Explaining the Entombed Algorithm

Leon Mächler
ENS, CNRS, PSL Research University
Département d'informatique, École normale supérieure
Paris, France
leon-philipp.machler@ens.fr

David Naccache
ENS, CNRS, PSL Research University
Département d'informatique, École normale supérieure
Paris, France
david.naccache@ens.fr

***Abstract*—In [1], John Aycock and Tara Copplestone pose an open question, namely the explanation of the mysterious lookup table used in the Entombed Game's Algorithm for two dimensional maze generation. The question attracted media attention (BBC etc.) and was open until today. This paper answers this question, explains the algorithm and extends it to three dimensions.**

***Index Terms*—Entombed Algorithm, Maze Generation, Invariants**

I. Introduction

When Aycock and Copplestone reverse engineered the code of the 1982 Atari Game "Entombed" they found an inexplicable algorithm that consistently creates novel playable two dimensional mazes [1]. The algorithm is interesting due to the fact that it's decisions are only based on local information which makes it fast and efficient. The local information is the 5 bit context of blocks surrounding the current position. For for each of the 32 possible scenari, a decision is predefined in a lookup table. It was unclear to Aycock and Copplestone how this lookup table was created, which they posed as an open question.

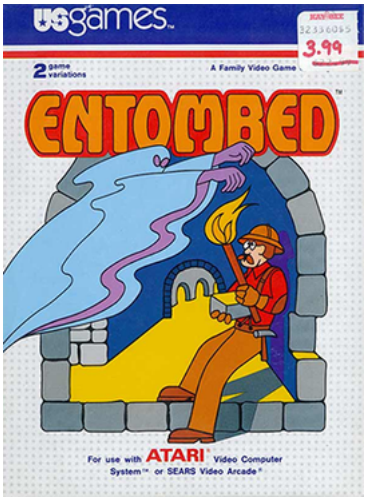

Figure 1. The game's coverart

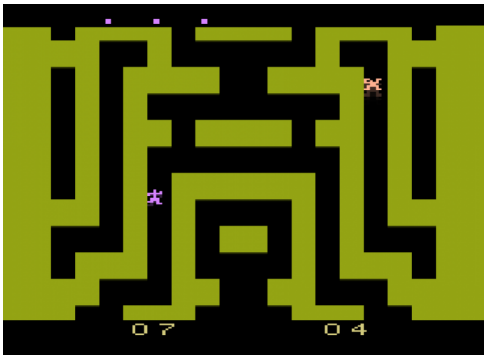

Figure 2. A typical maze generated by the algorithm

As of July 2021, the question was on Wikipedia's "List of unsolved problems in computer science"[1]:

> "*What is the algorithm for the lookup table that consistently generates playable mazes in the 1982 Atari 2600 game Entombed merely from the values of the five pixels adjacent to the next ones to be generated?*"

Whereas the Entombed Wikipedia page[2] states:

> "*The mechanics of how Entombed generated its mazes have been the subject of academic research and some legend, as the maze data itself, if stored directly, was too large to fit within the hardware limitations of the console, even with the left/right symmetry of the mazes. Researchers evaluated the game's ROM and discovered that the mazes were generated on-the-fly by the game using the state of five adjacent squares of the maze (wall or open) already generated to determine the next part of the maze through a lookup table, including potentially a random state. Sometimes the table generates mazes that would be unsolvable without the "make-break" item. The researchers spoke to Sidley, who said the algorithm came from another unnamed programmer, but Sidley himself could not decipher why it worked. Sidley said to the researchers of this programmer, "He told me it came upon him when he was drunk and whacked out of his brain." Later research, however,*

[1] en.wikipedia.org/wiki/List_of_unsolved_problems_in_computer_science
[2] en.wikipedia.org/wiki/Entombed_(Atari_2600)

978-1-6654-3886-5/21/$31.00

suggests that although the original programmer and his mathematical collaborator did devise the algorithm at a bar, they may have suggested or encouraged the drunken blackout story merely to avoid having to explain or assign intellectual property rights for the algorithm."

The question has attracted media attention[3][4][5][6][7] and resulted in a dedicated Reddit thread[8]. This paper shows that all of the choices encoded in the lookup table are consistent with maintaining three simple invariants. When neither choice would violate an invariant a random choice is made. We then extend the method to three dimensions.

II. The Algorithm

The goal of the algorithm is to create random fixed-width (possibly infinite length) bi-dimensional mazes. This is achieved by successively creating line after line in the maze where each line is built block by block. A block is represented by one bit that can be interpreted as either a wall (1) or a path (0). The choice for each block is based solely on the immediately surrounding five blocks (two to the left, three above). For all 32 possible neighboring block combinations, the decision is predefined in a lookup table and can either be $1, 0$ or random. When the neighboring blocks fall outside of the maze, either a random value or a predefined value is chosen instead. The decisions' hyper-locality allows for very cheap computational cost while still producing a vast plurality of possible output mazes. When this algorithm was discovered by Aycock and Copplestone in the code of an old Atari game called "Entombed", they could not explain how the lookup table was found or why it works so well. So they left its clarification as an open question.

Algorithm 1 Entombed Maze Algorithm

Input: The lookup table L, maze dimensions X and Y
Output: A binary maze M, where $M_{i,j} = 1$ iff a wall is at coordinate i, j.

```
for i, j ≤ X, Y do
    a ← M_{i,j-2}
    b ← M_{i,j-1}
    c ← M_{i-1,j-1}
    d ← M_{i-1,j}
    e ← M_{i-1,j+1}
    M_{i,j} ← L(a, b, c, d, e)
return M
```

[3]www.bbc.com/future/article/20190919-the-maze-puzzle-hidden-within-an-early-video-game

[4]www.wnycstudios.org/podcasts/tnyradiohour/segments/unearthing-entombed

[5]medium.com/codex/random-maze-from-entombed-8bb3b34e8f9b

[6]hackaday.com/2019/09/30/emtombed-secrets-partially-unearthed-as-researchers-dissect-clever-maze-generating-algorithm/

[7]www.techspot.com/news/85622-nobody-sure-what-makes-atari-2600-game-entombed.html

[8]www.reddit.com/r/math/comments/d8bgbu/a_mysterious_maze_algorithm

Note that in the game the maze was then also mirrored (as can be seen in figure 2) to further minimize computational cost. For the explanation of the lookup table this is of no importance.

III. The Lookup Table

The lookup table L can be explained by three invariants. No choice will ever violate one of the invariants and when for a given context neither choice would violate an invariant, a random choice is made. Note however that due to the fact that the context variables might fall outside the maze, depending on the auxiliary values that are used to replace them, effectively some invariants may be violated. The invariants are:

(1) No 2×2 squares of the same type are allowed
(2) No wall or path is allowed to start or end with thickness one
(3) Every path in any given line must be connected to a path in the next line

In invariant (2) "to start or end" is meant in a vertical sense i.e. top-down. Invariant (3) is enforced by ensuring that every 0 (path) in a line must either have a 0 in the line directly below it or a 0 to it's right. Since the lines are being built left to right at some point a continuous sequence of 0s would hit the wall on the right and then be connected to the line below. Here, it is important to remember that due to the fact that some of the context variables might fall outside of the maze it is possible that effectively the invariant is violated on either wall, depending on the auxiliary choice of values for a, b and e. In the algorithm's Atari implementation the auxiliary choice for a and b is $0, 1$ and for c and e it is random. This is why, from time to time, there are non-connected paths in the game. A choice of $a = b = e = 1$ would enforce full connectivity since then the context would always match the lookup table. In the mazes that were used for the game, connectivity of all paths was not desirable thus it was not enforced. In other words: the lookup table always enforces the invariants; it might just be the case that the context in the lookup table and the real maze context differ.

To see how invariants (1) (2) (3) explain the lookup table it suffices to express them in terms of the variables a, b, c, d, e. We can formulate all the rules stemming from the invariants as follows (The notation $010 \star\star \rightarrow 1$ would mean "no matter what d and e are, when $a = 0 \wedge b = 1 \wedge c = 0$ the choice $x = 1$ is made"):

- Invariant (1) gives:
  $\star\ 0\ 0\ 0\ \star \rightarrow 1$
  $\star\ 1\ 1\ 1\ \star \rightarrow 0$
- Invariant (2) gives:
  $\star\ \star\ 0\ 1\ 0 \rightarrow 1$
  $\star\ \star\ 1\ 0\ 1 \rightarrow 0$
  $0\ 1\ 0\ \star\ \star \rightarrow 1$
  $1\ 0\ 1\ \star\ \star \rightarrow 0$
- Invariant (3) gives:
  $\star\ 1\ 0\ 0\ 1 \rightarrow 0$
  $\star\ \star\ 1\ 0\ 1 \rightarrow 0$ (already a rule)

Note that to enforce invariant (3), it is sufficient to enforce (3) only for the paths that fall into variable d since every 0 in the row above will always fall into the role of variable d once.

Our claim that all entries in the lookup table can be explained by the invariants would predict that all remaining entries are random. This is almost the case. We still have one rule that seemingly cannot be explained, namely $00100 \to 0$. That remaining entry is explained by the remarkable fact that the invariants almost never contradict themselves. Only in one case they run into a contradiction, namely in the context 01001 where invariant (2) demands a 1 while invariant (3) demands a 0. The solution is elegant: the choice 1 is made and the rule $\star 0100 \to 0$ is added (which explains the remaining entry since $10100 \to 0$ already is a rule). This solves the problem that the path at variable d would be blocked (and thus violate invariant (3)) if the block to the left of variable c was a wall. The added rule enforces that such a situation will never arise (because for this to happen in the step before the context would have been $\star 0100$ and a 1 would have had been placed). It will ensure that whenever a context like 01001 is met, putting down a wall will not disconnect the path since the 0 at variable d will always be connected to the 0 at a.

Now we can also see how the predefined values for variables a, b, e play a role when they fall outside the maze. In the game's implementation the choice $a = 0, b = 1$ was made. This can lead to situations where a path is only being connected to the imaginary 0 that lies outside of the maze. Thus effectively a dead-end at the wall is created. A choice $a = b = e = 1$ will lead to no such dead-ends.

IV. The Lookup Table for 3 Dimensions

To extend the method to 3 Dimensions we decided to maintain the same invariants. A new context had to be chosen to accommodate for the third dimension. Note that more variables will lead to more rows in the lookup table that need to be filled. We propose a 10-variable context leading to an increase from 32 to 1024 possible decisions. It is of the following form (x is the new block that will be placed):

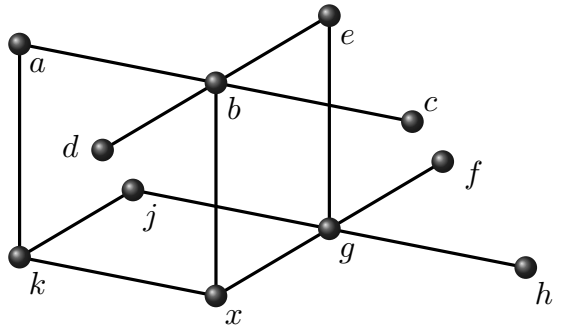

Figure 3. 3 Dimensional Maze Generation Context

The following rules are applied:

- Invariant (1) gives:

0	0	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	0	$\to$	1
1	1	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	1	$\to$	0
$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	0	$\star$	0	0	$\to$	1
$\star$	$\star$	$\star$	$\star$	$\star$	$\star$	1	$\star$	1	1	$\to$	0
$\star$	0	$\star$	$\star$	0	$\star$	0	$\star$	$\star$	$\star$	$\to$	1
$\star$	1	$\star$	$\star$	1	$\star$	1	$\star$	$\star$	$\star$	$\to$	0

- Invariant (2) gives:

0	1	0	0	0	$\star$	$\star$	$\star$	$\star$	$\star$	$\to$	1
1	0	1	1	1	$\star$	$\star$	$\star$	$\star$	$\star$	$\to$	0
$\star$	$\star$	$\star$	$\star$	0	0	1	0	0	$\star$	$\to$	1
$\star$	$\star$	$\star$	$\star$	1	1	0	1	1	$\star$	$\to$	0

- Invariant (3) gives:

0	0	1	1	1	$\star$	$\star$	$\star$	$\star$	1	$\to$	0
1	0	1	1	0	$\star$	1	$\star$	$\star$	$\star$	$\to$	0
0	0	1	1	0	$\star$	1	$\star$	$\star$	1	$\to$	0

Just as with the bi-dimensional invariants, conflicting situations can occur. I.e. in case of a context of the form $11\star\star001001$ invariant (1) demands a 0 since otherwise a 2×2 square of walls is formed along the variables a, b, k and the new block. But at the same time invariant (2) demands a 1 since otherwise the 1 at variable g will form the start of a wall with thickness one. Just as it was done for the 2D lookup table, new rules are added to prevent these contexts from appearing. Thus the following new rules are added:

- To prevent conflicts 1 1 $\star$ $\star$ 0 0 1 0 0 1 : $\star$ 1 1 $\star$ $\star$ $\star$ 0 1 $\star$ $\star$ $\to$ 0
- To prevent conflicts 0 0 $\star$ $\star$ 1 1 0 1 1 0 : $\star$ 0 0 $\star$ $\star$ $\star$ 1 0 $\star$ $\star$ $\to$ 1
- To prevent conflicts 0 1 0 0 0 $\star$ 1 $\star$ 1 1 : $\star$ 0 1 $\star$ $\star$ $\star$ 1 1 $\star$ $\star$ $\to$ 0
- To prevent conflicts 1 0 1 1 1 $\star$ 0 $\star$ 0 0 : $\star$ 1 0 $\star$ $\star$ $\star$ 0 0 $\star$ $\star$ $\to$ 1
- To prevent conflicts $\star$ 0 1 1 0 0 1 0 0 $\star$: $\star$ $\star$ 0 1 1 $\star$ 0 1 $\star$ $\star$ $\to$ 0

Naturally more dimensions lead to a bigger context which, in turn, calls for more rules. Therefore we also see more conflicts between the rules. In the bi-dimensional case there was one conflict in 32 possible contexts (roughly 3%) while in the three dimensional case we have 28 conflicts in 1024 possible contexts (also roughly 3%). Although in the higher dimensional case we have the added complexity that the conflict preventing rules themselves run into conflicts. Thus only 75% of the conflicts could be prevented which leads to roughly 0.7% of contexts in which a choice will violate an invariant. This number could be further minimized by increasing the size of the context or the maze generation could be re-run with a different random seed.

References

[1] Aycock, J., and Copplestone, T. Entombed: An archaeological examination of an atari 2600 game. CoRR abs/1811.02035(2018).

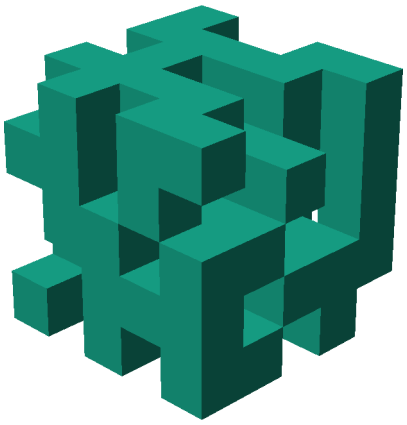

Figure 4. $5 \times 5 \times 5$ example maze

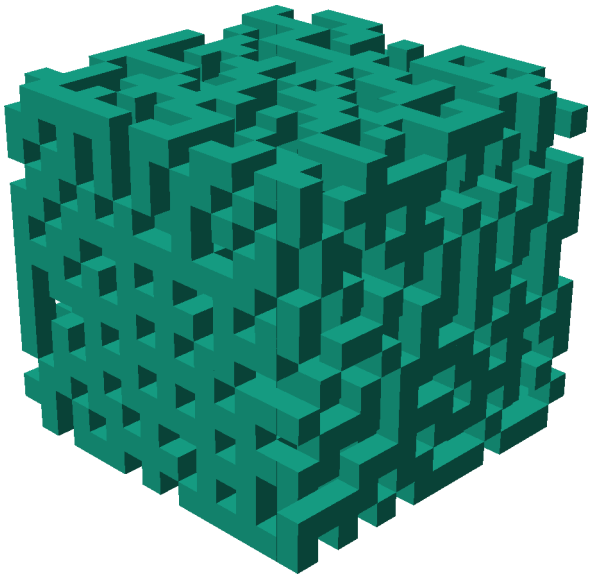

Figure 5. $15 \times 15 \times 15$ example maze

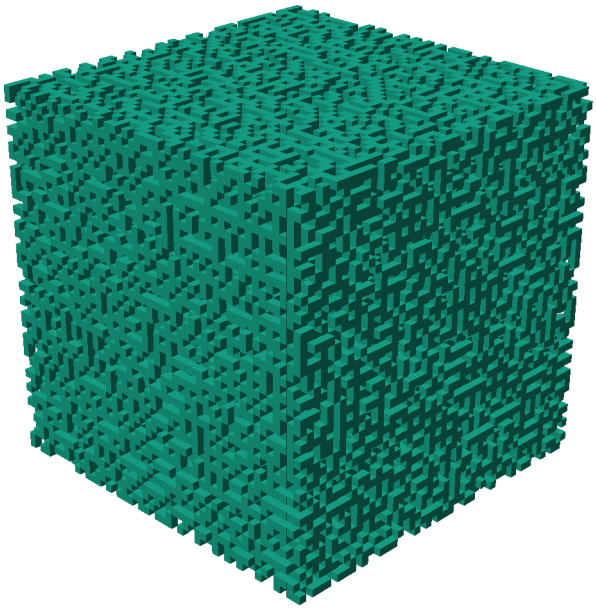

Figure 6. $50 \times 50 \times 50$ example maze