

Gaming is a hard job, but someone has to do it!

GIOVANNI VIGLIETTA
Department of Computer Science
University of Pisa
viglietta@gmail.com

September 21, 2012

Abstract

We establish some general schemes relating the computational complexity of a video game to the presence of certain common elements or mechanics, such as destroyable paths, collectible items, doors opened by keys or activated by buttons or pressure plates, etc. Then we apply such “metatheorems” to several video games published between 1980 and 1998, including Pac-Man, Tron, Lode Runner, Boulder Dash, Deflektor, Mindbender, Pipe Mania, Skweek, Prince of Persia, Lemmings, Doom, Puzzle Bobble 3, and Starcraft. We obtain both new results, and improvements or alternative proofs of previously known results.

1 Introduction

This work was inspired mainly by the recent papers on the computational complexity of video games by Forišek [4] and Cormode [2], along with the excellent surveys on related topics by Kendall et al. [7] and Demaine et al. [3], and may be regarded as their continuation on the same line of research. A conference version of this paper has appeared at FUN 2012 [12].

Our purpose is to single out certain recurring features or mechanics in a video game that enable general reduction schemes from known hard problems to the games we are considering. To this end, in Section 2 we produce several *metatheorems* that will be applied in Section 3 to a wealth of famous commercial video games, in order to automatically establish their hardness with respect to certain computational complexity classes (with a couple of exceptions).

Because most recent commercial games incorporate Turing-equivalent scripting languages that easily allow the design of undecidable puzzles as part of the gameplay, we will focus primarily on older, “scriptless” games. Our selection includes games published between 1980 and 1998, presented in alphabetical order for better reference. Not every game will be rigorously explained in all its aspects and details, but at least the game elements that are relevant to our proofs will be

introduced, so that any casual player will promptly recognize them and readily understand our constructions.

It is clear that, in order to meaningfully apply the standard computational complexity tools, a suitable *generalization* of each game must be considered. Since classic video games typically include only a finite set of levels, whose complexity is merely a constant, a way must be devised to automatically generate a class of infinitely many new levels of increasing size. Deciding which game elements are “scalable” and which are not is ultimately a matter of taste and common sense: when designing a generalization of a well-known game, one should remain as faithful as possible to the feeling and mechanics of the original version. For example, in a typical platform game, the number of platforms and the number of hazards in a level may increase as the level size grows. In contrast, the maximum height of a jump and the enemy AI should remain unchanged, as they are more inherent aspects of the game.

It is generally acknowledged that single-player games that are humanly “interesting” to play are complete either for **NP** or for **PSPACE** (for an introduction to general computational complexity theoretic concepts and classes, refer to [9]). **NP**-complete games include levels whose solution demands some degree of ingenuity, but such levels are usually solved within a polynomial number of “manipulations”, and the challenge is merely to find them. In contrast, the additional complexity of a **PSPACE**-complete game seems to reside in the presence of levels whose solution requires an exponential number of manipulations, and this may be perceived as a nuisance by the player, as it makes for tediously long playing sessions.

Several open problems remain for further research: whenever only the hardness of a game is proved with respect to some complexity class, the obviously implied question is whether the game is also complete for that class. Moreover, different variations of each game may be studied, obtained for instance by further restricting the set of game elements used in our hardness proofs. Indeed, the computational complexity of a game is expected to dramatically drop if some “critical” elements are removed from its levels. It is interesting to study the “complexity spectrum” of a game, as a function of the game parameters that we set. This has been done to some extent for the game of Lemmings, by different authors, as partly documented in Section 3.

2 Metatheorems

More often than not, games allow the player to control an *avatar*, either directly or indirectly. In some circumstances, an avatar may be identified within the game only through some sort of artifice or abstraction on the game mechanics. Throughout Section 2, we will stipulate that the player’s actions involve controlling an avatar, and that the elements of the game may be freely arranged in a plane lattice, or a higher dimensional space. At the very least, the set of game elements includes *walls* that cannot be traversed by the avatar, and can be arranged to form rooms,

paths, etc.

2.1 Location traversal and single-use paths

A game is said to exhibit the *location traversal* feature if the level designer can somehow force the player’s avatar to visit several specific game locations, arbitrarily connected together, in order to beat the level. Locations may be visited multiple times in any order, but the first one is usually fixed (starting location), and sometimes also the last one is (exit location). An example of location traversal is the *collecting items* feature discussed in [4]: a certain number of items are scattered across different locations, and the avatar’s task is to collect them all.

The *single-use paths* feature is the existence of configurations of game elements that act as paths connecting two locations, which can be traversed by the avatar at most once. A typical example are *breakable tiles*, which disappear as soon as the avatar walks on them.

Metatheorem 1. *Any game exhibiting both location traversal and single-use paths is NP-hard.*

Proof. We give a straightforward reduction from HAMILTONIAN CYCLE, which is NP-complete even for undirected 3-regular planar graphs [6]. Construct a plane embedding of a given 3-regular graph G (perhaps an orthogonal embedding, if needed) with an additional node u dangling from a distinguished node v . Then we convert such embedding into a valid level, by implementing each node as a location that must be visited by the avatar, and each edge as a single-use path. The starting location is placed in v and, if an exit location is required, it is placed in u .

Clearly, the last node the avatar must visit is u , because it has only one incident edge. Moreover, each node except v can be visited at most once: recall that G is 3-regular, hence reaching a given node $w \notin \{u, v\}$ for the first time implies the consumption of one of its incident edges. Then, leaving w consumes another incident edge, and reaching it a second time consumes the third incident edge. At this point, there is no way for the avatar to leave w , and therefore no way to reach the last node u . As for the starting node v , the incident edges are initially four, and one is immediately consumed. The second time the avatar reaches v , it must necessarily proceed to u , for otherwise u would become forever unreachable. It follows that the level is solvable if and only if the player can find a walk starting from v , touching every node (except u) exactly once, reaching v again, and then terminating in u . This is possible if and only if G contains a Hamiltonian cycle. \square

As Section 3 testifies, Metatheorem 1 has a wide range of applications, and it tends to yield game levels that are more “playable” than those resulting from the somewhat analogous [4, Metatheorem 2], which rely on a tight time limit to traverse a grid graph. Additionally, [4, Metatheorem 2] is prone to design complications in “anisotropic” games, in which the avatar moves at different speeds in different directions, for instance due to gravity effects.

2.2 Tokens and toll roads

We consider now another type of game mechanics: *tokens* and *toll roads*. Tokens are items that can be carried by the avatar, and *toll roads* are special paths connecting two locations. Whenever the avatar traverses a toll road, it must “spend” a token that it is carrying. If the avatar is carrying no token, then it cannot traverse a toll road.

We distinguish between two types of tokens: *collectible* tokens, which may be placed by the game designer at specific locations and can be picked up by the avatar, and *cumulative* tokens, any number of which can be carried around by the avatar at the same time. Section 3 will offer some examples of different types of tokens: for instance, Pac-Man features *power pills*, which may be regarded as collectible tokens that are not cumulative.

Metatheorem 2. *A game is NP-hard if either of the following holds:*

- (a) *The game features collectible tokens, toll roads, and location traversal.*
- (b) *The game features cumulative tokens, toll roads, and location traversal.*
- (c) *The game features collectible cumulative tokens, toll roads, and the avatar has to reach an exit location.*

Proof. Once again, we give a reduction from HAMILTONIAN CYCLE for all three parts of the metatheorem, varying it slightly depending on our hypotheses. For part (a), given an undirected 3-regular planar graph G , we construct an embedding as described in the proof of Metatheorem 1, and we implement each node as a location that has to be traversed by the avatar. Each edge is then implemented as a toll road, and one collectible token is placed in each node, except for the final node u , where we place no token, and the starting node v , where we place two tokens.

Notice that, if G has n nodes, there are exactly $n + 1$ locations that the avatar must visit, and $n + 1$ tokens in the level. Therefore, any feasible traversal of the level starts from v and has length at most $n + 1$. If the traversal must reach all locations, then at most one location may be visited twice. Moreover, v must be visited at least a second time, because it is the only neighbor of u . As a consequence, a valid traversal of the level must start from v , visit every other location except u exactly once, return in v , and end in u . It follows that, if G has no Hamiltonian cycle, then the level is unsolvable.

Conversely, let us assume that G has a Hamiltonian cycle, and let us show that the level is solvable. The avatar can traverse G , starting from v and ending in v again, along a Hamiltonian cycle, and finally it can reach u and solve the level. This traversal is valid even if tokens are not cumulative: upon reaching a new location, the avatar collects one new token and immediately spends it in a toll road. Likewise, when v is reached for the second time, the second token is collected, and it is immediately spent to reach u .

The construction for part (b) is the same, but instead of scattering $n + 1$ tokens throughout the level (where n is the number of nodes of G), we assume that the

avatar already carries $n + 1$ tokens as the game starts. Then a similar reasoning applies: exactly one location may be visited twice, which must be v because it is the starting location and the only neighbor of u . Therefore, u must be the last location to be visited, and the level is solvable if and only if G has a Hamiltonian cycle.

For part (c), we further modify the previous proof as follows: we construct the same embedding of G , and we place two tokens in every location, except in u , where we place no token. Then we implement each edge as a toll road, except the edge between v and u , which is implemented as a sequence of n toll roads. The starting location is v again, and the exit location is u . The avatar carries no token as the game starts.

There are $2n$ tokens in the level, and n of them must be used to travel from v to u , so at most n more tokens may be spent in other toll roads. Every time a toll road is traversed, one token is gained if a new location is reached (one token is spent and two are found), and one token is lost if an already visited location is reached. It follows that the player must find a walk in G that starts and ends in v , traverses at most n edges and visits n different nodes. This is equivalent to finding a Hamiltonian cycle in G . (Observe that the location traversal feature has been obtained here as a by-product of our construction, without being an explicit requirement.) \square

2.3 Doors and keys

A *door* is a game element that can be open or closed, and may be traversed by the avatar if and only if it is open. A *key* is a type of token that can be used by the avatar to open a closed door, upon contact. Any key can open any door, but a key is “consumed” as soon as it is used. Hence, the key-door paradigm is somewhat similar to the token-toll road one, with the difference that a door opened by a key remains open and can be traversed several times afterwards without consuming new keys.

We distinguish again between *collectible* keys, which can be found by the avatar and picked up, and *cumulative* keys, any number of which can be carried at the same time. Many examples of keys are found in platform games and adventure games. In Section 3, we will show how the Lemmings game features cumulative keys that are not collectible, although this will be established through non-trivial abstractions on the game mechanics.

To state the next result, which is an analogous of Metatheorem 2 for the key-door paradigm, we further need to introduce the concept of *one-way path*, which is a path that can be traversed by the avatar in one specific direction only.

Metatheorem 3. *A game is NP-hard if it contains doors and one-way paths, and either of the following holds:*

- (a) *The game features collectible keys and location traversal.*

(b) *The game features cumulative keys and location traversal.*

(c) *The game features collectible cumulative keys and the avatar has to reach an exit location.*

Proof. We reduce from HAMILTONIAN CYCLE, which is **NP**-complete even for directed planar graphs whose nodes have one incoming edge and two outgoing edges, or two incoming edges and one outgoing edge [10]. All three parts of our proof are based on the same construction: given one such directed graph G on n nodes, we pick a node v with indegree two and outdegree one (which exists), and we attach to it a new outgoing edge, ending in a new node u . Then we construct a plane embedding of this graph (maybe an orthogonal embedding), substituting each node with a game location, and each directed edge with a one-way path. v will be the avatar's starting location and, if an exit location is required, it is placed in u (u must be the final location anyway, because it has no outgoing edges). Moreover, we place a closed door in each one-way path, except in the path between v and u .

To prove part (a), place one key in each location, except u . The avatar must traverse every location by hypothesis, and the last one must be u . After collecting the first key in v , every time a new location (except u) is reached, one key is used to open a door, and one new key is found. On the other hand, as soon as an already visited location w is reached, the only key in the avatar's possession is lost and no key is found, so afterwards the avatar is bound to traverse only paths with no door (hence (v, u)) or with an already opened door. As a consequence, the level is solved if and only if $w = v$ and every location except u has been visited, which is possible if and only if G has a Hamiltonian cycle.

For part (b), we put no keys in the level, but we assume that the avatar already carries n keys as the game starts. Due to the location traversal feature, each location must be visited, and therefore at least two of its incident paths' doors must be opened (unless the location is u). Hence, n doors must be opened in total, and all the keys must be used. On the other hand, if all the three incident paths' doors of a location are opened, and all the locations are visited, a straightforward double counting argument shows that at least $n + 1$ keys have been used, which is unfeasible. Therefore, the avatar must follow a Hamiltonian cycle of G starting and ending in v , and then visit u .

Finally, for part (c), we place two keys in each location, except in u , and we place n doors in the path between v and u . No key is carried by the avatar as the game starts. Hence, the avatar must visit some locations to collect keys, return to v with at least n keys, and reach the exit in u . The same double counting argument used for part (b) reveals that, if $k \leq n$ distinct locations are visited before reaching u , then at least k doors must be opened. In particular, exactly k doors are opened if and only if a cycle of G is followed. Because visiting k distinct locations allows to collect exactly $2k$ keys, the only way to return in v with n keys is to follow a cycle of length $k = n$, i.e., a Hamiltonian cycle. \square

2.4 Doors and pressure plates

There are other ways to modify a door's status, such as activating a *pressure plate*. A pressure plate is a floor button that is pushed whenever the avatar steps on it, and its effect may be either the opening or the closure of a specific door. Each pressure plate is connected to just one door, and each door may be controlled by at most two pressure plates (one opens it, one closes it). Of course, all our hardness results will hold in the more general scenario in which any number of doors is controlled by the same pressure plate, or any number of pressure plates control the same door.

In [4, Metatheorem 3], Forišek shows (with a different terminology) that a game is **NP**-hard if the avatar has to reach an exit location, and the game elements include one-way paths, doors and pressure plates (or 1-buttons, see the next subsection) that can open doors. In the following metatheorem, we further explore the capabilities of pressure plates.

We say that a game allows *crossovers* if there is a way to prevent the avatar from switching between two crossing paths. Some 2-dimensional games natively implement crossovers through bridges or tunnels. In some other games, crossovers can be simulated through more complicated gadgets.

Metatheorem 4. *If a game features doors and pressure plates, and the avatar has to reach an exit location in order to win, then:*

- (a) *Even if no door can be closed by a pressure plate, and if crossovers are allowed, then the game is **P**-hard.*
- (b) *Even if no two pressure plates control the same door, the game is **NP**-hard.*
- (c) *If each door may be controlled by two pressure plates, then the game is **PSPACE**-hard.*

Proof. The reduction for part (a) is from MONOTONE CIRCUIT VALUE, and the OR and AND gates are implemented as in Figures 1a and 1b. The starting location is connected to all true input literals, and the exit is located on the output.

For part (b), observe that we can implement single-use paths as shown in Figure 1c: in order to traverse the gadget, the avatar must walk on both pressure plates, thus permanently closing both doors. Since we can also enforce location traversal by blocking the exit with several closed doors, which may be opened via as many pressure plates positioned in every location, we may indeed invoke Metatheorem 1.

Finally, to prove (c), we implement a reduction framework from TRUE QUANTIFIED BOOLEAN FORMULA, sketched in Figure 2. A given fully quantified Boolean formula $\exists x \forall y \exists z \dots \varphi(x, y, z, \dots)$, where φ is in 3-CNF, is translated into a row of *Quantifier gadgets*, followed by a row of *Clause gadgets*, connected by several paths.

Traversing a Quantifier gadget at any time sets the truth value of the corresponding Boolean variable. On the other hand, each Clause gadget can be traversed

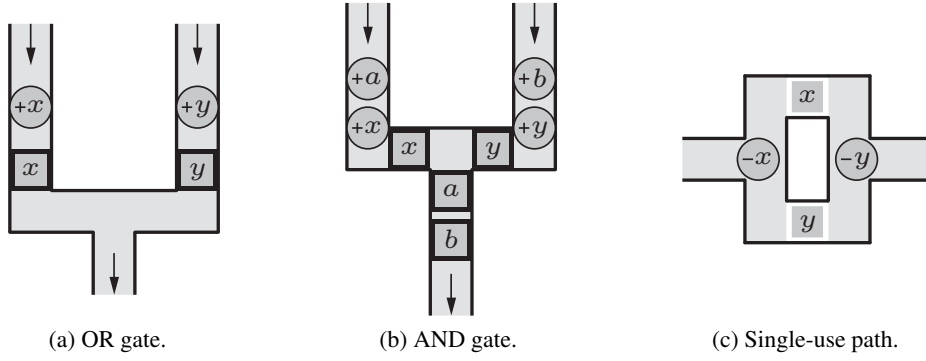


Figure 1: Several gadgets implemented with doors and pressure plates. A pressure plate that is labeled $+x$ (respectively, $-x$) opens (respectively, closes) the unique door labeled x , and a door is initially open (respectively, closed) if its border is white (respectively, black).

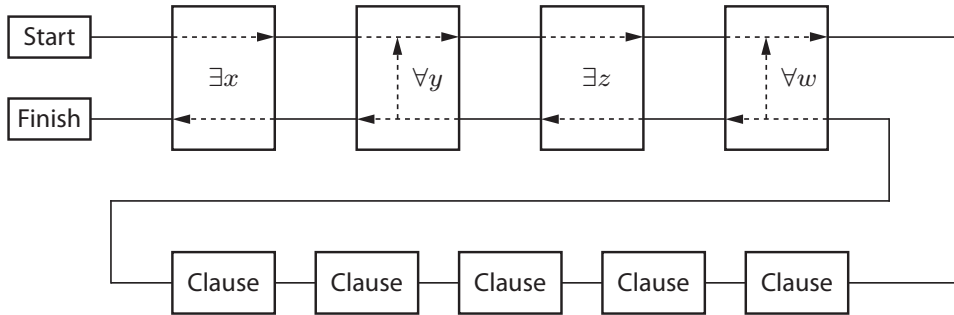


Figure 2: Sketch of the PSPACE-hardness framework.

if and only if the corresponding clause of φ is satisfied by the current variable assignments. Whenever traversing an Existential Quantifier gadget, the player can choose the truth value of the corresponding variable. On the other hand, the first time a Universal Quantifier gadget is traversed, the corresponding variable is set to true.

When all variables are set, the player attempts to traverse the Clause gadgets. If the player succeeds, he proceeds to the “lower parts” of the Quantifier gadgets, where he is rerouted to the last Universal Quantifier gadget in the sequence. The corresponding variable is then set to false, and φ is “evaluated” again by making the player walk through all the Clause gadgets.

The process continues, forcing the player to “backtrack” several times, setting all possible combinations of truth values for the universally quantified variables, and choosing appropriate values for the existentially quantified variables in the attempt to satisfy φ .

Finally, when all the necessary variable assignments have been tested and φ keeps being satisfied, i.e., if the overall quantified Boolean formula is true, the exit becomes accessible, and the player may finish the level. Conversely, if the quantified Boolean formula is false, there is no way for the player to operate doors in order to reach the exit.

Next we show how to implement all the components of our framework using just doors and pressure plates.

Clause gadgets are straightforwardly implemented, as shown in Figure 3. There is a door for each literal in the clause, and the avatar may traverse the clause if and only if at least one of the doors is open.

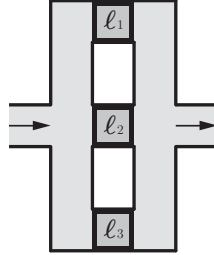


Figure 3: Clause gadget implemented with doors.

The Existential Quantifier gadget for variable x is illustrated in Figure 4. x_1 , x_2 , etc. (respectively, \bar{x}_1 , \bar{x}_2 , etc.) denote the positive (respectively, negative) occurrences of x in the clauses of φ .

When traversing the upper part of the gadget from left to right, the player must choose one of the two paths, thus setting the truth value of x to either true or false. This is done by appropriately opening or closing all the doors corresponding to occurrences of x in φ . The doors labeled a and b prevent leakage between the two different paths of the Existential Quantifier gadget, enforcing mutual exclusion.

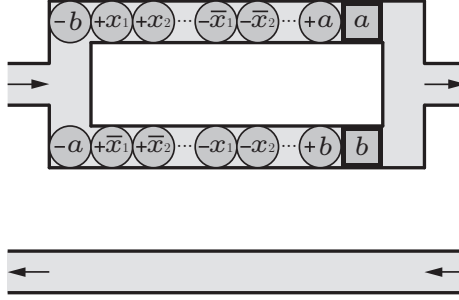


Figure 4: Existential Quantifier gadget implemented with doors and pressure plates. x_i (respectively, \bar{x}_i) denotes the i -th occurrence of literal x (respectively, $\neg x$) in φ .

Finally, the lower part of the gadget is traversed from right to left when the player backtracks, and it is simply a straight path.

A Universal Quantifier gadget for variable x is shown in Figure 5.

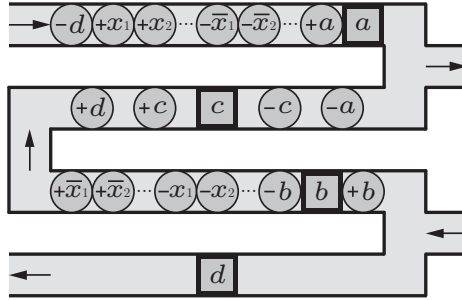


Figure 5: Universal Quantifier gadget implemented with doors and pressure plates.

When the avatar enters the gadget from the top left, door d gets closed and variable x is set to true. Then the avatar must exit to the top right, because door c cannot be traversed from right to left.

When backtracking the first time, the avatar enters from the bottom right and, because door d is still closed, it must take the upper path, thus setting variable x to false. Incidentally, door d gets opened and door a gets closed, thus preventing leakage to the top left entrance, and forcing the avatar to exit to the top right again.

When backtracking the second time (i.e., when both truth values of x have been tested), door d is open and the avatar may finally exit to the bottom left. When done backtracking, the avatar will eventually enter this gadget again from the top left, setting x to true again, etc.

We note that, as a result of our constructions, each door is operated by exactly two pressure plates. For instance, the door labeled x_1 , located in some Clause gadget, is opened and closed by exactly two pressure plates, both located in the Quantifier gadget corresponding to variable x . \square

Observe that our Metatheorem 4.c is an improvement on [4, Metatheorem 4], in that the *long fall* feature (and thus the concept of gravity) is not used, and it works with a more restrictive model of doors: in [4], arbitrarily many pressure plates can act on the same door, while we allow just two.

2.5 Doors and buttons

A *button* is similar to a pressure plate, except that the player may choose whether to push it or not, whenever his avatar encounters one.

Games with buttons are in general not harder than games with pressure plates, because a pressure plate can trivially simulate a button, as Figure 6a shows. However, since the converse statement is not as clear, we will allow a single button to act on several doors, in contrast with pressure plates. A button acting on k doors simultaneously is called a *k-button*.

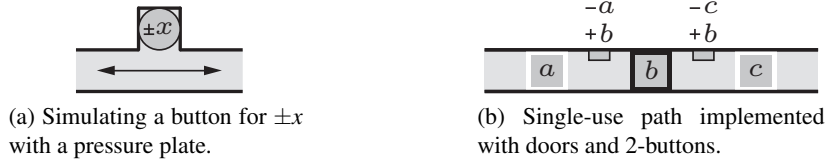


Figure 6: Button-related constructions. Small rectangles represent buttons, with hovering letters indicating their functions.

We obtain an analogous of Metatheorem 4 for buttons.

Metatheorem 5. *If a game features doors and k -buttons, and the avatar has to reach an exit location in order to win, then:*

- (a) *If $k \geq 1$ and crossovers are allowed, then the game is **P**-hard.*
- (b) *If $k \geq 2$, then the game is **NP**-hard.*
- (c) *If $k \geq 3$, then the game is **PSPACE**-hard.*

Proof. For part (a), we mirror the proof of Metatheorem 4.a, by using 1-buttons as opposed to pressure plates. Indeed, pressing a button to open a door is never a “wrong” move, if the goal is to reach the exit location.

For part (b), we implement single-use paths as in Figure 6b: in order to open door b , one of the two buttons has to be pressed, thus permanently closing door a or door c . Then we proceed as in the proof of Metatheorem 4.b.

Finally, for part (c), we use the gadget in Figure 7a to simulate a generic pressure plate for $\pm x$: the only way to traverse the gadget from left to right (the other direction is symmetric) is to press the buttons as indicated in Figures 7b, 7c, and 7d, incidentally activating also door x . Moreover, observe that, no matter how the six buttons are operated, there is no way to “break” the gadget by leaving its four

doors in an open-closed state that is not the original one. Now, by simulating general pressure plates, we can apply the **PSPACE**-hardness framework used for Metatheorem 4.c, concluding the proof. \square

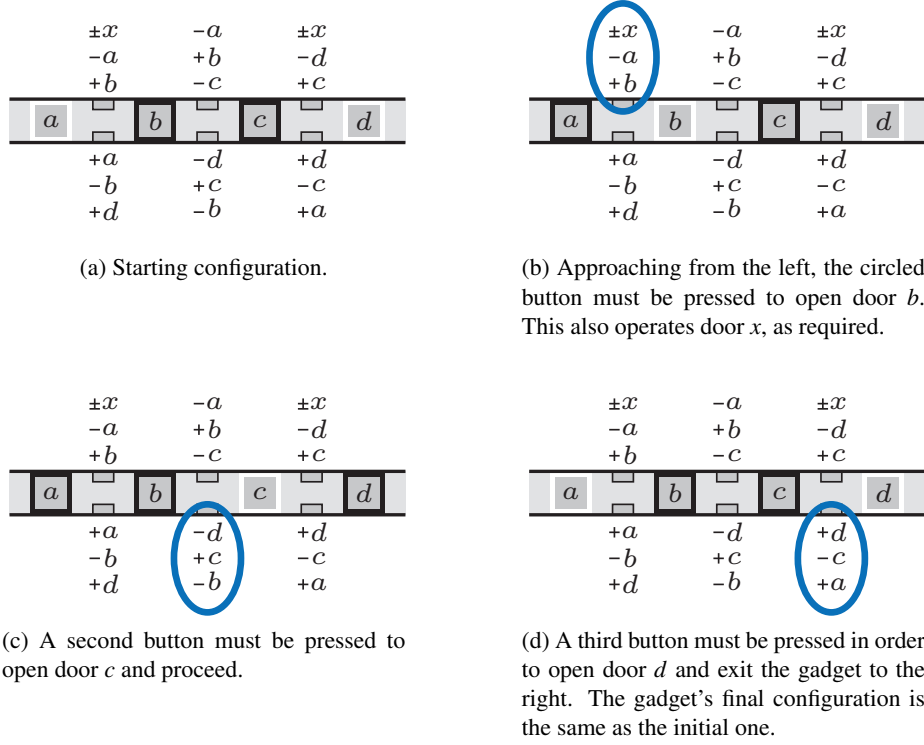


Figure 7: Simulating a pressure plate for $\pm x$ with four 3-buttons.

3 Applications and further results

3.1 Boulder Dash (First Star Software, 1984) is NP-hard

Game description. The game is similar to Sokoban, but with added gravity. The player-controlled avatar may push (but not pull) single boulders horizontally, excavate some special tiles, and must collect diamonds while avoiding monsters. When a certain amount of diamonds has been collected, an exit door appears, and the avatar has to reach it to beat the level. Gravity affects boulders and diamonds, but not the avatar or the monsters.

Complexity. A proof that “pushing blocks in gravity” is **NP**-hard has been given by Friedman [5], based on a rather involved reduction scheme and several gadgets

that may be adapted to work with the slightly different “physics” of Boulder Dash (namely, if a boulder is on the “edge” of a pit, it falls down even if it is not pushed).

We give a much simpler proof that relies on Metatheorem 1. Location traversal is trivially implemented, due to the presence of diamonds that have to be collected: we place one diamond in each relevant location, forcing the player to visit them all. A single-use path gadget is illustrated in Figure 8: when traversing the gadget in either direction for the first time, three boulders are pushed in the pits. On the second traversal attempt, the fourth boulder blocks the path.

Notably, we do not use diggable tiles or monsters in our reduction, although we do require diamonds.



(a) Initial configuration.



(b) Configuration after a traversal from left to right.



(c) On the second traversal attempt, a boulder blocks the way.

Figure 8: Single-use path for Boulder Dash.

3.2 Deflektor (Vortex Software, 1987) is in L

Game description. This is a psychedelic puzzle game in which a laser ray has to be reflected around in the level by rotating several mirrors. The laser ray is emanated by a laser beam and must be reflected to a predefined location, after hitting several items that must be collected, while avoiding static mines, and without reflecting the laser back to the source for too long (which overheats the beam). All the relevant game elements are static, or can rotate in place. There are 16 possible orientations for the laser ray, and making it reach some location basically boils down to finding the correct way to orient the player-controlled mirrors. Some tiles act as reflecting walls, some are opaque and absorb the ray, some special tiles act

as teleporters, others as self-rotating mirrors, or self-rotating *polarizators* that may be traversed by the ray only in one direction at a time. All polarizators have eight possible orientations, and rotate at the same speed (or they are static). There are also some prisms that randomly refract the ray, and some gremlins that attach to player-controlled mirrors and randomly reorient them.

Complexity. This is a remarkable example of an “easy” commercial game: Deflektor is solvable in logarithmic space, which is a quite uncommon feature for a puzzle game, and possibly contributed to its modest success.

The key observation is that the ray never needs to be reflected twice by the same mirror in order to reach some location, because it can be re-oriented to any direction already on its first reflection.

For our purposes, prisms count as a special type of player-orientable mirror, as they effectively refract the ray in any desired direction and at the right moment, after waiting a long-enough time. Similarly, self-rotating mirrors count as regular mirrors, as the player can indeed slow them down or accelerate them. On the other hand, gremlins may be disregarded in our analysis, as their presence is merely a nuisance and never really prevents a level from being solved.

Next we show how to reduce Deflektor to the **L** problem **UNDIRECTED CONNECTIVITY** [9, 11]. Recall that there are eight possible combined orientations of the polarizators, as they are either static or keep rotating at the same speed. Each combined orientation yields a *reachability graph* G_i on the game elements, which tells if a ray can be redirected by an object onto another. A reachability graph may be computed in **L** by shooting the 16 possible rays from each mirror (or prism) and extending them until each ray is absorbed or reaches a relevant game element, such as a mirror or a collectible item. This necessarily happens after a finite number of reflections, because the possible ray slopes are rational, and a ray that is never absorbed must have a periodic trajectory.

Let G^* be the disjoint union of all the G_i ’s, in which the eight copies of the laser beam are connected to a common *beam node*. Finding a path in G^* from the beam node to one of the eight copies of an object means that the laser ray can be redirected to that object after suitably rotating the mirrors, and after waiting for the polarizers to be properly oriented.

The final graph is obtained as the disjoint union of several copies of G^* , one for each item to collect. Let us arbitrarily order the items, and let G_j^* be the copy of G^* associated to the j -th item (here, the exit location counts as the last item in the list). Then, the eight copies of the j -th item in G_j^* are linked to the beam node of G_{j+1}^* . The beam node of G_1^* will be called the *starting node*, and the eight copies of the last item in the last copy of G^* are connected to a common node, which will be the *ending node*.

Thus, the final graph has a path from the starting node to the ending node if and only if the mirrors can be oriented in such a way that the first item in the list can be collected, then reoriented to collect the second item, etc., and finally the

exit location can be reached. Because items can be collected in any order, this is equivalent to solving the level.

3.3 Doom (id Software, 1993) is PSPACE-hard

Game description. Doom is a first-person shooter in which the player has to navigate a maze in search of an exit, surviving several enemies and collecting different weapons. The level editor is quite versatile, allowing the designer to implement switch-operated doors, trigger areas, elevators, teleporters, etc.

Complexity. Application of Metatheorem 4.c is immediate: pressure plates can be implemented via walkover lines and sector tags. In simple terms, whenever the player-controlled avatar crosses a certain line on the ground, a “block” somewhere in the level is moved to a predefined location, thus simulating the opening or closure of a door.

A similar claim holds for most FPS games (with the notable exception of Wolfenstein 3D), adventure games, and dungeon crawls. This includes RPGs such as Dungeon Master, The Eye of the Beholder, and Lands of Lore, which natively implement doors operated by pressure plates. All the point-and-click adventure games based on LucasArts’ SCUMM engine, such as Maniac Mansion and The Secret of Monkey Island, as well as most Sierra’s adventure games, easily fall in this category, too.

3.4 Lemmings (DMA Design, 1991) is NP-hard

Game description. 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. There is a limit to the number of times each skill can be assigned to a lemming, and skills range from building a short stair, to excavating a tunnel, to climbing vertical walls, etc. If no skill is assigned to a lemming, it keeps walking forward, turning around at walls, and falling into pits. Hazards include deadly pools of water or lava, several kinds of traps, and long falls. To beat a level, at least a given percentage of lemmings has to reach one of several exit portals within a time limit.

Complexity. The NP-hardness of Lemmings was already proved by Cormode in [2] using only Digger skills. More recently, the author showed that Lemmings is PSPACE-complete, even if there is only one lemming in the level, and only Basher and Builder skills are available (the technique used is based on a variant of Metatheorem 4.c) [13]. Here we propose a simple alternative proof of NP-hardness, which uses only Basher skills (that allow lemmings to dig horizontally) and relies on Metatheorem 3.b. Our construction can be easily modified to work with Miner skills, too (used to dig diagonally).

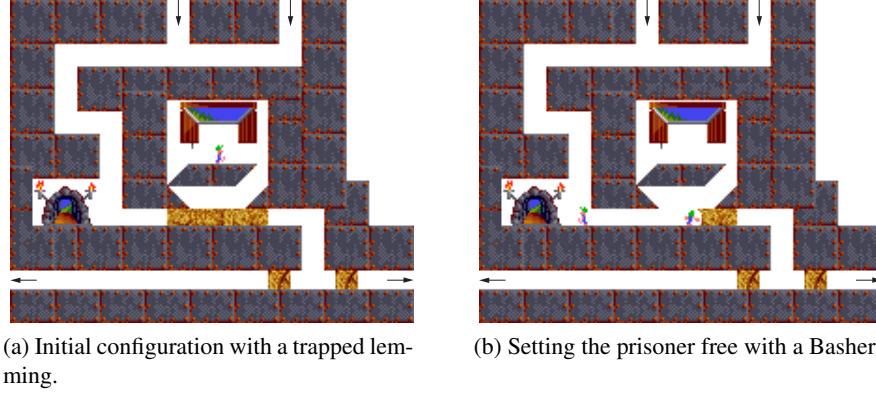


Figure 9: Enforcing location traversal in Lemmings.

We model each location as in Figure 9a. As the game starts, exactly one lemming joins the level from the trapdoor, and is bound to stay in the enclosed area, walking back and forth. If either the indegree or the outdegree of the location is less than two, we suitably remove some of the passages marked by arrows. In the starting location, we fit a second trapdoor that releases another lemming in the upper corridor: this lemming is not a prisoner, and will be the avatar. In the final location, we replace the outgoing passages with an exit portal, which is intended for the avatar. We appropriately connect locations together as the arrows suggest, and we make sure that the right outgoing passage of the starting location leads to the exit location (rather than the left passage). All the lemmings in the level must reach an exit portal, and the initially available skills are $2n + 2$ Bashers, where $n + 1$ is the amount of locations in the level.

The tiles with the steel texture in Figure 9 cannot be excavated, hence any lemming that is not the avatar is effectively trapped inside a cage, waiting for the avatar to rescue it by Bashing the ground below, as Figure 9b illustrates. The prisoner is bound to come out of its cage facing left, thus it will inevitably reach the nearby exit portal. This implies that the avatar must visit every location in the level (location traversal) and that at least $n + 1$ Basher skills have to be used to rescue all the prisoners. At any time, the number of available Basher skills minus the number of trapped lemmings will be understood as the number of keys carried by the avatar. Note that a small amount of ground (i.e., a door) must be Bashed in order to exit a location from an unvisited outgoing edge, and the initial amount of keys is $n + 1$. This agrees with the key-door paradigm and the requirements of Metatheorem 3.b, except for the presence of a door in the path between v and u , which is matched by the extra available key (this must be the last door to be opened anyway, hence the discrepancy can be safely ignored).

The passages connecting locations are indeed one-way paths by construction, since lemmings cannot change direction unless they encounter a wall. Observe that the avatar may choose which outgoing path to take on its first traversal of a

location. However, after the right path has been taken, there is no way to take the left path on the second traversal (but not vice versa). Fortunately, by inspecting the proof of Metatheorem 3.b, this does not appear to be an issue, because it is not restrictive to assume that each location will be visited only once, except for the starting location, in which both outgoing paths must be taken. But we made sure that the right path leads to the exit location, and therefore it must be the last path to be traversed, which is indeed feasible.

3.5 Lode Runner (Brøderbund, 1983) is NP-hard

Game description. The player-controlled avatar must collect gold pieces while avoiding monsters, and is able to dig holes into certain floor tiles (those that look like bricks), which regenerate after a few seconds. Both the avatar and the monsters may fall into such holes, and the avatar cannot jump. The avatar is killed when it is caught by a monster, but it can safely stand in the tile directly above a monster's head. Monsters behave deterministically, according to the player's moves, although their behavior is often quite counterintuitive, as they do not always take the shortest path toward the player's avatar. When every gold piece has been collected, a ladder appears, leading to the exit.

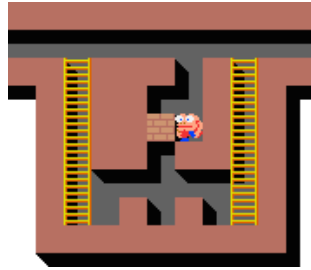
Complexity. We apply Metatheorem 1: location traversal is implied by the collecting items feature, and a single-use path is illustrated in Figure 10. On the first traversal, the avatar can safely land on top of the monster and dig a hole to the left. The AI will make the monster fall in the hole, so the avatar may follow it, land on its top again, and proceed through a ladder, while the brick tile regenerates and the monster remains trapped in the hole below. The avatar cannot attempt to traverse the gadget a second time without getting stuck in the hole where the monster previously was (recall that the avatar can neither jump, nor dig holes horizontally).

3.6 Mindbender (Magic Bytes, 1989) is NL-hard

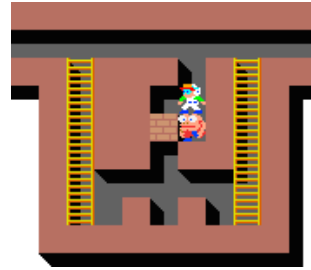
Game description. This is the fantasy-themed sequel of Deflektor (see above), with a wizard shooting a ray of light in some direction, static gnomes holding orientable mirrors, kettles that must be collected by hitting them with the ray of light, and several new game elements. These include collectible keys that open locks, wandering monsters that eat kettles, movable blocks, etc.

All the elements of Deflektor are recreated in Mindbender, with substantially identical mechanics, with one crucial exception: polarizators in Mindbender are manually orientable by the player, whereas in Deflektor they rotate on their own.

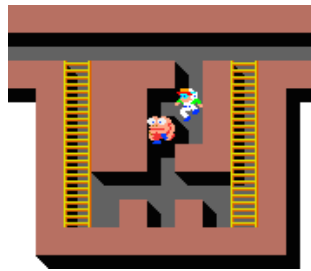
Complexity. The full game is easily NP-hard and arguably PSPACE-complete, but the interesting fact is that even the subgame that is supposed to be “isomorphic” to Deflektor is in fact NL-complete, thus harder than Deflektor.



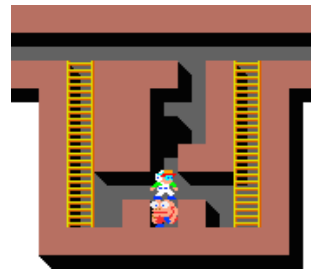
(a) Initial configuration.



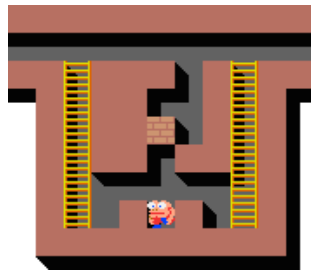
(b) Safely landing on top of the monster.



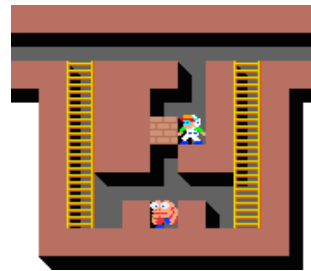
(c) Digging the brick tile to set the monster free.



(d) Following the monster down the hole and exiting through a ladder.



(e) The brick tile regenerates, and the monster remains trapped below.



(f) On the second traversal attempt, the avatar gets stuck.

Figure 10: Single-use path for Lode Runner.

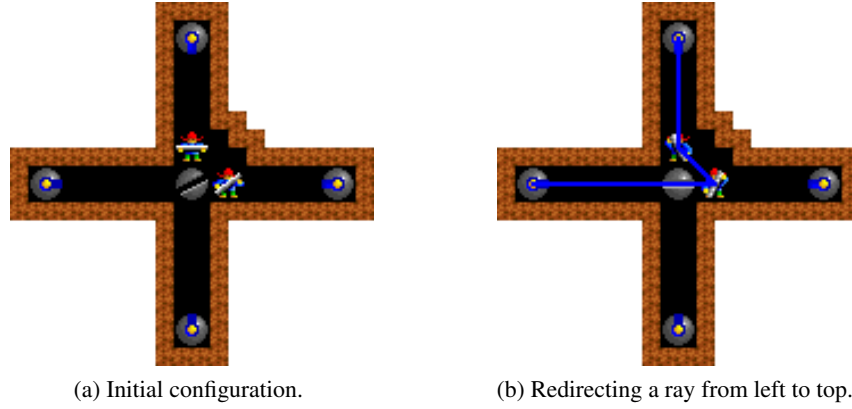


Figure 11: Implementing a directed graph in Mindbender.

We give a straightforward reduction from the **NL**-complete problem **DIRECTED CONNECTIVITY** [9]: first of all, we may assume that each node of the given graph has indegree and outdegree at most two. Then, each such node is modeled with the gadget in Figure 11. The left and bottom teleporters correspond to incoming edges, while the upper and right teleporters correspond to outgoing edges. Teleporters in different gadgets are connected together according to the topology of the given graph (which may be non-planar).

The central object is a polarizator, which can be oriented by the player and lets light rays pass in one direction only. The two gnomes can reflect the light either upward or rightward, as Figure 11b exemplifies. No light ray can be redirected from the left teleporter to the bottom one, or vice versa, due to the polarizator.

The starting node contains the wizard instead of the left (or bottom) teleporter, and the ending node contains the exit door instead of the top (or right) teleporter. The level is solvable if and only if a path exists from the starting node to the ending node. Remarkably, no kettle has been used in the reduction.

3.7 Pac-Man (Namco, 1980) is NP-hard

Game description. The player controls a yellow ball whose task is to collect all the *pills* in a maze, while avoiding *ghosts*. However, collecting some special *power pills* makes the avatar invulnerable for a short time, giving it the ability to temporarily disable ghosts upon contact. Despite this seeming simplicity, the full set of rules is quite complicated [1], although just a few are relevant to our purposes.

Ghosts come in four different colors, and a ghost's color determines its behavior. However, all ghosts alternate between Chase mode and Scatter mode. In Chase mode, they follow the avatar with different heuristics, and in Scatter mode they head toward a preset location. There is also a Frightened mode, which is entered when the avatar collects a power pill, and makes all ghosts move randomly.

After a few seconds, the effects of the power pill expire and all ghosts are back to Chase and Scatter modes. If a ghost in Frightened mode is touched by the avatar, it goes back to its starting location, a *ghost house*, and comes out again shortly.

As a general rule, every time there is a mode switch, all ghosts immediately reverse their direction. Other than that, ghosts may never reverse direction, not even upon reaching a maze intersection, and not even when in Frightened mode. (This is also a practical way for the player to tell when a mode switch occurs.)

Depending on the game level, all timings and speeds are subject to variations: ghosts may be faster or slower in different modes, and the durations of the three modes may vary. Usually, during Frightened mode, the avatar speeds up and the ghosts slow down.

Complexity. The decision problem is whether a level can be completed without losing lives. We assume full configurability of the amount of ghosts and ghost houses, speeds, and the durations of Chase, Scatter, and Frightened modes. We do not alter the basic game mechanics or the AI, though.

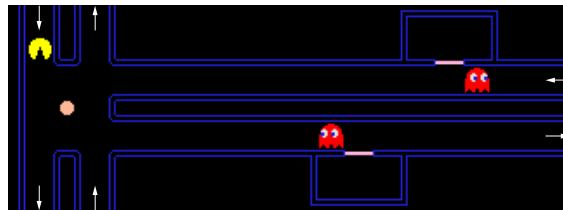
We prove **NP**-hardness by applying Metatheorem 2.a. A location with an adjacent toll road is sketched in Figure 12. Power pills are used to model tokens, so the starting location contains two power pills, and the final location contains none. Hence, to properly enforce location traversal, we further place a normal pill in the final location.

Each toll road is implemented as a pair of parallel maze corridors, each of which contains a ghost house somewhere, spawning one red ghost. The two corridors are intended to be traversed in opposite directions by the avatar (i.e., they are one-way paths).

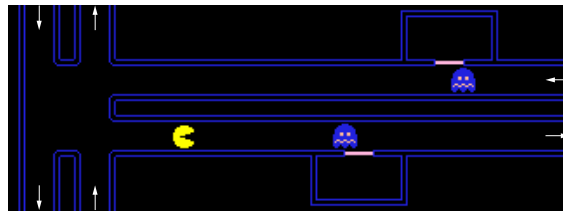
Chase and Scatter modes have the same duration, and all ghosts have the same speed in both modes. Let p be the number of tiles each ghost covers between two mode switches. Frightened mode lasts longer, but ghosts slow down, covering exactly f tiles while in that mode. We make sure that each ghost house is found exactly $d = p + (n + 1)f$ tiles away from its corridor’s entrance, and that each corridor is more than $2d$ tiles long.

As the game starts, the ghosts spawn in front of their respective ghost houses, and start in Chase mode, following the corridor in some direction. Whenever a mode switch occurs, all ghosts reverse direction, and they cannot change it again until the next mode switch, because they never reach a maze intersection. As a result, each ghost “patrols” a portion of length p of its own corridor. By construction, since Frightened mode can be entered at most $n + 1$ times, no ghost may ever leave its corridor.

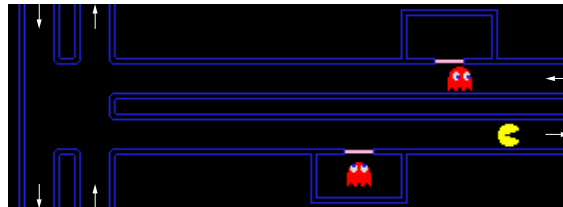
Upon collecting a power pill, the avatar’s speed increases in such a way that it can cover $2d$ tiles (or slightly more) into any adjacent corridor. By doing so, the avatar consumes a token and, if the corridor is traversed in the proper direction, the ghost is necessarily encountered and sent back to the ghost house. By the time the avatar has reached the end of the corridor, the power pill’s effects expire and the



(a) Initial configuration, approached from the top.



(b) Collecting the power pill to traverse the corridor.



(c) Exiting to the right while the ghosts recover.

Figure 12: Sketch of a location and toll road for Pac-Man.

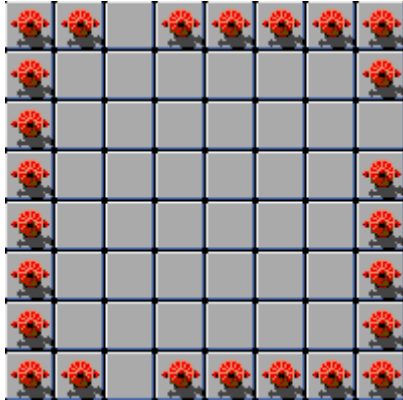
ghost comes out of the ghost house, making the toll road functional again.

3.8 Pipe Mania (The Assembly Line, 1989) is NP-complete

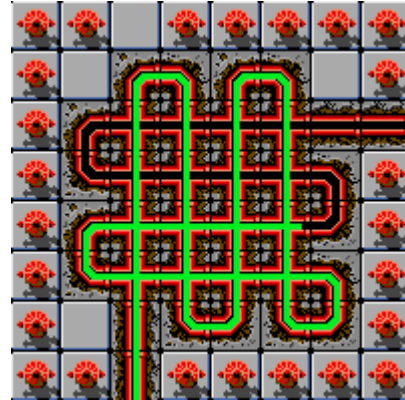
Game description. Not to be confused with KPlumber, with a similar theme but much different mechanics [7], in this puzzle game a long-enough pipe has to be constructed out of several pieces, randomly presented in a queue, starting from a given *source location*. After a timer expires, a stream of water starts flowing from the source into the pipes, and the game is won if and only if the stream traverses a given number of tiles before spilling out.

Since the player can keep constructing pipes on the same tile, “overwriting” the previous pieces until he gets the piece that he wants, he may indeed shape the pipe as he pleases, if the initial timer lasts long enough. Some obstacles are also present in each level, such as fire hydrants, on which pipes cannot be built.

Complexity. Membership in **NP** is obvious. For **NP**-hardness, we apply Metatheorem 1. We use obstacles to model the boundaries of locations and paths, as Figures 13a and 14a illustrate. The resulting paths are necessarily single-use, as only one pipe can fit in them.



(a) Sketch of a location.



(b) Building a pipe that uses most tiles twice.

Figure 13: Enforcing location traversal in Pipe Mania.

We still need to establish location traversal. Suppose we implemented our planar graph with orthogonal lines as edges and squares as nodes. Let ℓ be the total length of the paths plus the area of the starting node, and a be the side length of a generic node. If the number of nodes is $n + 1$, the number of paths is $\Theta(n)$, because the graph is 3-regular (refer to the proof of Metatheorem 1).

Imagine scaling our construction by an integer factor k , in such a way that all paths preserve their unit width, but just increase their length. All the nodes are also scaled in size, except the starting node, which remains constant. Then, the total

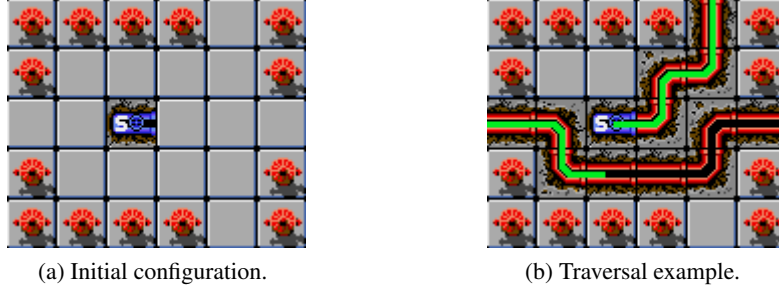


Figure 14: Starting location in Pipe Mania.

length of all paths plus the starting node area becomes $l' = kl - |\Theta(n)|$, and the area of a non-starting node becomes $A = k^2a^2 - |\Theta(ka)|$. When the pipe reaches a generic node, it can cover most of its tiles twice before taking the path toward another node (cross-shaped pieces must be used, see Figure 13b). The length of such a pipe is at least $A' = 2A - |\Theta(ka)| = 2k^2a^2 - |\Theta(ka)|$. Let us set k to a suitable $\Theta(n)$, so that l' becomes negligible compared to A' . Now, it is sufficient to set the required length of the pipe to nA' to ensure that all the nodes will be covered by it.

Recall that the starting node must be traversed twice by the pipe: Figure 14b shows an example of how this can be done.

3.9 Prince of Persia (Brøderbund, 1989) is PSPACE-complete

Game description. The player has to guide an avatar through several dungeon levels, opening gates, fighting guards, and avoiding traps. Most gates are operated by pressure plates. The avatar can walk, run, jump, climb, duck, fight, etc.

Complexity. The game's **PSPACE**-hardness was first proved in [4], but the rather involved construction may be replaced by a somewhat simpler one based on Metatheorem 4.c, which in addition does not rely on gravity, long falls, or on doors that can be opened by more than one pressure plate. To prevent the avatar from avoiding a pressure plate by jumping past it, we simply put it on an elevated tile, which has to be climbed in order to be traversed, as Figure 15 shows. We can even do without vertical walls (as in [4]), because they can be substituted with unopenable gates.

Membership in **PSPACE** = **NPSpace** is quite obvious, as the whole level's configuration can be stored in linear space, and enemy guards have a very simple pseudo-random fighting pattern.

3.10 Puzzle Bobble 3 (Taito, 1996) is NP-complete

Game description. In this Tetris-like puzzle game, levels are made of several colored bubbles, stacked in a hexagonal distribution. The player controls a cannon at the bottom of the screen, which can shoot new bubbles of random colors in any

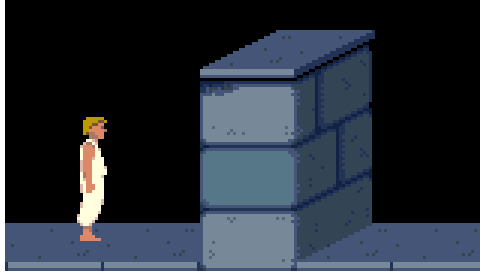


Figure 15: Unavoidable pressure plate in Prince of Persia.

direction. Bubbles attach to each other and, whenever at least three monochromatic bubbles form a connected set as a result of a shot, they pop. Monochromatic triplets may indeed be present in the initial level configuration, and they pop only when hit by a new bubble of the same color.

Some *anchors* hold the whole stack together and, as soon as a bubble is not in the same “connected component” with an anchor, it falls down and is eliminated.

Apart from colored bubbles, there are *stone blocks* that cannot be popped (but may fall if not held up by an anchor), and *rainbow bubbles* that turn the same color of any bubble that pops next to them, and can later be popped like normal bubbles. Notably, if a set of at least three adjacent monochromatic bubbles is formed as a result of some rainbow bubbles turning that color, they immediately pop. This may even induce a “chain reaction” of exploding rainbow bubbles, during which the player is not allowed to shoot a new bubble, and must wait for the explosions to finish.

The goal is to clear all anchors, and an anchor is cleared if and only if no bubble is attached to it.

Complexity. We prove **NP**-hardness by a reduction from PLANAR 3-SAT. Several variable gadgets (Figure 16) are stacked on top of each other, slightly staggered, on the far left of the construction. The clause gadgets (Figure 17) are on the right, far above the variable gadgets. To separate *variable layers* from each other and from the clause gadgets, we put long *shields* of stone blocks, extending from each variable gadget to the far right of the construction. The last shield (i.e., the one in the top layer) also extends all around the whole construction, on the right, top and left sides, preventing bubbles shot by the player from bouncing on the sides of the screen. Variables and clauses are connected via carefully shaped *fuses* made of rainbow bubbles, forking and bending as in Figure 16a.

Initially, only the bottom variable gadget is exposed, and the player may choose whether to pop the black or the white bubbles, which correspond to opposite truth values. Popping one of the two sets —say, the black one— causes three rainbow bubbles to turn black and pop immediately after. This triggers a chain reaction, in which at least three new rainbow bubbles turn black and pop at each step, consuming the fuse and eventually reaching the clause gadgets. At this point, a thin

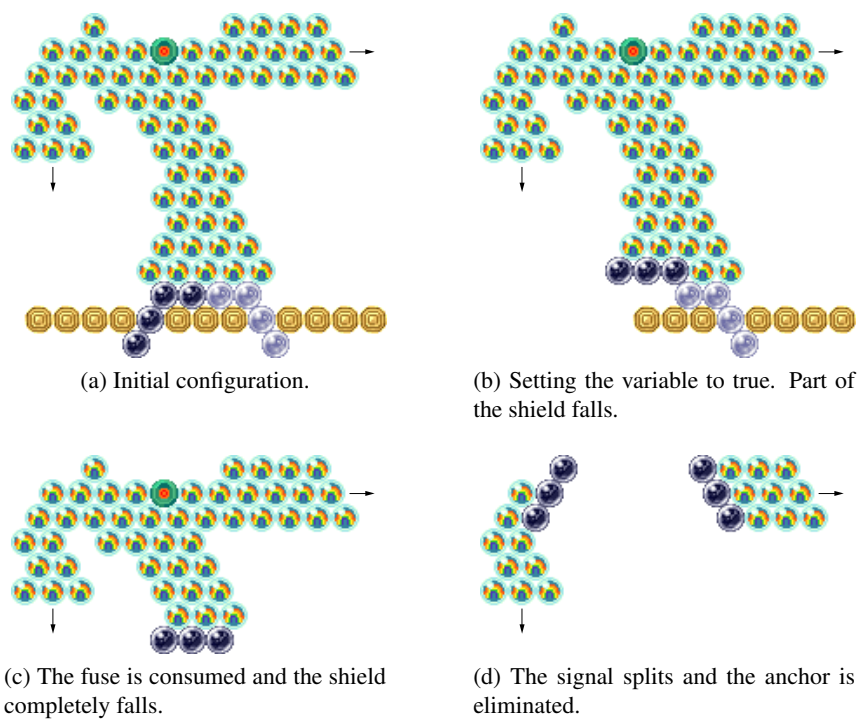


Figure 16: Variable gadget for Puzzle Bobble 3, exemplifying a fork of the fuse.

colored *wire* is reached in every clause gadget (see Figure 17), which pops if and only if it is black (its color tells whether the corresponding literal in the clause is positive or negative). If it pops, the explosion propagates inside the clause gadget, eliminating the anchor. Notice that the explosion can never “backfire” from the clause gadget and consume fuses corresponding to different variables, because each wire is connected to only two rainbow bubbles of its attached fuse.

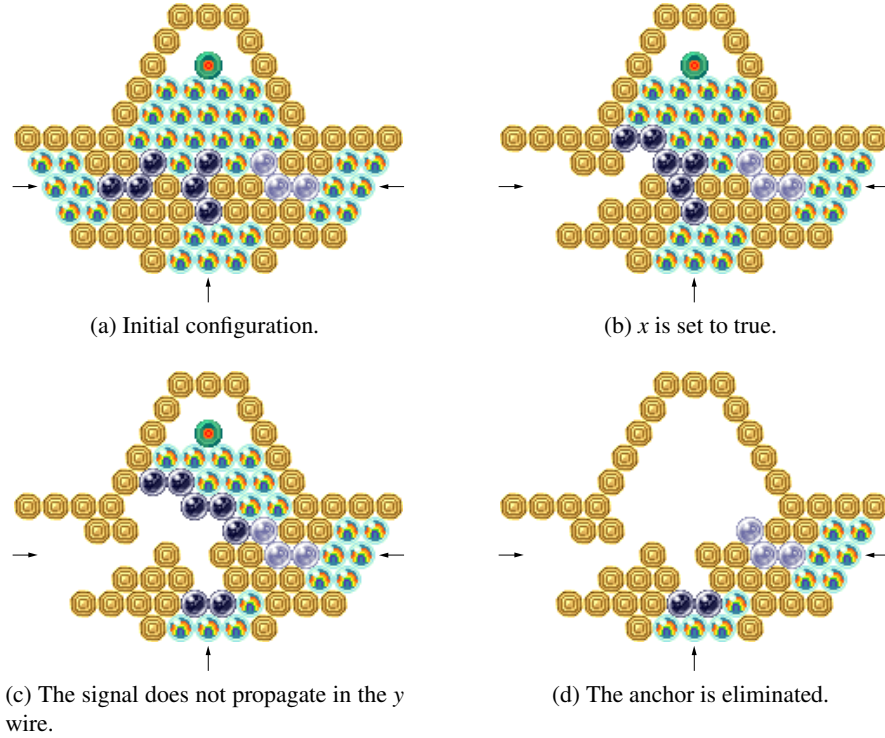


Figure 17: Clause gadget for Puzzle Bobble 3, corresponding to $(x \vee y \vee \neg z)$.

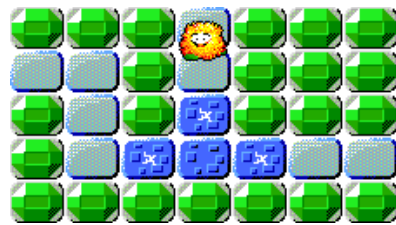
When the fuse of the first variable has been consumed, the remaining part of the variable layer falls, including the shield (see Figure 16). The second variable layer is then exposed, and the process continues until all fuses have been consumed, and all shields have fallen. What eventually remains are the “unsatisfied” clause gadgets, whose wires are now impossible to reach, due to the surrounding *sheaths* made of stone blocks. Notice that each variable layer has its own anchor, so its shield does not fall until the variable has been set by the player, even if all the clauses connected to that variable have already been satisfied.

This proves **NP**-hardness. Completeness holds under the assumption that the player can always choose the color of his next bubble, which is not far from true in most cases, since bubbles can be either discarded by making them bounce back to the bottom of the screen, or can be stacked somewhere (if done properly, not more than two bubbles per color need to be stacked at once).

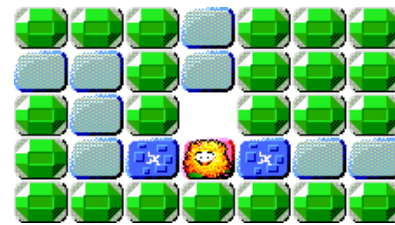
3.11 Skweek (Loriciels, 1989) is NP-hard

Game description. The player controls a furry ball that has to walk on blue tiles in order to paint them pink, while avoiding monsters. Some tiles are made of ice and do not have to be painted, the avatar slides on them and is unable to change direction until it reaches a different type of tile, or its slide is blocked by a wall. Some blue tiles fall apart when the avatar steps on them, opening a hole in the ground that becomes a deadly area. All the blue tiles have to be painted pink within a time limit in order to finish the level. Several power-ups randomly appear, including an exit door and teddy bears of several colors, which let the player immediately skip the level when collected.

Complexity. The decision problem is whether a given level can be completed without losing lives, regardless of the power-ups that may randomly appear. The presence of breakable tiles yields an immediate application of Metatheorem 1. Figure 18 shows how a location is constructed: location traversal is implied by the blue tiles, all of which have to be covered by the avatar. On the other hand, after traversing a path connecting two locations, the cracked tiles break and cannot be accessed again, making it a single-use path. The reason why we use ice tiles is merely that they need not be painted, so the player's purpose is in fact to visit all locations, rather than all paths. For this reason, even though there exists a power-up that prevents the avatar from sliding on ice, our construction still works as intended.



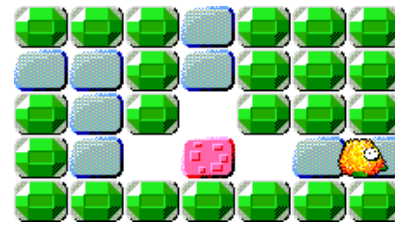
(a) Initial configuration, with the avatar approaching from the top path.



(b) Breaking the top tile and painting the central tile.



(c) Reaching the left tile and running back while it breaks.



(d) Breaking the last tile and exiting to the right, leaving no blue tile behind.

Figure 18: Implementing location traversal and single-use paths in Skweek.

Proving membership in **NP** would be almost straightforward, were it not for a

certain type of monster that turns tiles from pink to blue. On top of this, monster behavior is pseudo-random and partly depends on the player's moves. For these reasons, beating a level may conceivably take an exponentially long time, and any proof that the game lies in **NP** would have to be carefully crafted.

3.12 Starcraft (Blizzard Entertainment, 1998) is NP-hard

Game description. Starcraft is a real-time strategy game in which two or more players have to train an army in order to destroy each other's bases. Two types of resources can be gathered from the environment by special units called *workers*. Resources allow to make new buildings in order to train more units, thus forming an army that can be sent to war. There are three possible *races* to choose, each of which has its unique unit types, each one with different parameters, such as hit points, range, damage, speed, etc. Some units have special abilities, such as becoming invisible, casting offensive or defensive "spells", etc. A player loses if and only if all his buildings are destroyed, regardless of the amount of units that he still has, or the amount of resources he gathered.

Complexity. Most RTS games are expected to be **EXP**-hard, since they involve at least two players, and a match may last an arbitrarily long time. However, a simple **NP**-hardness proof can be given via Metatheorem 2.a. The same reasoning applies, with minor changes, to several RTS games other than Starcraft, such as Warcraft and Age of Empires.

We produce a configuration in a Protoss vs. Terran game, in which deciding whether the Protoss player can win or the game is a draw is **NP**-hard.

In our setting, the avatar will be a Protoss Probe (i.e., the Protoss race's worker unit). When several Probes will be found in the same area, we will identify one as the avatar, and all the other Probes will represent tokens carried by the avatar.

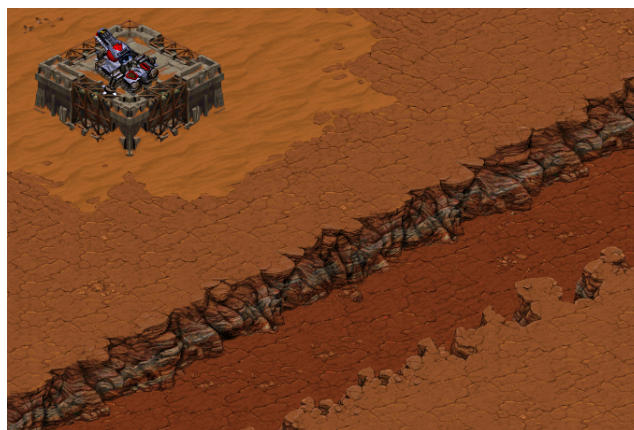


Figure 19: Toll road for Starcraft.

Consistently with our avatar-token abstraction, we present a toll road in Figure 19. If a lone Probe attempts to walk in the canyon, it is destroyed by a single shot of the enemy Siege Tank. But if two Probes walk together, only one can be targeted and destroyed, whereas the second Probe can make it past the Siege Tank while it reloads (the two Probes should stay a couple of tiles away from each other, to avoid splash damage). Paraphrasing, the avatar can traverse the toll road if and only if it is carrying a token. It does not matter which Probe is destroyed, because they are all equivalent. In general, if several Probes attempt to traverse the canyon together, at least one is destroyed, and at least one survives.

Figure 20 shows how to implement a token lying in some location. The building is a Protoss Nexus, and there is a Probe trapped behind a Mineral Field, worth exactly 100 Minerals. The Probe can gather up to eight Minerals at once, but then it must bring them to a Nexus before it can gather more Minerals. It follows that the trapped Probe cannot free itself, but it must wait for another Probe to set it free by bringing all the Minerals to the Nexus.

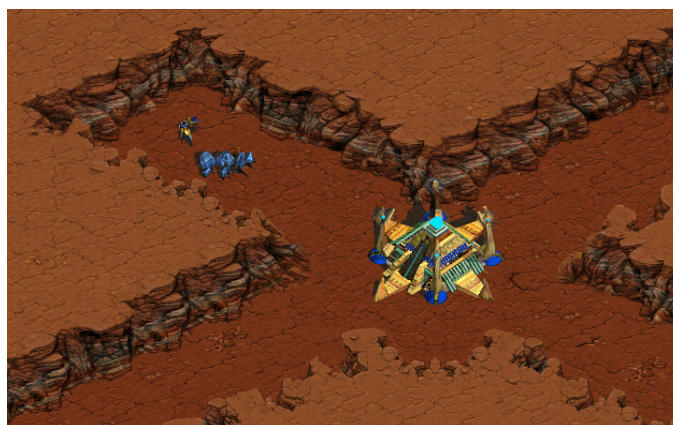


Figure 20: Implementing tokens in Starcraft.

In our analogy, gathering all the Minerals in a location to set a Probe free corresponds to picking up a token. If now we add a free Probe to the designated starting location, we are effectively placing an avatar there, which must set a new Probe free every time it needs to traverse a toll road.

We still have to enforce location traversal and ensure that no new Probes are trained. Recall from the proof of Metatheorem 2.a that there are $n - 1$ locations with one token, plus a starting location with two tokens, and a final location with no tokens. Therefore, in our generated map, the starting location has two Mineral Fields, and the final location has none. We place a Nexus and a Mineral Field in the final location as well (but no Probe, i.e., no token), so that there are $100(n + 2)$ Minerals in total, and at least 100 Minerals in each location.

In a different part of the map, we place $n + 2$ copies of the “crater” shown in Figure 21. In each crater, there is a Protoss Gateway supported by a Protoss Pylon, and a Terran Supply Depot. This completes our construction.



Figure 21: A Zealot must be trained to destroy the Supply Depot.

It is clear that the Terran player has no way to gather resources or train new units, in that he has no workers, and his only buildings are $n + 2$ Supply Depots. Moreover, each Siege Tank is bound to stay on a small platform, and switching from Siege mode to Tank mode has no use, because the shortened attack range would not allow it to hit any Probe.

On the other hand, let us assume that the Protoss player starts with no resources, either. Because there are no Vespene Geysers on the map, and only Minerals are available, there are only two kinds of units that the Protoss player may ever train: Probes from Nexi and Zealots from Gateways. Because none of them can fly and both are melee units, it follows that a Zealot must be trained from each Gateway in order to destroy the nearby Supply Depot. Since training a Zealot costs 100 Minerals, all the Minerals in every location must be gathered and spent just for training Zealots. Therefore, no additional buildings may be built, and no additional Probes may be trained at the Nexi. In other terms, just the initial avatar can be used, and all the locations must be reached, which implies the location traversal feature.

3.13 Tron (Bally Midway, 1982) is NP-hard

Game description. There are four subgames, one of which is a “light cycle” race between the player and several opponents. The race takes place in a rectangular grid whose external boundary is a deadly obstacle. The trail of each light cycle becomes a deadly obstacle as well, hence the safe areas become narrower and narrower as the race progresses. As soon as a light cycle hits an obstacle, it is eliminated, and also its trail is removed from the grid. The goal is to remain the sole survivor in the arena.

Complexity. This game becomes **PSPACE**-complete if played on abstract graphs [8] whereas, for the standard plane grid version, a simple **NP**-hardness proof can be given, as an application of Metatheorem 1.

First we properly embed the location-path structure into the grid, and then we show how to actually implement each component with light cycle trails.

We start from a grid-aligned embedding in which each node is a square, and

paths have unit width. Then we scale the construction up by some large-enough factor, while preserving the size of the nodes, and keeping all the paths of unit width. We do so to make the total area of all nodes negligible compared to the area of a face of the underlying plane graph.

Next we perform the same operation that we did for Pipe Mania (see above): with the same notation, we scale the construction by a factor $k = \Theta(n)$, so that the resulting combined path length l' is negligible with respect to the area of a single node A . (In contrast with Pipe Mania, though, the starting node counts as a regular node and its area does not contribute to l or l' .)

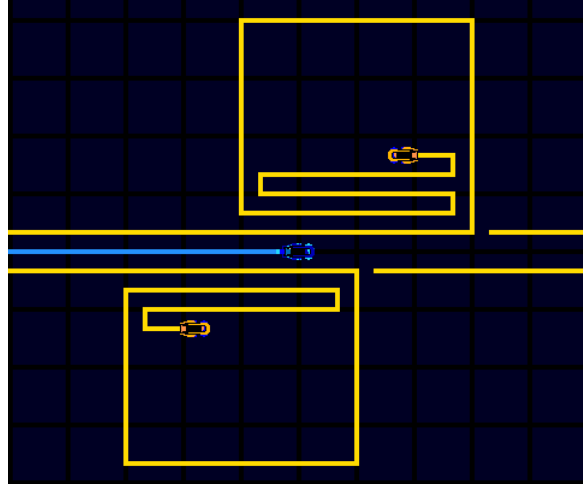


Figure 22: Sketch of a single-use path for Tron.

Then we proceed by implementing paths and locations, as sketched in Figure 22. Each opponent light cycle is responsible for drawing the border of a face of the plane graph underlying our construction (including the outer face), which is a grid-aligned polygon. When a light cycle is done drawing and meets its own trail again, it turns around and “traps” itself in a rectangle of area $(n + 1)A$ (or slightly smaller) inside the polygon it just outlined. This rectangle necessarily fits somewhere in the polygonal face, by the first step of the above construction. In Figure 22 we see a path, traversed by the player’s light cycle, which is bordered by two faces, the upper face arguably having a smaller perimeter than the lower one (because the upper rectangle is bigger, and a larger part of it has been covered).

While paths and nodes are constructed, we assume that the player “waits” by covering a small square in the starting node. This is feasible, because the perimeter of any face is much smaller than the area of a node, by construction. Then the actual race starts, and the player has to cover enough locations to survive longer than his opponents. Paths are obviously single-use, because they have unit width, and the player’s trail is an obstacle even for the player himself. Location traversal is implied by the fact that the player’s light cycle must cover at least a length of slightly less than $(n + 1)A$, so it must visit all nodes.

References

- [1] <http://home.comcast.net/~jpittman2/pacman/pacmandossier.html>.
- [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] E. Friedman. Pushing blocks in gravity is NP-hard. Manuscript, <http://www2.stetson.edu/~efriedma/papers/gravity.pdf>, 2002.
- [6] M. R. Garey, D. S. Johnson, L. Stockmeyer. Some simplified NP-complete problems. In *Proceedings of STOC'74*, 47-63, 1974.
- [7] G. Kendall, A. Parkes, and K. Spoerer. A survey of NP-complete puzzles. *International Computer Games Association Journal*, 31:13–34, 2008.
- [8] T. Miltzow. Tron, a combinatorial game on abstract graphs. In *Proceedings of FUN'12*, 293–304, 2012.
- [9] C. H. Papadimitriou. *Computational complexity*. Addison-Wesley Publishing Company, Inc., 1994.
- [10] J. Plesník. The NP-completeness of the Hamiltonian cycle problem in planar digraphs with degree bound two. *Information Processing Letters*, 8:199-201, 1979.
- [11] O. Reingold. Undirected connectivity in log-space. *Journal of the ACM*, 55:1–24, 2008.
- [12] G. Viglietta. Gaming is a hard job, but someone has to do it! In *Proceedings of FUN'12*, 357–367, 2012.
- [13] G. Viglietta. Lemmings is PSPACE-complete. Manuscript, <http://arxiv.org/abs/1202.6581>, 2012.