You Can't Solve These Super Mario Bros. Levels: Undecidable Mario Games

MIT Hardness Group[1]
CSAIL, Massachusetts Institute of Technology, 32 Vassar St., Cambridge, MA 02139, USA

Hayashi Ani
CSAIL, Massachusetts Institute of Technology, 32 Vassar St., Cambridge, MA 02139, USA

Erik D. Demaine
CSAIL, Massachusetts Institute of Technology, 32 Vassar St., Cambridge, MA 02139, USA

Holden Hall
CSAIL, Massachusetts Institute of Technology, 32 Vassar St., Cambridge, MA 02139, USA

Ricardo Ruiz
CSAIL, Massachusetts Institute of Technology, 32 Vassar St., Cambridge, MA 02139, USA

Naveen Venkat
CSAIL, Massachusetts Institute of Technology, 32 Vassar St., Cambridge, MA 02139, USA

Abstract

We prove RE-completeness (and thus undecidability) of several 2D games in the Super Mario Bros. platform video game series: the New Super Mario Bros. series (original, Wii, U, and 2), and both Super Mario Maker games in all five game styles (Super Mario Bros. 1 and 3, Super Mario World, New Super Mario Bros. U, and Super Mario 3D World). These results hold even when we restrict to constant-size levels and screens, but they do require generalizing to allow arbitrarily many enemies at each location and onscreen, as well as allowing for exponentially large (or no) timer.

In our Super Mario Maker reductions, we work within the standard screen size and use the property that the game engine remembers offscreen objects that are global because they are supported by "global ground". To prove these Mario results, we build a new theory of counter gadgets in the motion-planning-through-gadgets framework, and provide a suite of simple gadgets for which reachability is RE-complete.

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

Keywords and phrases video games, computational complexity, undecidability

Digital Object Identifier 10.4230/LIPIcs.FUN.2024.29

Category FUN with Algorithms

Acknowledgements This paper was initiated during open problem solving in the MIT class on Algorithmic Lower Bounds: Fun with Hardness Proofs (6.5440) taught by Erik Demaine in Fall 2023. We thank the other participants of that class for helpful discussions and providing an inspiring atmosphere. Portions of this paper originally appeared in Ani's master's thesis [2]. Several community level editors and emulators were very helpful in building and testing counters:

- *Reggie!* by the NSMBW Community — https://github.com/NSMBW-Community/Reggie-Updated
- *CoinKiller* by Arisotura — https://github.com/Arisotura/CoinKiller
- *Miyamoto* by aboood40091 — https://github.com/aboood40091/Miyamoto
- *Dolphin* by the Dolphin Emulator Project — https://dolphin-emu.org/
- *Cemu* by Team Cemu — https://cemu.info/
- *Citra* by Citra Emu — https://github.com/citra-emu/citra

[1] Artificial first author to highlight that the other authors (in alphabetical order) worked as an equal group. Please include all authors (including this one) in your bibliography, and refer to the authors as "MIT Hardness Group" (without "et al.").

© Hayashi Ani, Erik D. Demaine, Holden Hall, Ricardo Ruiz, Naveen Venkat;
licensed under Creative Commons License CC-BY 4.0

1 Introduction

In 2014, Hamilton [10] proved that solving a (bounded-size) level in the 2008 video game *Braid* is ***RE-complete*** (the same difficulty as the halting problem, and thus undecidable), even without its famous time-travel mechanic. The reduction is from ***counter machines*** [15, 16, 13]: a finite-state machine with a constant number of natural-number counters equipped with increment, decrement, and jump-if-zero instructions. The central idea was to represent the value of each counter in a Braid level by the number of enemies occupying a particular location in the level, exploiting that this number can be arbitrarily large even in a bounded-size level. The same paper conjectured that many other video games were amenable to this approach.

In this paper, we prove that this approach extends to several 2D games in the Super Mario Bros. platform video game series,[2] when we generalize them to remove any limits on time, the total number of enemies, or the number of enemies at each location. Most of these games (even the original Super Mario Bros. from 1985) have mechanics for spawning arbitrarily many enemies over time, but it is difficult to find the right combination of mechanics to implement the specific functionality of an increment/decrement/jump-if-zero counter. Specifically, we show that the following games are RE-complete by building computer machines where the number of enemies in a particular location represents the value of each counter:

1. The *New Super Mario Bros.* series (Section 3): New Super Mario Bros., New Super Mario Bros. Wii, New Super Mario Bros. U, and New Super Mario Bros. 2. One reduction covers all four games, using their powerful "event" game mechanic, where Mario's location can toggle the existence of blocks. We build an entire universal counter machine within a single screen of the the Wii version, so solving even a single-screen level is RE-complete.
2. The *Super Mario Maker* series (Section 4): Super Mario Maker 1 in all four game styles (Super Mario Bros. 1, Super Mario Bros. 3, Super Mario World, and New Super Mario Bros. U) and Super Mario Maker 2 in all five game styles (the same four, plus Super Mario 3D World).

All of these games are in RE: if a level is solvable, then there is a finite algorithm to solve them, by trying all possible sequences of inputs to the game, and simulating the result. Because any solvable level is (by definition) solvable in finite time, and the state in the game after any finite time is finite, this algorithm is finite. Thus, to prove RE-completeness, it remains to prove RE-hardness.

To simplify the process of proving such games RE-hard using counter machines, we develop in Section 2 a new theory of "counter gadgets" which shows that a *single* gadget diagram suffices to prove RE-hardness in a one-player game like Super Mario Bros. This framework builds on the ***motion-planning-through-gadgets framework*** developed in recent years [8, 3, 6, 5, 4, 5, 14, 11], starting with FUN 2018 [7]. In the single-player version of this framework, one agent (Mario) traverses an environment consisting of local "gadgets", with the locations of the gadgets connected together in a graph (representing freely traversable connections). Each ***gadget*** has a finite set of states and a finite set of possible transitions, where a transition $(q, a) \to (r, b)$ means that, when the gadget is in state q, the agent can enter at location a and leave at location b, while changing the state of the gadget to r. We generalize this framework to allow for *infinite-state* gadgets called ***counter gadgets***, where the states are the natural numbers and the traversals are among common operations such

[2] After all, "Braid is a postmodern Super Mario Bros." [12].

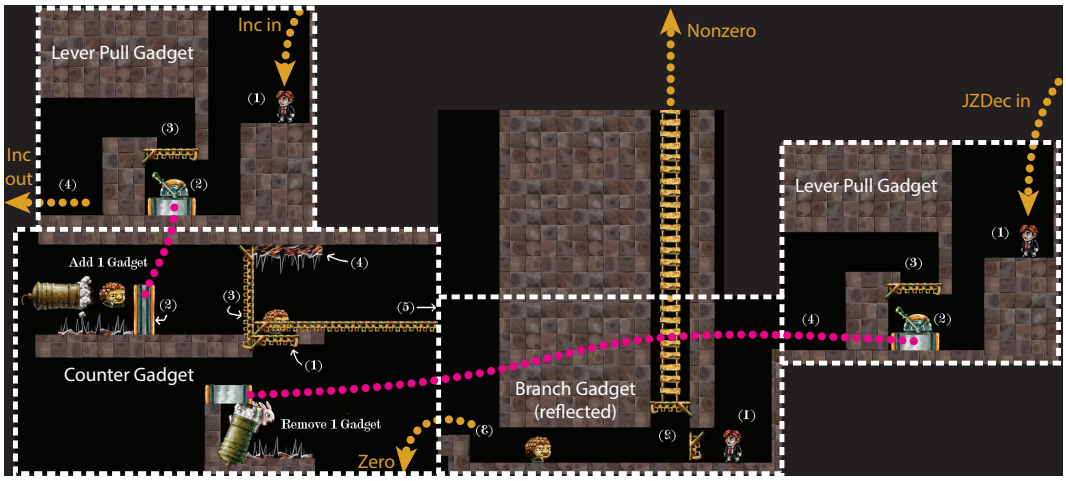


Figure 1 An Inc-JZDec counter gadget in Braid, built from the Level Pull, Counter, and Branch gadgets of [10].

as increment, decrement, traversable-if-zero, and traversable-if-nonzero. Specifically, we show that any one of the following counter gadgets suffices to prove RE-completeness of the ***reachability*** problem (can the agent get from one location to another?):

1. **Inc-Dec-JZ:** One traversal path that increments the counter value, another that decrements the counter value unless it is already zero, and a third that leads the agent to two different locations depending on whether the counter value is zero.
2. **Inc-JZDec:** One traversal path that increments the counter value, and another that leads the agent to two different locations depending on whether the counter value is zero, and if it is not, decrements the counter.
3. **Inc-DecNZ-PZ:** One traversal path that increments the counter value, another that decrements the counter value but which can be traversed only if the counter is nonzero, and a third that is traversable only if the counter value is zero.
   Notably, this RE-hardness result holds even when the connections between gadgets form a planar graph, so there is no need for a crossover gadget.
4. **Inc$[a, b]$-DecNZ$[c, d]$-PZ:** A generalization of the previous gadget where, when the agent traverses an increment or decrement path, it gets to choose how much to increment or decrement, within a range of $[a, b]$ or $[c, d]$, respectively, where $a, c > 0$. This robustness to unknown gadget behavior is helpful for building gadgets in video games, which can require very careful timing and alignment to force a specific number of increments or decrements, but forcing at least one and at most some constant is relatively easy.

Specifically, we build an Inc$[1, 2]$-DecNZ$[1, 2]$-PZ gadget in the New Super Mario Bros. series, and we build three different Inc-DecNZ-PZ gadgets for different variants of the Super Mario Maker series. Some of our gadgets are available for download and play [17]. On the plus side, we need to build only one main gadget for each game, together with a crossover gadget in some cases (such as New Super Mario Bros.). On the negative side, as we will see, that one main gadget is quite complex.

By contrast, the Braid proof [10] had six individual gadgets. We can instead combine three of them (Lever Pull, Counter, and Branch) in a straightforward way to build an Inc-JZDec gadget; see Figure 1. Then we only need a crossover gadget for the agent (Tim), which is one of the three crossover gadgets in [10]; the other two are no longer necessary.

Like Hamilton, we conjecture that our counter-gadget framework can be applied to prove RE-hardness of other video games as well. We discuss further possibilities for Super Mario Bros. games in Section 5.

2 Counter Gadgets

In this paper, we reduce from "reachability with gadgets", first explored in [7, 8]. We define a ***gadget*** $G = (Q, L, T)$ to consist of a set Q of ***states*** (not necessarily finite), a finite set L of ***locations***, and a set $T \subseteq (Q \times L)^2$ of allowed ***transitions*** on pairs of locations and states, each written in the form $(q, a) \to (r, b)$ where $q, r \in Q$ and $a, b \in L$.

An example of a gadget is the ***1-toggle***: it has a single path that the player can cross in only one direction, and every time they do, the allowed direction flips. Figure 2 gives a graphical representation. In this case, there are two locations $L = \{a, b\}$, two states $Q = \{R, L\}$, and $T = \{(0, a) \to (1, b), (1, b) \to (0, a)\}$.

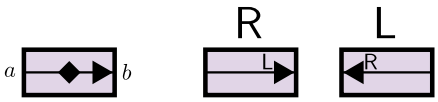


Figure 2 A 1-toggle, along with its state diagram on the right. When the player goes from a to b, the arrow flips.

Here, we consider gadgets with an infinite number of states, namely, one for each natural number. We further restrict a ***counter gadget*** to consist of ***counter components*** (see Figure 3) that interact with each other when put in the same gadget:

- **Inc[a, b]**. This is a directed tunnel that is always traversable. When the player traverses it, they choose a natural number i such that $a \leq i \leq b$. The gadget's state increments by i.
- **DecNZ[a, b]**. This is a directed tunnel that is traversable if and only if the gadget's state is at least a. When the player traverses it, they choose a natural number i such that $a \leq i \leq \min(s, b)$, where s is the gadget's state. The gadget's state decrements by i.
- **Dec[a, b]**. This is like DecNZ$[a, b]$, except that it is always traversable, and if the gadget's state would become negative, it instead becomes 0.
- **Inc**, **DecNZ**, **Dec**. These are short for Inc$[1, 1]$, DecNZ$[1, 1]$, and Dec$[1, 1]$, respectively.
- **PZ**. This is a directed tunnel that is traversable if and only if the gadget's state is 0. It does not change the state.
- **PNZ**. This is a directed tunnel that is traversable if and only if the gadget's state is not 0. It does not change the state. This is only defined for convenience of defining the JZ switch below.
- **JZ**. This is a switch formed by putting a PZ tunnel and a PNZ tunnel in the same gadget and combining their entrances.
- **JZDec**. This is a switch formed by putting a PZ tunnel and a DecNZ tunnel in the same gadget and combining their entrances.

2.1 Undecidable Gadgets via Counter Machines

A ***system*** of gadgets consists of instances of gadgets, an initial state for each gadget, and a graph connecting the gadgets' locations together. In the ***reachability*** problem, we are given a system of gadgets, a start location s, and a goal location t, and we want to know whether

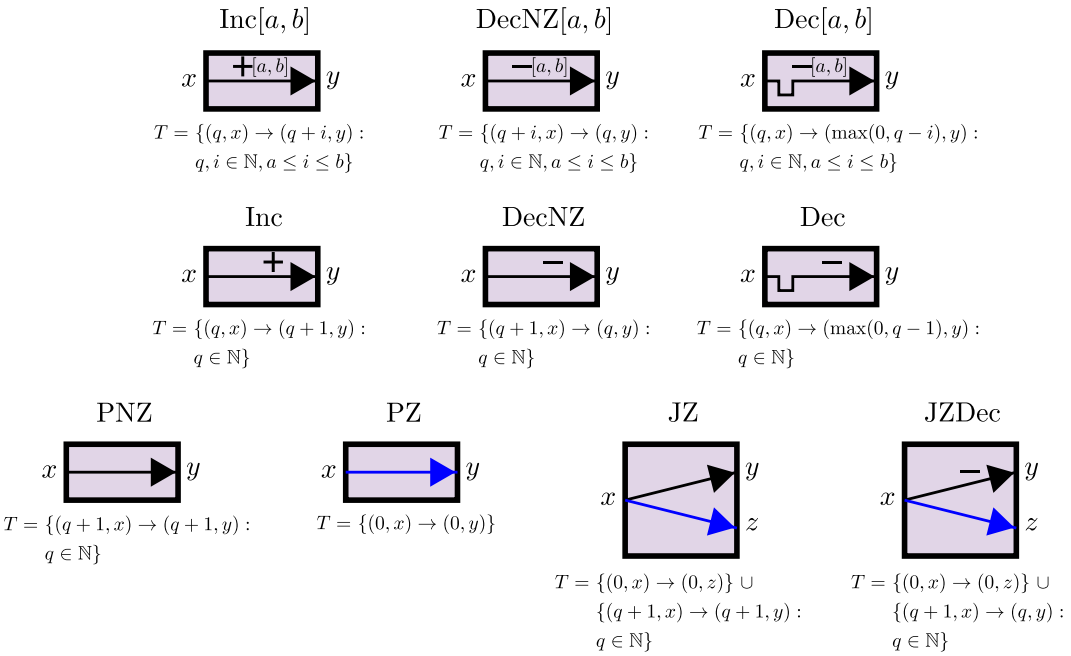


Figure 3 Counter components that are allowed in counter gadgets, along with their sets T of allowed traversals.

the player can get from s to t by making a sequence of gadget traversals and following edges of the connection graph.

With infinite-state gadgets, sometimes reachability is undecidable, since we can simulate counter machines. We use the fact that the counter-machine halting problem is RE-hard [15] as a starting point. First, we give a brief introduction to counter machines.

A ***counter machine*** is a set of counters and a sequence of instructions that run on those counters. The instructions are:

- `Inc(`c`)`: Add 1 to counter c and move on to the next instruction
- `Dec(`c`)`: Subtract 1 from counter c, leaving its value alone if it was 0, and move on to the next instruction.
- `JZ(`c`,`i`)`: Check if counter c is 0. If so, jump to instruction i. Otherwise, move on to the next instruction.
- `Halt`: Stop the machine.

It is RE-hard to determine whether a counter machine reaches a `Halt` instruction, even with just two counters [15], [16, pp. 255–258].

2.2 Inc-Dec-JZ is RE-complete

The Inc-Dec-JZ gadget (Figure 4) consists of, as the name indicates, an Inc tunnel, a Dec tunnel, and a JZ switch.

We prove that reachability with this gadget is NP-hard. The proof involves duplicating the Inc and Dec tunnels a bunch, then duplicating the JZ switch a bunch, then simulating a counter machine using multiple of the gadget, using each set of Inc, Dec, and JZ components as an instruction. We will show RE-hardness again, but with a different method where we build a gadget made for flow control, and use multiple copies of that gadget along with unaltered Inc-Dec-JZ gadgets.

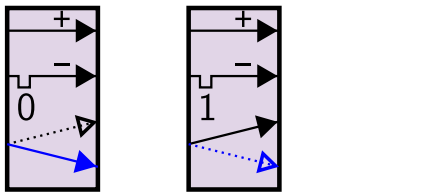


Figure 4 The Inc-Dec-JZ gadget, shown in states 0 and 1.

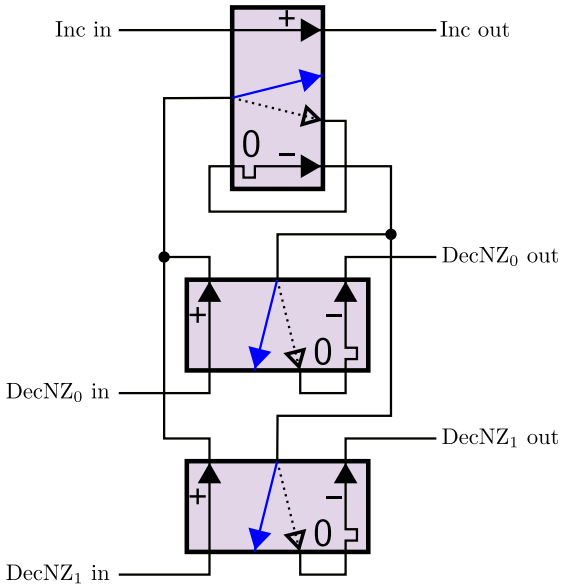


Figure 5 The Inc-DecNZ-DecNZ gadget, along with its simulation by Inc-Dec-JZ gadgets.

▶ **Theorem 1.** *Reachability with Inc-Dec-JZ is RE-complete.*

Proof. We show this by simulating a counter machine with `Inc(`c`)`, `Dec(`c`)`, and `JZ(`c`,`i`)` instructions, which increment counter c, decrement counter c (saturated at 0), and jump to instruction i if $c = 0$, respectively.

First, we build an ***Inc-DecNZ-DecNZ*** gadget to help with flow control. We simulate one as shown in Figure 5. If the agent enters via Inc in, they increment the top gadget and leave. If the agent enters via DecNZ$_i$ in, they eventually get stuck if the top counter is 0. Otherwise, they increment the middle/bottom gadget, decrement the top gadget, and decrement the gadget that they incremented before, leaving via DecNZ$_i$ out.

To construct the counter machine, we use an Inc-Dec-JZ gadget c for each counter c, and an Inc-DecNZ-DecNZ gadget i for each instruction $i\colon \cdots$. At a high level, we connect gadgets so that the agent flow works as follows:

- For an instruction i`:Inc(`c`)`, the agent increments instruction gadget i, increments the counter gadget c, finds the incremented instruction gadget i and decrements it, and then moves on to the next instruction gadget $i + 1$.
- For an instruction i`:Dec(`c`)`, the agent does the same as above, except that they decrement the counter gadget c.
- For an instruction i`:JZ(`c`,`i'`)`, the agent increments instruction gadget i and goes to check whether counter gadget c is in state 0. If it is, they decrement gadget i and branch to instruction gadget i'. Otherwise, they decrement gadget i using the other decrement path and move on to the next instruction gadget $i + 1$.

Figure 6 shows an example.

More concretely, we connect gadgets in the following ways, where $a[b]$ denotes location b of gadget a, I denotes the counter machine program, $I[i]$ denotes the ith instruction, and C denotes the set of counter gadgets:

- i[Inc out] $\sim$ c[Inc in] for all i, c where $I[i] =$ `Inc(`c`)`.
- i[Inc out] $\sim$ c[Dec in] for all i, c where $I[i] =$ `Dec(`c`)`.

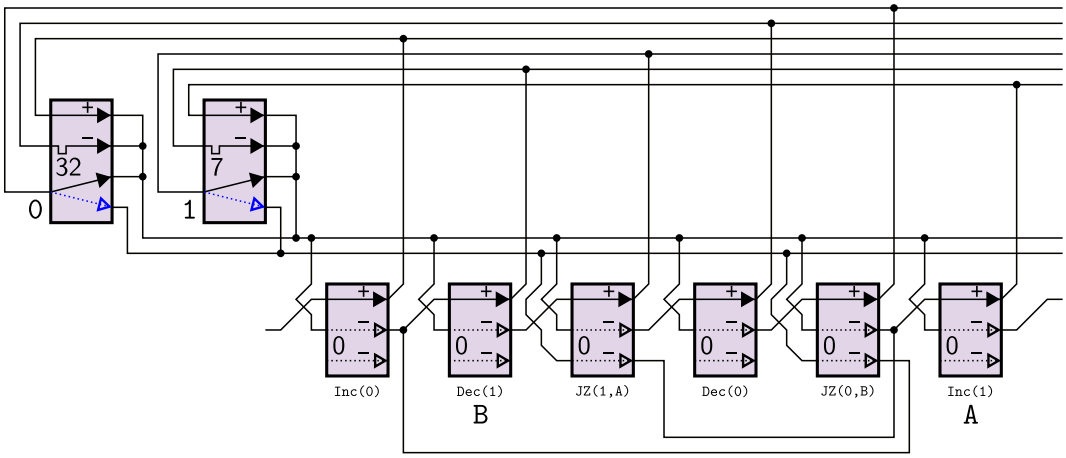


Figure 6 The Inc-Dec-JZ gadget simulating a counter machine.

- i[Inc out] $\sim$ c[JZ in] for all i, c, i' where $I[i] =$ `JZ(`c`,`i'`)`.
- c[Inc out] $\sim$ i[DecNZ$_0$ in] for all i, c where $0 \leq i < |I|$ and $c \in C$.
- c[Dec out] $\sim$ i[DecNZ$_0$ in] for all i, c where $0 \leq i < |I|$ and $c \in C$.
- c[JZ out $(\neq 0)$] $\sim$ i[DecNZ$_0$ in] for all i, c where $0 \leq i < |I|$ and $c \in C$.
- c[JZ out $(= 0)$] $\sim$ i[DecNZ$_1$ in] for all i, c where $0 \leq i < |I|$ and $c \in C$.
- i[DecNZ$_0$ out] $\sim$ $(i+1)$[Inc in] for all i where $0 \leq i < |I| - 1$.
- i[DecNZ$_1$ out] $\sim$ i'[Inc in] for all i, c, i' where $I[i] =$ `JZ(`c`,`i'`)`.

The agent starts just in front of the first instruction gadget, and any instruction gadget that corresponds to a `Halt` instruction is replaced with a goal. Then the agent can win if and only if the counter machine halts, reducing the counter machine halting problem to reachability. ◀

2.3 Inc-JZDec is RE-complete

The Inc-JZDec gadget (Figure 7) replaces the Dec and JZ components with a single JZDec switch. We prove that reachability with this gadget is RE-hard by simulating the Inc-Dec-JZ gadget.

▶ **Theorem 2.** *Reachability with Inc-JZDec is RE-complete.*

Proof. We reduce from reachability with Inc-Dec-JZ by simulating the Inc-Dec-JZ gadget, as shown in Figure 8. We use G_0 and G_1 to store the counter's value. If the agent enters via Inc in, they increment G_0 and G_1 and leave via Inc out. If the agent enters via Dec in, then if the counter is 0, they nearly immediately leave via Dec out. Otherwise, they decrement G_0, increment H_2, decrement G_1, decrement H_2, and leave via Dec out. Note that H_2 has no net change. If the agent enters via JZ in, if the counter is 0, they nearly immediately leave via JZ out $= 0$. Otherwise, they decrement G_1, increment H_1, increment G_1, decrement H_1, and leave via JZ out $\neq 0$, leaving no net change (except in H_0, but that gadget is just used as a diode). So this simulation works. ◀

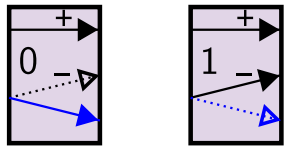


Figure 7 The Inc-JZDec gadget, shown in states 0 and 1.

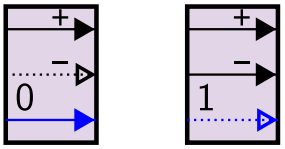


Figure 9 The Inc-DecNZ-PZ gadget, shown in states 0 and 1.

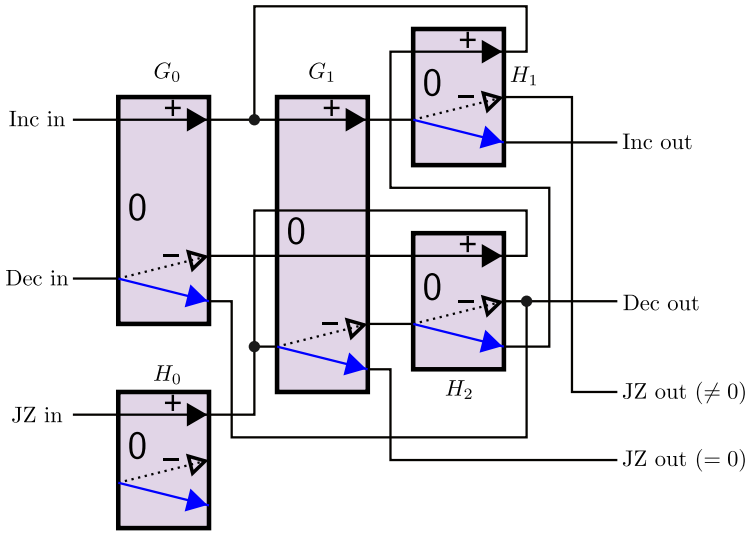


Figure 8 Simulation of Inc-Dec-JZ with Inc-JZDec. Gadgets G_0 and G_1 store the counter's value, while H_0, H_1 and H_2 are used for flow control.

2.4 Inc-DecNZ-PZ is RE-complete

The Inc-DecNZ-PZ gadget (Figure 9) replaces the JZDec switch with separate DecNZ and PZ tunnels. This gadget can easily simulate the Inc-JZDec gadget, by combining the entrances of DecNZ and PZ.

▶ **Corollary 3.** *Reachability with Inc-DecNZ-PZ is RE-complete.*

In addition, we have a planar result. In ***planar reachability***, we restrict to ***planar*** systems of gadgets where the graph of connections between gadgets' locations do not cross gadgets or each other (except at common endpoints).

▶ **Theorem 4.** *Planar reachability with any planar system of Inc-DecNZ-PZ gadgets is RE-complete.*

Proof. Ani et al. [3, Theorems 3.1 and 3.2] show that a crossover can be built from a ***symmetric self-closing door***: a gadget with two states $Q = \{1, 2\}$ and two possible traversal paths $L_1 \to R_1$ and $L_2 \to R_2$, where $L_1 \to R_1$ is possible only in state 1, $L_2 \to R_2$ is possible only in state 2, and every traversal switches the state. Figure 11 shows how to build a symmetric self-closing door from Inc-DecNZ (and thus Inc-DecNZ-PZ) in all cases. ◀

Thus, we do not need to build a crossover when reducing from reachability with the Inc-DecNZ-PZ gadget, even in a planar application.

2.5 Inc$[a, b]$-DecNZ$[c, d]$-PZ, $a > 0$, $c > 0$ is RE-complete

The Inc$[a, b]$-DecNZ$[c, d]$-PZ gadget (Figure 12) replaces the Inc and DecNZ tunnels with Inc$[a, b]$ and DecNZ$[c, d]$ tunnels, respectively. Recall that an Inc$[a, b]$ tunnel allows the player to choose an integer between a and b inclusive, and increment the gadget's state by

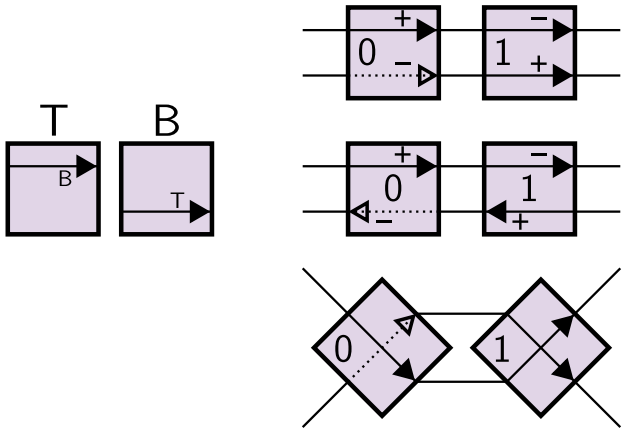


Figure 11 Left: State diagram of the symmetric self-closing door. Right: Simulation of a symmetric self-closing door, no matter the planar arrangement of tunnels in Inc-DecNZ.

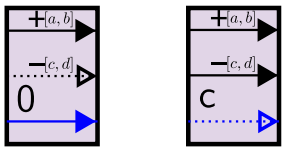


Figure 12 The Inc$[a, b]$-DecNZ$[c, d]$-PZ gadget, shown in states 0 and c.

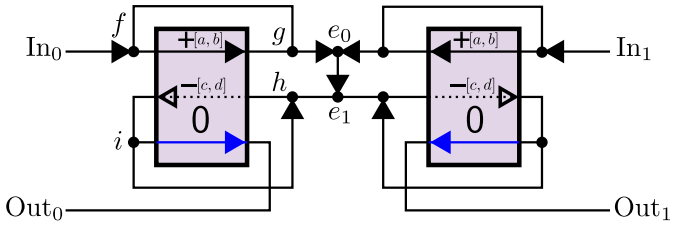


Figure 13 An edge duplicator built with the Inc$[a, b]$-DecNZ$[c, d]$-PZ gadget. The player can go from e_0 to e_1 along two different paths without leaking between them.

that amount. A DecNZ$[c, d]$ tunnel allows the player to choose an integer between c and d inclusive and decrement the state by that integer, but only if the result is nonnegative.

▶ **Theorem 5.** *Reachability with Inc$[a, b]$-DecNZ$[c, d]$-PZ is RE-hard if $a > 0$ and $c > 0$.*

Proof. We simulate the Inc-DecNZ-PZ gadget. First, we build an ***edge duplicator*** (Figure 13), which allows us to duplicate Inc$[a, b]$ and DecNZ$[c, d]$ tunnels as many times as we want. In the edge duplicator shown, $e_0 \to e_1$ is being duplicated. If the player enters via In_0, they go from f to g c times and go through e_0 to reach e_1. Then since the gadget on the right does not allow passage, they must go from h to i until the path to Out_0 opens up (that is, a times). Then they leave, and the left gadget is reset. This works symmetrically for $\text{In}_1 \to \text{Out}_1$.

Then we use two gadgets with as many Inc$[a, b]$ tunnels, as many DecNZ$[c, d]$ tunnels, and as many PZ lines as we want to simulate the Inc-DecNZ-PZ gadget (Figure 14). In fact, we use acd Inc$[a, b]$ tunnels and abd DecNZ$[c, d]$ tunnels in G_0, and bcd Inc$[a, b]$ tunnels and abc DecNZ$[c, d]$ tunnels in G_1.

Let $s(G)$ be the state of gadget G. We say that $G \sqsubset [i, j]$ if $s(G)$ is between i and j, and it is possible that $s(G) = i$, and also possible that $s(G) = j$. For example, if G starts at state 0, then $G \sqsubset [0, 0]$. If the player goes through the Inc$[a, b]$ tunnel, then $G \sqsubset [a, b]$. After going through the DecNZ$[c, d]$ tunnel, which is only possible if $b - c \geq 0$, $G \sqsubset [\max(a - d, 0), b - c]$. If $G \sqsubset [i, j]$, we define $\min(G) = i$ and $\max(G) = j$.

We maintain the invariant that $\max(G_0) = \min(G_1) = abcdn$, where n is state of the simulated Inc-DecNZ-PZ gadget.

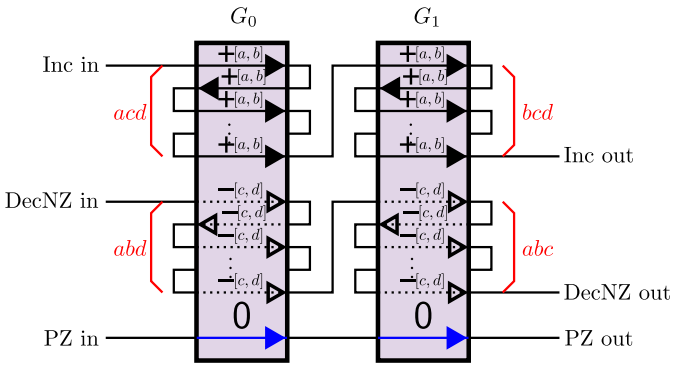


Figure 14 Simulation of Inc-DecNZ-PZ with Inc$[a, b]$-DecNZ$[c, d]$-PZ gadget. The numbers of copies of repeated tunnels are shown in red.

- If the player crosses the simulated Inc tunnel, then $\max(G_0) := \max(G_0) + b \cdot acd = abcd(n + 1)$ and $\min(G_1) := \min(G_1) + a \cdot bcd = abcd(n + 1)$.
- If the player successfully crosses the simulated DecNZ tunnel, then because of G_0, it was the case that $abcdn \geq c \cdot abd$, meaning that $n > 0$, which is what we want. Then $\max(G_0) := \max(G_0) - c \cdot abd = abcd(n-1)$ and $\min(G_1) := \min(G_1) - d \cdot abc = abcd(n-1)$. Note that since $\max(G_0) = \min(G_1)$, if G_0's portion was crossable, then so is G_1's portion.
- If the player successfully crosses the simulated PZ tunnel, then because of G_1, it was the case that $abcdn = 0$, meaning that $n = 0$. Then $\max(G_0) := 0$ and $\min(G_1) := 0$. If G_1's portion was crossable, then so is G_0's portion, because $\max(G_0) = \min(G_1)$.

So this simulation works. ◀

2.6 Constant-Size Levels

Universal Turing or counter machines let us strengthen our results to need just a constant number of gadgets. For example, Korec [13] designs a 2-counter machine U_{32} (consisting of 32 instructions over `Inc`, `Dec`, and `JZ`) that is ***strongly universal*** in the sense that there is a computable function f such that, given any 2-counter machine M, M applied to x and y produces the same result as U_{32} applies to counter values $f(x)$ and y. By applying the theorems above to U_{32}, we obtain a system of a constant number of counter gadgets that simulates U_{32} and thus any 2-counter machine and thus any Turing machine. Crucially, this system must start with arbitrary specified initial states, to represent $f(x)$ and y. This framework and its implications are described in more depth in [1].

Applied to Mario, this means that we can prove RE-hardness of constant-size levels, provided they can start with arbitrarily many enemies at each location.

Alternatively, if we must start with all counters in state 0, or Mario levels without any enemies, then we need a linear number of instructions to build up the initial counter values (by repeated addition and multiplication implemented by repeated addition).

3 New Super Mario Bros. Series

An earlier version of this work [2] showed that Generalized New Super Mario Bros. is undecidable by a reduction from reachability with the Inc-DecNZ-PZ gadget and a crossover. In this section, we will prove undecidability for all games in the New Super Mario Bros.

series by building a similar counter with the Inc[1, 2]-DecNZ[1, 2]-PZ gadget. Specifically, this gadget can be used in New Super Mario Bros., New Super Mario Bros. Wii, New Super Mario Bros. U, and New Super Mario Bros. 2. We use the number of enemies called Goombas to keep track of the counter state, and since we can simulate the Inc-DecNZ-PZ gadget by Theorem 14 we can obtain undecidability in constant size as in Section 2.6. However, this result is not necessarily planar, so we provide a simple crossover gadget.

In this section, we make heavy use of a mechanic in the New Super Mario Bros. games called ***events***, which allows the player to interact with blocks via switches and event controllers. The functionality of events that we will use can be summarized as follows, although exact implementations vary per game:

- Each event can be either on or off.
- Event controllers toggle one event based on the status of another.
- Location controllers toggle an event based on the status of enemies or players in a pre-defined location.
- There are blocks which appear or disappear according to the status of an event.

▶ **Theorem 6.** *The New Super Mario Bros. games are RE-complete.*

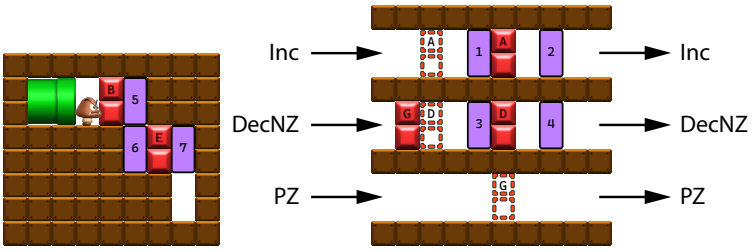
Proof. Our counter gadget is composed of two parts: the counter, pictured in Figure 15a, and the tunnel, pictured in Figure 15b. Each purple area represents a defined location. The groups of blocks associated with a letter are controlled by the event with the given letter[3]. When the event with the corresponding letter changes state, the blocks also change state between visible and invisible (where invisible blocks are also intangible). Invisible blocks are depicted as outlined instead of filled. There are also several invisible control objects:

- An enemy spawner spawns Goombas from the pipe .
- A location controller activates event A when a player enters location 1.
- An event controller activates event B when event A is activated.
- A location controller deactivates event A when a player enters location 2.
- A location controller activates event D when a player enters location 3.
- An event controller activates event E when event D is activated.
- A location controller deactivates event D when a player enters location 4.
- A location controller activates event C when an enemy enters location 5.
- An event controller deactivates event B when event C is activated.
- A location controller activates event F when an enemy enters location 7.
- An event controller deactivates event E when event F is activated.
- An event controller activates event G if and only if an enemy is in location 6.

Inc[1, 2]**.** When the player enters the Inc tunnel, they encounter location 1 which enables event A, preventing backtracking and opening the path through the tunnel. They then enter location 2, which reverses this change, preventing backtracking. At the same time, event A triggers event B which allows a Goomba to pass through the blocks tied to event B. That same Goomba enters location 5 which indirectly deactivates event B. Because of the way Goombas spread out when moving, only one or two Goombas will pass through during this time. The incremented Goomba(s) end up in location 6.

DecNZ[1, 2]**.** The DecNZ tunnel works analogously to the Inc tunnel, instead allowing one Goomba to pass through the blocks tied to event E. In addition, if a Goomba is

[3] In memory, events are numbered, not lettered, but we use letters to disambiguate with locations



(a) Counter (for Goombas only).

(b) Tunnel (for Mario only).

Figure 15 The New Super Mario Bros. counter gadget in state 0.

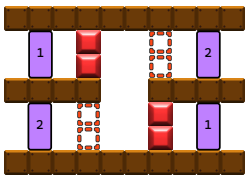


Figure 16 The New Super Mario Bros. crossover gadget.

not in location 6, i.e. the counter value is zero, event G will be disabled, and blocks tied to G will block traversal through the decrement tunnel, enforcing the NZ condition.

PZ. Similarly to how the NZ condition is enforced, if a Goomba is in location 6, event G is active and blocks tied to G block the path through PZ.

To complete the proof, we provide a crossover gadget, as pictured in Figure 16. This is another event-based gadget, where all blocks are controlled by one event, which is off as pictured. Entering either location labeled 1 will enable the event, toggling the states of all blocks and allowing traversal only between the top left and bottom right, and entering either location labeled 2 will disable the event, returning it to the state pictured and allowing traversal between the top right and bottom left. ◀

3.1 Constant-Size New Super Mario Bros. Wii

As described in Section 2.6, the above result implies undecidability for a level of constant size. In fact, we have explicitly built a universal counter machine in New Super Mario Bros. Wii within a *single screen*. Figure 17 depicts such a counter, as shown in the *Reggie* level editor. Our level file is available to download and play [17].

Specifically, this level builds the strongly universal counter machine U_{32} of Korec [13], following an approach taken for the video game *Baba Is You* [1]. The construction of this level is somewhat different from the reductions we have described, making many simplifications to fit the level on one screen. The left side of the level is devoted to the 8 registers, while the majority of the level is devoted to traversal tunnels which control the states of the gadget. Effectively, this is the same as the two components of our gadget featured in Theorem 6, but with extra space removed. Branches are achieved easily by having an event controller to check whether a Goomba is in the counter location. As pictured, the gadget has values of $0, 0, 1, 1, 0, 0, 1, 0$ for registers $0, 1, 2, 3, 4, 5, 6, 7$ respectively. The goal of the level is to traverse from the starting area (in the bottom left) to the pipe (bottom right) which leads to the flagpole. A mini-mushroom is provided in the starting area to help the player traverse more easily through tight tunnels, but is not essential to the functioning of the gadget.

Because this entire counter fits on one screen, it is unnecessary to generalize level size as we have with other games. However, we still need to generalize the game in a few ways. Specifically, the actual game features a timer, and the actual game engine will spawn a Goomba from each pipe only if there are not already eight Goombas on screen. Both of these limitations need to be removed to make the level fully work.

Because the functionality of events is effectively the same across the New Super Mario Bros. series, this same counter should be able to be built in the other games, although smaller Nintendo DS screen sizes seem to prevent making such a large screen size in the DS games.

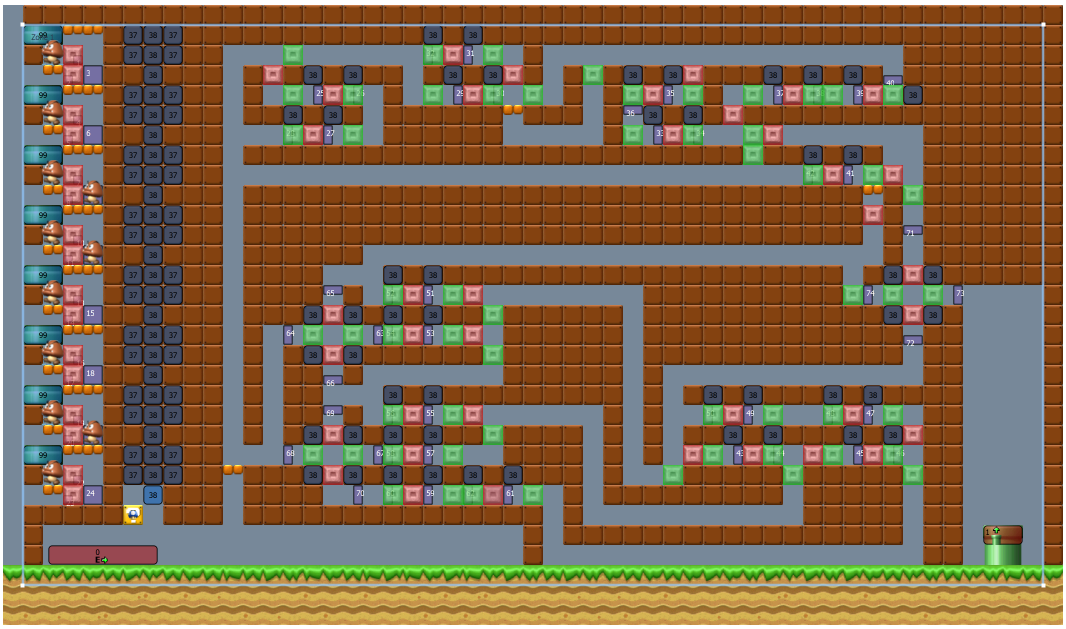


Figure 17 U_{32} in New Super Mario Bros. Wii.

4 Super Mario Maker

First we describe how the screen size (which we do not generalize from the implemented size) affects local vs. global behaviors in Super Mario Maker 1 and 2. As in most Mario games, memory and effects are typically limited to the extent of the screen — technically, the ***relevant screen*** which extends four blocks beyond the visible screen. For example, activating a P-switch temporarily turns coins into blocks, but only within the current relevant screen; as Mario and this screen moves, which coins are transformed changes, which can impact nearby enemies etc. Similarly, enemies typically ***spawn*** when the relevant screen first overlaps their start location, and then ***despawn*** when they leave that relevant screen.

But the Super Mario Maker game engines also define a notion of ***global*** objects [9] whose state is remembered even when it outside the relevant screen. Only a handful of objects are inherently global; for example, One-ways spawn at the beginning of the level and never despawn, while Yoshi spawns when reaching the screen but then never despawns. Crucially, an object becomes global if it is on Tracks, or is on top of another global object; this principle is called ***global ground***. We use this property to make global any objects we want to be remembered, such as the enemies representing the state of a counter.

The Super Mario Maker games are also unusual in that they allow level creation in multiple (4–5) game styles, which each offer some slightly unique mechanics. Our constructions that apply to four styles make use of only mechanics that are present in all four styles, and all four of the styles have the same physics.

4.1 Super Mario Maker 1

▶ **Theorem 7.** *All four game styles (Super Mario Bros. 1, Super Mario Bros. 3, Super Mario World, and New Super Mario Bros. U) of Super Mario Maker 1 are RE-complete.*

Proof. We reduce from planar reachability with the Inc-DecNZ-PZ gadget (Theorem 4).

Our gadget uses the following elements:

- **Solid ground:**  Solid ground with no special effects.
- **Semisolid platforms:**  Platforms which can be jumped through from below but are solid from above.
- **One-ways:**  Walls which allow entities (Mario, enemies, etc.) to pass the white bar only in the direction of the arrows.
- **Brick block:**  Solid block which can be hit from below to defeat enemies or bounce trampolines above it,
- **Coin:**  Transparent course element which allows Mario and enemies to pass through freely. If Mario touches a coin, it is collected and disappears from the course.
- **P-switches:**  Switches which turn coins into brick blocks and vice versa for the duration of a timer.
- **Tracks:** Rails specifying periodic movement of attached entities; platforms  on tracks are global ground, meaning that they and entities on them do not despawn when offscreen.
- **Spikes:**  Blocks which damage Mario upon contact.
- **Goombas:**  Enemies which damage Mario when he contacts from any direction but above. Can safely walk on spikes and are defeated by being jumped on, giving Mario a vertical boost similar to a jump. We use big (2×2) Goombas in this construction.
- **Trampoline:**  Item which bounces entities on it up into the air.
- **Pipes:**  Elements which periodically spawn a particular item or enemy (drawn next to the pipe) into the course. If the pipe spawns items, it will only do so if the last element that the pipe spawned is no longer loaded or exists. Enemies such as Goombas  spawn indefinitely. Their spawn frequency can be adjusted to one of four speeds.

The key idea in this gadget is similar to other Mario Inc-DecNZ-PZ reductions: use the number of Goombas  in a particular location to represent the value of a counter. The gadget has infinitely many enumerated states such that, for any integer $n \geq 0$, there exists a state with a collection of n Goombas. The particular construction of the gadget determines how to increase, decrease, and check for zero in the gadget.

The counter element of the gadget is a semisolid platform  on a track . The track makes the platform global ground, preventing the Goombas  from despawning as Mario moves away from the gadget.

Another critical element to the gadget construction is Goomba pushing. If two Goombas  are on the same y level and walk into each other, they both bounce off and move in the opposite direction. This property is useful for building single-Goomba chambers: if there is a one-way  into a chamber with the width of a single Goomba that can only be accessed from the side, at most one Goomba can occupy it. This is because, if any other Goombas attempt to walk into the space, they will bounce off and be unable to pass through the one-way.

Inc. On traversal of the increment path, the player is forced to add exactly one Goomba  to the gadget counter. Above the semisolid platform  counter, there is a long horizontal chamber with coins  for a floor. At one end of the corridor, a pipe  spawns Goombas which immediately fall through the coin floor. At the other end of the pipe, a one-way  leads to a space wide enough for exactly one Goomba bounded by coins: a single-Goomba chamber as described above. This chamber is level with the pipe and has a solid floor, unlike the coin floor of the corridor.

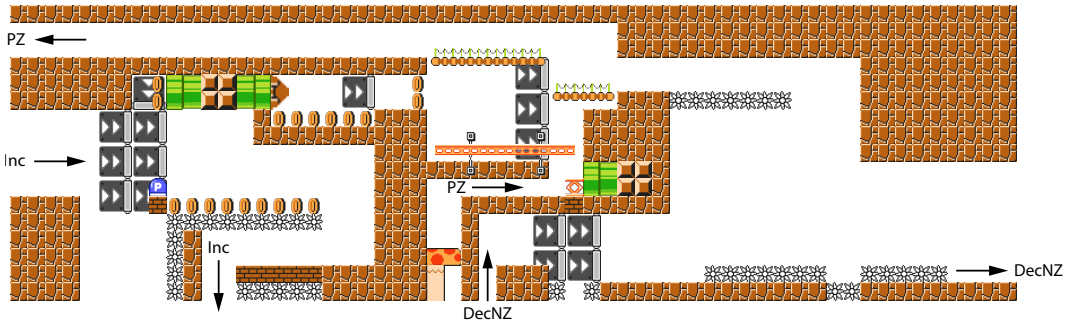


Figure 18 The counter gadget for Super Mario Maker.

On activation of a P-switch, the coins turn to brick blocks. Any Goombas already in the gadget are defeated instantly, and future spawned Goombas can freely walk towards the chamber. The coins bordering the chamber on the other side will be brick blocks, trapping exactly one Goomba in the chamber. Once the P-switch timer expires, the floor will transform back into coins and all the Goombas currently in the corridor will be too low to enter the chamber. The brick blocks at the end of the chamber will turn back into coins, allowing the Goomba to pass through and fall onto the semisolid platforms to increment the counter. As long as the corridor length and pipe spawn frequency guarantee that at least one Goomba reaches the chamber by the end of a P-switch cycle, every P-switch activation guarantees a counter increment by exactly one.

An Inc traversal path can be built such that Mario is forced to hit a P-switch exactly once. We accomplish this by spawning P-switches from a pipe in the ceiling, which fall onto a brick block behind a one-way. Beyond the brick block, a row of coins above a row of spikes leads to solid ground. After the solid ground, another row of spikes below a row of brick blocks lead out to the exit port. The gadget is constructed such that Mario must jump through a set of one-ways: one before hitting the P-switch and another immediately after.

To traverse, Mario must jump past the one-way and land on the P-switch, then run across the transformed brick blocks over the spikes and onto the solid ground. Once through the one-way, Mario cannot go back and activate another P-switch. Since the exit port is blocked by a spike row, Mario is forced to wait out the full P-switch timer, guaranteeing that the Goomba remains loaded until it can fall onto the counter and reach global ground.

DecNZ. At the end of the semisolid counter platform farthest from the chamber described above, another Goomba chamber is formed by a one-way and solid ground tiles. This guarantees that, in any state $n > 0$, exactly one Goomba is in the chambered section of the counter. Below the chamber, a pipe spawns trampolines onto a brick block which is accessible from below. Directly above the chamber, a semisolid platform leads toward a long fall. The platform is low enough such that a Goomba bounced up from the counter by a trampoline would land on the upper semisolid platform, no longer in the counter. The upper semisolid platform is bounded on one side by one-ways, and on the other it has a long fall onto a row of spikes.

The decrement works by having small Mario hit the brick block from below, bouncing the trampoline upwards and allowing it to bounce a Goomba up into the air onto the upper semisolid platform. Mario must then traverse the row of spikes, which

is too long to clear in a single jump. If the counter had more than one Goomba and the bounce succeeded, a Goomba will fall onto the spikes, allowing Mario to jump off it to gain extra height and clear the row, exiting the gadget. If the counter was at zero, Mario would be unable to clear the spikes and lose a life instead. This checks that the decrement removes at least one Goomba.

We enforce the condition that Mario can only decrement once per traversal by using one-ways. Mario is forced to jump through a one-way, hit the brick block, then fall through another one-way. Using this construction, Mario can only hit the brick block once, making a tight lower bound and guaranteeing decrement of exactly one enemy.

PZ. The traverse path requires that Mario jumps through the semisolid counter platform from below, in particular through the bottom of the single-Goomba chamber. If the gadget is in any nonzero state, there is guaranteed to be one Goomba in the chamber. Mario will take damage when making contact from below and lose a life. If the gadget is in a zero state instead, the Goomba chamber will be empty and Mario will be free to traverse. ◀

4.2 Super Mario Maker 2

▶ **Theorem 8.** *All four normal game styles (Super Mario Bros. 1, Super Mario Bros. 3, Super Mario World, and New Super Mario Bros. U) of Super Mario Maker 2 are RE-complete.*

The reduction for Super Mario Maker 1 from Section 4.1 might be adaptable to Super Mario Maker 2, but differing mechanics in the latter game (which allows picking up items from behind one-ways) would mean that the gadget would have to be altered for some of the game styles in order to prevent breaking the gadget. Thus, we demonstrate here a different gadget which works equally well for all four styles, making use of some mechanics exclusive to Super Mario Maker 2.

Proof. We reduce from planar reachability with the Inc-DecNZ-PZ gadget (Theorem 4). Figure 19 shows the gadget, and Figure 20 shows portions of a playthrough. We use some entities already described in Section 4.1: solid ground, semisolid platforms, one-ways, P-switches, tracks, spikes, platforms, goombas, and pipes. In addition, we use the following entities:

- **Donut blocks:** Semisolid blocks that begin to fall after Mario stands on them for a short amount of time. Placing many in a vertical drop effectively slows Mario's downward traversal.
- **Note blocks:** Bouncy blocks that cause any entity that walks on them to be bounced upward.
- **P-blocks:** These blocks flip from being outlines to being solid (or vice versa) while the P-switch is active. In this construction, they prevent the clown car from spawning out of the blue pipe unless the P-switch is active.
- **On/off switches:** When hit from below, these switches flip a global on/off state, toggling the solidity of on/off blocks: vs. .
- **On/off blocks:** These blocks come in and out of existence based on the state of the on/off switches : in the "on" red state, and in the "off" blue state.
- **Blaster:** These blasters shoot a shell in the direction of Mario if he is sufficiently close and the corresponding barrel of the blaster is not blocked. These shells can activate on/off switches .

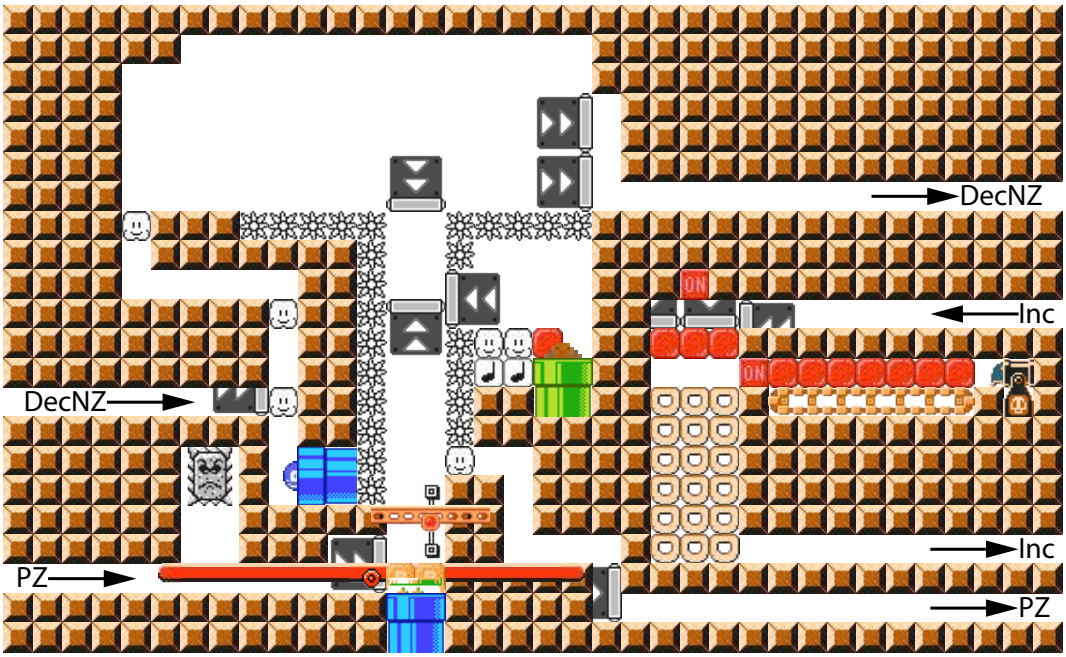
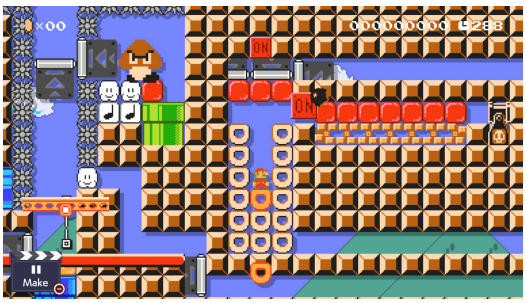
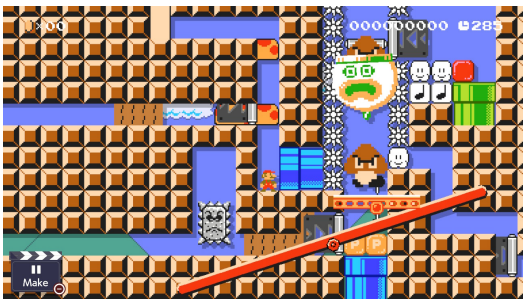


Figure 19 Complete Inc-DecNZ-PZ gadget for the normal game styles of Super Mario Maker 2.



(a) Inc path: The shell resets the on/off switch after exactly one Goomba has spawned.



(b) DecNZ path: A single Goomba rides the panicked clown car upward.

Figure 20 Screenshots from playing the gadget from Figure 19 in Super Mario Maker 2.

- **Clown cars:** These vehicles allow exactly one falling enemy to enter and ride them. If a loaded clown car touches spikes, then it will panic and fly upward.
- **Thwomps:** An enemy that charges downward if Mario is sufficiently close and at or below the level of the Thwomp, and then resets to its original location.
- **Seesaws:** A tilting platform that balances based on the weight on both sides of the center. If a Thwomp pounds on one side of the seesaw, it will launch up any enemies on the other side of the platform. This is used to get the Goombas into a falling state so that they can be picked up by the clown car.
- **Conveyors:** A moving platform that can slow down a shell that is moving in the opposite direction. In this construction, it allows us to use less space to achieve the timing for the shell that deactivates an activated on/off switch.

As before, the state of the counter gadget is the number of Goombas on the platform on the track.

PZ. When the gadget is in state 0, there are no Goombas on the platform, allowing the player to traverse the PZ path. When the gadget is in any state > 0, there is at least one Goomba on the platform, preventing the player from traversing the PZ path without taking damage (and thus dying).

Inc. When the player enters the increment path, the first tunnel is long enough that a shell will be launched left from the blaster and bounce within the single adjacent space. At the end of this tunnel, the leftward one-way ensures that the player cannot hit the on/off switch without fully activating the increment, and the downward facing one-ways and on/off blocks below ensure that the player cannot hit the on/off switch more than once.

Once the on/off switch is triggered, the green pipe, being no longer blocked from spawning, immediately starts spawning Goombas. At the same time, the shell, no longer confined to a single space, starts moving left toward an on/off switch. The shell's distance to the switch, along with the speed of the conveyor and the spawning speed of the green pipe, ensures that exactly one Goomba can spawn before the pipe is again blocked from spawning by the shell hitting the on/off switch, as shown in Figure 20a. This single Goomba moves left, bouncing up from walking on note blocks, and falls onto the platform on the track.

The rows of donut blocks that Mario must endure at the end of this path ensures that the player keeps the Goomba loaded on screen until it is on the global ground of the platform, so that the counter value is correct.

DecNZ. Meanwhile, when the player enters the decrement path, they fall onto a P-switch spawned from the the left blue pipe, activating the P-switch. (Note that Mario now blocks spawns from the pipe, so no new P-switches will spawn. And even if the player managed to fall into the hole exactly when the P-switch was spawning, since it is a 1-wide hole, they can never pick up the P-switch, and only fall onto it or block spawns from the pipe.)

This causes the bottom blue pipe, no longer being blocked from spawning, to spawn a clown car. At the same time, falling into this hole is just far enough down to trigger the Thwomp, which repeatedly charges downward onto the seesaw. This, in turn, causes the opposite end of the seesaw to shoot up, launching the Goombas.

Though it usually takes a few seconds for all of this to line up, eventually this results in the Goombas being shot up while the clown car is under them, causing one Goomba to enter the clown car while the rest fall back onto the platform. Now that it is entered, the clown car panics from touching spikes, and flies upward until it gets stuck between the two one-ways at the top of the spike column. Figure 20b shows the separation of a single Goomba and panicking of the clown car.

Now the player is free to jump up to the top tunnel of the decrement path, where they can clear the long jump over the spikes by bouncing on the Goomba in the clown car in the middle of the jump. If the player performs an illegal decrement from state 0, then the clown car will never fly up because it never gets entered, and Mario will not be able to clear the long jump over the spikes. ◀

The 3D world style of Super Mario Maker 2 has many different mechanics from that of the four regular styles, and so requires a different construction to show that it is RE-hard. See the full paper for details.

▶ **Theorem 9.** *The Super Mario 3D World style of Super Mario Maker 2 is RE-complete.*

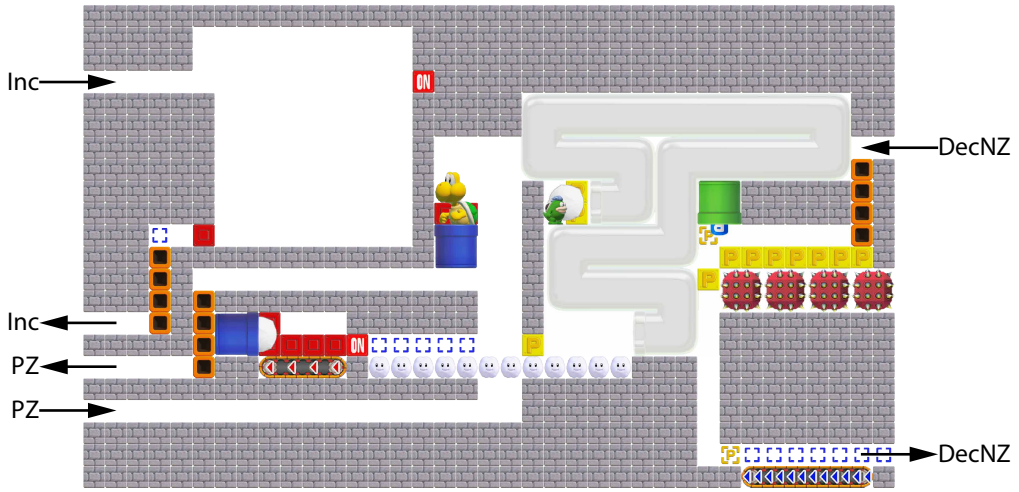


Figure 21 The complete Inc-DecNZ-PZ gadget for the 3D World game style of Super Mario Maker 2

Proof. Our gadget is presented in Figure 21. The details of this gadget are covered in the full version of our paper. ◀

5 Open Problems

Of the 2D Mario Games released since New Super Mario Bros., we have shown that all except for Super Mario Wonder are undecidable, and a natural open question is whether it is too. There is evidence which suggests that it might be based on the presence of events and infinitely spawning Goombas, but the game is still very new, and more research is needed to understand the mechanics of the game well enough to make further claims about undecidability.

There are also several older 2D Mario Games which have evidence that they might be able to build counters. In particular, any game with a Lakitu has a way of generating unlimited numbers of Spinies, Super Mario Bros. 3 and Super Mario World 2: Yoshi's Island both have enemies that can be generated from pipes, and Super Mario World has enemies generated by falling from the sky. Can we use any of the mechanics in those games to build counters? If any of these other games are not undecidable, it would also be noteworthy if we can obtain any other upper bounds on their complexity.

Finally, we showed that all games considered here are hard in constant-size levels, but it is not certain exactly what that constant is. We provided an explicit example of a single-screen counter in New Super Mario Bros. Wii, but we can definitely compact it further if we want to build the smallest possible universal counter machine. Furthermore, we can consider the Super Mario Maker games and whether it is possible to build a universal counter machine that fits inside of the standard constraint on level size.

References

1 Zachary Abel and Della Hendrickson. Baba is universal. In *Proceedings of the 12th International Conference on Fun with Algorithms (FUN 2024)*, pages 31:1–31:16, La Maddalena, Italy, June 2024.

2 Hayashi Ani. Unsimulability, universality, and undecidability in the gizmo framework. Master's thesis, Massachusetts Institute of Technology, 2023.

3 Hayashi Ani, Jeffrey Bosboom, Erik D. Demaine, Jenny Diomidova, Della Hendrickson, and Jayson Lynch. Walking through doors is hard, even without staircases: Proving PSPACE-hardness via planar assemblies of door gadgets. In *Proceedings of the 10th International Conference on Fun with Algorithms (FUN 2020)*, pages 3:1–3:23, La Maddalena, Italy, September 2020.

4 Hayashi Ani, Lily Chung, Erik D. Demaine, Jenny Diomidova, Della Hendrickson, and Jayson Lynch. Pushing blocks via checkable gadgets: PSPACE-completeness of Push-1F and Block/Box Dude. In *Proceedings of the 11th International Conference on Fun with Algorithms (FUN 2022)*, pages 2:1–2:30, Favignana, Italy, May–June 2022.

5 Hayashi Ani, Erik D. Demaine, Jenny Diomidova, Della Hendrickson, and Jayson Lynch. Traversability, reconfiguration, and reachability in the gadget framework. In *Proceedings of the 16th International Conference and Workshops on Algorithms and Computation (WALCOM 2022)*, volume 13174 of *Lecture Notes in Computer Science*, pages 47–58, Jember, Indonesia, March 2022.

6 Hayashi Ani, Erik D. Demaine, Della Hendrickson, and Jayson Lynch. Trains, games, and complexity: 0/1/2-player motion planning through input/output gadgets. In *Proceedings of the 16th International Conference and Workshops on Algorithms and Computation (WALCOM 2022)*, volume 13174 of *Lecture Notes in Computer Science*, pages 187–198, Jember, Indonesia, March 2022.

7 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.

8 Erik D. Demaine, Della Hendrickson, and Jayson Lynch. Toward a general theory of motion planning complexity: Characterizing which gadgets make games hard. In *Proceedings of the 11th Conference on Innovations in Theoretical Computer Science (ITCS 2020)*, pages 62:1–62:42, Seattle, Washington, January 12–14 2020.

9 flamewizzy21. "Don't leave me!" — Guide to global loading. Reddit post, January 2020. URL: https://www.reddit.com/r/MarioMaker/comments/eobdmj/dont_leave_me_guide_to_global_loading/.

10 Linus Hamilton. Braid is undecidable. arXiv:1412.0784, 2014. URL: http://arxiv.org/abs/1412.0784.

11 Della Hendrickson. Gadgets and gizmos: A formal model of simulation in the gadget framework for motion planning. Master's thesis, Massachusetts Institute of Technology, 2021.

12 Erik Kain. 'Braid' is a postmodern Super Mario Bros. Forbes Games blog, 18 June 2012. URL: https://www.forbes.com/sites/erikkain/2012/06/18/braid-is-a-postmodern-super-mario-bros/.

13 Ivan Korec. Small universal register machines. *Theoretical Computer Science*, 168(2):267–301, 1996. doi:10.1016/S0304-3975(96)00080-1.

14 Jayson Lynch. *A framework for proving the computational intractability of motion planning problems.* PhD thesis, Massachusetts Institute of Technology, 2020.

15 Marvin L. Minsky. Recursive unsolvability of Post's problem of "Tag" and other topics in theory of Turing machines. *Annals of Mathematics*, 74(3):437–455, 1961. URL: http://www.jstor.org/stable/1970290.

16 Marvin L. Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Inc., 1967.

17 MIT Hardness Group. Mario hardness gadgets. GitHub repository. URL: https://github.com/65440-2023/mario-hardness-gadgets.