Classic Nintendo Games are (Computationally) Hard*

Greg Aloupis[1]**, Erik D. Demaine[2], Alan Guo[2]***, and Giovanni Viglietta[3]

[1] Département d'Informatique, Université Libre de Bruxelles, aloupis.greg@gmail.com
[2] MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, USA, {edemaine,aguo}@mit.edu
[3] School of Electrical Engineering and Computer Science, University of Ottawa, Canada, viglietta@gmail.com

Abstract. We prove NP-hardness results for five of Nintendo's largest video game franchises: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokémon. Our results apply to generalized versions of Super Mario Bros. 1, 3, Lost Levels, and Super Mario World; Donkey Kong Country 1–3; all Legend of Zelda games; all Metroid games; and all Pokémon role-playing games. In addition, we prove PSPACE-completeness of the Donkey Kong Country games and several Legend of Zelda games.

1 Introduction

A series of recent papers have analyzed the computational complexity of playing many different video games [1,4,5,6], but the most well-known classic Nintendo games have yet to be included among these results. In this paper, we analyze some of the best-known Nintendo games of all time: Mario, Donkey Kong, Legend of Zelda, Metroid, and Pokémon. We prove that it is NP-hard, and in some cases PSPACE-hard, to play generalized versions of most games in these series. In particular, our NP-hardness results apply to the NES games Super Mario Bros., Super Mario Bros.: The Lost Levels, Super Mario Bros. 3, and Super Mario World (developed by Nintendo); to the SNES games Donkey Kong Country 1–3 (developed by Rare Ltd.); to all Legend of Zelda games (developed by Nintendo);[4] to all Metroid games (developed by Nintendo); and to all Pokémon role-playing games (developed by Game Freak and Creatures Inc.).[5] Our PSPACE-hardness results apply to to the SNES games Donkey Kong Country 1–3, and to The

* Full paper available as arXiv:1203.1895, http://arXiv.org/abs/1203.1895

** Chargé de Recherches du FNRS. Work initiated while at Institute of Information Science, Academia Sinica.

*** Partially supported by NSF grants CCF-0829672, CCF-1065125, and CCF-6922462.

[4] We exclude the Zelda CD-i games by Philips Media, which Nintendo does not list as part of the Legend of Zelda series.

[5] All products, company names, brand names, trademarks, and sprites are properties of their respective owners. Sprites are used here under Fair Use for the educational purpose of illustrating mathematical theorems.

Legend of Zelda: A Link to the Past. Some of the aforementioned games are also complete for either NP or PSPACE. All of these results are new.[6]

For these games, we consider the decision problem of reachability: given a stage or dungeon, is it possible to reach the goal point t from the start point s? Our results apply to generalizations of the games where we *only* generalize the map size and leave all other mechanics of the games as they are in their original settings. Most of our NP-hardness proofs are by reduction from 3-SAT, and rely on a common construction. Similarly, our PSPACE-completeness results for Legend of Zelda: A Link to the Past and Donkey Kong Country games are by a reduction from True Quantified Boolean Formula (TQBF), and rely on a common construction (inspired by a metatheorem from [5]). In addition, we show that several Zelda games are PSPACE-complete by reducing from PushPush-1 [3].

We can obtain some positive results if we bound the "memory" of the game. For example, recall that in Super Mario Bros. everything substantially off screen resets to its initial state. Thus, if we generalize the stage size in Super Mario Bros. but keep the screen size constant, then reachability of the goal can be decided in polynomial time: the state space is polynomial in size, so we can simply traverse the entire state space and check whether the goal is reachable. Similar results hold for the other games if we bound the screen size in Donkey Kong Country or the room size in Legend of Zelda, Metroid, and Pokémon. The screen-size bound is more realistic (though fairly large in practice), while there is no standard size for rooms in Metroid and Pokémon.

Membership in PSPACE. Most of the games considered are easy to show belong to PSPACE, because every game element's behavior is a simple (deterministic) function of the player's moves. Therefore, we can solve a level by making moves nondeterministically while maintaining the current game state (which is polynomial), and use that NPSPACE = PSPACE.

Some other games, such as Legend of Zelda and its sequels, also include enemies and other game elements that behave pseudorandomly. As long as the random seed can be encoded in a polynomial number of bits, which is the case in all reasonable implementations, the problem remains in PSPACE.

Game model and glitches. We adopt an idealized model of the games in which we assume that the rules of the games are as (we imagine) the game developers intended rather than as they are implemented. In particular, we assume the absence of major game-breaking glitches (for an example of a major game-breaking glitch, see [17], in which the speed runner "beats" Super Mario World in less than 3 minutes by performing a sequence of seemingly arbitrary and nonsensical actions, which fools the game into thinking the game is won). We view these glitches not as inherently part of the game but rather as artifacts of imperfect

[6] A humorous paper (http://www.cs.cmu.edu/~tom7/sigbovik/mariox.pdf) and video (http://www.youtube.com/watch?v=HhGI-GqAK9c) by Vargomax V. Vargomax claims that "generalized Super Mario Bros. is NP-complete", but both versions have no actual proof, only nonsensical content.

implementation. However, in the case of Super Mario Bros., the set of glitches has been well-documented [16] and we briefly show how our constructions can be modified to take into account glitches that would otherwise break them.

Organization. Due to space constraints, we give here only cursory proofs and only for one game per franchise. In Section 2, we present two general schematics used in almost all of our NP-hardness and PSPACE-hardness reductions. In Section 3, we prove that generalized Super Mario Bros. is NP-hard by constructing the appropriate gadgets for the construction given in Section 2. In Sections 6, and 7, we do the same for generalized Metroid, and Pokémon, respectively. Sections 4 and 5 show that the generalized Donkey Kong Country and generalized Legend of Zelda: A Link to the Past are PSPACE-complete, again by constructing the appropriate gadgets introduced in Section 2.

2 Frameworks for Platform Games

2.1 Framework for NP-hardness

We use a general framework for proving the NP-hardness of platform games, illustrated in Figure 1.

The framework reduces from the classic NP-complete problem 3-SAT: decide whether a 3-CNF Boolean formula can be made "true" by setting the variables appropriately. The player's character starts at the position labeled Start, then proceeds to the Variable gadgets. Each Variable gadget forces the player to make an exclusive choice of "true" (x) or "false" ($\neg x$) value for a variable in the formula. Either choice enables the player to follow paths leading to Clause gadgets,

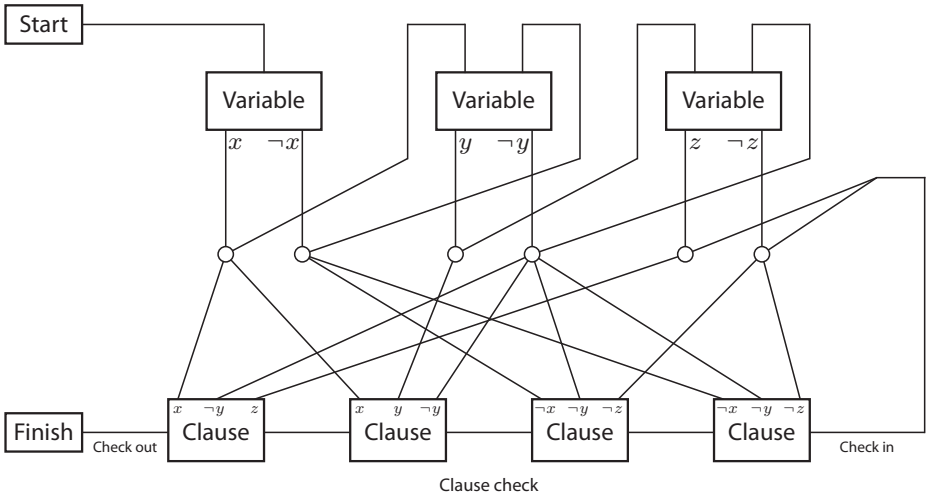

Fig. 1: General framework for NP-hardness

corresponding to the clauses containing that literal (x or $\neg x$). These paths may cross each other, but Crossover gadgets prevent the player from switching between crossing paths. By visiting a Clause gadget, the player can "unlock" the clause (a permanent state change), but cannot reach any of the other paths connecting to the Clause gadget. Finally, after traversing through all the Variable gadgets, the player must traverse a long "check" path, which passes through each Clause gadget, to reach the Finish position. The player can get through the check path if and only if each clause has been unlocked by some literal. Therefore, it suffices to implement Start, Variable, Clause, Finish, and Crossover gadgets to prove NP-hardness of each platform game.

Remark 2.1. The Crossover gadget only needs to be *unidirectional*, in the sense that each of the two crossing paths needs to be traversed in only one direction. This is sufficient because, for each path visiting a clause from a literal, instead of backtracking to the literal after visiting the clause, we can reroute directly to visit the next clause, so the player is never required to traverse a literal path in both directions.

Remark 2.2. It is safe to further assume in a Crossover gadget that each of the two crossing paths is traversed at most once, and that one path is never traversed before the other path (i.e., if both paths are traversed, the order of traversal is fixed). This is sufficient because two literal paths either are the two sides of the same Variable (and hence only one gets traversed), or they come from different Variables, in which case the one from the earlier Variable in the sequence is guaranteed to be traversed before the other (if it gets traversed at all). Thus it is safe to have a Crossover gadget, featuring two crossing paths A and B, which after traversing path B allows leakage from A to B. (However, leakage from B to A must still be prevented.)

2.2 Framework for PSPACE-hardness

For the PSPACE-hardness of Donkey Kong Country and Zelda: A Link to the Past, we apply a modified version of a framework described in [5, Metatheorem 2.c] and [6]. That framework reduces from the PSPACE-complete problem True Quantified Boolean Formula (TQBF), and involves some *doors*, which may be open or closed, and *pressure plates*, which open or close arbitrary doors as the player walks on them. Each pressure plate operates only one door.

By inspecting the reduction in [5, Metatheorem 2.c], we observe that, for each door, there is only one pressure plate opening it, and only one closing it. Moreover, it is evident that we may even allow the player to decide to "skip" a pressure plate that *opens* a door, because skipping it is never a good move (indeed, opening a door can only make new areas accessible). Hence, here we adopt a Door gadget that also incorporates the mechanisms to open and close it, as shown in Figure 2.

Three distinct paths enter the gadget from the left and exit to the right, without leakage. The "traverse" path implements the actual door, and may be

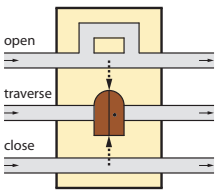

Fig. 2: Door gadget

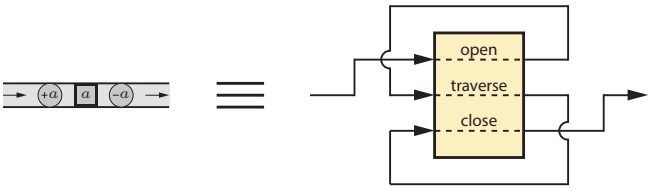

Fig. 3: Implementing doors and pressure plates with Door gadgets

traversed if and only if the gadget is in the open state. The other two paths allow to operate the door: as the player walks in the "close" path, the door closes; while as they walk in the "open" path, they are *allowed* to make the door open, but they may choose not to.

Figure 3 illustrates how to implement the framework in [5, Metatheorem 2.c] with our Door gadgets. Note that we need Crossover gadgets to do this.

3 Super Mario Bros.

Theorem 3.1. *It is NP-hard to decide whether the goal is reachable from the start of a stage in generalized Super Mario Bros.*

Proof. When generalizing the original Super Mario Bros., we assume that the screen size covers the entire level, because the game forbids Mario from going left of the screen. This generalization is not needed in later games, because those games allow Mario to go left. Figures 4, 5, 6, 7, and 8 shows all the gadgets. □

Glitches. Documentation on glitches present in Super Mario Bros. can be found in [16], which also describes how to recreate and abuse these glitches. Here we address two types of glitches that break our construction.

The first type allows Mario to walk through walls (for examples, see "Application: Jump into a wall just below a solid ceiling and walk through it" and

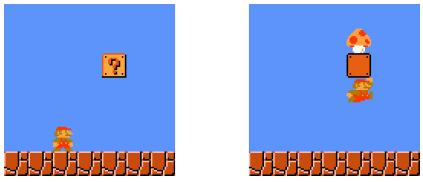

Fig. 4: Left: Start gadget for Super Mario Bros. Right: The item block contains a Super Mushroom

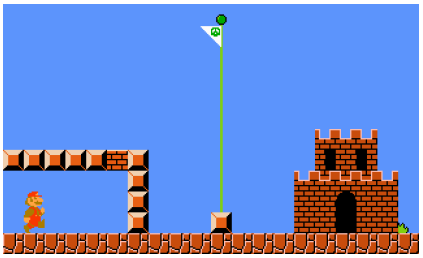

Fig. 5: Finish gadget for Super Mario Bros.

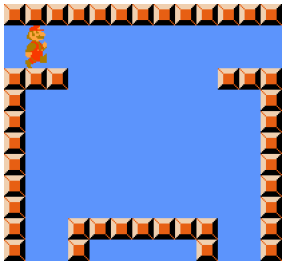

Fig. 6: Variable gadget for Super Mario Bros.

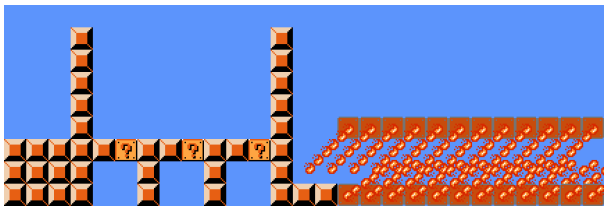

Fig. 7: Clause gadget for Super Mario Bros. The item blocks contain Power Stars

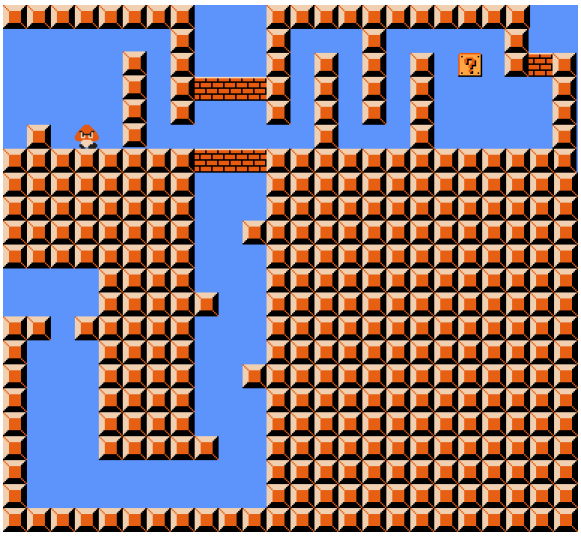

Fig. 8: Crossover gadget for Super Mario Bros.

"Application: Jump into a solid wall and walk through it" in [16]). This would break almost all of our gadgets because they depend on Mario's inability to walk through walls. Fortunately, our constructions can easily be fixed to address this issue as follows; see Figure 9. We replace a one-tile-wide wall with a much thicker wall and place an enemy in each row, preventing Mario from walking through the wall (except perhaps the topmost tile) without getting hurt.

The second type of glitch allows Mario to perform wall jumps, i.e., jump off the sides of walls to reach high places. This could potentially break one-way paths in our construction, which consist of very long falls. Fortunately, we can fix this by transforming our one-way paths as shown in Figure 10: widen the tunnel and place blocks on the sides so that, even if Mario tries to wall jump, he will eventually run into a block above him, preventing him from jumping any higher.

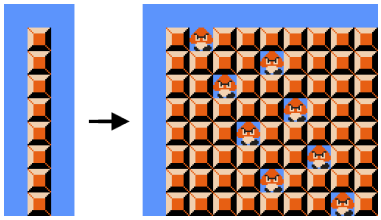

Fig. 9: Wall transformation for Super Mario Bros.

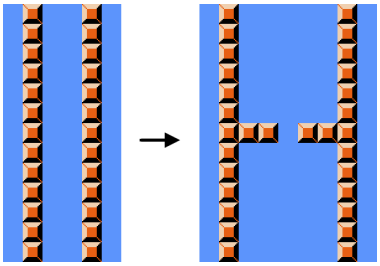

Fig. 10: One-way transformation for Super Mario Bros.

4 Donkey Kong Country

Theorem 4.1. *It is PSPACE-complete to decide whether the goal is reachable from the start of a stage in generalized Donkey Kong Country 1.*

Proof. We may assume that the player controls only a single Kong, by placing a DK barrel (a barrel containing the backup Kong member) at the start of the level, followed by a wall of red Zingers (which are not killable by Barrels). The Door gadget is illustrated in Figure 11. We use a Tire to model the open/closed state of the gadget, and moving swarms of Zingers to control the movements of the player. The door is closed if the Tire is located as shown in the picture, and is open if it is located up the slide. The ground is made of ice, so that both the Tire and the player slide on it when they gain some speed. The right-facing Zingers are static, while the left-facing ones move from left to right in swarms, as indicated by arrows. □

5 The Legend of Zelda

Several Zelda games—Ocarina of Time, Majora's Mask, Oracle of Seasons, The Minish Cap, and Twilight Princess—contain dungeons with ice blocks, which are pushed like normal blocks, except when pushed they slide all the way until they encounter an obstacle. These games therefore include as a special case PushPush-1 [3], which is PSPACE-complete. More interesting, we show:

Theorem 5.1. *It is PSPACE-complete to decide whether a given target location is reachable from a given start location in generalized Legend of Zelda: A Link to the Past.*

Proof. The Door gadget is depicted in the upper part of Figure 12. We use Switch-operated Gates (each Switch alternately opens and closes the Gate with the same number), and one-way Teleporters. Since all Gates in Legend of Zelda are initially closed, we first make the player traverse all the "initialize" paths in every Door gadget, which causes all Gates labeled '2' to open. The tiles labeled

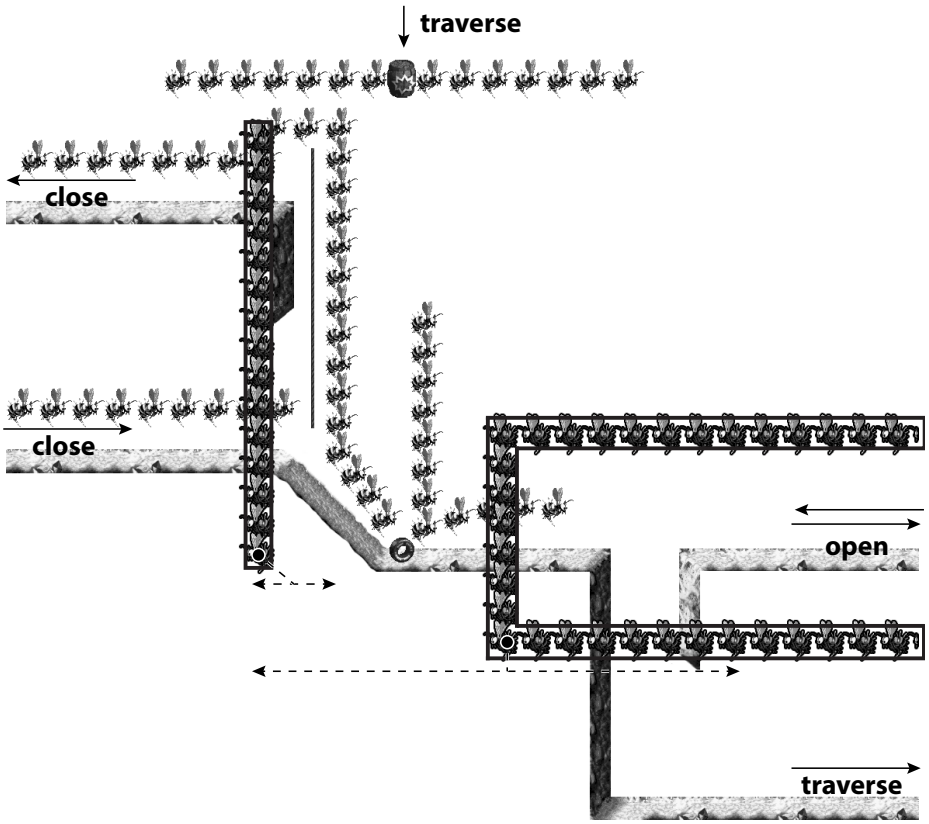

Fig. 11: Door gadget for Donkey Kong Country 1

'a' (resp. 'b') are implemented as lowered (resp. raised) Pillars, and can (resp. cannot) be traversed. When all the Door gadgets have been initialized, the gadget in the bottom part of Figure 12 is reached, which contains a Crystal Switch that toggles the raised-lowered state of all the Pillars (effectively changing every 'a' into a 'b', and vice versa). From there, the player may proceed to the "start" path, and the actual starting location of the level. □

6 Metroid

Theorem 6.1. *It is NP-hard to decide whether a given target location is reachable from a given start location in generalized Metroid.*

Proof. The Clause and Crossover gadgets are illustrated in Figures 13 and 14 respectively. In the Clause gadget, Samus can kill all the Zoomers from below to enable later traversal in Morph Ball mode. In the Crossover gadget, Samus

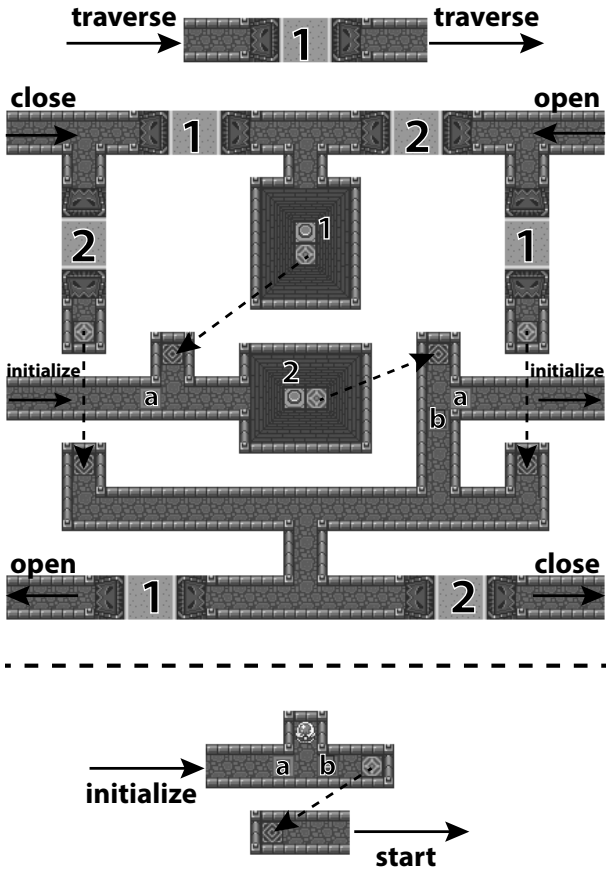

Fig. 12: Door gadget for Zelda

waits for a gap in the Zoomers in an upper area, then she can follow the Zoomers toward the center of the gadget, and fall down onto the lower platform. This platform is traversed by two streams of Zoomers, going in opposite directions, timed in such a way that, if Samus comes from the upper-left (respectively, upper-right) platform, she is forced to go right (respectively, left) to run away from the Zoomers. □

7 Pokémon

Theorem 7.1. *It is NP-complete to decide whether a given target location is reachable from a given start location in generalized Pokémon in which the only overworld game elements are enemy Trainers.*

Proof. In our implementations, we use three kinds of objects. Walls, represented by dark grey blocks, cannot be occupied or walked through. Trainers' lines of

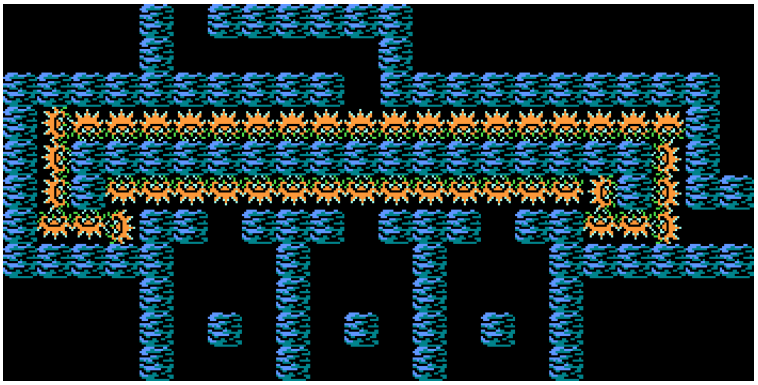

Fig. 13: Clause gadget for Metroid

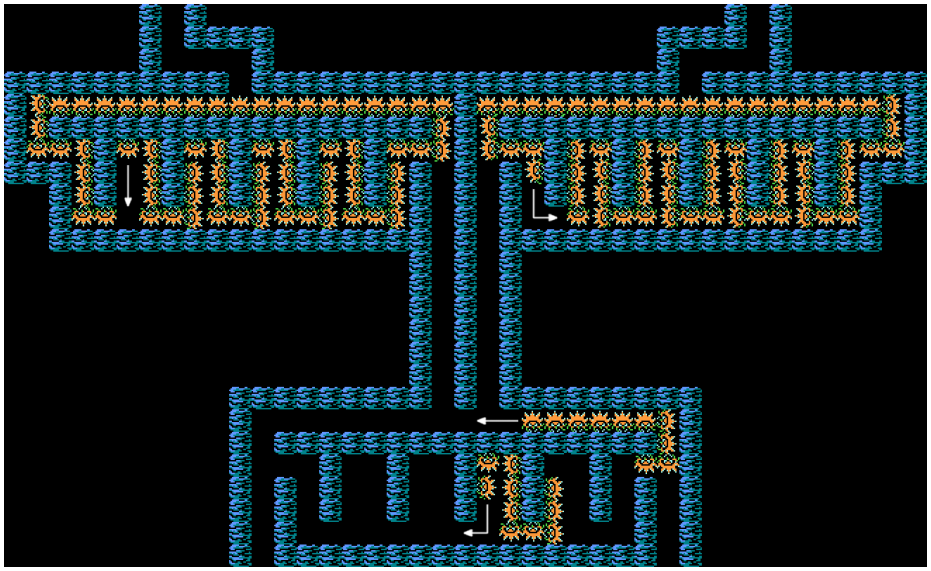

Fig. 14: Crossover gadget for Metroid

sight are indicated by translucent rectangles. We have two types of Trainers. Weak Trainers, represented by red rectangles, are Trainers whom the player can defeat with certainty without expending any effort, i.e., without consuming PP or taking damage. Strong Trainers, represented by blue rectangles, are Trainers against whom the player will always lose. The gadgets are illustrated in Figures 15, 16, 17, and 18. □

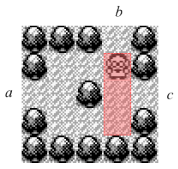

Fig. 15: Variable gadget for Pokémon

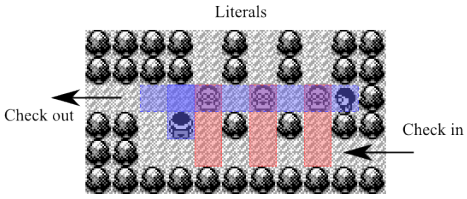

Fig. 16: Clause gadget for Pokémon

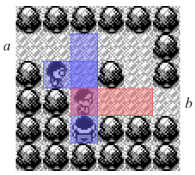

Fig. 17: Single-use path for Pokémon

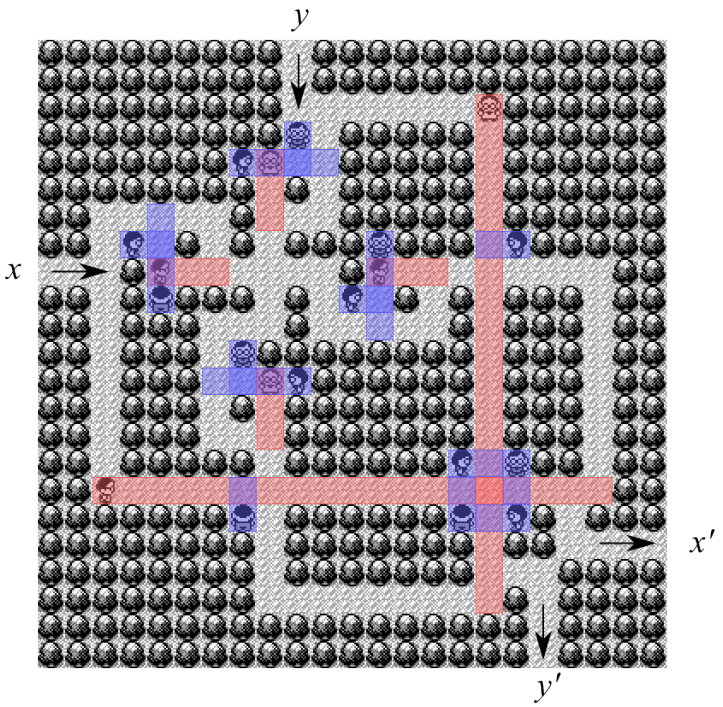

Fig. 18: Crossover gadget for Pokémon

References

1. Graham Cormode. The hardness of the Lemmings game, or Oh no, more NP-completeness proofs. In *Proceedings of the 3rd International Conference on Fun with Algorithms*, May 2004, pages 65–76.
2. Erik D. Demaine, Martin L. Demaine, and Joseph O'Rourke. PushPush and Push-1 are NP-hard in 2D. In *Proceedings of the 12th Annual Canadian Conference on Computational Geometry*, August 2000, pages 211–219.

3. Erik D. Demaine, Michael Hoffmann, and Markus Holzer. PushPush-k is PSPACE-Complete. In *Proceedings of the 3rd International Conference on Fun with Algorithms*, May 2004, pages 159–170.
4. Michal Forišek. Computational complexity of two-dimensional platform games. In *Proceedings of the 5th International Conference on Fun with Algorithms*, June 2010, pages 214–226.
5. Giovanni Viglietta. Gaming is a hard job, but someone has to do it! In *Proceedings of the 6th International conference on Fun with Algorithms*, June 2012, pages 357–367.
6. Giovanni Viglietta. Lemmings is PSPACE-complete. In *Proceedings of the 7th International conference on Fun with Algorithms*, to appear.
7. http://www.mariowiki.com/Super_Mario_Bros.
8. http://donkeykong.wikia.com/wiki/Donkey_Kong_Country
9. http://www.zeldawiki.org/The_Legend_of_Zelda_(Game)
10. http://www.zeldawiki.org/The_Legend_of_Zelda:_A_Link_to_the_Past
11. http://www.metroidwiki.org/wiki/Metroid_(game)
12. http://spriters-resource.com/
13. http://www.videogamesprites.net/
14. http://www.nesmaps.com/
15. http://www.snesmaps.com/
16. http://tasvideos.org/GameResources/NES/SuperMarioBros.html
17. Masterjun. SNES Super Mario World (USA) "glitched" in 02:36.4, 2012. http://www.youtube.com/watch?v=Syo5sI-iOgY, retrieved April 14, 2012.

Acknowledgments

This work was initiated at the 25th Bellairs Winter Workshop on Computational Geometry, co-organized by Erik Demaine and Godfried Toussaint, held on February 6–12, 2010, in Holetown, Barbados. We thank the other participants of that workshop—Brad Ballinger, Nadia Benbernou, Prosenjit Bose, David Charlton, Sébastien Collette, Mirela Damian, Martin Demaine, Karim Douïeb, Vida Dujmović, Robin Flatland, Ferran Hurtado, John Iacono, Krishnam Raju Jampani, Stefan Langerman, Anna Lubiw, Pat Morin, Vera Sacristán, Diane Souvaine, and Ryuhei Uehara—for providing a stimulating research environment. In particular, Nadia Benbernou was involved in initial discussions of Super Mario Bros.

We thank readers Bob Beals, Curtis Bright, Istvan Chung, Peter Schmidt-Nielsen, Patrick Xia, and the anonymous referees for helpful comments and corrections, and for "beta-testing" our constructions.

We also thank The Spriters Resource [12], VideoGameSprites [13], NES Maps [14], and SNES Maps [15] for serving as indispensable tools for providing easy and comprehensive access to the sprites used in our figures.

Finally, of course, we thank Nintendo and the associated developers for bringing these timeless classics to the world.