Computational complexity of two-dimensional platform games

Michal Forišek

Comenius University, Bratislava, Slovakia
forisek@dcs.fmph.uniba.sk

Abstract. We analyze the computational complexity of various two-dimensional platform games. We identify common properties of these games that allow us to state several meta-theorems: general constructions that allow us to identify a class of these games for which the set of solvable levels is NP-hard, and another class for which the set is even PSPACE-hard. Notably CommanderKeen is shown to be NP-hard, and PrinceOfPersia is shown to be PSPACE-complete.
We then analyze the related game Lemmings, where we show that an assumption made by Cormode in [3] is false. We construct a set of instances which only have exponentially long solutions. Thereby we invalidate Cormode's proof that the general version of the Lemmings decision problem is in NP. We then augment our construction to only include one entrance, which makes our instances perfectly natural within the context of the original game.

1 Overview

The area of two-player combinatorial games, and one-player games and puzzles has already been well researched. The earliest mathematical results are more than a century old (e.g., the analysis of the game *NIM*[1] [1]). The most important mathematical results related to two-player combinatorial games are the Sprague-Grundy theory [18, 8] and Conway's surreal numbers [2].

In past few decades researchers returned to this area with a new point of view: investigating the computational complexity of these puzzles and games. It turned out that almost all "interesting" puzzles and games are hard – assuming that $P \neq NP$ there is no efficient algorithm to solve/play them optimally. Among the most important results are the proofs that many of the two-player games are EXPTIME-complete (e.g., the generalizations of *Go* and *Checkers* to an $n \times n$ board, [15, 16]), and many single-player puzzles are either PSPACE-complete (e.g., solving *Sokoban* [4]) or NP-complete (e.g., checking whether a *Minesweeper* configuration is valid [10]). For more of these results, we recommend one of the survey papers [11, 6, 7].

[1] A convention for easier reading of this article: Throughout the entire article, italics (*SomeGame*) are used for the names of actual games, and small capitals (SomeGame) are used for the names of the underlying decision problems.

In Section 2 of this article we analyze the computational complexity of a class of previously ignored single-player games: two-dimensional platform games.

One of the already researched games is the game *Lemmings*, initially addressed by Cormode [3] and later by Spoerer [17]. In Section 3 we continue in the analysis of this game, disproving an assumption made by Cormode, and thereby invalidating his proof that LEMMINGS is in NP.

2 Hardness of 2D platform games

When researching the computational complexity of a given puzzle, we usually expect the puzzle to be hard – after all, the puzzles are designed with the goal to challenge the solver and to push her cognitive processes to the limit. However, when we turn to computer games, not all of them are puzzles. Many other games are perceived as simple in terms of necessary thinking. An example of such a set of games are the 2-dimensional platform games. In this section we will take a closer look at the computational complexity of these games, and surprisingly we will show that many of these games are actually very difficult to solve in general.

For many platform games we will prove that the set of all solvable instances (levels) is difficult: in some cases NP-hard, in some cases even PSPACE-hard. We will now define 2-dimensional platform games, identify a set of common features they exhibit, and then prove that some subsets of these features imply that solving the game is difficult.

A 2-dimensional platform game is a single-player game in which the player sequentially solves a set of levels. Each level is represented by a 2-dimensional map representing a vertical slice of a virtual world.[2] The player's goal is to move her avatar (i.e., the game character controlled by her) from its starting location into the designated final location. The player controls the avatar by issuing simple commands (step, jump, climb up/down, etc.), usually by pressing keys or buttons of an input device. Within the game world, the avatar is affected by some form of gravity. As a consequence the maximum jump height is limited.

Additionally, many of these games share the following features:

- **long fall:** The height of the longest safe fall (that does not hurt the avatar) is larger than the maximum jump height.
- **opening doors:** The game world may contain a variable number of doors and suitable mechanisms to open them.
- **closing doors:** The game world contains a mechanism to close doors, and a way to force the player to trigger such a mechanism.
- **collecting items:** The game world contains a set of items that have to be collected.
- **enemies:** The game contains enemy characters that must be killed or avoided in order to solve the level.

We will now show that some of these features are easy from the algorithmic point of view, but others imply that solving the game automatically is hard

[2] The world usually consists of horizontal platforms, hence the name "platform game".

– regardless of other details of the particular game. These results can be seen as "meta-theorems": for any particular game we can take the construction and adjust it to the details of the particular game.

Meta-theorem 1. *A 2D platform game where the levels are constant and there is no time limit is in P, even if the* ***collecting items*** *feature is present.*

Proof. The level and the set of allowed movements of the avatar uniquely determine a directed reachability graph. In this graph we label the vertices that correspond to locations of items. Additionally, we execute two depth-first searches to determine the set of vertices that are both reachable from the entrance and allow us to reach the exit. If there is a labeled vertex that is not in this set, we reject. Otherwise we drop the vertices that are not in the reachable set and then contract each strongly connected component into a single vertex, preserving labels. We accept iff all the labeled vertices in the resulting DAG lie on a single path from the entrance to the exit. This can be verified by considering the labeled vertices in topological order and verifying reachability for each adjacent pair. The algorithm is linear in the instance size if implemented properly.

Meta-theorem 2. *A 2D platform game where the* ***collecting items*** *feature is present and a time limit is present as a part of the instance is NP-hard.*

Proof. Itai et al. in [9] prove that the set of grid graphs with Hamilton cycles is NP-complete. This problem can be reduced to solving our platform game as follows: Given a grid graph G, locate the leftmost of its topmost vertices and call it u. The vertex u has degree at most 2. If this degree is less than 2, we reject. (I.e., produce an unsolvable instance of our game and terminate.) Otherwise, let v be the right neighbor of u. Obviously the edge uv must be a part of every Hamilton cycle in G. Hence there is a Hamilton cycle in G iff there is a Hamilton path that starts at u and ends at v. We now design the level in such a way that the map of the level corresponds to G. We place an item into each location that represents a vertex of G, except for u and v. We place the level entrance at u, the level exit at v, and set the time limit to $\alpha(|V_G| - 1)$, where α is the time needed to travel between any two adjacent locations.

Meta-theorem 3. *Any 2D platform game that exhibits the features* ***long fall*** *and* ***opening doors*** *is NP-hard.*

Proof. We will show how to reduce 3-Cnf-Sat to such a game. The main idea of the proof is that the door opening mechanism can be used for a non-local transfer of information. Given an instance of 3-Cnf-Sat, we can construct an instance of the game as follows:

The instance will be divided into two parts: the key part, where the avatar starts, and the door part it reaches after exiting the key part. The door part will be a vertical concatenation of door gadgets (Figure 1 left), each corresponding to a single clause in the 3-Cnf-Sat instance. The door gadget for the clause $c_n \equiv (l_{n,1} \vee l_{n,2} \vee l_{n,3})$ contains three doors labeled (n, l_1), (n, l_2), and (n, l_3).

The key part will be a vertical concatenation of variable gadgets (Figure 1 right). Each variable gadget will correspond to a single variable used in the 3-Cnf-Sat instance. For the variable x_i the keys represent mechanisms that unlock doors. The keys in the left part unlock the doors (n, x_i) for all n, and the keys in the right part unlock the doors $(n, \neg x_i)$ for all n.

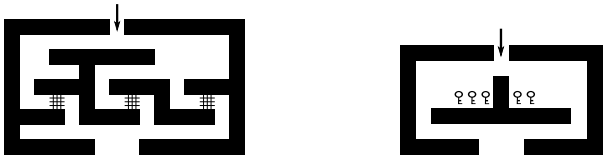

Fig. 1: The door gadget (left) and the variable gadget (right).

It is obvious that each valid solution of the 3-Cnf-Sat instance corresponds to a valid way how to traverse the level. On the other hand, if the vertical unit size of our gadgets exceeds the maximum jump height, we can easily verify that the player has no other reasonable choice. Finally, the game level can easily be created from the 3-Cnf-Sat instance in polynomial time, hence this is a valid reduction.

Note 1. Both gadgets in Figure 1 were designed so that no jumping is necessary anywhere in the final instance. Hence Meta-theorem 3 does also apply to games that do not include any jumping. For example, it can be extended to dungeon exploration games that involve one-way trapdoors to lower levels.

Corollary 1. *The following famous 2D platform games are NP-hard:*

Commander Keen, *Crystal Caves*, *Secret Agent*, *Bio Menace* (switches that activate moving platforms), *Jill of the Jungle*, *Hocus Pocus* (switches that toggle walls), generalized[3] *Duke Nukem* (keycards), *Crash Bandicoot*, *Jazz Jackrabbit 2* (crates that activate sets of new floor tiles once broken).

2.1 Prince of Persia is PSPACE-complete

In this section we state our main meta-theorem. In this case, instead of a generic proof we opt to present the construction for the popular platform game Prince of Persia.

Meta-theorem 4. *Any 2D platform game that exhibits the features* ***long fall***, ***opening doors*** *and* ***closing doors*** *is PSPACE-hard.*

Proof. The proof is a straightforward generalization of the proof presented below.

We will now define the PrinceOfPersia decision problem. An instance of this problem is a 2-dimensional map of the level, with some additional information. On the map, each cell contains one of a fixed set of tiles. For our

[3] The original only has keycards of 4 colors and it is in P. From Meta-theorem 1 it follows that the set of perfectly solvable Duke Nukem levels is in P as well.

construction, we need the following types: {nothing, entrance, exit, floor tile, floor tile with a pressure plate, door}.[4] Additionally, the doors have unique labels from some set $\mathcal{D}$, and each pressure plate is assigned a single label: "$+d$" if it opens door d, "$-d$" if it closes the door d. Multiple plates may open/close the same door.

The avatar (called Prince) can move in the following ways:[5] single step left /right; jump over at most three empty tiles left/right; climb to a floor tile diagonally above, if the above tile is empty; and descend into a pit ≤ 2 tiles deep.

Theorem 1. *The set* PrinceOfPersia *of solvable instances of the game defined above is PSPACE-hard.*

Proof. To prove that PrinceOfPersia is PSPACE-hard, we reduce the word problem for linear bounded automata WordLBA to our problem.

Consider an instance (M, w) of WordLBA with $M = (Q, \Sigma, \Gamma, \delta, q_0, L, R, F)$ and $|w| = n$. W.l.o.g. we may assume that the input and work alphabets are binary, i.e., $\Sigma = \Gamma = \{0, 1\}$. Q is the set of states, $F \subseteq Q$ is the set of final states. The symbols L and R are the left and right endmarker, respectively, δ is the non-deterministic transition function, and w is the input word.

We will now construct an instance of PrinceOfPersia that will be solvable if and only if M accepts w. The general layout of our instance is shown in Figure 2. The instance will consist of several gadgets. The gadgets will be designed in such a way that the only allowed way in which the avatar will be able to navigate the level will correspond to arrows in Figure 2.

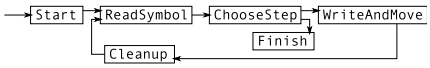

Fig. 2: Layout of gadgets in the PrinceOfPersia instance.

Our instance will contain several sets of doors that will represent various parts of the given word and LBA:

- the "accept" door a,
- two sets of "head location" doors $\{h_i, \overline{h_i} \mid 0 \leq i \leq n+1\}$,
- a set of "stored symbol" doors $\{s_{i,j} \mid 1 \leq i \leq n,\ j \in \{0, 1\}\}$,
- a set of "δ-function state" and "δ-function symbol" doors: $\{q_\pi, x_\pi \mid \pi \in \delta\}$,
- a set of "head movement" doors $\{m_{i,j} \mid 0 \leq i \leq n+1,\ -1 \leq j \leq 1\}$,
- and a set of "write symbol at location" doors $\{w_{i,j} \mid 1 \leq i \leq n,\ j \in \{0, 1\}\}$.

The gadgets shown in Figure 2 will be constructed in such a way that the following invariant holds for each x: Let S be the set of doors open at the moment when the avatar enters the ReadSymbol gadget for the x-th time. Then there is a configuration $(q, \overline{w}, k)$ of M such that:

- $(q, \overline{w}, k)$ is reachable in $x - 1$ steps from $(q_0, w, 1)$,

[4] The actual game includes other tiles as well, such as walls and spikes.

[5] Again, the description is simplified, but the differences do not matter for our proof.

– out of the head location doors, only doors h_k and $\overline{h}_k$ may be open,
– out of the stored symbol doors, only doors $s_{i,\overline{w}_i}$ may be open, for all i,
– out of the state doors q_π, only doors that correspond to q may be open,
– all other doors, including the accept door a, are closed.

From this invariant it immediately follows that the only way in which the avatar can solve the level is by simulating an accepting computation of M. The only thing left to prove is to show how to construct the individual gadgets that will enforce the above invariant.

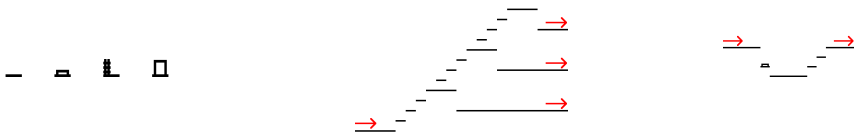

Fig. 3: Left: the set of four tiles we use – floor, pressure plate, door, and entrance/exit. Center and right: helper gadgets: multiple choice and forced activation of a plate.

Figure 3 shows the set of tiles we will use in our constructions, and the construction of two helper gadgets. The one in the center (denoted MC below) forces the avatar to choose one of the available paths, without the option to return later and change the choice. The right one (denoted FA(x)) forces the avatar to activate the pressure plate $-x$. This gadget will be used to enforce closing doors. Without the pressure plate this gadget can be used as a simple one-way wire (denoted by an arrow outside of a box). Using these helper gadgets we will now construct the gadgets in Figure 2.

Three of the gadgets are simple. The Start gadget contains pressure plates that open doors corresponding to the configuration $(q_0, w, 1)$. The Finish gadget contains the door a, blocking the way to the level exit. The Cleanup gadget consists of a series of FA gadgets that close all of the doors $m_{i,j}$, $w_{i,j}$, and x_π.

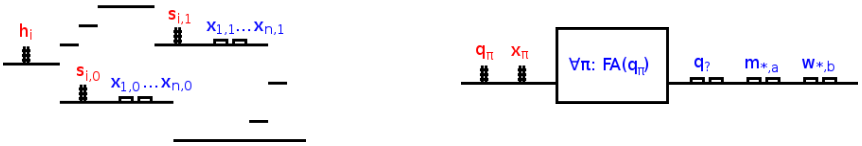

Fig. 4: Gadget parts: ReadSymbol (left), ChooseStep (right)

The ReadSymbol gadget contains a MC gadget with $n + 2$ outputs. Each of these outputs starts with one of the h_i doors. The rest of the gadget for one h_i door is shown in Figure 4 on the left. (Doors h_0 and h_{n+1} are special, their parts do not contain the symbol doors.)

Clearly, if the avatar wants to traverse this gadget, it must follow a path that corresponds to the LBA's current head location and the symbol underneath the head. On this path, the avatar is allowed to open any or all of those x_π doors that correspond to the current symbol.

The ChooseStep gadget contains a MC gadget with $|\delta|$ outputs, each of them corresponding to one element $\pi \in \delta$. When crossing this gadget the avatar is

forced to choose one of the (possibly multiple) paths that correspond to elements of δ for the LBA's current state and read symbol. If the new chosen state is final, the avatar is allowed to open the door a. Otherwise the avatar is forced to enter a gadget shown in Figure 4 on the right. When crossing this gadget the avatar is forced to close all open q_π doors, and then allowed to open those doors q_π, $m_{i,a}$ and $w_{i,b}$ that correspond to the new state, symbol b to write, and head movement a.

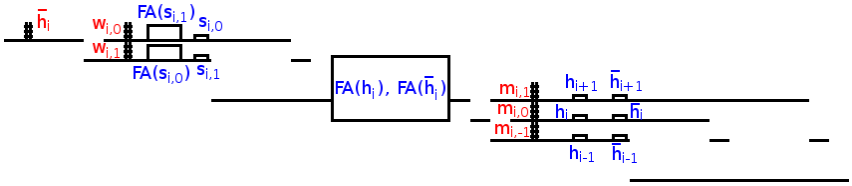

Fig. 5: Gadget part: WriteAndMove

The WriteAndMove gadget contains a MC gadget with $n+2$ outputs. Each output starts with one of the $\overline{h_i}$ doors, which forces the avatar to pick a path corresponding to the old head location. The rest of the gadget for a single output is shown in Figure 5.

In the first part (omitted for head positions 0 and $n+1$) the avatar crosses the correct write symbol door $w_{i,b}$, closes the door $s_{i,1-b}$ and opens $s_{i,b}$.

In the second part the avatar closes the current head location doors h_i and $\overline{h_i}$, crosses the correct movement door $m_{i,a}$, and opens the new head location doors h_{i+a} and $\overline{h_{i+a}}$.

That concludes the construction. An example of a full instance constructed in this way is in the appendix.

Note 2. In the original *Prince of Persia* game, there is an additional parameter of the instance: the amount of steps T after which an open door starts to close. Our construction can easily be modified to include this parameter, and it is even possible to set T to a small value approximately equal to the map size by adding a "refresh" block where the avatar can re-open any currently open door.

Corollary 2. PrinceOfPersia *is PSPACE-complete.*

Proof. Follows from Theorem 1, Savitch's theorem that NSPACE=PSPACE, and the obvious fact that PrinceOfPersia is in NSPACE.

3 Hardness of the Lemmings problem

One particular 2D game that is in many ways similar to our 2D platform games from the previous section is the game Lemmings. Already in 1998 McCarthy [13] mentions this game as a challenge for artifical intelligence. In [3] Cormode defines the Lemmings decision problem and argues that this problem is NP-complete. This claim is later repeated in [17] and it is conjectured that a variation of this problem is NP-complete as well.

However, Cormode's proof only holds for a restricted version of the LEMMINGS problem. This restriction is introduced on page 3 of [3] where an instance is defined. One element of the instance is the time limit, which Cormode defines as follows: "*limit*, the time limit, in discrete time units. For technical reasons, we will insist that the time limit is bounded by a polynomial in the size of the level. We conjecture that this restriction is unnecessary, by claiming that if it is not possible to complete the level in time polynomial in the level size, then it is not possible to complete the level at all."

The "technical reasons" are, in fact, just one reason: Cormode's proof that LEMMINGS $\in$ NP is trivial – guess the solution and verify it. This approach obviously fails if the number of theoretically possible configurations is not polynomial in the input size. Hence Cormode needs the restriction to artifically limit the number of reachable configurations.

In this section, we show that Cormode's conjecture is false by constructing a class of solvable LEMMINGS instances of increasing sizes such that their shortest solution length can not be bounded by a polynomial in the input size. We then conclude that the general LEMMINGS problem is NP-hard, as proved by Cormode, but so far we cannot tell whether it is in NP. The existence of instances with only exponentially long solutions leads us to conjecture that the general version of this decision problem is in fact not in NP.

3.1 Constructing the instances

In this section we construct a class of LEMMINGS instances for which the length of the shortest correct solution is not polynomial in the input size. Our construction is based on the idea of lemming synchronization – in order to solve the level, events in different places will have to happen at the same time. And for our instances, a suitable moment for this will only occur after an exponential number of steps has elapsed.

To construct our instances, we will only need a very small subset of the Lemmings universe: entrances, permeable walls, and an exit. The only lemming skill we will use will be the digger.

We will build two types of gadgets: a release gadget, that will only be able to release a lemming in certain points in time, and a synchronization gadget that must be reached by all lemmings in (almost) the same moment in time. The release gadget is shown in Figure 6 on the left, and the synchronization gadget is shown in Figure 6 on the right.

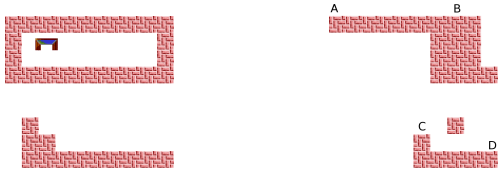

Fig. 6: The release gadget (left) and the synchronization gadget (right).

Lemma 1. *Let the release gadget be x steps wide. Assuming that no other lemming can reach the gadget, the lemming inside must start digging at a time that is an integer multiple of $2x$ in order to survive.*

Proof. At time 0 the entrance releases a lemming, at time 1 the lemming reaches the ground and starts walking towards the right. Without any interaction the lemming will keep on bouncing off walls, and reenter the same state every $2x$ ticks. In order to save it we have to give it the digger skill sooner or later. The bottom part of the gadget is in such a height that the lemming must dig in the leftmost place, otherwise the fall will kill it. Additionally, the lemming must be walking in the correct direction (left to right), otherwise it falls to its death immediately after reaching the bottom part of this gadget. This configuration will appear precisely at ticks that are positive integer multiples of $2x$.

Lemma 2. *For a single lemming entering the synchronization gadget, the only way to survive is to dig at B and leave the gadget at D.*

Proof. If the lemming does not dig anywhere, it will reach the right end of the top ledge and fall to its death. The only safe place where to dig is point B, anywhere else the dig ends in a fall that is too high.

Lemma 3. *If for each lemming the only way to the exit leads through the point A of a single synchronization gadget, then the lemmings can only survive if all other lemmings arrive at A at 2 to 8 ticks after the first one.*

Proof. Consider the first lemming that arrives at A. From Lemma 2 it follows that this lemming has to start digging at B. As soon as this lemming digs a hole that is deep enough, no other lemming will be able to cross this gadget – regardless of what it does, it will fall and die somewhere. Hence there is only one way how the other lemmings may survive: they must arrive during the constant amount of time when the first lemming digs – and enter the hole below B while it is shallow enough. On the other hand, a delay of at least 2 ticks is necessary. The hole must be already deep enough so that the other lemmings can not escape it.

Note that when the digger finishes the hole, some of the other lemmings fall out of it walking in the opposite direction. This is fixed by the wall at C that turns these lemmings around.

Given an integer n, we will now create an instance I_n of LEMMINGS as follows: The only skill available will be the digger, and the number of times it can be used will be $n+1$. We will have n lemmings, each of them starting in a separate release gadget. The lengths of the release gadgets will be $5p_1, \ldots, 5p_n$, where p_i is the i-th prime number. The outgoing paths from these gadgets will be merged together in such a way that the number of steps from each of the gadgets to the common meeting point will be the same – except for one gadget that will be two steps closer. The path from the meeting point will lead onto point A of a single synchronization gadget, and the synchronization gadget will output the lemmings from point D straight to the exit.

Figure 7 shows the instance I_4. The release gadgets have lengths 10, 15, 25, and 35. The smallest release gadget is shifted 2 steps to the right – the lemming from this gadget will be the one digging at the synchronization gadget.

Lemma 4. *Let $p_n\#$ be the primorial, i.e., the product of the first n primes. Then $n^{\lfloor n/2 \rfloor} \leq p_n\# \leq 2 \cdot n^{2n}$.*

Proof. Obviously follows from the bounds on p_n proved in [14].

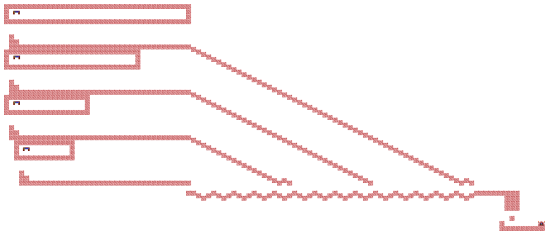

Fig. 7: The complete instance I_4.

Theorem 2. *Each instance I_n is solvable. The shortest number of ticks in which I_n can be solved exceeds $10n^{\lfloor n/2 \rfloor}$.*

Proof. In order to save all lemmings in I_n they all have to start digging out of their release gadgets at the same time. To prove this, assume the contrary. The release gadget lengths are multiples of 5, so by Lemma 1 the period of each lemming in the gadget is a multiple of 10. Hence if lemmings do not start digging at the same time, then the last lemming will leave the release gadget at least 10 ticks after the first one. Then the last lemming will arrive at A at least 8 ticks after the first one, and by Lemma 3 it will die, which is a contradiction.

Hence the lemmings must start digging at some time T that is a multiple of the period of each lemming. By the Chinese Remainder Theorem, the smallest such T is $T = 10p_n\#$, and by Lemma 4 we have $T > 10n^{\lfloor n/2 \rfloor}$.

We can easily verify that once all lemmings start digging at time T, the rest of the solution is short and unique.

Theorem 3. *The instance I_n can be described by $\Theta(n^2 \log n)$ bits.*

Proof. The height of the map is $\Theta(n)$, the width is $\Theta(n + p_n) = \Theta(p_n) = \Theta(n \log n)$, thus the entire map can be described using $\Theta(n^2 \log n)$ bits. We need to set a suitable time limit for the level. Clearly, the time limit $10p_n\#+47n \log n+47$ is sufficient. By Lemma 4 this number is $O(n^{2n})$, hence $O(\log(n^{2n})) = O(n \log n)$ bits are sufficient to encode it. All the remaining information (the skills vector, the total number of lemmings, and the number of lemmings to save) can be stored in $O(n)$ bits, even if using unary coding.

Corollary 3. *From Theorems 2 and 3 it follows that for the sequence of instances $\{I_n\}_{n=1}^{\infty}$ the length of the shortest solution grows exponentially in the instances' input sizes. Hence this sequence of instances is a counterexample to Cormode's proof that* Lemmings $\in$ *NP.*

Note 3. The above construction requires the player to do n actions at the same time (giving each of the lemmings the digger skill to free it from the release gadget). If we assume that the player can only make one action per tick, this is easily fixed by moving release gadget x exactly x steps away from the meeting point, for each x. In this situation the player's actions must occur in immediately consecutive steps.

Note 4. Both our construction and Cormode's original proof of NP-hardness involve multiple entrances. This is an unnatural construction, as most levels in the original Lemmings games only have a single entrance that releases all lemmings sequentially at a fixed rate. Below we prove that both results are true for LEMMINGS instances with a single entrance.

3.2 The distribution gadget

Our proof is based on the construction of a distribution gadget that takes a stream of lemmings (such as the one leaving a single entrance) and breaks it into individual lemmings, each on a separate path. The construction uses a method similar to the one used by the first lemming in the synchronization gadget: digging in a suitable place may decrease the height of the lemming's fall.

The distribution gadget for n lemmings is a generalization of the one shown in Figure 8. When employing this gadget we need to add n additional diggers to the skill vector, and create a separate exit path for each of the lemmings. Note that the bottom part of the gadget is placed exactly in such a height that a fall from the *top* of the top ledge would kill a lemming, whereas the fall from the *bottom* of the ledge is safe.

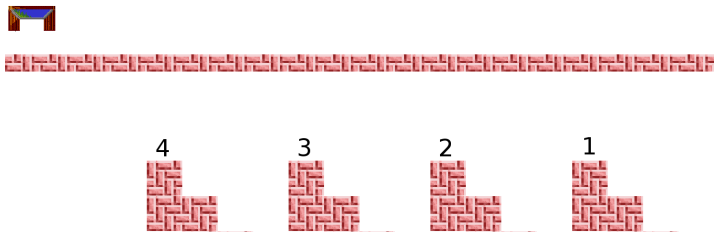

Fig. 8: The distribution gadget for four lemmings.

Theorem 4. *The only way in which all lemmings survive the distribution gadget is that the player makes each of them dig above one of the exit points.*

Proof. A lemming that is never assigned the digger skill will fall to its death – either at the end of the top ledge, or into a hole dug by some previous lemming. Hence each lemming must dig in order to survive, and there are precisely n places where a lemming can dig and survive the resulting fall. Note that the places for digging must be assigned right to left, i.e., the first lemming to get out of the entrance must be the one reaching point 1 in Figure 8.

Corollary 4. *In our construction of an instance I_n, we can start by using the distribution gadget. Then we can easily construct paths that will lead the separated lemmings to enter each of the individual release gadgets from above. Hence even if we restrict* Lemmings *to instances with only one entrance, there will still be instances with exponentially long solutions only.*

4 Conclusions

We have shown a general point of view on platform games that allowed us to find two classes of NP-hard platform games and one class of PSPACE-hard platform games. These classes include many well-known examples.

For the Lemmings problem, we have disproved Cormode's original assumption by showing that there are instances with only exponentially long solutions. Our construction works even if we limit ourselves to instances with only one entrance.

5 Open problems for further research

Clearly, Lemmings $\in$ PSPACE, as the number of reachable configurations is always at most exponential in the input size. An open question is whether Lemmings can actually be shown to be PSPACE-complete.

Our intuition suggests that this may indeed be the case. One observation is that the miner skill together with a combination of permeable and impermeable terrain can be used to free a trapped lemming from the outside. This may be exploited to "store" lemmings in various part of the level, use the presence/absence of these lemmings as memory, and have one other lemming free them when the memory state needs to be changed.

As for the general research of the computational complexity of games and puzzles, we are convinced that our approach from Section 2 is the correct direction for the future. As we documented in the overview, there are already many results on the hardness of individual puzzles. What we now need to discover are patterns in these results. What are the common features that make games and puzzles hard? Can then these observations help us analyze other games and puzzles? The recent Constraint Logic framework by Demaine and Hearn [5] is probably one of the first steps in this direction.

The topic of platform games is not exhausted in this article. While most of the games we examined are either obviously in P or covered by one of our metatheorems, there are some exceptions. Notably, so far we ignored the presence of enemies. We expect that in some cases the presence of enemies can make the games hard to solve. For instance, if enemies are present, the number of possible configurations may become exponential in the input size. If the enemies must be avoided, it should be possible to create instances that require enemy synchronization in order to be solvable, and this can force the solution to be exponentially long.

References

1. Charles L. Bouton. Nim, a game with a complete mathematical theory. *Annals of Mathematics*, 3:35–39, 1901/02.
2. John Horton Conway. *On Numbers and Games*. Academic Press, 1976.
3. 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.
4. Joseph Culbertson. Sokoban is PSPACE-complete. In *Proceedings of the International Conference on Fun with Algorithms*, pages 65–76, 1998.
5. Erik D. Demaine and Robert A. Hearn. Constraint logic: A uniform framework for modeling computation as games. In *Proceedings of the 23rd Annual IEEE Conference on Computational Complexity*, 2008.
6. Erik D. Demaine and Robert A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In Michael H. Albert and Richard J. Nowakowski, editors, *Games of No Chance 3*, volume 56 of *Mathematical Sciences Research Institute Publications*, pages 3–56. Cambridge University Press, 2009.
7. David Eppstein. Computational Complexity of Games and Puzzles, 2009. `http://www.ics.uci.edu/~eppstein/cgt/hard.html`.
8. P. M. Grundy. Mathematics and games. *Eureka*, 2:6–8, 1939.
9. Alon Itai, Christos H. Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton Paths in Grid Graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.
10. Richard Kaye. Minesweeper is NP-complete. *Mathematical Intelligencer*, 22(2):9–15, 2000.
11. G. Kendall, A. Parkes, and K. Spoerer. A Survey of NP-Complete Puzzles. *International Computer Games Association Journal*, 31(1):13–34, 2008.
12. David Lichtenstein. Planar Formulae and Their Uses. *SIAM Journal on Computing*, 11(2):329–343, 1982.
13. J. McCarthy. Partial formalizations and the Lemmings game, 1998. Technical report, Stanford University, Formal Reasoning Group.
14. Guy Robin. Estimation de la fonction de Tchebychef θ sur le k-ième nombre premier et grandes valeurs de la fonction $\omega(n)$ nombre de diviseurs premiers de n. *Acta Arith.*, 42(4):367–389, 1983.
15. J. M. Robson. The complexity of Go. In *Proceedings of the IFIP 9th World Computer Congress on Information Processing*, pages 413–417, 1983.
16. J. M. Robson. N by N Checkers is EXPTIME complete. *SIAM Journal on Computing*, 13(2):252–267, 1984.
17. Kristian Spoerer. *The Lemmings Puzzle: Computational Complexity of an Approach and Identification of Difficult Instances*. PhD thesis, 2007.
18. R. P. Sprague. Ueber mathematische Kampfspiele. *Tohoku Mathematical Journal*, 41:438–444, 1935/36.

A Example construction for Prince of Persia

The following is a complete level for Prince of Persia, representing the instance (M, w), where $|w| = 2$ and M is the smallest possible LBA with the δ-function given below, q_0 being the starting state, and q_2 the only final state of M.

The δ-function: $\delta(q_0, 0) = \{(q_0, 0, 1)\}$, $\delta(q_0, 1) = \{(q_1, 0, 1)\}$, $\delta(q_0, R) = \{(q_1, R, -1)\}$, and $\delta(q_1, 0) = \{(q_2, 1, 0)\}$.

The level is broken into two parts to make it fit on a single page. The places marked by a star and a hexagon should be attached to form the correct picture. Door and pressure plate labels were intentionally omitted, they can easily be reconstructed from the description in the article, if needed.