Universiteit Leiden
Opleiding Informatica

Theoretical Properties of 2048

Name: Mathé Zeegers

Date: 30/08/2016

1st supervisor: Dr. Walter Kosters
2nd supervisor: Dr. Hendrik Jan Hoogeboom

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

2048 is a sliding-puzzle game in which tiles containing powers of two are merged by sliding these on a grid and combining them to obtain higher-valued tiles. The goal of the game is to obtain the tile 2048 using the tiles of values 2 and 4 that are dropped on the board by the computer. We present a generalization of the game for which we will state and prove various properties of reachable boards. Furthermore, we will closely examine the one-dimensional case where the number of rows is equal to one.

By using theorems we present an algorithm that determines whether a given configuration is reachable during a game in which it is not mandatory to change the configuration of the board, and relate this to the game where changing the configuration is compulsory. In addition, the dynamics of the game are investigated when more different tiles are dropped on the board. Furthermore, games in which the player loses as fast as possible are examined and expressions or upper bounds for the number of moves to achieve this are derived. Finally, the possibilities of solving the game-theoretic value of the initial board in 2048 are discussed. It is shown, using a computer with 1.5TB of memory, that when the player plays optimally, a tile of value 128 can always be obtained.

Contents

1 Introduction

The game 2048 [1] has been very popular recently. It was developed by Gabriele Cirulli during a single weekend and released on March 9 in 2014. The game is based on 1024 by Veewo Studio and similar to Threes! by Asher Vollmer. 2048 is a sliding-puzzle game in which tiles that spawn on a four-by-four board have to be merged to obtain higher-valued tiles. The objective of the game is to obtain the tile 2048. An example of a reachable configuration is given as a screenshot in Figure 1.

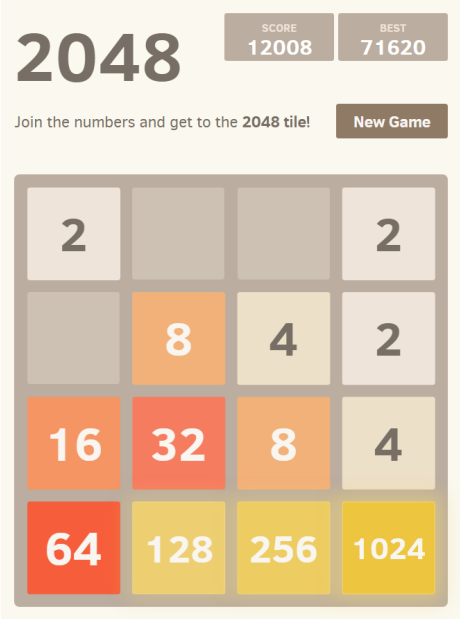

Figure 1: Reachable configuration of the game 2048, as it appears in [1].

During each turn a 2-valued tile or a 4-valued tile is dropped randomly on the board. Tile 2 is dropped with probability 0.9 and tile 4 with probability 0.1. The player then swipes the tiles in one of the four directions (up, down, left or right) and merges pairs of equally-valued tiles in the process. For example, two 2-valued tiles lining up to each other become a 4-valued tile by applying the correct move. A move is only allowed when the configuration on the board changes. If no more moves are available, the player loses. This is the case when the board is full and no tile can be merged. An example of a configuration and the possible moves is shown in Figure 2. This also shows the procedure of merging. In each row or column each tile can merge at most one time per move with another equally-valued tile. The order of the new tiles in a row or column that appear by merging depends on the direction of the move that is carried out.

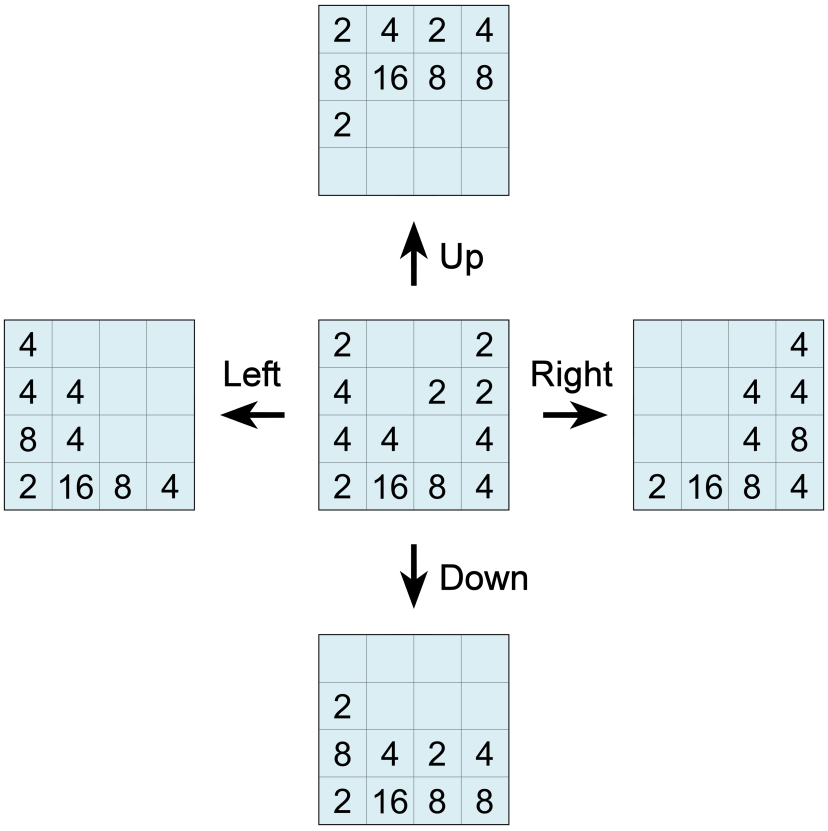

Figure 2: A random configuration with the resulting boards of the four possible moves of the player. The moves are indicated by the arrows.

The game also features a scoring mechanism. When two tiles with value 2^i merge to 2^{i+1}, the score increases with 2^{i+1} (with $i \geq 1$). While most research of 2048 is targeted towards obtaining the highest possible score, this will not be the main focus of this thesis.

Given the rules of the game, an important question is whether it is always possible to reach tile 2048 or not. Many of the results in this thesis are driven by this question.

This thesis is part of the master project for the master programme of Computer Science at the Leiden Institute of Advanced Computer Science (LIACS) of Leiden University, and is supervised by Walter Kosters and Hendrik Jan Hoogeboom.

In this thesis we will closely examine the theoretical properties of this game. In Section 2 previous work on 2048 is reviewed. We will modify and generalize the game to have a better understanding of the mechanics. To this end, the modifications to the game are described in Section 3 as well as programs and the computational resources that are used. Section 3.3 states the various problems

that will be analyzed in this project in the light of the modifications. Then, in Section 4 general properties of the boards will be stated and proved. Section 5 will be targeted to the one-dimensional version of the game. We will state and prove additional properties of the game and use these for an algorithm that checks whether a given board is reachable in the game. In Section 6 the influence of the maximum tile that can be dropped on the board (which is 4 in the original 2048 game) is investigated by employing theoretic methods as well as computational methods. Section 7 will be concerned with keeping the game as short as possible in terms of the number of moves. This will also be examined by theoretical and computational methods. In Section 8 the methods for solving 2048 are discussed. Section 9 collects all conclusions, and suggestions for future research are given, as well as predictions in which case 2048 is solvable.

2 Related work

Since the release of the game many variations have been developed [2, 3]. These variations range from simple graphical overhauls or small gameplay changes to versions that seriously change the rules to add interesting spins to the game, such as larger boards, multiple dimensions and different dynamics of the tiles. A selection of variants is given in Figure 3. Apart from this the original game was also picked up by researchers. The most important portion of the research focuses on heuristics for winning the game as often as possible. The earliest efforts included an agent by Chowdhury and Dhamodaran that applied a depth-limited expectimax algorithm to the game [4] and achieved a winning ratio of nearly 90%. Rodgers and Levine made a comparison between Monte Carlo Tree Search (MCTS) and Averaged Depth Limited Search (ADLS) [5], although the aim here (and in most subsequent research) was to maximize the score rather than win the game. It turned out that ADLS produces higher scores than MCTS. Meanwhile, Temporal Difference (TD) learning was applied to 2048 by Szubert and Jakowski [6]. Using N-tuple networks this method achieved a winning ratio of about 97%. Here an N-tuple contains values of N predetermined positions on the board and a look-up table containing weights for each observable sequence. Aiming to increase the rate at which large tiles are reached, Wu et al. [7] extended this approach by using Multi-Stage Temporal Difference (MSTD) learning to adapt the agent to several stages of the game. Their MSTD-program reached the 32768-tile with a rate of 31.75%, a significant improvement compared to the rate of 0% using TD. More recently, Gui et al. [8] adapted deep neural networks with reinforcement learning to 2048, but the networks performed worse than an agent that only uses reinforcement learning. Oka and Matsuzaki [9] extended the previous TD approaches by investigating the usefulness of various N-tuples. It appeared from their research that the design of the tuples is highly influential on the performance of the agent. The authors used 7-tuples to achieve the maximum score known as of yet (504660).

However, only little research has been devoted to the theoretical properties of the game, almost exclusively in the field of computational complexity. The earliest effort by Mehta [15] argued that the game is PSPACE-complete using a reduction from Nondeterministic Constraint Logic (NCL). However, it was debated by Abdelkader, Acharya and Dasler [16, 17] not only that the proof just showed that 2048 is NP-complete, but also that there are a few problems with the reduction (the blocks do not always move according to the game's rules, for example). In their paper, it is shown that 2048 is NP-hard using a reduction from 3-SAT and that, using the argument that 2048 is in NP by Christopher Chen [18], 2048 is NP-complete. However, in their version of the game it is assumed that all tiles in the game are present on the initial board and that during the game no new tiles are dropped. Later on, the problem of NP-completeness was picked up by Langerman and Uno [19]. They showed that the original game is NP-complete as well. Another proof by Gobbert [20] showed that 2048 is also NP-complete for all rectangular boards, but in this case only for obtaining tiles with certain values.

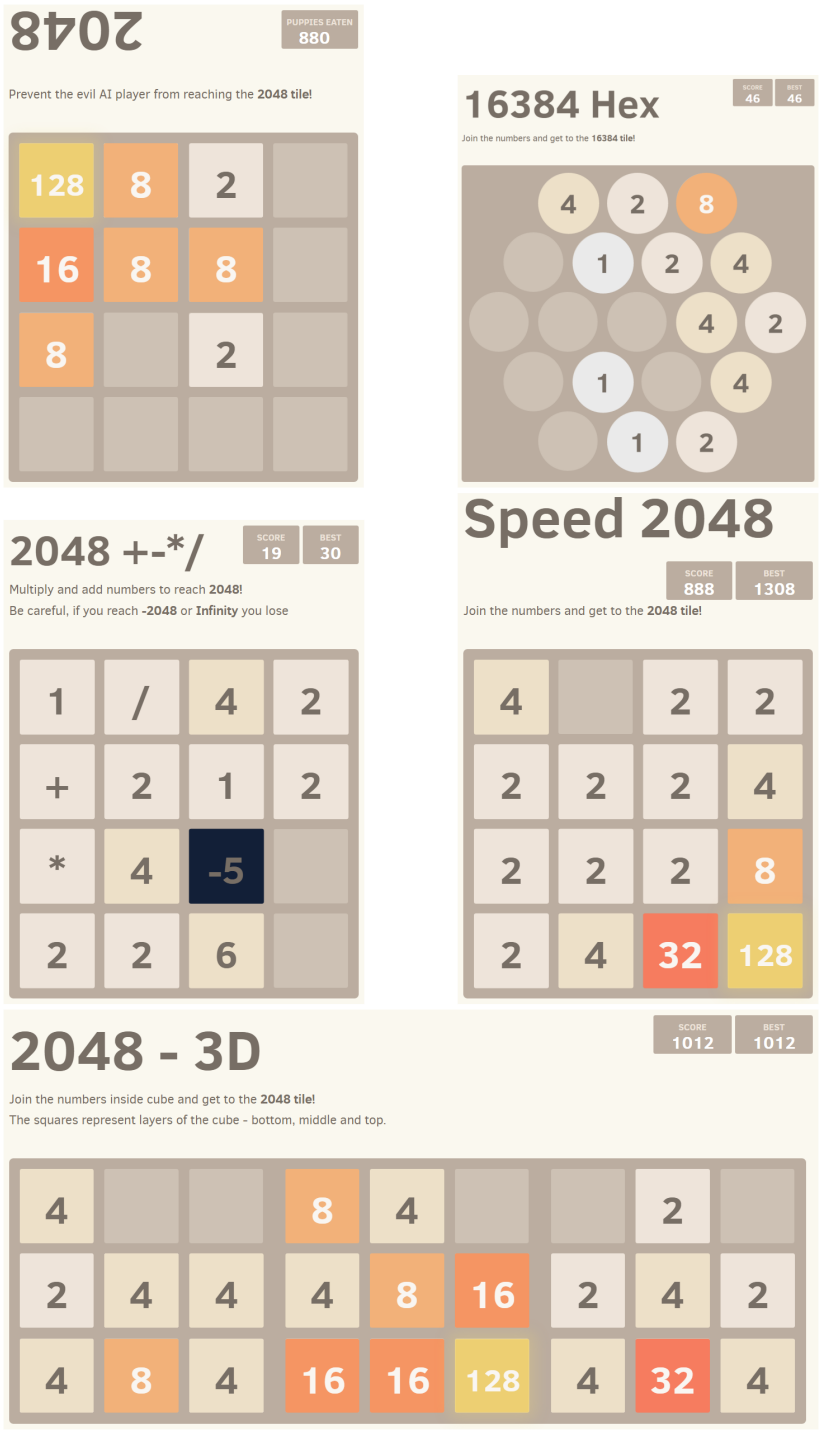

Figure 3: A small selection of the large amount of variations on 2048 that have appeared on the internet. Upper left: A variant with the roles of player and computer reversed. Upper right: A variant that is played on a small hexagonal board with six possible moves. Middle left: A variant where 2048 must be reached using combinations of numbers and operators. Middle right: A variant where 2-valued tiles are dropped after each move and after each elapsed second. Below: A 3-D variant of 2048 on a $3 \times 3 \times 3$ board. Images taken from [10, 11, 12, 13, 14] respectively.

In [21] the maximum-valued tile is determined for various boards and setups. In this version the computer is treated as a second player instead of a randomized tile dropper. The largest two-dimensional board that has been solved has 14 squares. It is strongly suggested that in case of perfect play by the computer the original 4×4 game cannot be won. We will revisit and extend some of the material covered in this paper.

It is also worth noting that several competitions regarding 2048 have been organized [22, 23]. In the Taiwan 2048-bot contest in 2014, the ranking was based on the maximum score, the average score, the maximum tile and the rate of high-valued tiles over 100 games. The top-three winning programs all used expectimax search and the second placed program was based on TD learning.

3 Methods and problem statement

In this section the modified game is presented and the conventional definitions and notations in this thesis are explained. In addition, the programs that are developed to obtain the results in this work will be briefly discussed.

3.1 Modifications and conventions

This thesis will use several simple modifications from the original game. First of all, we transform the game into a two-player game, where the first player drops tiles on the board and the second player swipes the tiles in one of the four directions. We will call the first player 'Dropper' and the second player 'Slider'. Dropper is the player that always starts on a board that is initially empty.
We will not restrict the board dimensions to 4×4. Instead, we denote the sizes by $m, n \in \mathbb{N}_+$. Here, m is the length of the vertical axis and n is the length of the horizontal axis. We take these parameters such that $m \leq n$, since the other cases lead to similar boards (when rotated).
The representation of tiles in the original game is also changed. Given a tile $x \geq 0$ in the original game, the following function V maps it to their new values:

$$V(x) = \begin{cases} 0 & \text{if } x = 0 \\ \log_2(x) & \text{if } x > 0 \end{cases}$$

In this case two i-valued tiles merge to one $(i+1)$-valued tile.
Dropper is allowed to place new tiles on an empty position (i.e. with value 0) on the board. The parameter $D_{\max}$ indicates the maximum value of tiles he is allowed drop.
In the original game the value of the objective tile is fixed. In the modified version we use parameter O to indicate the objective value. We assume $O > D_{\max}$. Unless stated otherwise, the objective tile is $O = m \cdot n + D_{\max}$. This is the smallest value that is certainly not obtainable, which will be shown in Theorem 6. We write O and leave out the dependency on the parameters.
The game ends when either the objective tile appears on the board or Slider cannot swipe in any direction changing the board. In the former case Slider wins and in the latter case Dropper wins.
As for notation, we use $C \in \mathbb{N}^{m \times n}$ to indicate a configuration during the game. We also leave out the dependency on the parameters in this notation. The value of a tile in row i and column j of the board is indicated by $C_{i,j}$, with $0 \leq i < m$ and $0 \leq j < n$. The board C^0 is the empty board (i.e. every site has value 0). For boards with $m = 1$ we will use a lowercase c (additional notation for this case will be explained in Section 5.1). Furthermore, we will use the terms 'board' and 'configuration' interchangeably. The same goes for the terms 'tile' and 'value'.
We will call a board C 'reachable' by Dropper when it is Dropper's turn and there exists a sequence of moves by both players that leads to C from C^0. The same holds for Slider.
We also introduce the notion of 'passing'. In the original 2048 game a valid move

by the player is when the configuration changes. In this paper we also consider the case where this is not mandatory. We refer to this version as a game with passing, or, equivalently, a passing game. A pass is a move where the player does not change the configuration. In this case, Slider only loses when the board is completely filled with tiles and he is not able to apply a non-passing move. Unless noted otherwise, we consider the game without passing by default.
As we will look at different properties of the game, the problems outlined in Section 3.3 will be approached in different ways in each section of the remainder of this paper and calls for slight modifications. In each section, we will explicitly describe what modifications are made on top of the modifications that have been described in this section.

3.2 Implementation

For the problems that will be presented in Section 3.3 a handful of programs have been developed. These programs are written in C++.
The first important program is an improvement of the one that is described in [21]. This program computes which player will win in case of optimal play for any specified setup. To save multiple evaluations of boards a hash table is used that stores the game-theoretic value of boards. Optimizations of the previous program include creating fewer new boards, only hashing boards that are reachable by Dropper and cramming five game-theoretic values of boards (this includes attainability of the board) into a byte instead of one value. More details of the program are given in [21] and more notes on optimizations and techniques will be given in Section 8. This program will be used in Sections 6, 7 and 8.
The second important program identifies boards that are reachable in a one-dimensional (i.e. $m = 1$) passing game. The core part of the algorithm will be explained in Section 5.
All computations in this thesis are done on a computer having a processor speed of 2.4GHz. The computer has a total of 1.5TB of RAM which is used for the hash table. This means that no more than $7.5 \cdot 10^{12}$ game-theoretic values can be stored in the hash table.

3.3 Problem statement

We aim to answer the following questions:

1. Given $m, n \in \mathbb{N}_+$ and $D_{\max} \in \mathbb{N}_+$, what are the properties of reachable boards?

2. Given $m, n \in \mathbb{N}_+$ and $D_{\max} \in \mathbb{N}_+$, is there an (efficient) algorithm that checks whether a given configuration is reachable in a passing game? Can this algorithm be used to check whether a given configuration is reachable in a non-passing game?

3. Given $m, n \in \mathbb{N}_+$, is there a value of $D_{\max}$ independent of n for which Dropper can play in a way that no two tiles can be merged by Slider during the entire game? What is the highest obtainable tiles for Slider, given a $D_{\max}$ value?

4. Given $m, n \in \mathbb{N}_+$ and $D_{\max} \in \mathbb{N}_+$, what is the fastest way to obtain a board where Dropper wins (where Dropper and Slider cooperate)?

5. Given $m = 3, n = 5$ and $D_{\max} = 2$, can we give a time and space efficient method to solve the question whether Slider or Dropper wins the modified game in case of optimal play? What can we say about $m = 4, n = 4$ and $D_{\max} = 2$?

These questions will be covered in Sections 4 through 8, with a separate section for each question.

4 General properties

In this section we will state and prove some useful facts of the modified game that will be helpful in later sections.

Theorem 1. *Let $m, n \in \mathbb{N}_+$ and let $D_{\max} \in \mathbb{N}_+$. Every reachable configuration C in the modified game has a corner point ($C_{0,0}, C_{0,n-1}, C_{m-1,0}$ or $C_{m-1,n-1}$) that has value at most $D_{\max}$.*

Proof. We prove this statement with induction to the number of moves N of Dropper and Slider. The statement is obviously true for $N = 0$ as board C^0 is still empty.
Assume that statement holds after $N' \geq 0$ moves. Without loss of generality we may assume corner point $C_{0,0}$ contains a value between 0 and $D_{\max}$. Now there are two cases:

- Assume it is Dropper's turn. When corner $C_{0,0}$ is empty he may drop a value between 1 and $D_{\max}$ there. Then after $N' + 1$ moves this corner contains a value between 0 and $D_{\max}$. When the corner contains a value between 1 and $D_{\max}$ this value stays the same when Dropper places his tile (elsewhere) on board C. In either case we see that the statement holds after $N' + 1$ moves.

- Assume it is Slider's turn. Let C' be the board after Slider applies his move. There are two slightly different cases:

  - Assume $C_{0,0} = 0$. When Slider swipes to the right or downwards then we still have $C'_{0,0} = 0$. When Slider swipes upwards then we have $C_{m-1,0} = 0$ and when Slider swipes to the left then we have $C_{0,n-1} = 0$. In all cases we have an empty corner point, so after $N'+1$ moves the statement holds.
  - Assume $0 < C_{0,0} \leq D_{\max}$. When Slider swipes to the right or downwards then $C'_{0,0} = C_{0,0}$ or $C'_{0,0} = 0$ holds. When Slider swipes upwards $C_{0,0}$ stays the same or merges. When it merges, then $C_{m-1,0} = 0$ and the statement holds after $N' + 1$ moves (but now the corner point is $C'_{m-1,0}$). When Slider swipes to the left and $C_{0,0}$ merges, then $C'_{0,n-1} = 0$ (and thus the desired corner point is $C'_{0,n-1}$). In all cases, we have a corner point in C' that has value between 0 and $D_{\max}$.

Thus after $N' + 1$ moves the statement holds. □

Theorem 2. *Let $m, n \in \mathbb{N}_+$. The following four statements hold:*

- *A move to the right is allowed if and only if the board contains a horizontal equally-valued adjacent pair or an empty position with a non-empty value directly left of it. In other words, it is allowed if and only if there exist i, j with $j < n - 1$, $C_{i,j} > 0$ and either $C_{i,j+1} = C_{i,j}$ or $C_{i,j+1} = 0$.*

- *A move to the left is allowed if and only if the board contains a horizontal equally-valued adjacent pair or an empty position with a non-empty value directly right of it. In other words, it is allowed if and only if there exist* i, j *with* $j < n-1$, $C_{i,j+1} > 0$ *and either* $C_{i,j} = C_{i,j+1}$ *or* $C_{i,j} = 0$.
- *A move upwards is allowed if and only if the board contains a vertical equally-valued adjacent pair or an empty position with a non-empty value directly below. In other words, it is allowed if and only if there exist* i, j *with* $i < m-1$, $C_{i+1,j} > 0$ *and either* $C_{i,j} = C_{i+1,j}$ *or* $C_{i,j} = 0$.
- *A move downwards is allowed if and only if the board contains a vertical equally-valued adjacent pair or an empty position with a non-empty value directly above it. In other words, it is allowed if and only if there exist* i, j *with* $i < m-1$, $C_{i,j} > 0$ *and either* $C_{i+1,j} = C_{i,j}$ *or* $C_{i+1,j} = 0$.

Proof. We will prove the first statement as the proofs of the other statements are analogous.
Assume that the board contains a horizontal equally-valued pair, so there are $i \geq 0$, $j < n-1$ with $C_{i,j} = C_{i,j+1}$, that will merge during a move to the right. When applying this move these two tiles merge into a tile with value $C_{i,j} + 1$. Thus, the number of tiles with value larger than $C_{i,j}$ has increased and the configuration on the board has changed.
Assume that the board contains an empty position, say $C_{i,j} = 0$ with $j > 0$ with a non-empty value directly left of it, say $C_{i,j-1} = y$ with $j, y > 0$. Assume that there are $k > 0$ non-empty tiles left of the $C_{i,j}$ tile. Applying a move to the right means that all k non-empty tiles move at least one position to the right. This means that after the move there are at most $k-1$ tiles left of the $C_{i,j}$ tile. This implies that the configuration has changed.
In both cases the configuration has changed and the move rightwards is therefore valid.
Conversely, assume each row does not contain an equally-valued adjacent pair or an empty position with a non-empty value directly left of it. Let $C_{i,j}$ be a non-zero tile on the board. Then because all tiles $C_{i,j'}$ with $j' > j$ are non-zero and because $C_{i,j+1} \neq C_{i,j}$ if $j < n$, it follows that $C_{i,j}$ will keep the same value after the move. Thus, each non-zero tile stays at the same place and its value will not change. Therefore, the configuration on the board will not change and the move is not allowed. □

From Theorem 2 we immediately have a necessary and sufficient condition for when the game is over for Slider.

Corollary 3. *Slider loses if and only if during his turn there are no empty positions or equally-valued adjacent pairs. In other words, Slider loses if an only if there are no* i, j *with* $C_{i,j} = 0$ *or* $C_{i,j} = C_{i,j+1}$ *if* $j < n-1$ *or* $C_{i,j} = C_{i+1,j}$ *if* $i < m-1$. □

Theorem 4. *Let* $m, n \in \mathbb{N}_+$. *Let* C *be a configuration that is reachable by Dropper. Then* C *has a boundary point (*$C_{i,j}$ *with either* $i = 0, i = m-1, j = 0$ *or* $j = n-1$*) with value* 0.

Proof. We will prove this statement with induction to the number of moves N of Dropper. Again, the statement is true for $N = 0$ as board C^0 is completely empty. Assume that the statement holds after $N' \geq 0$ moves. Let C be any configuration reachable by Dropper in exactly N' moves. By the induction hypothesis, configuration C has at least one zero-valued boundary point, say $C_{i,j}$. When Dropper places a new tile on a zero-valued non-boundary point then after applying any move by Slider either $C_{0,j}$, $C_{m-1,j}$, $C_{i,0}$ or $C_{i,n-1}$ is zero-valued. Therefore, assume that Dropper places a new tile on a boundary point. Now there are three situations:

- There are no empty tiles or equally-valued adjacent pairs. Then by Corollary 3 Slider loses and the game is over. So the statement holds.
- There are empty positions. Then after a move there is an empty boundary point, since after each move the number of tiles has not increased and all non-zero tiles are pushed at one side and all empty positions at the other side of the rows or columns (depending on the directions of the move).
- There are equally-valued adjacent pairs and no empty tiles. Then the number of tiles decreases and a new empty tile appears on the board when the move is applied in the direction parallel to the pair. By the nature of the move, this must be at a boundary point.

In all cases, there is a new boundary point after $N' + 1$ moves by Dropper and Slider. □

Theorem 5. *Let $m, n \in \mathbb{N}_+$. A configuration that is reachable by Dropper does not contain an adjacent pair of an empty position and a non-zero tile in all four directions at the same time.*
A configuration that is reachable by Slider contains these pairs not more than one time in at least one of the four directions.

Proof. Assume that a configuration reachable by Dropper contains an adjacent pair of an empty position and a non-zero tile in all four directions at the same time. This configuration cannot be C^0, so at least one move by Slider has been executed. Assume that the previous move was a swipe to the right. This contradicts the fact that we have an empty position and a non-zero tile left of it, since a move to the right would have moved the non-zero tile at least one position to the right. Swipes into other directions would lead to similar contradictions. The statement follows.
Since Dropper can place a new tile at any empty square on the board, the second statement is a direct consequence of the first statement. □

Theorem 6. *Let $m, n \in \mathbb{N}_+$ and let $D_{\max} \in \mathbb{N}_+$. The highest obtainable tile is not more than $mn + D_{\max} - 1$.*

Proof. Assume that the highest obtainable tile is $mn + D_{\max}$ and let C be any configuration containing tile $mn + D_{\max}$. Then there must exist a configuration $C_{mn+D_{\max}-1}$ preceding C with at least two tiles with value $mn + D_{\max} - 1$. This

implies that there must exist a configuration $C_{mn+D_{\max}-2}$ preceding $C_{mn+D_{\max}-1}$ with either two tiles with value $mn+D_{\max}-2$ and one tile of value $mn+D_{\max}-1$ or four tiles with values $mn + D_{\max} - 2$. This means that $C_{mn+D_{\max}-2}$ has at least three tiles, of which at least two have value $mn+D_{\max}-2$. In the same way there must exist a configuration $C_{mn+D_{\max}-3}$ preceding $C_{mn+D_{\max}-2}$ with at least four tiles, of which at least two have value $mn + D_{\max} - 3$. Continuing this reasoning there must exist a preceding configuration C_1 with at least $mn+D_{\max}$ tiles, containing at least two tiles with value 1. However, this is in contradiction with the fact the there are only mn squares. Thus, the highest obtainable tile is not more than $mn + D_{\max} - 1$. □

There are boards where value $mn + D_{\max} - 1$ is attainable. This will be shown for $D_{\max} = 1$ later on in Lemma 12, and the proof is extensible to larger values of $D_{\max}$.

5 One-dimensional case

In this section we will examine the one-dimensional modified 2048 game, i.e. the cases with $m = 1$. We are particularly interested which boards are possible in the game in case $D_{\max} = 1$. If we have an efficient way of generating all boards that can occur in the game, this would be helpful to determine the game-theoretic value of the modified game (see [21]).

5.1 Theoretical analysis

The modified game in the case $m = 1$ is slightly different from the case $m > 2$. Dropper is still to drop a 1-valued tile on an empty site of the board, but Slider is only allowed to swipe to the right or the left. Thus, Slider wins when a specified objective tile is obtained by repeatedly merging equally-valued tiles to the left or to the right to get higher-valued tiles.
For this game it is not easy to come up with an optimal strategy for Slider to win the game. In the one-dimensional case this is not the case for Dropper. It is best for him to place his tile next to another tile. In every stage of the game this move is uniquely defined and therefore the move by Dropper is deterministic. So the problem that remains is to find the best way for Slider to play the game.

Given the board length $n \in \mathbb{N}_+$, a board configuration is represented by a string of length n. To keep those strings readable we introduce the alphabet $\Sigma = \{.\} \cup \{1, \ldots, 9\} \cup \{a, \ldots, z\}$. The dot represents an empty tile (with value 0), $1, \ldots, 9$ represent tiles with values 1 to 9 and $a, \ldots, z$ are one-symbol substitutes for the tiles with values 10 to 35. In this thesis we hardly need values high values of tiles, and certainly not larger than 35, so we do not define symbols for them. Let Σ_n be the alphabet Σ restricted to the values 0 to n (we will prove later in Lemma 12 that n is the largest value that can occur on the board). Potential configurations are now elements of Σ_n^*, which is the set of strings $c \in \Sigma^*$ with $|c| = n$. The words "string", "configuration" and "board" are used interchangeably.

To define appropriate sets to work with for solving our problem we first need to distinguish the following two cases: the game where it is allowed to swipe to the right without merging, which is called "passing" in Section 3.1) and the game where this is not allowed. The convention here is that we assume that when a swipe to the left takes place the board is mirrored, so that all tiles stay at the right side of the board. This will be formalized shortly. Note that the first version of the game possibly allows more configurations than the second version.

We define the following three operations. A dropping operation d on a configuration c replaces the rightmost dot in c by a 1. A flipping operation f on c mirrors the smallest substring of c that contains all nondot-symbols. A merging operation g is defined analogous to the swipe to the right in the original 2048 game. For example, applying this operation on configuration $c = .12233$ gives $g(c) = ...134$.

Using the functions defined above we can define functions for swiping left and right consistently with the game. A swipe to the right is the composition $r \stackrel{\text{def}}{=} d \circ g$ and a swipe to the left is the composition $\ell \stackrel{\text{def}}{=} d \circ g \circ f$.

Let Σ_n^{pass} be the set that contains precisely all possible configurations in the modified one-dimensional 2048 game with board length n where passing is allowed and where it is Slider's turn. This set can be constructed as follows. First, the string c^0 containing only dots except for the last symbol being a 1 is an element of Σ_n^{pass}. A configuration c is in the set if it can be obtained from c^0 by repeatedly applying the function ℓ or r to it in any order.
Let Σ_n^{nopass} be the set that contains precisely all possible configurations in the modified one-dimensional 2048 game with board length n where passing is *not* allowed and where it is Slider's turn. The construction of this set is analogous to that of the set Σ_n^{pass}, except that it is not allowed to apply a move to the right on a configuration $c \in \Sigma_n^{\text{nopass}}$ when $r(c) = d(c)$ (or equivalently: $g(c) = c$). Note that $\Sigma_n^{\text{nopass}} \subseteq \Sigma_n^{\text{pass}}$ holds.

Before we proceed, we give examples that make the above definitions and statements clear.

Example 1. Take for example $n = 5$. A string $c = 124.2 \in \Sigma_n^*$ is a potential string. We can apply the defined functions on this configuration to obtain:

- $d(c) = 12412$
- $f(c) = 2.421$
- $g(c) = .1242$
- $r(c) = 11242$
- $\ell(c) = 12421$

However, it does not make much sense to consider this string c in the context of the one-dimensional modified 2048 game, as it is impossible to obtain this string from a sequence of moves ℓ and r applied to c^0. Therefore $c \in \Sigma_n^* \setminus \Sigma_n^{\text{pass}}$.
Consider the string $c' = ..112 \in \Sigma_n^*$. This string is an element of Σ_n^{pass} since there exists a series of moves applied to c^0 to obtain this c', namely $r(r(\ell(c^0)))$ or $r(r(r(c^0)))$. Note that there are in total four different possible sequences of moves to obtain c'. For all these sequences we have $c'' = ...12$ after applying the first two moves. Now we have that $r(c'') = ..112 = d(c'') = c'$. Therefore, $c' \notin \Sigma_n^{\text{nopass}}$.

The objective now is to give an algorithm that determines for a given configuration $c \in \Sigma_n^*$ whether it is an obtainable configuration in the game or not. In other words it should be able to determine if $c \in \Sigma_n^{\text{pass}}$ in the case where passing is allowed and $c \in \Sigma_n^{\text{nopass}}$ in the case where passing is not allowed. The following section focuses on the game with passing.

When given a configuration c, the algorithm first performs a set of checks on c. In this way, the algorithm can quickly reject many configurations that are not in Σ_n^{pass}. We first give lemmas and theorems that are used to show the correctness of these checks. The first lemmas are relatively simple but the lemmas and theorems get more complicated to prove as we move on. Note that most lemmas also automatically hold for Σ_n^{nopass}.

Lemma 7. *Let $c \in \Sigma_n^{\text{pass}}$. All dots are at the left side of c and all non-dot symbols are at the right side. The first non-dot symbol of c is $1 \in \Sigma_n$.*

Proof. This follows from the construction of Σ_n^{pass}. We prove this with induction to the number of moves k applied to c^0.
For $k = 0$ we have string c^0 which has only one non-dot symbol which is a 1.
Assume that the statement holds after k moves, and that we obtain c' after k moves. We can apply the $(k+1)$th move and obtain a new configuration by sliding to the right or to the left. A move to the right gives $r(c) = d(g(c))$. Operation g ensures that all non-dot symbols are at the end of the string and d puts a 1 in front of them. A move to the left gives $\ell(c) = d(g(f(c)))$. Again, applying g after f ensures that the non-dot symbols are at the end of the string and d puts a 1 in front of them. In both cases we have that the first non-dot symbol of the new string is a 1. □

In the following, three important properties of configurations in Σ_n^{pass} will be stated and proved. Before we do this, we need two lemmas and definitions.

Lemma 8. *Let $c \in \Sigma_n^{\text{pass}}$. The last two symbols of c cannot both be 1, except when the preceding symbol is a dot.*

Proof. Note that $\ell(c^0)$ gives a configuration that ends with two ones and has no preceding symbols (for example, when $n = 4$ we get $\ell(c^0) = ..11$).
Let c be a configuration that ends with $x11$ where $x \geq 1$. Since both 1s cannot be the result of a merge and none of the two could be dropped during a move that resulted in c (since there is an x in front of them), we should have the same 11-pair in a previous configuration c'. But since any 11-pair in c' would merge to a 2 for both moves, we cannot have 11 at the end of c. Contradiction. So there are no configurations that end with $x11$ with $x > 1$. □

We will also use the following important definition.

Definition 1. A configuration $c \in \Sigma_n^{\text{pass}}$ is *bitonic* when the symbols of the string are first only increasing and from some point onwards only decreasing in lexicographical order.

Example 2.

- c^0 is bitonic
- $c = .1234$ is bitonic.

- $c = 43321$ is bitonic.
- $c = 13642$ is bitonic.
- $c = ..111$ is bitonic.
- $c = 14251$ is *not* bitonic.

Definition 2. Let $c \in \Sigma_n^{\text{pass}}$. Let c_i be the ith symbol in c. Define the following four quantities:

$$T(c) \stackrel{\text{def}}{=} \sum_{\substack{i=1 \\ c_i>0}}^{n} 2^{c_i-1}$$

$$T(c,x) \stackrel{\text{def}}{=} \sum_{\substack{i=1 \\ 0<c_i<x}}^{n} 2^{c_i-1}$$

$$T_\ell(c,x) \stackrel{\text{def}}{=} \sum_{\substack{i=1 \\ 0<c_i<x \\ i \leq \min \arg\max_j \{c_j\}}}^{n} 2^{c_i-1}$$

$$T_r(c,x) \stackrel{\text{def}}{=} \sum_{\substack{i=1 \\ 0<c_i<x \\ i \geq \max \arg\max_j \{c_j\}}}^{n} 2^{c_i-1}$$

Lemma 9. *Let $c \in \Sigma_n^{\text{pass}}$. The number of moves to reach c is equal to $T(c) - 1$.*

Proof. To create a 2 there are two 1s needed. To create a 3 we need two 2s, for which in turn four 1s are needed. In general, to create a tile of value c_i there are 2^{c_i-1} tiles of value 1 needed. This implies that for every tile c_i on board c a total of 2^{c_i-1} tiles have spawned. So for all tiles on the board $\sum_{i=1}^{n} 2^{c_i-1}$ must have been spawned, which is equal to $T(c)$. Since during every move exactly 1 tile spawns and since c^0 has one 1-tile, it follows that the number of moves to reach c is equal to $T(c) - 1$. □

Theorem 10. *Let $c \in \Sigma_n^{\text{pass}}$. The following three properties hold:*

- *Any non-dot symbol $x \in \Sigma_n$ cannot appear more than two times consecutively in c.*
- *Configuration c is bitonic.*
- *Any non-dot symbol $x \in \Sigma_n$ cannot appear more than three times in c.*

Proof. We will prove this statement with induction to the number of moves. First of all, note that all three properties hold for configuration c^0.
Now, let $N \in \mathbb{N}_{>1}$ and assume that the properties hold for all configurations

c with $T(c) - 1 \leq N$. Let c' be any configuration that can be formed from configurations with $T(c)-1 = N$. For this configuration we have $T(c')-1 = N+1$. Assume that c' contains more than two equal symbols next to each other. Then c' contains a substring xxx with $x > 1$. Let $y = x - 1$. Let $\overline{c}$ be any configuration with $T(\overline{c}) - 1 = N$ for which c' is reachable by either swipe to the right or to the left. Since each x-valued tile in c' is either x or yy in $\overline{c}$, there are at most eight possible combinations for the substring in $\overline{c}$ that forms xxx in c':

1. $yyyyyy$
2. $yyyyx$
3. $yyxyy$
4. $xyyyy$
5. $yyxx$
6. $xyyx$
7. $xxyy$
8. xxx

Note that all of the above substrings are all impossible. By the induction hypothesis, substring $1, 2, 4$ and 8 are not possible since they have three or more equally-valued consecutive non-dot symbols. Substring 6 is not possible since it is not bitonic. Substrings $1, 2, 3, 4$ are not possible since they contain y four or more times. And substrings 5 and 7 are not possible since they form $(x + 1)$ when slid to the right or to the left. So none of the substrings is possible, which shows that xxx is not possible when $x > 1$.
In the special case $x = 1$ configuration $\overline{c}$ should have contained either 111 and 11, since 1 cannot be formed by merging other tiles (the latter case is when 111 in c' is preceded by a dot). But sliding this configuration to the right or to the left produces a 2-valued tile instead of 111. So xxx with $x = 1$ is also not possible. So we reached a contradiction. This shows that c' cannot contain non-dot symbols that appear more than three times consecutively.

Assume that c' is not bitonic. Then there are x, w, z with $x < w$, $x < z$, $x \geq 1$ such that $wxxz$ or wxz is a substring in c'. For $\overline{c}$ there are several possibilities. Note that when x is formed from $(x-1)(x-1)$ (in case $x > 1$), then $\overline{c}$ is certainly not bitonic, so this is not possible. Also note that c' can only be formed either when $w = x + 1$ and w was formed from $(w - 1)(w - 1)$ in $\overline{c}$ or when $z = x + 1$ and z was formed from $(z - 1)(z - 1)$ in $\overline{c}$. But then $\overline{c}$ contained the substring xxx, which is not possible by the induction hypothesis. Contradiction. So c' is bitonic.

Assume that c' contains four equally-valued non-dot symbols. Since c' is bitonic and cannot contain three equally-valued consecutive non-dot symbols, c' must

contain the substring $xxSxx$ (where S represents a non-empty substring that only contains values higher than x).
When $x = 1$, then the pair xx at the right of the substring S forms the end of c', since c' is bitonic and all dots are at the left of the string. By Lemma 8 this is not possible. So this gives a contradiction.
When $x > 2$, then the left pair must have been formed by $(x-1)(x-1)x$ in $\overline{c}$ and the right pair by $x(x-1)(x-1)$ (this follows from bitonicity and from the fact that xx merges to $(x+1)$. But then $\overline{c}$ contains $(x-1)$ four times, which, by the induction hypothesis, is not possible. Contradiction.
So c' does not contain more than three equally-valued non-dot symbols. □

Theorem 11. *Let $c \in \Sigma_n^{\text{pass}}$. For every symbol in x in c we have $T(c, x) \geq x - 1$.*

Proof. For obtaining x at some point the substring $1123 \ldots (x-2)(x-1)$ should occur on the board (note that it cannot occur in reverse order by Lemma 8). Merging this substring to x costs exactly $x-1$ moves, and since after every move a new 1 appears on the board, there is an additional value in c of at least $x - 1$ made up by tiles that are all lower than x, so $T(c, x) \geq x - 1$. □

Lemma 12. *Let $c \in \Sigma_n^{\text{pass}}$. The largest value that can occur in c is n. This value can be achieved by continuously sliding rightwards and is located at the end of the configuration.*

Proof. We will prove this theorem with induction to n. First, note that the statement holds for $n = 1$. Now, let $N \in \mathbb{N}_{>0}$ and assume that the statement holds for all $n < N$. Thus, by continuously sliding to the right we obtain tile value $N - 1$ at the end of a board c' of length $N - 1$. Now add an empty symbol at the beginning of c' to obtain $\overline{c}$. The configuration c' is bitonic, so the leftover values are increasing. By repeatedly applying the induction hypothesis we see that on the remaining spots (without $N - 1$) a new symbol of value $N - 1$ can be created and is located at the second to last spot, thus $\overline{c}_{N-1} = N - 1$. Merging this new symbol with $\overline{c}_N = N - 1$ at the rightmost spot gives the symbol with value N.
Assume that there is a configuration c with tile $N+1$. Then there is a configuration $\hat{c}$ from which c can be reached that contains two N-valued tiles. But then either of these two tiles has been constructed on $\hat{c}$, which is a board with less than N tiles. Contradiction. □

Theorem 13. *Let $c \in \Sigma_n^{\text{pass}}$. Then $T(c) \leq 2^n - 1$, and there exists a reachable configuration c' with $T(c') = 2^n - 1$.*

Proof. We know that a configuration $c' \in \Sigma_n$ with $T(c') = 2^n - 1$ must consist of values 1 to $n - 1$ that are used only once (since the binary expansion of $T(c')$ is unique). Note that there is a reachable configuration with these values, namely $12 \ldots n$. By using Lemma 12, this configuration can be obtained by continuously sliding to the right.
Suppose that there is a configuration c' with $T(c) \geq 2^n$. By Lemma 12 no tile with value higher than n can occur, so at least one of the tiles $1, \ldots, n$ must

occur more than once. Let x be the highest tile that occurs more than once. This implies that either of the x-valued tiles has been formed on a board of length $x-1$. By Lemma 12, this is impossible. Contradiction. □

Theorem 14. *Let $c \in \Sigma_n^{\text{pass}}$. Assume $c_i = x$ and $c_{i+1} = x$ with $1 \leq i < n$. Then the following statements hold:*

- *When $i < \min \arg\max_j \{c_j\}$, then $T_\ell(c, x) \leq x - 1$.*
- *When $i < \max \arg\max_j \{c_j\}$, then $T_r(c, x) \leq x - 2$.*
- *When $i \in \arg\max_j \{c_j\}$, then either $T_\ell(c, x) \leq x - 1$ or $T_r(c, x) \leq x - 2$.*

Proof. Assume $i < \min \arg\max_j \{c_j\}$. Then in a previous configuration the substring xx starting at position i should have been $(x-1)(x-1)x$. Continuing this reasoning, in a configuration c' which leads to c in $x-1$ steps the string xx was $1123\ldots(x-1)x$ (or the reverse).
Now assume without loss of generality that $T_\ell(c, x) = x$. For every move backwards at most one 1 can be removed from the tiles left of x. So not all tiles right of x can be removed within $x-1$ steps backward. This means that going backwards $x-1$ steps, we have substring $y1123\ldots(x-1)x$ in the configuration (or substring $x(x-1)\ldots3211y$) with $y \geq 1$. If $y > 1$ then the string is not bitonic and when $y = 1$ we have three consecutive 1 symbols. Theorem 10 shows that both cases are not possible. Contradiction. So $T_\ell(c, x) \leq x - 1$.
The argument for $i > \min \arg\max_j \{c_j\}$ is similar, except that during the first step backward it is not possible to remove a 1 (since the 1 that should be removed is the last symbol of c, not the first). Therefore $T_r(c, x) \leq x - 2$.
For $i \in \arg\max_j \{c_j\}$ we know that exactly one x-valued tile of xx must be expanded, thus in this case either $T_\ell(c, x) \leq x - 1$ or $T_r(c, x) \leq x - 2$ holds. □

Note that the statement of Theorem 14 can be refined. For example, if $c = 1224331$, then both $T_\ell(c, 2) = 1$ and $T_r(c, 3) = 1$ hold. However, for any configuration that resulted in c' by a move ℓ or r, the left 2 should be expanded and the right 3 should be expanded. Thus

$$1224331 \Leftarrow (1)1243221 \vee (1)2234211$$

The above notation indicates the configurations that result in c, where (1) indicates the 1 that will be dropped before the swipes take place. The first configuration results in c by ℓ and the second configuration by r. Note that both configurations are not in $c \in \Sigma_n^{\text{pass}}$ since the former configuration ends with substring 221 and the latter configuration ends with 11. The problem here lies in the fact that in each turn only one 1 is dropped. Thus when reasoning backwards, only one 1 is removed, while there may be multiple pairs in the configuration. A refinement of Theorem 14 would mean that we compare all values $T_\ell(c, x)$ with all values $T_r(c, y)$ where x and y are part of pairs, which may become very time consuming when c consists of many pairs.
Nevertheless, Theorem 14 is powerful enough for the purpose of the algorithm that is proposed in the next section.

5.2 Checking reachable configurations in the passing game

We will now describe an algorithm that, given an element $c \in \Sigma_n^*$, determines whether $c \in \Sigma_n^{\text{pass}}$.
In the first stages of the algorithm, configuration c will be subjected to a number of tests, which all run in polynomial time in terms of n. The aforementioned theorems in the previous section are used to check if an element $c \in \Sigma_n^*$ does not belong to Σ_n^{pass}. These tests check the following rules:

1. All zeroes in c are located at the left of c and the first nonzero symbol is a 1 (Lemma 7).
2. There is no 11 substring at the end of c (Lemma 8).
3. There are no 3 consecutive symbols in c representing the same value, except for value 0 (Theorem 10).
4. String c is bitonic (Theorem 10).
5. A tile $x > 0$ can occur at most 3 times in c (Theorem 10).
6. For every symbol x in c we have $T(c, x) \geq x - 1$ (Theorem 11). We say that c meets the twopower-rule for every tile. For example, for $n = 5$, the configuration $c = ..124$ is valid in terms of this rule. For $x = 2$ we have the sum $T(c, x) = 2^{1-1} = 1 \geq x - 1$, and for $x = 4$ we have the sum $T(c, x) = 2^{2-1} + 2^{1-1} = 3 \geq x - 1$. Note that the configuration $c' = ..125$ is invalid.
7. The total value of the powers of 2 of tiles on the board c is at most $2^n - 1$ (Theorem 13). For example, for $n = 5$ the value cannot be higher than $2^0 + 2^1 + 2^2 + 2^3 + 2^4 = 31 = 32 - 1 = 2^5 - 1$. The board $c' = 12345$ corresponds to this value.
8. For each pair xx that occurs in c the following should hold (Theorem 14):
   - If the pair is left of the highest tile, then the sum of all powers of 2 of the tiles left of xx should be at most $x - 1$.
   - If the pair is right of the highest tile, then the sum of all powers of 2 of the tiles right of xx should be at most $x - 2$.
   - If x is the highest tile on the board, then at least one of the two previous cases should hold.

When a configuration passes these tests a new phase of the algorithm starts. In this process we reason backwards, in the sense that we reconstruct a sequence of configurations that would lead from c^0 to the current configuration using valid moves. This can be done in a deterministic manner, making this process $O(2^n)$, although this vanishes in the average case as $n \to \infty$. The procedure to do this will be explained below.

Given a configuration c, we want to construct the previous configuration. This means that the first 1 in c is removed. Also, certain tiles may be expanded (for example, tile $x+1$ then becomes two tiles xx). However, we need to make sure that there is enough space to do this, since the length of the board is restricted to n. This means that when there are i empty positions (after removing the first 1), that at most i tiles are expanded. Moreover, when a tile $x+1$ is expanded to xx, then we also have expansions in the next $x-1$ rounds, so the length of the non-dot substring may blow up. Therefore, the idea is to expand as few tiles as possible.
There are situations where tiles *must* be expanded nonetheless. These can be grouped in three cases:

1. When we have a pair xx, then the outer tile (the tile that is furthest away from the highest tile) should be expanded. When x is the highest value then one of both tiles should be expanded. It turns out that it is better to expand the tile that has the lowest twopower value at his side (the reason for this follows from the third case). For example, when $c = .112331$ it is better to expand the right 3-valued tile, since this one has only value 1 at his side, whereas the other 3-tile has value 4 at his side.

2. When a tile x should be expanded it may be possible that other tiles also have to be expanded. This is because when this does not happen, we eventually get three consecutive tiles, which should not be possible.

3. A tile x should also be expanded when the powers of two of tiles lower than x sum up to $x-1$, or otherwise at some point we will not have 1s that can be removed. For example, when $c = ..1125$, we have to expand the 5 since the powers of two of its lower tiles sum up to 4. If we do not expand 5 immediately, but when doing it a move later, we get the following

$$\begin{aligned} ..1125 &\implies ...125 \\ &\implies ..1144 \\ &\implies ..1334 \\ &\implies ..2234 \end{aligned}$$

   and now there is no 1 that can be removed. In fact, ...125 is an invalid move since the twopower-rule does not hold for 5.
   Note that this rule does not apply when a tile should already be expanded because of the other two rules. For example, when we have the configuration $c = 1125655$, then only the rightmost 5 should be expanded.

We know that the initial configuration is invalid when, at some point, the number of tiles that should be expanded is larger than the number of empty positions. This is because it is optimal to expand as few tiles as possible.

When the necessary tiles have been expanded we only need to know whether the non-dot substring should be mirrored (corresponding to a move to the left). This is the case when at least one of the following conditions is true:

1. The first non-dot symbol is not a 1.

2. The configuration ends with 11.

3. The configuration ends with 21 and the 2 should be expanded in the next iteration of the algorithm. To determine this we should apply the process described above on the expanded board. So basically, we look forward one iteration.

A problem occurs when one of the above conditions is true but it is not possible to mirror the board. This is the case when in addition

1. The configuration begins with 11.

2. The configuration begins with 12 and the 2 should be expanded in the next iteration.

If either of these cases is true, then the configuration we started with is invalid.

This process of expanding and mirroring is repeated until either a configuration is rejected or until c^0 is reached. In the latter case the board will be accepted. The above is a general description of the algorithm; there are a few technical subtleties that we leave out here. The algorithm has been tested on boards of length up to $n = 14$, by generating all boards in Σ_n^* and applying the algorithm on each of them. In all these cases, it exactly accepts all elements that are in Σ_n^{pass} and rejects all configurations that are not in Σ_n^{pass}. This has been checked by comparing the results to the results generated by a brute force algorithm that plays the game in all possible ways.

5.3 Checking reachable configurations in the non-passing game

We now turn our attention to the version of the game where passing is not allowed. As mentioned before, we have $\Sigma_n^{\text{nopass}} \subseteq \Sigma_n^{\text{pass}}$. The elements in $\Sigma_n^{\text{pass}} \backslash \Sigma_n^{\text{nopass}}$ can be divided into two different categories:

1. Configurations for which the sum of powers of two is too high.

2. Configurations that do not belong to the first category, and are only reachable through an invalid move to the right. For example, the move of the form $123\ldots(x-1)x \implies 1123\ldots(x-1)x$ with $x > 1$ is not valid.

We will first try to identify the configurations that are in category 1. To this end, we analyze the libraries of configurations for boards up to value $n = 19$ to find configurations for which the values of powers of two is highest. For these configurations we look at the distributions of tiles. Table 1 indicates for each $0 < n \leq 19$ per tile value the number of occurrences on those boards.

Tile n	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	g
1	1															
2	1	1														
3	2	1														
4	1	2	1													
5	1	1	2	1												
6	1	1	2	1	1											
7	1	1	2	1	1	1										
8	1	1	2	1	2		1									
9	1	1	2	1	1	2	1									
10	1	1	2	1	1	2	1	1								
11	1	1	2	1	1	2	1	1	1							
12	1	1	2	1	1	2	1	2		1						
13	1	1	2	1	1	2	1	1	2	1						
14	1	1	2	1	1	2	1	1	2	1	1					
15	1	1	2	1	1	2	1	1	2	1	1	1				
16	1	1	2	1	1	2	1	1	2	1	2		1			
17	1	1	2	1	1	2	1	1	2	1	1	2	1			
18	1	1	2	1	1	2	1	1	2	1	1	2	1	1		
19	1	1	2	1	1	2	1	1	2	1	1	2	1	1	1	

Table 1: Frequency per tile per n for configurations that have the highest twopower value. Empty entries represent zeroes.

We see that there is a pattern in Table 1 from $n = 5$ onwards. For each value n we have that from a certain $(m + 1)$-valued tile on the frequency is always zero. For a given n this value m can be determined by

$$m_n = \left\lceil \frac{3}{4}\left(n + \frac{1}{6}\right)\right\rceil$$

We also see that each tile whose value $v \leq m$ is divisible by 3 has frequency 2, except for a few cases. When $n = 4k$ for some $k > 1$ we have that when $v = m$ the frequency is 1 and when $v = m - 1$ it is 0. Also, when $n = 4k - 1$ for some $k > 1$ we have that when $v = m$ that the frequency is 1. An algorithm that checks that the sum of powers of two on the board does not exceed the sum of powers of tiles times their frequencies in the corresponding row in Table 1, without actually having to compute these values, is as follows:

```
int count;
for(int i = m; i > 0; i--){
    count = 0;
    for(int j = 0; j < n; j++)
        if(array[j] == i)
            count++;
    if(count == 0) return true;
    if (n % 4 == 0 && i == m-1)
        if(count > 0) return false;
    if (n % 4 == 0 && i == m-2){
        if(count > 2) return false;
        if(count < 2) return true;
    }
    if (i != m && i%3 == 0){
        if(count < 2) return true;
        if(count > 2) return false;
    }
    else if (count > 1) return false;
}
return true;
```

This algorithm successfully rejects configurations that belong to category 1, so we now only have to concentrate on the other category. For the game without passing it is not allowed to move to the right without merging. This means that, when reasoning backwards, in each iteration at least one tile should be expanded or the configuration should be mirrored. As of yet, it is unclear what strategy is optimal in the sense that we only additionally reject all configurations in category 2, but not configurations in Σ_n^{pass}.

To identify the configurations in category 2 we will look at the game trees in the case where passing is allowed. The partial game tree for $n = 5$ is shown in Figure 4. The subtree of 11234 has not been entirely worked out, since we do not need it for our next analysis as all subsequent configurations belong the category 1.

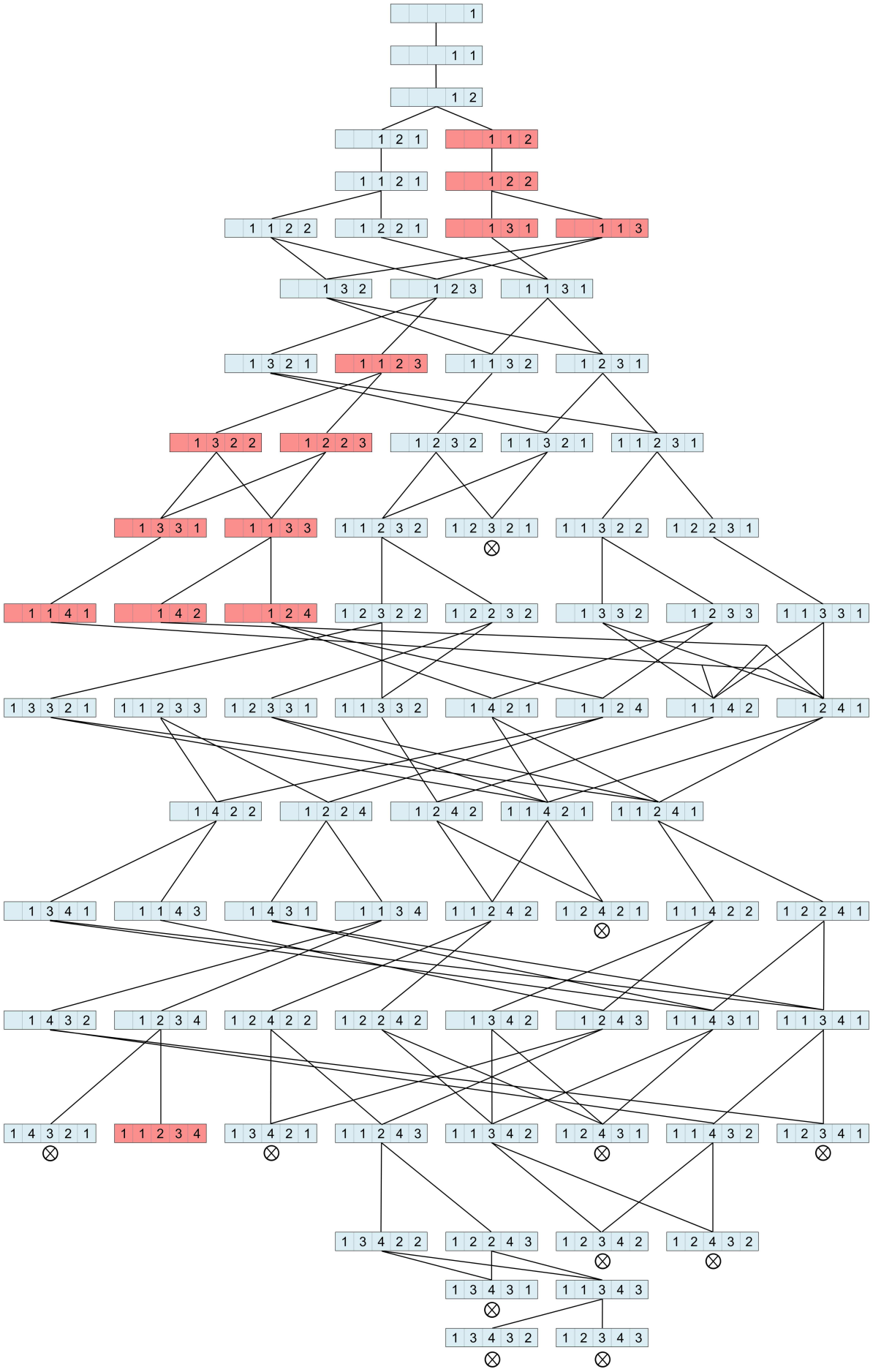

Figure 4: Partial game tree with passing for $n = 5$. Red configurations are not reachable in the non-passing game. Configuration 11234 is not worked out.

We see interesting areas in the game tree. The first one is the red area at configuration ..112, the second is the area at configuration .1123 and the third is the area at configuration 11234. The configurations in these areas all share the following characteristic: the only way to obtain these configurations is to pass through a configuration of the form $1123\ldots(x-1)x$ with $x > 1$. Hence, these configurations belong to category 2.
The question now is how to modify the algorithm described above to reject these configurations. A first thought would be to reject configurations that eventually end up in a configuration of the form $1123\ldots(x-1)x$. However, then we need to redirect the backtracking for configurations that are possible to obtain. For example, when we are in configuration ..123 we must go to .1122 instead of ..113. Note that when we go downwards from a configuration $1123\ldots(x-1)x$, the level that we reach configurations that are in Σ_n^{nopass} is the same regardless of the path we take (this has been fully verified for $2 \leq x \leq 9$). Therefore, when reasoning backwards we may determine the levels for which one may end up in the red areas and use this knowledge to find ways to avoid them. This appears to be extremely hard, since the depth of the subtrees indicated by the dashes displays strange behavior in terms of x. Table 2 indicates the depths in terms of x.

x	depth
2	2
3	3
4	4
5	6
6	8
7	18
8	19
9	31

Table 2: Depth of the red subtrees with roots of the form $1123\ldots(x-1)x$ for different values of x

Some of the values contained in these subtrees can be characterized in the following way. If the highest tile in a configuration c is $x > 0$, then the twopower-value of lower-valued tiles should sum up to x instead of $x-1$, thus $T(c, x) = x$. In addition, if c contains a substring of the form $yy(y+1)\ldots xx$ or its mirrored version with $0 < y < x$, then $T(c, y) = y$. For each x, this effectively rejects all configurations up to depth x in the corresponding subtrees. However, this does not cover the configurations lower in these subtrees. For example, for $x = 5$ the configurations in the last row of its subtree (including configurations 11622 and 1163) are still not recognized as unreachable. The reason for this is that these configuration are not only children of the subtree of 112345, but also from the subtree of 11245, which is also a configuration that only be reach using an invalid move. Characterizing all these configuration is very difficult and their relation

to the elements in $\Sigma_n^{\text{pass}} \setminus \Sigma_n^{\text{nopass}}$ are hard to analyze, because the game trees get very rapidly more complex as n increases and some problems only become apparent on levels deep in the trees.

6 Role of maximum dropped tile

In this section we will have a closer look at the role of $D_{\max}$ in the game. More specifically, we investigate the highest tile that Slider can obtain on various rectangular boards for fixed values of $D_{\max}$. We should immediately note that the higher the value of $D_{\max}$ is, the more computationally demanding it is to solve this question, since it increases the branching factor substantially. Therefore we first try to determine for different rectangular boards the value of $D_{\max}$ for which the Dropper can play in such a way that the Slider can never merge two tiles. We do this because if we find such a value i for a board it immediately implies that for $D_{\max} = i$ Slider can not obtain a value higher than i. Note that the reverse is not true.

6.1 Dropping different tiles to prevent merging

In this subsection we do not allow for merging tiles. If it happens, Dropper loses. We will first start with a simple example on a board for $m = 2$ and $n = 3$. Note that when we want to show that no merging can occur for a certain value of $D_{\max}$ the representative of same-valued tiles does not matter. We now set $D_{\max} = 3$. Also, without loss of generality (because of the previous and the fact that the board can be rotated and mirrored) we assume that Dropper can always obtain a board of the form

		1
	3	2

We now show that Slider cannot win by showing how Dropper has to play. In the next move Dropper should drop a 2 in the position aligned with 1 and 3. The only possible move for Slider is to move to the left

2	1	
3	2	

Then Dropper should drop a 3 at the lower position aligned to 2 (note that no other moves are possible). Then, whenever Slider moves upwards or to the right, Dropper can finish the game by dropping a 1-tile on the last open position.

So we have now shown that for $D_{\max} = 3$ the Slider can not merge any tile when Dropper plays optimally. Trivially, this also holds for $D_{\max} > 3$. For $D_{\max} = 2$ it is not possible since after two drops, Slider can always line up new tiles with either of these tiles and merge them in his next move.

The main question is now as follows: given a fixed size $m \in \mathbb{N}$, is there a value of $D_{\max}$ that is independent of $n \in \mathbb{N}$ such that no two tiles can be merged in case of optimal play by Dropper and Slider on a board of sizes m by n?

Showing that a value of $D_{\max}$ suffices, boils down to finding a strategy for Dropper to play optimally (in the sense that Slider can never merge two tiles). We can show that for $m = 1$ the value we look for is $D_{\max} = 2$. This is done in the following simple theorem.

Theorem 15. *Given a board with size $m = 1$, $n \in \mathbb{N}$ and $D_{max} \geq 2$. In case of optimal play by both Dropper and Slider, no tiles can be merged and the highest tile that Slider obtains is at most* 2*.*

Proof. After the first moves by both players we have a 1-tile at the right side of the board. The strategy of Dropper is simple: there is always exactly one empty position on the board adjacent to an existing tile. If this tile is 1, a 2-tile is dropped next to it and vice versa. This produces a alternating pattern of 1s and 2s on the board, none of which can be merged. This shows that the highest tile that Slider can obtain is 2. □

For $n = 2$ it is also possible to find a suitable $D_{\max}$ value. The arguments for proving that this value suffices are slightly more complicated.

Theorem 16. *Given a board with size $n = 2$, $m \in \mathbb{N}$ and $D_{max} = 4$. In case of optimal play by both Dropper and Slider, no tiles can be merged and the highest tile that Slider obtains is at most* 4*.*

Proof. First of all, note that it is possible for Dropper to obtain the following (initial) configuration after four moves (the tiles may be at the other side of the board, but then we obtain a similar configuration by mirroring the board horizontally)

x	z
y	v

where $x, y, z, v \in \{1, 2, 3, 4\}$ are pairwise different numbers (this is the case for the rest of the proof). For this configuration the distance between equally-valued positive numbers is not smaller than 2 (since there are no two equally-valued positive numbers in this configuration).
Now assume that we have a configuration C for which the following (*) holds: the horizontal distance between equally-valued numbers is not smaller than 2, possibly except for pairs of the following form

	x	z	
$\cdots$	y	x	$\cdots$

or (this is an exclusive 'or') for all pairs x embedded in the following form

	y	x	
$\cdots$	x	z	$\cdots$

With this we mean that diagonal pairs of the first form may occur (possibly multiple times) in C and diagonal pairs of the second form may occur (possibly multiple times) in C, but not both. Note that the initial configuration meets these conditions.

Now assume that we have a configuration in which all equally-valued numbers have horizontal distance larger than 1 and in which there are no diagonal pairs. Thus we have a configuration of the form

	x	z	
	y	v	$\cdots$

Dropper can now proceed by placing a new tile on one of the two positions with an underscore. Consider the lowest of the two. We should try to retain the form (*). Thus Dropper can only drop a z or a v on this position. Suppose Dropper drops a v. Then Slider can move upwards, and then Dropper places z on the lower position. This gives:

v	x	z	
z	y	v	$\cdots$

and the form of (*) is retained. However, Slider also may have swiped to the left. Mirroring the resulting board gives the following configuration:

?	s		w	z	x
r	t	$\cdots$	v	y	v

with $w = x$ or $w = y$ (since the distance between equally-valued numbers was 2 or larger), and $r, s, t \in \{1, 2, 3, 4\}$ being pairwise different numbers. When $w = y$ we see that a diagonal pair has formed. In general more pairs of this form may have been created, but they are all oriented in the same direction (to see this, suppose that a pair of the other orientation has been formed, then in the previous configuration they were aligned and could have been merged). We see that when Dropper places a tile not equal to s or r on the question mark the form of (*) is retained.

A similar argument holds when Dropper places a z on the lower underscore. Naturally, the arguments also hold when Dropper places a v or a z on the upper underscore.

Now assume that we have a configuration in which all equally-valued numbers have horizontal distance 2 or larger, except for pairs of the form

	x	z		
$\cdots$	y	x	$\cdots$	

and that there appears at least one such diagonal pair in the configuration, but no diagonal pairs with the other orientation (the proof for the reverse case is analogous). Then we have a configuration of the form

$\_$	x	z		
$\_$	y	v	$\cdots$	

with possibly $x = v$. Let $q \in \{1, 2, 3, 4\}$ be such that $q = v$ if $x \neq v$ and $q \neq x, q \neq y, q \neq z$ if $x = v$. In this case, Dropper cannot place a tile on the position marked by the upper underscore, because when Dropper swipes to the right the diagonal pairs become aligned. Thus, Dropper can only place tile q or z on the lower underscore. If Slider swipes upwards after Dropper placed either of these tiles, Dropper can place the other tile on the lower underscore. Then the form of (*) is retained.
On the other hand, Slider could also have swiped to the left. Assume Dropper places z on the lower underscore. A swipe to the left results, after mirroring, in

?	s		w	z	x	
r	t	$\cdots$	v	y	z	

with $w = x$ or $w = y$ (since the distance between equally-valued numbers was 2 or larger), and $r, s, t \in \{1, 2, 3, 4\}$ being pairwise different numbers. We see that at least one diagonal pair of the form

	x	z		
$\cdots$	y	x	$\cdots$	

has appeared. Also note that, there are no pairs with the other orientation (as otherwise it was possible to merge these tiles in the previous configuration). Again, we see that when Dropper places a tile not equal to s or r on the question mark the form of (*) is retained. A similar argument can be given when Slider

placed q on the lower underscore.

To summarize, we have given a strategy for Dropper for which the boards retain a property under the moves of Slider such that no tiles can be merged. Since all tiles in this proof are in $\{1, \ldots, 4\}$, it follows that the highest tile that Slider can obtain is at most 4.

□

For the other cases we resort to computational methods. We use a simple modification of the program in [21], declaring the game as a lose for Dropper when two tiles have merged. Note that two tiles have merged when the number of tiles on the board is less than the number of moves by Dropper.
Table 3 below gives for various boards the minimum $D_{\max}$ value that is needed to fill the entire board with tiles without merging.

m \ n	1	2	3	4	5	6	7	8	9
1	-	2	2	2	2	2	2	2	2
2	-	3	3	4	4	4	4	4	4
3	-	-	4	4	5				
4	-	-	-	6					

Table 3: Minimum amount of different tiles to play the game without ever merging two tiles in case of optimal play by both player, for various values of m and n. Empty entries are not yet computed due to time restrictions, as these computations are matters of weeks.

We see that the two upper rows corresponding to $m = 1$ and $m = 2$ are in accordance with Theorem 15 and Theorem 16. For $m = 2$ and $n < 4$ the minimum $D_{\max}$ values are even lower (as evidenced by the example at the start of this paragraph). The third row shows that for $m = 3$ the minimum $D_{\max}$ value is at least 5 and the fourth row shows that for $m = 4$ this value is at least 6. Also noteworthy is the fourth column, where the values for $m = 2$ and $m = 3$ are equal, but between $m = 3$ and $m = 4$ there is a jump of 2. This shows that the additional freedom for the players that is created by adding new rows or columns is not easy to characterize and strongly depends on the morphology of the boards.

6.2 Maximum reachable tiles for various settings

Now we take a more general approach by allowing the tiles to be merged and investigating what the highest reachable for Slider are in different setups. Again, we vary the parameters m and n and in addition also the $D_{\max}$ parameter. The values in the Table 3 in the previous paragraph are upper bounds for their

respective $D_{\max}$ cases, but they do not necessarily have to be sharp.

In Table 4 the results are outlined for the case $D_{\max} = 1$. We see that for all computed values with $m > 1$ the maximum reachable tile is equal to $m \cdot n$. Thus in this case Dropper has virtually no control over the strength of Slider's moves. Dropper only has some influence in the game when $m = 1$. This case has been discussed in Section 5.

m \ n	1	2	3	4	5	6	7	8	9
1	-	2	2	3	4	5	6	7	7
2	-	4	6	8	10	12			
3	-	-	9	12					
4	-	-	-						

Table 4: Maximum reachable tile for various boards in case of optimal play by both players for $D_{\max} = 1$.

Table 5 shows the results for $D_{\max} = 2$, with very different results compared to Table 4. As a result of Theorem 15 we see that the row for $m = 1$ consists entirely of 2s, and of course this holds for all values larger than $D_{\max} = 2$. The values in the rows and columns increase much slower with m and n. For $m = 3, n = 5$ only a lower bound has been computed, but given the structure of the table it is likely that the maximum reachable tile will indeed be 7. Of course, the value for $m = 4, n = 4$ is the result we are looking for, but the resources in terms of memory and computation time are limited as of yet (see Section 9 for a more detailed discussion). However, given the other values in the table, an educated guess for the maximum reachable tile for $m = 4, n = 4$ would be either 7 or maybe 8.

m \ n	1	2	3	4	5	6	7	8	9
1	-	-	2	2	2	2	2	2	2
2	-	3	4	5	5	6	6		
3	-	-	5	6	≥ 7				
4	-	-	-	≥ 7					

Table 5: Maximum reachable tile for various boards in case of optimal play by both players for $D_{\max} = 2$.

In contrast to the previous two tables, Table 6 corresponding to $D_{\max} = 3$ shows discrepancies between boards with an equal amount of squares but with different shapes for setups with $m > 1$. For instance, in the case $m = 2, n = 6$ the maximum reachable tile is 4, but in the case $m = 3, n = 4$, where the board

has the same amount of squares, the maximum reachable tile is 5. Similarly, for $m = 2, n = 8$ this value is 5 while for $m = 4, n = 4$ this is at least 6. This once again confirms that the shape of the board is important for the possiblities of the players, and that this difference is not only present between $m = 1$ and $m > 1$.

n / m	1	2	3	4	5	6	7	8	9
1	-	-	-	2	2	2	2	2	2
2	-	3	3	4	4	4	4	5	
3	-	-	4	5	5				
4	-	-	-	$\geq$6					

Table 6: Maximum reachable tile for various boards in case of optimal play by both players for $D_{\max} = 3$.

For $D_{\max} = 4$ the entire second row in Table 7 is not larger than 4, in accordance with Theorem 16. Comparing the other values in this table with those in Table 3, the case $m = 3, n = 5$ is worth mentioning. In this case we need at least five different tiles to entirely prevent any merging, but when tiles 1 to 4 are used Dropper still can manage that Slider loses when $O = 5$. As it appears, Dropper can let tiles merge without drastic results, as long as it does not set off a chain reaction in the sense that Slider reaches a 5.

n / m	1	2	3	4	5	6	7	8	9
1	-	-	-	2	2	2	2	2	2
2	-	3	3	4	4	4	4	4	4
3	-	-	4	4	4	$\geq$5			
4	-	-	-	5					

Table 7: Maximum reachable tile for various boards in case of optimal play by both players for $D_{\max} = 4$.

By comparing the four tables it is clear that the value of the maximum reachable tile is decreasing in $D_{\max}$. The rate of decrease is quite fast. Between $D_{\max} = 1$ and $D_{\max} = 2$ the computed values are roughly divided by two. This is different for each board. Indeed, the rate of decrease depends not only on $m \cdot n$, but also on m and n seperately. The limited values that can be determined due to the mechanics of the game makes it not possible to make a proper quantitative analysis on this, but it is still useful to keep these observations in mind when solving various games and estimating the resources and tools that and needed for this.

7 Short games

In this section we will address the problem of playing the game such that it ends as quickly as possible. In this case Dropper and Slider play cooperatively. The problem will be split up into different cases. We will consider different values of $D_{\max}$. In addition, we will look what happens if m and n are varied. Afterwards, we will relate the theoretic results to the computational results.

7.1 Theoretical analysis

As noted, Dropper wins when there are no empty tiles and there are no two adjacent tiles of the same value. This implies that a lower bound on the number of moves can be found by analyzing boards with alternating 1-valued and 2-valued tiles. See for example Figure 5. We refer to the set of these boards as the set of 1-2-patterned boards. Note that for $D_{\max} = 1$ for all m and n this set contains one board that is uniquely defined modulo symmetry. This also holds when m and n are odd, since by Theorem 1 there must be at least one corner point with value 1.
Note that for obtaining a lower total value of tiles, one of the 2-valued tiles must be a 1-valued tile or one 1-valued tile must be removed, but then by Corollary 3 the configuration is not a final board any more. The question is in which situations this pattern can be achieved.

2	1	2	1
1	2	1	2
2	1	2	1

Figure 5: 1-2-patterned board for $m = 3$ and $n = 4$.

We define $M(m, n, D_{\max})$ to be the smallest number of moves by Dropper needed to win the game, where Slider cooperates as good as possible.

Assume $D_{\max} \geq 2$. From Theorem 15 it follows that when $m = 1$ the alternating 1-2 pattern can be achieved. For each n, the pattern can be achieved by alternately dropping 1s and 2s.

For $m > 1$, the same trick can be repeated. Assume that Dropper starts placing tiles in the first row. When Slider swipes the tiles alternately to the left and to the right and Dropper places a 1 next to a 2 or vice versa in the first row, eventually the first row will be filled with a 1-2 pattern. Slider is then forced to swipe the entire row downwards. Now Slider and Dropper can repeat the same process in the first row and create another 1-2 pattern. They must take care that when Slider has to move downwards again no tiles can merge. To achieve this, Slider and Dropper can play in exactly the same way as they did for the first

row, but Slider must place a 1 where he placed a 2 in the previous row and vice versa. Sliding downwards now gives a 1-2 rectangular pattern in the lower part of the board. Repeating this process eventually gives a rectangular 1-2 pattern on the entire board. Figure 6 illustrates this process.

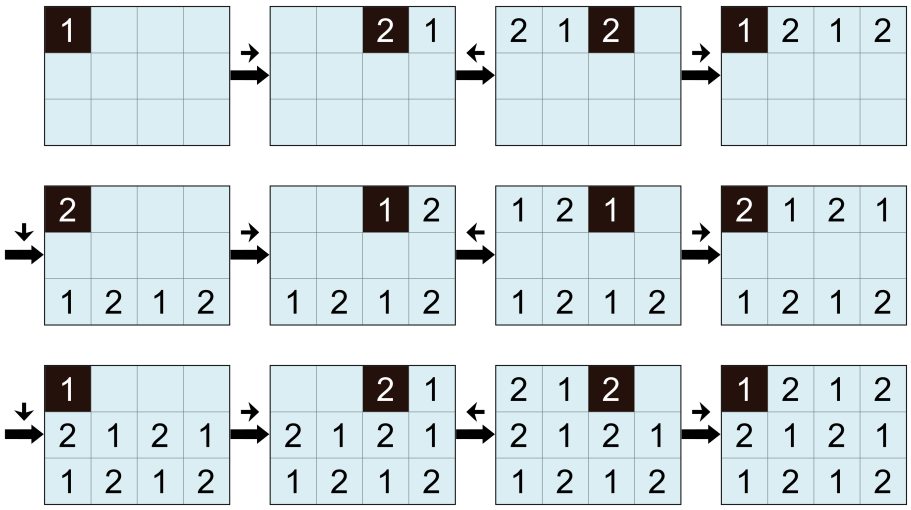

Figure 6: Illustration of the strategy for letting Dropper win where $D_{\max} = 2$ and $m = 3$ and $n = 4$. The newly dropped tiles are indicated by a black square and the directions of the moves of Slider are indicated by the small arrows.

The arguments above show that for $D_{\max} \geq 2$ a 1-2 pattern can be achieved on any rectangular board. For m and n fixed it takes exactly $m \cdot n$ moves for Dropper to completely fill the board and win the game, since every tile is dropped during exactly one move. For Slider it costs exactly $m \cdot n - 1$ moves to fill the board.

Now assume $D_{\max} = 1$. First of all, note that for $m = 2$ the 1-2 pattern can be created. For this see Figure 7. In this case it costs exactly 3 moves for Dropper per column and thus $M(2, n, 1) = 3n$ moves to fill the whole board.

Assume $m = 1$. We know that the final board should not have any empty tiles and there are not two adjacent tiles. Also, it can be shown (using a proof similar to that of Theorem 10) that the final board should also be bitonic. Therefore, for odd n the earliest obtainable final board winning for Dropper has the following form:

1	2	$\cdots$	$\frac{n+1}{2} - 1$	$\frac{n+1}{2}$	$\frac{n+1}{2} - 1$	$\cdots$	2	1

The earliest obtainable board winning for Dropper with $n \geq 4$ even has the

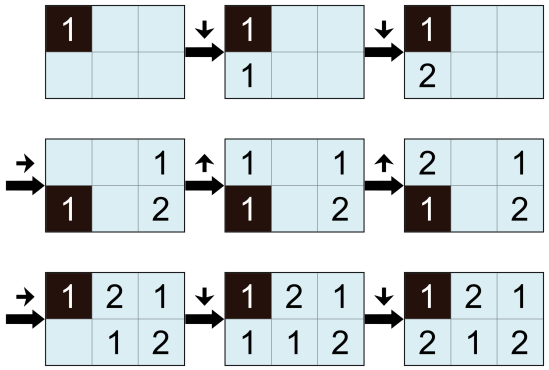

Figure 7: Illustration of the strategy for letting Dropper win where $D_{\max} = 1$ and $m = 2$ and $n = 3$.

following form:

1	2	$\cdots$	$\frac{n}{2}$	$\frac{n}{2}+1$	$\frac{n}{2}-1$	$\cdots$	2	1

The exception is for $n = 2$, where the earliest obtainable board winning for Dropper is

1	2

All cases of n can be expressed in one form:

1	2	$\cdots$	$\lceil\frac{n+1}{2}\rceil$	$\lfloor\frac{n-1}{2}\rfloor$	$\cdots$	2	1

We will now show that this board is obtainable by giving the strategy for Dropper and Slider that results in this board. First, Dropper places a tile anywhere on the board except for the rightmost site on the board. After this, Slider always swipes rightwards. During Dropper's next move, he places his tile next to another tile (this position is uniquely determined). When the leftmost tile on the board is larger than 1, Dropper places his tile at distance 2 from the leftmost tile. Dropper and Slider continue this strategy until the configuration the following configuration appears on the board:

$\cdots$	.	1	2	$\cdots$	$\lceil\frac{n+1}{2}\rceil-1$	$\lceil\frac{n+1}{2}\rceil$

This is always possible because the non-dot tiles in this configuration take up $\lceil \frac{n+1}{2} \rceil \leq n$ sites (note that non-dot tiles in any intermediate configuration do not take up more than $\lceil \frac{n+1}{2} \rceil$ sites as well). When this configuration appears, Slider swipes to the left to obtain the reverse configuration at the left side of the board, while at the right side of the board we have $n - \lceil \frac{n+1}{2} \rceil = \lfloor \frac{n-1}{2} \rfloor$ empty positions. Then (if $n > 2$) Dropper and Slider repeat the same procedure (but Slider now keeps sliding to the left) to obtain the remaining tiles of the earliest reachable configuration winning for Dropper.

Now we can determine the number of moves $M(1, n, 1)$ for Dropper needed to obtain this configuration. For this we need to take the sum of the powers of two:

$$\begin{aligned} M(1,n,1) &= \sum_{i=1}^{\lceil \frac{n+1}{2} \rceil} 2^{i-1} + \sum_{i=1}^{\lfloor \frac{n-1}{2} \rfloor} 2^{i-1} \\ &= 2^{\lceil \frac{n+1}{2} \rceil} - 1 + 2^{\lfloor \frac{n-1}{2} \rfloor} - 1 \\ &= 2^{\lceil \frac{n+1}{2} \rceil} + 2^{\lfloor \frac{n-1}{2} \rfloor} - 2 \end{aligned}$$

This gives:

$$M(1,n,1) = \begin{cases} 5 \cdot 2^{\frac{n}{2}-1} - 2 & \text{if } n \text{ even} \\ 3 \cdot 2^{\frac{n-1}{2}} - 2 & \text{if } n \text{ odd} \end{cases}$$

When comparing the cases of $m = 1$ and $m = 2$ for $D_{\max} = 1$ we see that the number of moves is linear in n for $m = 2$ and exponential in n for $m = 1$. The fact that sliding upwards or downwards is not possible for $m = 1$ creates a significant difference in the complexity of the number of moves to obtain a winning configuration for Dropper as quickly as possible.

Now assume $m > 2$. In this case it is not easy to show whether it is possible to obtain a 1-2-patterned board. Computations show that that for $m = 3, n = 3$ (achieved by a complicated sequence of moves) and $m = 3, m = 5$ it is indeed possible to obtain these boards (for some of these results see the tables in [21]). On the other hand, for $m = 3, n = 4$ and $m = 3, n = 6$ it is not possible at all. The same goes for $m = 4, n = 4$. For $m = 3, n = 4$ the earliest final board is the 1-2 pattern where one 2 is replaced by a 3. The process of reaching this board is again very complicated, and it is unclear whether it could be generalized to higher values of n.

For boards with $m > 2$ where m and n are both odd the number of moves in games that reach the earliest possible final board can be bounded from above by reducing the problem to the case of $m = 1$. For this process see Figure 8, assuming n is odd. First the lowest row is filled with 1s and 2s in such a way that there is a 1 at one edge and a 3 at the other edge. By using this 3 it is possible

to create a 1-2 patterned row above it, and afterwards a new row with a 3 can be created. This process is repeated until the last row, which will be filled in the same way as in the case of $m = 1$. This is possible because by construction the second-to-last row has 2s at both edges.

For boards with $m > 2$ where either m is odd and n even or vice versa the same strategy can be applied, but now the lowest row must be filled with 1s and 2s such that there is a 2 at both edges of the row. Then the same mechanism can be applied as in the previous case until the second-to-last row is filled, after which the last row is filled in the same way as in the case of $m = 1$.

The resulting boards of two example setups after using this approach are given in Figure 9 and Figure 10.

In case both m and n are even, a similar strategy could be employed, although there are a number of issues to cope with. First of all, there are no need for 3-valued tiles, since all rows can be filled with only 1s and 2s when m is even. For this process see Figure 11. Secondly, the case $m = 1$ can not be reused for the last row since in this way either the leftmost 1-2 or the rightmost 1-2 pair would line up with an identical pair in the row below. This can be solved nonetheless by creating the upper row in the following way (or a mirrored version):

1	2	$\cdots$	$\frac{n}{2}+1$	$\frac{n}{2}$	$\cdots$	3	2

The mirrored version is created in Figure 12, where $n = 6$. Figure 13 shows that the method is better when the board is rotated first, as this configuration can be obtained in fewer moves. Thus, when m and n are both even the method should be applied along the longest axis. The same statement for the first method (of Figure 8) holds when m and n are both odd.
Note that the method as described in Figure 11 can also be used when either m is odd and n even or vice versa. Therefore in this case two strategies can be employed. Which of the two strategies is faster depends on the values of m and n.

The number of moves by Dropper can be computer in the following way. In case m and n are both odd we have $\frac{n-1}{2}$ columns in which the number of 1s is equal to $\frac{m-1}{2}$ and the number of 2s is equal to $\frac{m+1}{2}$. Furthermore, we have $\frac{n-1}{2}$ columns containing one 3, in which the number of 2s is equal to $\frac{m-1}{2}$ and in which the number of 1s is equal to $\frac{m-1}{2}$. The number of moves to obtain the last column is equal to the case of $n = 1$ (the length of this column is equal to m here).
In case m and n are both even we have $n - 1$ columns in which the number of 1s is $\frac{m}{2}$ and the number of 2s is $\frac{m}{2}$. The number of moves to obtain the last column

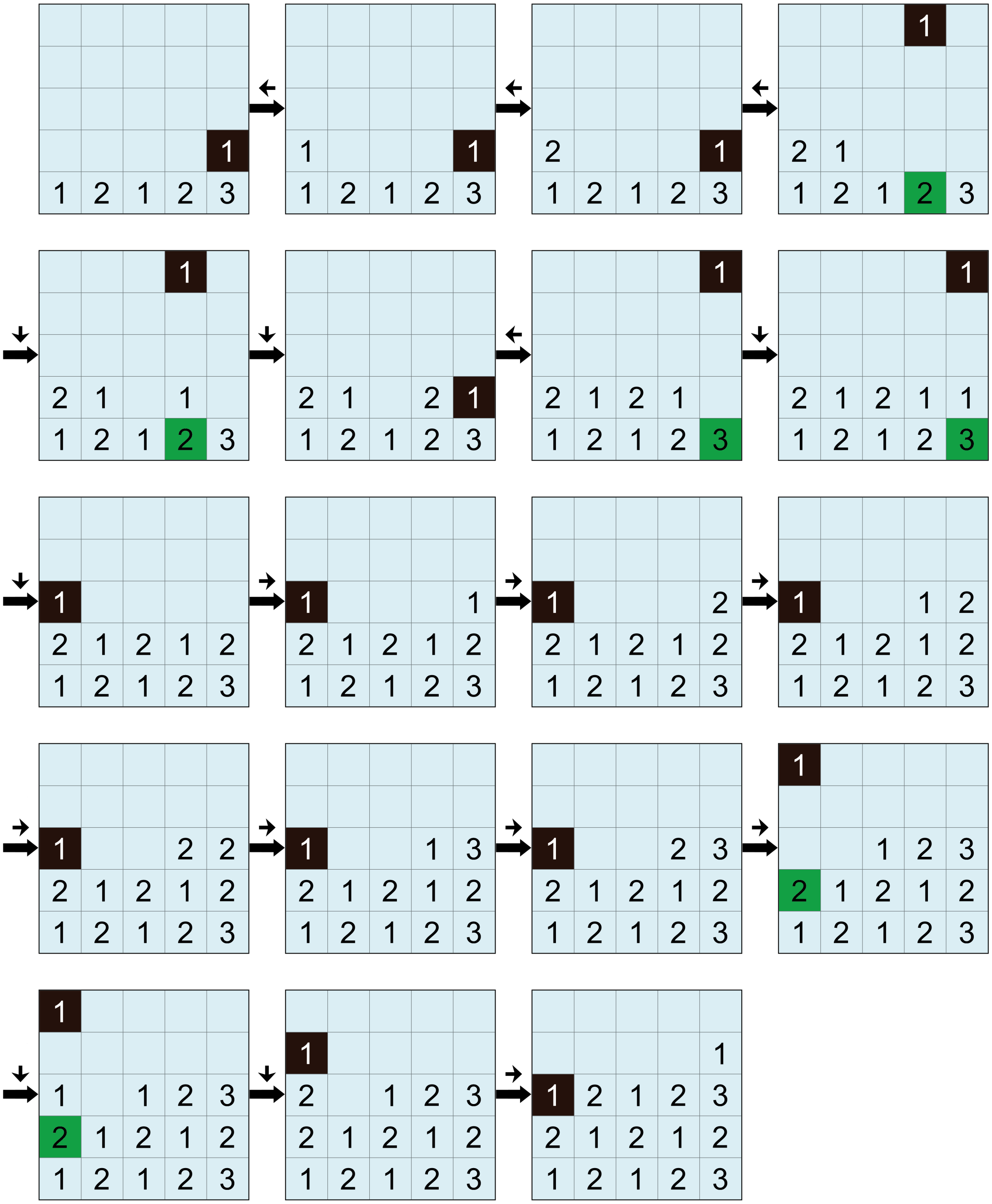

Figure 8: Illustration of the strategy for letting Dropper win as quickly as possible, where $D_{\max} = 1$ and both parameters m and n are odd. In this example we have $m = 5$ and $n = 5$. The green squares indicate the 2s and 3s that are exploited to create new 2s. By continuing this method and filling the last row as in the case of $m = 1$ we obtain the configuration in Figure 9.

1	2	3	2	1
2	1	2	1	2
1	2	1	2	3
2	1	2	1	2
1	2	1	2	3

Figure 9: Result of the given approach for $m = 5, n = 5, D_{\max} = 1$.

1	2	3	2	1
2	1	2	1	2
1	2	1	2	3
2	1	2	1	2

Figure 10: Result of the given approach for $m = 4, n = 5, D_{\max} = 1$.

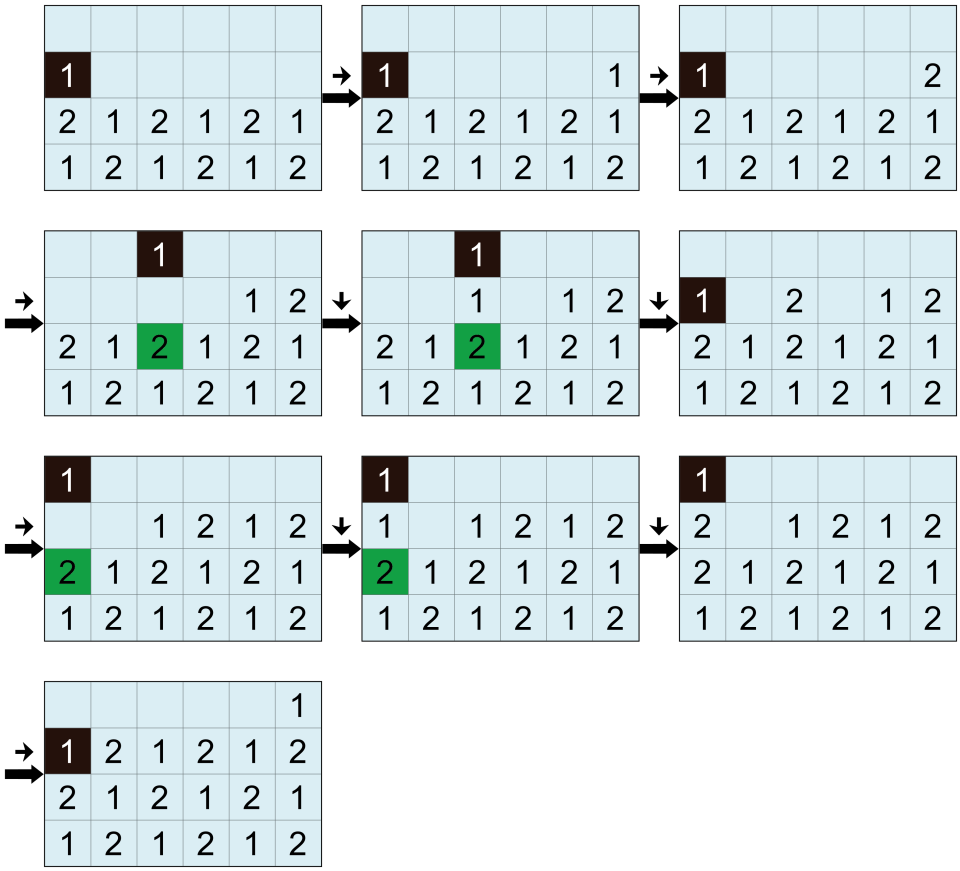

Figure 11: Illustration of the strategy for letting Dropper win as quickly as possible, where $D_{\max} = 1$ and both parameters m and n are even. In this example we have $m = 4$ and $n = 6$. The green squares indicate the 2s that are exploited to create new 2s. By continuing this method and filling the last row in a similar way as in the case of $m = 1$ we obtain the configuration in Figure 12.

2	3	4	3	2	1
1	2	1	2	1	2
2	1	2	1	2	1
1	2	1	2	1	2

Figure 12: Result of the given approach for $m = 4, n = 6, D_{\max} = 1$.

2	3	2	1
1	2	1	2
2	1	2	1
1	2	1	2
2	1	2	1
1	2	1	2

Figure 13: Result of the given approach for $m = 6, n = 4, D_{\max} = 1$.

is equal to

$$\begin{aligned}\sum_{i=1}^{\frac{m}{2}+1} 2^{i-1} + \sum_{i=2}^{\frac{m}{2}} 2^{i-1} &= 2^{\frac{m}{2}+1} - 1 + 2^{\frac{m}{2}} - 2 \\ &= 3 \cdot 2^{\frac{m}{2}} - 3\end{aligned}$$

In case the parameters are not both odd or even the upper bound is the minimum of the two strategies given above.

For the number of moves the above can be summarized into the following inequality, given $m \leq n$:

$$M(m,n) \leq \begin{cases} (3m+3)\frac{n-1}{2} + 3 \cdot 2^{\frac{m-1}{2}} - 2 & \text{if } m, n \text{ odd} \\ \frac{3}{2}m(n-1) + 3 \cdot 2^{\frac{m}{2}} - 3 & \text{if } m, n \text{ even} \\ \min(\frac{3}{2}n(m-1) + 3 \cdot 2^{\frac{n}{2}} - 3, & \\ \quad (3m+3)\frac{n-1}{2} + \frac{3m-1}{2} + 3 \cdot 2^{\frac{m-1}{2}} - 2) & \text{if } m \text{ odd, } n \text{ even} \\ \min(\frac{3}{2}m(n-1) + 3 \cdot 2^{\frac{m}{2}} - 3, & \\ \quad (3n+3)\frac{m-1}{2} + \frac{3n-1}{2} + 3 \cdot 2^{\frac{n-1}{2}} - 2) & \text{if } m \text{ even, } n \text{ odd} \end{cases}$$

Of course, the upper bound of the moves for Slider are equal to the expressions above minus one. We see that the upper bounds given above are exponential in either n or m (but not both). The parity of n and m determine in which parameter the expression is exponential.

In summary, we have that the minimal number of moves for Dropper to win the game $M(m, n, D_{\max})$ is equal to

$$M(m,n,D_{\max}) \begin{cases} = m \cdot n & \text{if } D_{\max} \geq 2 \\ = 5 \cdot 2^{\frac{n}{2}-1} - 2 & \text{if } m = 1, n \text{ even}, \\ & \quad D_{\max} = 1 \\ = 3 \cdot 2^{\frac{n-1}{2}} - 2 & \text{if } m = 1, n \text{ odd}, \\ & \quad D_{\max} = 1 \\ = 3n & \text{if } m = 2, D_{\max} = 1 \\ \leq (3m+3)\frac{n-1}{2} + 3 \cdot 2^{\frac{m-1}{2}} - 2 & \text{if } m > 2, n > 2 \text{ odd}, \\ & \quad D_{\max} = 1 \\ \leq \frac{3}{2}m(n-1) + 3 \cdot 2^{\frac{m}{2}} - 3 & \text{if } m > 2, n > 2 \text{ even}, \\ & \quad D_{\max} = 1 \\ \leq \min(\frac{3}{2}n(m-1) + 3 \cdot 2^{\frac{n}{2}} - 3, & \text{if } m > 2 \text{ odd}, \\ \quad (3m+3)\frac{n-1}{2} + \frac{3m-1}{2} + 3 \cdot 2^{\frac{m-1}{2}} - 2) & \quad n > 2 \text{ even}, \\ & \quad D_{\max} = 1 \\ \leq \min(\frac{3}{2}m(n-1) + 3 \cdot 2^{\frac{m}{2}} - 3, & \text{if } m > 2 \text{ even}, \\ \quad (3n+3)\frac{m-1}{2} + \frac{3n-1}{2} + 3 \cdot 2^{\frac{n-1}{2}} - 2) & \quad n > 2 \text{ odd}, \\ & \quad D_{\max} = 1 \end{cases}$$

7.2 Computational results

In this section we give a short overview of the number of moves and the configurations of the shortest games for various parameter values. For the equalities of $M(m,n,D_{\max})$ all computed results are in accordance with the values and configurations in the previous paragraph. Therefore, we will look at the boards for which an upper bound for the number of moves has been given. We only take boards for which the number of moves for the shortest game is actually computable. Note that the final configurations do not have to be unique. The boards and the final configurations of a corresponding shortest game are given in Figure 14.

In the case of $m = 3, n = 3$ we see that the given approach in the previous paragraph is not optimal as it takes 3 more moves. The upper bound for $M(m,n,D_{\max})$ in case of $m = 3, n = 4$ is sharp as the first argument of the minimum operation also evaluates to 21. Therefore, the suggested approach is optimal for this board. As a consequence, the approach is also very good for all boards with $m = 4$ and $n > 3$, as it always needs only one extra move.
For $m = 3, n = 5$ the given approach is not optimal as it takes 8 more moves. We that the suggested method for $m = 4, n = 4$ also takes only one more move than any optimal method (note that a 2 instead of a 1 is substituted for a 3). The results for boards with m and n odd suggest that there is a approach such that an 1-2 patterned board. However, the way how this could be achieved is still unknown.

1	2	1
2	1	2
1	2	1

1	2	3	2
2	1	2	1
1	2	1	2

1	2	1	2	1
2	1	2	1	2
1	2	1	2	1

2	1	2	1
1	3	1	2
2	1	2	1
1	2	1	2

Figure 14: Final configurations of shortest games with $D_{\max} = 1$. Upper left: $m = 3, n = 4$ and number of moves is 13. Upper right: $m = 3, n = 4$ and number of moves is 21. Lower left: $m = 3, n = 5$ and number of moves is 22. Lower right: $m = 4, n = 4$ and number of moves is 26.

8 Notes on solving the game

One of the more important questions for the modified 2048 game is whether a game is winning for Slider given values for m, n and $D_{\max}$. In the original 2048 game we have $D_{\max} = 2$, so this case is among the most interesting ones. As indicated in Table 5 the smallest boards that are still unsolved are $m = 3, n = 5$ and $m = 4, n = 4$. In this section we will discuss the techniques that are employed for solving this question for these boards. Before this is done, we will go more into detail about the first program described in Section 3.2.

8.1 Hashing

As in [21], the hash table is used to prevent the same board being evaluated more than one time. The hashing function for a board C is given by:

$$H_{m,n}(C) = \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} C_{m-1-i,n-1-j} \cdot h^{(m-i)n-j-1}$$

where h is the hashing parameter, which is usually set equal to parameter O. The function indicates that hashing is done from left to right and from the bottom to top. An example of this way of hashing is given in Figure 15. Using hash parameter $h = 7$, this configuration is hashed to a value of

$$\begin{aligned} H_{4,4}(C) &= C_{3,3} + C_{3,2} \cdot h + C_{3,1} \cdot h^2 + C_{3,0} \cdot h^3 + C_{2,3} \cdot h^4 + \dots \\ &= 3 + 6 \cdot 7 + 4 \cdot 49 + 3 \cdot 343 + 4 \cdot 2401 + \dots \\ &= 4764512287895 \end{aligned}$$

1			1
1	4		
2	5	1	4
3	4	6	3

Figure 15: Example of a configuration. The order of hashing is from left to right and bottom to top, so the order in terms of tiles is $3-6-4-3-4-1-5-2-0-0-4-1-1-0-0-1$.

The fact that h is chosen to be one larger than the highest value that can appear on the board ensures that no two boards are hashed to the same value. On the other hand, the hash table may become very large. By using Theorem 1 we know that the upper left corner is always at most $D_{\max}$ when rotating the board

appropriately. Therefore, a very good upper bound to the largest hash that is possible is given by:

$$\max_{\substack{C \in \mathbb{N}^{m \times n} \\ \text{reachable}}} H_{m,n}(C) \leq \hat{H}_{m,n,D_{\max}} \stackrel{\text{def}}{=} (D_{\max} + 1) \cdot h^{mn-1}$$

From now on we leave out the dependencies on m, n and $D_{\max}$ as these parameter values are always clear from the context. For every configuration there are three different states that can be stored in the hash table. The first state indicates that a board is not reachable, the second state indicates that the board is reachable and losing for Dropper and the third state indicates that the board is reachable and winning for Dropper. Note that we do not store information for positions reachable by Slider since this can be retrieved by evaluating the boards that result from the four possible moves. Since every configuration has three states it is possible to store five configurations in a byte without the need of overly time-consuming computations. Therefore, the number of bytes needed for the hash table given the values of m, n and $D_{\max}$ is roughly

$$\frac{1}{5}\hat{H} = \frac{1}{5}(D_{\max} + 1) \cdot h^{mn-1}$$

For an indication how important the hash table is, see Figure 16. This graph shows the time needed to evaluate the game-theoretic value of the initial configuration C^0 as a function of the number of entries in the hash table for $m = 3, n = 4, D_{\max} = 2$ and $O = 5$. In this case, an upper bound for the largest hash is 146484375, and the actual largest entry is 146454854. If a board is hashed to an entry that is larger than the number of entries it is evaluated normally, otherwise its game-theoretic value is taken over from the table if it has been evaluated before. It is clear from the figure that the number of available entries should be large enough, because at some point, when reducing the number of entries, the time needed to evaluate the game-theoretic value increases extremely fast. In this example, it happens when reducing the table with approximately 75%. This implies that the largest portion of reachable configurations is hashed to low values or that the most important positions are hashed to low values (or both of course). The behavior as seen in Figure 16 can be observed for any possible setup. Other examples for $m = 4, n = 4$ are shown in 17. Here the denseness of the hash table is even lower, usually not more than 0.5%.

Unfortunately, for larger values of m, n or O the moment at which the computation time increases drastically occurs even earlier. For example, for $m = 3, n = 4, D_{\max} = 2$ and $O = 6$ it takes 56 seconds using a hash table of size 1090000000. However, when reducing this table with 40% it takes approximately 53171 seconds. So the need for a hash table which length is almost equal to the the upper bound is even more important.

In Figure 16 we also see that the denseness of the table increases slowly with the reduction of the number of possible entries. With denseness of the table we

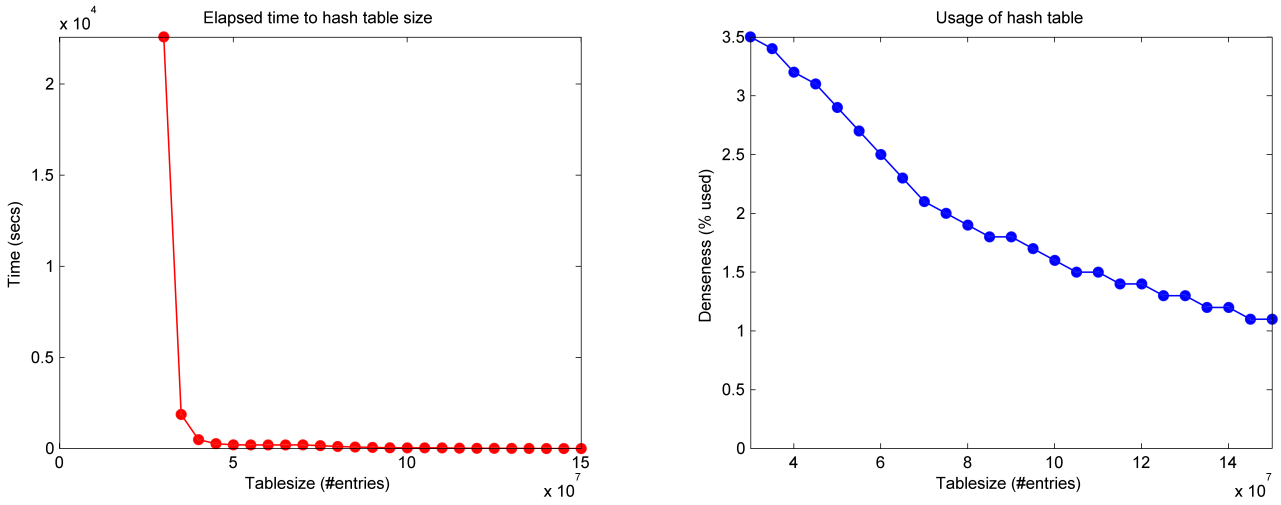

Figure 16: Left: Computation time for various lengths of the hash table. The parameter values are $m = 3, n = 4$ and $O = 5$. A boards is not hashed when its hash value exceeds the table size. Right: Denseness of the table for various lengths of the hash table. The parameter values are the same.

mean the number of unique hashed boards relative to the size of the table. The denseness increases as the number of entries reduces, although very slowly.

For 1.5TB of RAM it is in case of $m = 3$ and $n = 5$ possible to compute the game-theoretic value for all values of parameter O up to $O = 7$, since in this case the upper bound $\hat{H}$ evaluates to about $2.04 \cdot 10^{12}$, which is equivalent to a hash table of 407GB. In the case of $m = 4, n = 4$ the values can be computed up to $O = 6$, since in this case the upper bound $\hat{H}$ is about $1.42 \cdot 10^{12}$, which is equivalent to a hash table of 283GB. Nevertheless, both setups result into a win for Slider so the turning point is even higher than $O = 7$ and $O = 6$ respectively.

8.2 Hash table distributions

The graphs in the previous paragraph can be explained by inspecting the denseness of the resulting hash tables in various setups. We group the entries of the hash tables by placing them in one of 1000 bins of length approximately $\frac{1}{1000}\hat{H}$. For each bin we count how many boards are actually hashed to a value within that bin. The results give an impression of the distribution of the hashed boards within the hash table. The results for $m = 3, n = 4$ are shown in Figure 18. We see that the majority of hashed boards are in the first $\frac{1}{3}$ of the table. So we can conclude that the majority of boards have $C_{0,0} = 0$. This also explains the increase of computation time in Figure 16. Note that for increasing values of O

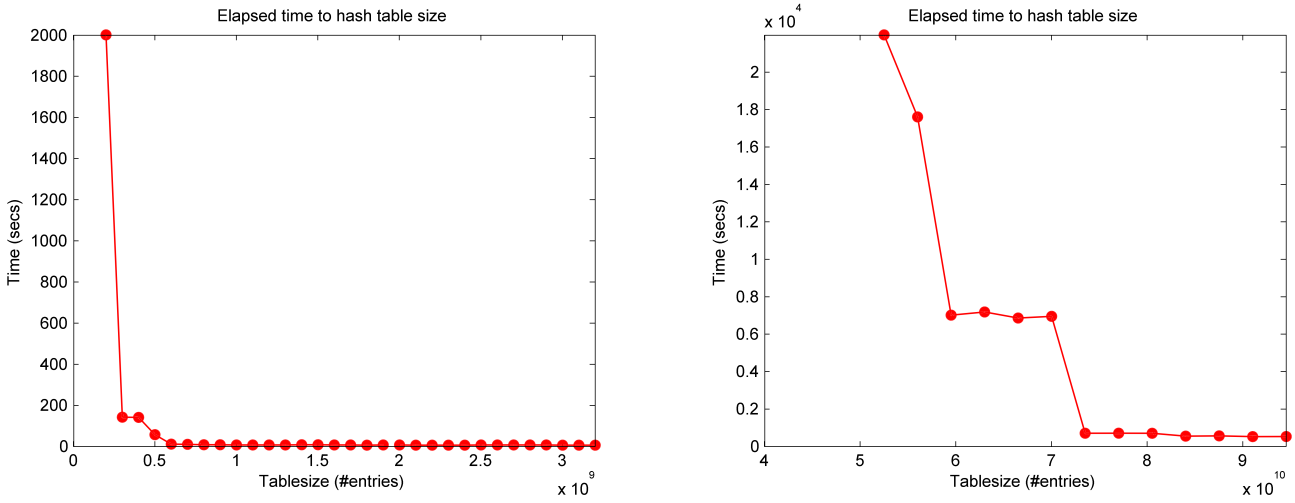

Figure 17: Left: Computation time for various lengths of the hash table. The parameter values are $m = 4, n = 4$ and $O = 4$. Right: Computation time for various lengths of the hash table. The parameter values are $m = 4, n = 4$ and $O = 5$.

we can see roughly the same regions in the graphs, although the values in the bins tend to get smoother as O increases. The results for another rectangular board with $m = 3, n = 5$ are shown in Figure 19. The behavior is approximately the same as for $m = 3, n = 4$, although here it is even easier to differentiate between well-used regions. The first $\frac{1}{3}$ of the table is used well. After this there is a region from approximately bin 450 to 650 with relatively many hashed boards. There are also two similar regions around bin 800 and at the final bins. Note that the regions tend to move a little backwards towards 0.

The results for square boards are different. In Figure 20 the results are shown for $m = 3, n = 3$ and in Figure 21 those for $m = 4, n = 4$. We see that in these cases an even larger portion of hashed boards is concentrated at lower valued bins. Figure 21 also explains the results shown in 17. For $O = 4$ there are apparently so few hashed values in later regions that the increase of computation time does not happen until 8% of the table size. For $O = 5$ the region from 600 to 800 is only moderately important as the computation time only multiplies by 10 here. It is not until about 57% that the computation time increases faster. The time will probably increase very fast for values lower than 30%.

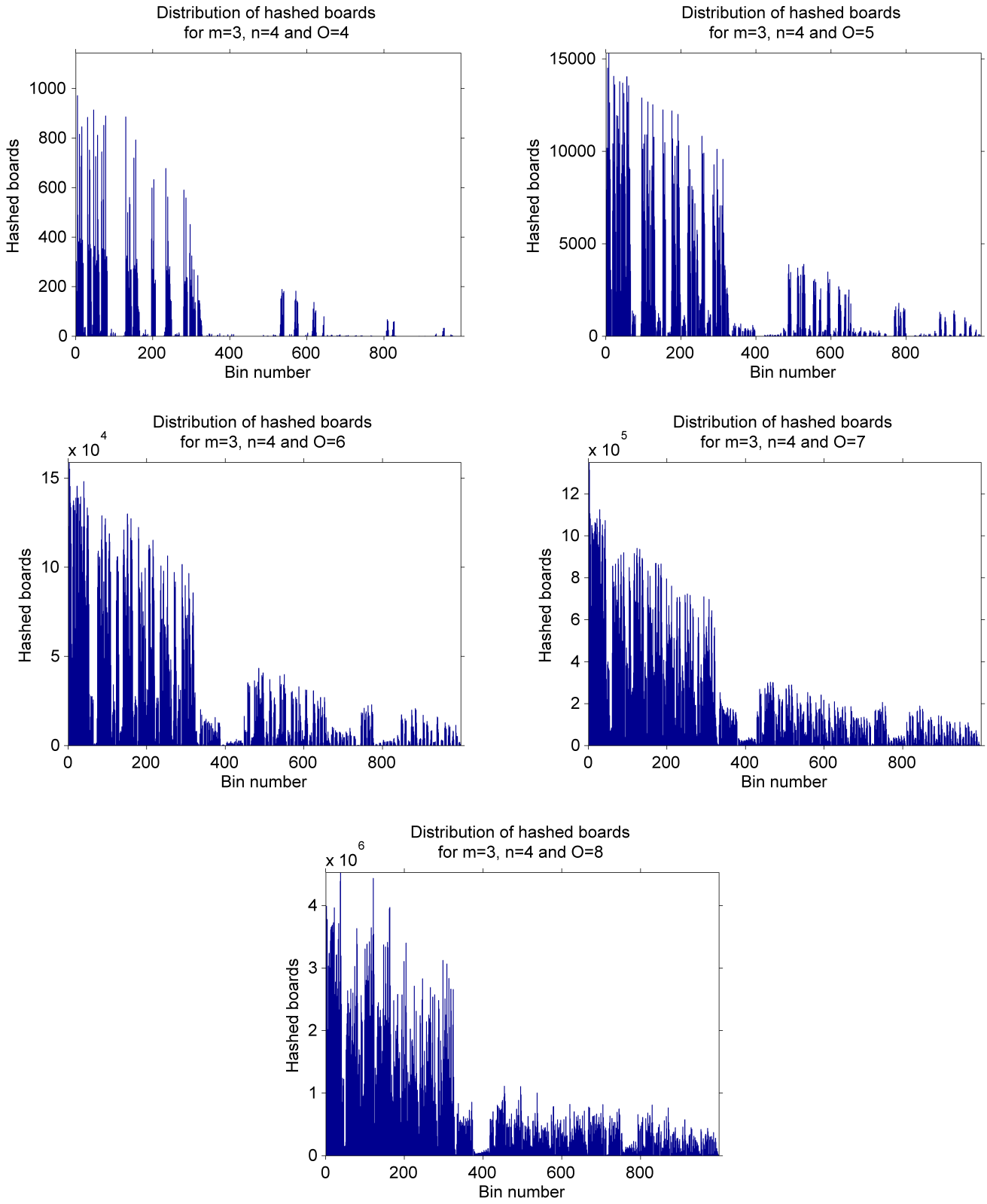

Figure 18: Distributions of used hash values. The hash values are grouped of in bins of size $\frac{1}{1000}\hat{H}$ and the values in the graph indicate the number of unique boards that are hashed into a bin. The parameter values are $m = 3, n = 4$ and O ranges from 4 to 8.

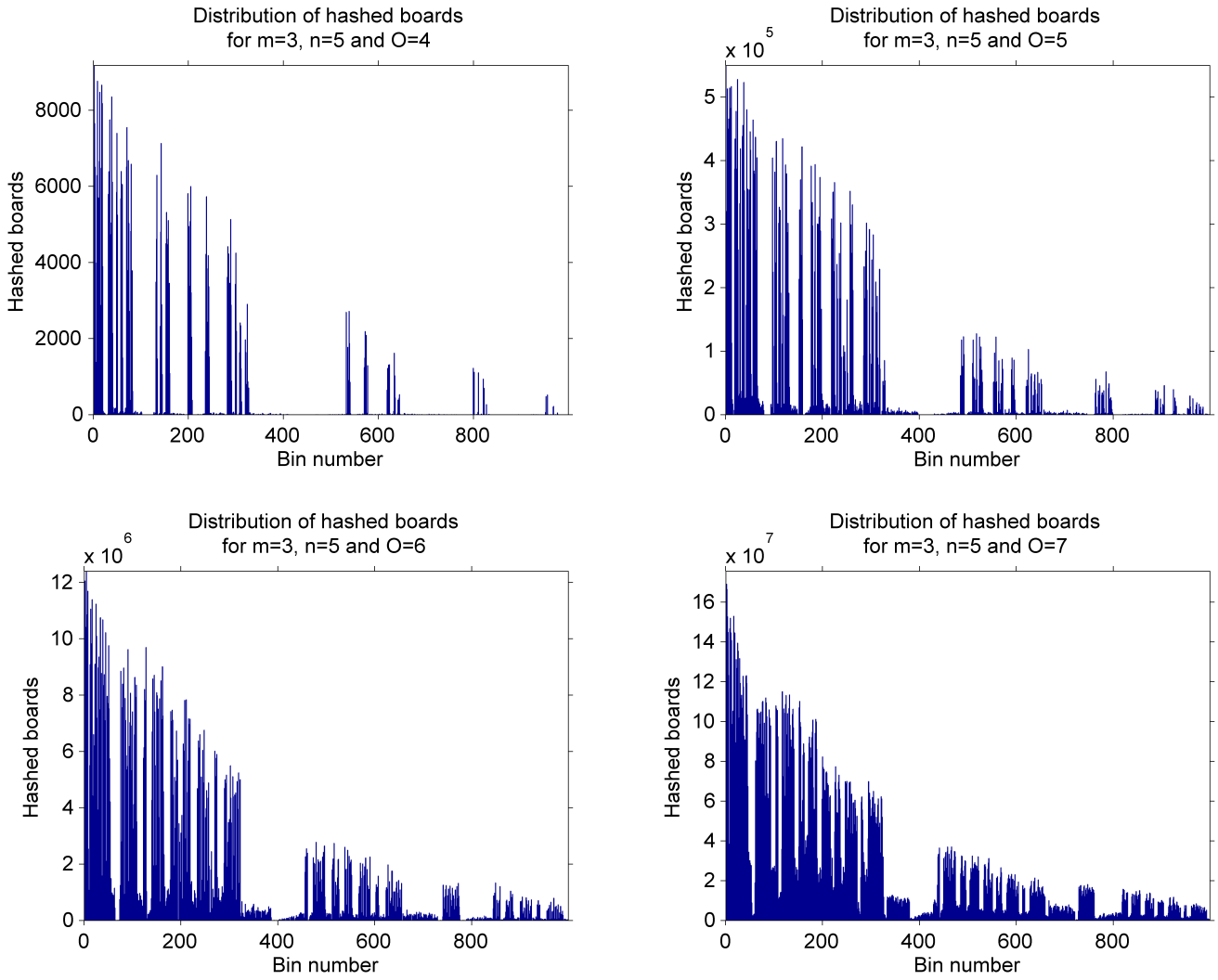

Figure 19: Distributions of used hash values. The hash values are grouped of in bins of size $\frac{1}{1000}\hat{H}$ and the values in the graph indicate the number of unique boards that are hashed into a bin. The parameter values are $m = 3, n = 5$ and O ranges from 4 to 7.

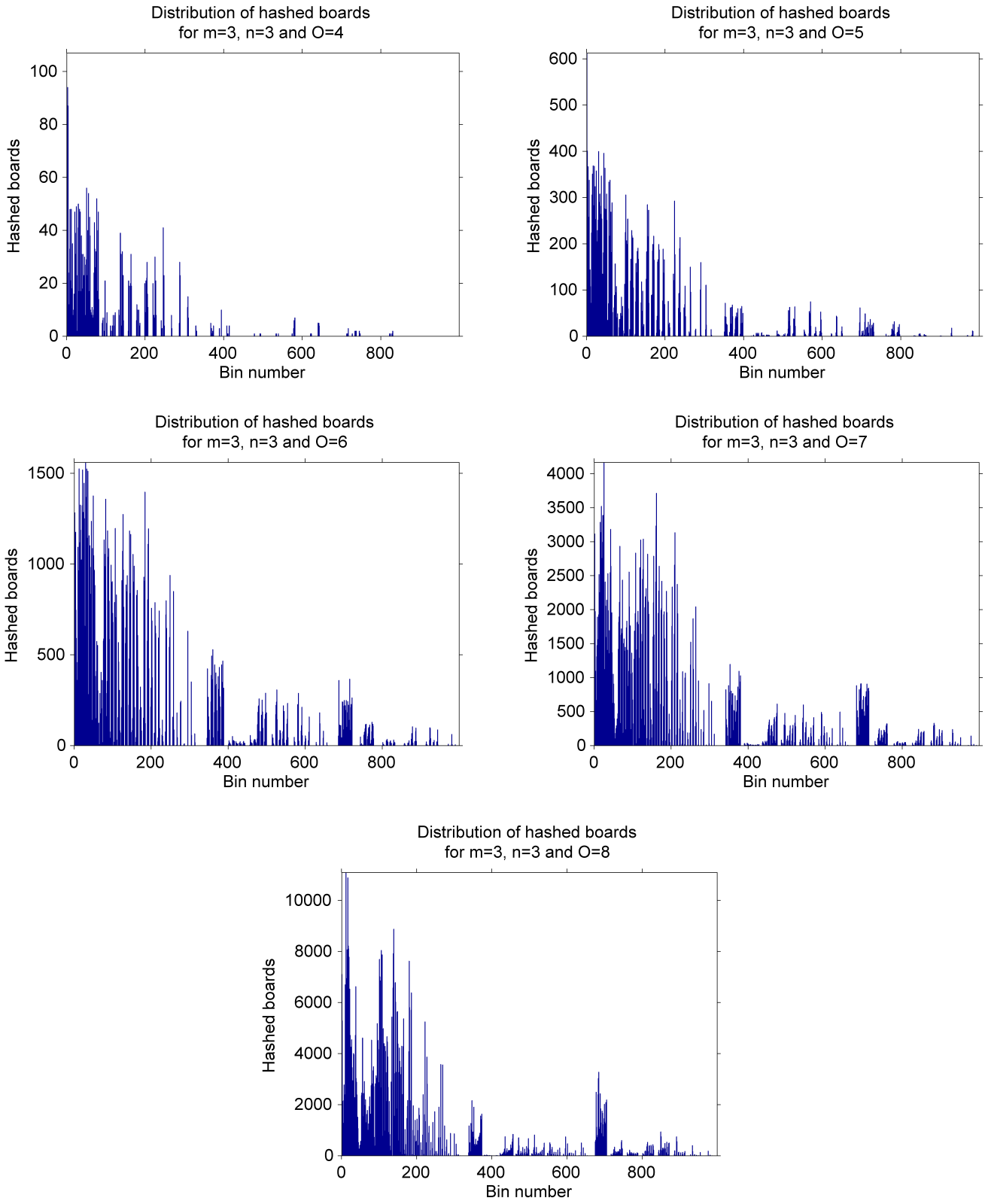

Figure 20: Distributions of used hash values. The hash values are grouped of in bins of size $\frac{1}{1000}\hat{H}$ and the values in the graph indicate the number of unique boards that are hashed into a bin. The parameter values are $m = 3, n = 3$ and O ranges from 4 to 8.

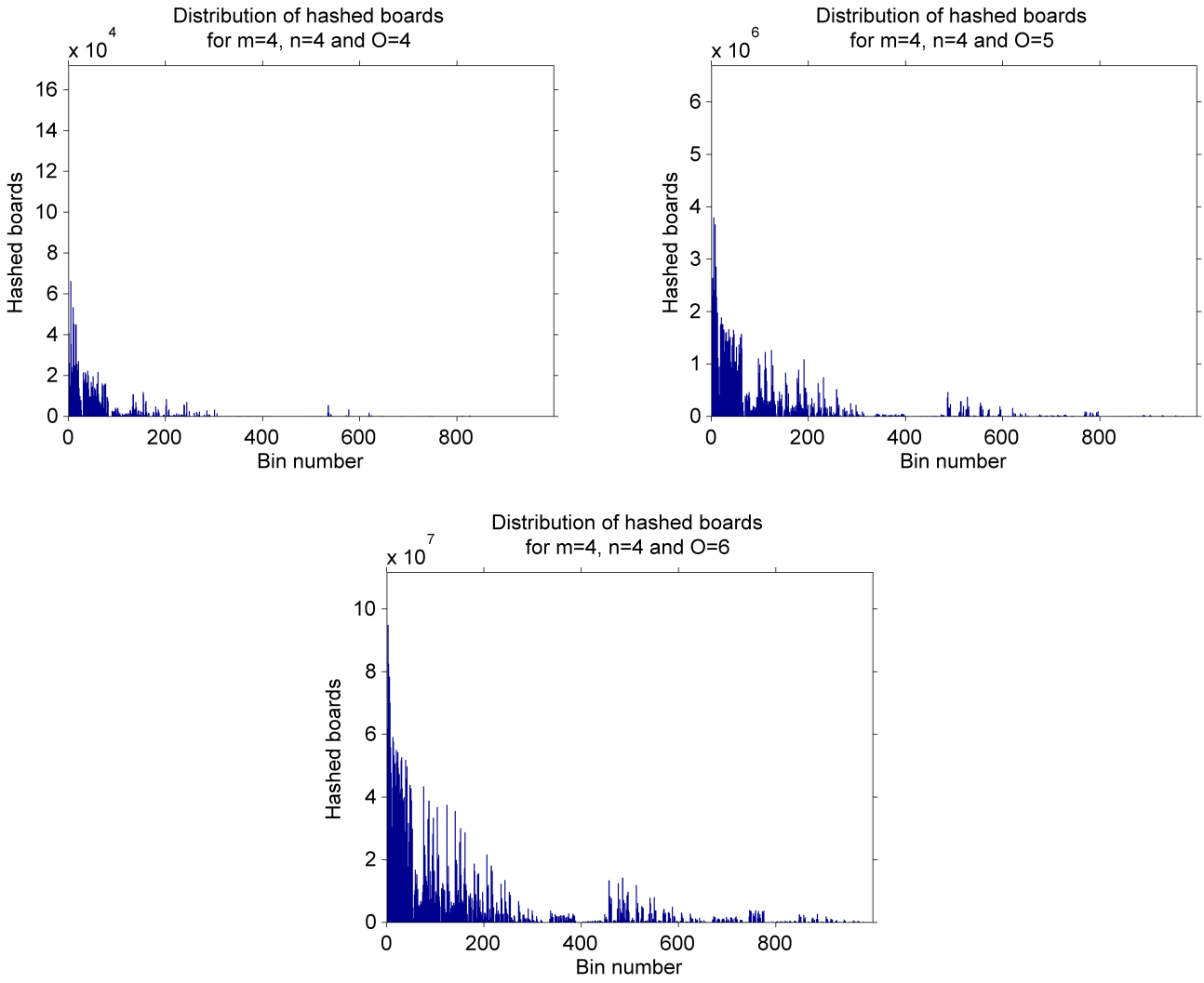

Figure 21: Distributions of used hash values. The hash values are grouped of in bins of size $\frac{1}{1000}\hat{H}$ and the values in the graph indicate the number of unique boards that are hashed into a bin. The parameter values are $m = 4, n = 4$ and O ranges from 4 to 6.

8.3 Compressing hash tables

Since the denseness of the hash table is usually not more than a few percent it is clear that by far not all configurations with a legal hash value can be reached. A subset of these unreachable configurations can be characterized by the theorems given in Section 4. These statements can be used to reduce the number of entries needed in the hash table, but this both very hard and not quite fruitful in terms of compression. However, the boards are always rotated in such a way that the hash value is minimized. This means that certain combinations of tiles in particular positions are not possible. It turns out that this is especially

advantageous for boards with $m = n$. In this particular case we see that $C_{0,1}$ cannot be larger than $C_{1,0}$ because otherwise the board would be mirrored in the diagonal axis. Similarly, $C_{0,0}$ cannot be larger than $C_{0,n-1}$ since otherwise the board would be rotated counterclockwise by 90° or mirrored in the vertical axis. As a consequence, we see that many of the bins in the tables in the previous paragraph are empty. Depending on the value of parameter O, we see that the fraction of the table that is unused because of $C_{0,1} < C_{1,0}$ is

$$\frac{\frac{1}{2}(O^2 - O)}{O^2} = \frac{O-1}{2O}$$

Similarly, depending on the value of parameter O and $D_{\max}$, we see that the fraction of the table that is unused because of $C_{0,0} < C_{0,n-1}$ is

$$\frac{\frac{1}{2}((D_{\max}+1)^2 - (D_{\max}+1))}{(D_{\max}+1)\cdot O} = \frac{D_{\max}}{2O}$$

given that $D_{\max} < O$. Note that the first expression converges to $\frac{1}{2}$ as $O \to \infty$, while the second expression converges to 0 as $O \to \infty$ with $D_{\max}$ fixed. Also, the second expression converges to $\frac{O-1}{2O}$ as $D_{\max} \uparrow O-1$, which in turn converges to $\frac{1}{2}$ as $O \to \infty$. This implies that the compression depends on the initial setup. In the case of $m = 4, n = 4$ and $O = 7$ the table could be reduced by respectively a factor $\frac{3}{7}$ and $\frac{1}{7}$, which makes it less than half as long. However, in this way we count the boards that have both $C_{0,1} > C_{1,0}$ and $C_{0,0} > C_{0,n-1}$ twice. Therefore, the table could be reduced to $\frac{4}{7} \cdot \frac{6}{7} + \frac{3}{49} = \frac{27}{49}$ of its length.
The only problem is that given a configuration C how much should be subtracted from the hash value such that no two configurations are accidentally hashed to the same values. Given a configuration C this can be done in the following way:

$$\begin{aligned}
H(C) = &\sum_{i=0}^{m-1}\sum_{j=0}^{n-1} C_{m-1-i,n-1-j}\cdot h^{(m-i)n-j-1} \\
&- C_{0,0}\cdot\frac{h^2-h}{2}\cdot h^{13} - \frac{C_{0,1}^2 - C_{0,1}}{2}\cdot h^{13} - C_{0,1}\cdot(C_{0,2}\cdot h + C_{0,3} + 1)\cdot h^{11} \\
&- \frac{C_{0,0}^2 - C_{0,0}}{2}\cdot h^{12} - C_{0,0}\cdot(C_{0,1}\cdot h + C_{0,2} + 1)\cdot h^{12} \\
&+ \frac{C_{0,0}^2 - C_{0,0}}{2}\cdot\frac{h^2-h}{2}\cdot h^{12} + \frac{C_{0,1}^2 - C_{0,1}}{2}\cdot C_{0,0}\cdot h^{12} \\
&+ C_{0,2}\cdot(C_{0,0}\cdot C_{0,1})\cdot h^{11} + \min(C_{0,0}, C_{0,3})\cdot C_{0,1}\cdot h^{11}
\end{aligned}$$

The first line in this expression is the usual hash value. The second line subtracts the number of times $C_{0,1} > C_{1,0}$ has occurred before this entry and the third line subtracts the number of times $C_{0,0} > C_{0,n-1}$ has occurred before. The fourth and the fifth line add the number of time both $C_{0,1} > C_{1,0}$ and $C_{0,0} > C_{0,n-1}$ have occurred. While the expression above can be simplified, there is a slight increase in computation time. The technique outlined above is also possible for other impossible combinations, such as for example $C_{0,0} > C_{0,n-1}$, $C_{0,0} > C_{m-1,n-1}$ and

$C_{0,1} = C_{1,0} \ \wedge \ C_{0,2} > C_{2,0}$. However, the hash function gets more complicated with each included feature, as we also need to compensate for configurations that more than one feature. Moreover, the relative compression decreases with each new feature included, resulting in less and less density gain.

Nevertheless, the above approach makes it possible to solve the case of $m = 4, n = 4, D_{\max} = 2$ and $O = 7$. The computation took about 3.5 days and the resulting distribution of hash values is given in Figure 22. The result is that the game is still winning for Slider. This means that when player plays optimally in 2048, he or she is always able to obtain tile 128.

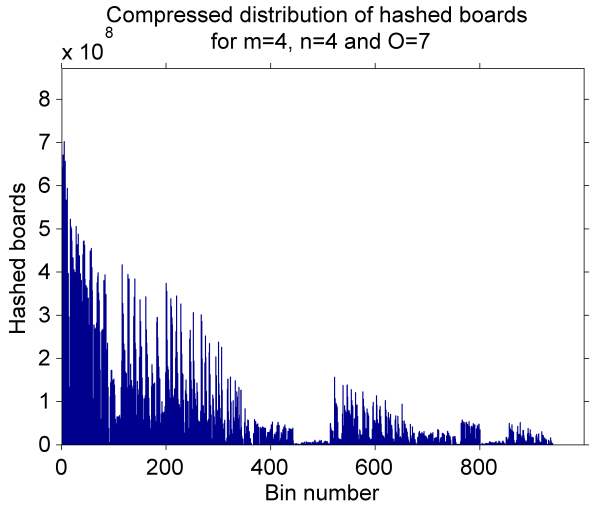

Figure 22: Distribution of used hash values for which a boards is hashed using the compression method described in this paragraph. The parameter values are $m = 4, n = 4, O = 7$.

9 Conclusions and future work

We have presented a generalization of the game 2048. For this version we have proved various theoretical aspects about the configurations of the board, the tiles that can appear on the board and conditions for applying moves. We have analyzed the one-dimensional version of the game. By stating and proving theorems we have developed an algorithm that can check whether a board is reachable in the passing game. The complexity of this algorithm is $O(2^n)$. We have given an expression for the number of moves of the shortest game in the one-dimensional version from which it follows that any brute force algorithm would have a complexity of

$$\Omega(2^{5\cdot 2^{\frac{n}{2}-1}}) \quad \text{or} \quad \Omega(2^{3\cdot 2^{\frac{n-1}{2}}})$$

depending on the parity of n. Therefore the algorithm is much better than any brute force method. The algorithm has been related to the non-passing version and the irregular behavior of the game trees in this version is discussed.
We showed that changing the maximum value $D_{\max}$ that Dropper can place on the board is influential on the highest tile that Slider can obtain. When this value is increased the highest obtainable tile is decreased. It is shown that the rate of decrease depends on m and n. We also showed that for $D_{\max} = 2$ and $m = 1$ the highest obtainable tile is 2 and that for $D_{\max} = 4$ and $m = 2$ the highest obtainable tile is 4.
In addition we analyzed the shortest possible games and gave strategies and expressions for (upper bounds of) the numbers of moves to achieve this. For $D_{\max} = 2$ and for $D_{\max} = 1$ and $m \leq 2$ we have given precise expressions for this, while for $D_{\max} = 1$ and $m > 2$ we have given upper bounds. Computational results indicated that the upper bounds are very good in some cases in the sense that the number of moves exceed the minimum by a constant value independent of the values of n. All given expressions are at most exponential in one of the parameters m and n.
Finally, we addressed the problem of solving the original game 2048 and the version of $m = 3, n = 5$ and $D_{\max} = 2$. The method of hashing has been explained and the density of the hash tables has been investigated. A method that exploits the symmetry of the board in the case $m = 4, n = 4$ has been explained and applied to the original. This showed that with optimal play it is always possible to achieve tile 128 in the original game.

It is suggested that for both the game on a board of 3×5 and 4×4 the highest obtainable tile in case of optimal play by both players is 8. However, for the 3×5 board we would need 2.64TB of RAM to show this and for the 4×4 we would need 21.12TB. The computation would take about a week in the former case and about three weeks in the latter case. By applying the methods described in Section 8.3 on two corner points for $m = 3, n = 5$ the amount of necessary memory can be reduced to $\frac{25}{32} \cdot 2.64 \approx 2.07$TB of RAM. Applying the method in exactly the same way as in that section for $m = 4, n = 4$ the amount of necessary memory

can be reduced to $\frac{71}{128} \cdot 21.12 \approx 11.72$TB of RAM. The method can be applied on more points but the reduction of memory is less with every newly included feature while it takes more computation and becomes more difficult to implement.

Another method to solve these games is to analyze the graphs given in Section 8.2. The sections in the hash table where only few tables are hashed to can be taken out of the table. This would save memory while the penalty of extra computation time would be very low. Additionally, similar graphs can be made on how often hash values in bins are requested by the program to see which sections of the hash tables are important for keeping the computation time low. This can also be combined with the height of the corresponding board in the game tree, as boards at low levels are probably more important than those on high levels. Finally, the hashing function can also be changed. The function would ideally hash boards at low levels to low values without the need of complicated computations.

More research can be devoted to the theoretical aspects of the game. The algorithm given in Section 5.2 for the passing game can be extended to the non-passing version, although this is expected to be very challenging considering the irregular structures in the game tree. Another interesting direction would be to show whether there exists a $D_{\max}$ value for which Dropper can play in such a way that Slider cannot merge two tile on a board with $m = 3$ or higher values of m. If this value exists, we have shown that it should be at least 5. Finally, better strategies can be divised for executing the shortest possible games for $m > 2, n > 2$ and $D_{\max} = 1$.

Other topics that can be investigated are the inclusion of scores, the dynamics in case of non-rectangular boards, the influence of special squares like holes in the board or the change of gameplay when the dropping or sliding mechanism is altered. There are numerous possibilities for additional research to 2048 or sliding-games in general, as evidenced by the huge amount of variations that can be found on the web.

References

[1] G. Cirulli, *2048*, http://gabrielecirulli.github.io/2048/, 2014 [accessed at 11.07.2016].

[2] 2048 Variants, Semi-long Blog, https://phenomist.wordpress.com/2048-variants/, 2014 [accessed at 11.07.2016].

[3] 20 (Slightly) Different 2048 Versions, Moment of Geekiness, http://www.momentofgeekiness.com/2014/03/22/list-20-2048-versions/ [accessed at 11.07.2016], 2014.

[4] G. Chowdhury and V. Dhamodaran, *2048 Using Expectimax*, , http://www.cs.uml.edu/ecg/uploads/AIfall14/vignesh_gayas_2048_project.pdf, 2014 [accessed at 11.07.2016].

[5] P. Rodgers and J. Levine, *An Investigation into 2048 AI strategies*, In Proceedings of 2014 IEEE Conference on Computational Intelligence and Games, 2 pages, 2014.

[6] M. Szubert and W. Jaśkowski, *Temporal difference learning of N-tuple networks for the game 2048*, In Proceedings of 2014 IEEE Conference on Computational Intelligence and Games, 8 pages, 2014.

[7] I. Wu, K. Yeh, C. Hsueh, C. Chang, C. Liang and H. Chiang, *Multi-Stage Temporal Difference Learning for 2048*, In Proceedings of Technologies and Applications of Artificial Intelligence (TAAI 2014), LNAI 8916, 366–378, Springer, 2014.

[8] H. Gui, T. Wei, C. Huang and I. Wu, *An Empirical Study on Applying Deep Reinforcement Learning to the Game 2048*, Workshop on Neural Networks in Games, The 9th International Conference on Computers and Games (CG2016), 2016.

[9] K. Oka and K. Matsuzaki, *Systematic Selection of N-tuple Networks for 2048*, In Proceedings of the 9th International Conference on Computers and Games (CG2016), 2016.

[10] M. Overlan and T. Hargreaves, 2048, https://sphere.chronosempire.org.uk/~HEx/8402/ [accessed at 17.08.2016].

[11] R. Basak, *16384 Hex*, http://rudradevbasak.github.io/16384_hex/ [accessed at 17.08.2016].

[12] F. Hamand, *2048 +-*/*, http://frankh.github.io/2048/ [accessed at 17.08.2016].

[13] R. Kandasamy, *2048 - 3D*, http://balderdash.github.io/2048/ [accessed at 17.08.2016].

[14] M. Opler, *2048 - 3D*, http://joppi.github.io/2048-3D/ [accessed at 17.08.2016].

[15] R. Mehta, *2048 is (PSPACE) Hard, but Sometimes Easy*, arXiv preprint arXiv:1408.6315, 2014.

[16] A. Abdelkader, A. Acharya and P. Dasler, *On the Complexity of Slide-and-Merge Games*, arXiv preprint arXiv:1501.03837, 2015.

[17] A. Abdelkader, A. Acharya and P. Dasler, *2048 Without New Tiles Is Still Hard*, In Proceedings of 8th International Conference on Fun with Algorithms (FUN 2016), 14 pages, 2016.

[18] C. Chen, *2048 is in NP*, http://blog.openendings.net/2014/03/2048-is-in-np.html [accessed at 11.07.2016].

[19] S. Langerman and Y. Uno, *Threes!, Fives, 1024!, and 2048 are hard*, arXiv preprint arXiv:1505.04274, 2015.

[20] M. Gobbert, *Edge Hop - Ein Modell zur Komplexitätsanalyse von kombinatorischen Spielen*, Master Thesis, Universität Trier, Germany, 2015, https://www.uni-trier.de/fileadmin/fb4/prof/INF/TIN/Veroeffentlichungen/Gob2015.pdf.

[21] M. Zeegers, *Research Project - 2048*, Leiden University, the Netherlands 2015.

[22] K. Yeh, C. Liang, K. Wu and I. Wu, *2048-Bot Tournament in Taiwan*, https://icga.leidenuniv.nl/wp-content/uploads/2015/04/2048-bot-tournament-report-1104.pdf, 2014 [accessed at 11.07.2016].

[23] W. Jaśkowski and M. Szubert, *Game 2048 AI controller competition @ GECCO 2015*, http://www.cs.put.poznan.pl/wjaskowski/pub/2015-GECCO-2048-Competition/GECCO-2015-2048-Competition-Results.pdf, 2015 [accessed at 11.07.2016].