Dr. Mario Puzzle Generation: Theory, Practice, & History (Famicom/NES)

Conference Paper · July 2019

CITATIONS
0

READS
349

1 author:

Aaron Williams
Williams College
68 PUBLICATIONS **520** CITATIONS

SEE PROFILE

Dr. Mario Puzzle Generation: Theory, Practice, & History (Famicom/NES)

Aaron Williams*

1 Introduction

Dr. Mario (ドクターマリオ) was released by Nintendo on the Famicom/NES (Japan/USA) in 1990. It influenced countless matching games (see Juul [3]) and was one of the first games with randomly generated puzzles (see Parish [6]).
As **mathematicians** we ask the following question.

- *How are Dr. Mario puzzles defined combinatorially?*

As **computer scientists** we ask the following question.

- *How can puzzles be generated algorithmically?*

As **retroarcheologists** (Aycock [1]) we ask the following.

- *How did Nintendo's programmers generate puzzles?*

As **video game historians** we ask the following question.

- *Why do all NES puzzles have $v \leq 84$ viruses?*

Finally, as **retrogamers** we ask the following question.

- *Can we make the NES game more difficult (and fun)?*

This extended abstract briefly answers these questions. We assume the reader is familiar with the NES game.

2 Graph Coloring

A *Dr. Mario puzzle* fits v viruses in the bottom r rows of a grid. All commercial releases have $c = 8$ columns and we assume this unless otherwise stated. Each virus is red, yellow, or blue and at most one is allowed per cell. For example, Figure 1 has several puzzles with viruses in the bottom $r = 13$ rows.

Remark 1. The hardest NES puzzles have difficulty level 20. These puzzles fit $v = 84$ viruses in the bottom $r = 13$ rows.

Nintendo never puts three consecutive same color viruses horizontally or vertically. In fact, they use a stronger constraint: Same color viruses are never two cells away horizontally or vertically. Thus, a virus in cell (a, b) forbids $(a+2, b)$, $(a-2, b)$, $(a, b+2)$, and $(a, b-2)$ from having that color virus.

We refer to the above constraints as *three-in-a-row* and *two-away*, respectively. A puzzle is *valid* if it satisfies the two-away (and hence the three-in-a-row) constraint.

For graph theorists a valid puzzle is simply a partially 3-colored r-by-c grid graph, except that edges connect vertices two rows/columns away. This graph has four connected components; each component is a standard grid graph containing vertices with the same pair of row and column parities (or *pairities*). Figure 2 illustrates the graph associated with the puzzle in Figure 1a and its component grid graphs.

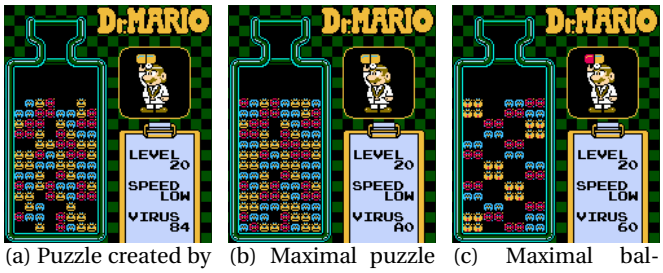
Remark 2. A Dr. Mario puzzle is valid if and only if its connected grid graphs inherit proper partial 3-colorings.

2.1 Maximal and Balanced Puzzles

A color is *available* for an empty cell unless adding that color of virus breaks the two-away constraint. A puzzle is *maximal* if no cells have available colors, i.e. its grid graphs are maximally 3-colored. Figure 1a isn't maximal. Figures 1b–1c are both maximal but have wildly different virus counts.

A Dr. Mario puzzle is *balanced* if its three colors have (at most) two consecutive frequencies. Figure 1c is balanced since each color is used $\frac{60}{3} = 20$ times.

*Williams College, aaron.williams@williams.edu



(a) Puzzle created by random seed 1394. It has $v = 84$ viruses.
(b) Maximal puzzle from seed 1394. It has $v = 100$ viruses.
(c) Maximal balanced puzzle. It has only $v = 60$ viruses.

Figure 1: Puzzles from (a) the original NES game, and (b) the NES game using Game Genie code `IANETZPA` to add viruses. Part (c) gives a hypothetical puzzle. Each puzzle fits v viruses in the bottom $r = 13$ rows of the playfield with $c = 8$ columns.

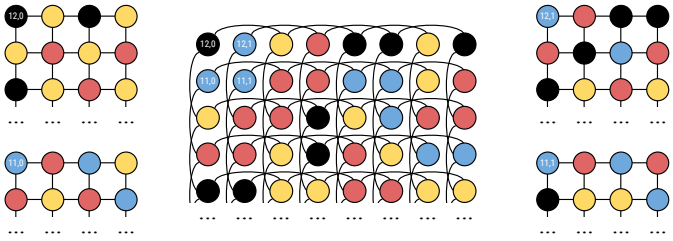


Figure 2: Figure 1a as graph coloring. Black denotes no color (i.e. no virus). The top-left grid graph has (even,even) pairity.

3 Algorithms

Now we design simple algorithms for generating random valid Dr. Mario puzzles with v viruses. We focus on algorithms that add one virus at a time and never backtrack.

Our first algorithm uses random positions and colors.

Algorithm 1. Choose a random cell and a random color until that color is available at that cell.

Our second algorithm also uses random positions, but it cycles through the colors to ensure the result is balanced.

Algorithm 2. Cycle through colors red, yellow, blue. Choose a random cell until that color is available.

Sadly, Algorithms 1–2 do not safely generate NES puzzles (see Remark 1). Specifically, Figure 1c proves Remark 3.

Remark 3. Algorithms 1 & 2 can fail with 60 viruses in 13 rows.

Next we consider randomizing only the colors.

Algorithm 3. Visit the cells in row-major order. Choose a random available color or no color for the cell; reweight this probability[1] to ensure v are added in total.

Notice that Algorithm 3 always works. This is because each cell has ≤ 2 prior neighbors in row-major order. Thus, every cell has at least one available color when it is visited.

Remark 4. Algorithms 3 always fits v viruses in an r-by-c grid, so long as the necessary condition $v \leq r \cdot c$ holds.

[1] If there are x remaining viruses and y remaining cells, then the probability of choosing to color a cell is $\frac{x}{y}$.

4 NES Game

Next we consider how Nintendo programmer Takahiro Harada generated random puzzles in the NES game. The pioneering work of user `nightmareci` in disassembling the NES machine code was invaluable in this investigation (see [4, 5]).

4.1 Harada's Algorithm

Harada's approach is similar to Algorithm 2 in that it chooses random positions and attempts to cycle through the colors. However, when it fails with its preferred choice, it proceeds in a manner that is similar to Algorithm 3. For further details including pseudocode see [5].

Algorithm H. Choose a random cell. Add the highest-priority available color to this cell. If no colors are available, then repeat on the next cell in row-major order.

In theory, Algorithm H can generate any balanced puzzle, including Figure 1c. Therefore, Nintendo's algorithm is also flawed—it does not safely generate NES puzzles!

Remark 5. Algorithms H can fail with 60 viruses in 13 rows.

If the NES game did generate Figure 1c, then it would *freeze* (i.e. infinite loop). Luckily, this never occurs in practice due to the limited form of randomization used in the game.

4.2 Nintendo's LFSR and Randomization

For randomization Harada used a 15-bit *linear feedback shift register* (see Golomb [2]) with primitive feedback polynomial $x^{15}+x^7+1$. Thus, bits 7 and 15 are tapped. So if the current state is $b_1 b_2 \cdots b_{15}$, then $x = (b_7 + b_{15}) \mod 2$ is the next random bit, and the next state is $x b_1 b_2 \cdots b_{14}$ (see Figure 3).

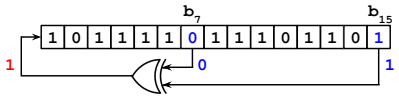


Figure 3: Nintendo's LFSR. For example, state 101111011101101 is followed by state 1101111011101 10.

Before generating a puzzle the NES game seeds the LFSR state to any non-zero value $b_1 b_2 \cdots b_{15}$. This gives an entry point into a cyclic stream of $2^{15}-1 = 32{,}767$ pseudorandom bits. Algorithm H then proceeds deterministically.

Remark 6. Dr. Mario (NES) has at most 32,767 distinct puzzles per r-by-c grid. (Figure 1c isn't one of them.)

Note that the number of NES puzzles pales in comparison to the number of valid puzzles. For example, there are more than $\binom{104}{84} > 10^{21}$ puzzles with $r = 13$ rows and $v = 84$ viruses.

The use of a 15-bit LFSR instead of a 16-bit LFSR was likely due to efficiency. Maximal length 16-bit LFSRs require four taps [7] and hence more instructions. Due to the 8-bit architecture of the NES, the state is stored across two bytes with b_{16} ignored. Thus, state `0x5EED` and `0x5EEC` are equivalent.

4.3 Analyzing Algorithm H

A translation of Algorithm H (and its LFSR) into C is provided by `nightmareci` in [4]. By modifying this program, we can run Algorithm H until it generates a maximal puzzle, instead of quitting after the required number of viruses. The *failure point* (i.e. the virus number not added) for all 32,767 puzzles with 13 rows appears in Figure 4. For example, Figure 1b is one of the 5,247 different levels with a failure point of 101.

Notice that Algorithm H never fails before the 85th virus. Thus, it never freezes the actual NES game.

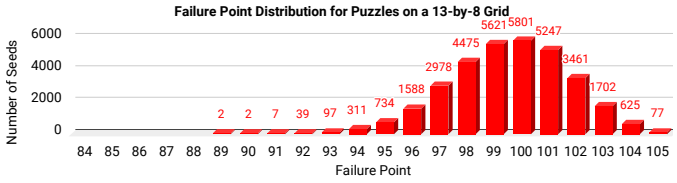


Figure 4: Distribution of Algorithm H failure points for level 20 puzzles using the Game Genie code `YANETZPA`.

5 Historical Signficance

Newer Dr. Mario titles have more viruses in the hardest puzzles. For example, *Dr. Mario Online Rx* (Wii) can fit $v = 99$ viruses in $r = 13$ rows. Our study explains why the NES game only has $v = 84$ in these puzzles. Simply put, Algorithm H was not robust enough; it can fail on the 89th viruses.

Prototypes of Dr. Mario exist under the name *Virus* [8] and have up to $v = 96$ viruses (but ignore the three-in-a-row constraint). Thus, Nintendo likley wanted more viruses in the NES game, but were limited by Algorithm H. Furthermore, the NES parameters were likely tuned to avoid game freezes.

Future work should target the Game Boy and SNES, which have a 15-by-8 playfield and up to $v = 96$ viruses, respectively.

6 Improving the NES Game with Game Genie Codes

In Section 4.3 we modified a simulation of the NES puzzle generation algorithm. Now we modify the real game in order to create more difficult (and fun) puzzles that can be played.

The *Game Genie* is an NES peripheral that modifies memory values to alter gameplay. Table 1 has new codes for the number of viruses in all levels, and the rows in level 20.

Code	All Levels
AANETZPA	−4 viruses
ZANETZPA	+4 viruses
LANETZPA	+8 viruses
GANETZPA	+12 viruses
IANETZPA	+16 viruses
TANETZPA	+20 viruses
YANETZPA	+24 viruses
AANETZPE	+28 viruses
PANETZPE	+32 viruses

Code	Level 20
PENZZUGE	10 rows
ZENZZUGE	11 rows
LENZZUGE	12 rows
IENZZUGE	14 rows
TENZZUGE	15 rows
YENZZUGE	16 rows

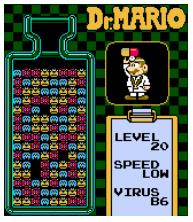


Table 1: New Game Genie codes for Dr. Mario (NES) and an impossibly hard puzzle. It has $v = B \cdot 10 + 6 \cdot 1 = 116$ viruses.

Some codes can cause freezing during puzzle generation. For example, `TANETZPA` completely fills level 20 puzzles with viruses and its success rate is only $\frac{77}{32767} = 0.23\%$ by Figure 4.

References

[1] J. Aycock. *Retrogame Archeology: Exploring Old Computer Games*. Springer, 1st edition, 5 2016.

[2] S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, 1981.

[3] J. Juul. Swap adjacent gems to make sets of three: A history of matching tile games. *Artifact*, 4(1):205–216, 2007.

[4] nightmareci. Dr. Mario virus placement. `https://tetrisconcept.net/threads/dr-mario-virus-placement.2037`, August 2012.

[5] nightmareci. Dr. Mario. `https://tetris.wiki/Dr._Mario`, March 2013.

[6] J. Parish. Dr. Mario retrospective: Heading for a malpractice suit | Game Boy Works 070. `https://www.youtube.com/watch?v=kGq1n9F1DFM`, July 2016.

[7] A. Partow. Primitive polynomial list. `https://www.partow.net/programming/polynomials/index.html`.

[8] Proto:Dr. Mario (NES). `https://tcrf.net/Proto:Dr._Mario_(NES)`.