Analysis of Fred Horn's Gloop Puzzle

Cameron Browne(✉)

Queensland University of Technology, Gardens Point, Brisbane 4000, Australia
`c.browne@qut.edu.au`

Abstract. Gloop is a tile-based combinatorial puzzle game with a strong topological basis, in which the player is assigned a number of challenges to complete with a particular set of tiles. This paper describes the computer-based analysis of a number of representative Gloop challenges, including the computer-assisted solution of a difficult problem that had stood for over a decade.

1 Introduction

Gloop is a tile-based combinatorial puzzle game by renowned Dutch game designer Fred Horn. It is based on the simple mathematical premise shown in Fig. 1: given a square tile with two equidistant *vertices* along each side (left), in how many topologically distinct ways can non-intersecting *paths* be drawn to connect different vertices? Figure 1 (right) shows one such possible tile.

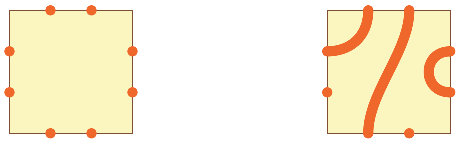

Fig. 1. Square tile with two vertices per side and non-intersecting paths between them.

The Gloop tile set consists of the complete set of distinct tiles that satisfy this premise, and the Gloop game consists of a number of solitaire puzzle challenges that the player can attempt to solve using this tile set. This paper examines four representative Gloop challenges, from the simplest to the most difficult, and demonstrates how computer analysis can provide a deeper understanding of such problems and yield key insights into them. It outlines one particular insight that led to the answer of a combinatorial challenge that had stood for over a decade.

Gloop was devised by Fred Horn for the Convention of Puzzlers held in The Hague, Holland, in 1995, where it was distributed on leaflets as "Puzzle 95" along with Challenges I, II and III listed below. The puzzle was not officially published until Dutch journal *Natuur & Techniek* presented it as their "Grote

Zomerpuzzel" ('Big Puzzle for the Summer') in 2003 [1,2] along with additional challenges from Horn including Challenge IV below.

Horn introduced me to the puzzle at the 2016 Computer and Games conference in Leiden. It has since been published as a physical set [3], for which it was renamed from "Puzzle 95" to "Gloop" (for "Get LOnger Or Perish" and the organic-looking shapes it produces). Further challenges were added for its release, and a domino-like two-player strategy game played with the tiles devised by Horn to supplement the solitaire puzzle challenges. This paper will focus on the solitaire puzzle version of the game.

2 Challenge I: Number of Tiles

The first challenge proposed by Horn in 1995 was:

How many distinct tiles are there?

There are $T = 91$ distinct Gloop tiles, as shown in Fig. 2, including reflections but not rotations. This number includes the blank tile with 0 paths (upper left). This set was enumerated manually by Horn at the time and has been verified by computer enumeration.

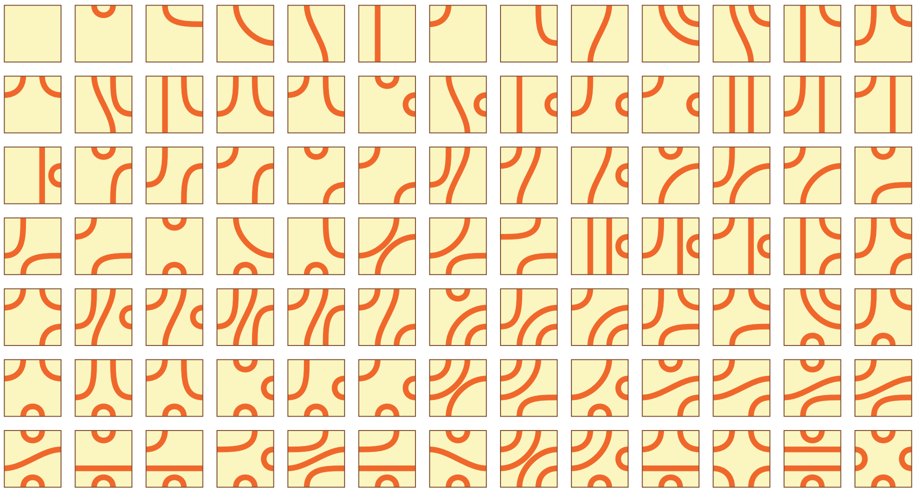

Fig. 2. The complete set of 91 Gloop tiles.

Note that if symmetry is ignored, then the total number of tiles is 323. This is the *Motzkin number* M_8, which indicates the number of different ways that non-intersecting chords can be drawn between 8 points on a circle [4]. However, we are only interested here in the unique set of $T = 91$ distinct tiles excluding rotations.

3 Challenge II: Valid Packings in a Rectangle

The second challenge proposed by Horn in 1995 was:

How to pack the tiles in a rectangle such that neighboring tile edges match?

Definition 1. *A packing of Gloop tiles is* **valid** *if adjacent tiles align exactly in a square grid, and every path segment end meets a neighbouring path segment end such that the final packing forms a set of closed contours.*

The dimensions of the target rectangle can be easily determined. There are two rectangles that fully pack $T = 91$ square tiles: 1×91 and 7×13. A 1×91 rectangle will obviously not allow a valid packing, since tiles with path ends on more than two sides would necessarily leave open contours, hence valid packings can only fill a 7×13 rectangle.

3.1 Complexity

Gloop belongs to the same class of *unsigned edge-matching puzzles* as the notoriously difficult Eternity II puzzle, which Demaine and Demaine have shown to be NP-hard [5]. Eternity II remains unsolved today, except by its inventor.

Ansótegui et al. distinguish between a *generic edge-matching puzzle* (GEMP) and a *framed generic edge-matching puzzle* (GEMP-F) [6]. Gloop is a GEMP but not a GEMP-F; while there is an outer frame formed by the boundary of the packing region which must contain *edge tiles* (i.e., tiles with at least one blank side), these can also be validly placed at interior cells in Gloop, which violates the frame condition. Ansótegui et al. further distinguish between *one-set* and *two-set* GEMPs, depending on whether the tile set can be separated into two subsets: those with edge colors specific to the frame set and those with edge colors specific to the interior set. Gloop is a one-set GEMP.

Denoting the absence or presence of path ends along a tile side as `0` and `1`, respectively, then each tile side must show one of the following four patterns: {`00`, `01`, `10`, `11`}. These are the 'colors' on the tile sides that must match neighboring colors. This low color count means that for any given exposed tile edge there will typically be many potential matching neighbors. This makes the task of finding a valid Gloop packing orders of magnitude easier than solving Eternity II, which involves many more tiles ($T = 256$ as opposed to $T = 91$) and many more edge colors (19 as opposed to 4).

Valid Gloop packings should therefore be relatively easy to find, and the solution shown in Fig. 3 was indeed soon submitted by Dutch puzzlist J.A.M. Mes in response to the 2003 publication of [1]. Note, however, that this solution was produced by a computer program and that no valid packing by hand has yet been recorded.

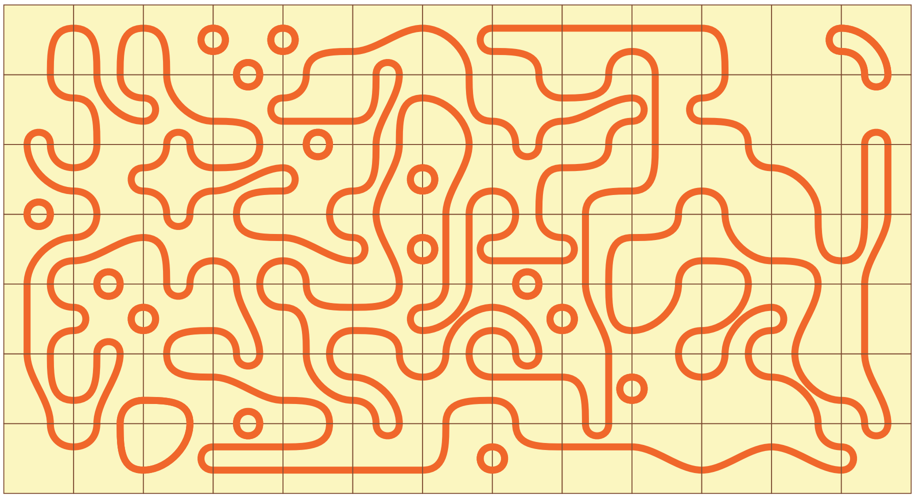

Fig. 3. A valid 7×13 packing by Mes [2].

3.2 Approach

To investigate the ease with which valid packings can be found, *depth-first search* (DFS) – the simplest form of combinatorial search – was implemented over various grid sizes $C = 4, 5, \ldots, 91$. Starting with an empty grid, each cell was filled with a valid tile placement in a random order, backtracking as necessary. For grid sizes with less than $C = 91$ cells, the most square $n \times n$ or $n \times (n-1)$ grid containing the required number of cells was chosen, even if it meant having one *ragged* or incomplete side. This choice of dimensions was made to minimize the number of exterior and corner tiles in each case, which were observed to be a limiting factor in achieving packings.

However, while DFS has the benefit of enumerating all possible solutions with the option of exiting early on the first solution found, it is highly sensitive to processing order [7]. DFS was found to produce valid packings either very quickly for a given placement order (within milliseconds), or very slowly once dead-end branches of the solution space had to be explored. It is quicker in the long run to halt such dead-end searches and try for a more amenable placement order.

For this reason, an improved *depth-first search with random restarts* (DFS-RR) was implemented. This involves performing DFS from the empty state, then shuffling the tile placement order and restarting if a solution is not found before a specified time limit is reached, until a solution is found.

The search was further optimized by heuristics including:

– Pre-ordering cell fill order from most constrained to least constrained cell.
– Backtracking if *any* unfilled cell has no possible tile placements, i.e., not just the next cell in the fill order.

– Backtracking if any unfilled region has an odd number of path ends opening onto it.

3.3 Results

DFS-RR was applied to all grid sizes $C = 4, 5, \ldots, 91$ with a default timeout of 0.1s, which proved most effective over most cases. Figure 4 shows solution times in seconds using DFS-RR with a timeout of 0.1s, averaged over 100 packings per grid size for Challenges II, III and IV. All timings were made on a single thread of standard laptop machine with a 2 GHz $i7$ processor.

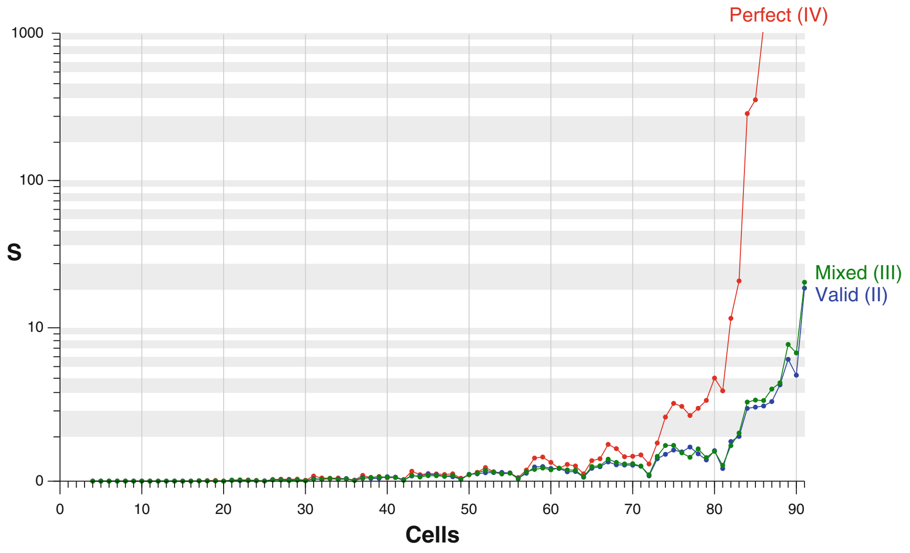

Fig. 4. Average solution times using DFS-RR (0.1 s) for the various challenges.

Valid packings are found very quickly using DFS-RR, in typically less than 1 s for grid sizes up to $C = 82$, to around 20 s for complete $T = 91$ packings on the full 7×13 rectangle. This challenge is very amenable to automated solution.

Note that timings decrease slightly at certain grid sizes against the general upwards trend: 49, 56, 64, 72, 81 and 90. These are the grid sizes that completely fill an $n \times n$ square or $n \times (n - 1)$ rectangle without a ragged edge, indicating that grids with simpler boundaries are easier to pack.

4 Challenge III: Mixed Packings in a Rectangle

The third challenge proposed by Horn in 1995 was:

How to pack the tiles in a rectangle to form a single contour plus circles?

Definition 2. *A closed circular contour formed by two semi-circular path segments is described as* **trivial** *and all other closed contours as* **non-trivial***.*

Definition 3. *A* **mixed** *packing of Gloop tiles is a valid packing formed by exactly one non-trivial contour and any number of trivial contours.*

For example, Fig. 5 shows a mixed solution consisting of one large non-trivial contour and six trivial circle contours, generated using DFS-RR with an additional heuristic:

– Backtracking if more than one non-trivial contour is closed.

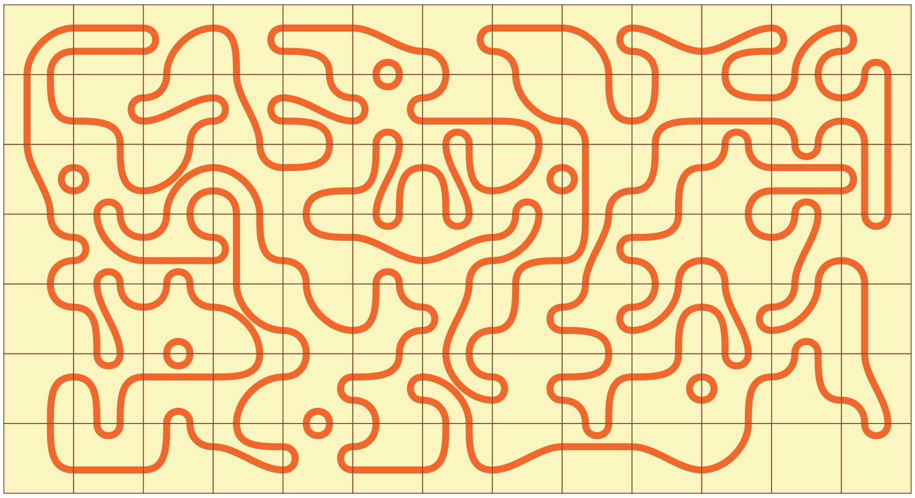

Fig. 5. Mixed solution with one non-trivial contour and six trivial (circular) contours.

This task proved almost as amenable to solution by DFS-RR as finding valid packings, as shown in the timings in Fig. 4. It can be seen that there is very little difference between the average time taken to find valid packings and mixed packings. The example shown in Fig. 5, with six trivial contours, has the lowest contour count of any mixed packing found so far.

5 Challenge IV: Perfect Packings in a Rectangle

An additional challenge was proposed by Horn in 2003 [1]:

How to pack the tiles in a rectangle to form a single closed contour?

Definition 4. *A packing of Gloop tiles is* **perfect** *if it is valid and consists of a single closed contour (which must necessarily be non-trivial).*

Finding perfect packings is the most difficult Gloop challenge, as tiles must not only satisfy the *local* constraint of matching their immediate neighbors, but must also now satisfy the *global* constraint that all path segments of all tiles eventually join to form part of the same contour. This global constraint elevates the puzzle from something simple to something much more difficult – impossible, as it turns out – and the question of perfect Gloop packings had remained an open one until this study.

5.1 Approach

DFS-RR was enhanced with the following heuristic:

– Eliminate placements that would create a trivial contour.

This enhancement was motivated by the fact that *no perfect packing can ever contain a trivial contour*, as that would constitute at least a second contour. The test for trivial placements is efficient to implement, and involves simply checking whether any given placement would make two tile sides with semi-circular path segments meet.

Additional search algorithms were implemented for this more difficult task:

– *A**: A* is a best-first search of neighbors in the state space, ordered in a priority queue by a cost function and a domain-specific heuristic function [7]. The heuristic function here was based on the number of potential future placements that each placement allowed.
– *Monte Carlo Tree Search* (MCTS): Standard UCT [8] was implemented, with each playout consisting of a series of random valid placements in the pre-defined cell order until no more such placements could be made, and returning as reward value the ratio of filled cells to total grid size. A default UCB exploration constant of 0.25 was used. The *Single-player Monte Carlo Tree Search* (SP-MCTS) variant [9], which proved effective for other solitaire puzzle domains, did not provide any benefit over standard UCT in this case.
– *Iterated local search* (ILS): ILS is a simple metaheuristic for tackling problems of high complexity by iteratively generating random states then perturbing them and *hill-climbing* to superior neighbors in the state space [10].

The ILS search implemented here exploits the fact that one valid packing can be transformed into another by *swapping* tiles with identical path end distributions. For example, Fig. 6 shows a sequence of tile swaps that reduces the contour count of a valid packing with each iteration to produce a perfect packing. The cells highlighted at each step show which tiles are swapped, and the contour highlighted is the one being merged with neighboring contours. Toulis [11] describes a similar tile swapping mechanism as a means of hill-climbing for the Eternity II puzzle.

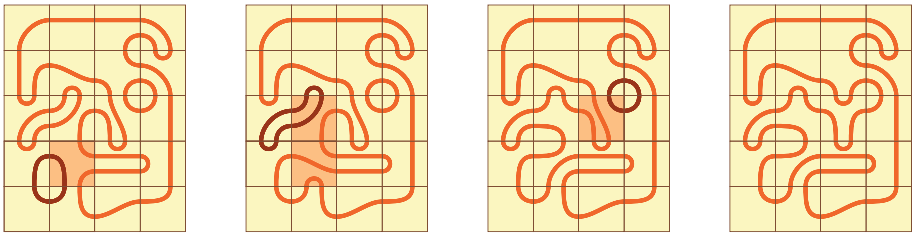

Fig. 6. A sequence of compatible tile swaps from a valid to a mixed to a perfect packing.

5.2 Results

The timings shown in Fig. 4 reveal that perfect packings are found almost as quickly as valid and mixed packings for smaller grid sizes using DFS-RR (0.1 s), but show a sudden exponential increase in solution time from around $C = 72$. No perfect packings were found beyond $C = 87$.

A* and MCTS were then applied to find perfect packings but performed so badly that their results are not worth reporting. This is probably due to the additional overhead required to perform these searches outweighing any benefit. ILS was found to significantly reduce search time for the $C = 87$ case, as shown in Table 1, but again could not find any perfect packings beyond $C = 87$. Timings are averaged over 100 attempts.

Table 1. Solution times for larger perfect packings (in seconds).

	C	Mean	Min	Max
DFS-RR (0.1s)	83	21.89 ± 8.90	0.17	108.11
	84	312.16 ± 91.83	1.37	946.33
	85	387.44 ± 108.24	10.93	1293.13
	86	1751.53 ± 544.42	41.38	6682.91
	87	11459.62 ± 3076.93	823.77	29821.78
ILS	83	251.03 ± 102.75	4.79	1279.57
	84	318.44 ± 178.52	1.49	2006.11
	85	616.07 ± 226.77	6.75	2325.19
	86	980.19 ± 382.84	19.20	3890.20
	87	4042.94 ± 1214.84	188.69	11304.94

5.3 Discussion

Valid and mixed packings proved highly amenable to a simple search, while larger perfect packings remained unobtainable using all methods tried. So why did perfect packings become so intractable at $C = 88$?

The answer to this question came after the game's publisher, Néstor Romeral Andrés, suggested checking for "problem" tiles that tended to be excluded from solutions, and it was indeed found that the tiles shown in Fig. 7 were excluded more often from larger perfect packings than other tiles. Further, the leftmost tile, which contains the maximum number of side-centred semicircles, was excluded from almost *every* perfect packing for larger grid sizes. This insight led to the following proof that the full $T = 91$ tile set does not allow a perfect packing.

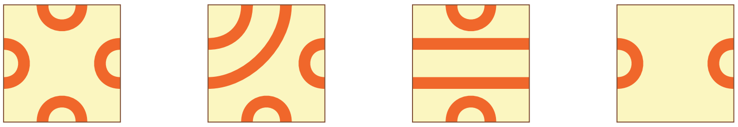

Fig. 7. Tiles typically missing from larger perfect packings.

5.4 Proof of No $T = 91$ Perfect Packings

This section provides a simple geometric proof that no perfect $T = 91$ is possible.

Definition 5. *Two path segment ends that terminate at vertices on the same tile side constitute an* **end pair***.*

There are four basic end pair types (Fig. 8).

1. A *cap* is a side-centered semicircle that connects the two vertices on a side.
2. An *extension* is a pair of parallel path segments that connect the two vertices of a side to the two vertices of another side.
3. A *split* is a pair of path segments that connect the two vertices of a side to a vertex on each of two different sides.
4. A *junction* occurs when three (or four) end pairs create a common region defined by their three (or four) path segments.

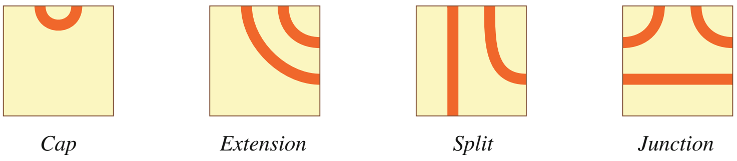

Fig. 8. Examples of the four basic end pair types.

More than one end pair type can co-exist on the same tile. The full set of $T = 91$ tiles, shown in Fig. 2, contains 58 caps, 13 extensions, 44 splits, 2 triple junctions and 1 quadruple junction.

Caps and Anticaps. Caps exist in a state of *atari*,[1] as they are one placement away from forming trivial closed contours. For example, Fig. 9 (a) shows a cap being closed by another cap, which can only occur if at least one of the tiles involved contains at least one other path segment, as shown (since no tile can be duplicated). Figure 9 (b) shows that extending a cap does not affect its state of *atari*; the contour it creates is still under threat of immediate closure.

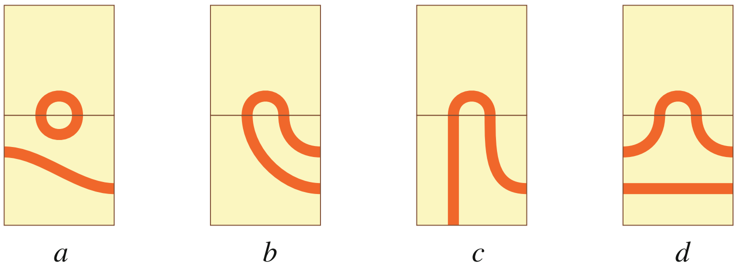

Fig. 9. The effect of each end pair type on caps.

Fortunately, splits defuse caps by deviating the open end pair to path ends on different tile sides (c), to remove the immediate threat of closure. Similarly, junctions inflate a cap's open end pair to two or more open end pairs that can not both be closed by a single tile placement (d). Splits and junctions therefore represent *anticaps* that negate a cap's immediate threat of closure.

There are a total of 48 anticaps in the $T = 91$ tile set, shown in Fig. 2, made up of 44 splits and 4 anticaps provided by the three junction tiles. This leads to a simple proof that the full $T = 91$ tile set does not allow a perfect packing.

Lemma 1: *No perfect packing can contain any cap pairs.*

Proof: At least one of the tiles containing the semi-circular path segments of the cap pair (or extended cap pair) must contain at least one other path segment, which will create at least one other contour in addition to the cap pair.

Lemma 2: *Every cap in a perfect packing must have a corresponding anticap.*

Proof: Every contour in a valid packing must be closed. However, any cap closed by another cap, either directly or through extension, would violate Lemma 1 and disallow a perfect packing. Every cap in a perfect packing must therefore have a corresponding anticap, to separate it from all other caps.

Theorem: *The $T = 91$ tile set does not allow a perfect packing.*

Proof: The full $T = 91$ tile set contains 58 caps but only 48 anticaps. There are not enough anticaps to separate all caps, hence at least one cap pair (or extended cap pair) must occur. This violates Lemma 2, hence the full $T = 91$ tile set does not allow a perfect packing. QED.

[1] To borrow a term from the board game Go.

5.5 Gloop Arithmetic

This analysis gives rise to a simple Gloop arithmetic that can be applied to any subset of tiles that allows a valid packing, to indicate whether they potentially also allow a perfect packing. Let c denote the number of caps, s denote the number of splits, and j denote the total anticap value of junctions within the subset. The *cap sum* (CS) of the tile set is then given by $CS = c - s - j$. For example, the cap sum of the perfect 3×3 packing shown in Fig. 10 is $CS = 4 - 3 - 1 = 0$. Extensions do not affect the cap sum so are not counted.

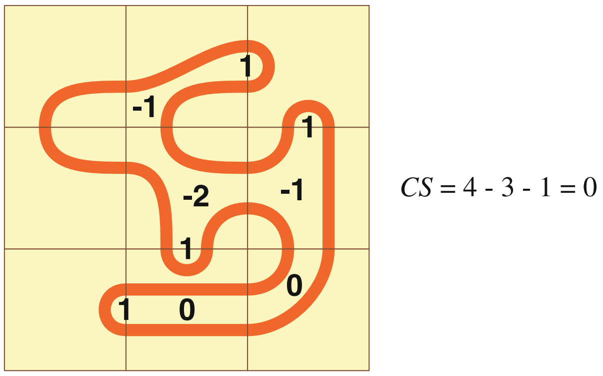

Fig. 10. This perfect 3×3 packing has a cap sum of 0.

The cap sum for any valid packing must be even, as each end pair must have a corresponding end pair. The theoretical upper limit on cap sums for perfect packings, $CS = 2$, cannot be achieved in practice (the single-cap tile would have to be duplicated). The cap sum of a tile set must therefore be an even number ≤ 0 if it is to allow a perfect packing, which provides a simple test. For example, the full $T = 91$ tile set has a cap sum of $CS = 58 - 44 - 4 = 10$, hence does not allow a perfect packing. $T = 87$ is the largest tile subset whose cap sum can be reduced to $CS = 0$ if the most cap-heavy tiles are judiciously removed, hence is the largest subset of Gloop tiles that allows a perfect packing.

6 Conclusion

This study demonstrates a successful collaboration between human and computer-based approaches to solving a difficult problem – the question of perfect Gloop packings had stood for over a decade – where neither approach had succeeded in isolation. Modeling the puzzle mathematically allowed the automated search for solutions, which revealed the unexpected $C = 87$ limit, which inspired the questions leading to the manual proof of no perfect packing for the complete tile set. This negative answer required the revision of the puzzle challenges for its recent release [3] so that players were not set an impossible task.

While the experimental results were initially disappointing, with valid and mixed packings amenable to simple search and complete perfect packings not achieved by any method tried, the resulting geometric proof of no complete perfect packing was satisfying to derive. The 'no trivial contour' heuristic described in Sect. 5.1 forecast this proof by implementing the fact that no two caps can ever meet in a perfect packing. This could also be considered an inspiration for the proof, and shows that simply analyzing a problem for implementation can yield useful insights into it. As Perlis observes: *We measure our understanding (and control) by the extent to which we can mathematize an activity* [12].

An interesting question for further investigation is the following: could the 'no complete perfect packing' proof have been derived through purely automatic means, e.g., as a constraint satisfaction or SAT problem?

Acknowledgments. Thanks to Fred Horn for introducing me to Puzzle 95 (aka Gloop), Néstor Romeral Andrés and Robin Barrett for Gloop-based discussions, and Frederic Maire for suggestions on improving the proof. This work was funded by a QUT Vice-Chancellor's Research Fellowship.

References

1. Horn, F.: Grote Zomerpuzzel: Doorlopende Puzzel. Natuur & Techniek, pp. 52–53, July/August 2003. (in Dutch)
2. Horn, F.: De Grote Zomerpuzzel: Opgelost. Natuur & Techniek, pp. 54–55, October 2003. (in Dutch)
3. Romeral Andrés, N.: Gloop. Nestorgames (2016). http://nestorgames.com/#gloop_detail
4. Motzkin, T.S.: Relations between hypersurface cross ratios, and a combinatorial formula for partitions of a polygon, for permanent preponderance, and for non-associative products. Bull. Am. Math. Soc. **54**, 352–360 (1948)
5. Demaine, E.D., Demaine, M.L.: Jigsaw puzzles, edge matching, and polyomino packing: connections and complexity. Graphs Comb. **23**, 195–208 (2007)
6. Ansótegui, C., Béjar, R., Fernández, C., Mateu, C.: How hard is a commercial puzzle: the eternity II challenge. In: 2008 Conference on Artificial Intelligence Research and Development, pp. 99–108. IOS Press (2008)
7. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Pearson Education, New Jersey (2009)
8. Browne, C., Powley, E., Whitehouse, D., Lucas, S., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Trans. Comput. Intell. AI Games **4**, 1–43 (2012)
9. Schadd, M.P.D., Winands, M.H.M., van den Herik, H.J., Chaslot, G.M.J.-B., Uiterwijk, J.W.H.M.: Single-player monte-carlo tree search. In: van den Herik, H.J., Xu, X., Ma, Z., Winands, M.H.M. (eds.) CG 2008. LNCS, vol. 5131, pp. 1–12. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-87608-3_1
10. Lourenço, H.R., Martin, O., Stützle, T.: Iterated local search. In: Glover, F., Kochenberger, G.A. (eds.) Handbook of Metaheuristics, vol. 57, pp. 321–353. Kluwer Academic Publishers, Dordrecht (2003). https://doi.org/10.1007/0-306-48056-5_11
11. Toulis, P.: The eternity puzzle. Technical report, Harvard University (2009)
12. Perlis, A.: Epigrams on programming. SIGPLAN Not. **17**, 7–13 (1982)