Switches are PSPACE-Complete

Jonathan Gabor * Aaron Williams †

Abstract

Switches is a grid-based puzzle game invented by Jonathan Gabor and implemented using MIT's Scratch programming language in 2014. The puzzle is based on the 'switch' mechanism which allows the player to toggle the presence and absence of barriers by walking over a switch of the same color. At first glance the mechanism seems to be similar to previously studied video game mechanisms including pressure plates and doors, but it is in fact quite different. We prove that deciding if a Switches puzzle is solvable is PSPACE-complete and furthermore, this hardness result is true even when the puzzle is only $r = 3$ rows in height. On the other hand, we provide a polynomial-time algorithm for solving Switches puzzles with $r = 1$ row. The computational complexity of the problem with $r = 2$ is open.

1 Introduction

Switches is a puzzle game that was invented by Jonathan Gabor in 2014 while he was a high school student. The puzzle was implemented using MIT's visual programming language called Scratch [8]. This implementation is available online as Switches v2.1 `https://scratch.mit.edu/projects/33587070/`. A new implementation containing playable versions of every level discussed in this paper is available as Switches Remastered `https://scratch.mit.edu/projects/203220688/`.

The puzzle was designed to be played on an r-by-c grid, and each object is placed inside of a single cell. The player's goal on each level is to move their avatar from the start location to the goal location called the portal. The core mechanism involves *switches* and *doors*. Each door is independently on or off and a door is only a barrier to the player's movement when it is on. When a player steps on a switch, then the state of all doors of the same color are toggled. (The player toggles any switch they touch, and they must move to another before toggling it again.) There can be multiple switches of the same color, multiple doors of the same color in either state, and multiple colors that operate independently.

*Bard College at Simon's Rock, Massachusetts, `jgabor16@simons-rock.edu`

†Bard College at Simon's Rock, Massachusetts, `awilliams@simons-rock.edu`

A sample level and its solution are given in Figure 1, along with a legend of graphical symbols.

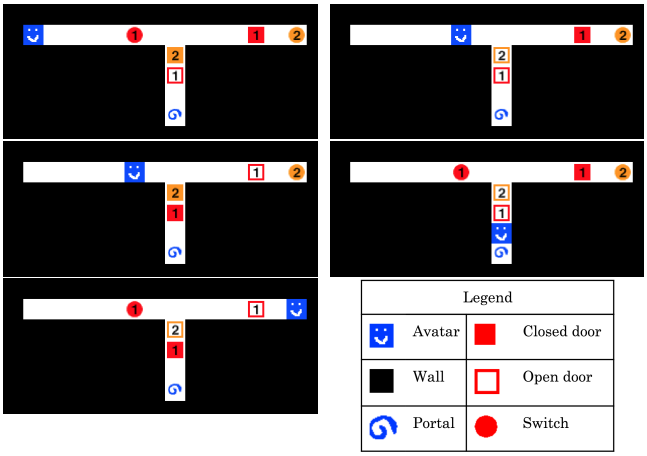

Figure 1: The player solves the 13-by-5 level by walking over the red switch, followed by the orange switch, and then by again walking over the red switch. The images should be read in column-major order.

At first glance, this mechanism may seem to be quite similar to previously studied video game mechanisms such the pressure plate mechanism that was examined by Viglietta [9]. More specifically, a switch behaves like an 'on' pressure plate combined with an 'off' pressure plate, so one might try to simulate pressure plates using switches. However, this is far more difficult than it sounds, because in all but a few cases, levels of switches are reversible. In other words, the player can return to any state they were previously in by performing the previously made moves in reverse[1] order. On the other hand, this is not true for the pressure plate mechanism. Similarly, the mechanisms and hardness results obtained by Aloupis, Demaine, Guo, and Viglietta [1] do not seem to apply to this puzzle. The authors are unaware of any previous puzzle that uses the switch mechanism, but it seems possible that such a puzzle could exist given the mechanism's simplicity.

The Switches decision problem takes a Switches level on an r-by-c grid as input, and the output is 'yes' or 'no'

[1]There is some subtlety to reversibility in Switches levels. If the players avatar is on a blank tile or an open door, then they can return to any previous state by reversing their previous moves. Furthermore, if they can reach a blank tile, they can return to any previous state. However, there are also cases where player can trap themselves.

depending on whether the level is solvable or not. We prove that this decision problem is PSPACE-complete by a reduction from True Quantified Boolean Formula (TQBF). More remarkably, we are able to show that the decision problem remains PSPACE-complete when restricted to levels that have at most $r = 3$ rows. This differentiates it from other PSPACE puzzle games such as Sokoban [2] and Rush Hour [3]. We refer the readers to Hearn and Demaine [6] for further results on the hardness of puzzles and games.

To complement our hardness results, we also prove that Switches puzzles with 1 row can be solved in polynomial-time. The complexity of the decision problem for Switch levels with 2 rows is presently unknown and is a compelling open problem.

The paper is structured as follows. In Section 2 we show how to determine if a level with $r = 1$ rows is solvable in polynomial-time. In Section 3 we show that Switches is NP-hard via a standard 3SAT reduction. In Section 4 we provide a level construction that forces the player to iterate through the well-known binary reflected Gray code. Section 5 then combines the results of Section 3 and 4 to obtain our main PSPACE-hardness result. Section 6 concludes with open problems.

To our knowledge our this article marks the first time that an original Scratch game has been proven to be NP-hard or PSPACE-hard. According to Wikipedia [10],"Scratch has influenced many other programming environments and is now considered a standard for introductory coding experiences for children." As a result, this paper shows that the 'fun' of computational complexity is not just for adults.

2 Polynomial-Time Algorithm for 1 Row

In this section we provide a polynomial-time algorithm for solving Switches levels with $r = 1$ row. Figure 2 gives examples of solvable and unsolvable levels with $r = 1$.

(a) This level is solvable by alternately moving left and right to flip switch 1, switch 2, left switch 4, right switch 4, switch 3, right switch 4, switch 5, switch 2, switch 4, and then moving to the portal.

(b) This level is unsolvable because the player can never reach switch 5 since one of the two 3-doors will always be closed.

Figure 2: Two similar levels with $r = 1$.

Throughout this section we can assume the portal appears on the rightmost square of a level without loss of generality. To understand this assumption, first notice that we can assume that the portal appears to the right of the player's initial position, since otherwise we can instead consider the mirror image of the level. Next notice that any squares to the right of the portal cannot be accessed by the player.

We start this section by considering clusters of adjacent switches, and then by showing how to manipulate their state. Then we consider levels in which the avatar starts on the leftmost square. Finally, we consider levels in which the avatar does not start on the leftmost square.

2.1 Clusters, Configurations, and Traversals

We define a cluster of switches (or simply a cluster) to be a maximal sequence of adjacent switches. In a level with 1 row, membership in a cluster is reflexive, symmetric, and transitive, so the switches partition uniquely into clusters. Figure 3 gives an illustration of clusters.

Figure 3: This level has three clusters.

A left-to-right traversal of a cluster is a sequence of moves in which the player starts at the square immediately to the left of the cluster, ends at the square immediately to the right of the cluster, and at no time moves outside of this region. The traversal ends when the player moves to the next square to the right. We similarly define a right-to-left, right-to-right, and left-to left traversal of a cluster.

When playing a Switches level each color is in one of two states which we call the color's current parity (or parity for short). A cluster containing switches with d distinct colors has 2^d different configurations based on the current parities of these colors. We now show that the player can set a cluster to any configuration during a left-to-right traversal.

Lemma 1 *Suppose the player is standing to the left of a cluster, and the square to the right of the cluster is either open or has the same color as a switch within the cluster. Then the player can set the cluster to any configuration during a left-to-right traversal. Furthermore, the number of steps is linear in terms of the length of the cluster.*

Proof. Suppose that the cluster contains k switches. For convenience let us number the squares from left-to-right starting at 0 from the player's position. In other words, we are focused on the squares numbered 0, 1, 2, . . . , k+1 where 0 and k+1 are immediately outside of the cluster. The pseudocode below uses 'L' and 'R' to denote left and right moves, respectively. The basic idea

is to set the switches to their desired parity from left-to-right. More specifically, if we are standing on square s and the switch on square s-1 has the wrong parity, then we move L then R to correct it, and continue. There are special cases to handle at the right side of the cluster, and we discuss those in more detail below.

```
R
for s = 1, 2, ..., k-1
  R
  if the switch on square s has the wrong parity
    LR
if the switch on square k has the wrong parity
  if square k+1 is a door with switch k's color
    LRRLLRR
  else
    RLR
else
  R
```

After each iteration of the for loop, the avatar will be one space farther to the right, and the color of switch to the players left will be in the correct parity. After completing the for loop, the players avatar will be on the rightmost switch. If this switch is in the correct parity, the player can simply exit the cluster by moving to the right. If it is in the incorrect parity the player will usually be able to fix this by moving right, and then left. Then the player can exit the cluster. Doing this will not affect the parity of any other color because the tile to the right of the player cannot be a switch. However, it is possible that there is a door of the last switchs color directly to the players right. In this case, the player can move left and then right twice (the first time the player moves right, they open the door). However, now the switch on square $k-2$ will be in the wrong parity. This can be fixed by moving left twice and then right twice. This will change the parity of switch $k-2$, but leave the parity of switch $k-1$ constant. □

If the player must end on the same side of the cluster they started on, they can simply move all the way to the left of the cluster, then follow the above algorithm ignoring the leftmost switch. Then, if this switch is in the wrong parity, the player can move all the way to the left, and all the way to the right to fix it.

Lemma 1 also applies to right-to-left and right-to-right traversals, respectively, so long as the player starts on the square to the cluster's immediate right.

2.2 Left-to-Right Levels

Now we focus on $r = 1$ levels in which the player starts on the leftmost square. We refer to these levels as left-to-right levels; when solving these levels we can prove that the player never needs to backtrack to a previously traversed cluster.

For each door, let $s(d)$ be the location of the rightmost switch of its color to its left.

Lemma 2 *A left-to-right level is solvable if and only if the following conditions both hold: If two doors have the same s(d), then they have the same parity; If $s(d)$ is undefined for a door, then that door is initially open.*

Proof. We begin by proving the forward direction. Consider the first point. Obviously, when changing the parity of a door d, the player must either be to the left of location $s(d)$ or to the right of that door. To proceed to the right, they must make that door open. Therefore, to proceed to the right of two doors with the same s(d), they must make both of them open when at position $s(d)$. This is only possible if they have the same parity.

Now consider the second point. If $s(d)$ is undefined, then there is no switch to the left of that door. Then if it is initially closed, the player cannot change its parity, until they go to the right of it, but they cant go to the right of it until they change its parity.

Now consider the reverse direction. We claim that any level satisfying the two points above can be solved using the following linear time algorithm.

Let a clusters ideal configuration be the configuration such that for each switch in it, if that switch is at location $s(d)$ for some door d, the switch is in the parity such that door d is open.

Moving from left to right, we adjust each cluster to its ideal configuration. The only way this algorithm could fail is if the player encounters a closed door which prevents them from traveling farther the right. However, such a door must have a defined $s(d)$. Then it must have been made open. Therefore, encountering a closed door is impossible. □

The algorithms in Lemma 1 runs in linear time in terms of the length of the cluster. Because each cluster is only adjusted once, and the total length of all the clusters is capped by the length of the level, this algorithm runs in linear time.

Theorem 3 *Any solvable level with $r = 1$ row can be solved in polynomial time.*

Proof. Let L_0 be the location of the portal. Let L_{n+1} be the leftmost location the player must reach before reaching location L_n if n is even, and the rightmost such location if n is odd (usually this location will be a switch of the color of a closed door blocking location L_n).

The player can travel from L_{n+1} to L_n in a linear amount of time by Lemma 2.

We will now demonstrate that L_{n+2} is always in between L_{n+1} and L_n (inclusively). Without loss of generality assume that n is even. Because L_{n+2} is the rightmost location the player must reach before L_{n+1}, it is to

the right of L_{n+1} (inclusively). Since L_n is the rightmost location the player must reach before reaching location L_{n-1}, and the player must visit L_{n+2} before visiting L_n, L_{n+2} must to the left of L_n (if it was to the right, then it would be the rightmost location before visiting L_{n-1}).

It follows that if $L_n = L_{n+1}$, for all $m > n$, $L_m = L_n$. Then there is some k which is the lowest value such that $L_k = L_{k+1}$. Then, all L_a with $a < k$ must be distinct. Because there are only a linear number of locations in the level, and moving between each requires a linear amount of time, the level can be solved in $O(n^2)$ time. □

3 NP-Hardness

In this section we prove that the Switches problem is NP-hard by a reduction from 3SAT.

Suppose that we are given an instance of 3SAT ϕ with clauses $c_1, c_2, \ldots, c_m$ and variables $x_1, x_2, x_3, \ldots, x_n$. We construct a Switches level $S(\phi)$ that has 3 rows and $n + 2m + 4$ columns. The level $S(\phi)$ uses n colors in total and there is a single switch of color i. The number of initially open or closed doors of color i is given by the number of positive or negative x_i literals in ϕ, respectively.

The level is organized as follows. The Avatar starts on the left side of the level and to their right is a variable corridor of height 1 and width n+1. The variable corridor contains a variable cluster which is a cluster containing one switch of each color . This corridor leads into a room of height 3 and width 2m+1 called the clause room. Every second column in the clause room is blank, and between these blank columns are columns associated with each of the clauses. Each clause column consists of three doors in a vertical line. The colors of the doors are given by the variable of the literal in the associated clause, and these doors are initially open or closed based on whether the said literals are true or false. The portal is located to the right of the clause room. Figure 4 illustrates this construction.

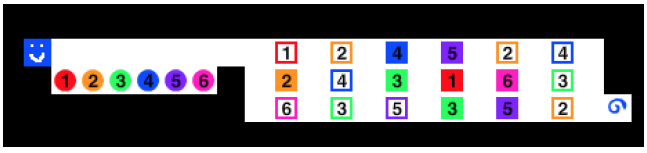

Figure 4: $S(\phi)$ for $\phi = (x_3 \vee \neg x_4 \vee x_5) \wedge \ldots$.

Theorem 4 *Switches is NP-hard.*

Proof. Given an instance of 3SAT ϕ we construct the level $S(\phi)$ as described. Suppose that ϕ is satisfiable by an assignment A which sets variable x_i to a_i for all $1 \leq i \leq n$. By the results of Section 2 the Avatar can perform a left-to-right traversal of the variable cluster with the following property: The switch of color i is switched an even number of times if $a_i = True$ and and odd number of times if $a_i = False$. Now consider a given clause $c_i = (\ell_u \ell_v \ell_w)$ where ℓ_u, ℓ_v, and ℓ_w are either positive or negative literals of variables x_u, x_v, x_w, respectively. Observe that the ith clause column is traversable if and only at least one of its three doors are open. Also recall that the doors associated with positive literals start open, and doors associated with negative literals start closed within $S(\phi)$. Since clause c_i is satisfied the assignment A, it must be that the Avatar's left-to-right traversal of the variable cluster results in the ith clause column being traversable. Therefore, the Avatar can traverse the entire clause room and reach the portal. Therefore, if ϕ is satisfiable, then $S(\phi)$ is solvable.

Suppose that $S(\phi)$ is solvable and consider a particular solution. When the Avatar reaches the portal let $p_i \in \{0, 1\}$ denote the parity of the number of times that switch i was switched during this solution. That is, $p_i = 0$ if switch i was switched an even number of times during the solution, and $p_i = 1$ if switch i was switched an odd number of times during the solution. We construct an assignment A for ϕ as follows: $x_i = True$ if $p_i = 0$, and $x_i = False$ if $p_i = 1$. Due to the structure of the $S(\phi)$ level, when the Avatar reaches the portal, it must be that each one of the clause columns is traversable. Therefore, in each clause in ϕ there must be at least one literal that evaluates to true with respect to assignment A. Therefore, ϕ is solvable.

The reduction is completed by noting that the size of $S(\phi)$ is polynomially bounded by the size of ϕ. □

4 Exponentially Long Levels

In this section we construct Switches levels that require an exponential number of moves to solve. More specifically, the levels contain n distinct colors, and the level forces the player to iterate over all 2^n different states or parities for these colors.

The binary reflected Gray code was previously used in a similar manner in a paper by Greenblatt, Kopinsky, North, Tyrrell, and Williams [5] for the puzzle game MazezaM. The presentation here closely resembles a similar section in that paper.

4.1 Binary Reflected Gray Code

Let $\mathcal{B}(n)$ be the set of n-bit binary strings. The *weight* of $b_1 b_2 \cdots b_n \in \mathcal{B}(n)$ is its bitwise sum $\sum_{i=1}^{n} b_i$. We use exponents to denote bitwise concatenation. For example, $1^4 = 1111$ is the only string of weight four in $\mathcal{B}(4)$.

The *binary reflected Gray code* (BRGC) is an ordering of $\mathcal{B}(n)$ attributed to Gray [4]. In the order each pair of consecutive strings have Hamming distance one (i.e.

they differ in exactly one bit). The order starts with 0^n and ends with $0^{n-1}1$. The BRGC for $n = 4$ is below with overlines showing the bit that changes to create the next string:

$$\bar{0}000, 1\bar{0}00, \bar{1}100, 01\bar{0}0, \bar{0}110, 1\bar{1}10, \bar{1}010, 001\bar{0},$$
$$\bar{0}011, 1\bar{0}11, \bar{1}111, 01\bar{1}1, \bar{0}101, 1\bar{1}01, \bar{1}001, 0001.$$

Now we explain how to create each successive string in the BRGC starting from the initial string 0^n.

Definition 1 *Each $b_1b_2\cdots b_n \in \mathcal{B}(n)$ has up to two* active bits*: (a) its leftmost bit b_1, and (b) its bit immediately to the right of its leftmost* 1*.*

For example, the leftmost 1 in $b_1b_2b_3b_4b_5b_6 = 000111$ is $b_4 = 1$; therefore, its active bits are b_1 and b_5. Every binary string has two active bits except 0^n and $0^{n-1}1$.

The following theorem is well-known (see Knuth [7]).

Theorem 5 *If $b_1b_2\cdots b_n$ has even weight, then complementing active bit (a) gives the next string in the BRGC. Otherwise, if $b_1b_2\cdots b_n$ has odd weight, then complementing active bit (b) gives the next string.*

On the other hand, complementing the 'other' active bit of $b_1b_2\cdots b_n$ gives the previous string in the BRGC.

For example, 000111 has odd weight, so 000101 is the next string in the BRGC and 100111 is the previous.

4.2 Gray Code Level

Now we construct a level **Gray**(n) based on the BRGC for n-bit binary strings. The construction is illustrated in Figure 5. Remarkably, **Gray**(n) has only $r = 3$ rows $c = 2n + 1$ in general.

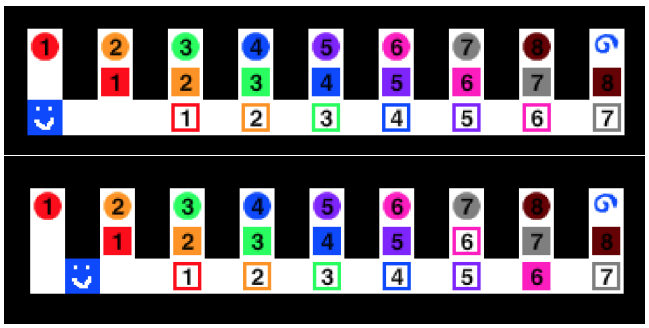

Figure 5: The level **Gray**(8) in state $b_1b_2\cdots b_8 = 00000000$ (top) and $b_1b_2\cdots b_8 = 00000100$ (bottom). In the latter case observe that only those columns with switches 1 and 7 at the top are accessible as per Theorem 5.

The level a corridor along the bottom row, and then $n+1$ columns protruding upwards from the bottom row. The tops of these columns include switches for colors $1, 2, \ldots, n$ and the portal, respectively. The switch for color 1 is not protected, and the column with switch i at the top is protected by a door of color $i - 1$ which is initially on. Furthermore, the corridor below the column with switch i at the top is protected by a door of color $i - 2$ which is initially off.

We associated a binary string $b_1b_2\cdots b_n$ with the state of **Gray**(n)'s switches that is initially $00\cdots0$. Due to the structure of the level and Theorem 5 we have the following theorem.

Theorem 6 *To complete* **Gray**(n) *the player must iterate over all 2^n states of* **Gray**(n) *according to the binary reflected Gray code.*

5 PSPACE Hardness

In this section we prove that Switches is PSPACE-hard. We do this by combining the results of Sections 3 and 4 to create a level $Q(\phi)$ that models a True Quantified Boolean Formula (TQBF) ϕ. More specifically, we model the unquantified Boolean 3SAT formula within ϕ as in Section 3, and then we force the player to iterate overall all possible 2^u states of the u universally quantified variables according Section 4.

When creating the level we utilize the fact that the leftmost bit is changed every second time in the binary reflected Gray code. We leverage this fact by horizontally separating the leftmost bit and the remaining bits in the level, and placing the 3SAT construction between them. In other words, we force the player to traverse the 3SAT formula (from left-to-right or right-to-left) every time they wish to change one of the quantified variable bits. Thus, the player must ensure that the 3SAT formula is satisfied before they can make progress in the higher-level problem that is iterating through the binary reflected Gray code.

A sample construction involving $n = 8$ variables is shown in Figure 6. Due to width restrictions we illustrate the sample level with 9 rows; the taller rows can easily be turned (at the expense of making the level much wider) to create a level with 3 rows.

Theorem 7 *The Switches decision problem is PSPACE-hard even when restricted to levels with $r = 3$ rows.*

Finally, we prove membership in PSPACE below.

Lemma 8 *The Switches decision problem is in PSPACE.*

Proof. Consider an r-by-c level with d distinct colors of switches. There are at most $2^d \cdot rc$ possible states for this level, where 2^d counts the number of different states for the colors, and rc is an upper bound on the number of

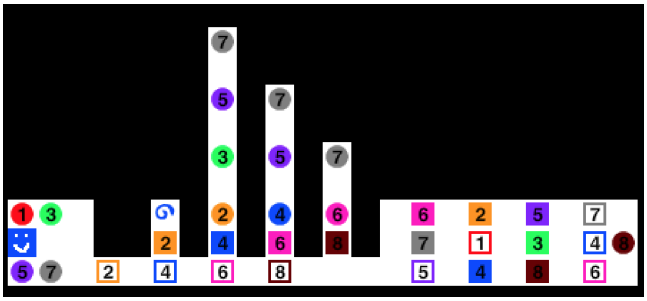

Figure 6: Level $P(\phi)$ which models the TQBF formula ϕ with quantified variables $\exists x_1 \forall x_2 \exists x_3 \forall x_4 \exists x_5 \forall x_6 \exists x_7 \forall x_8$. The bottom of the level contains clause gadgets including the rightmost clause $x_7 \vee x_4 \vee x_6$.

locations for the player's avatar. Therefore, we can represent an individual state of the level with $d \cdot log_2(rc)$ bits. Similarly, $d \cdot log_2(rc)$ bits bits are sufficient for counting from 0 to $2^d \cdot rc - 1$. Therefore, we can now establish membership in NPSPACE by nondeterministically moving the avatar in one of the four cardinal directions, and it keeping a counter for the number of times we have done this. If the avatar reaches the portal, then we stop the algorithm and answer yes. Otherwise, once the counter exceeds the number of possible states then we terminate the algorithm and answer no. If there is a solution to the level, then there will be at least one path through the computation that answers yes. Since we used only $d \cdot log_2(rc)$ bits of storage, this establishes membership in NPSPACE, and by Savitch's theorem, PSPACE. □

Corollary 1 *The Switches decision problem is PSPACE-Complete.*

6 Final Remarks

In this paper we investigated the computational complexity of the Switches puzzle game. There are a number of interesting open problems:

- What is the computational complexity of solving Switches levels with r = 2 rows?
- Our PSPACE-hardness reduction uses an arbitrarily large number of colors. What is the computational complexity if the number of colors is a constant?
- Our PSPACE-hardness reduction uses an arbitrarily large number of doors per color. What is the computational complexity if each doors color can be used only a constant number of times?
- Our PSPACE-hardness reduction uses up to two switches per color. What is the computational complexity if no two switches have the same color?
- Are there any other geometric puzzle games created on Scratch that are NP-hard or PSPACE-hard?

Regarding the 2-row case, there are several reasons to think that it should be solvable in polynomial time. First, when there are two rows, if the player's avatar is in the same column as a square, then the player can immediately reach that square. This means that the player cannot "pass by" an object while being unable to interact with it, something essential to constructing exponential orderings. Second, with only two rows, there is no obvious way to implement clause gadgets. Third, if there are no switches between 2 points in a level, determining whether there exists a set of parities to travel between those points can be done in polynomial time, since it can be reduced to a 2-sat problem.

We also mention that the original Switches implementation has an additional dual switch mechanism in which two switches are toggled simultaneously. We did not require its use to establish PSPACE-hardness, and its inclusion does not change the problem's inclusion in PSPACE.

References

[1] G. Aloupis, Erik D. Demaine, A. Guo, and G. Viglietta. Classic Nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015.

[2] J. Culbertson. Sokoban is PSPACE-complete. In *Fun With Algorithms (FUN 1998)*, Lecture Notes in Computer Science, pages 65–76. Carleton Scientific, 1998.

[3] G. W. Flake and E. B. Baum. Rush hour is PSPACE-complete, or "why you should generously tip parking lot attendants". *Theor. Comput. Sci.*, 270:895–911, 2002.

[4] F. Gray. Pulse code communication. *U.S. Patent 2,632,058*, 1947.

[5] A. Greenblatt, J. Kopinsky, B. North, M. Tyrrell, and A. Williams. MazezaM levels with exponentially long solutions. In *20th Japan Conference on Discrete and Computational Geometry, Graphs, and Games (JCDCGGG 2017)*, page 2 pages, 2017.

[6] Robert A. Hearn and Erik D. Demaine. *Games, Puzzles, and Computation*. A. K. Peters, Ltd., Natick, MA, USA, 2009.

[7] D. E. Knuth. *The Art of Computer Programming*, volume 4: Generating All Tuples and Permutations. Addison-Wesley, 2005.

[8] Scratch. About scratch, 2018. URL: `https://scratch.mit.edu/about`.

[9] G. Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54(4):595–621, 2014.

[10] Wikipedia. Scratch (programming language), 2018. URL: `https://en.wikipedia.org/wiki/Scratch_(programming_language)`.