Lemmings Is PSPACE-Complete

Giovanni Viglietta

University of Ottawa, Canada,
viglietta@gmail.com

Abstract. Lemmings is a computer puzzle game developed by DMA Design and published by Psygnosis in 1991, in which the player has to guide a tribe of lemming creatures to safety through a hazardous landscape, by assigning them specific skills that modify their behavior in different ways. In this paper we study the optimization problem of saving the highest number of lemmings in a given landscape with a given number of available skills.

We prove that the game is **PSPACE**-complete, even if there is only one lemming to save. We thereby settle an open problem posed by Cormode in 2004, and again by Forišek in 2010. However, if we restrict the game to levels in which the available Builder, Basher, and Miner skills are only polynomially many (and there is any number of other skills), then the game is solvable in **NP**.

Furthermore, we show that saving the maximum number of lemmings is **APX**-hard, even when only Climber skills are available. This contrasts with the membership in **P** of the decision problem restricted to levels with no "deadly areas" (such as water or traps) and only Climber and Floater skills, as previously established by Cormode.

1 Introduction

Lemmings is a popular computer game originally developed by DMA Design for PC and Commodore Amiga. Since its first release in 1991, by Psygnosis, several ports, sequels and imitations have appeared, for various systems. The game revolves around the behavior of some creatures called *lemmings*, which deterministically walk across a landscape, turning around at walls, and blindly falling into pitfalls or drowning in water. The player's goal is to guide the highest number of lemmings through the landscape, from their respective *entrance locations* to any *exit location*, within a certain amount of time. To do so, the player has an arsenal of *skills* that he can individually assign to lemmings, in order to modify their behavior in different ways, and hopefully prevent them from perishing. Because the number of available skills is limited, and most skills have just a temporary effect, the player must carefully plan his strategy, which makes Lemmings a challenging puzzle game.

In this paper we study the computational complexity of the optimization problem of saving the highest number of lemmings in a given level of the game, contributing to a fast-growing branch of research delightfully surveyed in [3,6].

In [7], McCarthy first studied the game of Lemmings as an archetypical model for the logical approach to AI, attempting a formalization of the game using situation calculus, and discussing the features that make Lemmings a challenge to both experimental and theoretical AI. Spoerer later used genetic algorithms to generate successful solutions for a severely simplified version of Lemmings [9].

In [2], Cormode established several complexity results related to another simplified version of Lemmings. In Cormode's model, the landscape contains no *deadly areas* such as water, lava or traps, the player can assign skills to several different lemmings at the same time instant, and the time limit to complete each level is bounded by a polynomial in the size of the level itself. Cormode's paper shows the **NP**-completeness of deciding if a level of such a game is solvable, even when only a single lemming is present, and only Digger skills are available. It is further shown that, if only Floater and Climber skills are available, then solvability is decidable in **P**.

The rationale behind Cormode's assumption on the time limit is the claim that any level of Lemmings is either unsolvable or solvable within a polynomial amount of time. Later, in [4], Forišek disproved such a claim by constructing a class of levels whose solutions involve "waiting" an exponentially long time for certain configurations to occur, hence suggesting that the full Lemmings game may fail to be in **NP**. Both Cormode and Forišek conjectured that Lemmings, with no restrictions, is **PSPACE**-complete. Cormode also asked for the computational complexity of classes of game instances with different combinations of initially available skills.

Recently, in [10], the author gave an independent **NP**-hardness proof that works for instances with only Basher skills, and observed that a similar argument can be extended to instances with only Miner skills.

Our contribution. In Section 2 we define LEMMINGS, the optimization problem of maximizing the number of saved lemmings in a given level. One of the novelties of our approach is that we do not aim at studying a simplified or conveniently modified version of the game, but our model incorporates every aspect and feature of the original Lemmings game developed by DMA Design, including the known glitches.[1] (The only, obvious, exception is that we allow arbitrarily large levels with arbitrarily many *objects*.)

In Section 3 and Section 4 we argue that what separates the "harder" levels of LEMMINGS from the "easier" ones is the number of constructive and destructive commands that can be assigned to lemmings. Namely, if the number of Builder skills and the number of Basher skills are both exponential in the size of the level (or *unlimited*), then we are able to construct a **PSPACE**-complete class of instances with the bonus feature of having only one lemming each. Conversely, we show that the decision problem restricted to instances with only polynomially many available Builder, Basher, and Miner skills (and any number of other skills) belongs to **NP**. We thus provide an adequate answer to the open problem of Cormode and Forišek on the complexity of the full Lemmings game.

[1] See `http://www.lemmingsforums.com/index.php?topic=525.0`.

In Section 5 we discuss the restriction of LEMMINGS to instances with only Climber skills, and we give a proof of its **APX**-hardness, which also implies that computing approximate solutions with a relative error lower than 1/8 is **NP**-hard. Combined with Cormode's results, this suggests that what makes levels with only Climber skills "hard" is the presence of *traps*.

All our constructions have been tested with the DOS version of the original Lemmings game, and can be downloaded as a *level pack* from `http://giovannivi glietta.com/files/lemmings/gadgets.dat`.

2 Game Definition

We model LEMMINGS as an optimization problem (refer to [1]) whose instances are *levels* of the form $\mathcal{L} = (time, terrain, steel, objects, lemmings, rate, skills)$.

Time. In Lemmings, time is discretized and subdivided into *time units*. Accordingly, in each level of LEMMINGS, *time* is the amount of time units that the player has to achieve his goal of saving as many lemmings as possible. The value of *time* is assumed to be at most exponential in the size of the landscape (see below), or *unlimited*.

Landscape. *terrain*, *steel* and *objects* collectively define the *landscape* of the level:

- *terrain* is a rectangular array of *cells*, each of which is the size of a pixel and can be *empty* or *solid*. Informally, this is a bitmap containing the "shape" of the landscape: lemmings can freely walk across empty cells, but are stopped by solid cells. It is convenient to consider *terrain* as (logically) partitioned into *blocks* of 4×4 cells.
- *steel* is a rectangular array that tells whether each *terrain* block is "made of steel" or is "permeable". It may be viewed as a block-aligned "mask" that is overlaid on *terrain*, and is used to check if solid *terrain* cells may be "excavated" by Bombers, Bashers, Miners or Diggers (see below). Notice that each *terrain* 4×4 block is either entirely made of steel or entirely permeable, regardless of the amount of solid cells that it actually contains.
- *objects* is an array (of length polynomial in the size of *terrain*) whose elements have a *position* within the landscape, a *trigger area*, a *type*, and an optional *delay* parameter (whose value is bounded by a polynomial). Like steel masks, trigger areas are block-aligned bitmaps that are overlaid on *terrain*. However, if a block hosts the trigger area of some object, it cannot be made of steel, and hence it must be permeable. There are four types of objects:
  - *Entrance*. Each lemming enters the level through an entrance (see below).
  - *Exit*. A lemming reaching the trigger area of an exit is "rescued" and is removed from the game. There may be several exits in the same level.
  - *Deadly zone*. A lemming lying in the trigger area of a deadly zone instantly dies and is removed from the game. However, after a deadly zone has killed a lemming, it remains harmless for k time units, where k is the

object's delay parameter, and then it becomes deadly again. During that window of k time units, lemmings can safely traverse the trigger area. (Even if several lemmings enter the trigger area at the same time unit, only one is killed immediately.) Deadly zones with $k > 0$ are represented in Lemmings as "traps", such as presses, gallows poles and electrocuting devices; deadly zones with $k = 0$ are represented as water or lava.

- *One-way wall.* The *terrain* cells underlying its trigger area are perceived as permeable by Bashers and Miners going in one direction, and made of steel by Bashers and Miners going in the opposite direction (see below).

Notice that both *steel* and the trigger areas of objects have a coarser resolution than *terrain*, due to the file format that Lemmings uses to store levels. We also stress that a 4×4 block cannot simultaneously be made of steel and be part of the trigger area of an object. These features will add an extra challenge to the design of our gadgets.

Lemmings. The *lemmings* parameter of a level is the total amount of lemmings that the level contains (which is assumed to be bounded by a polynomial in the size of *terrain*). Lemmings enter the game one at a time, at a frequency given by the parameter *rate*. If several entrances are present, they release lemmings in turns, following an order determined by their position in the *objects* array.

Upon entering the land, each lemming is facing right, and is normally a *Faller*, which falls vertically through empty *terrain* cells due to gravity, until it lands on a solid cell. Then it becomes a *Walker*, which keeps walking straight (in the direction it is facing) as long as it can. In Lemmings, the sprite of a lemming is between nine and ten pixels high, depending on the animation frame. However, only one cell matters for collision detection with the landscape, which is the lemming's *pin*. The pin is located one cell below the lemming's feet, and its exact position varies depending on the animation frame and the direction the lemming is facing.

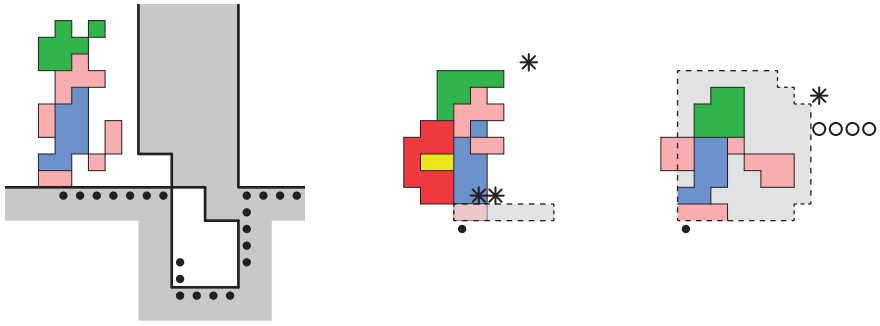
On flat ground, a Walker's pin moves forward by eight cells every four time units. Between time units, the collision detection algorithm first moves the Walker's pin forward by one cell, no matter if it is solid or empty. Then *terrain* cells are checked to determine the lemming's behavior.

If the pin has reached a solid cell, then the cells above are also checked. If the lowest empty cell is eight cells above the pin or higher, then the slope is too high and the Walker reverses its direction. Otherwise, the pin "jumps" above by at most two cells, and then goes further up by one cell per time unit, until the top is reached.

Otherwise, if the pin has reached an empty cell, the lemming falls down until it reaches a solid cell again. On the first time unit, the pin's position instantly drops by at most four cells. If a solid cell has not been reached yet, then the lemming becomes a Faller and its pin gradually moves down, by at most two cells per time unit. If the fall is longer than 63 cells (or crosses the bottom of the terrain), the lemming dies and is removed from the game.

Depending on the Walker's animation frame, the above procedure may be repeated between one and three times per time unit (on "almost flat" ground).

Figure 1(a) illustrates an example, in which dots represent the final positions of the lemming's pin each time the collision detection algorithm is executed.



(a) Walker (b) Builder (c) Basher

Fig. 1. (a) Sequence of pins (black dots) of the lemming, as it walks rightward over the gray solid cells. (b) First step of the stairway (dashed area), and the three cells that are tested for solidity (asterisks). (c) Cells dug on the first stroke (dashed area), the cell tested for permeability (asterisk), and the four cells tested for solidity (circles).

Skills. Finally, the level parameter *skills* is an array containing the number of skills that the player can assign to lemmings. We will assume that all skill quantities are bounded by an exponential in the size of *terrain*, or *unlimited*. The skills are:

- *Climber*. A permanent skill that makes a lemming climb vertical rows of more than six solid cells, at an average speed of one cell every two time units, instead of turning around like a Walker. As soon as a Climber reaches the "top of a wall", it starts behaving like a Walker again. If it hits a "ceiling" while it is climbing, it turns around and falls back down.
- *Floater*. A permanent skill that makes a lemming survive falls of any height. Floaters also fall slower than Fallers.
- *Bomber*. Makes a lemming explode after a short amount of time units. A Bomber keeps behaving normally until it actually blows up, also turning the surrounding *terrain* cells from solid to empty, provided that the Bomber's pin lies on a permeable cell.
- *Blocker*. Makes a lemming stand in place and act as a wall for the other lemmings. Climbers cannot climb on Blockers.
- *Builder*. Makes a lemming construct a "stairway" by turning empty *terrain* cells into solid ones. Each "step" of the stairway is six cells wide and one cell high, and is laid on top of the Builder's pin, as Figure 1(b) indicates. Then three *test cells*, represented as asterisks in the figure, are checked for solidity. If all of them are empty, the Builder's pin is moved one cell up and two cells

forward, and a new step is laid. Otherwise, the Builder turns around and becomes a Walker. After dropping 12 bricks, a Builder becomes a Walker anyway, and proceeds forward as usual.

- *Basher*. Makes a lemming "dig" a horizontal hole in the direction it is facing, by turning solid *terrain* cells into empty cells. Upon assignment of the skill, the lemming checks the *test cell* marked by an asterisk in Figure 1(c). If it is permeable (no matter if it is solid or empty), the lemming becomes a Basher and makes a hole shaped like the dashed area. It then proceeds digging forward at five cells per *stroke*. It only stops when it falls into a hole (then it becomes a Faller), or when it encounters a steel cell in the location marked by the asterisk (then it turns around and becomes a Walker), or when all the four circled cells are empty (then it becomes a Walker without turning around). One-way walls are treated as steel or permeable cells, depending on their orientation.
- *Miner*. Similar to Basher, but a Miner digs diagonally.
- *Digger*. Similar to Basher, but a Digger digs vertically.

Builders, Bashers, Miners, and Diggers can be interrupted by the player at any time by assigning them a different skill (that is not a Climber or a Floater skill). Blockers can be interrupted only by digging the solid *terrain* cell on which they stand, or by assigning them a Bomber skill, which kills them. We remark that assigning a Basher skill to a Walker that is not facing a wall will make it stroke once, with no effect other than delaying its walk for a couple of time units.

Actions. A player's *action* is the assignment of a certain skill to a certain lemming at a certain time. Actions are done by "clicking" on lemmings. At most one skill can be assigned per time unit. In particular, if several lemmings lie under the "cursor" at the same time, the skill is assigned only to one lemming. An action is encoded by a lemming's position in the lemmings array, a skill identifier, and a *timestamp*.

A *feasible solution* of Lemmings is then a finite sequence of actions that are compatible with each other and with the given amount of available skills. To complete the definition of Lemmings, we still need a *measure function*, which is obviously the number of lemmings that the player saves within the time limit, by making a certain sequence of actions given by a feasible solution.

3 Instances Solvable in NP

Here we consider the restriction of Lemmings to instances whose number of initially available Builder, Basher, and Miner skills is bounded by a polynomial in the size of the landscape. We prove that the decision version of such a restricted problem is in **NP**. This result extends [2, Lemma 1] by Cormode, which states that Lemmings is in **NP**, provided that the time limit to solve a level is polynomial (and in particular there are polynomially many available skills).

Theorem 1. *The decision version of* Lemmings*, restricted to levels in which the Builder, Basher, and Miner skills are polynomially many, belongs to* ***NP****.*

Proof. Recall that the total number of lemmings is polynomial in the number of *terrain* cells. It follows that permanent skills, i.e., Climber and Floater skills, can be assigned only polynomially many times, and therefore involve a polynomial number of moves. The same holds, for obvious reasons, also for Bomber skills.

Observe that Digger skills can be assigned only to lemmings that can effectively dig some solid cells. Because the initial number of these cells is polynomial, and each cell can be "restored" at most once per available Builder, it follows that Digger skills involve at most a polynomial number of moves.

Blocker skills can also be assigned polynomially many times, because a Blocker can be interrupted only by killing it with a Bomber skill, or by digging the terrain on which it stands.

Finally, Builder, Basher, and Miner skills are polynomially many by assumption. It follows that the total number of actions performed by the player is bounded by a polynomial.

Let us consider a feasible solution that saves a certain number of lemmings in a given level. We will transform it into a new feasible solution of polynomial size that saves as many lemmings. By the above reasoning, we only have to prove that the timestamps of all moves have polynomial size.

It is easily seen that *terrain* may change only at polynomially many time units. Indeed, *terrain* can be altered only as a consequence of a player's action. Moreover, the only way to create new solid cells is via a Builder, which affects at most 72 cells. Hence the number of cells that can be restored is bounded by a polynomial. As a consequence, also the cells that can be destroyed is bounded by a polynomial.

Hence there are at most polynomially many maximal timespans during which no skill is assigned, *terrain* does not change, and no lemming dies. Let $[t, t']$ be one such maximal timespan. Because there exists at most an exponential amount of combined configurations of all lemmings, the configuration at time t' is reached also at some time $t'' \leqslant t'$ such that $t'' - t$ is bounded by an exponential. Therefore we may assume that all timestamps are bounded by an exponential in the size of the level (even if the level's time limit is unlimited), and that in turn all of them can be encoded using polynomially many digits.

Now we show that a polynomial-size sequence of actions is a valid certificate, by arguing that it is possible to compute in polynomial time the number of lemmings that it saves. Indeed, the polynomially many transitions between time units in which moves are made, or *terrain* changes, or some lemming dies can be simulated in polynomial time, because each transition involves a constant number of operations and tests for each lemming. The remaining time intervals are polynomially many and may be exponentially long. Observe that, during each such interval of time, lemmings do not interfere with each other, and hence each of them follows a polynomially long periodic path. So each lemming's periodic path is computed independently, and the whole time interval is divided by that period, in order to efficiently compute the lemming's final position (with no need of explicitly simulating exponentially many transitions). □

As a side note, we observe that the above proof does not easily extend to instances with exponentially many Basher or Miner skills. Indeed, in contrast with Digger skills, these skills can be assigned not only to effectively dig a positive amount of solid cells, but also to delay a Walker for a couple of time units, or to reverse its direction, without altering *terrain* (refer to Section 2). A similar observation holds for Builder skills.

4 PSPACE-Complete Instances

Next we show that there are classes of levels of LEMMINGS that are **PSPACE**-hard. Due to Theorem 1, it comes as no surprise that such levels have exponentially many (or unlimited) available Builder and Basher skills.

Theorem 2. LEMMINGS *is* ***PSPACE****-complete, even restricted to levels with only one lemming, and only Builder and Basher skills.*

Proof. The membership in **NPSPACE** of LEMMINGS is obvious, because each game configuration can be stored in polynomial space, and the configuration graph can be efficiently navigated. The membership in **PSPACE** thus follows from Savitch's Theorem (see [8]).

As for **PSPACE**-hardness, we apply [10, Metatheorem 2.c], which is based on a reduction from QUANTIFIED BOOLEAN FORMULA involving a player-controlled *avatar*, a *starting location*, an *exit location*, several *paths*, *pressure plates* and *doors*. In our implementation, the only lemming in the level will be the avatar, which will be controlled by the player via the assignment of Builder and Basher skills at very specific locations. We build the level in such a way that every 4×4 cell is made of steel, unless it contains the trigger area of a deadly zone (recall from Section 2 deadly zones must be permeable).

Figure 2 shows how paths are implemented. White space denotes empty *terrain* cells, shaded space denotes solid cells, and black dots mark the positions occupied by the lemming's pin as paths are traversed following the arrows. Each large crossed square represents a 4×4 block containing the trigger area of a deadly zone. Collectively, deadly zones prevent the lemming from striving from its path. The one in Figure 2(b) is also a one-way path, because it cannot be traversed from right to left. This is attached after the crossover in Figure 2(e) (one copy is attached to the right, and a symmetric copy to the left), so that the lemming cannot take the wrong path, should it accidentally reverse its direction anywhere after the crossover. Observe that some of the paths may occasionally be "broken" if the lemming becomes a Basher at the right time. However, this action has the only possible effect of rendering some paths unusable. Moreover, there is no need to implement a way of letting the avatar reverse its direction in the middle of a path, because this is not necessary to solve the levels constructed in the proof of [10, Metatheorem 2.c].

It is easy to see that any directed graph can be embedded in the plane by suitably arranging copies of the five gadgets in Figure 2 and their mirror images, provided that *forks* are implemented. This is done with the *selector gadget* in

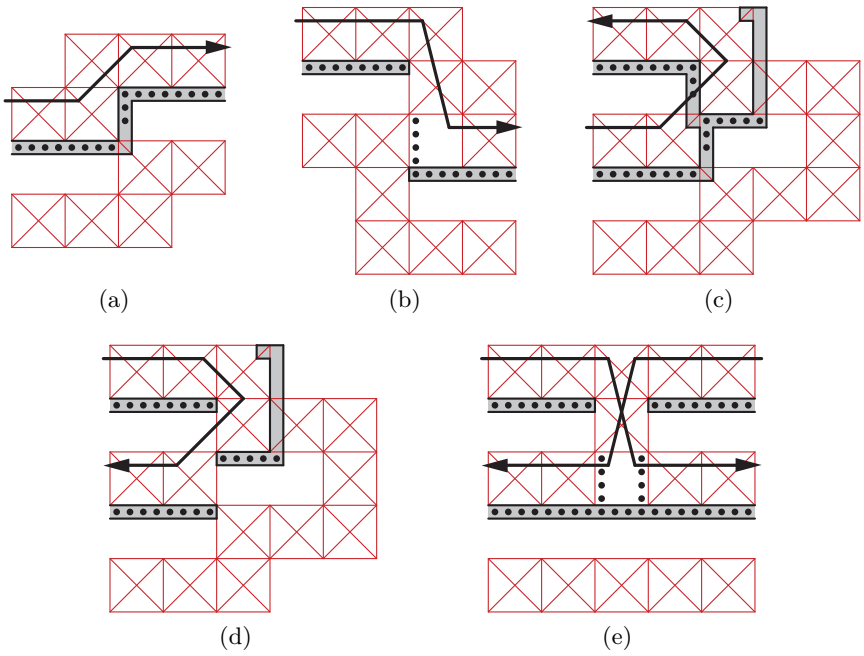
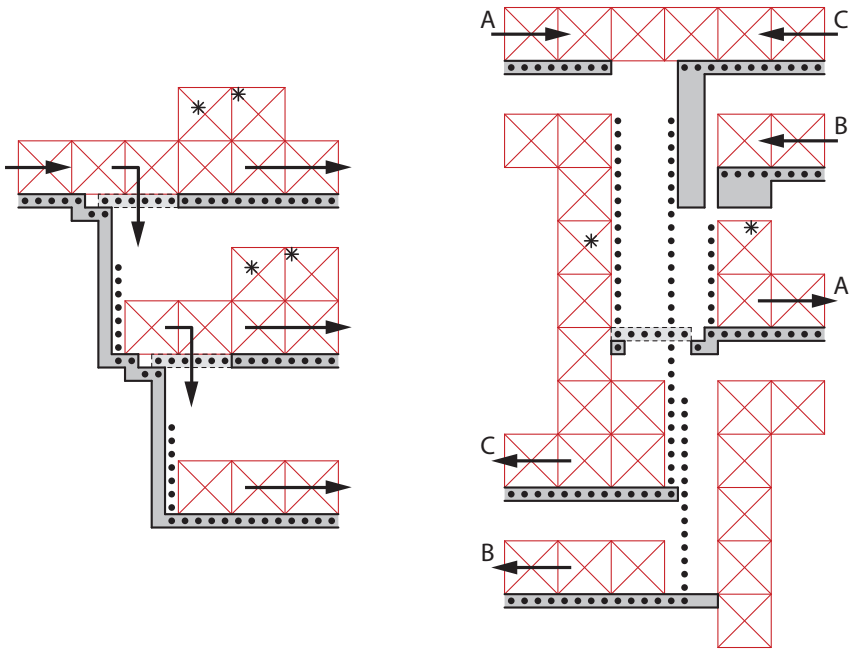


Fig. 2. Paths. The lemming's pin must never touch the deadly zones.

Figure 3(a). The lemming enters from the left, and then the player may redirect it to any of the three exits on the right. The deadly zones are the only permeable blocks, and they are positioned in such a way that a Builder can lay a single step of a stairway (indicated by a dashed rectangle in Figure 3(a)), climb on it, and then immediately become a Basher to stop building further steps and proceed to the right as a Walker. Moreover, if a step is already present when the lemming arrives, it can be removed by assigning a Basher skill right before the lemming climbs on it. This will cause the lemming to excavate precisely the 6-cell step with one stroke (refer to Figure 1(c)) and then fall down. Any other way of assigning skills is either ineffective, or deadly, or prevents lower areas from being reached (which never helps the lemming reach its final goal). The asterisks in the figure mark the cells that are tested for permeability when the lemming becomes a Basher as described above.

Finally, we need doors, which are areas that can be traversed by the avatar if and only if they are *open*. For each door, there are exactly two pressure plates located somewhere in the level, which open and close the door, respectively. Pressure plates are activated whenever the avatar traverses them, but we observe that the proof of [10, Metatheorem 2.c] keeps working even if the pressure plates that open doors are implemented as *buttons*, i.e., the avatar is not forced to activate them upon traversal, but may or may not do it, at the player's will.



(a) Selector (b) Door

Fig. 3. PSPACE-hardness gadgets

Indeed, opening a door has the only effect of expanding the set of locations that can be reached by the avatar, so it is never "wrong" to do it as soon as possible.

Our *door gadget* is depicted in Figure 3(b), where the door is considered open if and only if the central dashed rectangle is made of empty cells. A pressure plate is implemented "indirectly", as a path that starts from the location that should contain it, reaches the corresponding door gadget from the proper direction, and then leads back to where it started. The two locations marked with a letter A (respectively, B) are connected to the pressure plate that closes (respectively, opens) the door.

If the lemming is coming from A and the door is open, then it must construct a stairway step, thus closing the door, and then stop immediately, using a Basher skill, in order to proceed to the right without "hitting the ceiling" and turning around (refer to Figure 1(b)). If it does anything different, it is bound to enter some deadly zone and perish. In particular, it is straightforward to see that if it tries to build an additional stairway step when the door is already closed, then it cannot become a Basher because the permeable block is too low, so it eventually hits the ceiling, turns left and dies in the deadly zones no matter what it does.

If the lemming is coming from B, then it can open the door with a Basher skill, right before falling down, thanks to the permeable blocks on the left. If it

ever becomes a Builder, then it is bound to die in a few time units, as it can be easily verified.

When the lemming actually attempts to cross the door, it enters from the path marked with a letter C, and survives if and only if the door is open. Again, becoming a Builder at any time would kill it, no matter what it does next.

This completes the construction. It is clear, also referring to the proof of [10, Metatheorem 2.c], that each of these levels is either unsolvable with any amount of Builder and Basher skills, or solvable with exponentially many of them. □

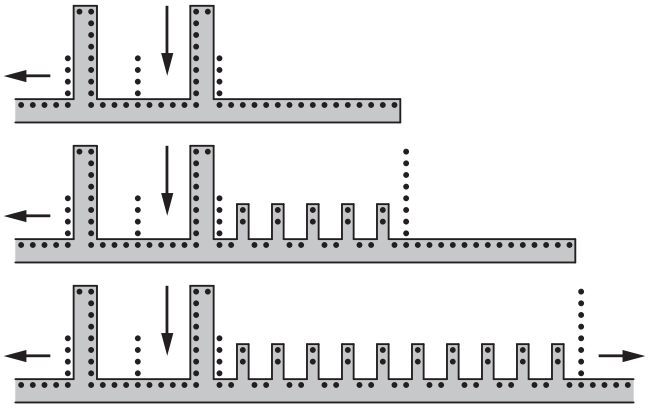
5 Inapproximability

Here we consider the restriction of LEMMINGS to instances with only Climber skills. By Theorem 1, this variation is solvable in **NP**, while its further restriction to levels with no deadly zones is solvable in **P**, due to [2, Theorem 2]. We now show that the presence of deadly zones makes LEMMINGS **APX**-hard, and thus not in **P** (unless $\mathbf{P} = \mathbf{NP}$, see [1]).

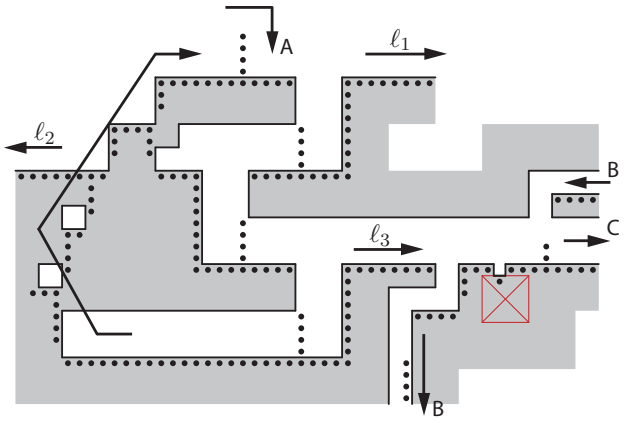
Theorem 3. LEMMINGS *is* ***APX****-hard, even restricted to levels in which only Climber skills are available.*

Proof. We give an L-reduction from the **APX**-complete problem MAX-3-SAT (refer to [1]), in which the number of satisfied clauses of a 3-CNF Boolean formula has to be maximized. We need *variable gadgets* and *clause gadgets*, both depicted in Figure 4, wired together with the paths of Figure 2, which have the same properties highlighted in the proof of Theorem 2, no matter if several lemmings traverse them simultaneously, and some of them are Climbers.

Referring to Figure 4(a), a gadget for variable x is made of several *layers*, each containing an entrance for exactly one lemming (vertical arrows). There are $2k - 1$ layers, where k is the number of occurrences of x in the formula. All lemmings are initially confined in a small area, until the player decides to make them escape, either from the left or from the right, by assigning them a Climber skill at the correct time. All lemmings exiting from the same side eventually reach a common path, on which a row of $k - 1$ traps is found (not shown in Figure 4(a)), each with a parameter of eight time units. Because each trap kills at least one lemming upon traversal, no two lemmings can exit the gadget from different sides and survive. The way to guarantee that k lemmings may indeed safely exit the gadget (all from the same side) is to "synchronize" them by delaying those on lower layers, via a series of 3-cell *bumps*. Each bump delays a Walker by exactly one time unit, so that the pins of all the $2k - 1$ lemmings will eventually lie within three cells from each other (depending on their animation frames), and exactly one lemming will be killed by each of the $k - 1$ traps. So, the truth value of x will be encoded by the side from which k (or fewer) lemmings exit the corresponding gadget. After coming out of the true (respectively, false) side of the variable gadget, the group of lemmings traverses, one by one, all the clause gadgets containing a positive (respectively, negative)



(a) Variable



(b) Clause

Fig. 4. APX-hardness gadgets

occurrence of x. The group then reaches a pool of water, so that any remaining lemming is killed.

Figure 4(b) shows a gadget for clause $(\ell_1 \vee \ell_2 \vee \ell_3)$, in which a single lemming, entering from the top (arrow with letter A), is bound to walk in a loop until the player assigns it a Climber skill and makes it escape from one of three exits, each corresponding to a literal of the clause. After each exit, a trap is encountered (only shown for literal ℓ_3 in Figure 4(b)), and then a path safely leads to a level exit (arrow with letter C). The same trap is also traversed, in the opposite direction, by a path coming from the variable corresponding to that literal (arrows with letter B), in such a way that the clause lemming can be saved by the group of variable lemmings if and only if the literal is true according to the chosen assignment. In order to guarantee synchronization and make sure

that the trap is reached by the head of the group of variable lemmings just a couple of time units before the upcoming clause lemming, a series of bumps is added to path B slightly before the gadget is reached. Indeed, notice that all the clause gadgets have the same shape and size, so each clause lemming will complete its loop in a constant number of time units, say, d. Hence, each clause lemming will have a chance of exiting the gadget from each of the three exits exactly once every d time units, and therefore it is sufficient to add at most $d-1$ bumps to each path entering a clause gadget to enforce synchronization.

Clearly, only the clause lemmings can possibly be saved in these levels, and each of them may indeed be saved if and only if at least one literal of its clause gadget is true according to the assignment encoded by the corresponding variable lemmings. Therefore, the reduction preserves the optimal value, and any solution that saves n lemmings in one of these levels can be trivially converted into a variable assignment that satisfies exactly n clauses of the corresponding Boolean formula. As a consequence, this is an L-reduction. □

It follows that the optimal number of saved lemmings is not approximable within a small-enough ratio, even in this severely restricted case.

Corollary 1. *Computing approximate solutions to* Lemmings *with a relative error lower than* $1/8$ *is* ***NP****-hard, even for levels with only Climber skills.*

Proof. The proof of Theorem 3 describes an L-reduction from Max-3-Sat with $\beta = \gamma = 1$ (refer to [1, Definition 8.4]), hence the claim follows from [5, Theorem 6.1]. □

References

1. C. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and approximation: Combinatorial optimization problems and their approximability properties.* Springer, 2003.
2. G. Cormode. The hardness of the Lemmings game, or Oh no, more NP-completeness proofs. In *Proceedings of FUN'04*, 65–76, 2004.
3. E. D. Demaine and R. A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory. In *Games of No Chance 3*, edited by M. H. Albert and R. J. Nowakowski, MSRI Publications, 56:3–56, 2009.
4. M. Forišek. Computational complexity of two-dimensional platform games. In *Proceedings of FUN'10*, 214–226, 2010.
5. J. Håstad. Some optimal inapproximability results. In *Proceedings of STOC'97*, 1–10, 1997.
6. G. Kendall, A. Parkes, and K. Spoerer. A survey of NP-complete puzzles. *International Computer Games Association Journal*, 31:13–34, 2008.
7. J. McCarthy. *Partial formalizations and the Lemmings game.* Technical report, Stanford University, Formal Reasoning Group, 1998.
8. C. H. Papadimitriou. *Computational complexity.* Addison-Wesley Publishing Company, Inc., 1994.
9. K. Spoerer. *The Lemmings puzzle: computational complexity of an approach and identification of difficult instances.* Ph.D. thesis, University of Nottingham, 2007.
10. G. Viglietta. Gaming is a hard job, but someone has to do it! In *Proceedings of FUN'12*, 357–367, 2012.