Mario Kart is Hard

Jeffrey Bosboom[1], Erik D. Demaine[1], Adam Hesterberg[1], Jayson Lynch[1], and Erik Waingarten[2⋆]

[1] MIT Computer Science and Artificial Intelligence Laboratory, 32 Vassar St., Cambridge, MA 02139, {jbosboom,edemaine,achester,jaysonl}@mit.edu
[2] Department of Computer Science, Columbia University, 1214 Amsterdam Avenue, New York, NY 10027, eaw@cs.columbia.edu

Abstract. Nintendo's Mario Kart is perhaps the most popular racing video game franchise. Players race alone or against opponents to finish in the fastest time possible. Players can also use items to attack and defend from other racers. We prove two hardness results for generalized Mario Kart: deciding whether a driver can finish a course alone in some given time is NP-hard, and deciding whether a player can beat an opponent in a race is PSPACE-hard.

1 Introduction

Mario Kart is a popular racing video game series published by Nintendo, starting with Super Mario Kart on SNES in 1992 and since adapted to eleven platforms, most recently Mario Kart 8 on Wii U in 2014; see Table 1. The series has sold over 100 million game copies, and contains the best-selling racing game ever, Mario Kart Wii [Gui14]. The games feature characters from the classic Nintendo series Super Mario Bros. and Donkey Kong.

In this paper, we analyze the computational complexity of most Mario Kart games, showing that optimal gameplay is computationally intractable. Our results follow a series of recent work on the computational complexity of video games, including the broad work of Forisek [For10] and Viglietta [Vig14] as well as the specific analyses of classic Nintendo games [ADGV15].

In Mario Kart, each player picks a character and a race track. There are three modes of play: players race against each other (racing), a player races alone to finish in the fastest time possible (time trial), and players battle in an arena (battle). We focus here on the first two modes. Each race track features its own set of obstacles and geometry.

⋆ Work performed while at MIT.

	Game Title	Game System	Release Date	Sales	3D?
1.	Super Mario Kart	Super NES	August 27, 1992	8.76M	no
2.	Mario Kart 64	Nintendo 64	December 14, 1996	9.87M	yes
3.	Mario Kart: Super Circuit	Game Boy Advance	July 21, 2001	5.47M	no
4.	Mario Kart: Double Dash!!	Nintendo GameCube	November 7, 2003	6.95M	yes
5.	Mario Kart DS	Nintendo DS	November 14, 2005	23.56M	yes
6.	Mario Kart Wii	Wii	April 10, 2008	35.53M	yes
7.	Mario Kart 7	Nintendo 3DS	December 1, 2011	12.19M	yes
8.	Mario Kart 8	Wii U	May 29, 2014	5.87M	yes
9.	Mario Kart: Arcade GP	arcade	October 2005	?	yes
10.	Mario Kart: Arcade GP 2	arcade	March 14, 2007	?	yes
11.	Mario Kart: Arcade GP DX	arcade	July 25, 2013	?	yes

Table 1. History and total sales [Sal] of Mario Kart. Our results apply to all games with 3D tracks.

A particularly distinctive feature of Mario Kart is that players may acquire items (also known as power-ups). Items temporarily give players special abilities. Each Mario Kart game has its own set of items, but two items are common to all Mario Kart games: Koopa shells and bananas. Koopa shells come in multiple colors; our reduction only uses the green shells, which we refer to simply as shells. Shells are shot at other players and, upon contact, temporarily stun them, reducing their speed and control. Bananas can be dropped by players along the track, and any player who runs over a banana becomes temporarily stunned. Crucially, shells can destroy bananas.

In this paper, we consider generalized versions of time trial and racing. We allow race tracks to be any size and have carefully placed items on the track. We more precisely define our model of the game in Section 2. In Section 3, we show that time trial is NP-hard, that is, it is NP-hard to decide whether a lone player can finish a race track in time at most t. In Section 4, we show PSPACE-hardness of racing: it is PSPACE-hard to decide whether a player can win the race against even a single opposing player. Finally, Section 5 considers upper bounds.

The items used in our reductions are present in all Mario Kart games. Our reductions use the "Rainbow Road" style of racetrack. These tracks are present in every game, but our reductions require them to be three-dimensional, which they are in Mario Kart 64 and in every game since Mario Kart: Double Dash!!. The proofs thus apply to nine of the Mario Kart games (Games 2 and 4–11 in Table 1). Super Mario Kart and Mario Kart Super Circuit lack tracks with multiple altitudes, presumably from the lack of power in the Super NES and Game Boy Advance systems, and so our proofs do not apply to them.

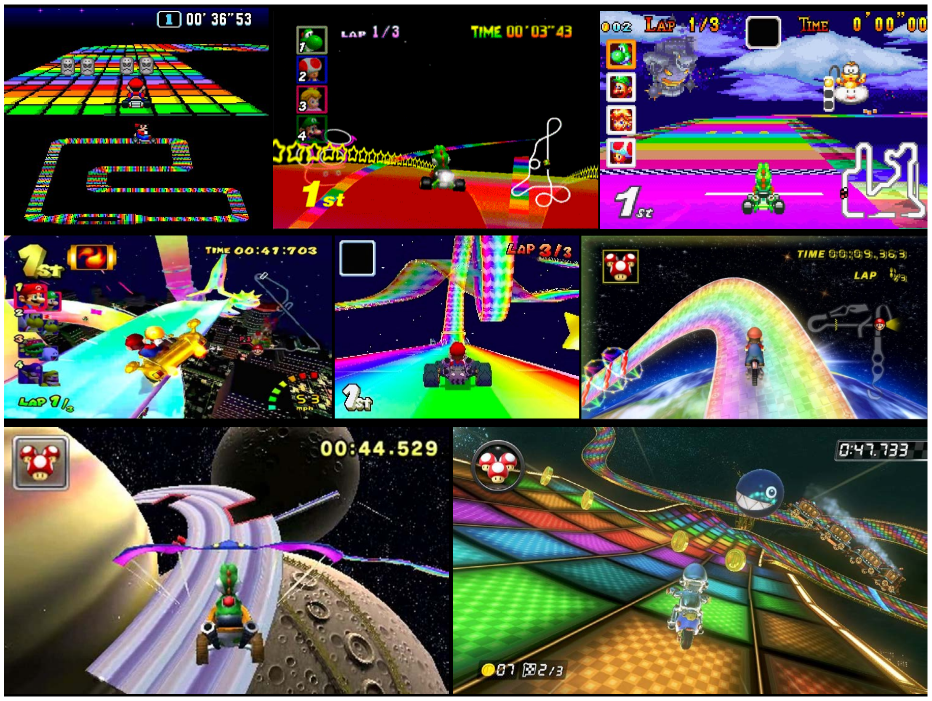


Fig. 1. Screenshots of Rainbow Road tracks from Mario Kart 1–8 (Table 1).

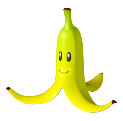
2 Model

In our mathematical model of Mario Kart, each player's state consists of a position, orientation, and speed. The track is a two-dimensional surface in Euclidean 3-space. The player generally controls their acceleration, with limits on speed and position imposed by the track. Leaving the bounds of the racetrack does not result in death, with players being respawned on the track after a significant speed and time penalty.

Computationally, we assume that we can compute the optimal traversal of a track described by a constant number of real parameters, and that this optimal traversal time typically changes continuously with the real parameters. This allows us to, for example, tweak multiple pieces of the track to have nearly identical optimal traversal times. In fact, we require that these assumptions hold only up to an error factor of $1 + O(1/n^c)$, that is, up to $O(\log n)$ bits. We leave to future work the careful analysis of the physics and geometry of actual Mario Kart implementations, and the evaluation of the validity of our assumptions.[3]

Players obtain items from item boxes which are at fixed locations on the track, and regenerate after a fixed amount of time. We use two kinds of items common to all Mario Kart games to date, each of which can be used only once:

1. **Bananas**. Bananas are slippery. When a player drives over a banana (or is hit by one), the driver slips and spins temporarily out of control, resulting in a temporary slowdown. Bananas can be dropped immediately behind the player, or thrown up and ahead with a fixed trajectory. Once a banana lands on the track, there are two ways to remove it: either a player drives over it, or the banana is hit by a shell (described below).



2. **Green Shells**. A green shell is one of the many attacks in Mario Kart. The player can shoot a green shell like a projectile. If a green shell hits a driver, the driver is temporarily knocked out. A green shell can also remove a banana if the banana is hit first. (Green shells should not be confused with red shells, which can lock onto a target driver.) Green shells follow a particular direction, are subject to gravity, and bounce off of walls. After some time, green shells become inactive and disappear.

A driver can possess only one item at a time. For example, if a driver picks up a green shell, s/he cannot pick up another item until s/he uses the green shell. However, in most Mario Kart games (with the notable exception of Mario Kart 8), it is possible to "use" a green shell or banana without throwing it: a driver can hold a green shell or banana behind the car before throwing it, allowing them to pick up one additional item. The items still must be used in order.

In our reductions, we will assume that some bananas have already been placed on the track, but this does not occur in any real Mario Kart

[3] We conjecture that implementations model the position and velocity vector of a player by floating-point numbers, discretize time into fixed-duration intervals, and model the track by a collection of succinctly describable segments and turns. For a sufficiently fine discretization of time, this model should approach our continuous model. To compute the optimal traversal time of a constant-complexity track, we can finitely sample the position/velocity space and search the resulting state graph. We conjecture that a polynomial-resolution sampling suffices to approximate the optimal traversal time to the needed $1 + O(1/n^c)$ accuracy for our reductions.

tracks. In fact, we assume that the game has already been played for some time, e.g., previous laps of the track, and the computational question is whether Player 1 can win within one final lap from the given track configuration. We can easily add "initialization" paths and banana item boxes to the track, ensuring that the initial configuration of placed bananas would actually be reachable from an initially empty track. By making these initialization paths very long, they will not affect optimal play of the final lap under consideration.

In this way, we can also assume that two players start at very different positions on the track. The finish line is shared between the two players, but is fairly wide. Thus we can cross the finish line with two equally elevated and separated paths for the two players, guaranteeing no interaction near the finish, to effectively allow distinct goal locations for the two players.

3 Time Trial is NP-Hard

First we study the following solo ("time trial") variant of Mario Kart:

Theorem 1. *It is* NP*-hard to determine whether a driver can finish a given course in at most t time, in the absence of opponents.*

3.1 Proof Structure

The reduction is from 3SAT. Given a Boolean formula ϕ with variables $x_1, x_2, \ldots, x_n$, we build a level with the "Rainbow Road" style. The driver first drives through each variable gadget in sequence. In each variable gadget, the player can decide whether to set each variable to true or false. After setting all the variables, the driver must traverse each clause gadget. The driver will be able to complete the level without delay if and only if the variable assignments chosen in the gadgets form a satisfying assignment for ϕ.

Figure 2 gives a schematic overview of the reduction. Each node labeled x_i corresponds to a variable gadget, and each node labeled c_i corresponds to a clause gadget. The solid lines correspond to the path in the level. The dashed lines indicate that a variable or its negation is contained in a given clause. In our case, the dashed lines also correspond to clause gadgets being reachable by green shells when thrown from the variable gadgets. We prevent players from following the dashed paths.

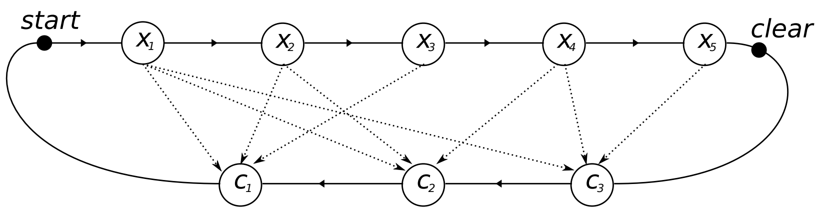


Fig. 2. General reduction structure. Dashed lines correspond to reachability of green shells.

3.2 Variable Gadget

For each variable x_i, we have one variable gadget as shown in Figure 3. The variable gadget first splits the road into two. The driver must choose which of the two directions to follow, corresponding to the truth setting of x_i. We refer to the two split roads as *literal roads* x_i and $\overline{x_i}$. Both literal roads have the same optimal travel time.

Each literal road has a sequence of visits to clause gadgets corresponding to clauses containing the literal. Literal road x_i goes above the clauses containing the literal x_i, and similarly for $\overline{x_i}$. Each road has a green shell item which can be fired into the clause gadget. When a literal road is above each clause, the driver can pick up a green shell and shoot it down to the clause, where it will remove a banana.

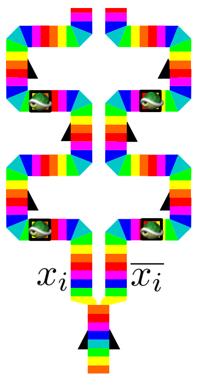


Fig. 3. A variable gadget where Player 1 assigns x_i. Player 1 goes left to set x_i to true, and goes right to set $\overline{x_i}$ to true.

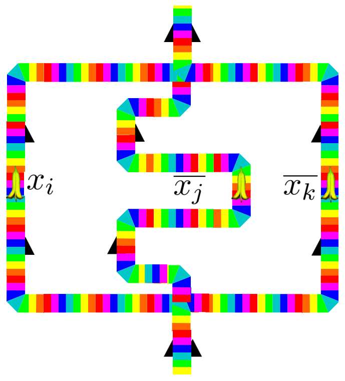


Fig. 4. Clause gadgets split into three literals. They are considered false if a banana remains on the path.

3.3 Clause Gadget

The clause gadget, seen in Figure 4 splits the road into three equal-length paths, one for each literal, that later merge. Each path has an initially placed and unavoidable banana. Thus, if any of the bananas has been destroyed by a green shell, the player can choose that path and traverse the gadget quickly. Otherwise, the player must hit a banana and incur a speed and time penalty—assuming that the player is not carrying any green shells.

3.4 Clearing Held Items

To guarantee that the player traverses the sequence of clause gadgets without any green shells, we add a *clearing gadget* between the sequence of variable gadgets and the sequence of clause gadgets. The clearing gadget, shown in Figure 5, forces the driver to afterward hold no items (behind the car or otherwise).

There are actually two different gadgets, depending on whether the Mario Kart game permits carrying a second item behind the car. For games where this is impossible (currently just Mario Kart 8), the gadget consists of a single green shell item box followed by an already placed banana. Otherwise, we have two green shell item boxes followed by two already placed bananas. The distance between the item boxes and bananas is longer than the lifetime of a shell. Thus, to avoid slowdown from the bananas, the player must use all storable green shells (either just picked up or stored from before) and be left holding nothing.

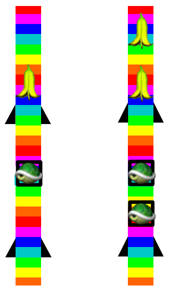


Fig. 5. Two types of Clear Gadgets.

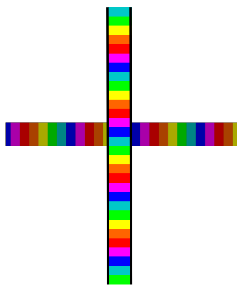


Fig. 6. A Crossover Gadget. The vertical path is placed higher in the level with a wall along the track.

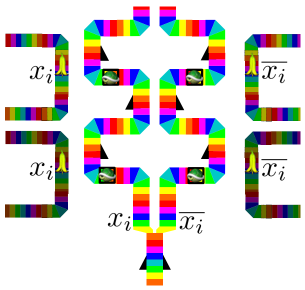


Fig. 7. Variable gadget being able to unlock clause. Once Player 1 assigns x_i, it can shoot shells to unlock clauses where x_i appears.

3.5 Crossover Gadget

Crossover gadgets are relatively simple given the three-dimensional nature of Rainbow Road levels, so one road can pass over another road; see Figure 6. To ensure that the player does not jump from the upper road to the lower road, and that the player does not throw a shell from the upper road to the lower road, we surround the sides of the upper road with vertical walls, for sufficient length before and after the intersection.

3.6 Putting Gadgets Together

Figure 7 shows how a literal road of a variable gadget interacts with each clause gadget containing the literal. By bringing the variable road somewhat close and above the clause road, the player can shoot the green shell from the variable and destroy the banana in the clause, without slowing down. This action "unlocks" the clause gadget for later traversal, corresponding to satisfying the clause.

However, we cannot place the roads too close to each other, or else the player could jump from the variable road to the lower clause road. Fortunately, there is a suitable distance traversable by shells but not by players, because shells move faster than players. (Alternatively, even if players could move as fast as shells, this property could be arranged by having the shell bounce off of a floating vertical wall, which the player could not do.)

Finally we describe how to lay out the gadgets. Because there is a constant maximum speed that can be attained on a flat track, there a constant size of gadget with straight tracks as inputs/outputs that guarantees two properties: (1) the player cannot traverse from a gadget to a gadget not logically connected to it, and (2) the player normalizes to a standard maximum straight-away speed before entering the next gadget. We use this constant gadget size as our unit size. The literals, crossovers, and their connecting lines can be laid out orthogonally on an $O(n+m) \times O(n+m)$ unit square grid in polynomial time [BK94]. We may then need to tweak some of the path distances to have the same optimal traversal times. If we scale up the grid by a factor of $c(n+m)$, then we can "wiggle" each track segment on the grid to have length between $c(n+m)$ and $c^2(n+m)^2$, which suffices to unify paths of length between 1 and $O(n+m)$ on the original grid. It is important that we are able to make separate tracks take close to the same traversal time because the reduction separates the winning kart by the constant amount of time lost by hitting a banana. Because we choose different routes for each clause and variable, we need to be able to match track lengths with an accuracy of $1/(n+m)^{O(1)}$ with only a $(n+m)^{O(1)}$ blowup in size and using a polynomial amount of computation time. This is covered by our model assumptions in Section 2. Thus we can lay out the gadgets in a polynomial-time reduction.

4 Racing is PSPACE-Hard

We now study the following two-player variant of Mario Kart, where players race against each other:

Theorem 2. *It is* PSPACE*-hard to decide whether Player 1 has a forced win in a two-player Mario Kart race from given starting positions for the players.*

4.1 Proof Structure

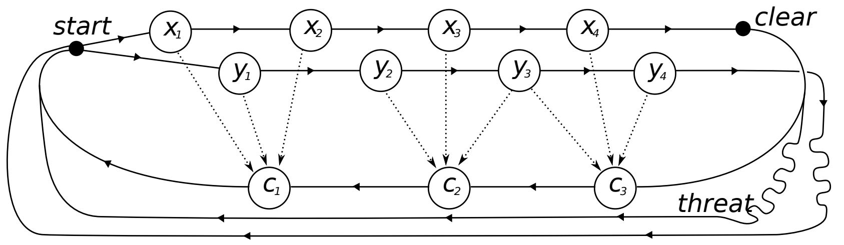


Fig. 8. General reduction structure for 2 players. Dashed lines correspond to reachability of green shells and bananas.

The reduction is from Q3SAT: decide a quantified Boolean formula $\phi = \exists x_1 : \forall y_1 : \exists x_2 : \forall y_2 : \cdots \exists x_{n/2} : \forall y_{n/2} : \phi'(x_1, \ldots, x_{n/2}, y_1, \ldots, y_{n/2})$ where ϕ' is in 3CNF, has a satisfying assignment. We construct the track similar to the NP-hardness proof, but with Player 1 setting the existentially quantified variables and Player 2 setting the universally quantified variables; refer to Figure 8. As in the proof for NP-hardness, Player 1 will shoot shells from an elevated road to clear bananas from clause gadgets. Player 2, who sets the universal quantified variables is on a separate elevated road throwing bananas into clause gadgets. While each player sets a variable, the other player is forced along a higher road of the same traversal time, within visual range so that both players know the variable setting; see Figure 10. This way, we get the alternating behavior and perfect information while setting variables. The overall path Player 1 takes is slightly shorter than Player 2. So if Player 1 can get through the clauses without hitting any bananas, s/he will win. If Player 1 runs over any bananas and slips, Player 2 will win.

Player 2 can "cheat" in a variety of ways, but all of them consume time. For these cases, Player 1 has an alternative winning path that bypasses all clauses, but takes longer than if Player 2 plays "straight". This threat prevents Player 2 from cheating (in optimal play).

4.2 Clause Gadget

As shown in Figure 11, the clause gadget is a road that splits into one road per literal, as in the NP-hardness proof. The literals of existentially quantified variables are initially blocked by a banana, as in the NP-hardness proof, while literals of universally quantified variables are initially empty.

4.3 Variable Gadget

Player 1's (existential) variable gadgets are the same as in the NP-hardness proof (Figure 3): each gadget forks to make the player choose between setting x_i or $\overline{x_i}$ to true, with each fork passing by all the clauses containing that literal, so the player can shoot a shell down to remove the banana from that existential variable's literal instance.

Player 2's (universal) variable gadgets have the same structure, but as shown in Figure 9, the player instead sets y_i or $\overline{y_i}$ to *false* by shooting bananas (picked up from item boxes in the variable) down into literal instances in the clause gadgets, filling what was initially empty.

4.4 Putting Gadgets Together

Existential variable gadgets and clause gadgets interact as in the NP-hardness proof. Universal variable gadgets interact with clause gadgets at a closer distance, given the lobbed trajectory of bananas. To prevent Player 2 from jumping down to the clause gadget in this situation, we can use a vertical wall or rail that is tall enough to block the player but not tall enough to block a thrown banana.

We use the same crossover gadgets as the NP-hardness proof (Figure 6), and the same clearing gadget (Figure 5) before Player 1 enters the sequence of clause gadgets. Everywhere else, whenever a player would be helped by an item, that item is presented by an item box, so it never helps to hold onto an item for later. (Note that it does not help to block

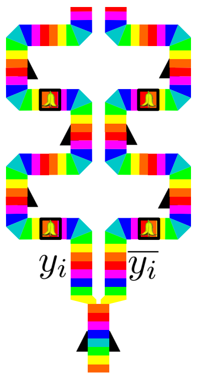


Fig. 9. Variable gadget for Player 2. Player 2 assigns y_i and grabs bananas to throw to the clause gadgets.

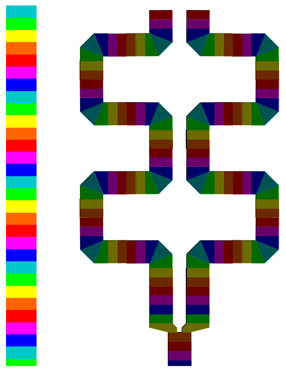


Fig. 10. Observation of other player. The variable gadget (grayed out) appears below in 3-dimensional space.

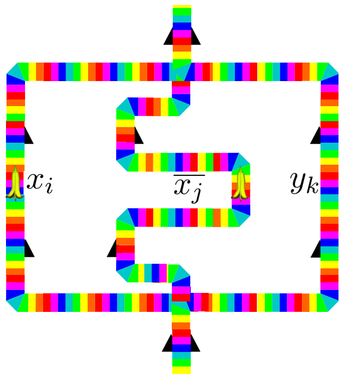


Fig. 11. Clause gadget split into literals. A clause splits into the three literals which comprise the clause. Note that since y_k is a variable set by Player 2, there is no banana on the path until Player 2 throws a banana down.

a literal with two bananas instead of just one. A single banana penalty is enough for Player 2 to win.)

After all variables have been set, Player 1 drives through the clause gadgets while Player 2 drives along a winding road slightly longer to traverse than the road through the clause gadgets. If all clauses are satisfied (have at least one literal branch without a banana), Player 1 wins; otherwise, Player 1 must drive through at least one banana and slow down. In this case, Player 2 wins, by setting the "slightly longer" amount to strictly less than the banana penalty. (For a more comfortable construction, we can repeat every clause k times, allowing the difference to be strictly less than k times the banana penalty.)

Player 2 can attempt to "cheat" in a couple of ways: traversing both sides of a universal variable gadget, or waiting to choose the value of a universal variable gadget until after Player 1 chooses the next variable (breaking the quantifier structure). In this case, Player 2 will fall behind relative to the intended traversal. This would be worthwhile if Player 2 could slow down Player 1 substantially as a result, but the availability of the slightly longer threat path means that Player 1 can avoid all clauses and thus all slowdowns in this case. Player 1 also cannot afford to cheat in these ways, because s/he starts with only a small advantage, and is unable to slow down Player 2.

Gadget layout can be done analogous to Section 3.

5 Conclusion

In practice, players in Mario Kart generally make forward progress on the track, other than short aberrations caused by attacks, and have knowledge (via the minimap) of the state of all players. These assumptions imply a polynomial bound on the length of solutions, which in turn implies that our results are tight: time trial is NP-complete and racing is PSPACE-complete. Without the game-length assumption, however, we only know containment in PSPACE and EXPTIME, respectively, and it is plausible that we could establish corresponding hardness. With hidden information (unknown state of the track or items held by opponents), Mario Kart racing is potentially as hard as 2EXPTIME.

References

ADGV15. Greg Aloupis, Erik D. Demaine, Alan Guo, and Giovanni Viglietta. Classic Nintendo games are (computationally) hard. *Theoretical Computer Science*, 586:135–160, 2015.

BK94. Therese Biedl and Goos Kant. A better heuristic for orthogonal graph drawings. In *AlgorithmsESA'94*, pages 24–35. Springer, 1994.

For10. Michal Forisek. Computational complexity of two-dimensional platform games. In *Proceedings of the 5th International Conference on Fun with Algorithms*, pages 214–227, 2010.

Gui14. Guinness World Records. Best-selling racing videogame. http://www.guinnessworldrecords.com/world-records/best-selling-racing-video-game/, 2014.

Sal. Sales figures based on http://www.polygon.com/2014/5/15/5718168/mario-kart-series-sales, http://www.nintendo.co.jp/ir/en/sales/software/3ds.html, and http://www.nintendo.co.jp/ir/en/sales/software/wiiu.html, as of November 2015.

Vig14. Giovanni Viglietta. Gaming is a hard job, but someone has to do it! *Theory of Computing Systems*, 54(4):595–621, 2014.