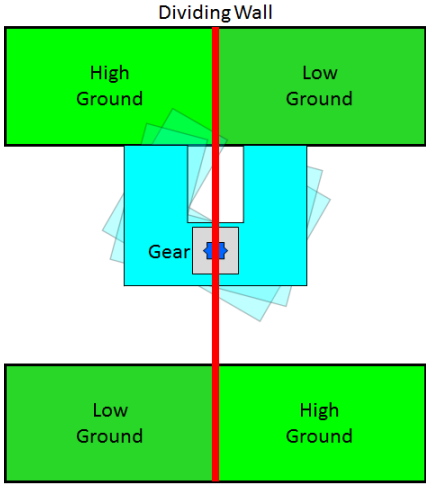

Figure 27 Top view of a simulation of a symmetric self-closing door.

4.3 Super Mario 64/Super Mario 64 DS

Super Mario 64 is a 3D Mario game for the Nintendo 64 where Mario collects Stars from courses inside paintings to save the princess, who is trapped behind a painting. Super Mario 64 DS is a remake of Super Mario 64 for the Nintendo DS (still in 3D), featuring the same courses as in Super Mario 64 plus new courses, as well as the ability to play as characters other than Mario. In this reduction, we will primarily make use of quicksand, which will defeat Mario if he lands in it, and the ghost enemy Boo.

The Boo is an enemy that (with normal parameters) chases Mario if he is looking away from it and is less than a certain distance away. Once Mario gets too far, the Boo moves back to its original position. Unlike most enemies, jumping on a Boo does not kill it, but instead sends it a short distance forward or backward, which we will use to help Mario cross the quicksand. Some walls stop the Boo but it can go through certain walls that normal Mario cannot go through, we call these Boo-only walls. The Boo is also unable to go through doors. We also make use of one-way walls which Mario and the Boo can go through in one direction but not the other.

For the setup, we use one Boo in Super Mario 64 DS and two Boos in Super Mario 64. Performing a kick while in the air sends Mario a short distance up and can normally only be performed once per jump. But Mario can kick after jumping on a Boo in Super Mario 64 DS even if he already kicked, allowing him to jump on the same Boo. This is not true in Super Mario 64, so jumping on a second Boo is necessary to stall long enough to jump on the first Boo again.

▶ **Theorem 4.3.** *Collecting a Star in a Super Mario 64/Super Mario 64 DS course is PSPACE-hard assuming no course size limits.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 2.4), where the target to reach is a Star. The simulation is shown in Figure 28.

In the setup below, Mario goes from port 1 to port 2 and opens the port 3 to port 4 traversal by going through the door on the bottom-left and hopping on the Boo(s) to the top-left. Then Mario lets the Boo(s) chase him a little to turn the Boo(s), and hops on the Boo(s) to push it into the top-right. Finally, Mario goes through the top-left door. Mario cannot just jump to the other side because the distance is too far. He also cannot go into the traverse path because of the Boo-only wall. The Boo(s) will try to go back to its home, but cannot because it is stuck behind a 1-way wall and a regular wall. If Mario does not move the Boo(s) to the top-right, it still cannot get back to its home because of a different 1-way wall, so Mario cannot leave the port 1 to port 2 traversal open.

Mario goes from port 3 to port 4 by going through the top-right door and hopping on the Boo(s) to the bottom-right, then going through the bottom-right door. The Boo(s) will go back to its original position at the bottom left on its own.

Mario cannot lure the Boo(s) away from the gadget because it is completely walled in except for the doors, which the Boo(s) cannot go through. ◀

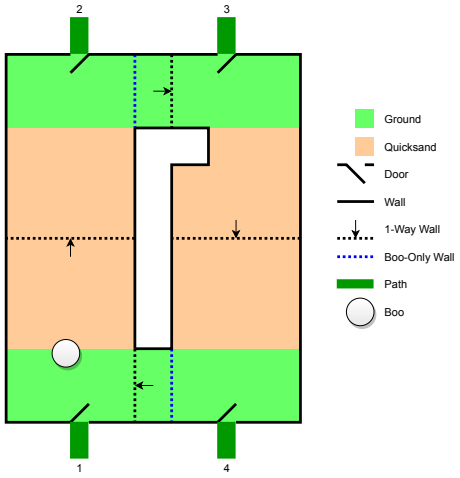

Figure 28 Simulation of a symmetric self-closing door in Super Mario 64 DS. In Super Mario 64, there are 2 Boos instead of 1. The ground and quicksand are on the same vertical level. The room is covered by a ceiling. The hallways are too wide to wall jump across.

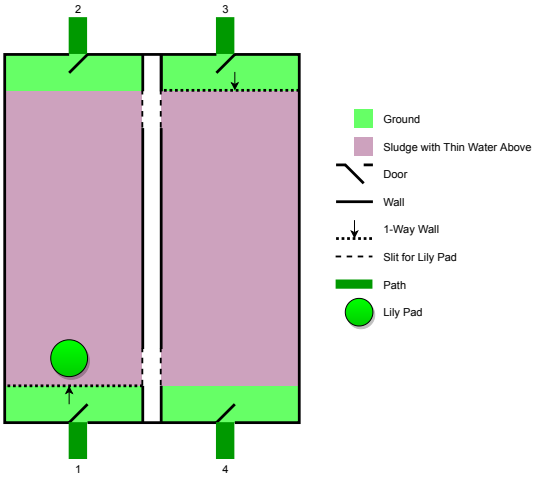

Figure 29 Simulation of a symmetric self-closing door in Super Mario Sunshine. The slits allow the Lily Pad to cross without allowing bulky Mario to do so. The hallways are too wide to wall jump across.

4.4 Super Mario Sunshine

Super Mario Sunshine is a 3D Mario game for the GameCube where Mario is falsely accused of spreading graffiti and is forced to clean it up before he can leave. Like Super Mario 64, this game includes one-way walls. This game features a new device, F.L.U.D.D., attached to Mario's back that allows him to spray water. Lily Pads float on water; the player can ride a Lily Pad and cause it to move by spraying water in the opposite direction. Sludge is an environmental hazard which kills Mario if he touches it. The general goal of a level is to collect Shrine Sprites.

▶ **Theorem 4.4.** *Collecting a Shine Sprite in a Super Mario Sunshine level is PSPACE-hard assuming no level size limits.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 2.4), where the target to collect is a Shine Sprite. The simulation of a symmetric self-closing door is shown in Figure 29.

The thin water above the sludge prevents the Lily Pad from disintegrating, while preventing Mario from crossing without using the Lily Pad. Mario goes from port 1 to port 2 and opens the port 3 to port 4 traversal by crossing the 1-way wall and riding the Lily Pad across, then moves the Lily Pad partially across the slit so it can be accessed from the other side. He cannot leak to the section between port 3 and port 4 because the slits are too thin. The sludge is too long to simply jump to the other side, so the Lily Pad is needed. Mario cannot do anything from port 2 because the 1-way wall blocks him from going to port 1. Mario goes from port 3 to port 4 in a similar manner. ◀

4.5 Super Mario Galaxy

Super Mario Galaxy is a 3D Mario game for the Wii where Mario goes to space. He encounters alien creatures along the way and collects Power Stars to restore the power of a spaceship. The game features downward gravity, upward gravity, sideways gravity, spherical gravity, cubical gravity, tubular gravity, cylindrical gravity that allows infinite freefall, W-shaped gravity, gravity that cannot make up its mind, and most importantly, controllable gravity.

Dark matter disintegrates Mario when he touches it, resulting in death. The Gravity Switch changes the direction of gravity when spun and can be spun multiple times.

▶ **Theorem 4.5.** *Collecting a Power Star in a Super Mario Galaxy galaxy is PSPACE-hard assuming no galaxy size limits.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 2.4), where the target to collect is a Star. The simulation of a symmetric self-closing door is shown in Figure 30.

The Gravity Switch in this construction switches gravity between down and up. Mario goes from port 1 to port 2 by crossing the 1-way wall and hitting the Gravity Switch on his way to the right. This is forced because of a pit of dark matter, and closes the port 1 to port 2 traversal because when gravity points up, attempting the traversal would land Mario on dark matter. At the same time, it opens the port 3 to port 4 traversal. Mario cannot enter port 2 and do anything useful because flipping the Gravity Switch means falling in the pit of dark matter. Mario goes from port 3 to port 4 in a similar manner. ◀

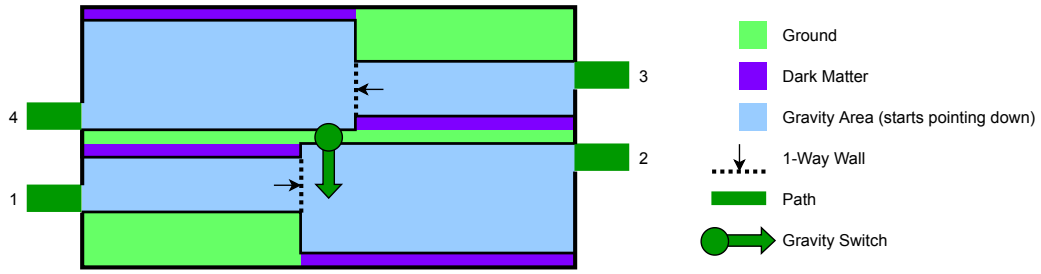

Figure 30 Simulation of a symmetric self-closing door in Super Mario Galaxy. This is a side view and is essentially 2-dimensional.

4.6 Super Mario Odyssey

Super Mario Odyssey is a 3D Mario game for the Switch where Mario travels to different kingdoms collecting Power Moons and eventually goes to the Moon. Mario has the ability (via his hat Cappy) to capture certain enemies and objects to use their powers, but such objects tend to reset position after being uncaptured, so we will not be using them here.

We make use of a Jaxi, poison, and timed platforms. A Jaxi is a statue lion that can be ridden safely across poison, which is a hazard that kills Mario. A timed switch makes some event happen for a specific amount of time. In our reduction, timed switch X makes platform X appear for just long enough for Mario to make a traversal.

▶ **Theorem 4.6.** *Collecting a Power Moon in a Super Mario Odyssey kingdom is PSPACE-hard assuming no kingdom size limit.*

Proof. We reduce from 1-player motion planning with the symmetric self-closing door (Theorem 2.4), where the target to reach is a Power Moon. The simulation of a symmetric self-closing door is shown in Figure 31.

Mario goes from port 1 to port 2 by pressing timed switch A, riding the Jaxi to the right, and traversing platform A. This opens the port 3 to port 4 traversal while closing the port 1 to port 2 traversal. Mario cannot go to port 3 because of the wide gap, or to port 4 because platform B is gone. The Jaxi is required because the poison it is on is very wide. Mario cannot do anything useful if he tries to enter from port 2 or port 4 because the platforms would be gone. Mario goes from port 3 to port 4 in a similar manner. ◀

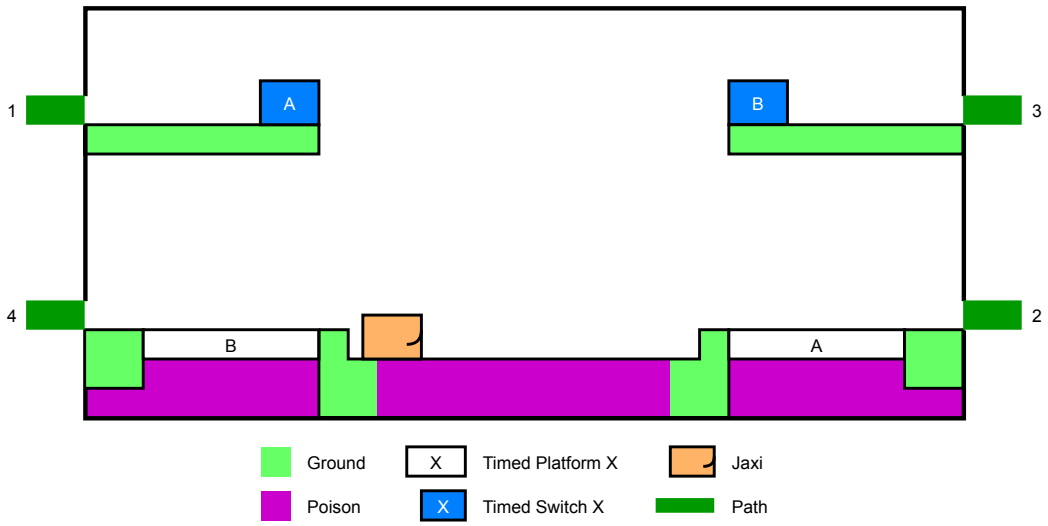

Figure 31 Simulation of a symmetric self-closing door in Super Mario Odyssey. This is a side view and is essentially 2-dimensional. All strips of poison are way too wide for Mario to cross with his various aerial skills, and the platforms with timed switches are too high to get to from below.

References

1 Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015. Originally appeared at FUN 2014.

2 Joshua Ani, Sualeh Asif, Erik D. Demaine, Yevhenii Diomidov, Dylan Hendrickson, Jayson Lynch, Sarah Scheffler, and Adam Suhl. PSPACE-completeness of pulling blocks to reach a goal. In *Abstracts from the 22nd Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2019)*, pages 31–32, Tokyo, Japan, September 2019.

3 Erik D. Demaine, Martin L. Demaine, Michael Hoffmann, and Joseph O'Rourke. Pushing blocks is hard. *Computational Geometry: Theory and Applications*, 26(1):21–36, August 2003.

4 Erik D. Demaine, Isaac Grosof, Jayson Lynch, and Mikhail Rudoy. Computational complexity of motion planning of a robot through simple gadgets. In *Proceedings of the 9th International Conference on Fun with Algorithms (FUN 2018)*, pages 18:1–18:21, La Maddalena, Italy, June 2018.

5 Erik D. Demaine, Dylan H. Hendrickson, and Jayson Lynch. Toward a general complexity theory of motion planning: Characterizing which gadgets make games hard. In *Proceedings of the 11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, pages 62:1–62:42, Seattle, January 2020. doi:10.4230/LIPIcs.ITCS.2020.62.

6 Erik D. Demaine, Giovanni Viglietta, and Aaron Williams. Super Mario Bros. is harder/easier than we thought. In *Proceedings of the 8th International Conference on Fun with Algorithms (FUN 2016)*, pages 13:1–13:14, La Maddalena, Italy, June 2016.

7 Michal Forišek. Computational complexity of two-dimensional platform games. In *Proceedings of the 5th International Conference on Fun with Algorithms (FUN 2010)*, volume 6099 of *Lecture Notes in Computer Science*, 2010. doi:10.1007/978-3-642-13122-6_22.

8 Alan Hazelden, Lee Shang Lun, and Allison Walker. Sokobond. https://www.sokobond.com/, 2014.

9 Tom C. van der Zanden and Hand L. Bodlaender. PSPACE-completeness of Bloxorz and of games with 2-buttons. arXiv:1411.5951, 2014. URL: https://arXiv.org/abs/1411.5951.

10 Giovanni Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54(4):595–621, 2014. Originally appeared at FUN 2012. doi:10.1007/s00224-013-9497-5.

11 Giovanni Viglietta. Lemmings is PSPACE-complete. *Theoretical Computer Science*, 586:120–134, 2015. Originally appeared at FUN 2014. doi:10.1016/j.tcs.2015.01.055.