Symmetry: Culture and Science
Vol. 31, No. 1, 77-90, 2020
https://doi.org/10.26830/symmetry_2020_1_077

SEARCH FOR A COMPUTER AIDED SOLUTION OF THE 7×7 SQUARE POLY-UNIVERSE: AN ALGORITHMIC APPROACH

András Triznya[1], Vilmos Katona[2*]

[1] Independent Researcher, 9 Thaly Kálmán utca, Budapest 1165, Hungary.
E-mail: triznya.andras@gmail.com
[2] Institute of Applied Arts, Simonyi Károly Faculty of Engineering, Wood Sciences and Applied Arts University of Sopron, 1 Cházár András tér, Sopron 9400, Hungary.
E-mail: katona.vilmos@uni-sopron.hu
ORCID: 0000-0002-0299-2897
[*]corresponding author

Abstract: *Saxon's Poly-Universe was created to show that a logic-stimulating mathematical game could derive from an artistic concept. It combines colours with basic geometrical shapes on two-dimensional plane to set up three different board games (square, triangle and circle). In these games, players can customise the rules, the difficulty level of which increases with the compactness of the form, and the homogeneousness of the pattern preferred. We have chosen the square pack to test software algorithms capable of solving the most difficult player requirements possible. The purpose was also to optimise the computing time with the hardware capacity.*

Keywords: mathematics and art, science education, gamification, C# programming, combinatorial optimisation, test driven development, Poly-Universe, MADI art, János Saxon Szász

1 OUR HISTORY

About a decade ago we created some algorithms either to solve some non-trivial problems, or for visual entertainment purposes.

There was a recurring theme about these—computing capacity is limited, be it memory consumption or processing performance.

Problems with an intuitive human solution are often nigh impossible for a computer. Thus emerges the need for not just coding optimisation, but a logical evaluation of the solving algorithm to try and reduce its complexity (cf. Palócz and Katona, 2019).

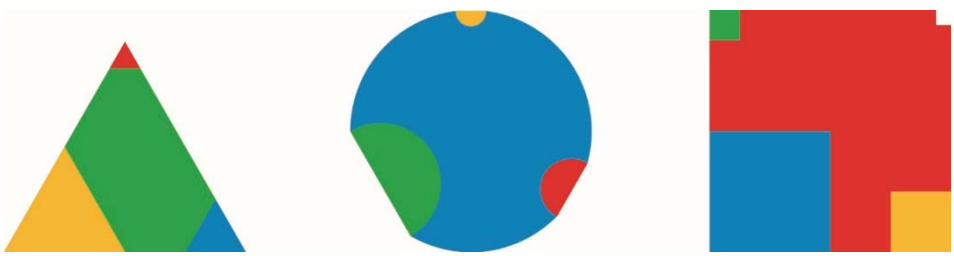

Figure 1: Basic elements of Poly-Universe: triangle, circle and square.
(Source: Saxon Szász, Stettner, *et al.*, 2019)

2 ABOUT THE GAME

Hungarian artist János Saxon Szász (Perneczky, 2002) created the first basic artwork in 1979 he called the 'Universe', and continued to carve and paint more complex MADI objects as modules of Poly-Universe till 2009. He chose three elemental geometrical shapes of the plane—the triangle, the circle and the square—each to which he added other inscribed shapes (Fig. 1). He drew smaller squares with different colours and sizes to the main square's corners. The same game applied to the triangle, and three different semi-circles were attached to the basic circle, both having a triangular pattern therefore. After producing three basic prototypes, Saxon Szász (2010, pp. 85–111) noticed that he might apply different combinations of the given colours to the same shapes. As a consequential result, 24-piece packs were worked out from every prototype (Fig. 2).

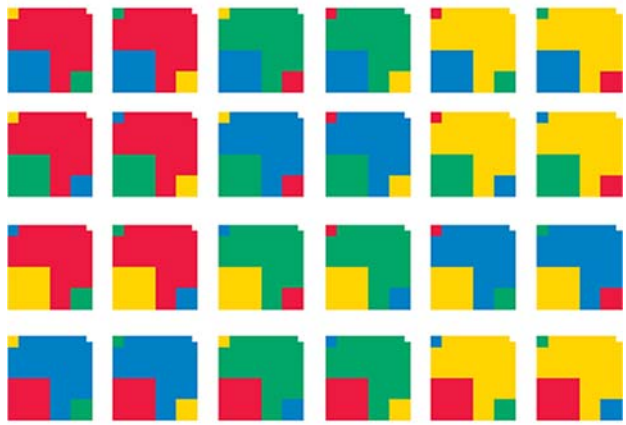

Figure 2: The 24-element square pack based on the possible colour combinations.
(Source: Saxon Szász, Stettner, *et al.*, 2019)

The pieces could then be organised into greater free or tied forms with optional combinatorial logics, where players could choose among the possible rules. Most evidently, the shapes were the nexus between the pieces. For example, players could attach the identical shapes of the same or a different colour to each other in order to find interesting patterns. Aligned with an educational research (Saxon Szász, Stettner, *et al.*, 2019, 407–414[1]; Darvas, 2019), a combinatorial quest evolved from this experiment, setting up two criteria:

1. Use all $n \times 24$ elements to create a greater form (n is a positive integer), and
2. Apply the same rules to all the elements used.

There are some which proved to be less or more feasible among the possible rules. Based on certain experiences without the use of computers, the difficulty of solutions increased with the compactness of the greater form and the homogeneousness of the corner patterns. The latter means that only identical colours and shapes meet at the corners. Our investigation focused on the possible solutions for the notably most difficult rules (the most compact form with the most homogeneous corner patterns).

[1] The numbers represent task sheets, not pages.

We have chosen the square pack also because it proved to be the hardest among the three to solve with the given rules. The positive integer of the first criterion was defined as $n = 2$, for it is easy to see that $n = 1$ does not provide any solution for the square pack. With these premises, we developed a software algorithm to comply with the following requests:

1. Use complete square pack with 48 elements ($n = 2$),
2. Choose the most compact form (7×7 square with one empty slot), and
3. Give the most homogeneous corner pattern (only identical shapes and colours meet).

Compared to the triangle or the circle, the peculiarity of the square pack is that one of the corner shapes are 'missing', which means it has no colour, and its tiny shape is cut out of the base square. The tiny hole may either be considered as if it had a 'fifth colour' that is always combined with the tiniest corner shape (see the square in Fig. 1). Other corner squares alternate between three different sizes and four different colours (red, yellow, green and blue) compared randomly, but with the condition that the same combinations may only be used n times (one full table pack contains 24 different elements).

3 THE TERMINOLOGY

While such a definition of the Poly-Universe keeps it within limited parameters (Saxon Szász, 2010), we understand it to be a game played on a board, placing cards from one's hands following certain rules. In order to get into the concept, we need to develop a vocabulary. Our definitions are as follows.

Game: used to describe the entire activity.

Game State: denotes the status of the board and our hand at a given point in time.

Card: piece of equipment to be placed. In this game, cards are squares with four edges, each of a different size and colour, one of them a small cut.

Pack: a full collection of distinct cards, one of each possibilities.

Hand: a collection of cards.

Edge: part of the card that can be considered for matching purposes. Edges have a colour (or shape) as well as size. The base colour can be considered an edge, too, for the lack of a better word.

Joker Edge: an edge that can be matched with anything else.

Joker [Card]: a card consisting of joker edges.

Board: the area where cards are being placed.

Layout: the angle and position of the cards, relative to each other.

4 EARLY CONCEPTS

The very first approach has been to use Windows Excel to try to produce all possible board combinations of the $n = 1$ basic pack (24 pieces and 5×5 square board with one empty slot), and to simply validate them with the following rules:

1. Adjacent cards should have matching colours, and
2. No card shall be used twice.

In all cases, the row number would be split (using a combination of modulus and integer division).

The most basic solver tries all cards in all cells, which results in 24^24 variations (cf. Saxon Szász, 2010, p. 110). While its simplicity may appear promising, the combinations to validate are overwhelmingly numerous. One slight refinement ensures that cards cannot be reused, producing 24! (620448401733239439360000) variations—an 80 bit number (2^79.04).

This reduces the cases massively by removing one cause for invalidity; however, it also adds a significant amount of calculation: for every cell, previously used cards have to be excluded. However, this is also computationally burdensome, while still being restricted to a single pack. Two things became clear very quickly:

1. Excel cannot contain the required number of rows, and
2. It cannot handle numbers large enough to describe individual cases.

5 THE SECOND APPROACH

Some logical restrictions have been noted. First, the starting card can be fixed, as the game is symmetric. Colours can be considered as parameters (numbers or letters), so any solution found has 24 variations for all possible colour combinations.

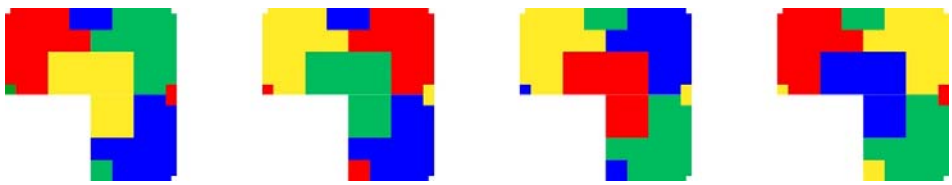

Figure 3: A combination of three-card sets from the 24-card pack, applying the colour-shape-match rule—fourth cards cannot be placed. (Drawing by the authors)

Secondly, a two-by-two cell with the shaped edges at its corners cannot be filled using a single pack (see Fig. 3). Once the first card is placed, the two cards adjacent to it are deterministic: two edges are fixed, the third is the one not yet used. This leads to a scenario where the fourth card needs to have the same colour at its diagonal edges. Hence a 5×5 board cannot be filled using a single pack.

Parallel to this, a new idea has been considered: since this game requires edges to match completely (colour and size, also meaning shape), it is possible to consider the connection points on the board to be solved (Fig. 4). This presented a last opportunity to try to produce all possible placements given their very limited nature. It still proved to be a dead end, stretching the limits of the tool with a small board.

6 THE C# APPROACH

We used PRO C# 7 multi-paradigm programming language for generating, testing and optimizing (Troelsen and Japikse, 2017). During the process, we considered some relevant ‘mantras’ mentioned by today’s developers and learning sites[2]:

[2] See https://www.dofactory.com , or https://dzone.com .

Keep writing flexible, with extendible code. This usually means adding various abstractions to the code, at the expense of runtime and memory consumption, regarding additional objects to instantiate and interfaces to resolve (cf. Gibson and Senn, 1989).

Premature optimisation leads to unreadable code.

Do not implement everything in front, because the chance that *You Ain't Gonna Need It* (YAGNI) is very high (Wäyrynen, 2004).

Keep It Simple, Stupid (KISS). If there is a straightforward solution, prefer that over a complicated, fancy one (Kemerer, 1995).

+	R	R	+	+	Y	Y	+	+	R	R	+	+	Y
B	G	G	B	B	R	R	B	B	Y	Y	G	G	B
B	G	G	B	B	R	R	B	B	Y	Y	G	G	B
+	Y	Y	+	+	G	G	+	+	R	R	+	+	R
+	Y	*	*	+	G	G	+	+	R	R	+	+	R
R	B	*	*	Y	R	R	Y	Y	G	G	Y	Y	B
R	B	B	Y	Y	R	R	Y	Y	G	G	Y	Y	B
+	G	G	+	+	B	B	+	+	B	B	+	+	R
+	G	G	+	+	B	B	+	+	B	B	+	+	R
B	Y	Y	R	R	Y	Y	G	G	R	R	G	G	B
B	Y	Y	R	R	Y	Y	G	G	R	R	G	G	B
+	G	G	+	+	B	B	+	+	Y	Y	+	+	Y
+	G	G	+	+	B	B	+	+	Y	Y	+	+	Y
Y	B	B	R	R	G	G	R	R	G	G	R	R	B

Figure 4: A possible solution of the two-pack board (7×7) provided by the test console. Each card has an identical pair, and all the 2×24 cards are consumed. Assuming that the shapes are fixed, we focused on the combination of colours (R: red, Y: yellow, G: green, B: blue, +: small cut, *: joker edge) that match at the edges. The top left corners of the boards are solved logically.

Depending on personal experience, such advice can seem contradictory. In any case, refactoring is possible.

Given our earlier experiences with brute force algorithms, we opted for a simple solution to try to increase our chances at an acceptable runtime and memory consumption. This means using simple data structures, such as integer arrays, and trying to extract that part of the model, which does not change while progressing to a solution. Where optimisation did not seem beneficial, we left the door open to improve testability and extendibility.

For our first attempt, we picked the easiest rule set: straightforward to implement and easy on resources. This rule set assumes:

A rectangle-shaped grid, where a few slots can be left empty (represented by a joker card).

Neighbouring edges must match their size and colour.

Specifically, trying to fill a 7×7 board with 2 packs of cards and a single joker (hole).

Other variations exist with an elevated level of possibilities based on the original concept of the inventor Saxon:

Open shape: the algorithm has to figure out where to put the cards, while avoiding arriving at identical partial solutions in multiple ways.

Shifted cards: instead of creating a textbook grid, cards could be aligned to match the size of the coloured edge.

Different colours: instead of matching colours, they can or must be different.

Different edges: instead of matching sizes, they can or must be different.

Going beyond edge matching and requiring distant edges to meet rules, possibly expressed otherwise: in each group of four, each colour/shape combination must appear exactly once.

5 colours (pack of 60 cards with the cut edge or 120 cards without).

We realised and utilised the symmetry of the board:

> Alignment: The card alignment is the same in all corners, meaning that the board implicitly contains its horizontally, vertically and diagonally mirrored solution. At the same time it is possible that we find multiple solutions which can be transformed into each other.
>
> Colours: once we find a solution, we can swap the colours around freely, which results in 24 variations.
>
> The algorithm does not use edge size explicitly, so any size variations (diagonal flipping) are valid.

In order to avoid identical duplicate partial solutions, the board is solved in a deterministic sequence. The main idea was to solve row by row, then column by column. It is slightly better, though, to solve the game in 2×2 blocks: once the first card is placed, the ones next to it have two edges defined, reducing the number of possible cases. Doing so, one must realize that jokers must be placed proactively, e.g., right away in the top left corner, instead of waiting to get stuck.

Starting with all 49 possible variations is not better—any cases with the joker in the last 3 rows (or columns) would be solved the same way, ultimately failing. A deterministic sequence, so that we do not get the same result, could be slightly optimized yet for jokers.

Other ideas came up like a deterministic first piece, and a fixed layout due to the rules. The latter helped with the shapes, since if they are known, we may focus on the colours. An easier rule set results in a fixed layout and the smallest number of possibilities, compared to the “different edge”, “different colour” or “non-deterministic layout” variations.

We also researched the possibility when we start with all the possible locations of the joker, and with the possibility of multiple jokers. Next, we noticed there is no solution with a joker at the edge. It means that we would test a lot of identical cases with the joker in the bottom line.

7 TESTING

Unit testing is a fad these days. We could not identify very many cases, but verifying the basic logic of the game was useful, resulting in important questions to answer:

> Are the cards in the pack different?
>
> Are there exactly two states in the first generation, placing either the joker or first card?
>
> Does the single-pack, no-joker game run following its deterministic path?

Our focus has been on a (rather simple) presentation of game state and verifying its correctness. In addition to observing cases, we also added the ability to observe the progression of a single solution by keeping parent-child relationships. This allowed us to verify that cards are indeed placed in the correct order.

7.1 Case A: 2×2 board, 1 card pack, 1 joker

Result: There are four results, with the joker in each position. (Mathematically, these are mirrors of a single solution, the board being symmetric.)

Once the first non-joker card is placed, the next two cards are determined (two edges must match, and there is only one other card with the same two edges, the one with a different 3rd colour), eventually requiring the 4th card to have the same colour on the opposite edges (Fig. 5).

*	*	R	+
*	*	G	B
R	G	G	B
+	Y	Y	+

Figure 5: A single hand determined by a joker card in the top left corner (R: red, Y: yellow, G: green, B: blue, +: small cut, *: joker edge).

7.2 Case B: 2×2 board, 2 card pack, 0 jokers

Result: There are two results, with an additional partial solution with 3 cards (Fig. 6).

+	R	R	+
B	G	G	B
B	G	G	B
+	Y	Y	+
+	R	R	+
B	G	G	Y
B	G	G	Y
+	R	R	+

Figure 6: Two hands determined by the same starter card in the top left corner (R: red, Y: yellow, G: green, B: blue, +: small cut, *: joker edge).

The first card is given. There are two options to the right: the same card again, or the other one with the same two edges. Starting with two identical cards on the first row, there are only two (identical) cards remaining with the bottom colours, with the other colour. However, starting with two different cards, the bottom colour can only be the same as the top colour—red in this case—since the two cards must match and the other three colours are used.

8 RAMP IT UP

Once convinced that the program is correct, we gradually increased the board size to 7×7 (1 joker) and 8×6 (without). The program finds 2 solutions for the 7×7 case (see Fig. 4 and Fig. 7; cf. Saxon Szász, Stettner, *et al.*, 2019, 414), and 12 for the 8×6.

The 8×6 is much lighter, taking about 0.274 seconds, peaking at 6048 ways to place 25 cards. The 7×7 on the other hand takes 13.281 seconds, peaking at 289036 ways to place 30 cards (including the joker), consuming overall 2–4 GB memory in the process. Curiously enough, while there are 198702 ways to place 20 cards already, it drops to 62690 ways for 27 cards.

Having so many intermittent solutions inspired us to improve the mechanism to reduce the number of branches, hence the upgrade from a row-by-row to a 2×2 approach. While the change indeed reduced the number of intermediate solutions and thus memory consumption, the runtime did not change a bit (in the debugger) due to a very few extra mathematical operations - suggesting that minimising abstractions was probably a good idea.

For other possible use cases, we have to consider the freedom of rules. Assuming a progressed game, it might tell whether there is a solution, show any or all, and allow hints.

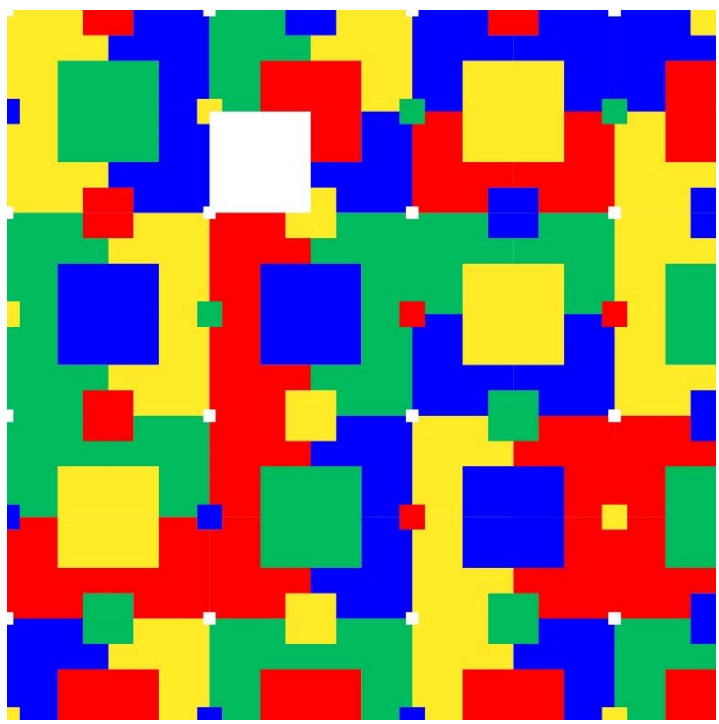

Figure 7: A possible solution of the two-pack board (7×7) provided by the test console. Each card has an identical pair, and all the 2×24 cards are consumed. Shapes and colours match at the edges. (Drawing by the authors)

Considering the resource requirements even for this most simple case, improvements are needed to make it viable for players, especially for rules giving more freedom. As achievements, we could save partial solutions. We did not use threads, which could improve runtime while hiding processing efficiency.

9 FURTHER PROSPECTS

Applying a new concept with the simple design (YAGNI) and premature optimisation, we got completely opposite results. The advanced method to find the next cell generated fewer cases with the same runtime.

For further extensions we need to decide if we go along with object-oriented or procedural solutions. We also need to decide if we should use a static container for method references or solver classes.

An exploration of multithreading would require to split the processing into two parts, which means the algorithm would find the next cell first before it fills it. But can we process two threads, or follow multiple queues? Are generations needed anyway?

If a rule set produces too many solutions, processing could be altered from horizontal to vertical (generations). After finding a result, the process should be stopped and saved. It is also possible that calculation follows a diagonal or Z path (in Excel: A1, B1, A2, B2, C1, C2, D1, D2...) on the board. If a joker is placed, the game should continue with another four pieces.

In each case, we should be able to save the actual state of the calculation process. It could improve the computer's capacity if we discard the partial and unsuccessful results. This diary may also help reconstruct threads, but it overtaxes the memory beyond a given amount. Overall, Saxon's Poly-Universe is a challenge not just to human thinking, but to modern computational science as well.

REFERENCES

Darvas, G. (2019) Poly-Universe in School Education, *Symmetry: Culture and Science*, 30, 3, 251-255. https://doi.org/10.26830/symmetry_2019_3_251

Gibson, V.R. and Senn, J.A. (1989) System structure and software maintenance performance, *Communications of the ACM*, 32, 3, 347–358. https://doi.org/10.1145/62065.62073

Kemerer, C.F. (1995) Software complexity and software maintenance: A survey of empirical research, *Annals of Software Engineering*, 1, 1–22. https://doi.org/10.1007/BF02249043

Palócz, K. and Katona, V. (2019) The applicability of geometrical games in designing modular housing solutions, *Symmetry: Culture and Science*, 30, 1, 25–41. https://doi.org/10.26830/symmetry_2019_1_025

Perneczky, G (2002) *The Poly-Dimensional Fields of Saxon-Szász*, Budapest: International Mobile MADI Museum Foundation, ISBN 963-204-948-9.

Saxon Szász, J. (2010) *Poly-Universe of Saxon*, Budapest: Author's Edition, ISBN 978-963-08-0303-8.

Szász SAXON, J., Stettner, E., eds. (2019) *PUSE (Poly-Universe in School Education) METHODOLOGY – Visual Experience Based Mathematics Education*, Szokolya: Poly-Universe Ltd. (Publisher: Zs. Dárdai), [open access in pdf from http://poly-universe.com/puse-methodology/ 254 p. ISBN 978-615-81267-1-7].

Troelsen, A. and Japikse, P. (2017) *Pro C# 7 With .NET and .NET Core*, 8th ed., New York: Apress, ixv + 1372 pp. https://doi.org/10.1007/978-1-4842-3018-3

Wäyrynen, J., *et al*. (2004) Security engineering and eXtreme programming: An impossible marriage? In: Zannier, C., Erdogmus, H. and Lindstrom, L., eds. *Extreme Programming and Agile Methods – XP/Agile Universe 2004: 4th Conference on Extreme Programming and Agile Methods* (Calgary, Canada, August 15–18, 2004, Proceedings), Berlin: Springer, 117–128. https://doi.org/10.1007/978-3-540-27777-4_12