Analyzing Generalized Harary's Tic-Tac-Toe Using Automated Algorithms

Ege Kabasakaloğlu

July 25, 2021

Abstract

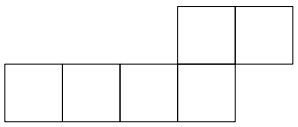
In this paper, we propose a generalized version of the mathematical game Harary's Tic-Tac-Toe. Using Z3, we implement automated algorithms to find domino paving strategies and solve the game for finite boards. We use these algorithms to analyze restricted versions of the "Snaky Problem".

1 Introduction

Harary's Tic-Tac-Toe, first devised by Frank Harary in 1977, is a two-player game played on an infinite rectangular board. The players take turns marking cells on the board. The first player, called the Maker, wins if he manages to mark any set of cells congruent to a predetermined polyomino. The second player, called the Breaker, wins if he can prevent the Maker from achieving his goal. Almost all of the polyominoes in the original game are known to be losing for the Maker, except eleven known to be winning, and one undetermined polyomino called Snaky[1]. In our paper, we propose a generalized version of this game.

In our version, we consider each rotation and reflection of a polyomino as unique, and the Maker tries to achieve any shape in a set of different polyominoes, instead of just one polyomino. It turns out that in this variant of the game, both the winning and losing sets of polyominoes are infinite, so it is hopeless to analyze every situation individually. To combat this, we create several algorithms using the satisfiability modulo theory solver Z3 to automatically classify sets of polyominoes as winning or losing. The first of these algorithms is one that automatically finds domino paving strategies for the Breaker. These strategies were also used to solve a major portion of the original game[1], although previously these strategies could only be constructed by hand. The second one is an optimized brute force algorithm to solve the game for finite boards. This algorithm makes use of various strategies to cut off significant portions of the search tree.

Finally, we use the methods we developed to analyze and gain insight into the only unsolved polyomino in the original game: Snaky.



Snaky

We analyzed the subsets of all rotations and reflections of Snaky, and found that, while some subsets are easily solved with our domino paving algorithm, others pose problems almost as impressive as Snaky itself. We also analyzed these subsets on finite boards of sizes up to 11 by 11 and discussed how our results could be improved.

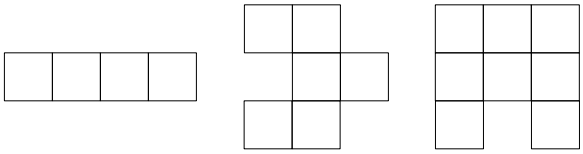
2 Tools Used

All of the algorithms described in the paper were implemented with Z3 version 4.8.10 and C++17. Z3 is a state-of-the-art satisfiability modulo solver. A satisfiability modulo theory, or SMT for short, is a decision problem for formulas that contain many aspects of various mathematical theories, such as the theory of real numbers, the theory of integers, etc. Z3 makes use of different "logics" for each of these different theories to solve them independently and then merges the results. For the algorithms described in this paper that make use of Z3, we reduced each problem to boolean variables and boolean logic before using Z3 to solve them. This approach proved to be effective, as Z3 is well optimized for boolean logic. Moreover, boolean satisfiability problems are NP-Complete, meaning Z3 can find solutions to any such problem we give it, given enough time. We used C++ for its speed and helpful Z3 library.

If you want to examine the codes described in this paper, the github link is `https://github.com/egekabas/PioneerEge`.

3 Description of the game

The game is played on an infinite rectangular board, or more formally, a set of cells that are the translations of the unit square $[0,1] \times [0,1]$ by vectors of $\mathbb{Z}^2$ [2]. Two players called the Maker and the Breaker take turns marking previously unmarked cells. The goal of the first player (the Maker) is to mark a set of cells that corresponds to a translation of any of the polyominoes in a predetermined set. The goal of the second player (the Breaker) is to prevent the Maker from achieving his goal. In our definition of the game, polyominoes are sets of cells such that each cell in a polyomino can be reached by another cell by going through adjacent cells that are also a part of that polyomino. Moreover, each rotation or reflection of a polyomino is also considered a unique polyomino. A set of polyominoes is called winning if the first player can win when playing using that set, or losing if not.



Examples of polyominoes

4 There is an infinite amount of winning sets

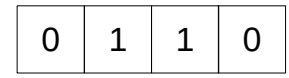
In the original game, Harary's Tic-Tac-Toe, a large majority of the proofs came from the fact that almost all polyominoes are losing when the Breaker uses a simple tactic, so each of the remaining handful of situations could be analyzed individually[1]. It is evident that, in our version of the game there are infinite winning sets, as you can combine a polyomino of size one with any other polyomino and get a winning set. However, the number of winning sets is still infinite even if we limit the types of sets allowed.

Definition 4.1 *A set of polyominoes is called new if none of its proper subsets are winning.*

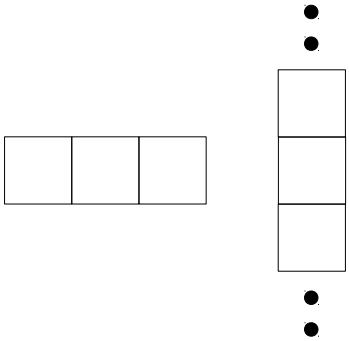
Theorem 4.1 *There is an infinite number of new sets of polyominoes that are winning.*

We can prove the theorem by proposing a method to construct infinite amounts of winning new sets. We first define a horizontal line polyomino as a polyomino created by stacking cells side by side. Moreover, we define a vertical line polyomino as a horizontal line polyomino rotated 90 degrees. Then, we must notice that every vertical or horizontal line polyomino of a size greater than two is losing by itself. This statement can be proved quite easily using the domino paving method discussed later in the paper. Finally, we can prove that the set consisting of a horizontal line polyomino of size three, and a vertical line polyomino of any size greater than two, is both new and winning. Each time the

Maker marks a cell, the Breaker must mark a cell in the same row. Otherwise, the Maker can achieve the position shown in the picture, and there is no way to prevent the Maker from winning.



0 represents unmarked cells, 1 represents cells marked by player 1. No matter what move the second player makes, the first player can achieve the horizontal polyomino of size three.

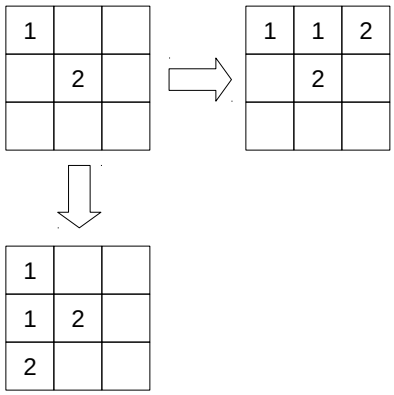


The set of these two polyominoes are winning and new, no matter what the second shape's size is.

Using this knowledge, the Maker can freely mark cells in a column without any resistance and achieve a vertical line polyomino of any size.

Proposition 4.1 *Let S(n) be the set consisting of all the polyominoes of size n. Then, for any n, S(n) is winning.*

We first encountered this proposition in a paper called "Polyominoes with minimum site-perimeter and full set achievement games"[2]. This paper states that the correctness of this proposition is trivial as the Maker can simply mark the cell to the right or bottom of the last cell he marked and can never be blocked. However, this is not true.



As we can see in the image, if, as his first move, the Breaker marks the cell to the bottom right corner of the cell the Maker marked, he can then completely cut off the Maker no matter what move he makes. However, just because this proof is incorrect does not mean the proposition is wrong. We even strongly suspect it might be true; however, there does not seem to be a simple proof, and providing a lengthy proof would be out of the scope of this paper.

As we can see, because the number of winning sets seems to be infinite even if we try to limit them, it is unfeasible to analyze them one-by-one. This was the primary motivation for us to create algorithms to construct strategies automatically.

5 The Domino Paving Strategy

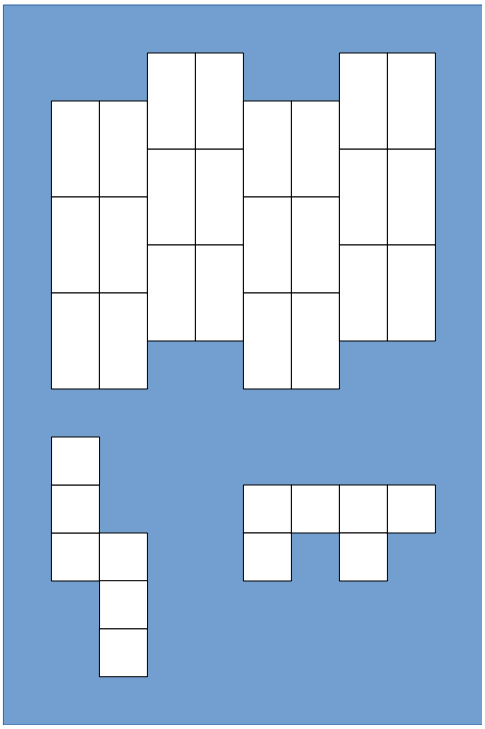
The main strategy used in the original paper to solve most of the game was the domino paving strategy[1], which can also be applied to our version of the game. In this strategy, the Breaker covers

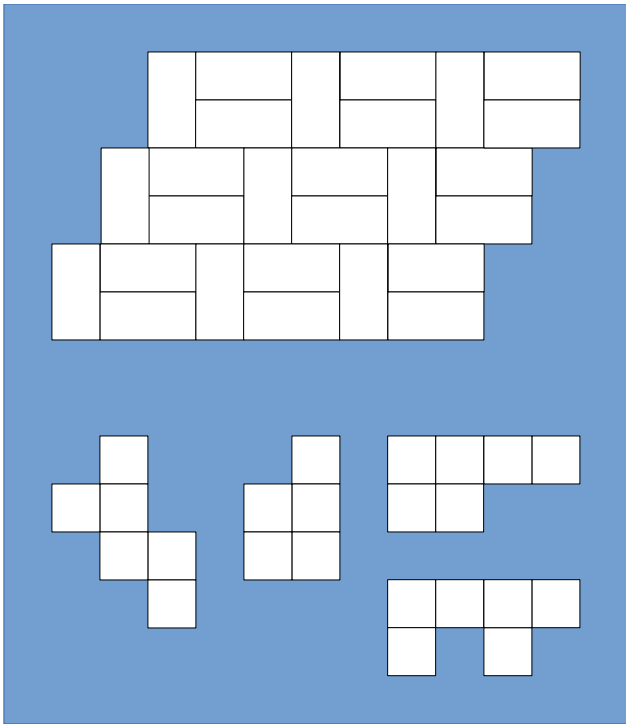
the game board with imaginary domino tiles (two-by-one rectangles). Two domino tiles cannot overlap. Whenever the Maker marks a cell that is covered by a domino tile, the Breaker marks the other cell covered by that domino tile. If there is no way to place any of the polyominoes in a set on the board without containing a full domino tile, then the Breaker can always prevent the Maker from achieving any of the polyominoes in that set. Because the board is infinite, the only feasible way to find a working domino paving strategy is to create a repeating domino pattern.

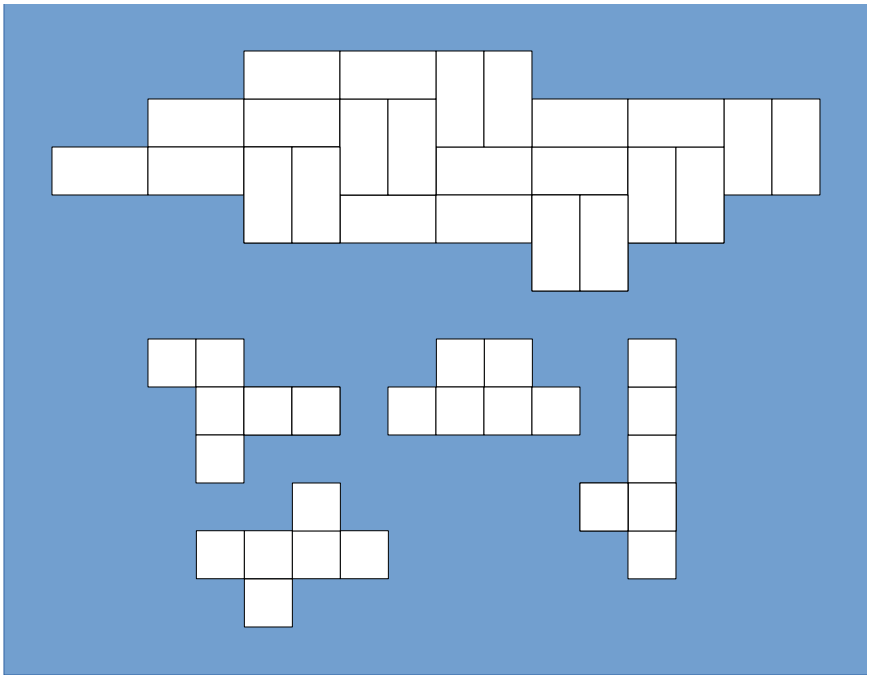
We can check if there is a repeating domino pattern of a specific size that provides a working strategy against a set of polyominoes using Z3. We can represent the direction of the domino tile at each cell in the pattern using boolean variables. Two cells are covered by the same domino tile if their domino tile directions point to each other. Finally, we must go through every possible placement of each polyomino in the set and add to our model that each placement must contain at least one full domino tile. Note that every cell does not necessarily need to be covered by a domino tile, but this is not a problem at all. If the Maker marks a cell that is not covered by a domino tile or the other cell covered by the domino tile is already marked, the Breaker can merely choose a random cell to mark.

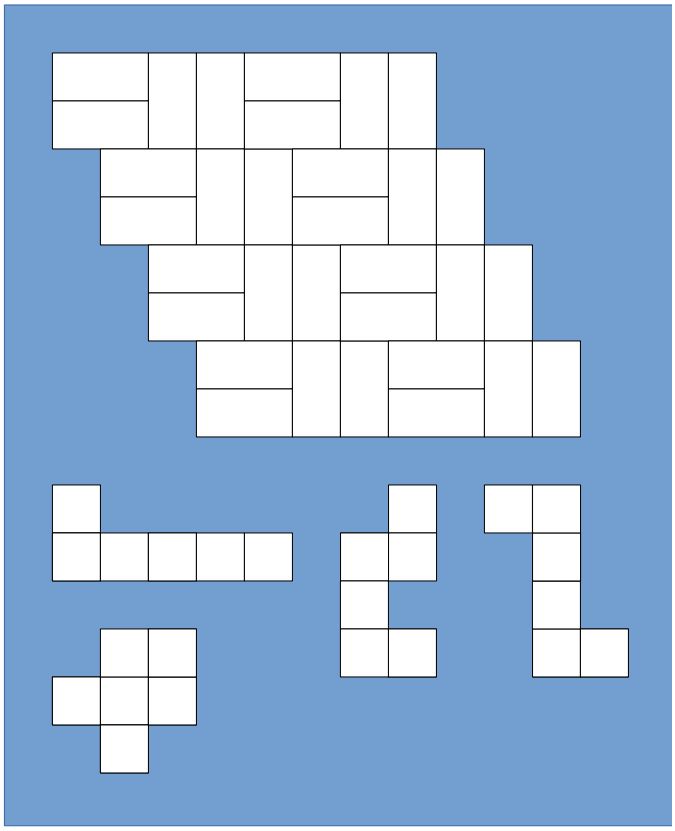
One last thing to be mindful of is that bigger repeating domino patterns do not always necessarily get better results. For example, there might be a winning 4 by 4 domino pattern but not a 6 by 5 one. Therefore, if we want to be sure whether there is a winning domino pattern of size up to a particular value, we must check all possible sizes. Luckily, our algorithm works quite fast, and we can check for all the possible board sizes up to 16 by 16 in less than a few seconds.

We ran the algorithm on a considerable number of random sets of polyominoes. As follows are some of the more interesting domino patterns we found. In the illustrations, we extended most domino patterns to make them easier to understand for the reader. Each domino pattern is given in the same box with the set of polyominoes it can defend against.









It is essential to realize that these patterns were the smallest possible patterns to defend against their respective sets, so no simpler working patterns exist. Coming up with and checking the validity of these patterns would be extremely time-consuming for a human, but the program managed it under one second each.

6 Solving the Game for Finite Boards

While the game is played on an infinite board, a program to solve the game for finite boards is helpful. If the Maker can win on a finite board, he can also win on an infinite board. The same is not true for the Breaker. However, solving the game on a finite board can still provide insight to the solution for an infinite board.

The simplest algorithm that comes to mind is to brute force through all possible moves each player can make. We can also hold the states we previously calculated in a data structure to avoid unnecessary recalculations. However, because the number of different states in which the board can be grows exponentially, this simple approach becomes prohibitively slow following boards sizes greater than 4 by 4. Still, this method can start to perform much better if we implement a number of optimizations.

6.1 The Pairing Defense

This strategy is merely a generalized version of the domino paving strategy. Every cell on the board is grouped into pairs. When the Maker marks a cell, the Breaker marks the other cell in the same pair as that cell. The Breaker can win using this strategy if it is impossible to place any of the polyominoes without containing at least one pair.

Implementing an algorithm to look for this strategy using Z3 is also similar to how we implemented the domino paving strategy. Instead of using boolean variables to represent the direction of the domino tile at a cell, we use them to represent whether two cells are in a pair together. This strategy is slower than the domino paving strategy, as the number of boolean variables is proportionate to n^2 instead of n, where n is the board size (or the pattern size for the domino paving algorithm). Fortunately,

it is not as slow as one might think. This is because Z3 is well-optimized and it can avoid needlessly calculating certain situations. For example, there is no benefit from putting into a pair two cells that can never be covered by a single polyomino.

We can implement this strategy readily into our brute force algorithm. Every turn, we can check if there is a pairing defense strategy. If there is, then the Breaker wins, and there is no need for further calculation. Moreover, if it is the Breaker's turn, then we should look for a strategy after the Breaker makes a move. Instead of going one-by-one through all the moves the Breaker can make, we can add more boolean variables to our Z3 algorithm representing the moves the Breaker can make.

This strategy can also be used on infinite boards by using a repeating pattern, again similar to the domino paving strategy. However, it is more complex to calculate than the domino paving strategy, and during our testing we weren't able to identify a situation where it is more effective.

We must note that this strategy is similar to the ones created by hand to prove that Snaky is losing on a board of size 8 by 8 [3]. However, as mentioned later in the paper, our algorithm is effective for board sizes much larger than in that article (or humanly plausible).

6.2 J. Beck's Theorems for Achievement Games on Hypergraphs

In his paper, J. Beck proved two theorems related to (1, 1) achievement games on hypergraphs[4]. A hypergraph consists of vertices and edges, which are sets of vertices. Players take turns marking one unmarked vertex. The first player wins if he marks a set of vertices corresponding to an edge, and the second player wins if he can prevent the first player from doing so.

Theorem 6.1 *Let $\varkappa$ be a hypergraph, and let A denote an edge and $|A|$ the number of vertices in that edge. Then, if*

$$\sum_{A \in \varkappa} 2^{-|A|} < 1/2$$

then the second player has a winning strategy.

Theorem 6.2 *Let $v(\varkappa)$ denote the number of vertices of $\varkappa$ and let $d_2(\varkappa)$ denote the maximum number of edges of $\varkappa$ containing two given vertices. If*

$$\sum_{A \in \varkappa} 2^{-|A|} > \frac{d_2(\varkappa) v(\varkappa)}{8}$$

then the first player has a winning strategy.

Our game can be defined as a game on a hypergraph: the vertices are the cells, and the edges are the different possible placements of every polyomino in our set on the board. Therefore, we can easily apply the theorems above to our game. After every move, if any of the theorems above hold true, there is no need for further calculation.

6.3 Ordering Moves Based on J. Beck's Methods

Both theorems mentioned above provide similar strategies. In certain situations, the Maker can force a win if he maximizes the value $\sum_{A \in \varkappa} 2^{-|A|}$ each move. In certain other situations, the Breaker can force a win if he minimizes the same value each move [4].

Moreover, even when the theorems cannot prove that these strategies are winning, they still can provide helpful results. Thus, another improvement to our brute force algorithm is to rank all the possible moves by their resulting values $\sum_{A \in \varkappa} 2^{-|A|}$ either in increasing or decreasing order depending on whose turn it is, and to calculate each move in order based on these rankings. This strategy speeds up the algorithm, because if we find out a player can force a win by making a specific move, then there is no need to calculate any other moves.

Interestingly, it has been proven that for the original game, if the Breaker always makes the move that is best according to this strategy, then the Maker can force a win for Snaky[3]. However, this does not affect the results we get as we can always consider multiple moves before coming to a conclusion.

6.4 Depth Pruning

Another simple but very effective way we can speed up our program is, after reaching a certain depth in our calculations, if we still have not reached a definite result using our other methods, we merely stop looking. We mark the result of this state as ambiguous. Then, if any of the players can force a win without reaching that state, it will not matter that we do not know the result for that state. Note that if the maximum depth we selected is not large enough, we may not come up with a definite result. To combat this, we can repeat the process with an increased maximum depth, and we only need to check states that were previously marked as ambiguous.

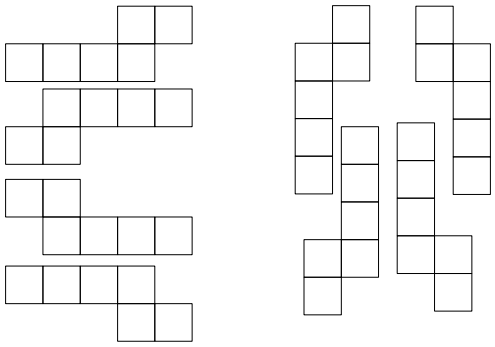
7 Analyzing Snaky

In the original game, the polyomino called Snaky has been analyzed extensively. In our version of the game, each rotation and reflection is considered a unique polyomino. Therefore, we can mimic the original game for Snaky by playing the game with a set consisting of all the rotations and reflections of Snaky, or we can use only a subset of these reflections and rotations to create unique situations to analyze. In this section, we will analyze these subsets of rotations and reflections and see if we can find any helpful results.

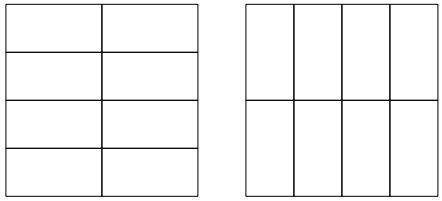
7.1 Domino Defense for Subsets of Snaky

It has been previously proven that there is no pairing strategy (and therefore no domino paving strategy) for Snaky[5]. However, this may not hold true for subsets of rotations and reflections of snaky. Using the program mentioned in section 4, for all the subsets of rotations and reflections of Snaky we tried to find a domino pattern of size up to 16 by 16 that would provide a successful strategy for the Breaker.

We can categorize each of the 8 rotations and reflections of Snaky as shown.



If all the polyominoes in our set are selected only from the vertical or the horizontal group, there is a very simple domino defense strategy.



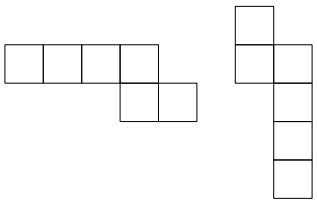
The domino paving strategies for horizontal and vertical shapes, respectively.

However, if the set contains at least one polyomino from both the vertical and the horizontal group, there will not be a domino pairing strategy, at least for a size up to 16 by 16. This is a very surprising result, as every other polyomino in the original game was proven to be losing using this strategy[1], but it does not apply to most subsets of Snaky. In the future, if we want to solve Snaky, we might want to first solve these subsets and then slowly work our way to the complete problem.

7.2 Solving Subsets of Snaky on Finite Boards

In the previous subsection, we saw that we were not able to solve most of the sets consisting of rotations and reflections of Snaky using a domino paving strategy. Fortunately, we can still analyze those sets using the program mentioned in section 6.

The method mentioned in subsection 6.1 is especially effective against sets of smaller sizes. Therefore, as an example, using our program, we were able to automatically prove that the set shown in the picture



is losing on a size 11 by 11 board. To put that size into perspective, the largest board that has been solved for Snaky is size 8 by 8 [3]. The calculations took 40 minutes on an Intel® Core™ i5-7200U CPU @ 2.50GHz with 8 GB RAM. One of the situations that arise as the program proves this set is losing on a size 11 by 11 board is shown on the figure below.

	a	b	c	d	e	f	g	h	i	j	k
11			O			k	w	Q		m	
10		U	r	O	2	g	k	w	Q	b	m
9	f	X	U	o	o	g	Y	h	l	l	b
8	f	X	J	t	t	i	i	Y	h	p	y
7	C	N	N	J	1	S	B	W	c	p	y
6	v	I	I	r	j	z	S	G	W	M	c
5	q	{	q	Z	K	z	A	A	G	u	M
4	d	C	T	n	K	R	Z	B	E	u	
3		x	x	n	H	H	T	R	a	E	
2	P	v	{	e	L	j	F	s	D	a	
1		d		P	L	e	F	s	D	V	V

After the Maker marks e7, Z3 proves that the Breaker can win by marking e10 and using a pairing defense. In the picture, two cells are in a pair if they have the same case sensitive letter (or the symbol {) written over them.

8 Further Research

Given enough time and a strong computer, most of the sets of polyominoes (at least for up to a certain size) could be analyzed and proven to be losing using the domino paving finder program. That sort of classification would make the infinite number of possibilities more manageable. During our limited testing (we only tested for a limited number of pattern sizes) we could not find a solution for only 193 of the 5915 sets of size two of polyominoes of size less than 7. (We must note that these numbers contain only the polyominoes that weren't winning by themselves and that the sets considered were all pairwise-distinct.)

Doing that kind of general analysis with our program that solves the game for finite boards is much less feasible as it is considerably slower for more complex cases. However, it could be used to analyze Snaky more deeply(and the subsets of its rotations and reflections). A more powerful computer with more time could easily improve upon the solutions we found. Moreover, the algorithm itself could possibly be improved given that the problems the SMT solver must solve to find pairing strategies are similar to each other.

We also showed that, even if we restrict the reflections and rotations of snaky, the resulting problem is not at all trivial. So, in order to solve Snaky in the future, we might first want to solve these restricted versions of Snaky. Alternatively, just analyzing them and proving other things about them (for example, if they are paving winners or losers) could be useful as well.

Finally, the improvement that would possibly be the most helpful to our research (but arguably be the most difficult) is to develop a new way to effectively classify sets of polyominoes as winning. Currently, the only way to prove a set is winning comes from the fact that if a set is winning on a finite board, it is also winning on an infinite board. However, checking for finite boards is time-consuming even with our optimized algorithm, and some sets could be winning only on boards that are larger than a large size far beyond the scope of our algorithm. Therefore, unless more methods are discovered, a portion of this game will always remain a mystery.

9 Conclusion

The mathematical game put forward in this paper presents the mathematical world with an infinite number of unique problems. The sheer number of different cases makes it impractical to analyze each one individually, but due to the similar nature of each case, a practical solution is possible. Automated algorithms can be used to find strategies and analyze the game. In our paper, we devised two such algorithms. One of them is able to find domino paving strategies, and the other one is a brute force algorithm with clever optimizations that can solve the game for finite boards. The efficiency of these algorithms are satisfactory, and hopefully, in the future, our understanding of the game can be improved by using them.

Additionally, it turned out that our research was also valuable in providing greater insight into the only unsolved case in the original game created in 1977. We showed that only a small number of the subsets of rotations and reflections of Snaky could be defended against with a domino paving pattern strategy of size up to 16 by 16. The remaining subsets all pose interesting problems and solving them could help solve Snaky as well. We also solved some of these subsets for finite boards up to size 11 by 11.

Hopefully, in this paper, we have demonstrated that satisfiability modulo theory solvers like Z3 can be used to automatically devise strategies too complicated for brute force algorithms and too tedious for humans. We hope that these algorithms and the methods used to create them will be useful not only in the context of this game, but for all similar mathematical problems.

References

[1] Martin Gardner. Mathematical games. *Scientific American*, 240(4):18–36, 1979.

[2] Nándor Sieben. Polyominoes with minimum site-perimeter and full set achievement games. *European Journal of Combinatorics*, 29(1):108–117, 2008.

[3] Immanuel Halupczok and Jan-Christoph Schlage-Puchta. Achieving snaky. *Integers [electronic only]*, 1, 01 2007.

[4] J. Beck. Remarks on positional games. i. *Acta Mathematica Academiae Scientiarum Hungaricae*, 40:65–71, 1982.

[5] András Csernenszky, Ryan Martin, and András Pluhár. On the complexity of chooser-picker positional games. *Integers*, 2012:427–444, 11 2011.