Advanced Spikes 'n' Stuff: An NP-Hard Puzzle Game in Which All Tutorials Are Efficiently Solvable

Christian Ikenmeyer
University of Warwick, UK

Dylan Khangure
University of Warwick, UK

Abstract

We adjust Alan Hazelden's 2017 polynomial time solvable puzzle game Spikes 'n' Stuff: We obtain the NP-complete puzzle game Advanced Spikes 'n' Stuff with 3 trap types so that each strict subset of the traps results in a polynomial time solvable puzzle game. We think of this as a "hard game in which all tutorial levels are easy". The polynomial time algorithms for solving the tutorial games turn out to be quite different to each other.

While numerous papers analyze the complexity of games and which game objects make a game NP-hard, our paper is the first to study a game where the NP-hardness can only be achieved by a combination of all game objects, assuming P differs from NP.

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

Keywords and phrases computational complexity, P vs NP, motion planning, games

Digital Object Identifier 10.4230/LIPIcs.FUN.2024.18

1 Motivation

By a *tutorial* for a game we mean the game with some of its parts removed. The exact notion of what constitutes a "removable part" of a game is arbitrary, but for many games there are natural choices. For games such as Alan Hazelden's 2017 puzzle game *Spikes 'n' Stuff* [1], where the player navigates through a maze with different types of traps, the natural removable parts are the different types of traps. In a very similar manner, natural tutorials have been investigated for example in [9], where the removable parts are the interactive objects (which are not necessarily traps) in a level of the computer game Portal. One main observation in [9] is that many of the interactive objects in Portal are NP-hard on their own, i.e., in these cases the tutorial in which only one type of interactive object is present is already NP-hard. We list many more such examples from the literature in Section 3.

The original Spikes 'n' Stuff game is a turn-based game that is solvable in polynomial time (in the size of the level), because each level only has a state space of polynomially bounded size. We adjust this game slightly so that we get the NP-hard game Advanced Spikes 'n' Stuff, which is a game that is polynomial-time solvable if levels contain only two of the three traps, see Figure 1. The main technical difficulty that we overcome in this paper is that at least one trap type has to result in a superpolynomially large state space, but this trap type is not allowed to make the game NP-hard on its own or combined with any one of the other two trap types. This is the first NP-hard game that we know for which all tutorials are solvable in polynomial time.

Note that artificial NP-hard "games" in which all tutorials are easy can readily be created for example as follows (with two removable "game" parts instead of three): The game is to solve a 3SAT instance, and the two removable parts are *negation of variables* and *disjunction*.

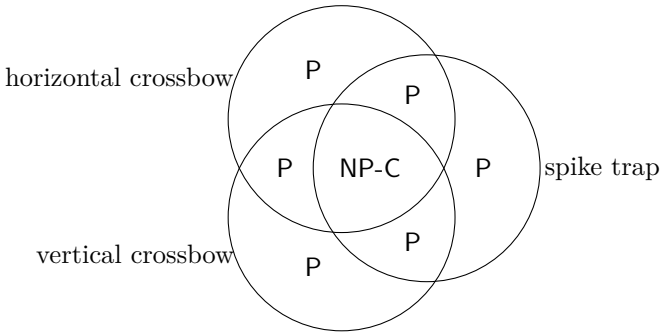


Figure 1 If all three trap types in Advanced Spikes 'n' Stuff are present, the game is NP-complete. Otherwise it is solvable in polynomial time.

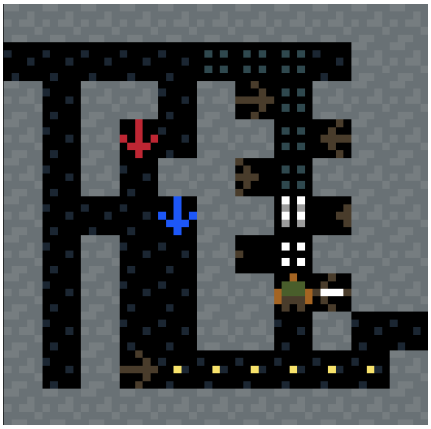
While 3SAT is NP-complete, if we remove the negation of variables, the problem becomes trivial, whereas when we remove disjunctions, then all clauses have length 1, which also trivializes the problem.

Our point is that Advanced Spikes 'n' Stuff is only a slight variation of the fun game Spikes 'n' Stuff, and therefore our techniques (or refinements thereof) may have implications on actual game design approaches in the future. As a first insight, we can say that the fact that all tutorials are easier to solve than the complete game forces the player to rethink their strategies (that they have developed while playing through tutorial levels) when playing the complete game. Forcing the player to re-think established assumptions is usually a desirable property in puzzle games.

2 Advanced Spikes 'n' Stuff

Advanced Spikes 'n' Stuff is a turn-based single player puzzle game where the goal is to move an agent from the start position to the finish position through a maze without the agent getting killed by traps. The game is played on a square grid of tiles. Every tile is either walkable without a spike trap, or walkable with a spike trap, or a wall, or a horizontal crossbow trap (facing left or right), or a vertical crossbow trap (facing up or down). Each trap has an internal state, where spike traps have three states $0, 1, 2$, vertical crossbows have two states $0, 1$, and horizontal crossbows have six states $0, 1, \ldots, 5$. The state 0 is called the idle state. In addition to this state, every *vertical* crossbow is either *functional* or *deactivated*. Horizontal crossbows count as always *functional*. We define the *range* of a crossbow to be the set of tiles in a direct line in front of the crossbow up to the next unwalkable tile. Figure 2 depicts an example level of Advanced Spikes 'n' Stuff.

On each turn, first the agent must move either horizontally or vertically from its tile p to an adjacent walkable tile q. The direction is chosen by the player. If q is a state 1 spike trap or q is in the range of a functional state 1 crossbow, then the agent dies and the player loses the game. Traps in other states do not harm the agent. At this point, if the player hasn't lost yet, all traps that are not in their idle state or deactivated advance their state by 1, coming back to their idle state after their maximal state is reached. If p is a spike trap, then now that spike trap's state is set to 1. If q is in the range of an idle functional crossbow trap, the crossbow trap's state is now set to 1. If q is left or right of a vertical crossbow trap, that crossbow trap is set to *deactivated* and its state is set to 0. At this point, the turn ends and the player must now make the next choice in which direction to move.



The light gray coloured tiles are the walls, and the darker coloured tiles are the walkable tiles. The start position is the tile at the top-left of the level, and the finish position is the tile at the bottom-right. The agent takes the form of a person and can be seen near the goal position. Each spike trap state is shown in this level. The spike trap immediately above the agent is in state 1 and the one immediately above that is in state 2. All other spike traps in the level are in state 0. Similarly all states of the horizontal crossbow trap are present. The crossbow adjacent to the player is in state 1. Going up from this position, the crossbows are in states 2,3,4 and 5 respectively, with the one at the very top being idle (in state 0). Two vertical crossbows are also in this level; the red one is *functional* and the blue one is *deactivated*. The notion of the *range* of a crossbow is also highlighted in this level using the horizontal crossbow towards the bottom. The walkable tiles with small gold squares are precisely those tiles which are in the range of this crossbow. These gold squares are hidden from the player during the game, and are merely used here for illustrative purposes.

Figure 2 Sample level of Advanced Spikes 'n' Stuff.

A game state of this turn-based game consists of the start and finish positions and the tile arrangement (which do not change throughout the game), the agent's position, the states of all traps, and the functional/deactivated flag of all vertical crossbow traps. The game starts with all traps in their idle state and all vertical crossbows functional. For the sake of simplicity, we assume that

the start tile is not in range of any crossbow trap and the tiles adjacent to the start tile are not spike traps, and (1)

the tiles left and right to of vertical crossbow traps and their adjacent tiles are not spike traps, and they are not in range of a horizontal crossbow trap. (2)

Formally, we consider the problem $\mathsf{ASnS}(S)$ of deciding given an Advanced Spikes 'n' Stuff level satisfying (1) and (2) whether or not the player can move the agent from the start tile to the finish tile without the agent dying by traps. The parameter S is a subset of the set of symbols $\{h, s, v\}$, where $s \in S$ indicates that the levels can have spike traps, $h \in S$ indicates that the levels can have horizontal crossbow traps, and $v \in S$ indicates that the level can have vertical crossbow traps.

▶ **Theorem 2.1** (Main Theorem). $\mathsf{ASnS}(\{h, s, v\})$ *is* NP*-complete, but* $\mathsf{ASnS}(S) \in \mathsf{P}$ *for all strict subsets* $S \subsetneqq \{h, s, v\}$.

Theorem 2.1 is illustrated in Figure 1. Since a game cannot get harder if possible game pieces are removed, the results $\mathsf{ASnS}(\{h\}) \in \mathsf{P}$, $\mathsf{ASnS}(\{s\}) \in \mathsf{P}$, $\mathsf{ASnS}(\{v\}) \in \mathsf{P}$ are direct corollaries from $\mathsf{ASnS}(\{h, s\}) \in \mathsf{P}$, $\mathsf{ASnS}(\{s, v\}) \in \mathsf{P}$.

We prove $\mathsf{ASnS}(\{h, s\}) \in \mathsf{P}$ in Corollary 4.3, $\mathsf{ASnS}(\{v, s\}) \in \mathsf{P}$ in §5, and $\mathsf{ASnS}(\{h, v\}) \in \mathsf{P}$ in §6, interestingly with different proof strategies: While $\mathsf{ASnS}(\{h, s\}) \in \mathsf{P}$ follows from the polynomially bounded state space, the argument for $\mathsf{ASnS}(\{v, s\}) \in \mathsf{P}$ partitions the exponentially large state space into polynomially large pieces and greedily traverses polynomially many pieces, and $\mathsf{ASnS}(\{h, v\}) \in \mathsf{P}$ uses the same strategy, but with a more involved argument, because in the presence of horizontal crossbow traps, the agent cannot

safely traverse a safe walk in the reverse direction (see the discussion in §6). We prove the NP-completeness of ASnS($\{h, s, v\}$) in Theorem 7.3. We use a standard framework for constructing a polynomial-time reduction from 3SAT to ASnS($\{h, s, v\}$), closely based on [3]. Early versions of this framework appear in the hardness proofs for games such as SOKOBAN [10] and PushPush [8], but it has since been refined, as can be seen in the papers [2] and [7].

The original game Spikes 'n' Stuff

In the original Spikes 'n' Stuff the horizontal crossbow traps have only 4 states and the vertical crossbow traps work in the same way as the horizontal crossbows traps (and there are some other minor changes about arrows hitting each other mid-air and the player pulling a treasure, which add some twists to the game, but these are not essential, so we do not discuss them). Therefore, no crossbow trap can be deactivated, which implies that the state space has polynomially bounded size, see Lemma 4.2. Hence, the original Spikes 'n' Stuff game is solvable in time that is polynomially bounded in the size of the game board, for example by breadth-first-search in the state space.

3 Related Work

[11] proves various theorems about the complexity of two-dimensional platform games. For example, a level containing items for the player to collect is, on its own, polynomial-time solvable, but adding in a time-limit immediately makes it NP-hard. Moreover, platform games with drops longer than the maximum jump height are already NP-hard.

[13] proves that games where doors can be opened with pressure plates, but not closed, are in P, but adding the ability to close doors makes that game NP-hard. Similarly, it is shown that games with buttons that act on only a single door are in P, but games where a single button may act on two or more doors are NP-hard. Moreover, if a game contains a feature that forces the player to visit various locations, and there are single-use paths or "toll roads" (a certain number of a specific item must be collected to pass), that game is NP-hard.

In the game Lemmings, agents can be assigned skills by the player which alter their behaviour. For example, the *Builder* skill allows the Lemming to construct a bridge, and the *Digger* skill allows the Lemming to dig vertically downwards (see [6], [13] and [14]). One sensible way of defining a tutorial of Lemmings is to consider the game we get by restricting the skills we are allowed to give to the agents (and this is indeed how the actual game is set up). Some of these tutorials where only a single skill can be given to the agents are already NP-hard (even in simplified models of the game): in [6] hardness is proved using only *Digger* skills, and [13] presents a construction that only requires *Basher* or *Miner* skills to achieve hardness.

In [3], we see that even if we consider classic Nintendo games with a subset of their original features we may still get a game that is NP-hard. The original Super Mario Bros. is NP-hard with only the following game features: Super Mushrooms to turn Mario into Super Mario (who can break blocks but cannot fit into narrow horizontal corridors), Goombas to turn Super Mario back to normal Mario (who can fit into narrow horizontal corridors but cannot break blocks) vertical drops that are higher than the maximum jump height, and Stars which provide temporary invulnerability from Firebars. Pokémon is already NP-hard if it contains nothing other than pushable blocks, and also if pushable blocks are not present and the only feature used is battles with enemy Trainers (in this second case, the game is actually NP-complete). Similarly, the original Legend of Zelda is NP-hard even if block

pushing is the only feature that is kept. In later Zelda games, there are ice blocks. These are blocks that when pushed, slide until they hit an obstruction. It is shown that with just this feature, the game is in fact PSPACE-complete.

Another platform game is Celeste, where the player navigates through various levels containing obstacles. The character they control can move in eight directions, and has the ability to jump, dash and temporarily grab onto walls. The game contains many different mechanics and types of obstacles, but [2] proves that even if only the following are present, the game is already NP-hard: platforms that break when stood on, button-operated doors, and special types of blocks that teleport the player in a straight line. [5] solves the PSPACE-hardness for several of its tutorials.

[12] explores a tiling problem that is based on the game KPlumber. The input to the problem is a grid of tiles, with each tile being one of six possible types (O, C, D, S, T, X), and the goal is to rotate the tiles in order to reach a desired goal state. If the tiles C, D, S are used, the tutorial is already NP-complete. The same holds for C, S, T. For some subsets of $\{O, C, D, S, T, X\}$ the complexity has not been classified.

Another game that has been studied which contains many natural tutorials is Tetris. Simpler versions of the game can be obtained by, for example, restricting the size of the gameboard, or the pieces that are available. Some of these simpler versions are proven to be solvable in polynomial time in [4], some are NP-hard, where the tutorials are distinguished by board size, piece size, and empty/non-empty initial board state. We have not found any analysis in the literature about the complexity when the set of tetris pieces is restricted to a subset.

4 The game states in Advanced Spikes 'n' Stuff

In this section we prove $\mathsf{ASnS}(\{h, s\}) \in \mathsf{P}$.

▶ **Definition 4.1.** *A game state is called* tame *if at most 2 spike traps are not in the idle state and at most 12 crossbow traps are not in their idle state.*

Note that this definition poses no constraints on the *functional/deactivated* flags of the vertical crossbow traps, which means that the set of tame game states can be exponential in size.

▶ **Lemma 4.2.** *In a game of Advanced Spikes 'n' Stuff, only tame game states can be reached.*

Proof. During each move the agent can only set one spike trap to state 1. Hence, during a move only one spike trap can go from state 1 to state 2.

Similarly, during each move the agent can only trigger 2 crossbow traps, which have 6 or 2 states each, and $2 \max\{2, 6\} = 12$. ◀

▶ **Corollary 4.3.** $\mathsf{ASnS}(\{h, s\}) \in \mathsf{P}$.

Proof. If no vertical crossbow traps are present, then the size of the set of tame game states is polynomially bounded. By Lemma 4.2 it is sufficient to consider only this space. Hence, a breadth-first-search in the space of tame game states solves $\mathsf{ASnS}(\{h, s\})$ in polynomial time. ◀

5 ASnS($\{s, v\}$) $\in$ P

We prove that ASnS($\{s, v\}$) $\in$ P. First, note that

- a spike trap kills an agent if and only if the agent moves from the spike trap to an adjacent tile and immediately back on the next turn.
- a functional vertical crossbow trap kills the agent if and only the agent makes a vertical move within its range.

(3)

Let V denote the set of vertical crossbow traps. For any subset of $S \subseteq V$ of functional vertical crossbow traps, let $R_S \subseteq S$ denote the set of functional vertical crossbow traps the agent can deactivate when starting at the start tile and making game moves without dying and without deactivating any other functional vertical crossbow trap. For each fixed S, the set R_S can be determined in polynomial time using the observation in the above two bullet points, for example by a breadth-first-search in the subset T_S of the space of tame game states that have exactly the vertical crossbows S functional. Note that the size of T_S is polynomially bounded.

The polynomial-time algorithm to solve ASnS($\{s, v\}$) initializes $S_0 := V$. Then it loops the following for $i = 0, \ldots$:

1. If the game can be solved by just traversing game states in T_{S_i}, return `true`.
2. Determine R_{S_i}.
3. If $R_{S_i} = \emptyset$, return `false`. Else, pick an arbitrary $r_i \in R_{S_i}$ and define $S_{i+1} = S_i \setminus \{r_i\}$ and continue the loop with the next i.

If `true` is returned, then the solution to the level is by first moving the agent to deactivate r_0, then moving the agent back to the start, then deactivating r_1, then moving the agent back to the start, and so on, until finally traversing the game states in T_{S_i} to the finish tile. The crucial observation here is that the properties (3) and (2) imply that going back to the start by tracing the steps backwards will not kill the agent.

If `false` is returned, then S_i is the unique maximal set of functional vertical crossbow traps that can be reached from the start game state. But in T_{S_i} there is no traversal to a finish game state. This finishes the proof of ASnS($\{s, v\}$) $\in$ P.

6 ASnS($\{h, v\}$) $\in$ P

The proof outline is the same as in §5. The only difference is that once the agent deactivates a vertical crossbow trap, it might not be possible to trace the same walk back to the start without being killed, see Figure 3. First, note that the agent gets killed if it makes a vertical step in the range of a functioning vertical crossbow trap. Now, generalize this observation from 2 states (vertical crossbow trap) to 6 states (horizontal crossbow trap): the agent can only make at most 4 successive horizontal moves within the range of a horizontal crossbow trap without being killed. The following Theorem 6.1 states the "converse".

▶ **Theorem 6.1.** *If a level has no spike traps, then for a walk w from tile p to tile q that does not make more than 4 successive horizontal moves within the range of a horizontal crossbow trap and that does not make a vertical move within the range of a functional vertical crossbow trap, there exists a walk w' from p to q such that the agent does not get killed when traversing w'.*

Theorem 6.1 implies ASnS($\{h, v\}$) $\in$ P as follows. Since the property of w of avoiding 4 successive horizontal moves in ranges of horizontal crossbow traps and avoiding vertical moves in ranges of vertical crossbow traps is symmetric, traversing w back to the start can be done safely by replacing it by w' (where we use the safety assumptions (1) and (2)), which finishes the proof of ASnS($\{h, v\}$) $\in$ P.

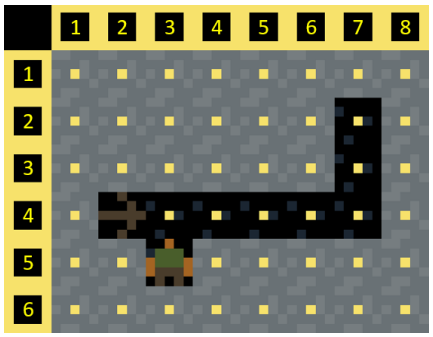


Figure 3 The walk (3,4), (3,5), (3,4), (4,4), (5,4), (6,4), (7,4), (7,3), (7,2) can be safely traversed from the agent's current position, but attempting to traverse this walk in reverse will cause the agent to be killed.

Proof of Theorem 6.1. First, observe that the agent cannot be killed by any horizontal crossbow trap when making a vertical step. The reason is that if a vertical step moves the agent into the range of a horizontal crossbow trap and the crossbow trap was in state 1 (which would kill the agent), then the agent would have set its state to 1 in the last turn. But in the last turn the agent was not in the range (otherwise, a vertical step would move the agent out of the range). Analogously, the agent cannot be killed by a vertical crossbow trap when making a horizontal step. Hence, in w the player is not killed by a vertical crossbow trap.

Now, take the walk w and adjust it as follows to obtain w': Every move in which the agent enters and not immediately leaves the range of a crossbow trap (this must necessarily be a horizontal crossbow trap by assumption on w) that is in state $i \in \{0, 2, 3, 4, 5\}$, the agent repeatedly takes one step back and then one step forward: once if $i = 0$, 0 times if $i = 2$, 3 times if $i \in \{3, 4\}$, and 2 times if $i = 5$. The agent now proceeds traversing w and enters the range with the crossbow trap in state 2, which immediately moves to state 3. The agent can now safely make 4 horizontal steps in the range, while the state of the crossbow trap goes to 4, 5, 0, 1. As discussed before, the new vertical steps cannot get the agent killed by any horizontal crossbow traps, and clearly not by a vertical crossbow trap. An example of this construction of w' is given in Figure 4. ◀

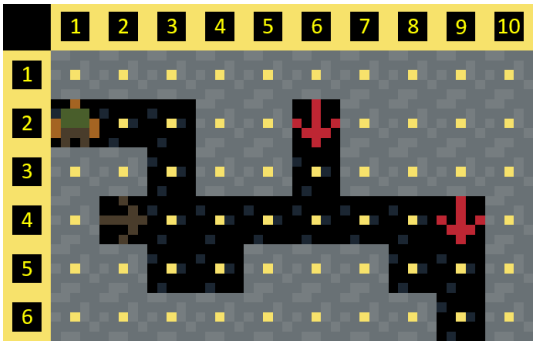


Figure 4 The walk $w = ((1,2), (2,2), (3,2), (3,3), (3,4), (3,5), (4,5), (4,4), (5,4), (6,4), (7,4), (8,4), (8,5), (9,5), (9,6))$ will get the agent killed at $(7,4)$, but the agent survives when using $w' = ((1,2), (2,2), (3,2), (3,3), (3,4), (3,5), (4,5), (4,4),$ **(4,5)**, **(4,4)**, **(4,5)**, **(4,4)**, **(4,5)**, **(4,4)**, $(5,4), (6,4), (7,4), (8,4), (8,5), (9,5), (9,6))$.

7 NP-completeness

▶ **Theorem 7.1.** ASnS($\{h, s, v\}$) $\in$ NP.

Proof. We prove that every level can be solved in a number of steps that is polynomially bounded in the level size. This proves the theorem, because the step sequence gives a polynomially sized witness for the solvability.

Consider an arbitrary level that the player can complete. Since the level is completable, there exists some walk w that the agent is able to traverse from the start tile to the finish tile without being killed. Let $(d_1, d_2, \ldots, d_k)$ be the sequence of vertical crossbows that the agent deactivates during w, in chronological order of deactivation. Note that each crossbow can be deactivated at most once, so k cannot exceed the number of tiles of the level. For $i \in \{0, 1, \ldots, k\}$ let D_i denote the set of tame states (see Definition 4.1) in which exactly the vertical crossbows $\{d_1, \ldots, d_k\}$ are deactivated. By assumption and by Lemma 4.2 there exists a path p from the start game state through the space $D_0 \cup D_1 \cup \cdots \cup D_k$ to a finish game state in which each edge is a game move. Since k is polynomially bounded, and the cardinality of each D_i is polynomially bounded, the length of a shortest path p is polynomially bounded. Such a path serves as the desired NP witness. ◀

7.1 NP-hardness

The construction uses one-way gadgets (where the agent can only pass through in one direction), variable gadgets, clause gadgets, and crossover gadgets: one variable gadget for each variable in the 3SAT instance, and one clause gadget for each clause. Each variable gadget is connected to its clause gadgets by paths, and also to the next variable gadget, see Figure 5.

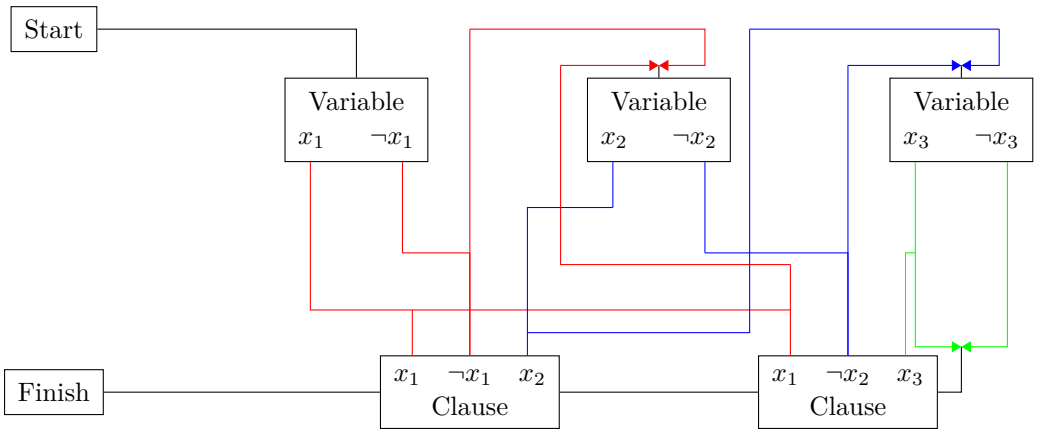


Figure 5 NP-hardness framework: The agent starts at *Start* and tries to get to *Finish*, traversing the edges. Arrows stand for one-way gadgets. Each 4-way intersection must be crossed in a straight manner. This is ensured by crossover gadgets. Our crossover gadget is *not* symmetrical and care must be taken with its placement. Colored edges of the same color belong to the same *section*.

The one-way gadget is constructed as depicted on the left side in Figure 6. It is easy to see that a player can traverse this gadget from north to south by entering the range of the crossbow to set its state to 1, backtracking one step to avoid being hit by the arrow, then proceeding in the obvious way to the exit, which is at the south. However, if the player tries to traverse this gadget from the south to the north, they will find themselves in the state

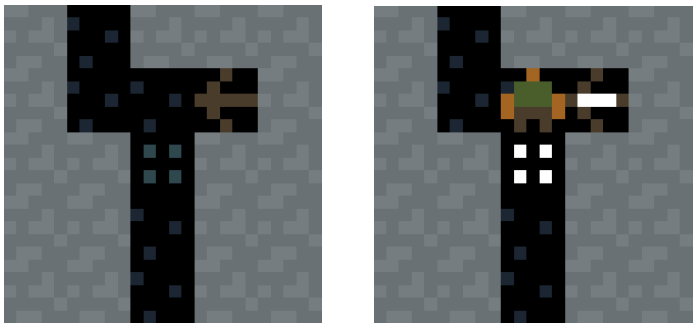


Figure 6 Left: One-way gadget, only traversable from north to south, not from south to north. Right: Traversing one-way gadget from south to north results in agent being killed.

depicted in Figure 6 on the right side. This is clearly a losing position: moving west causes the agent to be killed by the horizontal crossbow trap, and moving south causes them to be killed by the spike trap.

The solvability of levels is *not* invariant under rotations, but for each gadget we also have the gadgets in all four rotations, which is illustrated in Figure 7.

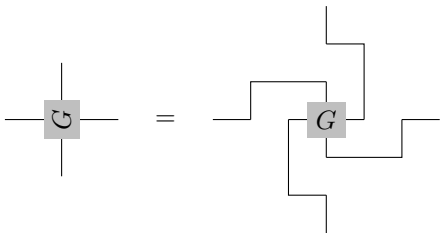


Figure 7 Constructing a 90° rotated version of a gadget G. The depicted gadget G has 1 input on each side, but this construction obviously works for any number of inputs on any side.

The variable gadget is constructed from one-way gadgets as depicted in Figure 8.

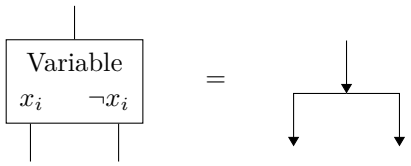


Figure 8 The variable gadget consists of three one-way gadgets.

The clause gadget is also easy to build, see Figure 9. There are three entrances at the north side of the gadget (that correspond to the three literals in the clause that this gadget is representing). When the agent uses one of these entrances, they can unlock the gadget by setting a vertical crossbow trap to *deactivated*. When the agent enters the gadget from the east, they can traverse it to the west if and only if it is unlocked, i.e., it has been unlocked in at least one of the northern entrances.

7.2 The Crossover gadget and its placement

The final gadget required for the construction is the crossover gadget, see Figure 10. There are two paths, which we refer to as the *north-south* path and the *east-west* path, which cross

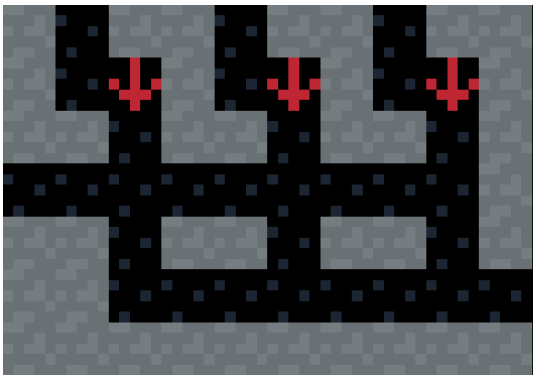


Figure 9 The clause gadget.

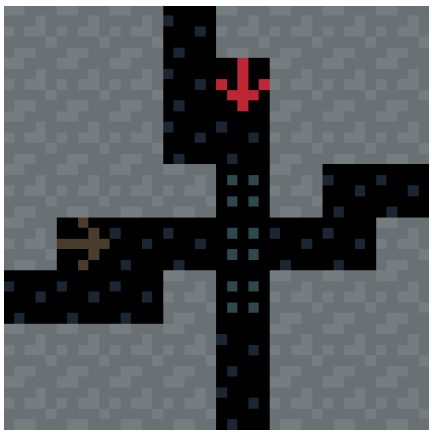


Figure 10 The crossover gadget in its closed state, in the *can be opened at north* rotation.

at the spike trap in the center. This gadget comes equipped with a *state*, which is *closed* if the vertical crossbow is functional, and *open* if it is deactivated. The key properties of the gadget are explained in the following lemma.

▶ **Lemma 7.2.** *If the crossover gadget is closed, then the agent can traverse east to west and vice versa, and the agent can open the gadget from the north. If the crossover gadget is open, then the agent can traverse from east or west to* any *direction, while from the north and south the agent can only go to the north and south.*

Proof. If the gadget is closed and the agent comes from the south, the vertical crossbow trap will kill the agent. Obviously, the agent can open the gadget from the north by deactivating the vertical crossbow trap. If the gadget is open and the agent comes from the north or south, then the horizontal crossbow trap together with the spike traps prevents the agent to go east or west, but north and south are possible.

Independent of the gadget's state, the gadget can be traversed east to west or vice versa by the agent entering the horizontal crossbow's range, taking one step back, and then moving across. This also works for moving to the north or south, but only if the gadget is open, i.e., the vertical crossbow is *deactivated*. If the gadget is closed, the vertical crossbow is *functional*, which prohibits any turning onto the *north-south* path. ◀

Recall that the crossover gadget comes in all 4 rotations, see Figure 7. We now show how this gadget is used in our construction. Recall Figure 5. Where two paths cross, we insert a crossover gadget in one of the four rotations, but the choice of rotation is not arbitrary.

As seen in Figure 5, we have an ordering $x_1, x_2, \ldots, x_n$ on the set of variables in the 3SAT instance. The edge colors in Figure 5 indicate so-called *sections*: The i-th section consists of the two walks $w_{i,0}, w_{i,1}$ originating from the variable gadget for x_i, one corresponding to setting x_i to 0 (`false`) and the other to setting it to 1 (`true`): These walks touch all of the clause gadgets containing the literal they correspond to, and then go to the next variable gadget. For each intersection of two walks, it will be important from which direction the walks enter the intersection for the first time when traversed. To place the crossover gadgets, first consider intersections of walks in different sections: w_{i,b_i}, w_{j,b_j} for $i \neq j$, w.l.o.g. $i < j$. We place the crossover gadget so that the vertical crossbow trap is at the place where w_{j,b_j} enters the intersection for the first time when traversed. We claim that the agent can only first enter the variable gadget 1, then section 1, then the variable gadget 2, then section 2, and so on, until the agent has to traverse all clause gadgets and reaches the finish tile. This can be seen as follows. Clearly, the agent has to first enter variable gadget 1 and then section 1. The placement of the crossover gadgets guarantees that the agent cannot open any crossover gadget of section 1 with any section $j \neq 1$. Therefore, the agent must traverse to variable gadget 2 and enter section 2. By the same argument, in section 2 the agent cannot open any crossover gadget of section 2 with any section $j > 2$. However, the agent can open the crossover gadgets of section 2 with section 1. But the placement guarantees that the agent cannot enter section 1 from these crossover gadgets. Therefore, the agent has to enter variable gadget 3 and section 3, and so on.

When placing crossover gadgets at the intersection of $w_{i,0}$ and $w_{i,1}$, we take the orientation so that the vertical crossbow trap is at the place where $w_{i,0}$ enters the intersection for the first time when traversed. This makes sure that if the player chooses $x_i = 1$, then the agent only ever encounters east-west crossings of crossover gadgets, which means that the agent cannot open these gadgets and hence cannot traverse east or west, and if the player chooses $x_i = 0$, then the agent only ever encounters north-south crossings of crossover gadgets, and hence can open the gadgets, but not traverse east or west.

▶ **Theorem 7.3.** ASnS$\{h, s, v\}$ *is* NP*-complete.*

Proof. The proof is now obvious. We have seen that in order to reach the finish tile, the agent must traverse section 1, then section 2, and so on, and choose exactly one Boolean value for each variable. And after choosing the variable values, the agent can traverse the clause gadgets if and only if the agent touched each clause gadget. ◀

7.3 Full example

Figure 11 depicts the Advanced Spikes 'n' Stuff level built from the 3SAT instance $\phi = (x_1 \vee \neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3)$. Setting each of the variables x_1, x_2 and x_3 to `true` makes ϕ evaluate to `true`. It can be easily verified that if the player makes these choices at the variable gadgets, they are able to unlock both clause gadgets during their traversal, and are hence able to complete the level.

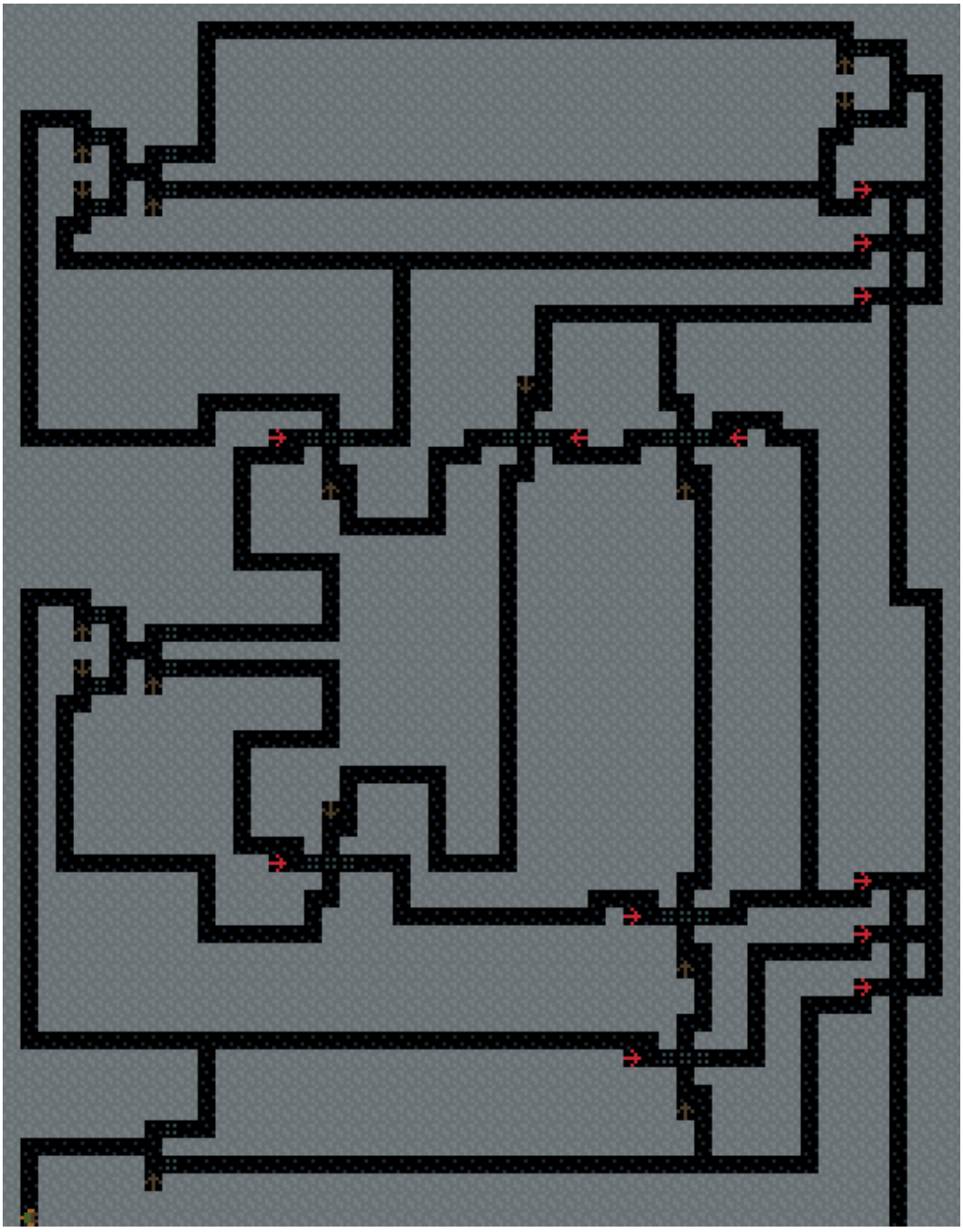


Figure 11 Advanced Spikes 'n' Stuff level built from $\phi = (x_1 \vee \neg x_1 \vee x_2) \wedge (x_1 \vee \neg x_2 \vee x_3)$. The situation is rotated by 90° from Figure 5. The variable gadges are on the west, x_1 to x_3 from south to north. A southern traversal sets the variable to `true`, a northern traversal sets it to `false`. The clause gadget for $(x_1 \vee \neg x_1 \vee x_2)$ is at the south east, and the clause gadget for $(x_1 \vee \neg x_2 \vee x_3)$ is at the north east.

References

1 Spikes 'n' Stuff. `https://draknek.itch.io/spikes-n-stuff`, `https://alan.draknek.org/games/puzzlescript/spikes-n-stuff.php`. Accessed: 25/02/2024.

2 Zeeshan Ahmed, Alapan Chaudhuri, Kunwar Shaanjeet Singh Grover, Ashwin Rao, Kushagra Garg, and Pulak Malhotra. Classifying Celeste as NP Complete. In *International Conference on Foundations of Computer Science & Technology, Chennai, India*, November 2022.

3 Greg Aloupis, Erik D Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo Games are (Computationally) Hard. *Theoretical Computer Science*, 586:135–160, 2015.

4 Sualeh Asif, Michael Coulombe, Erik D Demaine, Martin L Demaine, Adam Hesterberg, Jayson Lynch, and Mihir Singhal. Tetris is NP-hard even with $O(1)$ rows or columns. *Journal of Information Processing*, 28:942–958, 2020.

5 Lily Chung and Erik D. Demaine. Celeste is PSPACE-hard. *Thai Journal of Mathematics*, 21(4):671–686, December 2023.

6 Graham Cormode. The Hardness of the Lemmings Game, or Oh no, more NP-Completeness Proofs. In *Proceedings of Third International Conference on Fun with Algorithms*, pages 65–76, 2004.

7 Diogo M Costa, Alexandre P Francisco, and Luís Russo. Hardness of Modern Games. arXiv:2005.10506, 2020.

8 Erik D Demaine, Martin L Demaine, and Joseph O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *Proceedings of the 12th Annual Canadian Conference on Computational Geometry (CCCG 2000)*, pages 211–219, August 2000.

9 Erik D. Demaine, Joshua Lockhart, and Jayson Lynch. The Computational Complexity of Portal and other 3D Video Games. In *9th International Conference on Fun with Algorithms (FUN 2018)*, volume 100, pages 19:1–19:22, 2018.

10 Dorit Dor and Uri Zwick. SOKOBAN and other motion planning problems. *Computational Geometry*, 13(4):215–228, 1999.

11 Michal Forišek. Computational complexity of two-dimensional platform games. In *Fun with Algorithms: 5th International Conference, FUN 2010, Ischia, Italy, June 2-4, 2010. Proceedings 5*, pages 214–227. Springer, 2010.

12 Daniel Král, Vladan Majerech, Jiří Sgall, Tomáš Tichý, and Gerhard Woeginger. It is tough to be a plumber. *Theoretical computer science*, 313:473–484, 2004.

13 Giovanni Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54:595–621, 2014.

14 Giovanni Viglietta. Lemmings is PSPACE-complete. *Theoretical Computer Science*, 586:120–134, 2015.